Computer, Network, Software, and Hardware Engineering with Applications

IEEE Press
445 Hoes Lane
Piscataway, NJ 08854

IEEE Press Editorial Board
Lajos Hanzo, *Editor in Chief*

R. Abhari	M. El-Hawary	O. P. Malik
J. Anderson	B-M. Haemmerli	S. Nahavandi
G. W. Arnold	M. Lanzerotti	T. Samad
F. Canavero	D. Jacobson	G. Zobrist

Kenneth Moore, *Director of IEEE Book and Information Services (BIS)*

Technical Reviewers

Michael R. Lyu
The Chinese University of Hong Kong

Daniel Zulaica
Naval Postgraduate School

Computer, Network, Software, and Hardware Engineering with Applications

Norman F. Schneidewind

IEEE PRESS

A John Wiley & Sons, Inc., Publication

Copyright © 2012 by the Institute of Electrical and Electronics Engineers, Inc.

Published by John Wiley & Sons, Inc., Hoboken, New Jersey. All rights reserved.
Published simultaneously in Canada

No part of this publication may be reproduced, stored in a retrieval system, or transmitted in any form or by any means, electronic, mechanical, photocopying, recording, scanning, or otherwise, except as permitted under Section 107 or 108 of the 1976 United States Copyright Act, without either the prior written permission of the Publisher, or authorization through payment of the appropriate per-copy fee to the Copyright Clearance Center, Inc., 222 Rosewood Drive, Danvers, MA 01923, (978) 750-8400, fax (978) 750-4470, or on the web at www.copyright.com. Requests to the Publisher for permission should be addressed to the Permissions Department, John Wiley & Sons, Inc., 111 River Street, Hoboken, NJ 07030, (201) 748-6011, fax (201) 748-6008, or online at http://www.wiley.com/go/permissions.

Limit of Liability/Disclaimer of Warranty: While the publisher and author have used their best efforts in preparing this book, they make no representations or warranties with respect to the accuracy or completeness of the contents of this book and specifically disclaim any implied warranties of merchantability or fitness for a particular purpose. No warranty may be created or extended by sales representatives or written sales materials. The advice and strategies contained herein may not be suitable for your situation. You should consult with a professional where appropriate. Neither the publisher nor author shall be liable for any loss of profit or any other commercial damages, including but not limited to special, incidental, consequential, or other damages.

For general information on our other products and services or for technical support, please contact our Customer Care Department within the United States at (800) 762-2974, outside the United States at (317) 572-3993 or fax (317) 572-4002.

Wiley also publishes its books in a variety of electronic formats. Some content that appears in print may not be available in electronic formats. For more information about Wiley products, visit our web site at www.wiley.com.

Library of Congress Cataloging-in-Publication Data:

Schneidewind, Norman.
Computer, network, software, and hardware engineering with applications / Norman Schneidewind.
p. cm.
Includes index.
ISBN 978-1-118-03745-4 (cloth)
1. Computer engineering. 2. Computer networks. 3. Software engineering. I. Title.
TK7885.S2564 2012
005.1–dc23

2011033591

Printed in the United States of America

10 9 8 7 6 5 4 3 2 1

Contents

Preface vii

About the Author ix

Part One Computer Engineering

1. Digital Logic and Microprocessor Design 3

2. Case Study in Computer Design 63

3. Analog and Digital Computer Interactions 83

Part Two Network Engineering

4. Integrated Software and Real-Time System Design with Applications 99

5. Network Systems 125

6. Future Internet Performance Models 143

7. Network Standards 211

8. Network Reliability and Availability Metrics 228

Part Three Software Engineering

9. Programming Languages 263

10. Operating Systems 286

11. Software Reliability and Safety 303

Part Four Integration of Disciplines

12. Integration of Hardware and Software Reliability 315

Part Five Applications

13. Applying Neural Networks to Software Reliability Assessment 337

14. Web Site Design 354

15. Mobile Device Engineering 377

16. Signal-Driven Software Model for Mobile Devices 396

17. Object-Oriented Analysis and Design Applied to Mathematical Software 420

18. Tutorial on Hardware and Software Reliability, Maintainability, and Availability 443

Practice Problems with Solutions 1 466

Practice Problems with Solutions 2 504

Index 556

Preface

There are many books on computers, networks, and software engineering but none that integrate the three with *applications*. Integration is important because, increasingly, software dominates the performance, reliability, maintainability, and availability of complex computer and systems. Books on software engineering typically portray software as if it exists in a vacuum with no relationship to the wider system. This is wrong because a system is more than software. It is comprised of people, organizations, processes, hardware, and software. All of these components must be considered in an integrative fashion when designing systems. On the other hand, books on computers and networks do not demonstrate a deep understanding of the intricacies of developing software. In this book you will learn, for example, how to *quantitatively* analyze the performance, reliability, maintainability, and availability of computers, networks, and software in relation to the *total system*. Furthermore, you will learn how to evaluate and mitigate the risk of deploying integrated systems. You will learn how to apply many models dealing with the optimization of systems. Numerous quantitative examples are provided to help you understand and interpret model results.

The following topics are covered:

- application of quantitative models to solving computer, network, and software engineering problems
- mathematical and statistical models of reliability, maintainability, and availability
- statistical process and product control
- fault tree analysis
- risk management
- software metrics
- resource allocation and assignment
- software reliability models and tools
- computer security
- optimal network routing

Solutions to problems that consider only a single facet of a problem are doomed to be suboptimal. Because of its breadth, this book provides a new perspective for computer, network, and software engineers to consider the big picture in order to develop optimal solutions.

This book can be used as a text, handbook, and reference by advanced undergraduates and first-year graduate students in academia as well as by computer, network, and software engineer practitioners in the worldwide industry.

Norman F. Schneidewind
Professor Emeritus of Information Sciences
Department of Information Sciences
and the Software Engineering Group
Naval Postgraduate School

About the Author

Dr. Norman F. Schneidewind is Professor Emeritus of Information Sciences in the Department of Information Sciences and the Software Engineering Group at the Naval Postgraduate School. He is now doing research and publishing articles and books in software reliability engineering with his consulting company Computer Research. Dr. Schneidewind is a Fellow of the Institute of Electrical and Electronics Engineers (IEEE), elected in 1992 for "contributions to software measurement models in reliability and metrics, and for leadership in advancing the field of software maintenance." In 2001, he received the IEEE "Reliability Engineer of the Year" award from the IEEE Reliability Society. In 2011, he received the "Outstanding Engineer" award from the IEEE Santa Clara Valley Section. In 1993 and 1999, he received awards for Outstanding Research Achievement by the Naval Postgraduate School. Dr. Schneidewind was selected for an IEEE-USA Congressional Fellowship in 2005 and worked with the Committee on Homeland Security and Government Affairs, United States Senate, focusing on homeland security and cyber security (see photo below).

In July 2011, Dr. Schneidewind was named the Outstanding Engineer of Santa Clara Valley by the IEEE Chapter of Santa Clara Valley. In addition, he has been named Outstanding Engineer of the San Francisco Bay Area. Furthermore, he has been named Outstanding Engineer of Region 6 of the IEEE.

IEEE-USA's four Government Fellows began their Fellowships in January 2005: Randall Brouwer (with Rep. Dana Rohrabacher); Gordon Day (with Sen. Jay Rockefeller); Norman Schneidewind (on the Senate Homeland Security Committee); and Nick Zayed (with the State Department Office of Science and Technology Cooperation).

Shown at the Jefferson Memorial in Washington, D.C., are, from left to right, IEEE-USA Government Fellows Norman Schneidewind, Nick Zayed, Randall Brouwer, and Gordon Day.

In March 2006, he received the IEEE Computer Society Outstanding Contribution Award "for outstanding technical and leadership contributions as the Chair of the Working Group revising IEEE Standard 982.1," signed by Debra Cooper, President of the IEEE.

He is the developer of the Schneidewind software reliability model that is used by the National Aeronautics and Space Administration (NASA) to assist in the prediction of software reliability of the Space Shuttle by the Naval Surface Warfare Center for Tomahawk cruise missile launch and Trident software reliability prediction, and by the Marine Corps Tactical Systems Support Activity for distributed system software reliability assessment and prediction. This model is one of the models recommended by the IEEE/AIAA Recommended Practice for Software Reliability. In addition, the model is implemented in the Statistical Modeling and Estimation of Reliability Functions for Software (SMERFS) software reliability modeling tool.

Dr. Schneidewind has been interviewed by several organizations regarding his work in software reliability, including the following: a *New York Times* article, which was published on February 7, 2003, about the Space Shuttle software development process in conjunction with the Columbia tragedy and by the Associated Press about the same subject; National Public Radio, Montgomery, Alabama on April 1, 2002; and by *The Bent*, Tau Beta Pi's (all engineering society) magazine, about his professional accomplishments on November 4. 2002. This article was part of a series about prominent Tau Beta Pi members.

He is a member of the IEEE-USA Committee on Communications and Information Technology Policy (CCIP). The objective of the CCIP is to influence the communication and information technology policies of the executive and legislative branches of federal and state governments. His primary contribution is developing policies and models to defeat cyber security attacks. He has also contributed to IEEE-USA Committee on Communications Policy in the area of personal identification privacy and security.

Part One

Computer Engineering

Chapter 1

Digital Logic and Microprocessor Design

This chapter focuses on the fundamentals of digital logic and design, with numerous examples from both computer hardware design and "everyday life" events to demonstrate that digital logic is not confined to designing computers. My objective is to equip the engineer or student with sufficient knowledge of design principles to be able to design a digital computer. In addition, I integrate the important role that software plays in modern computer systems with the hardware design principles. Numerous design examples and solved problems are provided to support learning objectives.

MICROPROCESSOR DESIGN

Functions

Using its arithmetic logic unit(ALU), a microprocessor can perform mathematical and logic operations like addition, subtraction, multiplication, division, and comparison. Modern microprocessors contain complete floating-point processors that can perform extremely sophisticated operations on large variable-length numbers. In addition, a microprocessor can perform the following functions:

Move data from one memory location to another.

Make decisions and jump to a new set of computer program instructions based on those decisions.

Use an RD (read) and WR (write) line to tell the memory whether it wants to read from or write to the addressed location.

Use a clock line to transmit clock pulses (CPs) to sequence the microprocessor. For example, when numbers are added by the microprocessor, which you

Computer, Network, Software, and Hardware Engineering with Applications, First Edition. Norman F. Schneidewind.
© 2012 the Institute of Electrical and Electronics Engineers, Inc. Published 2012 by John Wiley & Sons, Inc.

will see later, addition takes place bit by bit, and the clock triggers each binary bit addition to ultimately form a decimal result.

Uses a reset line to reset the program counter to zero and restart execution.

Components

Microprocessor components are the building blocks of modern computers. These components are the following:

- ***ALU.*** Consists of accumulators, registers, and control unit.
  - The ALU executes instructions and manipulates data.
  - An 8-bit ALU can add, subtract, multiply, and divide two 8-bit numbers, while a 32-bit ALU can manipulate 8-bit, 16-bit, and 32-bit numbers.
  - An 8-bit ALU would have to execute four instructions to add two 32-bit numbers (four add instructions, each of which adds 8-bit numbers), whereas a 32-bit ALU can do it in one instruction.
- ***Accumulator.*** Holds data and instructions for processing by the ALU.
- ***Register.*** Temporary storage of instructions and data.
  - ***Program Counter* (PC).** Contains the address of next instruction to be executed
  - ***Instruction Register* (IR).** Holds address of current instruction being executed
  - ***General Registers.*** Holds operator (e.g., code for add instruction), operands (e.g., numbers to be added), and data while an instruction is executed
- ***Stack.*** Temporary storage of instructions and data, usually on a last in, first out (LIFO) basis. Also called push-down stack.
- ***Control Unit.*** Fetches and decodes instructions, generates signals for the ALU to execute instructions
- ***Busses***
  - ***Address Bus.*** Path over which addresses flow for directing memory and input/output (I/O) data transfers. An address bus may be 8, 16, or 32 bits wide that sends an address to memory or I/O for accessing memory or I/O.
  - ***Data Bus.*** Transfers data. A data bus may be 8, 16, or 32 bits wide that can send data to memory or I/O and receive data from memory or I/O. The number of address bus lines determine the amount of addressable memory (n lines = 2^n addressable words).
  - ***Control Bus.*** Communicates control and status information.
- ***Chip.*** A chip is also called an integrated circuit. Generally it is a small, thin piece of silicon onto which the transistors making up the microprocessor have been etched. A chip might be as large as an inch on a side and can contain tens of millions of transistors. Simpler processors might consist of a few

thousand transistors etched onto a chip just a few millimeters square. Microns are the width of the smallest wire on the chip. For comparison, a human hair is 100 μm thick. As the feature size on the chip goes down, the number of transistors rises.

Characteristics

Microprocessor characteristics govern the speed and functionality of computer operations. Important characteristics include the following presented in the succeeding paragraphs.

Smaller microprocessors can be combined into a larger one (four 4-bit microprocessors combined into one 16-bit microprocessor).

A crystal-controlled clock sequences the operations of a microprocessor (e.g., the sequence of computer program instruction execution) by generating CPs. Clock speed is specified in cycles per second, where 1 MHz is equal to 1 million cycles per second. Clock speed is the maximum speed of the chip.

Instructions require one or more clock cycles to execute the following, depending on its complexity: fetch instruction from memory, decode the operation code, fetch operands from memory, execute the instruction, and store the result in memory. In addition to clock speed, an important performance metric is the number of floating-point operations per second or flops.

***Complex instruction set computing* (CISC).** A single instruction can perform several operations. This design simplifies programming because, for example, a single instruction can fetch instruction from memory, decode the operation code, fetch operands from memory, execute the instruction, and store the result in memory. However, the downside is the relatively slow speed of the computer [RAF05].

***Reduced instruction set computing* (RISC).** Several operations are required to execute a single instruction. This design provides high speed, for example, well suited to real-time applications that must meet deadlines, but at the expense of relatively complex programming.

Performance

One measure of the computing power of a microprocessor is its clock speed, measured in millions of cycles per second (MHz). It usually takes from one to seven cycles of a microprocessor's internal clock to fully process an instruction. The faster the internal clock, the more instructions can be processed per unit of time. For the microprocessors in laptop and desktop computers, clock speeds are usually greater than 100 MHz. The fastest microprocessors can run at a speed of 2 GHz. From a user standpoint, the most important performance metric is program execution time, defined as [HAR07]:

Program execution time = (Number of instructions in program)
∗(Clock cycles per instruction)∗(Time per clock cycle).

Another measure of performance is the number of instructions that can be processed per second, referred to as MIPS, for million instructions per second. The MIPS rating of a microprocessor depends on both the clock speed and the number of instructions that can be executed per clock cycle. Simple microprocessors can execute a maximum of one instruction per clock cycle. Advanced microprocessors can execute up to six or eight instructions per clock cycle. The relationship between clock speed and MIPS is not straightforward, however, because some instructions may take more than one clock cycle to execute, depending on the program. The product of clock speed and the number of instructions that can be executed per cycle may be greater than MIPS. The maximum clock speed is a function of the manufacturing process and delays within the chip. MIPS is proportional to the clock speed and inversely proportional to the number of clock cycles per instruction.

Another indication of microprocessor speed is the word length, as measured by the number of bits of information that can be transferred simultaneously. Long words allow the microprocessor to handle data and perform complex tasks more efficiently. The number of bits per word has been steadily increasing with the growth of circuit technology. Thus 4-, 8-, 16-, 32-, and 64-bit microprocessors are now common. Some personal computers use 32-bit microprocessors. More powerful computers use 64-bit microprocessors. The 4-, 8-, or 16-bit devices are usually employed in simple embedded applications, such as microwave ovens, electric shavers, and televisions. Figure 1.1 shows the microprocessor architecture.

Pipeline Systems

An important aid to performance is the pipeline system. The purpose of a pipeline system is to reduce delay caused by the computer processor having to wait for instructions to complete. With a pipeline design, the processor begins the execution of the next instruction while the current instruction is executing. Thus, various phases of instruction execution are overlapped. The concept is to keep the pipeline full, with as many execution sequences as possible. For example, due to overlapped instruction execution, each instruction overlaps during $(n - 1)$ clock cycles, and each of $m = 4$ instructions requires one clock cycle, yielding $(n - 1) + m = 7$ clock cycles, total, as shown in Figure 1.2.

Problem: How is the *increase in speed*, obtained by a pipelined system over a conventional system, computed?

Answer: Using Figure 1.2 as an example, the increase is computed as follows:

The number of clock cycles required in conventional system is $mn = 4 * 4 = 16$ in the example of Figure 1.2. Thus, the decrease in number of clock cycles for a pipelined system is:

$$mn - ((n-1)+m) = 16 - 7 = 9,$$

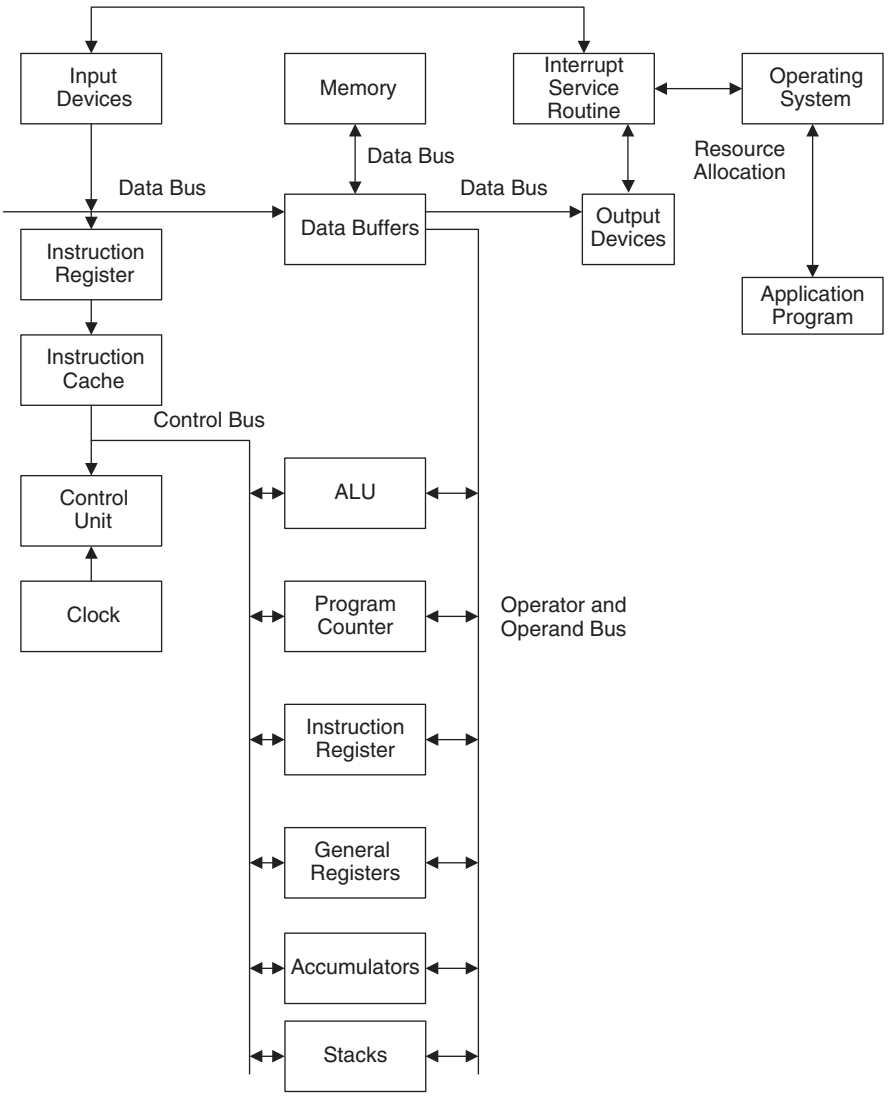


Figure 1.1 Microprocessor architecture.

and the *increase in speed* (number of clock cycles required in conventional system/ number of clock cycles required in a pipelined system) is:

$$(mn)/((n-1)+m) = n/(((n-1)/m)+1) = 16/7 = 2.286.$$

If m is large, the increase in speed approaches n clock cycles per instruction—maximum speed increase.

The pipeline *throughput* is defined as the *number of instructions*, m, per *total clock cycle time* required to process m instructions:

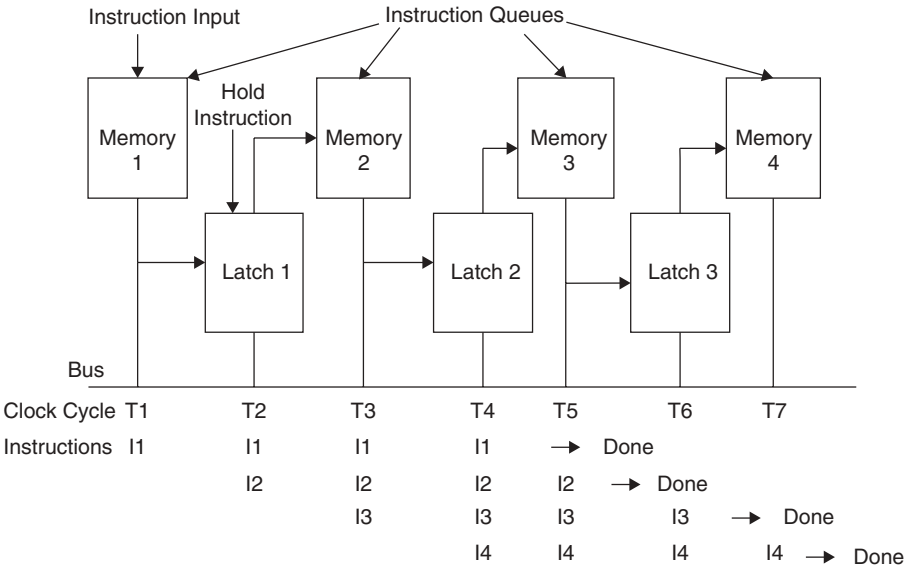


Figure 1.2 Pipelined system. n, clock cycle per instruction; m, instructions, each requiring one clock cycle; (n − 1) + m = 7 clock cycles (each instruction overlaps for [n − 1] clock cycles).

$$\frac{\text{m instructions}}{\text{Number of clock cycles per instruction} * \text{Time per clock cycle}} = \frac{\text{m}}{\text{m} + (\text{n} - 1)\text{T}},$$

where T is clock cycle time per instruction.

Problem: Compute the throughput of the pipeline microprocessor in Figure 1.2.

Answer: For a clock speed of 10 Mhz (10^7 clock cycles per second), T = $1/10^7$ seconds, the throughput is:

$$\text{m} / ((\text{m} + \text{n} - 1)\text{T}) = 4 / ((7)(1/107)) = (4)(107) / 7 = 5.71 \text{ MIPS}.$$

Pipeline efficiency is computed as: speed increase/maximum speed increase (n = 4 clock cycles per instruction) = 2.286/4 = 0.5715.

Pipeline System Delay

When a pipeline instruction is unable to complete on the scheduled clock cycle, then

- Finish the earlier instructions on schedule and
- Delay the later instructions
- This is called stalling the pipeline

*Structural hazard*s are pipeline hardware delays.

Example: Memory does not respond to a request as fast as it is expected.

Data hazards arise when data are not ready in a pipeline at the time they are needed.

Example: An instruction needs data in a register that a previous instruction is still modifying.

Control hazards arise when the central processing unit (CPU) needs to manage a pipeline but instead must increment the program counter.

Example: Nonpipelined conditional branch instruction jumps to a pipelined instruction.

Problem: Delay in a pipelined operation is illustrated in this problem that compares the clock cycle delay for nonjump instructions with that of jump instructions.

If a jump instruction is executed in the pipelined CPU in Figure 1.2, what is the clock cycle delay?

Answer: Since the target of the jump instruction (another instruction) cannot be decoded (i.e., program counter updated) until the jump instruction is executed, there is a delay of three clock cycles.

Problem: What cam be done in a pipeline system to maintain performance when a *structural hazard* occurs?

Answer: More resources can be employed, if available, or the pipeline can be stalled (i.e., no instructions executed until needed hardware is available).

Problem: Is the microprocessor architecture in Figure 1.1 a pipeline computer?

Answer: No, it is not because only one instruction can be executed at a time.

Problem: What determines the clock cycle frequency of a pipeline system?

Answer: The clock cycle frequency of a *pipeline system* is governed by the *pipeline* with the slowest processing time. For example, whichever pipeline queue in Figure 1.2 experiences the slowest processing determines clock cycle frequency.

Operating System

The operating system contains the software necessary to manage the resources of a computer system. An example is a signal called an interrupt that is used to indicate to the microprocessor that an I/O device needs attention (i.e., data input or data output) or that there is an error condition (e.g., attempted divide by zero). The interrupt service routine is shown in Figure 1.1. In addition to managing resources, the operating system is responsible for allocating resources, for example, allocating memory to the application program, as depicted in Figure 1.1.

Memory

Because computer performance depends on the characteristics of memory systems in addition to the microprocessor architecture, it is important to consider the former

[HAR07]. Two important types of memory systems are main memory (random access memory, RAM) and secondary memory (hard disk, USB flash). Main memory can be divided between a relatively slow RAM for program and data access and a fast cache memory for accessing recently used instructions and data. In addition, secondary memory can be classified as virtual, meaning that pages on a hard disk can be mapped to main memory locations under the control of a memory management unit. A microprocessor may be equipped with special hardware, called direct memory access (DMA), which allows I/O devices to communicate directly with memory rather than using intermediate devices (such as data buffers in Fig. 1.1).

RAM

RAM contains bytes of information that the microprocessor can read or write, depending on whether the RD or WR line is activated. One problem with RAM chips is that they are volatile; the RAM contents are lost once the power goes off. That is why the microprocessor needs read-only memory (ROM).

ROM

All microprocessors contain ROM. A ROM chip is programmed with a permanent collection of preset bytes. The address bus tells the ROM chip which byte to read and place on the data bus. The RD line signal causes the ROM chip to transfer the selected byte to the data bus. On a personal computer, the program in the ROM is called the BIOS (basic input/output system). When the microprocessor starts, it begins executing instructions it finds in the BIOS. The BIOS instructions test the hardware, and then control is transferred to the hard disk to fetch the boot sector. The boot sector is another small program that the BIOS stores in RAM after reading it from the disk. The microprocessor then begins executing the boot sector's instructions from RAM. The boot sector program will tell the microprocessor to fetch more instructions from the hard disk into RAM, which the microprocessor then executes, and so on. This is how the microprocessor loads and executes the entire operating system.

Read/Write (R/W) Control Line

This single wire is driven by the microprocessor to control memory functions. If the R/W control line is asserted as a logical 1 (i.e., true), then the microprocessor performs a read operation. If it is asserted as a logic 0 (i.e., false), then the microprocessor performs a write operation. The relationship between logic level and voltage level can vary, depending on the implementation. For example, a logical 0 corresponds to a voltage of 0 V, and a logical 1 corresponds to a voltage of 5 V. Figure 1.3 is a block diagram of the microprocessor and memory, showing the R/W control line.

Address Bus

These wires are controlled by the microprocessor to select a particular location in memory for reading or writing. The microprocessor in Figure 1.3 uses a memory chip that has 15 address wires.

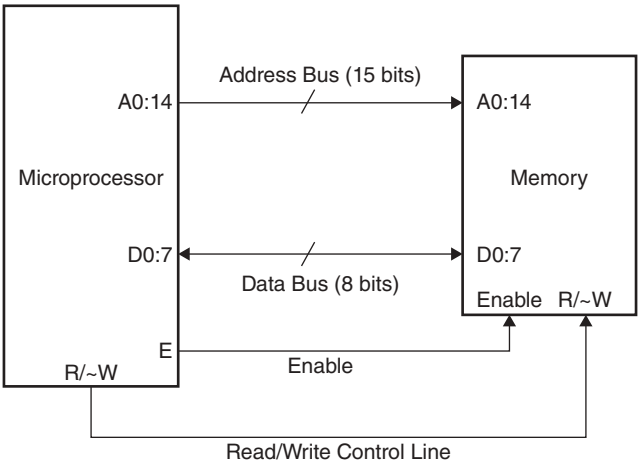


Figure 1.3 Diagram of microprocessor and memory.

Problem: How many locations can be addressed in Figure 1.3?

Answer: Since each wire has two states (it can be a digital 1 or a 0), $2^{15} = 32{,}768$ locations are possible. Thus, the system is said to have 32K of memory (1K = 1024 bytes).

Data Bus

These wires are used to pass data between the microprocessor and the memory. When data are written to the memory, the microprocessor drives the data bus; when data are read from the memory, memory drives the bus. In the example, in Figure 1.3, there are eight data wires (or bits). These wires can transfer one of 2^8 or 256 different binary values per transfer. The data size of 8 bits is commonly referred to as a byte. A data size of 4 bits is frequently referred to as a nibble.

Memory Enable Control Line

This wire, called the Enable line, connects to the enable circuitry of the memory in Figure 1.3. When the memory is enabled, it performs either a read or write operation as determined by the status of the R/W line.

Memory System Performance

Memory system performance is computed by considering hit and miss rates and the order of accessing memory components: cache memory, main memory, and hard disk. These rates are related to whether the instructions or data that are required by a program are available, first, in the cache memory, or second, in the main memory. If the instructions or data are in the cache, the access is scored as a cache hit; otherwise, the access is scored as a cache miss. Similarly, if the instructions or data

are not in the cache but are in main memory, the access is scored as a main memory hit; otherwise, the access is scored as a main memory miss because the instructions or data are only available on the hard disk [HAR07]. Thus, hit and miss rates are computed as follows:

$$\text{Cache hit rate (CHR)} = \frac{\text{Number of cache hits}}{\text{Total number of memory accesses}},$$

$$\text{Cache miss rate (CMR)} = \frac{\text{Number of cache misses}}{\text{Total number of memory accesses}},$$

$$\text{Main memory hit rate (MMHR)} = \frac{\text{Number of main memory hits}}{\text{Total number of memory accesses}},$$

$$\text{Main memory miss rate (MMMR)} = \frac{\text{Number of main memory misses}}{\text{Total number of memory accesses}},$$

$$\text{Number of hard disk accesses (HAD)} = \text{Total number of memory accesses} - (\text{Number of cache memory hits} + \text{Number of main memory hits} + \text{Number of main memory misses}).$$

Note that when there is a cache memory miss, the main memory access is attempted. Thus, it is not necessary to count cache memory misses in the foregoing computation:

$$\text{Hard disk access rate (HDAR)} = \text{HAD} / \text{Total number of memory accesses}.$$

Problem: For example, consider the following case:

4000 total number of memory accesses

1200 cache accesses are hits and 800 are misses

Of the 800 cache misses that require access to the main memory, 200 are hits and 600 are misses

Compute CHR, CMR, MMHR, MMMR, HAD, and HDAR.

Answer: CHR = 1200/4000 = 30%

CMR = 800/4000 = 20%

MMHR = 200/4000 = 5%

MMMR = 600/4000 = 1%

HAD = 4000 − (1200 + 200 + 600) = 2000

HDAR = 1200/4000 = 50%

Another memory performance metric is average access time (AAT), which is computed as follows:

$$\text{AAT} = \text{CHR} * (\text{cache access time}) + \text{MMHR} * (\text{main memory access time}) + \text{HDAR} * (\text{hard disk access time}).$$

Problem: For the following typical access times: cache = 2 ns, main memory = 60 ns, and hard disk = 35 ms, and using the above hit and miss access rates, compute the AAT.

Answer: AAT = (0.30)(2) + (0.04)(60) + (0.50)(35 $*$ 10^6) ns = 20.50 $*$ 10^6 ns (of course, hard disk access time dominates).

Multiplexing Data and Address Signals

On the Motorola 68HC11 microprocessor, in Figure 1.4, the 8-bit address/data bus takes turns acting as an address bus and a data bus. When a memory location is accessed (for reading or writing), the bus first acts as an address bus, transmitting the 8 lower-order bits of the address. Then the bus functions as a data bus, either transmitting a data byte (for a write cycle) or receiving a data byte (for a read cycle). This kind of split-personality bus is referred to as a multiplexed address and data bus. The support needed by the memory is provided by an 8-bit latch (a device that can store an address), using a multiplexed address/data bus. This chip (HC373) performs the function of latching the lower 8 address bits, when combined with the upper 7 address bits from the microprocessor, will provide the full 15-bit address for reading or writing data.

Figure 1.4 shows how the latch is wired. The upper 7 address bits run directly from the microprocessor to the memory. The lower 8 *address bits* are multiplexed with 8 *data bits*. When an *address* appears on the wires AD: 07, the latch connects the address bits of the microprocessor to the memory. On the other hand, when *data* appears on the wires AD0:7, the latch connects the data bits of the microprocessor to the memory. An additional signal, the address strobe (AS) output of the

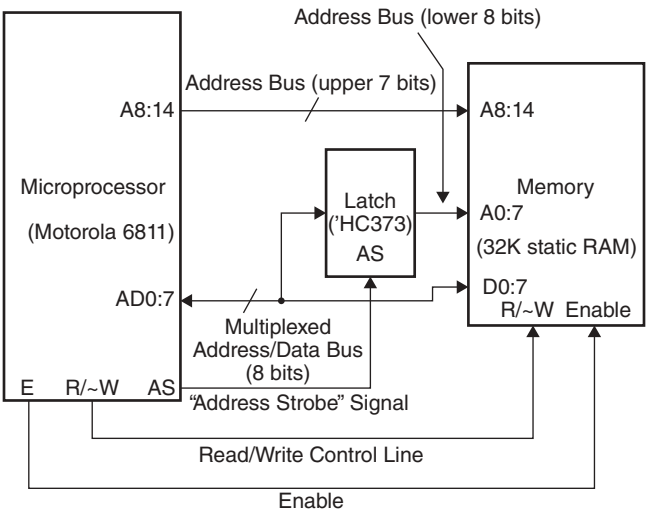


Figure 1.4 Block diagram of microprocessor and memory with latch.

microprocessor, tells the latch when to obtain the address bits from the address/data bus. When the full 15-bit address is available to the memory (upper 7 bits direct from the microprocessor (wires A8: 14) and lower 8 bits from the latch (wires AD: 07), the read or write access can occur. Because the address/data bus is also wired directly to the memory, data can flow in either direction between the memory and the microprocessor. The entire process is managed by the microprocessor. The Enable (E) clock, the R/W line, and the AS line perform in tight synchronization to make sure these operations happen in the correct sequence and within the timing capacities of the microprocessor hardware.

Memory Mapping the RAM

Memory mapping refers to allocating blocks of memory to different functions, such as the operating system and the application program. If a microprocessor has 15 address bits, it has 32,728 (32K bytes) of addressable locations that can be mapped. This address space would be used by the 32K memory chip in Figure 1.5. The technique used to map the memory is fairly simple. Whenever the microprocessor's A15 (the highest order address bit) is logic 1, the high-order address bit is selected. The other 15 address bits (A0 through A14) determine the address within that 32K block. If A15 is logic 0, the 32K block is not selected.

A NAND gate (actually a portion of a programmable logic device called a PAL) is used to enable the memory when A15 and the E Clock equal 1 in Figure 1.5. (See the "Digital Logic" section below for the explanation of NAND and other gates).

The E Clock controls the timing of the chip enable line. Some memory chips use an active low (sometimes called "active false") signal to enable inputs, meaning that they are enabled when the enable input is 0. The method for denoting an input that is active low (i.e., 0) is shown in Figure 1.5, where the chip enable input connects to a circle; this circle indicates an active low input. Also, the name for the signal, CE, is prefixed with a ~ symbol.

Interrupt Handling

The microprocessor has a bank of interrupt vectors, as shown in Figure 1.5, which are hardware-defined locations in the memory address space where the microproces-

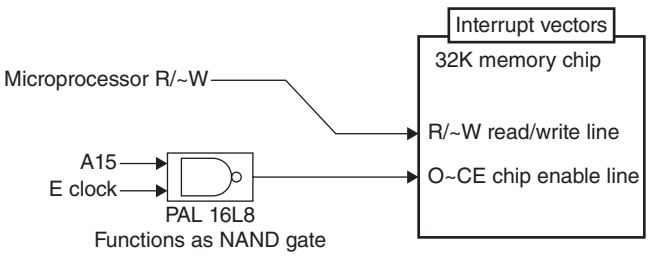


Figure 1.5 Enabling the memory.

Table 1.1 NOT Truth Table

Input	Output
A	$\overline{A}$
0	1
1	0

sor expects to find pointers to interrupt handling routines, for processing input and output data, arithmetic overflow, and so on. Also, when the microprocessor is reset, it finds the reset vector to determine where it should begin running a program. These vectors are located in the address space of the memory.

DIGITAL LOGIC

The fundamental logic operations of a microprocessor are performed by the following circuits. The results of those operations are represented in truth tables, where the binary value 0 is considered "low" (e.g., low voltage) and the binary value 1 is considered "high" (e.g., high voltage). While digital logic is used in the design of microprocessors, "everyday" examples are provided to show that the logic operations are not restricted to microprocessors.

NOT: represented in Table 1.1 and implemented with an inverter in Figure 1.6.

Application: The application is to complement the input A, producing the output $\overline{A}$.

Microprocessor example: the binary bit input was caused by an arithmetic overflow condition, so it is ignored and *not* used in the computation.

Everyday example: if we are to leave on an automobile trip, where A = 1 represents leaving at 1000, $\overline{A} = 0$ represents all times *not* equal to 1000.

OR: represented in Table 1.2 and implemented with OR gate in Figure 1.6.

Application: The application is to produce a 1 output if *any* or *both* of the inputs are 1.

Microprocessor example: the inputs are binary bits from memory stick or hard disk, so the microprocessor can accept *either* or *both* to perform a computation, depending on the current computer program instruction.

Everyday example: if A = 1 represents the decision to purchase a house and B = 1 represents the decision to purchase an automobile, Z = 1 represents the decision to purchase a house *or* an automobile *or* both.

AND: represented n Table 1.3 and implemented with an AND gate in Figure 1.6.

Application: The application is to produce a 1 output if *all* inputs are 1.

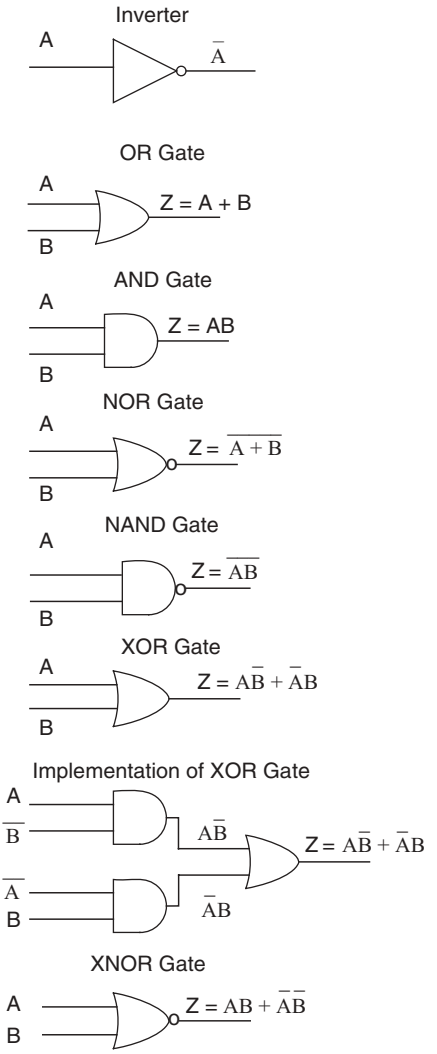


Figure 1.6 Logic operations.

Table 1.2 OR Truth Table

Input	Input	Output
A	B	Z = A + B
0	0	0
0	1	1
1	0	1
1	1	1

Table 1.3 AND Truth Table

Input	Input	Output
A	B	Z = AB
0	0	0
0	1	0
1	0	0
1	1	1

Table 1.4 NOR Truth Table

Input	Input	Output
A	B	$Z = \overline{A} + \overline{B}$
0	0	1
0	1	0
1	0	0
1	1	0

Table 1.5 NAND Truth Table

Input	Input	Output
A	B	$Z = \overline{AB}$
0	0	1
0	1	1
1	0	1
1	1	0

Microprocessor example: the microprocessor uses a signal Z = 1 to tell it that an interrupt has occurred on input line A *and* signifying that data input occurs on B, which the microprocessor will transfer to its memory.

Everyday example: if A = 1 represents a gas station and B = 1 represents a restaurant, we would stop our automobile at location Z, if Z has *both* a gas station *and* a restaurant.

NOR: represented in Table 1.4 and implemented with NOR gate in Figure 1.6.

Application: The application is to produce a 1 output if all inputs are 0.

Microprocessor example: the microprocessor Z = 1 output is recognized as interrupt code AB = 00.

Everyday example: if A = 0 represents the decision to *not* purchase a home and B = 0 represents the decision *not* to purchase an automobile, then Z = 1 represents the decision to *neither* purchase a home *nor* purchase an automobile.

NAND: represented in Table 1.5 and implemented with NAND gate in Figure 1.6.

Application: The application is to produce a 1 output if all inputs are *not* 1.

Microprocessor example: the microprocessor program produces the complement of the product of binary bits. This would be the case, for example, when Z = 1 signals that 0s occur on *either or both* of two input channels.

Everyday example: if A = 1 represents a gas station and B = 1 represents a restaurant, we would stop our automobile at location Z, if Z has only a gas station, or has only a restaurant, or has neither (i.e., rest stop).

Exclusive OR (XOR): represented in Table 1.6 and implemented with EXCLUSIVE OR gate in Figure 1.6. The figure also shows how the gate can be implemented, using AND and OR gates.

Application: The application is to produce a 1 output if *any* of the inputs is 1, but *not all* inputs are 1, and *not all* inputs are 0.

Microprocessor example: the main microprocessor receives a signal Z = 1 from the output of the I/O microprocessor that a binary bit A = 1 from a memory stick *or* B = 1 from a hard disk, and is ready for input, but these inputs are *not concurrent*.

Everyday example: if A = 1 represents the decision to purchase a house and B = 1 represents the decision to purchase an automobile, Z = 1 represents the decision to purchase a house *or* an automobile, but *not both at the same time*.

Exclusive NOR (XNOR): represented in Table 1.7 and implemented with XNOR gate in Figure 1.6. The *NOR* gate is the negation of the *XOR* gate from Table 1.6, as indicated in Table 1.7.

Table 1.6 EXCLUSIVE OR Truth Table

Input	Input	Output
A	B	$Z = A\overline{B} + \overline{A}B$
0	0	0
0	1	1
1	0	1
1	1	0

Table 1.7 EXCLUSIVE NOR (XNOR) Truth Table

Input	Input	Output
A	B	$Z = A\overline{B} + \overline{A}B = (A\overline{B})(\overline{A}B) = (\overline{A} + B)(A + \overline{B}) = \overline{A}A + AB + \overline{A}\overline{B} + B\overline{B} = AB + \overline{A}\overline{B}$
0	0	1
0	1	0
1	0	0
1	1	1

Application: The application is to produce a 1 output if all inputs are 0 *or* all inputs are 1.

Microprocessor example: Two hard drives are identified as A = 0 and A = 1; two flash memories are identified as B = 0, and B = 1. The microprocessor is programmed to input data from a hard drive and a flash memory *concurrently*. Therefore, it reads A = 0 *and* B = 0 *or* A = 1 *and* B = 1.

Everyday example: if A = 1 represents a gas station and B = 1 represents a restaurant, we would stop our automobile at location Z, if Z has *neither* a gas station *nor* a restaurant (i.e., rest stop) *or* has *both* a gas station and restaurant (i.e., get gas and eat).

De Morgan's theorem [GRE80] is used to simplify complex logic equations and the resultant digital logic. The theorem is used to simplify relatively simple expressions, as contrasted with Karnaugh maps (K-maps), described in the next section. The application of this theorem is shown in the following example:

$$\text{Theorem: } \overline{A+B} = \overline{A}\,\overline{B} \text{ and } \overline{AB} = \overline{A} + \overline{B}.$$

Suppose it is required to simplify $F = \overline{((\overline{AB})(\overline{AB}))}$.

Applying the theorem:

$$\overline{AB} = \overline{A} + \overline{B}, (\overline{AB})(\overline{AB}) = (\overline{A} + \overline{B})(\overline{A} + \overline{B})$$
$$= \overline{A}\ \overline{A} + \overline{A}\ \overline{B} + \overline{A}\ \overline{B} + \overline{B}\ \overline{B} = \overline{A} + \overline{A}\ \overline{B} + \overline{B} = \overline{A} + (\overline{A} + 1)\overline{B} = \overline{A} + \overline{B}$$
$$F = \overline{(\overline{A} + \overline{B})}(\overline{A} + \overline{B} = \overline{(\overline{A} + \overline{B})} + \overline{(\overline{A} + \overline{B})} = AB + AB = B.$$

Then, use Table 1.8 to demonstrate the equivalence between $\overline{((\overline{AB})(\overline{AB}))}$ and AB.

K-MAPS

A K-map in Table 1.9 is used to minimize a complex Boolean expression [RAF05]. Each square of a K-map represents a minterm (i.e., product terms). The process proceeds by listing the binary equivalents of the terms A and BC on the axes of Table 1.9, ordering them so that there is only a 1-bit difference between adjacent cells. Then, the minimum number of cells is enclosed. Next, minterms are identified

Table 1.8 Truth Table to Demonstrate Equivalence between F and AB

A	B	$\overline{AB}$	$\overline{AB}\,\overline{AB}$	$F = \overline{((\overline{AB})(\overline{AB}))}$	AB
0	0	1	1	0	0
0	1	1	1	0	0
1	0	1	1	0	0
1	1	0	0	1	1

according to terms that are common to all cells in the enclosure. Last, the product terms are summed. Notice what a clever method this is. Minimization is achieved by noting the combination of terms that yields the minimum difference!

Example: Simplify $F = \overline{A}\ \overline{B}\ \overline{C} + A\ \overline{B}\ \overline{C} + \overline{A}\ \overline{B}\ C + A\ \overline{B}\ C$.

Table 1.9 K-Map for $F = \overline{A}\overline{B}\overline{C} + A\overline{B}\overline{C} + \overline{A}\overline{B}C + A\overline{B}C$

		$\overline{B}\,\overline{C}$	$\overline{B}C$	BC	$B\overline{C}$
		00	01	11	10
$\overline{A}$	0	1	1		
A	1	1	1		

In minterm form, $F = \overline{A}\ \overline{B}\ \overline{C} + A\ \overline{B}\ \overline{C} + \overline{A}\ \overline{B}\ C + A\ \overline{B}\ C = \overline{B}$

In the K-map, $\overline{B}$ is common to the enclosed minterms. Therefore, $F = \overline{B}$. Table 1.10 demonstrates this result. The considerable reduction from the original function would result in significant savings in circuitry to implement the function.

Prime Implicant

A prime implicant is the *product term* obtained by enclosing the *maximum* number of adjacent cells in a K-map. For example, in the K-map of Table 1.9, $F = \overline{B}$ is a prime implicant. The prime implicant is only useful for providing a name for the maximum enclosure in a K-map.

Quine–McCluskey Method

This method is an alternative to the K-map for minimizing a Boolean function. The method is illustrated in Table 1.11 by minimizing the function $F = \overline{A}\ \overline{B}\ \overline{C} + A\ \overline{B}\ \overline{C} + \overline{A}\ \overline{B}\ C + A\ \overline{B}\ C$, where these minterms are placed in Table

Table 1.10 F Function Truth Table

A	B	C	$F = \overline{A}\ \overline{B}\ \overline{C} + A\ \overline{B}\ \overline{C} + \overline{A}\ \overline{B}\ C + A\ \overline{B}\ C$	$\overline{F} = \overline{B}$
0	0	0	1	1
0	0	1	1	1
0	1	0	0	0
0	1	1	0	0
1	0	0	1	1
1	0	1	1	1
1	1	0	0	0
1	1	1	0	0

Table 1.11 Quine–McCluskey Method for $F = \overline{A}\ \overline{B}\ \overline{C} + A\ \overline{B}\ \overline{C} + \overline{A}\ \overline{B}\ C + A\ \overline{B}\ C = \overline{B}$

Minterm		ABC	Difference of 1 Minterms		Difference of 1 Minterms	Minterms	Prime implicant
0	$\overline{A}\ \overline{B}\ \overline{C}$	000	0,1	00-	0,1,4,5	-0-	$\overline{B}$
1	$\overline{A}\ \overline{B}\ C$	001					
4	$A\ \overline{B}\ \overline{C}$	100	4,5	10-			
5	$A\ \overline{B}\ C$	101					

Table 1.12 One-Bit Adder Truth Table

A	B	Q	CO
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

1.11. This method is used to represent a difference of 1 between two adjacent minterms, such as $\overline{A}\ \overline{B}\ \overline{C}$ and $\overline{A}\ \overline{B}\ C$, yielding $\overline{A}\ \overline{B}\text{-} = 00\text{-}$. The symbol - is placed where there is a difference in minterm bit values, such as between 00- and 10- in Table 1.11, yielding -0-. This process continues until the four minterms 0, 1, 4, and 5 show a difference of 1 (00- compared with 10-), yielding prime implicant $\overline{B}$(-0-). The same result is obtained as was the case using the K-map in Table 1.9. Of the two methods, the K-map is easier to apply.

COMBINATIONAL CIRCUITS

These are circuits that use logic gates to produce outputs at any time that are only dependent on the *current* values of the inputs, meaning that it is not necessary to use a CP to trigger outputs [HAR07]. A typical combinational circuit is the adder.

One-Bit Adder with Carry Out

A and B are added, producing Q output and CO (carry out). Q and CO are implemented according to the truth table shown in Table 1.12.

Two-bit Adder with Carry In and CO

What if you want to add two 8-bit bytes? This becomes slightly harder. In this case, you need to create a full binary adder. The difference between a full adder and the

Table 1.13 Two-Bit Adder Truth Table

					Q = 1	CO = 1
CI	A	B	Q	CO	Minterms	Minterms
0	0	0	0	0		
0	0	1	1	0	$\overline{CI}\ \overline{A}\ B$	
0	1	0	1	0	$\overline{CI}\ A\ \overline{B}$	
0	1	1	0	1		$\overline{CI}\ A\ B$
1	0	0	1	0	$CI\ \overline{A}\ \overline{B}$	
1	0	1	0	1		$CI\ \overline{A}\ B$
1	1	0	0	1		$CI\ A\ \overline{B}$
1	1	1	1	1	$CI\ A\ B$	$CI\ A\ B$

Q Product Terms: $\overline{CI}\ \overline{A}\ B + \overline{CI}\ A\ \overline{B} + CI\ \overline{A}\ \overline{B} + CIAB$

$Q = \overline{CI}\ (\overline{A}\ B + A\ \overline{B}) + CI\ (\overline{A\ B} + AB)$

CO Product Terms: $\overline{CI}A\ B + CI\ \overline{A}\ B + CI\ A\ \overline{B} + CI\ A\ B = AB\ (\overline{CI} + CI) + CI(\overline{A}\ B + A\ \overline{B})$

$CO = AB + CI\ (\overline{A}\ B + A\ \overline{B})$

Table 1.14 K-Map for $Q = \overline{CI}\ \overline{A}\ B + \overline{CI}\ A\ \overline{B} + CI\ \overline{A}\ \overline{B} + CIAB = \overline{CI}(\overline{A}\ B + A\ \overline{B}) + CI(\overline{A}\ \overline{B} + AB)$

	AB			
CI	00	01	11	10
0		1		1
0	1		1	

$CI\overline{A}\ \overline{B}$ $\quad \overline{CI}\overline{A}\ B$ $\quad CIAB$ $\quad \overline{CI}A\ \overline{B}$

1-bit adder is that a full adder accepts A and B inputs plus a carry-in (CI) input. Once you have a full adder, you can string eight of them together to create a byte-wide adder and cascade the carry bit from one adder to the next. The truth table for a full adder is slightly more complicated than the previous truth table because now there are 3 input bits.

A combinational circuit minterm is represented by a product in a row of the truth table as shown in Table 1.13, corresponding to a 1 in the Q or CO output columns; for example, the fourth row for CO and the second row for Q in Table 1.13 [GIB80]. The values of Q and CO product terms are obtained by ORing the products in each row of Table 1.13 where Q = 1 or CO = 1, and then summing these terms, followed by simplifying the expressions, as demonstrated in Table 1.13. Further simplification *may* be possible by using a K-map.

As can be seen in Table 1.14, the adder output Q cannot be simplified by using a K-map because there are no adjacent cells. However, simplification is achieved

Table 1.15 K-Map for Carry Out $(CO) = \overline{C}IAB + C\overline{I}AB + CIA\overline{B} + CIAB = AB + CI(\overline{A}B + A\overline{B})$

	AB			
CI	00	01	11	10
0			1	
1		1	1	1

$CI\,\overline{A}\,B$ $\quad \overline{C}IAB$ $\quad CIAB$ $\quad AB$ $\quad CIA\,\overline{B}$

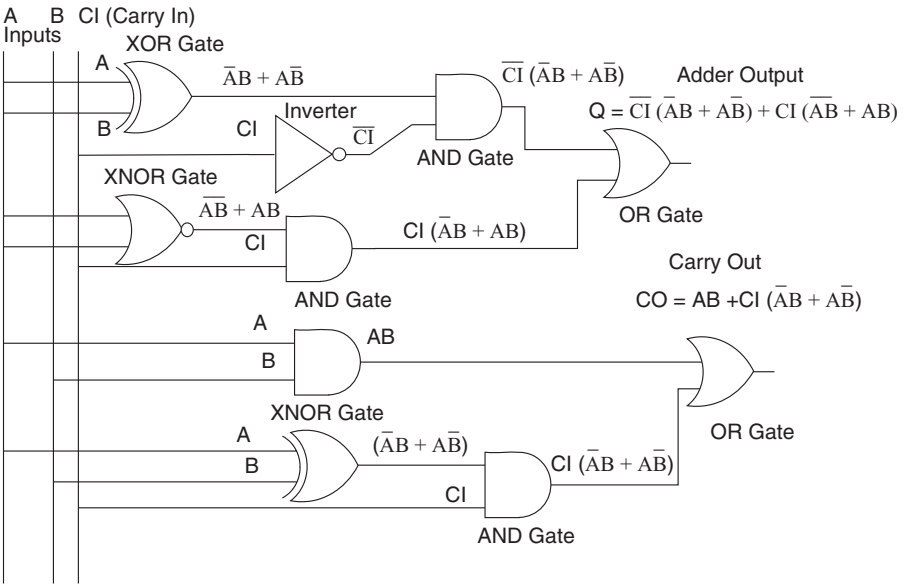


Figure 1.7 Adder circuit.

for CO, as shown in Table 1.15, producing $CO = AB + CI(\overline{A}B + A\overline{B})$. The relevant minterm cells in Table 1.15 that comprise the minimized function are outlined in red. Minterm logic is called *sum of products*. The full adder logic that corresponds to the minterms in Table 1.13 is shown in Figure 1.7, showing the adder output Q and the CO.

MULTIPLE OUTPUT COMBINATIONAL CIRCUITS

Combinational circuits can have multiple outputs [RAF05]. Each output is expressed as a function of the inputs, as shown in Table 1.16, where the inputs are binary-coded decimal (BCD) bits W, X, Y, and Z, corresponding to the decimal digits 0,…, 9. A

Table 1.16 Truth Table for Binary-Coded Decimal (BCD) Converter

Decimal digit	BCD input bits				Computer display segment output bits						
	W	X	Y	Z	a	b	c	d	e	f	g
0	0	0	0	0	**1**	**1**	**1**	**1**	**1**	**1**	0
1	0	0	0	1	0	**1**	**1**	0	0	0	0
2	0	0	1	0	**1**	**1**	0	**1**	**1**	0	**1**
3	0	0	1	1	**1**	**1**	**1**	**1**	0	0	**1**
4	0	1	0	0	0	**1**	**1**	0	0	**1**	**1**
5	0	1	0	1	**1**	0	**1**	**1**	0	**1**	**1**
6	0	1	1	0	0	0	**1**	**1**	**1**	**1**	**1**
7	0	1	1	1	**1**	**1**	0	**1**	0	0	0
8	1	0	0	0	**1**	**1**	**1**	**1**	**1**	**1**	**1**
9	1	0	0	1	**1**	**1**	**1**	0	0	**1**	**1**

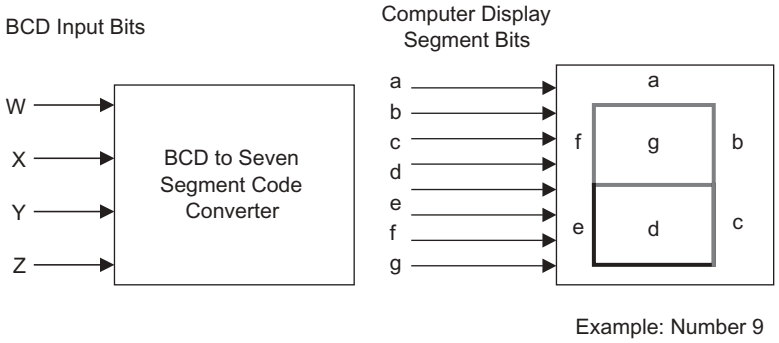


Figure 1.8 BCD to seven-segment code converter.

binary coded decimal converter is an example shown in Figure 1.8, showing how the number 9 can be displayed. The outputs are computer display segment bits a,…, g that represent the 1s necessary to generate the display decimal numbers. The code converter transforms the BCD numbers 0000,…, 1001 to display segments. The converter does not represent decimal numbers greater than 9. The K-maps use "don't cares" = Xs in order to simplify the logic; the "don't cares" should not be confused with the BCD bit = X in Table 1.16. The "don't cares" are used to advantage in forming minterms, as, for example, in Tables 1.17–1.23.

In order to generate the K-maps, place a 1 in the K-map cells corresponding to the 1s that appear in Table 1.16. For example, for *segment a* in Table 1.17, a 1 is recorded in the cell WXYZ = 0000, corresponding to the **1** (bolded) in the *segment a* column in Table 1.16.

The K-maps will lead to simplifying the equations for the seven-segment computer display (Fig. 1.8). The equations will then be used to design the digital logic circuit in Figures 1.9 and 1.10.

Table 1.17 K-Map for Segment a

	YZ			
WX	00	01	11	10
00	1		1	1
01		1	1	
11	X	X	X	X
10	1	1	X	X

W $\quad \overline{W}\,\overline{X}\,\overline{Z}$ $\quad XZ$ $\quad YZ$

$a = W + \overline{W}\,\overline{X}\,\overline{Z} + Z(X + Y)$.

Table 1.18 K-Map for Segment b

	YZ			
WX	00	01	11	10
00	1	1	1	1
01	1		1	
11	X	X	X	X
10	1	1	X	X

$\overline{Y}\,\overline{Z}$ $\quad W\ \overline{W}\,\overline{X}$ $\quad YZ$

$b = W + \overline{W}\,\overline{X} + YZ + \overline{Y}\,\overline{Z}$.

Table 1.19 K-Map for Segment c

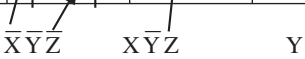
	YZ			
WX	00	01	11	10
00	1	1		
01	1	1		1
11	X	X	X	X
10	1	1	X	X

W $\quad \overline{Y}$ $\quad \overline{X}\,YZ$ $\quad XY\,\overline{Z}$

$c = W + \overline{Y} + \overline{X}\,YZ + XY\,\overline{Z} = W + \overline{Y} + Y(\overline{X}Z + X\overline{Z})$.

Table 1.20 K-Map for Segment d

	YZ			
WX	00	01	11	10
00	1		1	1
01		1	1	1
11	X	X	X	X
10	1		X	X



$d = \overline{X}\,\overline{Y}\,\overline{Z} + X\overline{Y}Z + Y = \overline{Y}(\overline{X}\,\overline{Z} + XZ) + Y.$

Table 1.21 K-Map for Segment e

	YZ			
WX	00	01	11	10
00	1			1
01				1
11	X	X	X	X
10	1		X	X

$e = \overline{Z}(\overline{X}\,\overline{Y} + Y).$

Table 1.22 K-Map for Segment f

	YZ			
WX	00	01	11	10
00	1			
01	1	1		1
11	X	X	X	X
10	1	1	X	X

$f = \overline{Z}(\overline{Y} + XY) + W + X\overline{Y}.$

Table 1.23 K-Map for Segment g

	YZ			
WX	00	01	11	10
00			1	1
01	1	1		1
11	X	X	X	X
10	1	1	X	X

$W \quad \overline{W}X\overline{Y} \quad \overline{W}\,\overline{X}Y \quad Y\overline{Z}$

$g = \overline{W}(X\overline{Y} + \overline{X}Y) + W + Y\overline{Z}.$

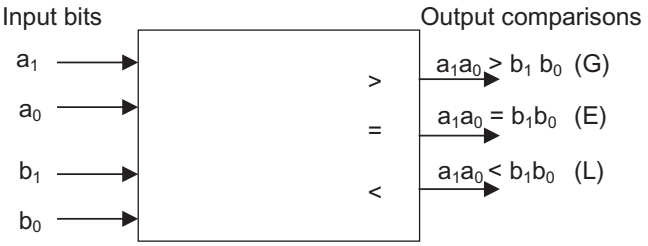


Figure 1.9 Two-bit comparator block diagram.

Comparators

A comparator is another type of combinational circuit. Its block diagram is shown in Figure 1.9 and the corresponding logic diagram is shown in Figure 1.10. For example, as Figure 1.10 shows, a comparator can be designed to compare two 2-bit quantities for greater-than (G), equal-to (E), and less-than (L) conditions. By minimizing the logic in Table 1.24, as accomplished by the K-maps in Tables 1.25–1.27, the logic circuit is designed in Figure 1.9. The K-maps are generated by recording a 1 in cells corresponding to 1s in Table 1.24; for example, placing a 1 in the cells a_1, a_0, b_1, and $b_0 = 0100$ for G in Table 1.24. Notice, as opposed to previous examples, there are no "don't care" conditions because all four comparator bits are relevant.

Decoders

A decoder is a combinational circuit that, when enabled, selects one of 2^n inputs and produces a 1 output, where n is the number of input bits, as shown in Figure 1.11. After this block diagram is displayed, the truth table (Table 1.28), is formulated, showing the relationship between inputs and outputs, where an output term 1 is

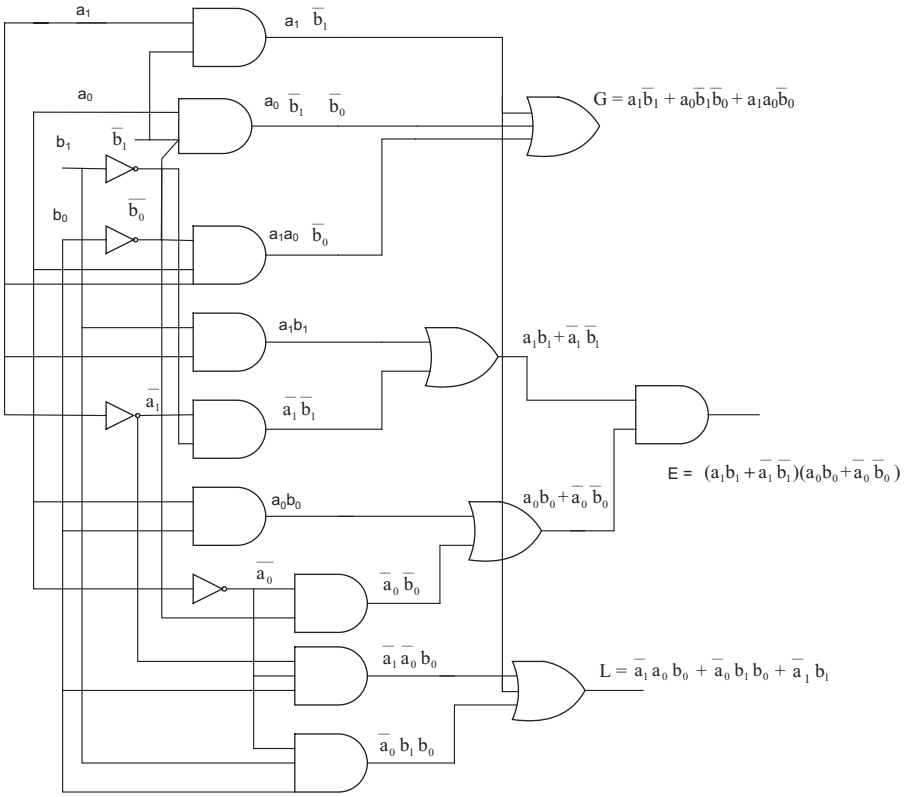


Figure 1.10 Two-bit comparator logic diagram.

generated according to the appearance of 0s and 1s in the inputs columns; for example, $d_3 = E\,\bar{x}_1\,\bar{x}_0 = 1$ for $E\,\bar{x}_1\,\bar{x}_0 = 100$.

Finally, Table 1.28 is used to design the logic diagram in Figure 1.11. Applying K-maps to minimize the logic of the truth table is not necessary because there is only a single 1 output for each combination of inputs in Table 1.28. However, the truth table is used to generate the output equations, which will lead to the design of the logic diagram in Figure 1.11. An application of the decoder is to select an operand (i.e., 4-bit output $d_0d_1d_2d_3$) in a computer instruction, based on the operation code (i.e., 2-bit input x_1x_0) in the instruction, when the instruction execution enable is high (E = 1).

Encoders

Encoders produce n output bits in accordance with the value of 2^n input bits, as shown in the block diagram of Figure 1.12. Like the decoder, it is not necessary to develop K-maps of the outputs as a function of the inputs because of the inherent simplicity of the circuit logic in Figure 1.12. Equations that emerge from the

Table 1.24 Truth Table for Two-Bit Comparator

Inputs				Outputs		
a_1	a_0	b_1	b_0	G: $a_1a_0 > b_1b_0$	E: $a_1a_0 = b_1b_0$	L: $a_1a_0 < b_1b_0$
0	0	0	0	0	**1**	0
0	0	0	1	0	0	**1**
0	0	1	0	0	0	**1**
0	0	1	1	0	0	**1**
0	1	0	0	**1**	0	0
0	1	0	1	0	**1**	0
0	1	1	0	0	0	**1**
0	1	1	1	0	0	**1**
1	0	0	0	**1**	0	0
1	0	0	1	**1**	0	0
1	0	1	0	0	**1**	0
1	0	1	1	0	0	**1**
1	1	0	0	**1**	0	0
1	1	0	1	**1**	0	0
1	1	1	0	**1**	0	0
1	1	1	1	0	**1**	0

Table 1.25 K-Map for Output G:$a_1a_0 > b_1b_0$

Inputs		Inputs b_1b_0			
		00	01	11	10
a_1a_0	00				
	01	1			
	11	1	1		1
	10	1	1		

$a_0\overline{b_1}\,\overline{b_0}$ $a_1\overline{b_1}$ $a_1a_0\overline{b_0}$

$G = a_0\overline{b_1}\,\overline{b_0} + a_1\overline{b_1} + a_1a_0\overline{b_0}.$

Table 1.26 K-Map for Output E:$a_1a_0 = b_1b_0$

Inputs		Inputs b_1b_0			
		00	01	11	10
a_1a_0	00	1			
	01		1		
	11			1	
	10				1

$\overline{a_1}\,\overline{a_0}\,\overline{b_1}\,\overline{b_0}$ $\overline{a_1}\,a_0\,\overline{b_1}\,b_0$ $a_1a_0b_1b_0$ $a_1\overline{a_0}\,b_1\,\overline{b_0}$

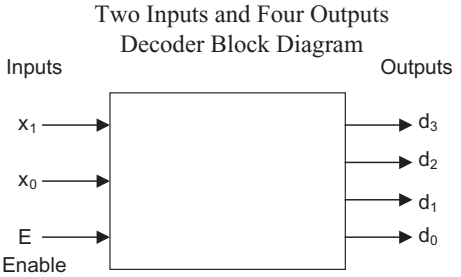
$$E = \overline{a_1}\,\overline{a_0}\,\overline{b_1}\,\overline{b_0} + \overline{a_1}a_0\overline{b_1}b_0 + a_1a_0b_1b_0 + a_1\overline{a_0}b_1\overline{b_0}$$
$$= \overline{a_1}\,\overline{b_1}(a_0b_0 + \overline{a_0}\,\overline{b_0}) + a_1b_1(a_0b_0 + \overline{a_0}\,\overline{b_0}) = (a_1b_1 + \overline{a_1}\,\overline{b_1})(a_0b_0 + \overline{a_0}\,\overline{b_0}).$$

Table 1.27 K-Map for Output L:$a_1a_0 < b_1b_0$

Inputs		Inputs b_1b_0			
		00	01	11	10
a_1a_0	00		1	1	1
	01			1	1
	11				
	10				

$\overline{a_1}\,\overline{a_0}\,b_0$ $\overline{a_0}\,b_1\,b_0$ $\overline{a_1}\,b_1$

$$L = \overline{a_1}\,\overline{a_0}b_0 + \overline{a_0}b_1b_0 + \overline{a_1}b_1.$$



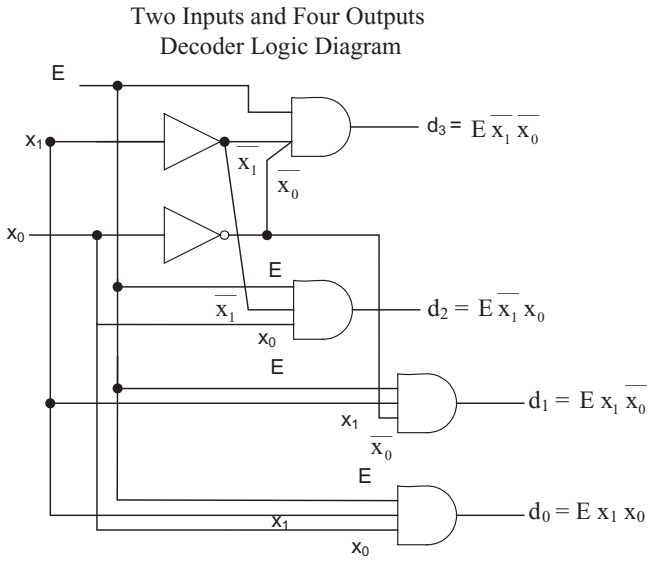


Figure 1.11 Two inputs and four outputs decoder block and logic diagrams.

Table 1.28 Truth Table for Two Inputs and Four Outputs Decoder

Inputs			Outputs			
E (Enable)	x_1	x_0	d_3	d_2	d_1	d_0
1	0	0	**1**	0	0	0
1	0	1	0	**1**	0	0
1	1	0	0	0	**1**	0
1	1	1	0	0	0	**1**

$d_3 = E\overline{x_1}\,\overline{x_0};\ d_2 = E\overline{x_1}x_0;\ d_1 = Ex_1\overline{x_0};\ d_0 = Ex_1x_0.$

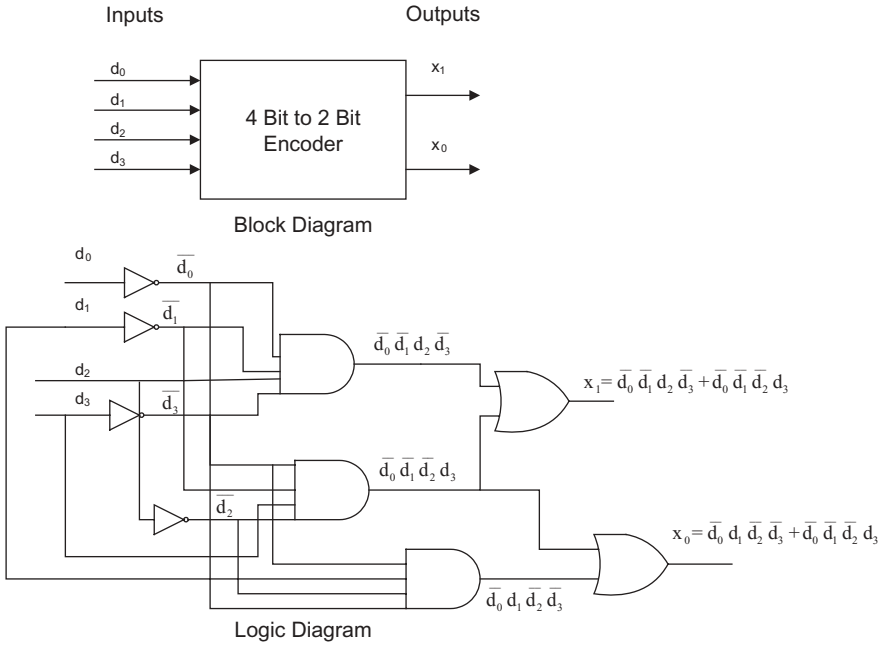


Figure 1.12 The 4-bit to 2-bit encoder block and logic diagrams.

relationships in the truth table (Table 1.29) are used to design the logic circuit in Figure 1.12. The outputs x_1 and x_0 are generated as the sum of the products of inputs where there are **1**s in the x_1 and x_0 columns as signified by the bolded quantities.

An application of the encoder is data compression in which we could shrink 4 bits of input to 2 bits of output in a database application that deals with large quantities of data. For example, representing $d_0d_1d_2d_3 = 0100$ as $x_1x_0 = 01$.

Table 1.29 Truth Table for 4-Bit to 2-Bit Decoder

Inputs				Outputs	
d_0	d_1	d_2	d_3	x_1	x_0
1	0	0	0	0	0
0	1	0	0	0	**1**
0	0	1	0	**1**	0
0	0	0	1	**1**	**1**

$x_1 = \overline{d_0}\,\overline{d_1}d_2\overline{d_3} + \overline{d_0}\overline{d_1}\overline{d_2}d_3$, $x_0 = \overline{d_0}d_1\overline{d_2}\overline{d_3} + \overline{d_0}\overline{d_1}\overline{d_2}d_3$.

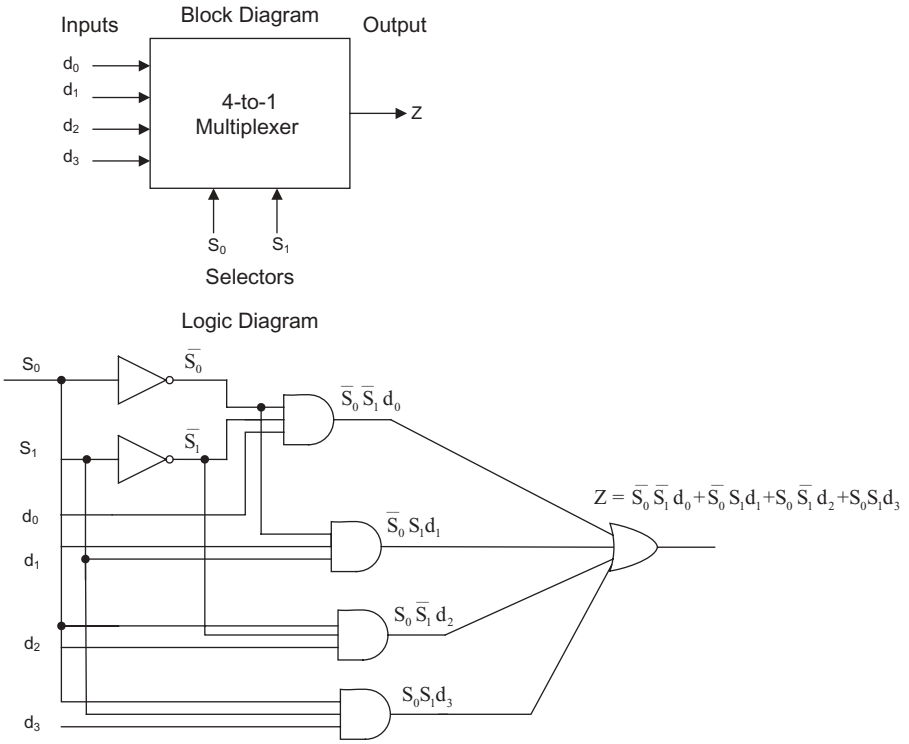


Figure 1.13 The 4-to-1 multiplexer block and logic diagrams.

Multiplexers

A multiplexer acts as a data selector, meaning that if the multiplexer has n select lines, one of 2^n inputs can be selected as the output. For example, in Figure 1.13, using selector lines S_0 and S_1, one of four inputs, d_0, d_1, d_2, d_3, can be selected at the output Z. The output equation for Z is derived from Table 1.30, noting that a given output is produced for given values of the selectors, for example, $Z = d_0$ when

Table 1.30 Truth Table for 4-to-1 Multiplexer

Selector		Output
S0	S1	Z
0	0	d_0
0	1	d_1
1	0	d_2
1	1	d_3

$Z = \overline{S_0}\,\overline{S_1}d_0 + \overline{S_0}S_1d_1 + S_0\overline{S_1}d_2 + S_0S_1d_3.$

$\overline{S_0}\,\overline{S_1} = 11$. Multiplexers differ from decoders and encoders by virtue of select lines that cause inputs to be produced at the output. An application is to combine data received from the Internet on input lines d_0, d_1, d_2, and d_3 onto a single microprocessor memory line Z, if an Internet interrupt has occurred, that has a code represented by selector lines S_0S_1.

Demultiplexers

A demultiplexer causes an input x to be transferred to one of 2^n output lines, where n is the number of select inputs in Figure 1.14. Output equations for a demultiplexer with two select inputs and four outputs are shown in the truth table, Table 1.31. The demultiplexer does the reverse of the multiplexer; for example, it distributes Internet data, which have been multiplexed on input line x, to each of four microprocessor output ports d_0, d_1, d_3, and d_4.For example, Internet data will be distributed to output port d_0 when $\overline{S_0}\ \overline{S_1} = 11$ in Table 1.31.

SEQUENTIAL CIRCUITS

A *clocked synchronous sequential circuit* uses flip-flops to store data, and its outputs depend on both the *previous* and *current* values of inputs [HAR07]. These circuits are called state machines, wherein states are stored in flip-flops, and state changes are triggered by CPs. In an *asynchronous sequential circuit*, the completion of an operation starts the next operation (i.e., a clock is not needed).

Flip-Flops and Latches

A flip-flop is a *clocked synchronous sequential circuit* with a 1-bit memory. The output of the flip-flop can be changed by the rising or falling edge of a CP. A clock prevents the flip-flop from changing state when spurious inputs occur. Instability can arise if inputs change during the CP. This problem is avoided by holding data stable for specified periods of time before and after the CP. The former period is called *setup time* and the latter is called *hold time*.

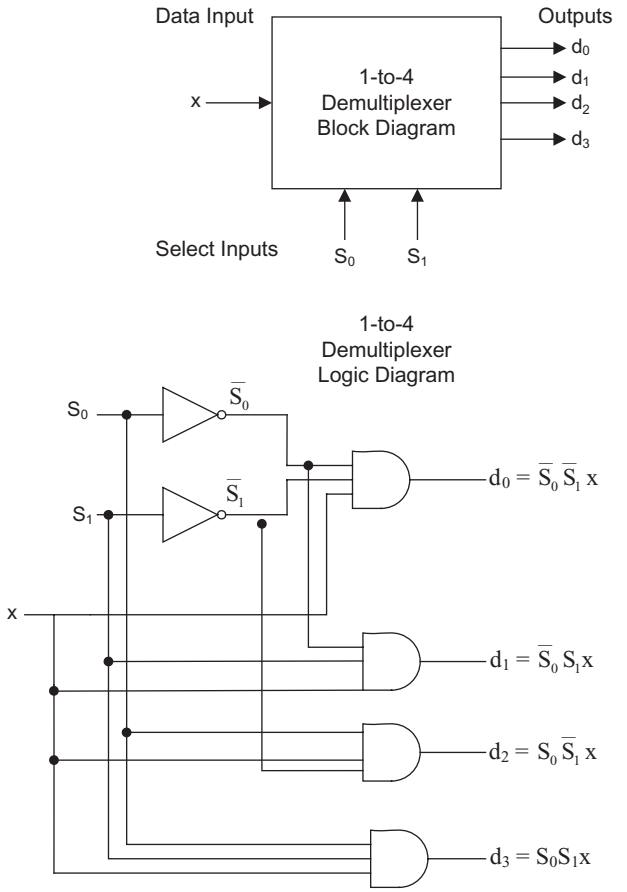


Figure 1.14 The 1-to-4 demultiplexer block and logic diagrams.

Table 1.31 Truth Table for 1-to-4 Demultiplexer

Select inputs		Data input	Data output			
S_0	S_1		d_0	d_1	d_2	d_3
0	0	x	$\overline{S_0}\overline{S_1}x$	0	0	0
0	1	x	0	$\overline{S_0}S_1x$	0	0
1	0	x	0	0	$S_0\overline{S_1}x$	0
1	1	x	0	0	0	S_0S_1x

$d_0 = \overline{S_0}\overline{S_1}x$; $d_1 = \overline{S_0}S_1x$; $d_2 = S_0\overline{S_1}x$; $d_3 = S_0S_1x$.

Table 1.32 SR Latch Truth Table Using NOR Gates

S	Q(t) (present state)	R	$\overline{Q}$(t) (present state)	Q(t + 1) (Gate #1) (next state)	$\overline{Q}$(t + 1) (next state)
0	0	0	1	0 (no change)	1(no change)
1	0	1	1	0 (illegal)	0 (illegal)
0	1	1 (reset)	0	0 (change state)	1(change state)
1 (set)	0	0	1	1(change state)	0(change state)

Flip-flops use storage circuits called latches. The term "latch" refers to the ability to receive and hold data (set) until the latch is reset. The most common latch is the SR (set–reset). An application of a latch is to set and hold an interrupt flag when an input device needs attention by the microprocessor. A flip-flop is a latch with clock input (CLK). Flip-flops implement changes in circuit states that are triggered by a CP. For example, when the CP and the input line cause the flip-flop to assume the set state, a computer program would execute a branch operation; when the CP and the input line cause the flip-flop to assume the reset state, a computer program would return to the main line of the program. An interesting question is how a latch or flip-flop manages to be in the initial state. The answer is that the latch or flip-flop will be in the initial state determined by the initial state settings wired into the flip-flop.

SR Latch

The logic rules of the SR latch are the following:

NOR Gate output = 1, if *all* inputs = 0; output = 0, if *any* input = 1.

These rules are applied in the truth table shown in Table 1.32 and the logic diagram in Figure 1.15. Notice in Table 1.32 and Figure 1.15 that there are illegal next states in the case of S = 1 and R = 1 because it is not possible to simultaneously set and reset the latch.

Reset–Set (RS) Flip-Flop

The RS flip-flop is a clocked SR latch. This flip-flop is important because all other flip-flops are derived from it. Figure 1.16 shows the implementation of this flip-flop using NAND gates and the truth table, Table 1.33, shows the gate relationships for present state at time t and next state at time (t + 1), including simultaneous set and reset that should be avoided. In Figure 1.16, notice that there is feedback from Gate 3 to Gate 4 of Q(t + 1) and from Gate 4 to Gate 3 of $\overline{Q}$(t + 1).

The design in Figure 1.16 is obtained by employing the equations below, which in turn are obtained from Table 1.33 and the K-map in Table 1.34. The components of the equations are annotated on Figure 1.16. The K-map is constructed by noting whether the next state output Q(t + 1) is a 1. If it is, the corresponding present state

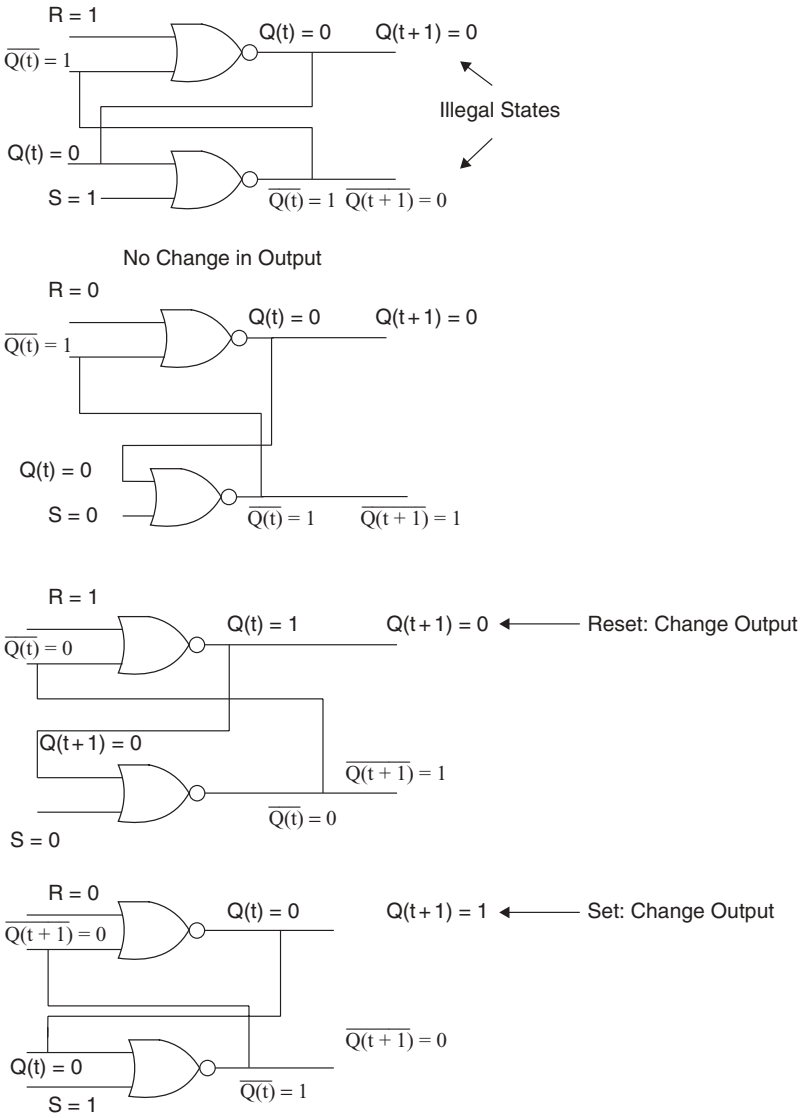


Figure 1.15 SR latch logic diagram.

output Q(t) is inserted into the K-map. The corresponding next state and present state outputs are bolded in Table 1.33. You can see that Table 1.33 contains eight entries, corresponding to whether the Present State Q(t) (Gate #3) is **0** or **1**; however, Figure 1.16 shows five cases, sufficient to demonstrate the logic of the RS flip-flop.

Based on Table 1.33, the K-map is constructed in Table 1.34. Then the K-map is used to formulate the equations for the flip-flop:

Table 1.33 RS Flip-Flop Truth Table

S (Gate #1)	R (Gate #2)	Present state Q(t) (Gate #3)	Next state Q(t + 1) (Gate #3)	Present state $\overline{Q}(t)$ (Gate #4)	Next state $\overline{Q}(t+1)$ (Gate #4)
0	0	0	0 (hold)	**1**	**1**
0	0	**1**	**1** (hold)	0	0
0	1 (reset)	0	0	**1**	**1**
0	1 (reset)	1	0	**0**	**1**
1 (set)	0	**0**	**1**	1	0
1 (set)	0	**1**	**1**	0	0
1(illegal)	1(illegal)	**0**	**1**	1	0
1(illegal)	1(illegal)	**1**	**1**	0	0

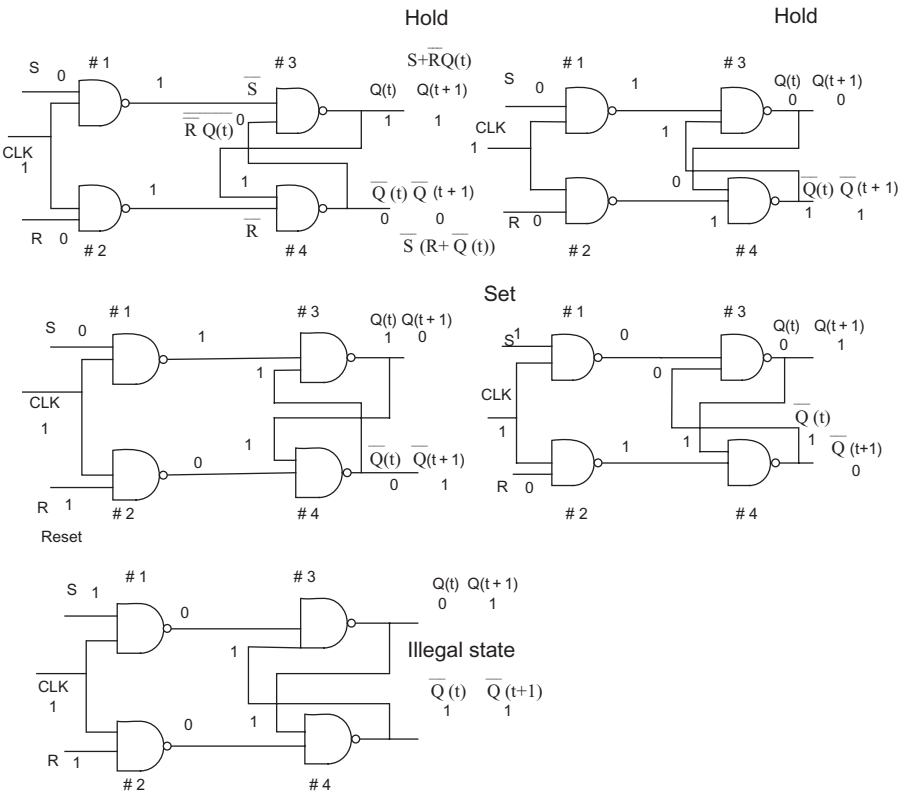


Figure 1.16 RS flip-flop.

Table 1.34a K-Map

S	R	Present State Q(t) (Gate #3)	
		0	1
0	0		1
0	1		
1	1	1	1
1	0	1	1

S $\bar{R}$ Q(t)

Table 1.34b K-Map

S	R	Present State $\bar{Q}$(t) (Gate #4)	
		0	1
0	0		1
0	1	1	1
1	1		
1	0		

$\bar{S}$ R $\bar{S}$ $\bar{Q}$(t)

Table 1.34a: (Gate #3): $Q(t+1) = S + \bar{R}\ Q(t)$

Table 1.34b: (Gate #4): $\overline{Q(t+1)} = \bar{S}(R + \bar{Q}(t))$

Problem: What are the illegal states of the RS flip-flop?

Answer: The states S = 1 (set) and R = 1 (reset) are not allowed in an RS flip-flop because set and reset cannot exist simultaneously (indeterminate state).

Delay (D) Flip-Flop

The D or delay flip-flop, shown in Figure 1.17, uses NAND gates. It is widely used in computers for transferring data. Several of these flip-flops can be used to design a CPU register, where each flip-flop is used to store 1 bit [RAF05]. This flip-flop delays the input appearing at the output by one CP. The D input goes directly into the S input and the complement of the D input goes to the R input. The D input is sampled during the occurrence of the CP. If D is 1, the flip-flop is switched to the set state (unless it was already set). If D is 0, the flip-flop switches to the clear state. If CP = 1, the output Q(t + 1) of the upper flip-flop is fed to the input of the lower flip-flop in Figure 1.17. On the other hand, if CP = 0, Q(t) of the upper flip-flop is fed to the input of the lower flip-flop.

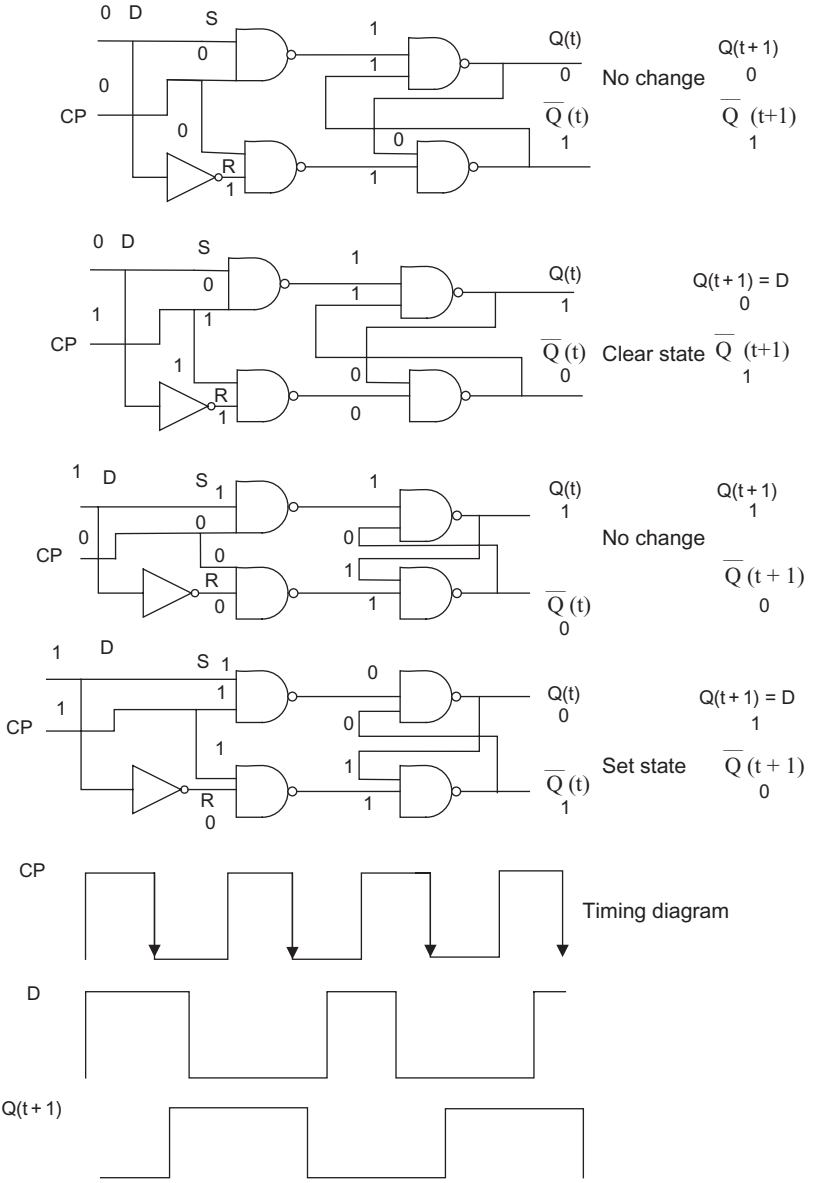


Figure 1.17 D flip-flop.

Problem: Given the above rules for the behavior of the D flip-flop, develop its truth table.

Solution: These relationships are embodied in Table 1.35.

A D flip-flop circuit can also be triggered by the negative-going edge of the CP, as opposed to being activated by pulse duration. The timing diagram for such a circuit

Table 1.35 D Flip-Flop Truth Table

D	CP	Present state Q(t)	Next state Q(t + 1) = D when CP = 1	Present state $\overline{Q}(t)$	Next state $\overline{Q}(t+1) = \overline{D}$ when CP = 1
0	0	0	0 (no change)	1	1 (no change)
0	1	1	0 (clear)	0	1
1	0	1	1 (no change)	0	0 (no change)
1	1	0	1 (set)	1	0

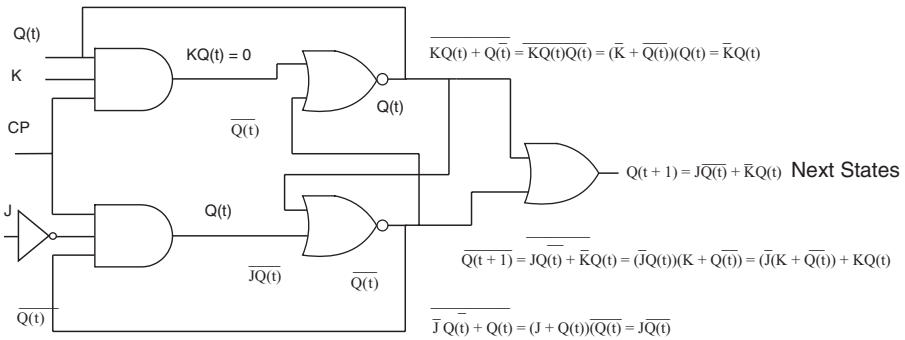


Figure 1.18 JK flip-flop circuit.

is shown in Figure 1.17. As the timing diagram shows, the D input is reflected in the Q(t + 1) (next state) output on the negative edge of the CP. Q(t + 1) follows the D input regardless of the present state Q(t), *if CP = 1*. If CP = 0, there is no change in the output. This property can be applied, for example, to transferring data from an input device (D) to microprocessor memory port Q(t + 1), according to the data transfer rules of Figure 1.17.

JK Flip-flop

A JK flip-flop is a refinement of the RS flip-flop by defining and allowing the illegal state of the RS flip-flop. In Figure 1.16, inputs J and K behave like inputs S and R to set and clear the flip-flop (note that in a JK flip-flop, the letter J is for set and the letter K is for clear). When logic 1 inputs are applied to both J and K simultaneously, the flip-flop switches to its complement state (e.g., if Q = 0, it switches to Q = 1 in Figure 1.18).

Note that because of the feedback connection in the JK flip-flop, a CP signal that remains a 1 (while J = K = 1) after the outputs have been complemented once will cause repeated and continuous transitions of the outputs. To avoid this, the CPs must have a time duration less than the propagation delay through the flip-flop.

Table 1.36 shows how the state of output Q at t + 1 changes as a function of the original state of Q(t) and the set input J and the clear input K. The K-map for

Table 1.36 JK Flip-Flop Truth Table

J	K	CP	Q(t) present state	Q(t + 1) next state	$\overline{Q}(t)$ present state	$\overline{Q}(t+1)$ next state
0	0	1	0	0	1	1
0	1 (clear)	1	0	0	1	1
1(set)	0	1	0	**1**	1	0
1	1	1	0	**1**	1	0
0	0	1	1	**1**	0	0
0	1(clear)	1	1	0	0	1
1(set)	0	1	1	**1**	0	0
1	1	1	1	0	0	1

Table 1.37 K-Map for JK Flip-Flop

J	K	Q(t) Present State	Q(t) Present State
		0	1
0	0		1
0	1		
1	1	1	
1	0	1	1

$J\overline{Q}(t)$ $\overline{K}Q(t)$

JK flip-flop in Table 1.37 is derived from the truth table in Table 1.36 by plugging **1**s in the map wherever there is a Q(t + 1) = 1 in the Table 1.36 (bolded). For example, when J = 0, K = 0, Q(t) = 1, and Q t + 1) = 1 in Table 1.36, a 1 is placed in the Q(t) = 1 column in Table 1.37.

Problem: Based on the K-map, what are the next state equations for Q(t + 1) and $\overline{Q}(t+1)$?

Answer: Referring to Table 1.37, the next state Q(t + 1) is governed by the following equation:

$$Q(t+1) = J\,\overline{Q}(t) + \overline{K}\,Q(t).$$

Using this equation for Q(t + 1), the equation for $\overline{Q}(t+1)$ can be computed as follows:

$$\overline{Q(t+1)} = \overline{J\,\overline{Q(t)} + K}\;Q(t) = (\overline{J} + Q(t))(K + \overline{Q(t)}) = (\overline{J}(K + \overline{Q(t)}) + K\,Q(t).$$

These equations are annotated on Figure 1.18.

Table 1.38 T Flip-Flop Truth Table

T	CP	Q(t)	$Q(t+1) = T\,\overline{Q(t)} + \overline{T}\,Q(t)$	Q(t)	$\overline{Q}(t+1) = T\,Q(t) + \overline{T}\,\overline{Q(t)}$
0	1	0	0 (no change)	1	1 (no change)
1	1	0	1 (toggle)	1	0 (toggle)
0	1	1	1 (no change)	0	0 (no change)
1	1	1	0 (toggle)	0	1(toggle)

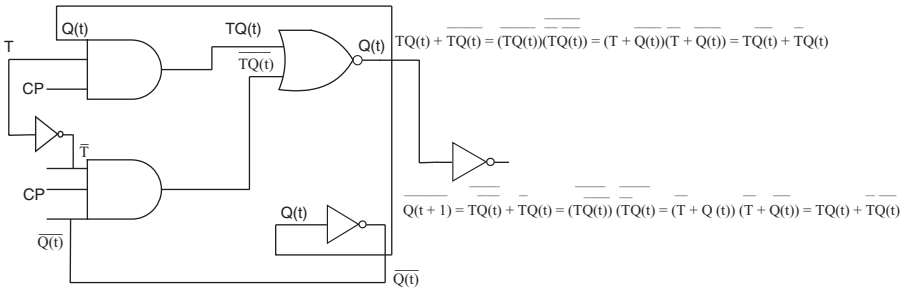


Figure 1.19 T flip-flop circuit diagram.

T Flip-Flop

The T flip-flop is a single input version of the JK flip-flop [RAF05]. It is typically used in the design of binary counters (covered later in the section "Design of Binary Counters," where complementation of the output is required. For example, in Table 1.38 when T = 1, the input Q(t) is toggled, producing its complement in output Q(t + 1). By examining the gate operations in Figure 1.19, at the Q output, we see that:

$$Q(t+1) = \overline{TQ(t) + \overline{T}\,\overline{Q(t)}} = (\overline{TQ(t)})(\overline{\overline{T}\,\overline{Q(t)}}) = (\overline{T} + \overline{Q(t)})(T + Q(t)) = T\,\overline{Q(t)} + \overline{T}\,Q(t).$$

Furthermore, the equation for $\overline{Q}(t+1)$ is derived as follows:

$$\begin{aligned}\overline{Q(t+1)} &= \overline{T\,\overline{Q(t)} + \overline{T}\,Q(t)} = \overline{(T\,\overline{Q(t)})}\,\overline{(\overline{T}\,Q(t))} \\ &= (\overline{T} + Q(t))(T + \overline{Q(t)}) = T\,Q(t) + \overline{T}\,\overline{Q(t)}.\end{aligned}$$

Note that in Figure 1.19 feedback from the flip-flop outputs to the inputs is used to obtain the desired outputs at time t + 1.

Problem: Based on the above equations, develop the T flip-flop truth table.

Solution: The truth table is shown in Table 1.38.

Triggering of Flip-Flops

There are situations where it is useful to have the output change only at the rising or falling edge of the CP, rather than *during* the CP. This stabilizes the circuit because

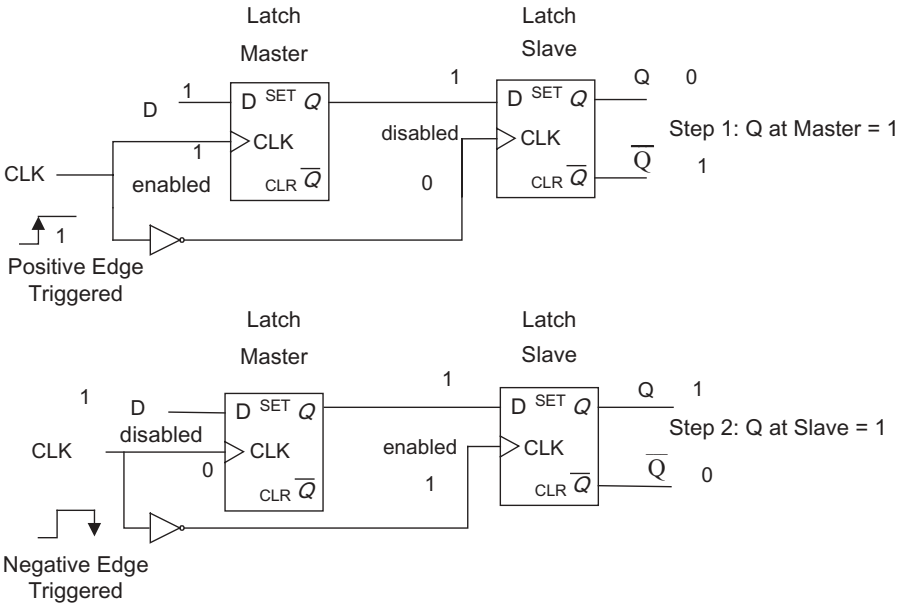


Figure 1.20 Edge-triggered flip-flop.

all changes are synchronized to the rising or falling edge of the CP. For example, when an input interrupt occurs, it should be held by the microprocessor until it can be serviced *during* the CP and only released on the *falling edge* of the CP. An edge-triggered flip-flop achieves this by combining a pair of latches in series. Figure 1.20 shows an edge-triggered D flip-flop where two D latches are connected in series, one directly, and one through an inverter. The first latch is called the master latch. When CLK is a 1 at Step 1, with a positive edge trigger, the master latch is enabled but the second latch, called the slave latch, is disabled with a negative edge trigger, so that a 1 is produced at the Q output of the master latch and a 0 is produced at the output of the slave latch. A 1 is produced at the master latch output because when CLK = 1, the Q output follows the D input. Contrariwise, when CLK is a 0 at Step 2, with a negative edge trigger, the master latch is disabled but the slave latch is enabled with a positive edge trigger (a negative edge is made positive with an inverter) so that a 1 is produced at the Q output of the slave latch by the Q output at the slave latch following the D input. In Step 2 it is assumed that Q still equals 1 in the master latch from Step 1. The Q output of the master latch does not change when CLK = 0, so that a 1 is transferred from the master latch to the slave latch.

Analysis of Asynchronous Sequential Circuits

As you have seen, edge-triggered flip-flops change state at the edge of a synchronizing CP. Many circuits require the initialization of flip-flops to a known state

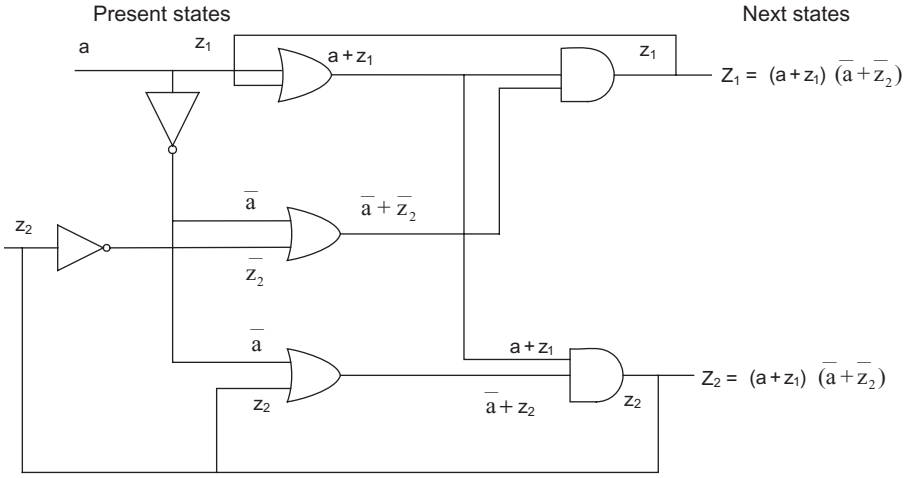


Figure 1.21 Analysis of asynchronous sequential circuit.

independent of the clock signal. Sequential circuits that change states whenever a change in input values occurs, independent of the clock, are referred to as *asynchronous sequential circuits*. Synchronous sequential circuits, latches, and flip-flops, on the other hand, change state only at the edge of the CP. For asynchronous sequential circuits, inputs are used to either set or clear the circuit *without* using the clock. Figure 1.21 is an example of an asynchronous sequential circuit. The next state equations for Z_1 and Z_2—as a function of present states a, z_1, and z_2—provide the logic for the outputs of the circuit in Figure 1.21. Feedback from outputs to inputs in Figure 1.21 produces the desired next states. The output equation

$$Z_1 = (a + z_1)(\bar{a} + \overline{z_2}) = a\,\bar{a} + \bar{a}\,z_1 + a\,\overline{z_2} + z_1\,\overline{z_2} = \bar{a}\,z_1 + a\,\overline{z_2} + z_1\,\overline{z_2}$$

can be reduced because the term $a\,\bar{a} = 0$, and the last term $z_1\,\overline{z_2}$ does not change the value of the equation, as demonstrated by the K-map in Table 1.40 that is used to minimize this equation, producing $Z_1 = \bar{a}\,z_1 + a\,\overline{z_2}$. Thus, the resultant terms $\bar{a}\,z_1$ and $a\,\overline{z_2}$ are identified in the K-map. The validity of this transformation is shown in the truth table for Z_1, Table 1.39. The K-map in Table 1.40 is produced by recording 1s in the map corresponding to **1**s (bolded) that appear for Z_1 in the truth table. This example demonstrates the fact that K-maps can accomplish Boolean expression reduction that is not possible with algebraic manipulation.

Problem: Reduce output equation Z_2 by developing the truth table and corresponding K-map.

Solution: The output equation $Z_2 = (a + z_1)(\bar{a} + z_2) = a\,\bar{a} + \bar{a}\,z_1 + a\,z_2 + z_1\,z_2 = \bar{a}\,z_1 + a\,z_2$ can be reduced, as shown above, because the first term $a\,\bar{a} = 0$ and the last term does not change the value of the equation, as demonstrated by the K-map in Table 1.41 that is used to minimize this equation, producing $Z_2 = \bar{a}\,z_1 + a\,z_2$,where it is shown that the term z_1z_2 is redundant. Thus, the resultant terms $\bar{a}\,z_1$ and $a\,z_2$ are identified in the K-map. The validity of this

Table 1.39 Truth Table for $Z_1 = (a + z_1)(\bar{a} + \overline{z_2})\bar{a}\, z_1 + a\, \overline{z_2}$

a	z_1	z_2	Decimal code = az_1z_2	$(a + z_1)$	$(\bar{a} + \overline{z_2})$	$Z_1 = (a + z_1)(\bar{a} + \overline{z_2})$	$\bar{a}\, z_1$	$a\, \overline{z_2}$	$Z_1 = \bar{a}\, z_1 + a\, \overline{z_2}$
0	0	0	0	0	1	0	0	0	0
0	0	1	1	0	1	0	0	0	0
0	1	0	**2**	1	1	**1**	1	0	**1**
0	1	1	**3**	1	1	**1**	1	0	**1**
1	0	0	**4**	1	1	**1**	0	1	**1**
1	0	1	5	1	0	0	0	0	0
1	1	0	**6**	1	1	**1**	0	1	**1**
1	1	1	7	1	0	0	0	0	0

Table 1.40 K-Map for $Z_1 = (a + z_1)(\bar{a} + \overline{z_2}) = \bar{a}\, z_1 + a\, \overline{z_2}$

z_1	z_2	a	
		0	1
0	0	0	**1**
0	1	0	0
1	1	**1**	0
1	0	**1**	**1**

$\bar{a}\, z_1$ $z_1\overline{z_2}$ $a\overline{z_2}$

redundant

Table 1.41 K-Map for $Z_2 = (a + z_1)(\bar{a} + z_2) = \bar{a}\, z_1 + a\, z_2$

z_1	z_2	a	
		0	1
0	0	0	0
0	1	0	**1**
1	1	**1**	**1**
1	0	**1**	1

$\bar{a}\, z_1$ z_1z_2 az_2

redundant

transformation is shown in the truth table for Z_2 (Table 1.42). The K-map is produced by recording 1s in the map corresponding to **1**s (bolded) that appear for Z_2 in the truth table.

The state transition table, depicting the state changes in transitioning from input variables a, z_1, and z_2 to output variables Z_1 and Z_2, is shown in Table 1.43. This

Table 1.42 Truth Table for $Z_2 = (a + z_1)(\bar{a} + z_2)$

a	z_1	z_2	Decimal code = az_1z_2	$(a + z_1)$	$(\bar{a} + z_2)$	$Z_2 = Z_2 = (a + z_1)(\bar{a} + z_2)$	$\bar{a}\, z_1$	az_2	$Z_2 = \bar{a}\, z_1 + a\, z_2$
0	0	0	0	0	1	0	0	0	0
0	0	1	1	0	1	0	0	0	0
0	1	0	**2**	1	1	**1**	1	0	**1**
0	1	1	**3**	1	1	**1**	1	0	**1**
1	0	0	4	1	0	0	0	0	0
1	0	1	**5**	1	1	**1**	0	1	**1**
1	1	0	6	1	0	0	0	0	0
1	1	1	7	1	1	**1**	0	1	**1**

Table 1.43 State Transition Table for Asynchronous Sequential Circuit

Present state		Next state			
		a = 0		a = 1	
z_1	z_2	$Z_1 = \bar{a}\, z_1 + a\, \overline{z_2}$	$Z_2 = \bar{a}\, z_1 + a\, z_2$	$Z_1 = \bar{a}\, z_1 + a\, \overline{z_2}$	$Z_2 = \bar{a}\, z_1 + a\, z_2$
0	0	0	0	1	0
0	1	0	0	0	1
1	0	1	1	1	0
1	1	1	1	0	1

table is constructed by noting the values of Z_1 corresponding to a = 0 and a = 1 and values of Z_2 corresponding to a = 0 and a = 1 in Tables 1.39 and 1.42, respectively, and recording the relationships in Table 1.43. Table 1.43 is used to indicate transitions from microprocessor state $Z_1 = 1$ to state $Z_2 = 1$ and vice versa. Consider the following application: when a = 1, $z_1 = 0$, and $z_2 = 0$ (decimal code 4), Z_1 is in the next state = 1 processing transactions. However, when a = 1, $z_1 = 0$, and $z_2 = 1$ (decimal code 5), the microprocessor transitions to the next state $Z_2 = 1$ to receive additional transaction input.

Another application of the asynchronous sequential circuit is the occurrence of asynchronous inputs to a microprocessor that arrive from the Internet, not on schedule (not governed by CP), but unscheduled (i.e., asynchronously). For example, let a, z_1, and z_2 be the binary bits of a decimal transaction code, arriving from the Internet, in a database application, where one type of transaction is processed by a microprocessor at its input Z_1 and the second type at its input Z_2. Suppose the allowable decimal codes at Z_1 are **2**, **3**, **4**, and **6** in Table 1.39 (bolded), and the allowable codes at Z_2 are **2**, **3**, **5**, and **7** in Table 1.42 (bolded). Then, Tables 1.39 and 1.42 provide the required transaction processing logic for Z_1 and Z_2, respectively.

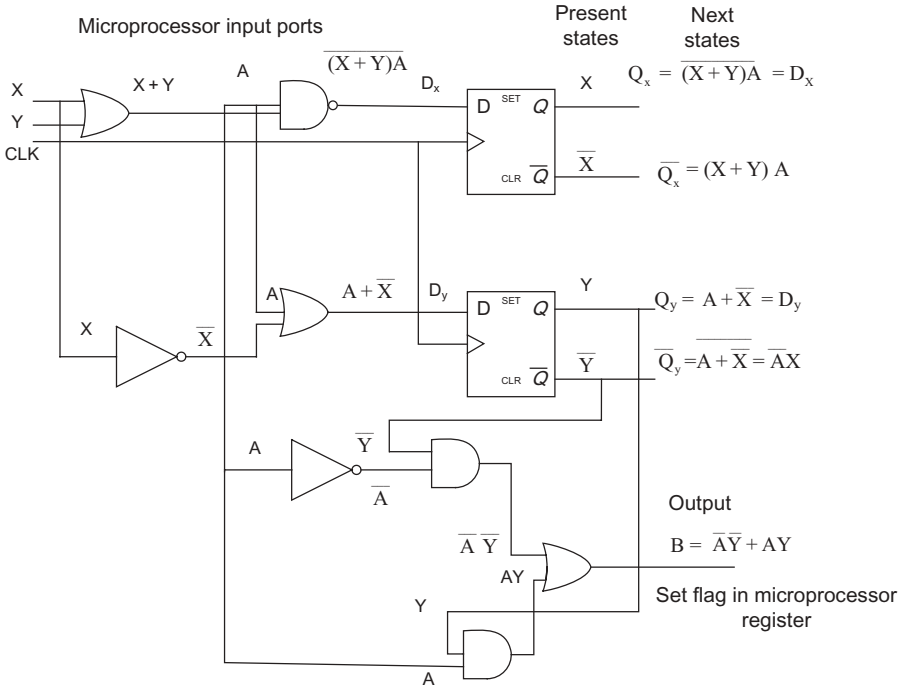


Figure 1.22 D flip-flops in asynchronous sequential circuit.

Relationship among Inputs, Flip-Flops, and Output States

Figure 1.22 shows an example of analyzing the inputs, D flip-flops, and output states of an asynchronous sequential circuit. The diagram shows the equations for the next states Q_x and Q_y, as a function of the present states D_x and D_y, recalling that for D flip-flops, output Q follows input D.

The equations below produce the values shown in the state transition table, Table 1.44, which shows the relationships among components.

$$Q_x = \overline{(X+Y)A} = D_x,$$
$$\overline{Q_x} = (X+Y)A,$$
$$Q_y = A + \overline{X} = D_y,$$
$$Q_y = \overline{A+\overline{X}} = \overline{A}X,$$
$$B = \overline{A}\,\overline{Y} + AY.$$

An application is the processing of transaction code bits occurring at microprocessor input ports X, Y, and A. An output B = 1 is produced by setting a flag B in a microprocessor register when correct transaction codes are received. For example, if decimal interrupt code 1, 3, 4, 6, or 7, corresponding to X, Y, A = 001, 011, 100,

Table 1.44 State Transition Table for the Analysis of Asynchronous Sequential Circuit

Inputs						Next state				Flip-flop inputs		Output
X	$\overline{X}$	Y	$\overline{Y}$	A	$\overline{A}$	Qx = Dx	$\overline{Q_x}$	Qy = Dy	$\overline{Q_y}$	$D_x = Q_x = (X + Y)A$	$D_y = Q_y = A + X$	$B = \overline{A}\overline{Y} + AY$
0	1	0	1	0	1	1	0	1	0	1	1	0
0	1	**0**	1	**1**	0	1	0	1	0	1	1	**1**
0	1	1	0	0	1	1	0	1	0	1	1	0
0	1	**1**	0	**1**	0	0	1	1	0	0	1	**1**
1	0	0	1	**0**	1	1	0	0	1	1	0	**1**
1	0	0	1	1	0	0	1	1	0	0	1	0
1	0	**1**	0	**0**	1	1	0	0	1	1	0	**1**
1	0	**1**	0	**1**	0	0	1	1	0	0	1	**1**

110, or 111 in Table 1.44, respectively, is received, the flag would be set. The microprocessor queries this flag to determine when to process transactions. The bolded terms in Table 1.44 indicate when the flag B is set.

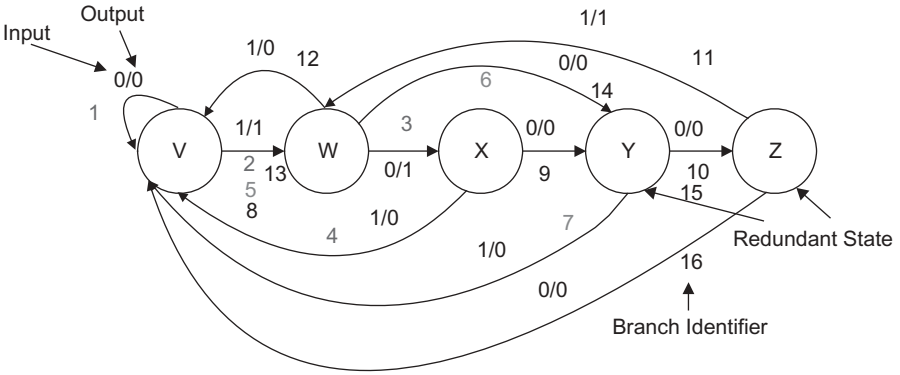
TYPES OF SYNCHRONOUS SEQUENTIAL CIRCUITS

Mealy and Moore Machines

In the Mealy machine, the output states depend on the inputs and the present states of the flip-flops [RAF05]. In the Moore machine, output states depend only on the present states of the flip-flops. For example, a Mealy machine would be used to control the execution sequence of a microprocessor that uses *both* data inputs and the current state of the program (i.e., program address) to decide which instruction to execute next (e.g., doing database management using input data from the Internet). On the other hand, the Moore machine would be used to control microprocessor program execution when *only* the current state of the program is relevant (e.g., doing a matrix multiplication on data stored in memory). Thus, the Mealy machine is the more versatile of the two.

Minimization of States

Figure 1.23 shows a state diagram for a synchronous sequential circuit, which is classified as a Mealy machine because outputs depend on *both* present states and inputs, where two of the paths are highlighted in red and green. It may be possible to minimize the number of states in these circuits by developing the state sequence diagram, based on Figure 1.23, to see whether there are any redundant states. If there are, the reduction in states is reflected in the revised state sequence table. Using Figure 1.23 and the original state sequence table, Table 1.45, state Z is identified as being redundant because the next state for both states V and Z is W, and the state changes have the same inputs and outputs (1, 1). Therefore, state Z is noted as



States: V,W,X,Y,Z
Path Sequences: (1,2,3,4) (5,6,7) (8,3,9,10,11,12) (13,14,15,16)
Input Sequence: (0 1 0 1) (1 0 1) (1 0 0 0 1 1) (1 0 0 0)
Output Sequence:(0 1 1 0) (1 0 0) (1 1 0 0 1 0) (1 0 0 0)

Figure 1.23 State diagram for minimization of states.

Table 1.45 Original State Sequence Table

Originating state	V	V	W	X	**Y**	W	V	W	**Z**	States Y and Z are redundant
Branch	1	2	3	4	**6**	12	10	5	**12**	
Input	0	1	0	1	**1**	1	1	0	**0**	
Next state	V	W	X	V	**V**	V	W	Y	**V**	
Output	0	1	1	0	**0**	0	1	0	**0**	

Table 1.46 Revised State Sequence Table (Eliminating Redundant States)

	Next state		Output	
Present state	Input = 0	Input = 1	Input = 0	Input = 1
V	V	W	0	1
W	X	V	1	0
X		V		0

redundant in Figure 1.23 and Table 1.45. Another state indicated as redundant in Figure 1.23 and Table 1.45 is Y because both Y and W have the next state V, with same state change inputs and outputs (1, 0). State Y is also noted as redundant in Figure 1.23 and Table 1.45. Therefore, states Z and Y do not appear in the revised state sequence table, Table 1.46.

Figure 1.24 shows the result of eliminating redundant states in the state diagram. It is important to note that it may not be possible to eliminate "redundant states"

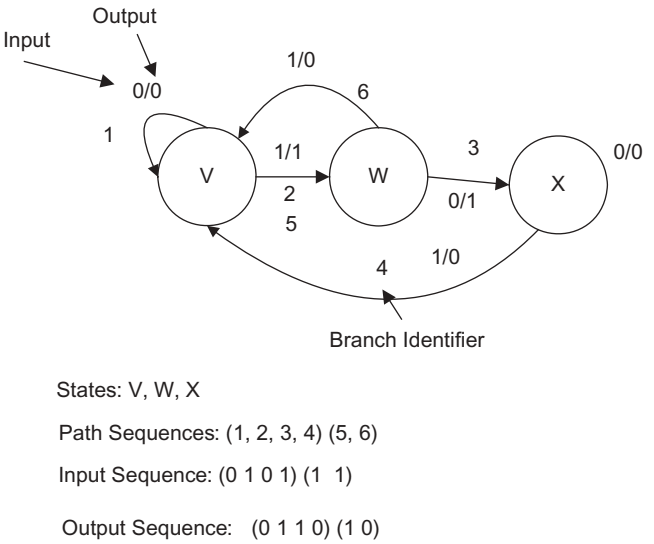


Figure 1.24 Reduced state diagram.

because these states could be associated with important functions. For example, redundant states could be associated with two microprocessors—one the primary, currently executing, and the other, a backup, redundant microprocessor, designed to take over if the primary fails. However, in general, digital circuitry can be simplified by eliminating redundant states.

Design of Synchronous Sequential Circuits

To design synchronous sequential circuits, or any circuit for that matter, start with your objective. For example, suppose you want a microprocessor to produce an output Z dependent on input A (e.g., input data A has arrived from the Internet, and the microprocessor produces output Z); the present state of your computer program is represented by X (e.g., ready to read input data A) and the present state of the input buffer A is represented by Y (e.g., input buffer A empty). You need to identify the transition to the next computer program state, X+, (e.g., fill buffer with input A data) and Y+ (e.g., input A buffer full). Thus, referring to the state diagram in Figure 1.25, if an input occurs on microprocessor line A = 1, and the present program state are X = 1, Y = 1, representing instruction ready to execute and input buffer A empty, respectively, output is produced on microprocessor line Z = 1, and the program transitions to next state X+ = 0 (fill buffer) and Y+ = 0, (input buffer A full). The state diagram in Figure 1.25 is an example of a Mealy machine circuit specification because outputs depend on both inputs and states of the circuit.

To design your circuit, identify the states, inputs that cause state transitions, and outputs produced by inputs and state transitions, as in the above example. Then, note

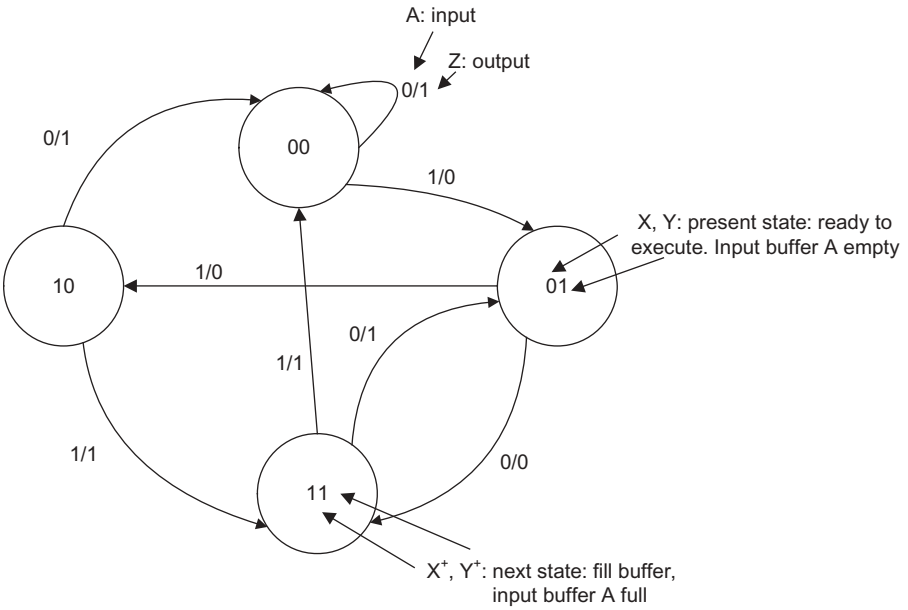


Figure 1.25 State diagram for design of sequential circuit.

Table 1.47 State Table for Sequential Circuit

Present states		Input	Next states		Flip-flop inputs		Output
X	Y	A	X^+	Y^+	$D_x = X^+ = \overline{X}Y + X\overline{Y}A$	$D_y = Y^+ = \overline{Y}A + Y\overline{A}$	$Z = \overline{Y}\,\overline{A} + X$
0	0	0	0	0	0	0	**1**
0	0	1	0	1	0	**1**	0
0	1	0	1	1	**1**	**1**	0
0	1	1	1	0	**1**	0	0
1	0	0	0	0	0	0	**1**
1	0	1	1	1	**1**	**1**	**1**
1	1	0	0	1	0	**1**	**1**
1	1	1	0	0	0	0	**1**

in Figure 1.25 and in Table 1.47 the present states X and Y and input $\underline{A}$ that generate next state output $X^+ = 1$. For example X = 0, Y = 1, and A = 1 (or $\overline{X}YA$) produce $X^+ = 1$. Next, for example, use the D flip-flop, noting that the output corresponding to next state X^+ is designated as D_x and its formulation is the following:

$$D_x = \overline{X}YA + \overline{X}Y\overline{A} + X\overline{Y}A = \overline{X}Y(A + \overline{A}) = \overline{X}Y + X\overline{Y}A.$$

Similarly, produce the next state Y^+ formulation in terms of a D flip-flop output, as follows:

$$D_y = \overline{X}\,\overline{Y}A + X\overline{Y}A + \overline{X}Y\overline{A} + XY\overline{A} = \overline{Y}A(\overline{X} + X) + Y\overline{A}(\overline{X} + X) = (\overline{Y}A + Y\overline{A}).$$

Also, develop the equation for output Z by noting in Figure 1.25 the present states X and Y and input A that generate Z = 1 output, producing the following equation:

$$Z = \overline{X}\,\overline{Y}\,\overline{A} + X\overline{Y}\,\overline{A} + XYA + XY\overline{A} = \overline{Y}\,\overline{A}(\overline{X} + X) + XY(A + \overline{A}) = (\overline{Y}\,\overline{A} + XY).$$

Then, using these equations, develop the state table in Table 1.47. Next, formulate the K-maps in Tables 1.48–1.50. Note that to construct the K-maps, 1s are placed in the cells of the maps wherever 1s appear for D_x, D_y, and Z in the state table. Recall that for D flip-flops, inputs are equal to the next states of the circuit. Last, based on the flip-flop and output equations, design the circuit in Figure 1.26.

Message Processing Design

Synchronous sequential circuits are highly adaptable to message processing systems, as shown in Figure 1.27. As shown in the figure, a message processing system

Table 1.48 K-Map for $D_x = \overline{X}YA + \overline{X}Y\overline{A} + X\overline{Y}A = \overline{X}Y(A + \overline{A}) = \overline{X}Y + X\overline{Y}A$

	YA			
X	00	01	11	10
0			1	1
1		1		

$X\overline{Y}A$ $\overline{X}Y$

Table 1.49 K-Map for $D_y = \overline{X}\,\overline{Y}A + X\overline{Y}A + \overline{X}Y\overline{A} + XY\overline{A} = \overline{Y}A(\overline{X} + X) + Y\overline{A}(\overline{X} + X) = (\overline{Y}A + Y\overline{A})$

	YA			
X	00	01	11	10
0		1		1
1		1		1

$\overline{Y}A$ $Y\overline{A}$

Table 1.50 K-Map for $Z = \overline{X}\,\overline{Y}\,\overline{A} + X\overline{Y}\,\overline{A} + XYA + XY\overline{A} = \overline{Y}A(X + X) + XY(A + \overline{A}) = (\overline{Y}A + XY)$

	YA			
X	00	01	11	10
0	1			
1	1		1	1

$\overline{Y}\,\overline{A}$ XY

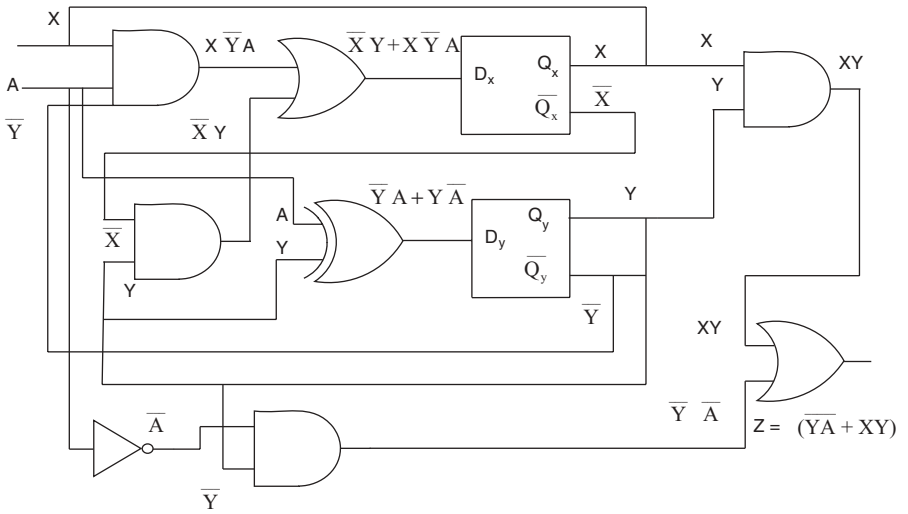
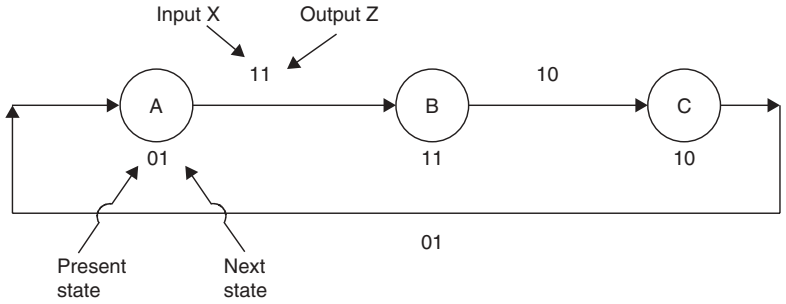


Figure 1.26 Logic diagram for synchronous digital circuit.



Detects message output sequence 101

State transitions

State A detects input X = 1 and outputs Z = 1 to state B
State B receives 1 from A and outputs Z = 0 to state C
State C receives 0 from B and outputs Z = 1 to state A

Figure 1.27 Message processing state diagram.

involves a sequence of inputs X with the objective of the circuit detecting a bit pattern, such as 101. The circuit accomplishes this objective by changing state according to the bit pattern received. When the desired bit pattern is recognized, the sequence 101 is generated at the output. An application is the detection of computer program operation codes by a microprocessor. For example, if the operation code for the add instruction is the decimal 5 (binary 101), the output 101 would be generated in Figure 1.27 designating that the add instruction should be executed.

The first step in the design process is to specify the state transitions, as shown in Figure 1.27, where the desired detected bit pattern is shown. State transitions are identified that will serial process the incoming bit stream, looking for the desired pattern in Figure 1.27. Additional steps involve designing the state transition table in Table 1.51 to represent the logic of Figure 1.27 in a tabular form and selecting a flip-flop type to implement state transitions. In this case, the T flip-flop is selected because its output toggles with each CP. If T = 1, the flip-flop causes complementation of the present state. This is the logic required to detect the input sequence 101 in Figure 1.28.

Table 1.51 State Transition Table

		Input X = 1	Input X = 1	Input X = 1
Present state	Present T flip-flop binary state Q	Next T flip-flop state Q	Next T flip-flop binary state Q	Output Z
A	0	B	1	1
B	0	C	1	0
C	0	A	1	1

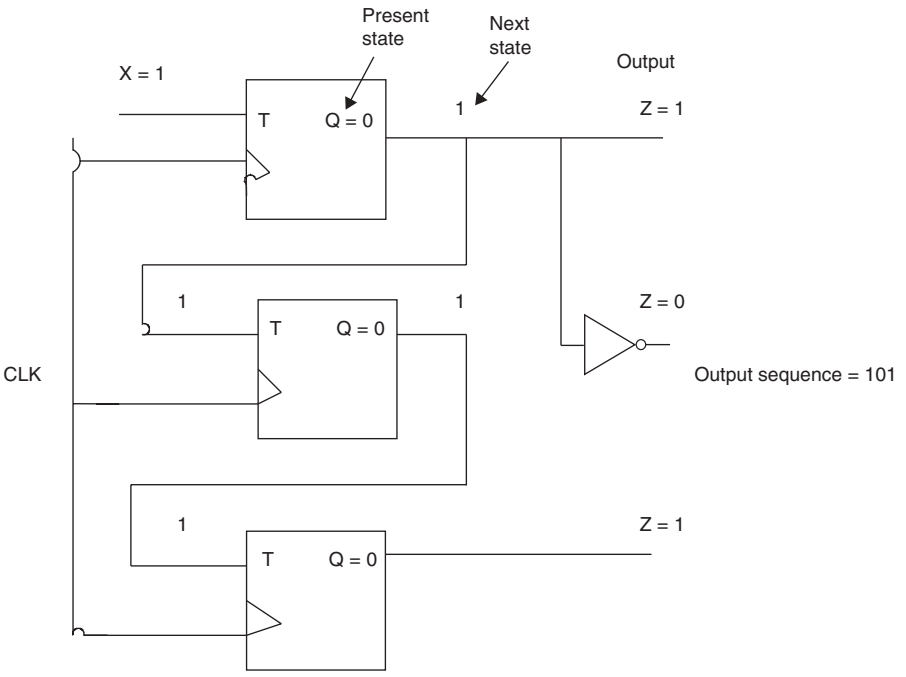


Figure 1.28 Message processing circuit.

Design of Binary Counters

Two-Bit Counter

The binary counter is an example of a synchronous sequential circuit designed to count a sequence of binary digits. For example, if the counter can count two binary digits at a time, it would be able to process the following sequence of digits: 00, 01, 10, and 11. Thus, the counter can count 2^n binary numbers, using flip-flops (e.g., T flip-flops), where n is the number of binary bits in the count. Figure 1.29 shows the state transition diagram for a 2-bit binary counter that implements the binary sequence count rules (e.g., if the sequence is 00, it is recognized by the next state 01). After Figure 1.29 has been constructed, the state table (Table 1.52) for flip-flops 1 and 2 is developed followed by the state table (Table 1.53) for flip-flops 3 and 4. The outputs b_0 and b_1 follow the logic rule: $T\overline{Q(t)} + \overline{T}Q(t)$ in Figure 1.29. Note

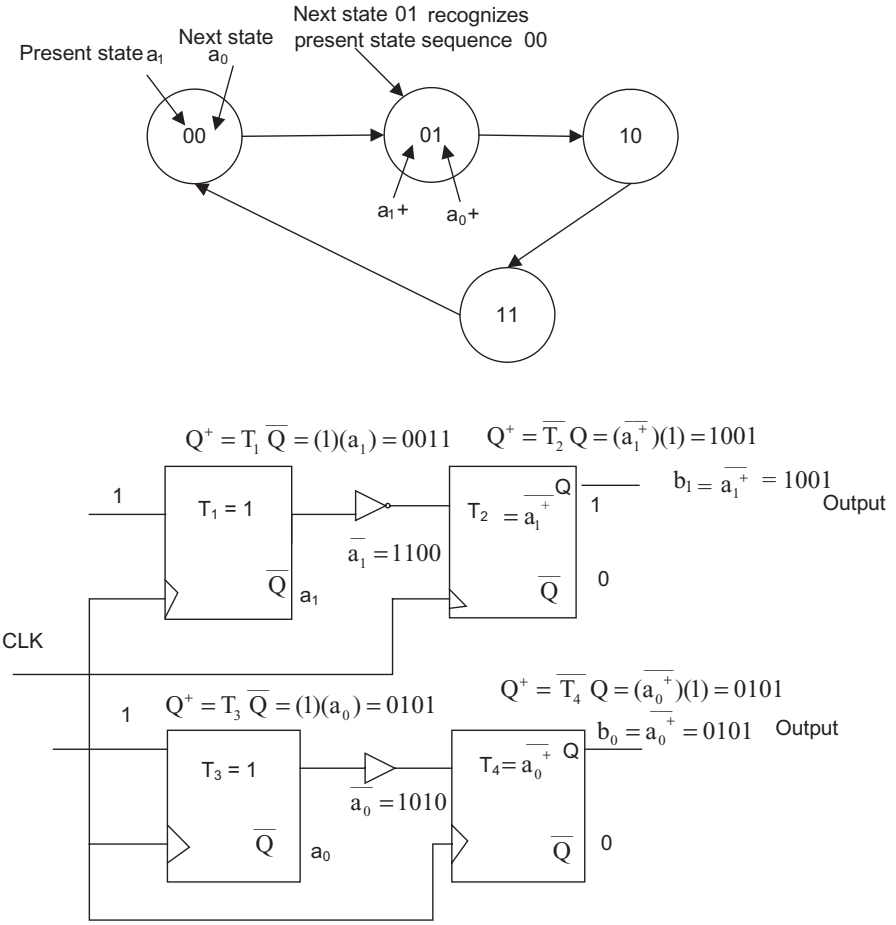


Figure 1.29 Binary counter state transition diagram and circuit.

Table 1.52 Binary Sequence Counter State Table

Present state		Next state flip-flop 1	Next state flip-flop 2	Output
a_1	a_1^+	$Q^+ = T_1\overline{Q} = (1)(a_1)$	$Q^+ = \overline{T_2}Q = (\overline{a_1^+})(1)$	$b_1 = \overline{a_1^+}$
0	0	0	1	1
0	1	0	0	0
1	1	1	0	0
1	0	1	1	1

Table 1.53 Binary Sequence Counter State Table

Present state		Next state flip-flop 3	Next state flip-flop 4	Output
a_0	a_0^+	$Q^+ = T_3\overline{Q} = (1)(a_0)$	$Q^+ = \overline{T_4}Q = (\overline{a_0^+})(1)$	$b_0 = \overline{a_0^+}$
0	0	0	0	0
0	1	0	1	1
1	1	1	0	0
1	0	1	1	1

that an inverter is inserted between the flip-flops in Figure 1.29 in order to achieve the correct state transitions.

Three-Bit Counter

A 3-bit counter design proceeds by first constructing the state diagram in Figure 1.30, with present and next states annotated. Next, using JK flip-flops, show the 3-bit counter excitation table (Table 1.54), noting flip-flop states and flip-flop inputs. The salient state conditions can be summarized as follows: when Q = 0 and J = 0, no change in state; when J = 1, set the flip-flop; when K = 1, clear the flip-flop; and when Q = 1 and K = 0, no state change. The reader may wonder how the present states are obtained in Figure 1.30. The answer is that present states correspond to the present states of the flip-flops that, in turn, correspond to the condition where there is no CP (e.g., $a_2a_1a_0 = 000$).

To demonstrate the validity of the JK flip-flop transformations in Figure 1.30, recall the fundamental property of the JK flip-flop: $Q^+(\text{next state}) = J\,\overline{Q(t)} + \overline{K}\,Q(t)$. For example, in the state transition $a_2a_1a_0 = 000$ $a_2^+a_1^+a_0^+ = 001$, applying Q^+(next state) yields:

$$a_2^+ = J_2\overline{Q_2(t)} + \overline{K_2}Q_2(t) = a_1a_0\overline{a_2} + \overline{a_1a_0}a_2.$$

Thus,

$$a_2^+ = a_1a_0\overline{a_2} + \overline{a_1a_0}a_2 = 001 + \overline{000} = 0,$$
$$a_1^+ = J_1\overline{Q_1(t)} + \overline{K_1}Q_1(t) = a_0\overline{a_1} + \overline{a_0}a_1.$$

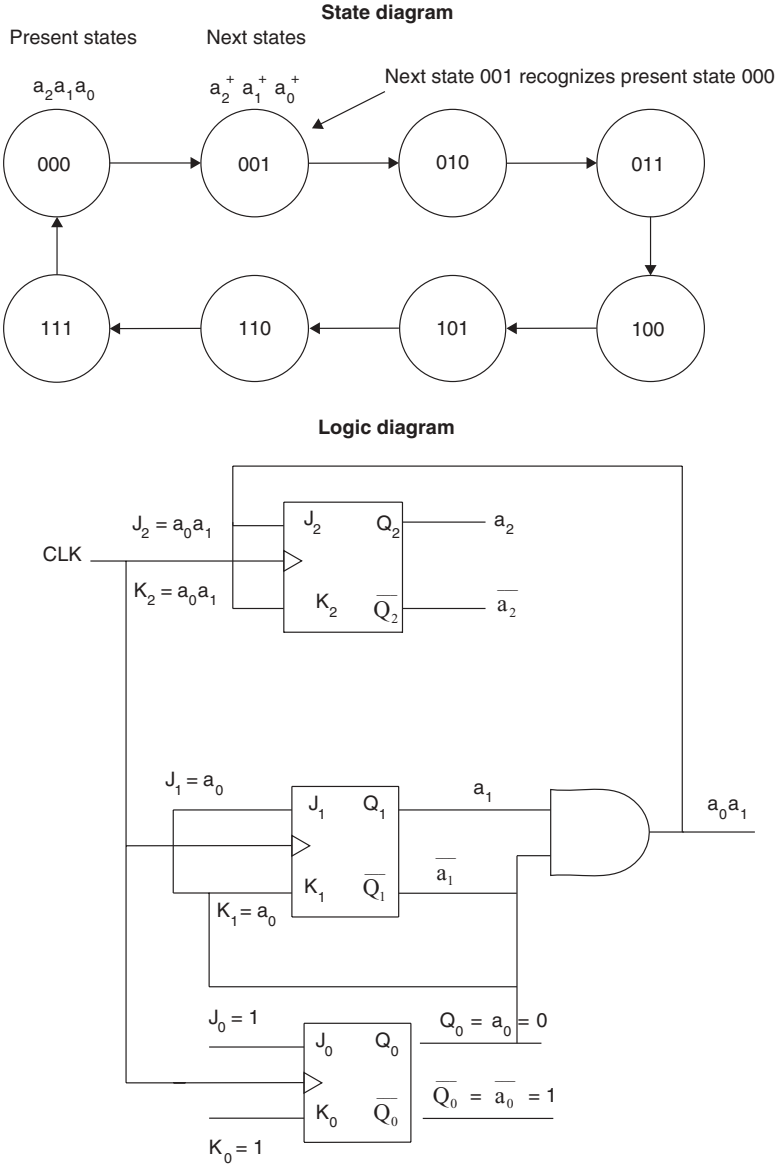


Figure 1.30 Three-bit counter state diagram and logic diagram.

Thus,

$$a_1^+ = a_0\overline{a_1} + \overline{a_0}a_1 = 01 + 10 = 0,$$
$$a_0^+ = J_0\overline{Q_0(t)} + \overline{K_0}Q_0(t) = 1\overline{a_0} + 0a_0 = 11 + 00 = 1.$$

Thus, the state transition $a_2a_1a_0 = 000$ $a_2^+a_1^+a_0^+ = 001$ is demonstrated.

Next, using Figure 1.30, formulate the truth table (Table 1.54), incorporating the state transitions from Figure 1.30 and the flip-flop inputs that generate these transitions. Next, the K-maps in Tables 1.55–1.60, by noting the flip-flop inputs that are bolded in Table 1.54, and resultant equations, are developed for the flip-flop inputs.

Table 1.54 Three-Bit Counter Truth Table

Present State			Next state			Flip-flop inputs					
a_2	a_1	a_0	a_2^+	a_1^+	a_0^+	$J_2 = a_1a_0$	$K_2 = a1_1a_0$	$J_1 = a_0$	$K_1 = a_0$	$J_0 = 1$	$K_0 = 1$
0	0	0	0	0	1	0	0	0	0	1	1
0	**0**	**1**	0	1	0	0	0	**1**	**1**	1	1
0	1	0	0	1	1	0	0	0	0	1	1
0	**1**	**1**	1	0	0	**1**	**1**	**1**	**1**	1	1
1	0	0	1	0	1	0	0	0	0	1	1
1	**0**	**1**	1	1	0	0	0	**1**	**1**	1	1
1	1	0	1	1	1	0	0	0	0	1	1
1	**1**	**1**	0	0	0	**1**	**1**	**1**	**1**	1	1

Table 1.55 K-Map for J_2

		a_1a_0			
		00	01	11	10
a_2	0			1	
	1			1	

$J_2 = a_1a_0$

Table 1.56 K-Map for K_2

		a_1a_0			
		00	01	11	10
a_2	0			1	
	1			1	

$K_2 = a_1a_0$

Table 1.57 K-Map for J_1

		a_1a_0			
		00	01	11	10
a_2	0		1	1	
	1		1	1	

$J_1 = a_0$

Table 1.58 K-Map for K_1

		a_1a_0			
		00	01	11	10
a_2	0	X	1	1	
	1	X	1	1	

$K_1 = a_0$

Table 1.59 K-Map for J_0

		a_1a_0			
		00	01	11	10
a_2	0	1	1	1	1
	1	1	1	1	1

$J_0 = 1$

Table 1.60 K-Map for K_0

		a_1a_0			
		00	01	11	10
a_2	0	1	1	1	1
	1	1	1	1	1

$K_0 = 1$

Shift Register Design

The design process starts by documenting the elements of the basic building block of the shift register—called the basic cell in Figure 1.31—comprised of the multiplexer and the D flip-flop. The D flip-flop is used because the flip-flop Q output follows the multiplexer basic cell D input, thus enabling the shift operation. The

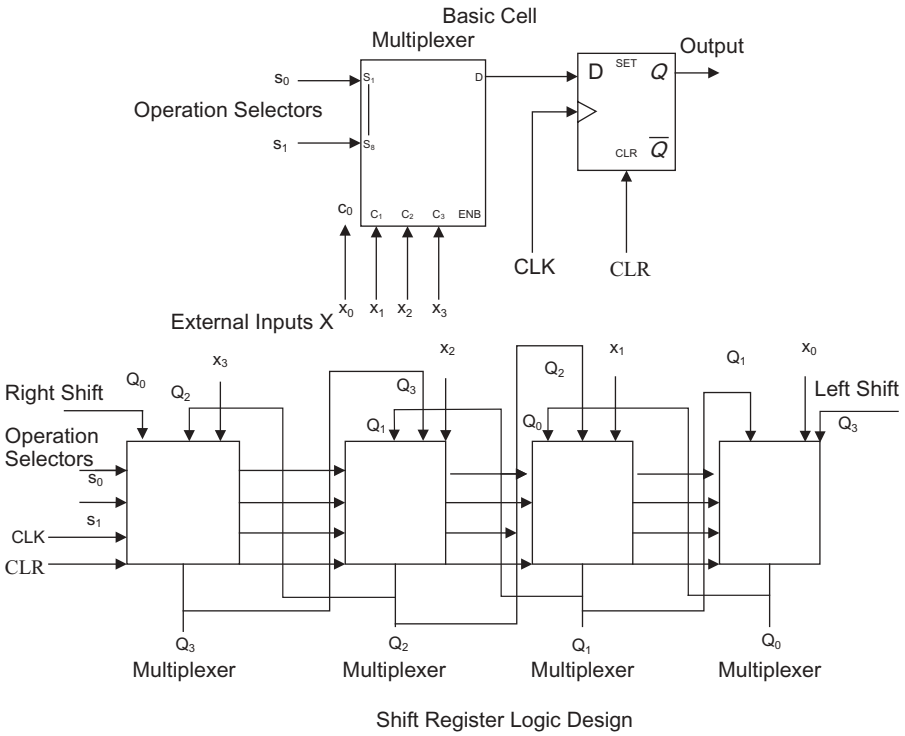


Figure 1.31 Basic cell and logic design of shift register.

basic cell is replicated in the shift register logic design, also shown in Figure 1.31. The shift register operates in Figure 1.31 by shifting the least significant bit, x_0, for a left shift, one flip-flop output to the left on each CP. For a right shift, the most significant bit, x_3, is shifted one flip-flop output to the right on each CP. These shifts are referred to as "end around" because for a right shift, the least significant bit, represented by Q_3 in Table 1.61, is shifted to the most significant bit position. Moreover, in a left shift, the most significant bit, represented by Q_0 in Table 1.61, is shifted to the least significant bit position. The type of shift is based on the values of the operation selectors in Table 1.61.

RAM DESIGN

There are two types of RAM: static and dynamic. Static RAM stores data in flip-flops. Dynamic RAM stores data in capacitors. Because capacitors gradually lose their charge, dynamic RAM must be refreshed periodically. A RAM circuit is shown in Figure 1.32 where 1 bit, 0 or 1, can either be read or written depending on whether a read or write operation is selected and whether a 1 or 0 appears at the input.

Table 1.61 Truth Table for Shift Register

Operation selectors		Clock input CLK	Clear input CLR	Operation	Input	Output
s_0	s_1					
0	0		1	Clear	$Q_0\ Q_1\ Q_2\ Q_3$	0000
0	1		0	No operation	$Q_0\ Q_1\ Q_2\ Q_3$	$Q_0\ Q_1\ Q_2\ Q_3$
1	0	⎍	0	Shift right "end around"	$Q_0\ Q_1\ Q_2\ Q_3$	$Q_3\ Q_0\ Q_1\ Q_2$
1	1	⎍	0	Shift left "end around"	$Q_0\ Q_1\ Q_2\ Q_3$	$Q_1\ Q_2\ Q_3\ Q_0$

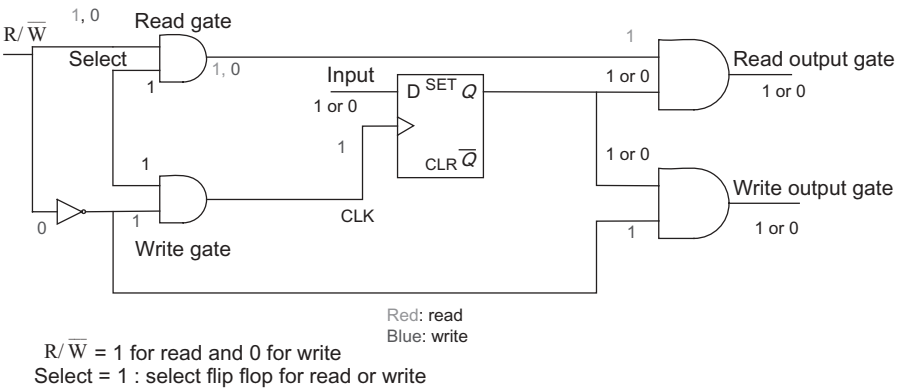


Figure 1.32 Random access memory (RAM) circuit.

HARDWARE DESCRIPTION LANGUAGE (HDL)

Given the complexity of some digital circuits, implementing them can be error prone. Therefore, as a design aid, aimed to increase design productivity and reduce errors, HDLs have been developed. In electronics, an HDL is any language from a class of computer languages for formal description of electronic circuits, and more specifically, digital logic. It can describe the circuit's operation, its design and organization, and tests to verify its operation by means of simulation.

Using the proper subset of virtually any HDL, a software program called a synthesizer can infer hardware logic operations from the language statements and produce equivalent hardware functions to implement the specified logic. Synthesizers use clock edges as the way to time a circuit.

HDLs are text-based expressions of the logical and timing characteristics of electronic systems. Like concurrent programming languages, HDL syntax and semantics includes notations for expressing concurrency. Languages whose only purpose is to express circuit connectivity between blocks are classified as computer-aided design languages.

The *automated* steps in using an HDL are the following:

Develop the logic diagram, using truth tables.

Generate the logic equations corresponding to the truth table relationships.

Minimize the logic equations, if necessary, using K-maps.

Use the simulator component of the HDL to verify the correct operation of the circuit logic, in particular, test timing constraints.

More details on HDL can be found in Salcic and Smailagic [SAL08].

SUMMARY

This chapter has provided the reader with numerous microprocessor design fundamentals and practical examples that lay the groundwork for the practicing engineer or student to design a complete microprocessor. In addition to elucidating principles, the chapter explained why circuits operate the way they do. Furthermore, there was a focus on design process to provide the reader with a road map to successful design. Last, many examples of digital logic were drawn from everyday experience to show the reader that the application of digital logic is not limited to designing microprocessors.

REFERENCES

[GIB80] G. A. Gibson and Y. Liu, *Microcomputers for Engineers and Scientists*. Englewood Cliffs, NJ: Prentice-Hall, Inc., 1980.

[GRE80] S. E. Greenfield, *The Architecture of Microcomputers*. Cambridge, MA: Winthrop Publishers, Inc., 1980.

[HAR07] D. M. Harris and S. L. Harris, *Digital Design and Computer Architecture*. New York: Elsevier, 2007.

[RAF05] M. Rafiquzzaman, *Fundamentals of Digital Logic and Microcomputer Design*. New York: Wiley-Interscience, 2005.

[SAL08] Z. Salcic and A. Smailagic, *Digital Systems Design and Prototyping: Using Field Programmable Logic and Hardware Description Languages*. Boston: Kluwer Academic Publishers, 2000.

Chapter 2

Case Study in Computer Design

The objective of this chapter is to provide the reader with a case study illustrating design principles, design decisions, and the analysis of performance and reliability that are products of the design process, using the computer-based design of an elevator system as an example. This chapter builds upon the fundamentals of Chapter 1: Digital Logic and Microprocessor Design.

DESIGN PRINCIPLES

According to Harris and Harris [HAR07], the following design principles should be used to develop an effective design:

Simplicity favors regularity, meaning that functions exhibiting regularity should be implemented in simple hardware designs, as opposed to complex software designs that would be the choice for functions exhibiting nonregularity. For example, elevator push button controls would be implemented in hardware, whereas the algorithm for determining direction of travel would be implemented in software.

Make the common case fast, meaning that frequently executed functions should be implemented in a design that provides for fast execution. For example, the elevator door open and close function could be implemented in a fast microprocessor.

Smaller is faster, meaning that small components, such as cache memory, are smaller and faster than large and slow main memory. Thus, for example, the data about frequently requested floors in an elevator system (e.g., lobby floor) would be stored in the cache, as opposed to basement floor data that would be stored in main memory.

Computer, Network, Software, and Hardware Engineering with Applications, First Edition. Norman F. Schneidewind.
© 2012 the Institute of Electrical and Electronics Engineers, Inc. Published 2012 by John Wiley & Sons, Inc.

DESIGN DECISIONS

There are a number of decisions that must be made as part of the design process that we explain as follows:

Control. If the elements (see Table 2.1) of a system can operate independently, such as a Web system implemented on the Internet, control should be distributed. On the other hand, if there must be surveillance of the elements in order to coordinate operations, such as users generating floor requests in an elevator system, control should be centralized.

Storage. For systems with large storage demands and modest data access time specifications, such as a database management system, secondary storage is a requirement. However, for embedded systems in which the data volume generated is small but access time must be short to meet user demands, fast cache memory is the primary storage requirement. An example is an elevator system that must store simultaneous user floor requests and have quick access to those requests in a cache memory.

Communication. The major contenders for the system communication vehicle are the bus and point-to-point connectivity. This is a very interesting design decision because it is not obvious that one alternative would be better than the other for a given application. The point-to-point alternative provides dedicated communication but becomes unwieldy if applied to many components because the hardware connectivity becomes complex and costly. However, if high speed communication is essential, point-to-point communication would be used because competing for *bus* bandwidth would be infeasible for meeting high speed communication requirements, such as among the elevator system devices shown in Figure 2.1. Bus communication is attractive for applications that have modest speed requirements, but where there is a multiplicity of devices that must communicate frequently, such as an Ethernet local network.

Topology. Communication and topology are intimately related because communication paths are the elements of a topology. For example, in point-to-point connectivity, nodes (e.g., components) are directly connected by links (e.g., communication cables), whereas in a bus system, all nodes are connected to a single link. In addition, topology is related to component and device count, which, in turn, are related to system hardware cost. Thus, by defining topology, designers can estimate hardware cost.

IDENTIFICATION OF SYSTEM ELEMENTS

Using an elevator example, Tables 2.1 and 2.2 provide a manifest of the linkage of elements to the computer design process, where elements are the objects comprising a system. That is, the attributes of the elements are characterized in order to visualize how elements are related. These attributes will be used in various facets of the design

Table 2.1 Identification of System Elements

Element	Purpose	Source(s)	Destination	Response time	Transfer rate	Format and storage requirement	Computation
Sequence j	Identify order and direction of elevator traversal	Floor request i N_i, current floor location N_c	Destination floor N_d	Sequence response time T_j	See Table 2.2	2 bits (four sequences)	Order and direction of elevator traversal
Floor request i N_i	Request service	Push button	Elevator controller	Sequence response time T_j	See Table 2.2	3 BCD integer digits (100 floors)	Identify sequence j and traversal direction
Current floor location N_c	Identify elevator location	Floor detector	Elevator controller		See Table 2.2	3 BCD integer digits (100 floors)	Identify sequence j and traversal direction
Destination floor N_d	Identify destination floor	Push button	Elevator controller		See Table 2.2	3 BCD integer digits (100 floors)	Identify sequence j and traversal direction
Probability of traversing sequence j P_j	"Chance" of traversing sequence j	Elevator controller microprocessor				4 BCD floating-point digits	Uses N_i, N_c, N_d
Sequence j response time T_j	Quantifies user expectations	Elevator controller microprocessor				6 BCD floating-point digits	Uses N_i, N_c, N_d, P_j, t_f, t_{oc}

(*Continued*)

Table 2.1 (*Continued*)

Element	Purpose	Source(s)	Destination	Response time	Transfer rate	Format and storage requirement	Computation
Single floor traversal time t_f	Parameter in computation of T_j	Elevator controller microprocessor				1 BCD integer digit	Specified as 5 seconds and 1 second
Number of failures n_f	Used in computation of failure rate	Elevator controller microprocessor				1 BCD integer digit	Specified as 1, 2, 3, 4, and 5 failures
Floor travel distance for test i n_i	Used in computation of failure rate	Elevator controller microprocessor				6 BCD floating-point digits	Uses N_i, N_c, N_d
Door open and close time t_{oc}	Parameter in computation of T_j	Elevator controller microprocessor				1 BCD floating-point digit	Specified as1 second
Sequence j failure rate λ_j	Variable in prediction of R_j	Elevator controller microprocessor				6 BCD floating-point digits	Uses n_f, n_i, T_j
Sequence j reliability R_j	Predicts probability of zero failures	Elevator controller microprocessor				4 BCD floating-point digits	Uses T_j, λ_j, P_j

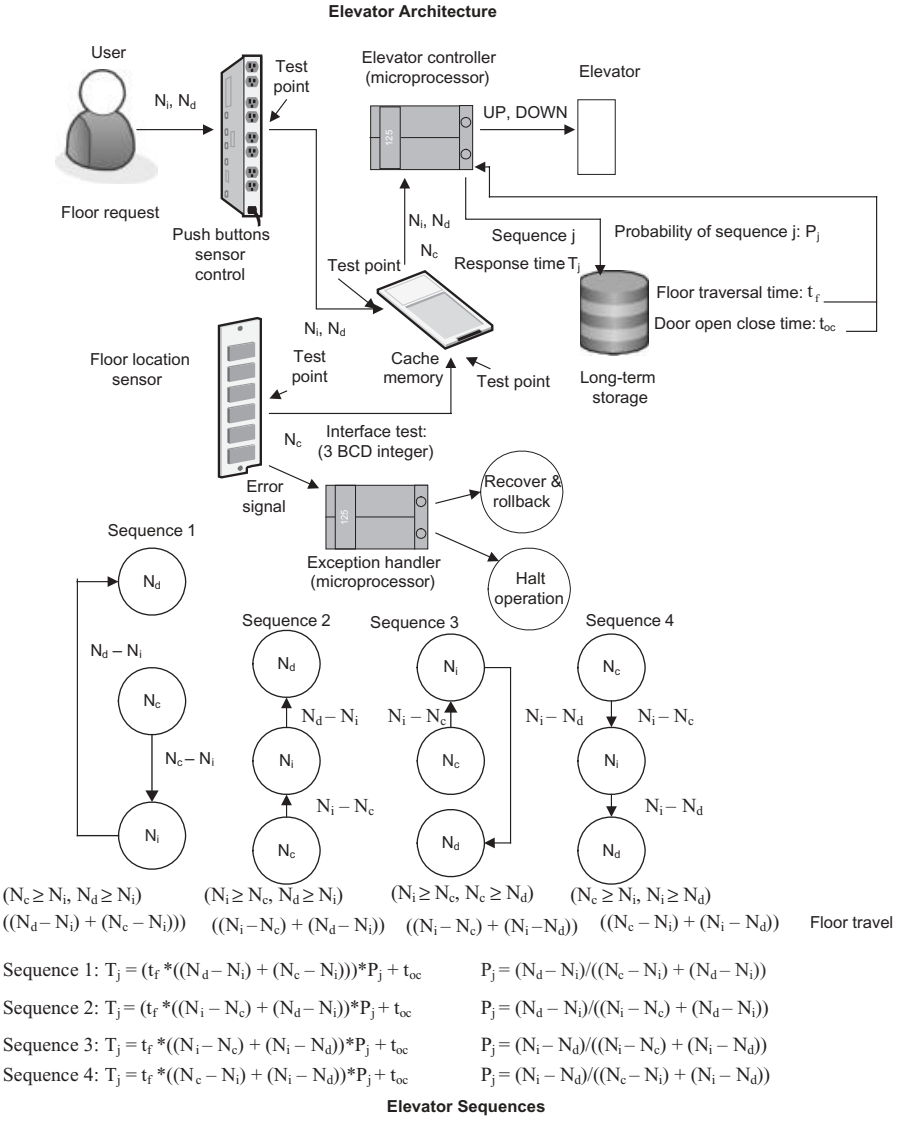


Figure 2.1 Elevator system architecture. N_i, request floor location; N_c, current floor location; N_d, destination floor location; T_j: sequence j response time; P_j, probability of sequence j; t_{oc}, door open/close time; t_f, single floor traversal time.

process. Some of the elements, such as *floor request i*, represent actions in an elevator system, while others, such as *probability of traversing sequence j*, are metrics for evaluating elevator system performance; therefore, these elements have no "source" nor "destination." Element transfer rates are measured by floor travel distances relative to the time of travel:

Table 2.2 Transfer Rates (Floors per Second)

Element	Sequence	5 seconds floor traversal time	1 second floor traversal time
Sequence j	1	0.3845	1.8490
Sequence j	2	2.2082	10.2118
Sequence j	3	0.5054	2.4605
Sequence j	4	1.0765	5.1246
Floor request i N_i	1	0.8175	3.9309
Floor request i N_i	2	0.8166	3.9265
Floor request i N_i	3	0.8900	4.2796
Floor request i N_i	4	0.7054	3.3920
Current floor location N_c	1	0.7906	3.8013
Current floor location N_c	2	0.8025	3.8586
Current floor location N_c	3	0.7468	3.5911
Current floor location N_c	4	0.7455	3.5844
Destination floor N_d	1	0.8331	4.0060
Destination floor N_d	2	0.7705	3.7051
Destination floor N_d	3	0.8017	3.8550
Destination floor N_d	4	0.7955	3.8252

Sequence j travel distance:

$$\frac{\sum_{i=1}^{n} n_i}{\sum_{i=1}^{n} T_j(i)},$$

where n_i is travel distance for floor request i; Tj(i) is the response time for sequence j and floor request i; and n is the number of floor requests.

Request floor travel distance N_i:

$$\frac{\sum_{i=1}^{n} N_i(j,n)}{\sum_{i=1}^{n} T_j(i)},$$

where $N_i(j, n)$ is the request floor location i for sequence j and the number of floor requests n.

Current floor travel distance N_c:

$$\frac{\sum_{i=1}^{n} N_c(j,n)}{\sum_{i=1}^{n} T_j(i)},$$

where $N_c(j, n)$ is the current floor location for sequence j and number of floor requests n.

Destination floor travel distance N_d:

$$\frac{\sum_{i=1}^{n} N_d(j,n)}{\sum_{i=1}^{n} T_j(i)},$$

where $N_d(j, n)$ is the destination floor location for sequence j and the number of floor requests n.

Transfer rates are computed using the above expressions and recorded in Table 2.2. Table 2.2 shows that, in general, sequence j transfer rates are higher for the shorter floor traversal time alternative (1 second). This result is expected because more floors are traversed during shorter response times. This result can be used to advantage by anticipating *prior* to system implementation the transfer rates produced by performance alternatives.

ARCHITECTURAL DESIGN

Computer architecture involves the organization and functions of various elements into a unified system. Architectural design involves conceptualizing the subsystems of a system in terms of components, units, functions, elements, and connections. Table 2.3 documents the architectural relationships for the elevator example, where elements from Table 2.1 and Figure 2.1 are related to the architecture. An important architectural feature includes test points in Table 2.3 and Figure 2.1 for testing and evaluating the reliability of the system.

TEST STRATEGIES

An excellent test strategy is to view testing as a means of fault prevention [SIG90], meaning that if we identify a system's vulnerabilities before committing to programming and detailed hardware design, we can avoid these weak spots when actually committing to code and hardware implementation. For example, in an elevator system, the sequences of floor traversals are critical in realizing a reliable operation. Thus, a key strategy of fault prevention is to focus on critical sequences

Table 2.3 Architectural Relationships

Subsystem	Component	Unit	Function	Element(s)	Connections
Push button sensor	Push button control	Push buttons test point	Generates user requests	Floor request i N_i, destination floor N_d	Elevator controller
Floor location sensor	Error signal generator	Test point	Provides floor locations	Current floor location N_c	Cache memory exception handler
Cache memory		Test point	Short-term storage of N_i, N_d, and N_c	Current floor location N_c	Floor location sensor elevator controller
Elevator	Elevator controller	Microprocessor	Computes sequence j, sequence j response time T_j, and probability of sequence j P_j	Sequence j, sequence j response time Tj, and probability of sequence j P_j	Elevator controller cache memory long-term storage
Long-term storage			Long-term storage: Sequence j, sequence j response time T_j, and probability of sequence j P_j	Sequence j, sequence j response time T_j, and probability of sequence j P_j	Elevator controller
Exception handler	Microprocessor	Error signal generator	Process error signals	Current floor location N_c	Recover and rollback

[GAN08], emphasizing stress testing (e.g., simulating multiple concurrent user floor requests).

The test specification should contain the functions to be tested, types of faults expected, input data, expected outputs, pass/fail criteria, test environment, and test schedule [SIG90]. The types of faults expected is critical and warrants elaboration [SIG90]. One type of fault is classified as functional; it pertains to faults arising from errors in designing and implementing functions, such as programming an elevator to go down when it is supposed to go up. To avoid or correct this problem, functional testing is designed to simulate critical functions and to compare the test result with the expected result. If the two are unequal, the relevant documentation is analyzed to identify the source of the error. For example, if a test result shows an elevator going down when it is supposed to up, the floor traversal sequencing algorithm would be investigated.

System faults can arise due to components not being properly interfaced, such as a system bus not having the bandwidth required to accommodate the myriad of devices connected to it. Another type of system fault is failing to specify the correct capacity of components, such as a cache size that is too small for storing floor traversal information in Figure 2.1. A key process fault is generated by incorrect processing of sequences, for example, sequence j directing the elevator to go down, when it is supposed to go up, in Figure 2.1. Data faults are created by incorrect specification of value, limit, or format. For example, the user floor request data in Table 2.1 is specified as three binary-coded decimal (BCD) integer digits in a 100-floor elevator system. If only two BCD digits floating-point digits were specified, there would be a limit error (only 99 floors could be accommodated) and a format error (floating-point representation). Values would also be incorrect if the sensor control in Figure 2.1 reads request floor 5 instead of the correct floor 10.

Test Plan

Now, we illustrate test planning by developing a sequence-oriented test plan for an elevator system that includes the analysis of critical faults and test plan support functions.

Critical Faults

One type of critical fault occurs when boundary values are not processed correctly [SIG90]. For example, this fault occurs when elevator floor location processing results in floor location exceeding 100 in a 100-floor elevator system, or the location is computed to be less than one. This problem can be mitigated by providing test points at data entry locations in Figure 2.1 and checking for boundary value errors.

Another type of critical fault can occur at the interface of two elements [SIG90]. This type of fault occurs, for example, when two elements must match with respect to transmitted data type. Using the interface between the floor location sensor and the cache memory in Figure 2.1 as an example, the data type must be 3 BCD integer,

which was originally specified in Table 2.1. This problem is controlled by using integration testing to ensure that every interface behaves properly.

Test Plan Support Functions

In order to conduct tests in the absence of certain elements, drivers that substitute for missing *calling* elements are used. In addition, stubs that substitute for missing *called* elements are used. This would be the case, for example, in testing Sequence 1 in Figure 2.1, when the request floor location (N_i) element is to be tested, but both the current floor location (N_c) and destination floor location (N_d) elements have not been implemented. In this situation a driver would substitute for N_c (calling element) and a stub would substitute for N_d (called element).

Integration testing involves testing the hardware and software for each sequence on an incremental basis, using the test points in Figure 2.1. For example, once the testing of the floor location sensor is completed, it is incorporated into the elevator system and the cache memory is the next element tested. This process continues until all elements have been tested. If any hardware or software changes are made along the way, *regression testing* is invoked to retest all elements that had been tested up to the point of the changes [SIG90].

Environmental testing, which can be equated to system testing and acceptance testing, is designed to test and evaluate a system in its operational environment. However, during system development, it may not be possible to test some systems in their *actual* operational environment. Obviously, it would not be feasible to test elevators in various buildings where they will be installed because these environments would be unknown during development. Instead, the elevator system manufacturer uses a test bed to create environmental conditions as close to the ultimate environment as possible for testing the *physical* system. However, prior to this phase, the operation of elements and their sequences is simulated, which will be described in a later section. Environmental testing includes not only testing of functions, but performance testing as well. For example, the elevator system in Figure 2.1 would be tested for the correct ordering of sequences and, in addition, tested for the correct computation of sequence response times.

Test Data Design

The design of test data is as important as developing test plans because these plans would be worthless if the test data do not support the plan. In addition to inputting the correct floor location data (3 BCD integer) in Figure 2.1, *incorrect* data (e.g., 2 BCD floating point) should be inputted to test the system response. Systems should be equipped with an exception handler, as shown in Figure 2.1, to be activated when errors occur, such as incorrect data type. This is an excellent example of coordinating test data design with test plans: erroneous test data are incorporated in the test plan to test the system response—recover from the error and rollback to the last correct operation, or halt the operation if it is impossible to recover from the error. Table 2.1 is an excellent source for designing test data, because for each element, the purpose, source, format, storage requirement, and computation are documented. This

information can be correlated with the architectural perspective in Figure 2.1 (e.g., location of test points) to develop integrated test plans and data.

FAULT DETECTION AND CORRECTION

In the design process, it is important to plan for both fault detection and correction. Some methods, like simple parity detection, can only detect. Other methods, such as the cyclic redundancy check (CRC), include sufficient information to allow both detection and correction.

Parity Error Detection

If the number of one bits, including the parity bit **P** bit, is an even number, an even parity error signal is generated; otherwise, if the number of one bits, including the parity bit, is an odd number, an odd parity error signal is generated.

For example, if it is desired to use even parity error detection in a digital circuit, which of the data below would generate an *even* parity error signal? The solution is shown below.

P

0010 odd parity

0100 odd parity

0101 even parity (solution)

0111 odd parity

CRC

Data can be represented by a polynomial M(x): $x^n + x^{n-1} + x^{n-2} + \ldots + x^0$, x = 2 and the exponents correspond to bit positions: 0 for position 0, 1 for position 1, and so on. The degree of the polynomial is n that is equal to the highest bit position. For example, if we consider floor 50 in the elevator system, the polynomial representation of $M(x) = x^5 + x^4 + x^1 = 32 + 16 + 2 = 50_{10} = 110010_2$, and the degree n = 5.

Continuing the example, the sender and receiver must agree on a generator polynomial G(x) of degree $k \leq n$ in advance of transmission. Both the high and low bits of G(x) must be 1. M(x) must be longer than G(x). In addition, k zeros are appended to M(x), yielding the augmented message $T(x) = M(x)\, x^k$.

The details of the example follow:

$$M(x) = x^5 + x^4 + x^1.$$

Use $G(x) = x + 1 = 3_{10} = 11_2$, because M(x) can be divided by G(x) (i.e., the degree of G(x) = 1 ≤ degree of M(x) = 5).

Degree k = degree 1; therefore, append one zero to M(x), yielding

$$T(x) = M(x)x^k = x^6 + x^5 + x^2 = 1100100_2 = 100_{10}.$$

Divide T(x) by G(x), using modulo 2 division, and record remainder R(x) using modulo 2 division:

$$
\begin{array}{r}
100001 \\
11\overline{)1100100} \\
11 \\
\hline
0000100 \\
11 \\
\hline
\mathrm{R(x)} = 01
\end{array}
.
$$

Now, append the remainder R(x) = 01 to the original message M(x), using modulo 2 addition, and *transmit* M(x)R(x) = $11001001_2 = 201_{10}$ (check: 201_{10} is divisible by 3 with 0 remainder).

At the receiver, divide (M(x)R(x)) by G(x) and check for zero remainder. If this is the case, there is no error in transmission; otherwise, there is one error, so retransmit:

$$
\begin{array}{r}
100001 \\
11\overline{)1100100} \\
11 \\
\hline
0000100 \\
11 \\
\hline
11 \\
11 \\
\hline
0 \text{ (no error)}
\end{array}
.
$$

SEQUENCE ANALYSIS

Sequence Relationships

We perform sequence analysis to provide a structure for predicting reliability, availability, and performance. Figure 2.1 shows the architecture for processing the data that are used for constructing the sequences. In addition, the figure shows the details of the sequences. These details are used for predicting reliability, availability, and performance. This is accomplished by first, identifying the relationships among the request floor location, N_i, the current floor location, N_c, and the destination floor location, N_d. Using these relationships, floor travel is identified in order to predict the probability of the elevator system invoking sequence j. Once the probabilities are predicted, the response time for each sequence is predicted for each of the floor traversal times. Since the floor locations are unknown prior to implementing the system, uniformly distributed random numbers, multiplied by 100, are used to generate floor locations for a 100-floor system. In order to achieve statistical validity, 100 tests are simulated for each of the four sequences that are shown in Figure 2.1. Then,

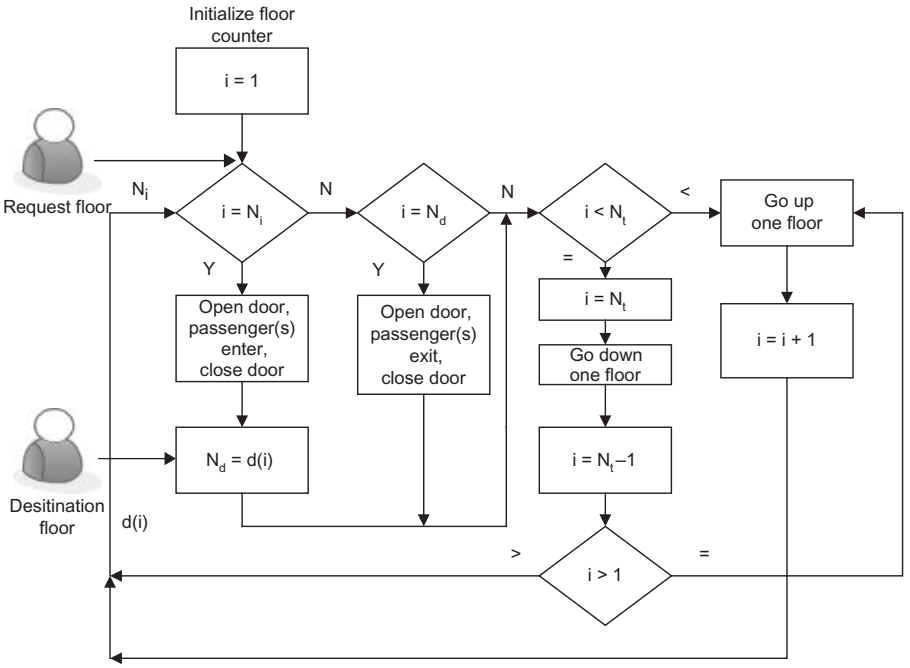


Figure 2.2 Elevator logic diagram. N_i, request floor; N_d, destination floor; d(i), user-entered destination floor; N_t, top floor.

the floor constraints (e.g., ($N_c \geq N_i$, $N_d \geq N_i$) for Sequence 1), which are documented in Figure 2.1, are used to identify the combinations of floor locations that are valid for a given sequence. These valid combinations are used to generate floor travel distances for each sequence. Next, the sequence probabilities are computed as a function of floor travel distances as shown in Figure 2.1.

In addition to the sequence analysis of Figure 2.1, the logic of the elevator operations are shown in Figure 2.2 to allow us to visualize how the elevator system executes the logic steps to control to transport passengers from floor to floor. This diagram would be the basis for the software design of the floor traversal algorithm.

SEQUENCE PROBABILITY AND SEQUENCE RESPONSE TIME PREDICTIONS AND ANALYSIS

Sequence Probability

Sequence 1

(1) Elevator goes *down* from current floor N_c to request floor N_i, then (2) goes *up* from request floor N_i to destination floor N_d ($N_c \geq N_i$, $N_d \geq N_i$):

$$P_j = (N_d - N_i) / ((N_c - N_i) + (N_d - N_i)).$$

Sequence 2

(1) Elevator goes *up* from current floor N_c to request floor N_i, then (2) goes *up* from request floor N_i to destination floor N_d ($N_i \geq N_c$, $N_d \geq N_i$):

$$P_j = (N_d - N_i) / ((N_i - N_c) + (N_d - N_i)).$$

Sequence 3

(1) Elevator goes *up* from current floor N_c to request floor N_i, then (2) goes *down* from request floor N_i to destination floor N_d ($N_i \geq N_c$, $N_c \geq N_d$):

$$P_j = (N_i - N_d) / ((N_i - N_c) + (N_c - N_d)).$$

Sequence 4

(1) Elevator goes *down* from current floor N_c to request floor N_i, then (2) goes *down* from request floor N_i to destination floor N_d ($N_c \geq N_i$, $N_i \geq N_d$):

$$P_j = (N_i - N_d) / ((N_c - N_i) + (N_i - N_d)).$$

Sequence Response Time

Response time predictions are based on the above predictions of sequence probability, which correspond to the sequences depicted in Figure 2.1.Response time is predicted for each sequence j and floor traversal time, t_f, using the following equations:

Sequence 1

$$T_j = (t_f * ((N_d - N_i) + (N_c - N_i))) * P_j + t_{oc}.$$

Sequence 2

$$T_j = (t_f * ((N_i - N_c) + (N_d - N_i))) * P_j + t_{oc}.$$

Sequence 3

$$T_j = t_f * ((N_i - N_c) + (N_c - N_d)) * P_j + t_{oc}.$$

Sequence 4

$$T_j = t_f * ((N_c - N_i) + (N_i - N_d)) * P_j + t_{oc}.$$

The purpose of the predictions is twofold: (1) to assess in advance of implementation which *performance alternative* would satisfy the performance requirement and (2) to identify the *sequence(s)* that would satisfy the performance requirement. This objective is accomplished by simulated testing. Figures 2.3 and 2.4 address this purpose. Figure 2.3 pertains to the 5-second floor alternative and shows that none of the sequences satisfy the performance requirement over the complete range of tests. Figure 2.4 is a little more encouraging, showing that Sequence 2 satisfies the requirement. Unfortunately, since it is infeasible to provide an elevator system with a floor traversal time of less than 1 second, the performance in actual operation is

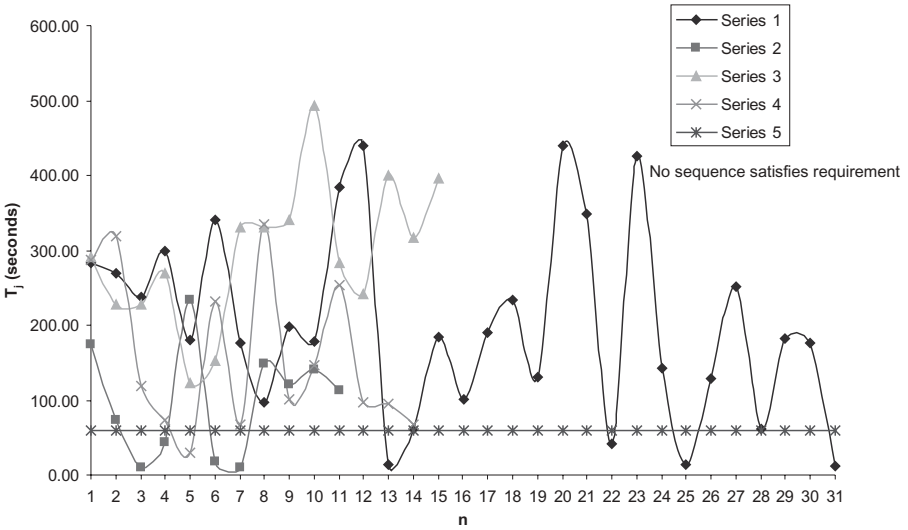


Figure 2.3 Elevator system: sequence response time T_j versus number of tests n for 5 seconds floor traversal time. Series 1: Sequence 1: mean = 200.77 seconds, total = 6223.87. Series 2: Sequence 2: mean = 95.53 seconds, total = 1083.85. Series 3: Sequence 3: mean = 295.97 seconds, total = 4735.48. Series 4: Sequence 4: mean = 158.80 seconds, total = 2223.17. Series 5: required response time.

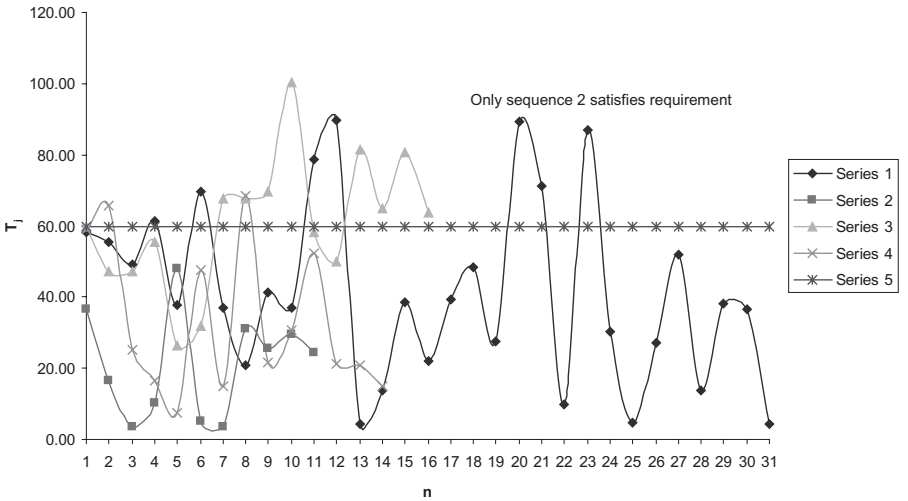


Figure 2.4 Elevator system: sequence j response time T_j versus number of tests n for 1 second floor traversal time. Series 1: Sequence 1: mean = 41.75 seconds, total = 129.37 seconds. Series 2: Sequence 2: mean = 21.31 seconds, total = 234.37 seconds. Series 3: Sequence 3: mean = 60.79 seconds, total = 972.70 seconds. Series 4: Sequence 4: mean = 33.36 seconds, total = 467.03 seconds. Series 5: required response time.

likely to be undesirable for some of the floor requests. However, this performance may be acceptable because the majority of all sequence response times in Figure 2.4 satisfies the requirement over all of the tests.

SEQUENCE FAILURE RATE

In order to predict sequence reliability, it is necessary to estimate sequence j failure rate λ_j, a parameter that is used in the prediction of sequence j reliability. This parameter is estimated using the number of failures, n_f, that is specified to occur during n tests of sequence j, and sequence j response time, T_j. In addition, we postulate that the *expected* number of failures in sequence j is proportional to sequence j floor traversal distance for test i, n_i, with respect to total floor traversal distance for sequence j, based on the premise that the larger the floor traversal distance, the higher the probability of failure. Putting these factors together, we arrive at the following:

$$\lambda_j = n_f \left(\frac{n_i}{\sum_{i=1}^{n} n_i} \right) / \left(\sum_{i=1}^{n} (T_j) \right).$$

A key determinate of sequence failure rate is whether there are failures in delivering information from source to destination [YOU09], such as user floor request to sensor control in Figure 2.2. This factor is captured in the above failure rate prediction by the specified number of failures n_f.

RELIABILITY

In developing real-time reliability predictions, it is important that the predictions reflect *operational reliability* [SUN05]. That is, reliability must be cast in the context of operational conditions, such as differences in floor traversal times in the elevator system. Otherwise, the predictions will not represent user requirements. We adhere to this principle by using sequence response time, which represents operational conditions, in the formulation of reliability.

The unreliability of sequence j, UR_j, is predicted by using the probability of sequence j, P_j, sequence failure rate λ_j, and sequence j response time, T_j, assuming exponentially distributed response time. The distinction between normal and complex operations is important in characterizing reliability [PET06]. This is why we assume exponentially distributed response time, which is based on the premise that reliability degrades fast with increasingly complex operations, as represented by increasing floor traversal time and resultant increasing response time:

$$UR_j = (P_j)(1 - e^{-\lambda_j T_j}).$$

Then, sequence j reliability R_j can be predicted as follows:

$$R_j = 1 - ((P_j)(1 - e^{-\lambda_j T_j})).$$

Because numerous predictions of reliability are made due to the fact that sequences are simulated *n* times during tests, it is appropriate to predict the mean value, as follows:

$$MR_j = \frac{\sum_{j=1}^{n} R_j}{n}.$$

Figures 2.5 and 2.6 address the reliability issue, predicting sequence j reliability as a function of a number of tests. Figure 2.5 shows that for a number of specified failures = 1 and floor traversal time = 5 seconds, all sequences satisfy the reliability requirement. In addition, Figure 2.6 indicates that for a number of specified failures = 5 and floor traversal time = 1 second, all of the sequences satisfy the reliability requirement. Although the reliability requirement is satisfied in both Figures 2.5 and 2.6, if a high reliability system is desired, operating in a dense failure environment, significant testing would be required to bring the system into conformance with the reliability requirement.

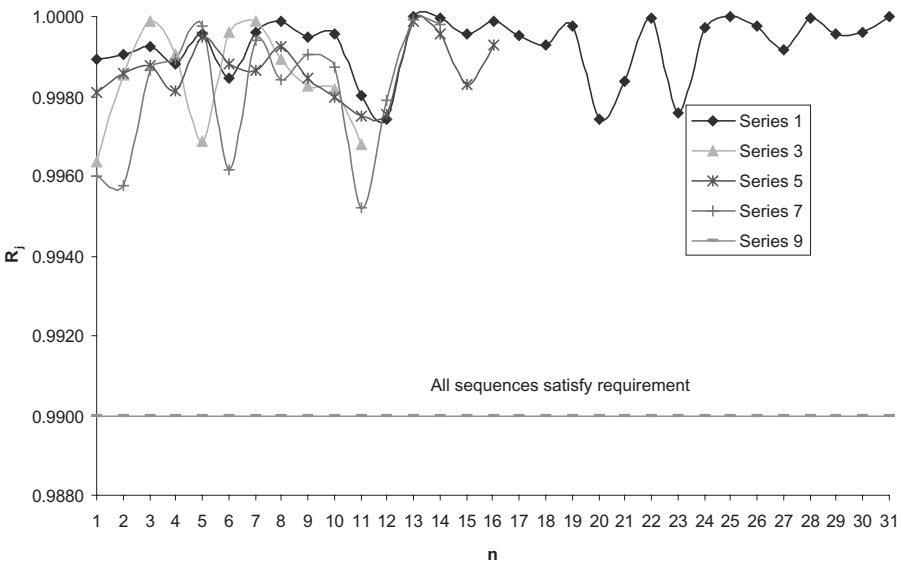


Figure 2.5 Elevator system: reliability of sequence j R_j versus number of tests n for sequence j number of failures = 1 and floor travel time = 5 seconds. Series 1: Sequence 1, mean = 0.9993. Series 3: Sequence 2, mean = 0.9984. Series 5: Sequence 3, mean = 0.9986. Series 7: Sequence 4, mean = 0.9981. Series 9: required reliability.

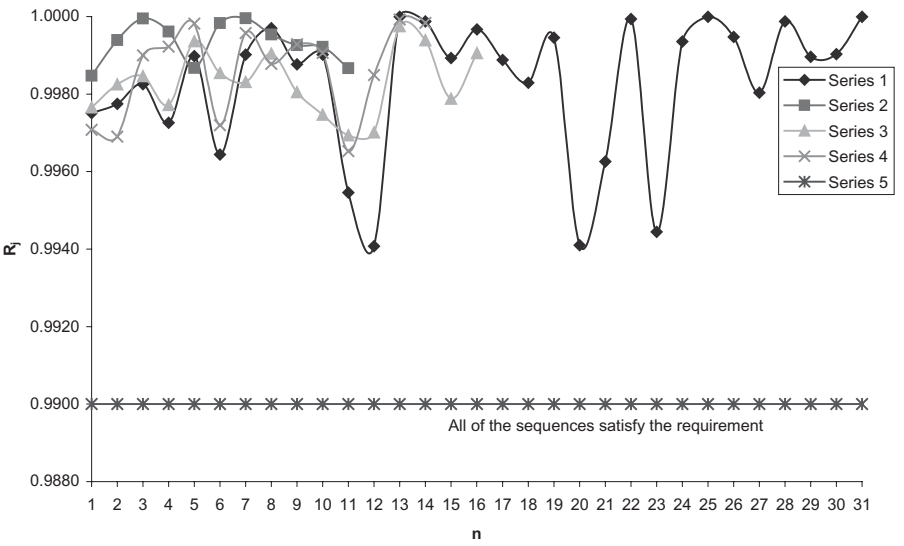


Figure 2.6 Elevator system: reliability of sequence j R_j versus number of tests n for sequence j number of failures = 5 and floor traversal time = 1 second. Series 1: Sequence 1, mean = 0.9964. Series 2: Sequence 2, mean = 0.9921. Series 3: Sequence 3, mean = 0.9933. Series 4: Sequence 4, mean = 0.9908. Series 5: required reliability.

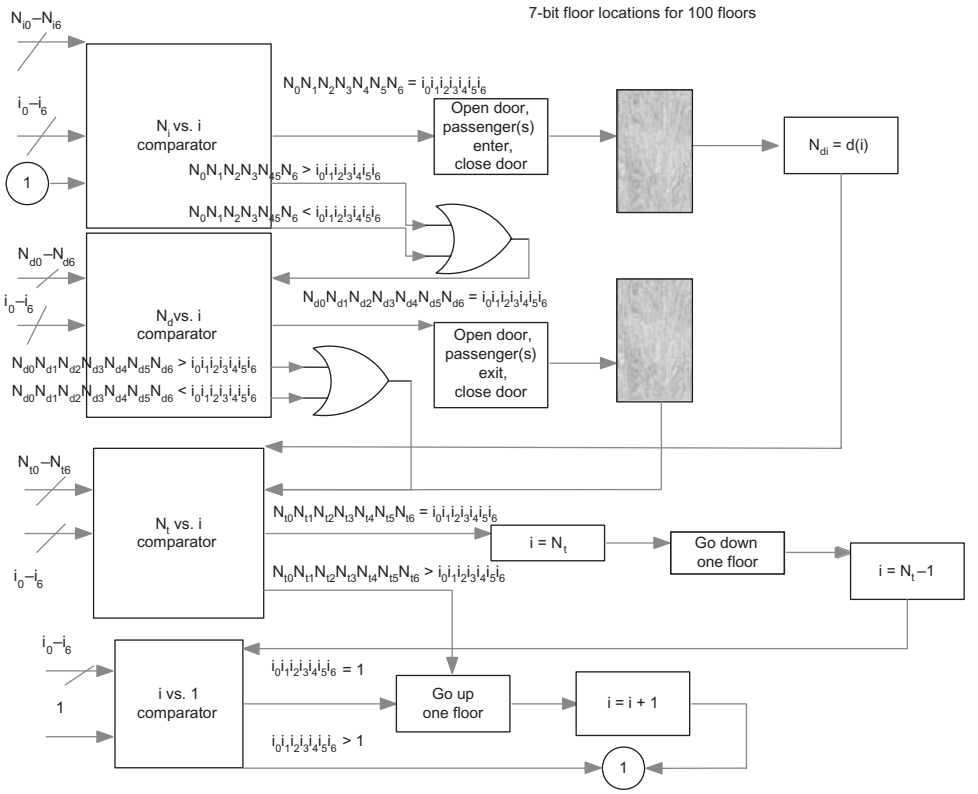


Figure 2.7 Elevator comparator circuit. N_i, request floor location; N_d, destination floor location; i, floor counter; N_t, top floor; d(i), user-entered destination floor.

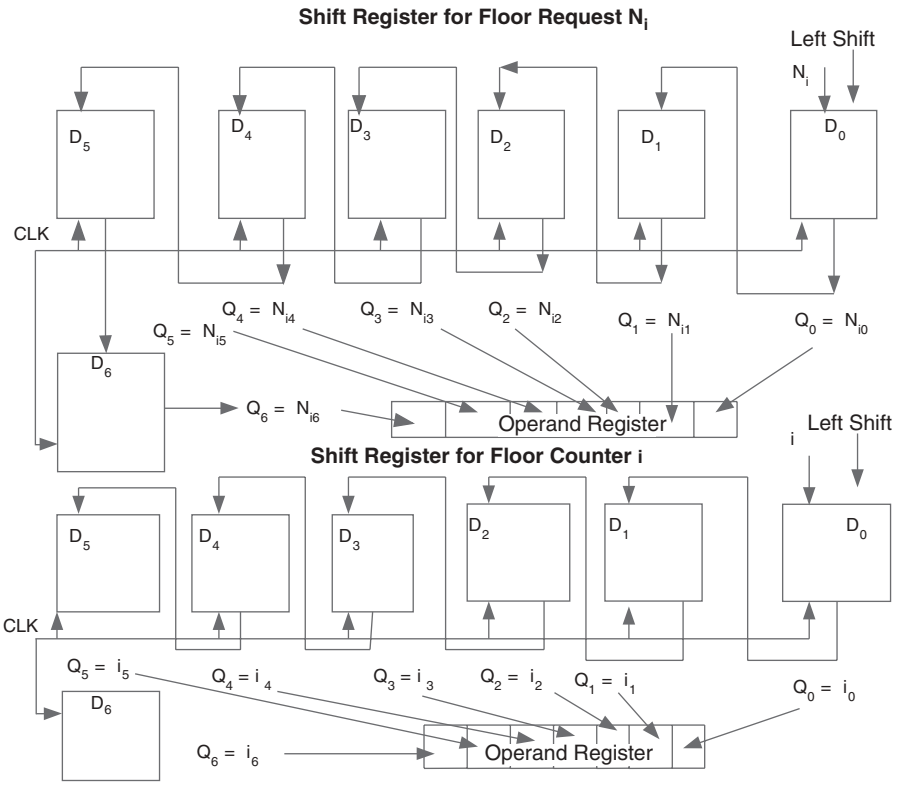


Figure 2.8 Elevator detailed logic diagram. i, floor counter; N_i, request floor bit number; D, flip-flop input; Q, flip-flop output; CLK, clock input.

DETAILED DESIGN

Next, the detailed design process is illustrated by implementing the logic processes from Figure 2.2, as shown in Figure 2.7, where the detailed logic steps relate the floor request variable N_i and the floor counter parameter i. This is accomplished by using shift register logic on these two quantities, as shown in Figure 2.8, in order to store and align all 7 bits of the quantities so that they can be compared for equality and inequality, as governed by the relationships in Figure 2.2.

SUMMARY

A road map has been presented for guiding the engineer in making correct analyses and decisions in developing computer-based systems. An elevator system was used to illustrate the myriad of factors that must be considered in bringing a concept of a system to fruition as an integrated hardware–software system.

REFERENCES

[GAN08] J. Ganssle (ed.), *Embedded Systems: World Class Designs*. New York: Elsevier, 2008.

[HAR07] D. M. Harris and S. L. Harris, *Digital Design and Computer Architecture*. New York: Elsevier, 2007.

[PET06] N. Russ, G. Peter, R. Berlin, and B. Ulmer, "Lessons learned: on-board software test automation using IBM rational test realtime," *Second IEEE International Conference on Space Mission Challenges for Information Technology*, 2006, p. 305.

[SIG90] C. D. Sigwart, G. L. Van Meer, and J. C. Hansen, *Software Engineering: A Project-Oriented Approach*. Irvine, CA: Franklin, Beedle, & Associates, Inc., 1990.

[SUN05] Y. Sun, L. Cheng, H. Liu, and S. He, "Power system operational reliability evaluation based on real-time operating state," *The 7th International Power Engineering Conference*, Volume 2, November 29, 2005–December 2, 2005, pp. 722–727.

[YOU09] K. Mizanian, H. Yousefi, and A. H. Jahangir, "Modeling and evaluating reliable real-time degree in multi-hop wireless sensor networks." *IEEE Sarnoff Symposium*, March 30–April 1, 2009, pp. 1–6.

Chapter 3

Analog and Digital Computer Interactions

While digital computers dominate today's computer marketplace, digital computers have important interactions with analog devices; for example, a smart meter installed in a smart electric grid requires interactions between an analog voltage sensor, an analog-to-digital converter, a digital-to-analog converter, and a digital computer for computing power usage. A communication system is also required for distributing power usage data to the customer's premises and to the electric utility office. This chapter will provide the reader with the background in interfacing analog devices with digital computers that is necessary for designing and evaluating such systems.

INTRODUCTION

Analog Computer Background

An analog computer is a form of computer that uses the continuously changeable aspects of physical phenomena such as electrical, mechanical, or hydraulic quantities to model the problem being solved. In contrast, digital computers represent varying quantities incrementally as their numerical values change. Mechanical analog computers were very important in gunfire control in World War II and the Korean War; they were made in significant numbers. In particular, development of transistors made electronic analog computers practical, and before digital computers had developed sufficiently, they were commonly used in science and industry. In particular, perhaps the best known example of an analog computer is the slide rule.

The similarity between linear mechanical components, such as springs and dashpots (viscous fluid dampers), and electrical components, such as capacitors, inductors, and resistors, is striking in terms of mathematics. They can be modeled using equations that are of essentially the same form. However, the difference

Computer, Network, Software, and Hardware Engineering with Applications, First Edition. Norman F. Schneidewind.
© 2012 the Institute of Electrical and Electronics Engineers, Inc. Published 2012 by John Wiley & Sons, Inc.

between these systems is what makes analog computing useful. If one considers a simple mass–spring system, constructing the physical system would require making or modifying the springs and masses. This would be followed by attaching them to each other and to an appropriate anchor, collecting test equipment with the appropriate input range, and finally, taking measurements. In more complicated cases, such as suspensions for racing cars, experimental construction, modification, and testing is not so simple or inexpensive.

The electrical equivalent of a physical system can be constructed with a few operational amplifiers and some components, such as resistors and capacitors; all electrical measurements can be made with an oscilloscope. In the circuit, the simulated stiffness of the spring, for example, can be changed by adjusting a potentiometer. The electrical system is an analogy to the physical system, hence the name, but it is less expensive to construct, generally safer, and typically much easier to modify.

An electric circuit can typically operate at higher frequencies than the physical system being simulated. This allows the simulation to run faster than real time (which could, in some instances, be hours, weeks, or longer). These electric circuits can perform a wide variety of simulations. For example, voltage can simulate water pressure and electric current can simulate rate of flow. Analog computers are especially well suited to representing situations described by differential equations.

Analog-to-Digital and Digital-to-Analog Components

Sensor

A voltage sensor reads voltage at the input of an analog-to-digital (A/D) converter circuit, as shown in Figure 3.1, and the output of a digital-to-analog (D/A) converter in Figure 3.6. Note that Figures 3.1 and 3.6 depict a smart electric meter system with the sensor reading the input voltage or output voltage, respectively; other signals, such as current, could be sensed in other applications.

Operational Amplifier

An operational amplifier produces an output voltage that is larger than its input. For example, in Figures 3.1 and 3.6, the voltage sensor does not have the capability to read a full-scale voltage signal; its range is only 10 V. Therefore, an operational amplifier is used to boost the signal to the required level for conversion. Operational amplifiers are important building blocks for a wide range of electronic circuits.

Low-Pass Filter

A low-pass filter is needed to reduce high frequency signal noise by only passing signals to the multiplexer in Figure 3.1 and to the customer premise and public electric utility in Figure 3.6 that have had the high frequency noise components eliminated [GAN08].

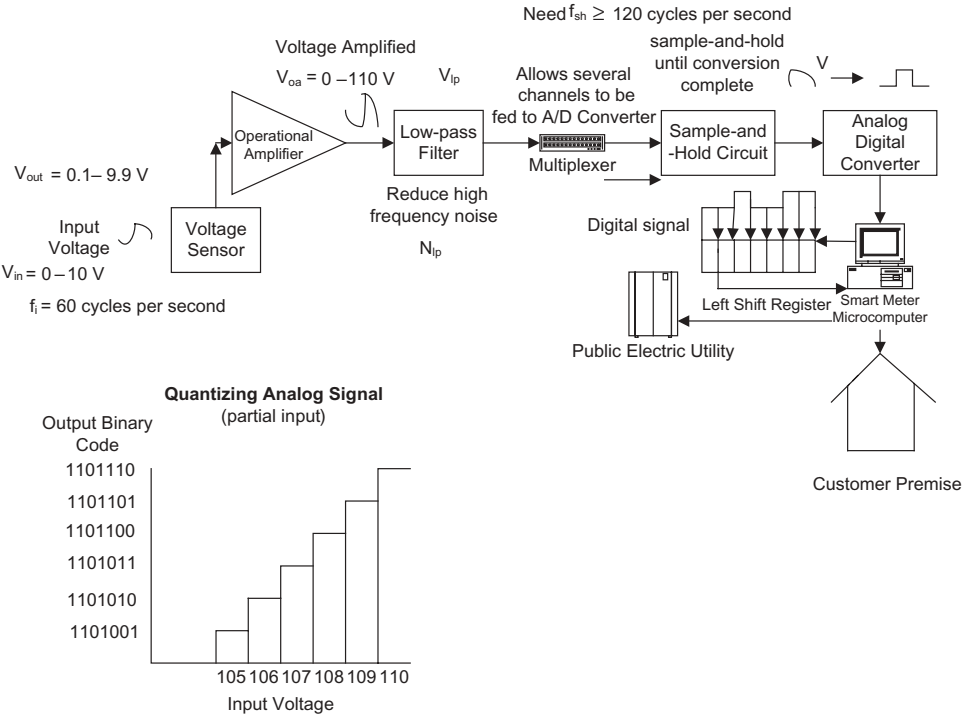


Figure 3.1 A/D conversion system.

Multiplexer

Because both A/D converters and D/A converters are expensive, a multiplexer is used in Figure 3.1 to allow several analog signals to be processed for conversion by a single A/D converter, and a multiplexer is used in Figure 3.6 to provide several channels to be fed to the D/A converter [GAN08]. In the Figure 3.1 example, the several analog inputs could be voltage signals from several customers in the neighborhood. In the Figure 3.6 example, the several digital outputs from the microcomputer could be destined for conversion to analog voltages for a voltage regulation application in the customer premise and public electric utility.

Sample-and-Hold Circuit

A sample-and-hold circuit is used to avoid having the input change while A/D conversion is taking place in Figure 3.1 and having the digital output change while D/A conversion is taking place in Figure 3.6 [GAN08].

A/D Converter

The details of the A/D converter and the conversion process are shown in Figure 3.2.The capacitor C in Figure 3.2 assists in the conversion of analog input to digital

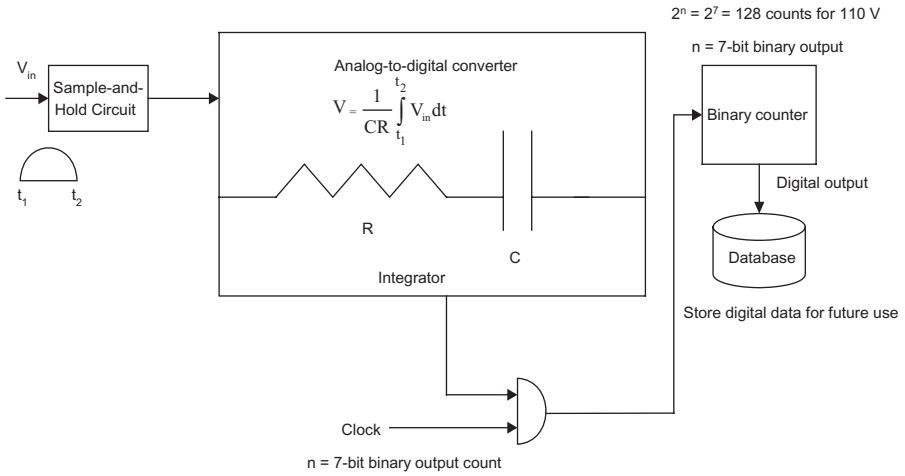


Figure 3.2 A/D converter.

output by the duration of its charge. This is accomplished by measuring the time it takes to charge and discharge the capacitor into the resistor R. The larger the value of C, for a given value of R, the longer it takes to charge and discharge the capacitor, and, hence, the slower the rise and fall in voltage, respectively. Conversely, the smaller the value of C, the less time it takes to charge and discharge the capacitor, and, hence, the faster the rise and fall in voltage, respectively. The converter integrates the varying analog input signal voltage V_{in} in Figure 3.2 during the time period t_1, t_2. At the end of this process, C has been charged by V_{in} to generate the voltage V across the resistance–capacitance (RC) circuit, given by:

$$V = \frac{1}{CR}\int_{t_1}^{t_2} V_{in}dt.$$

Finally, the voltage V is converted to the digital output.

Figure 3.1 shows an example of the results of the conversion process called "quantizing analog signal"—a process that converts an analog signal to a digital binary code. For example, 110 V is transformed into 1101110. This process is not perfect. There are errors that result because it is impossible for a conversion system to perfectly represent the input. These errors are called "quantizing errors," which will be addressed in a later section.

Smart Meter Microcomputer

This computer contains a left shift register that formats the bit-by-bit A/D conversion so that the complete digital signal (e.g., 1101110 = 110 V) is ready for distribution to customer premise and public electric utility in Figure 3.1. In Figure 3.2, the microcomputer is assigned to provide its stored digital signal to the input of the D/A converter.

Analog Computer Limitations

Analog computers have limitations. An analog signal is comprised of three characteristics: alternating current (AC) voltage and current magnitudes, frequency, and phase. The range limitations of these characteristics limit analog computers. These limits include operational amplifier amplification capability, gain, frequency response, noise, and nonlinearities in A/D conversions.

Analog–Digital Computer Contrast

The drawback of analog computers in imitating physical systems is that analog electronics are limited by the range over which the variables may vary. This is called *dynamic range*. They are also limited by noise levels. In contrast, digital computer floating-point calculations have a comparatively huge dynamic range (good modern handheld scientific/engineering calculators have exponents of 500).

An electronic digital system uses two voltage levels to represent binary numbers. In many cases, the binary numbers are simply codes that correspond, for instance, to brightness of primary colors, or letters of the alphabet. In contrast, the electronic analog computer manipulates electrical voltages that are proportional to the magnitudes of quantities in the problem being solved.

The accuracy of an analog computer is limited by its computing elements as well as quality of the internal power and electrical interconnections. The precision of the analog computer display readout is limited chiefly by the precision of the readout equipment, generally three or four significant figures. The precision of a digital computer is limited by its word size and degree of precision arithmetic. While the process is relatively slow, any practical degree of precision can be provided that might be needed.

Quantizing Step Size and Error

The quantizing step size for A/D conversion is defined by Q:

$$Q = R / 2^n,$$

where R is the range (110 V in Figure 3.1) and n is the number of bits used to code the digital output [GAN08]. Thus, for the smart meter example in Figure 3.1, where n = 7 bits,

$$Q = 110 / 128 = 0.8594 \text{ V per bit.}$$

Since Q is the smallest value recognizable by the A/D converter, it represents the error of conversion. In order to evaluate the rate of error occurrence with respect to the number of encoding bits n, the following derivative is produced:

$$\frac{d(Q)}{d(n)} = \frac{-R * n * 2^{(n-1)}}{(2^n)^2} = \frac{-R * n * 2^{(n-1)}}{(2^n)(2^n)} = \frac{-R * n}{2 * 2^n} = \frac{-R * n}{2^{(n+1)}}.$$

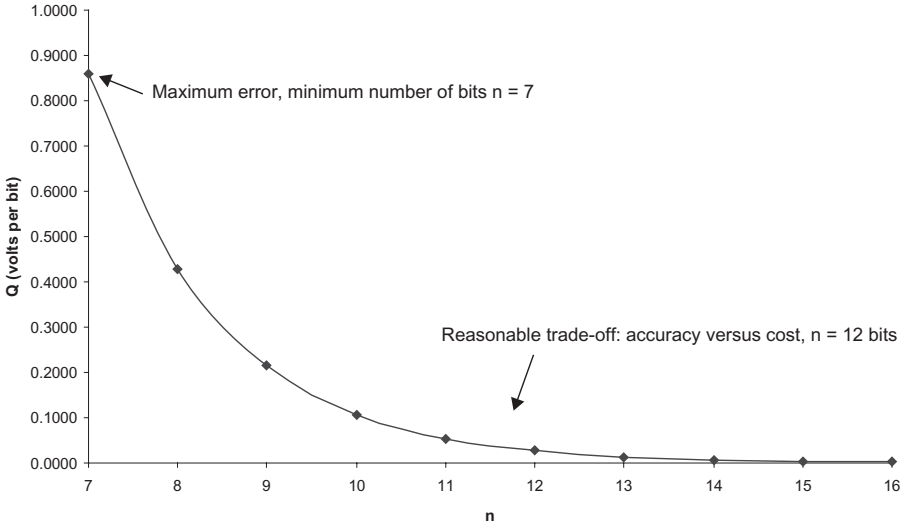


Figure 3.3 Quantizing error Q versus number of digital code bits n.

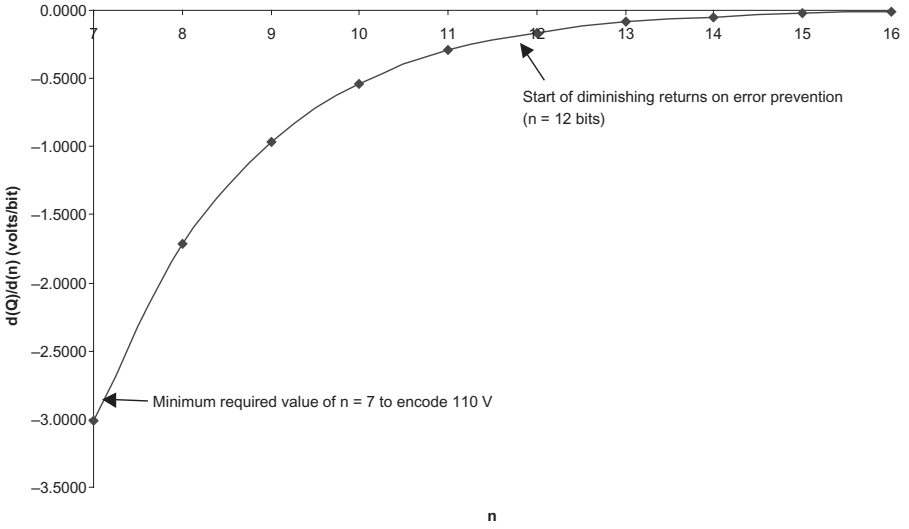


Figure 3.4 Rate of change of A/D conversion error, d(Q)/d(n) versus number of digital code bits n.

Figures 3.3 and 3.4 show that to encode 110 V in the smart meter application, 7 bits are required. However, by using more than the minimum, say 12 bits, the quantizing error can be significantly reduced, but more than 12 bits would not be cost-effective because at this point diminishing returns sets in.

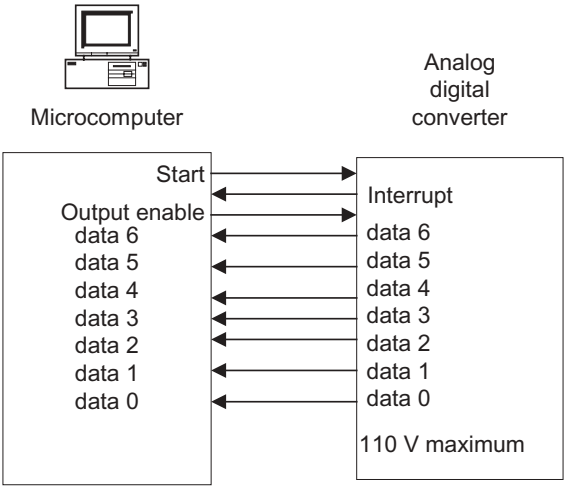


Figure 3.5 Interrupt processing with A/D converter. Start, microcomputer commands converted to start A/D conversion; output enable, enables digital output from converter to microcomputer; interrupt, converter signals microcomputer that it has digital data to transmit.

Microcomputer Input/Output (I/O) Applications

An A/D converter is very useful for demonstrating the various methods that an I/O device (e.g., A/D converter) can use to communicate with a microcomputer [RAF05]. For example, Figure 3.5 shows the interaction between an A/D converter and microcomputer, using interrupt processing. This method of I/O communication is very efficient because the microcomputer only has to be diverted from its main processing task when there are data from an I/O device to be processed. In Figure 3.5, this is accomplished by the interchange of commands between the converter and the microcomputer: the microcomputer commands the converter to start converting, the microcomputer signals to the converter that transfer of digital data from converter has been enabled, and the converter signals the microcomputer, via an interrupt, that digital data are ready for transmission on the data lines.

D/A CONVERSION

In addition to A/D conversion, it is important to understand how the reverse process works—D/A conversion. For example, you have seen that A/D conversion is an important component of smart meters in smart electric grid systems. But this is not the end of the story because D/A conversion is needed in these systems to take the digital voltage data as input to the D/A converter and use the resultant analog voltage output to act as a voltage regulator of the electric distribution system in Figure 3.6. This function is required because power disruptions could cause the voltage delivered to customer premises to be of the wrong magnitude. D/A conversion will not

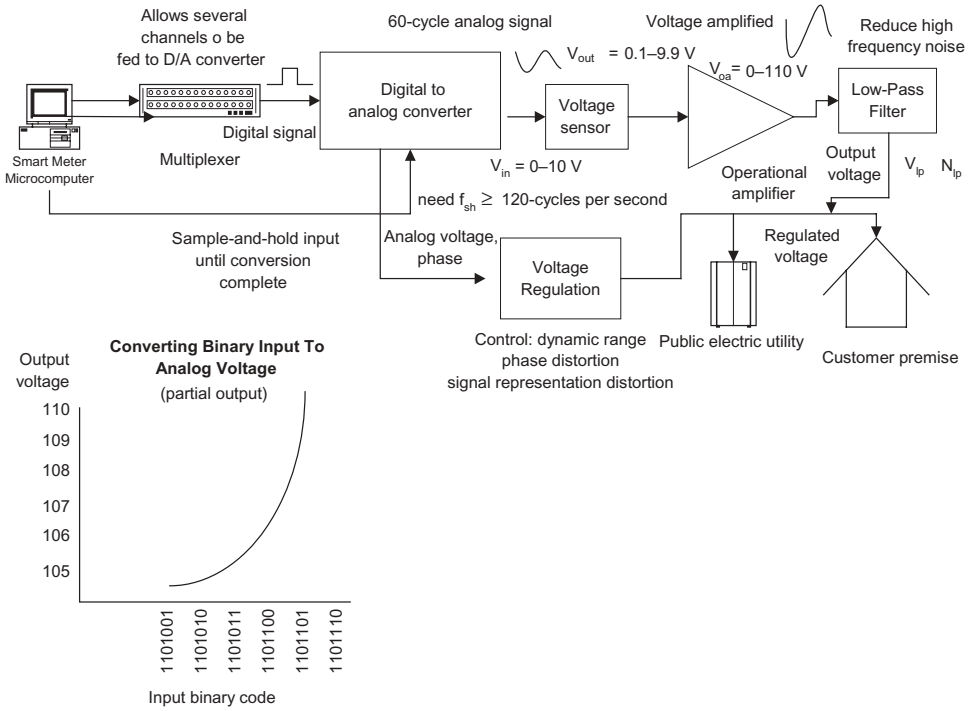


Figure 3.6 D/A conversion system.

be addressed as an isolated subject. Rather, it will be treated as part of a unified system that includes A/D conversion.

D/A Description

A D/A converter is a device for converting information that is in the form of a digital signal comprised of discrete binary bits (e.g., binary coded voltage) to a continuously varying analog signal (e.g., voltage sine wave) in Figure 3.6. D/A converters are used to present the results of digital computation (A/D voltage conversion in Figure 3.1) and storage (digital data stored in database in Figure 3.2) as input to the D/A converter in Figure 3.6 for eventual application in voltage regulation.

D/A Performance

Resolution

This is the number of possible output levels the D/A is designed to reproduce. This is stated as the number of bits it uses. For example, a 1-bit D/A is designed to reproduce two voltage levels while an 8-bit D/A is designed to reproduce 256 voltage levels. Thus the quantizing error Q is given by the following:

$$Q = 2^n / R,$$

where n is the number of binary bits produced by the digital signal in Figure 3.6 and R is the voltage range of the D/A output. Thus, the D/A quantizing error is the inverse of the A/D quantizing error.

Maximum Sampling Frequency

This is a measurement of the maximum speed at which the D/A (or A/D) circuitry must operate to reproduce the correct output. As stated in the Nyquist–Shannon sampling theorem, a signal must be sampled at least twice its frequency in order to produce the desired output signal. The period is the duration of one cycle in a repeating event, so the period is the reciprocal of the frequency. For example, the 60-cycle input voltage in the A/D converter of Figure 3.1 must be sampled at least 120 cycles per second. Correspondingly, in the D/A converter of Figure 3.6, the digital representation of the original 60-cycle voltage from A/D conversion must be sampled at least 120 cycles per second to reproduce a 60-cycle signal at the output of the D/A converter.

Monotonicity

This refers to the ability of a D/A converter's analog output to move only in the direction that the digital input moves (i.e., if the input increases, the output increases) This characteristic is very important when a D/A converter is used for low frequency signals, such as 60-cycle voltage, as shown in Figure 3.6.

Distortion

Distortion is the alteration of the original shape of the analog signal, such as a voltage signal. Distortion can be minimized by using an adequate number of bits in the digital representation of the analog signal, such as 7 bits, and a sampling rate of the digital signal for D/A conversion of at least twice its original frequency (at least 120 cycles per second) in Figure 3.6.

Dynamic Range

This is the absolute ratio between the smallest and largest possible values of a signal-changeable quantity, such as between the smallest and largest values of an analog voltage sine wave. In this example, if there is a perfect dynamic range, the ratio = |+110 V/−110 V| = 1. Deviations from the perfect ratio, either greater or smaller, are indicative of signal distortion.

Phase Distortion

This problem occurs when the original phase of a signal in the input of the A/D converter is not faithfully reproduced in the output of the D/A converter. For example,

the phase of the voltage sine wave sensed in the A/D converter of Figure 3.1 may not be faithfully reproduced at the output of the D/A converter in Figure 3.6. Phase distortion is measured by the difference between the correct phase and the phase that is reproduced at the output of the D/A converter.

Signal Representation Distortion

This problem occurs when, for example, a 1-V difference in the A/D converter in Figure 3.1 does not result in a 1-bit difference in the digital encoding. The problem would also occur when a 1-bit difference in the input of the D/A converter in Figure 3.6 does not result in a 1-V change in the output in Figure 3.6. Thus, this distortion is measured by the difference between the correct signal change in *adjacent* values and the actual change in *adjacent* values.

Dynamic range, phase distortion, and signal representation distortion are controlled by the voltage regulation function shown in Figure 3.6.

Nonlinearity Distortion

This occurs when the plot of the output signal versus the input signal is not a straight line, which is measured by the difference between the correct value and the value realized by D/A conversion [GAN08]. For example, in Figure 3.7, using assumed error values, the difference between actual and realized voltages values is plotted. This type of plot is extremely useful because it indicates the range where the converted voltage is either too high or too low. After these initial measurements have been made, the gain of the microcomputer-controlled operational amplifier in

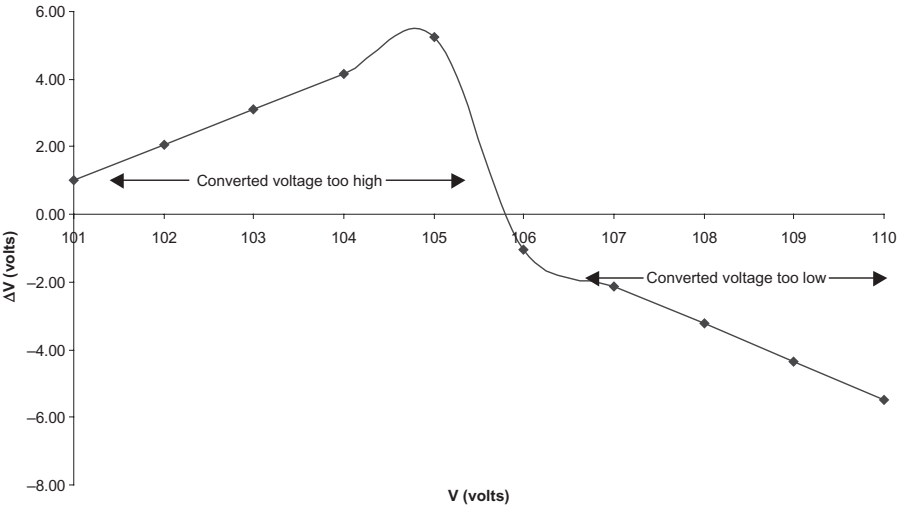


Figure 3.7 Difference between correct voltage and actual voltage, ΔV, versus correct voltage V.

Figure 3.6 would be adjusted to bring the converted voltage in line with the desired values.

CONVERSION SYSTEM ERRORS

At this point in the development of A/D and D/A conversion, it is time to focus on the errors that could arise in each component, whether A/D or D/A, and aggregate the component errors to produce an overall system error that can be used to judge the accuracy of conversion from analog input in the A/D converter in Figure 3.1 to D/A analog output in Figure 3.6.

A/D and D/A Converter

The error attributed to this device was previously described as a "quantizing error" in an earlier section and portrayed in Figures 3.3 and 3.4.

Voltage Sensor

A sensor is a device that receives and responds to a signal. A sensor's sensitivity indicates how much the sensor's output changes when the measured quantity changes. This sensitivity can be interpreted as sensor error. The sensor error, E_s, is computed by the following ratio:

$$E_s = \frac{\Delta V_{out}}{\Delta V_{in}},$$

where ΔV_{out} is the change in sensor output voltage in Figures 3.1 and 3.6 and ΔV_{in} is the change in sensor input voltage in Figures 3.1 and 3.6. Ideally, this ratio should equal one. Deviations from the ideal measure indicate sensor error. Additionally, the resolution error is determined by the smallest change in ΔV_{in} that can be detected at the sensor output. For example, in Figures 3.1 and 3.6, if the smallest change in $V_{in} = 0.1$ V, can this change be detected in V_{out}?

Another type of sensor error occurs when the input range exceeds the output range. For example, in Figures 3.1 and 3.6, while the input range of V_{in} is 0–10 V, the output range of V_{out} is 0.1–9.9 V. Thus, $V_{in} = 0$ and 10 V cannot be represented by V_{out}.

Since the changes in sensor input and output voltages could be any values, simulation can be used to generate random changes in voltage, say 100 times, and compute the resultant values of E_s. One example is shown in Figure 3.8, where the Excel random number generator RAND was used to generate random changes in V_{in} and V_{out}, and then E_s was computed and plotted against the correct A/D voltage. Because RAND generates uniformly distributed numbers between 0 and 1, these numbers were multiplied by 10—the maximum V_{in} voltage—in order to compute values of ΔV_{in} and ΔV_{out}.

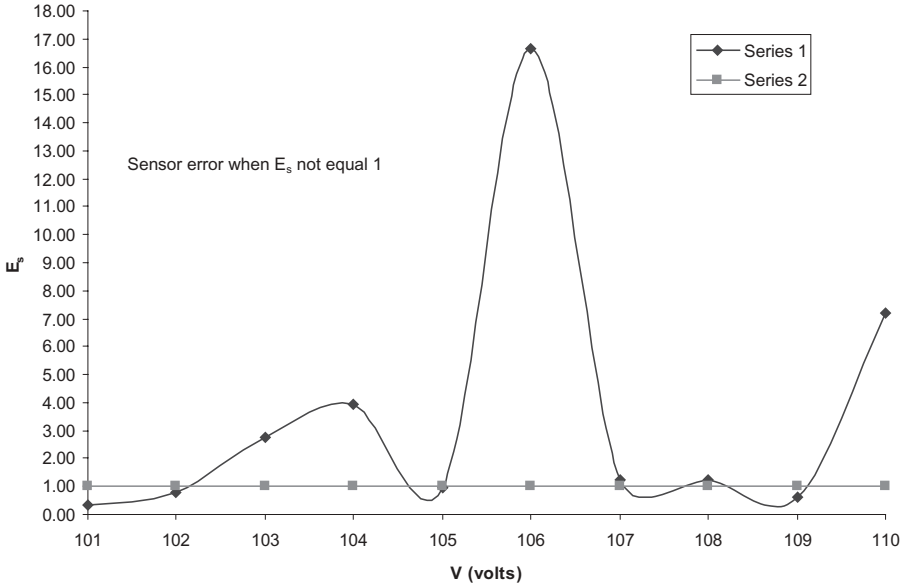


Figure 3.8 Sensor output voltage change/sensor input voltage change, E_s, versus correct A/D input voltage, V. Series 1: Actual E_s. Series 2: Required E_s.

Operational Amplifier

The operational amplifier in Figures 3.1 and 3.6 may fail to produce a correct amplification of the signal produced by the sensor output, V_{out}. This error is computed by the expression E_{oa}. A similar simulation error analysis can be performed to analyze the operational amplifier, as was the case for the voltage sensor:

$$E_{oa} = A - \frac{\Delta V_{oa}}{\Delta V_{out}},$$

where A is the required amplification factor, ΔV_{oa} is the actual change in operational amplifier voltage, and ΔV_{out} is the actual change in voltage sensor output voltage that is delivered to the input of the operational amplifier. Given the voltage sensor output range and operational amplifier output range in Figures 3.1 and 3.6, A = (110–0)/(9.9–0.1) = 11.22.

Low-Pass Filter

The low-pass filter in Figures 3.1 and 3.2 is also subject to error because it may not faithfully eliminate high frequency noise generated by the operational amplifier in Figure 3.1. The ability of the filter to eliminate this noise is measured by the signal-to-noise ratio, S/N. For the low-pass filter, S/N is computed by the following:

$$S / N = V_{ip} / N_{lp},$$

where V_{ip} is the voltage signal and N_{lp} is the high frequency noise signal produced by the low-pass filter in Figures 3.1 and 3.6. Thus, if $V_{ip} = 100$ V and $N_{lp} = 0.1$ V, S/N = 1000.

Sample-and-Hold Circuit

Since it was stated previously that the sample-and-hold circuit must sample input at a rate at least twice the frequency of the input in order to produce the desired output, the error, E_{sh}, in the sample-and-hold circuit of Figures 3.1 and 3.6 can be formulated as follows:

$$E_{sh} = f_{sh} / 2f_i,$$

where f_{sh} is the required sampling frequency and f_i is the desired signal frequency emanating from the input analog voltage in Figure 3.1. An error arises if $E_{sh} < 0.5$ (i.e., $f_{sh} < f_i$).

Summary of Conversion Errors

Since there are a variety of errors associated with A/D and D/A conversions, these errors are summarized in Table 3.1 in order to identify the key relationships that lead to error occurrence. Now, examining Table 3.1, the key findings concerning error analysis are the following:

To achieve an optimal trade-off between quantizing error and cost, use 12 binary encoding bits even though only 7 bits are required.

To minimize voltage sensor error, the sensor should produce an output change-to-input change ratio = 1.

To minimize operational amplifier error, ensure that the output-to-input ratio, $\Delta V_{oa}/\Delta V_{out}$, is equal to the amplification factor A.

To minimize low-pass filter error, maximize the S/N for given values of analog signal voltage (i.e., minimize noise signal).

To prevent sample-and-hold circuit error, ensure that the circuit can sample at a frequency f_{sh} > desired frequency f_i.

CHAPTER SUMMARY

The reader has been introduced to important concepts about devices that interconnect with digital computers—the A/D converter and the D/A converter. This objective has been achieved by considering these signal conversion circuits as a single integrated system, using a smart electric meter system as an example. Circuit diagrams were developed illustrating various facets of conversion logic. Extensive error analysis was performed on all converter circuit

Table 3.1 Summary of Conversion Errors

A/D conversion	D/A conversion	Figure(s)	Optimal number of binary encoding bits	Key relationship
Quantizing error $R/2^n$		Figure 3.3	12	Accuracy cost–benefit trade-off
$2^n/R$	Quantizing error		12	Accuracy cost–benefit trade-off
Rate of change of conversion error		Figure 3.4	12	Accuracy cost–benefit trade-off
Voltage sensor error	Voltage sensor error	Figures 3.1, 3.6, and 3.7		$E_s = \frac{\Delta V_{out}}{\Delta V_{in}} = 1$
Operational amplifier error	Operational amplifier error	Figures 3.1 and 3.6		$E_{oa} = A - \frac{\Delta V_{oa}}{\Delta V_{out}}$ $A = 11.22$
Low-pass filter	Low-pass filter	Figures 3.1 and 3.6		$S/N = V_{ip}/N_{lp}$
Sample-and-hold circuit	Sample-and-hold circuit	Figures 3.1 and 3.6		$E_{sh} = f_{sh}/2\ f_i$ Error if $f_{sh} < f_i$

components in order to identify the best circuit performance values consistent with achieving cost-effective systems.

Reader Problem: You have learned that the number of bits n required to digitally encode an analog signal with a range R is related by the equation $R = 2^n$. Suppose the range is to be R = 120 V, what is the minimum number of bits required to encode this signal?

Solution: $\log_{10}R = n\log_{10}2$, $n = \log_{10}R/\log_{10}2 = \log_{10}120/\log_{10}2 = 2.079/0.301 = 6.91$ (7 bits rounded up).

REFERENCES

[GAN08] J. Ganssle (ed.), *Embedded Systems: World Class Designs*. Amsterdam: Elsevier, 2008.

[RAF05] M. Rafiquzzaman, *Fundamentals of Digital Logic and Microcomputer Design*. New York: Wiley-Interscience, 2005.

Part Two

Network Engineering

Chapter 4

Integrated Software and Real-Time System Design with Applications

Approaches for designing real-time software and hardware on an integrated basis are presented. By "integrated" it is meant that the interaction of software and hardware *during* program execution is addressed in the system design. For example, software outputs of the executable system that are fed to the hardware subsystem are represented in the software and hardware designs. Another aspect of this design approach is, first, to develop the real-time system generic design of a particular artifact, such as a state diagram, and then to use the generic design to guide the development of the application-specific design. An elevator system is used as the design example because it has interesting properties such as interruptible floor traversal sequences. The series of design representations starts with generic and application-specific system-level functions and ends with integrated testing and performance evaluations. An important aspect of the integrated design approach is that exclusive use of abstract representations is unwise because it is important to consider the physical properties of the real-world system, such as elevator floor travel sequences. Without this perspective, critical aspects of real-time system operations such as elevator direction of travel may be overlooked. Several metrics of real-time system performance are modeled and evaluated.

INTRODUCTION

Having learned the fundamentals of computer design, both digital and analog, in previous chapters, you are prepared to learn a very important application area: real-time systems. Real-time control hardware and software has been applied to a wide variety of real-world systems for diverse military, aerospace, industrial, medical,

Computer, Network, Software, and Hardware Engineering with Applications, First Edition. Norman F. Schneidewind.
© 2012 the Institute of Electrical and Electronics Engineers, Inc. Published 2012 by John Wiley & Sons, Inc.

and civil applications. Most real-time systems are comprised of heterogeneous components including sensors, microprocessors, and actuators. These components intensively interact with each other and with their environments. Thus, there are many dynamic and uncertain factors in these systems. Such a system needs to satisfy all the functional requirements and timeliness demands. In real-time systems, system resources such as microprocessor cycles, communication bandwidth, and storage memory are restricted, and thus efficient resource allocation in different operational scenarios is required. As a result, the design of complex real-time systems is quite challenging and is distinct from the conventional non-real-time design [WAN08]. As real-time computer systems become larger and more complex, so their analysis becomes increasingly difficult. Much of the skill in developing these systems lies in choosing the most appropriate theories and tools for different stages of development and different aspects of the system [CAD98]. The approach in this chapter is to use models, such as state diagrams, simulated testing, and event sequencing (e.g., elevator floor traversal sequences) that are appropriate for real-time system analysis.

Objectives

While there are many worthy papers addressed to single aspects of real-time design, such as scheduling [GUP10], there is an absence of an integrated approach. Thus, the aim of this chapter is to develop an integrated and comprehensive design approach with the objective of providing engineers with a road map for improving real-time system design. According to Wang et al. [WAN04], model-based software development has been shown to be a promising approach to real-time design problems. In this approach, the software is first modeled abstractly without considering its execution platform, and then transformed to a software design model on the target platform. However, as mentioned in the abstract, this approach should not be carried too far because if the abstract model is divorced from reality, it will do a poor job of representing the real-world system.

Design Challenges

Today, many computer systems are being used to measure and control real-world processes. The execution of these systems and their control programs is therefore bound to timing constraints imposed by the real-world process [PLA84; SID06]. Thus, timing constraints are addressed in analyzing real-time system performance.

Unfortunately, real-time software is particularly difficult to design. In addition to ever more complex functional requirements, real-time software has to satisfy a set of stringent nonfunctional requirements, such as maximum permissible response time (e.g., maximum elevator system response time) and throughput (e.g., elevator system passenger throughput). Often, the inability of real-time software to meet its primary nonfunctional requirements becomes apparent only in the later stages of

development. When this happens, the design may have to be heavily and hurriedly modified, even if all the functional requirements are satisfied, resulting in cost and schedule overruns as well as unreliable and unmaintainable code. This unhappy situation is primarily due to the common practice of postponing all consideration of so-called platform issues until the application logic of the software has been satisfactorily designed. Although "platform-independent design" is a good idea in principle, because it allows separation of application concerns and implementation, it is often carried to extremes. In particular, it is dangerous in situations where the physical characteristics of the platform (e.g., elevator system floor traversal control) can have a fundamental impact on the application logic (e.g., elevator system floor traversal control software design) [SEL03].Therefore, because an abstract representation of our design would be of limited value, the abstract analysis is illustrated with an elevator system. I chose the elevator example because it presents many design challenges and everyone can relate to this system. Recognize that an abstract approach can only be applied for marrying software and hardware design. When testing and performance evaluation are performed, the particular characteristics of the application must be considered. For example, the response time to elevator floor requests must be evaluated through simulated performance testing.

Steps in Real-Time System Design

Real-time system design can be accomplished by the following steps [KOY90; OST98]:

Elicit and document the service requirements in terms of the environment (e.g., elevator response time requirement geared to the number of user floor requests during a specified time).

Using the environment-based service requirements, specify the system controller characteristics (e.g., specify elevator controller properties for managing efficient floor traversal scenarios).

Based on the controller specifications, develop software and hardware designs that achieve system requirements and correct interaction among system components (e.g., develop elevator system *integrated* software and hardware designs that achieve response time requirements and correct interplay between elevator system floor request control and motion control).

Apply the rule of considering real-world operational details during abstract design by using a mix of abstraction and operational detail views. For example, observing how an elevator operates (e.g., processing service requests) provides insight into how real-time systems must function in general. That is, if you observe how an elevator control organizes operations in order to service as many floors as possible in minimum time, in a given traversal (i.e., using interrupts to develop an optimal schedule), you can apply this observation to designing schedules for real-time systems.

REAL-TIME SYSTEM PROPERTIES

Execution Time

Some researchers consider periodic real-time independent tasks with known periods and worst-case execution times in their design approach [GUP10]. This view is quite restrictive because it would be unusual for a real-time system to have "known periods and worst-case execution times." The more representative situation is periods of operation and execution times that are driven asynchronously by inputs that occur at unpredictable times, and, hence, produce unknown execution times (e.g., elevator system).

Implementation Elements

The reduced instruction set computing (RISC) architecture requires several operations to execute a single instruction. However, this design provides high speed; for example, it is well suited to real-time applications that must meet deadlines, but at the expense of relatively complex programming.

Objects. The objects in the elevator system are user, system controller, system storage, operations, and error control, as shown in Figure 4.1.

Asynchronous Circuits. Due to the unpredictable nature of inputs and operations in real-time systems, hardware design is accomplished with asynchronous circuits

Performance

The response time is the difference in time between completion of request and initiation of request (e.g., difference in time between elevator reaching designated floor and user pushing the Up or Down button).

Operations that must meet deadlines (e.g., elevator travel satisfies response time requirement).

Operations schedule (e.g., elevator schedule maximizes number of floors traversed in traveling from current floor to most distant floor).

Control Functions

The following control functions are shown in Figure 4.1:

Interruptible sequence of operations causing interrupts to be processed out of sequence (e.g., changing directions of elevator floor travel sequences).

Multiple threads of control caused by concurrent inputs (e.g., multiple *concurrent* elevator floor requests) [MOO02].

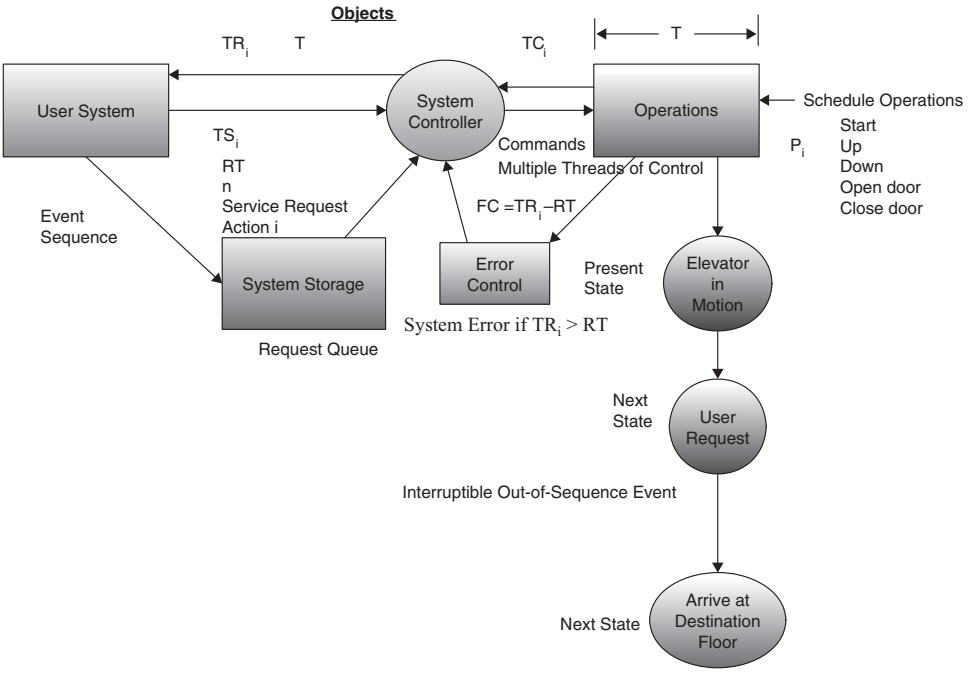


Figure 4.1 Generic real-time system design. TS_i, time of service request i; RT, required response time; $TR_i = TS_i - TC_i$, response time of service request i; TC_i, time of completion of service request i; P_i, probability of completing service request i in required response time; n, number of services request in operational time T; $FC = TR_i - RT$, feedback correction; T, duration of service operations.

Control commands are issued, for example, by System Controller to Operations.

Feedback control is the response time error is fed back from the Error Control to Controller.

System queues are used to store backlog of user requests (e.g., queues of elevator floor requests) [MOO02].

Design Levels

In Harris and Harris [HAR07], advice is offered regarding using design levels to accomplish system design as follows:

Hierarchy. Divide system into modules that are easer to understand than the complete system.

Modularity. Produce modules that have well-defined functions and interfaces that can easily interconnect.

Regularity. Find modules with common functions (i.e., interchangeable parts).

This approach may be satisfactory for a general approach to design but does not completely satisfy real-time requirements. Instead, it is suggested that real-time system hierarchies are rare or nonexistent. Real-time module topology is essentially flat. For example, the user system and system controller in Figure 4.1 interact on the same level. Real-time systems can have well-defined functions but interfaces may not easily interconnect because inputs may arrive at unpredictable times, making it difficult for the system controller to respond in a timely manner. For example, in Figure 4.1, user system requests must be queued because the system controller is unable to respond to all requests immediately. Lastly, real-time systems are one-of-a-kind; they are not mass produced. Therefore, common functions with interchangeable parts do not hold. These peculiarities of real-time systems will be recognized when developing the design approach. The synopsis of requirements postulated at three levels is listed below for both generic and application-specific cases.

Real-Time System Requirements

System Level

Generic

Response time: variable response time, mean response time, maximum response time, minimum response time, throughput

Application Specific

Response time: time between elevator floor request and arrival at destination floor and mean value of these times

Throughput: number of floor requests processed per elevator operational time

Software Level

Generic

Map between system level requirements and software routines (see Fig. 4.8)

Application Specific

For example, routines for optimally sequencing elevator floor requests.

Hardware Level

Generic

Map between system-level requirements and hardware components and between hardware components and software routines (see Fig. 4.4)

Application Specific

For example, input/output (I/O) channels must have a sufficient transfer rate to satisfy elevator system response time requirements

DESIGN PROCESS ELEMENTS

It is important to have a close relationship between the user system and the system control functions [BOA77], as demonstrated by the generic design process in Figure 4.1. To implement this approach, apply the following elements of the design process that are listed below for both generic and application-specific (elevator) cases.

Event sequence: series of state transitions

Elevator responds to sequence of floor push-button events

Interruptible event sequence causing state transition

Sequence of elevator floor traversals is modified to service as many requests possible in a given floor traversal

Time of service request

Time when the following occur: user pushes Up button, user pushes Down button

States and state transitions

User pushes Up floor button at request floor → elevator goes up or down or is at request floor → door opens → door closes → elevator goes up to destination floor

User pushes Down floor button at request floor → elevator goes up or down or is at request floor → elevator stops → door opens → door closes → elevator goes down to destination floor

Controls

Down travel control, up travel control, start control, stop control, door open control, door close control

System storage

Elevator event sequence storage requirements: present event, next event, present state, next state, next state transition, and storage capacity necessary for effective communication among software and hardware components [BAG97].

Interrupts

User floor request while the elevator is in motion

INTEGRATED SOFTWARE–HARDWARE DESIGN

Putting software and hardware design in separate bins is a big mistake because the operations of software and hardware are intimately related. For example, in interrupt processing, an interrupt signal generated by hardware triggers software interrupt processing routines. Thus, when designing systems, processing a requirement should be considered as a resource allocation problem. For example, in an elevator system, the signals generated by pushing buttons for floor requests are allocated to electronic

circuitry. These signals are fed to software routines for determining the direction and distance the elevator must travel to service requests.

A problem in system design is the appropriate allocation of functions between software and hardware design [AYA02]. Resolve this problem by allocating logic functions, such as identifying the elevator floor travel sequences in the architectural design of Figure 4.4 and allocating the resulting control functions of elevator control (Up, Down, and Open and Close doors) to the hardware design in Figure 4.5.

Time-Driven versus Event-Driven Software Design Styles

Time-driven software design style corresponds to using cyclic activities triggered by time. This software style is naturally suited for the implementation of periodic activities, such as software implementation of control loop behavior in embedded control systems. In contrast, software written in the event-driven style typically waits for an event to occur, and then reacts to it by making an appropriate decision or computation, and then enters a dormant state waiting for the next event [SEL96]. Elevator systems are event-driven (i.e., elevator controller responds to floor request event). Therefore, elevator controller software must be designed to develop a floor traversal schedule when requests arrive. However, in doing so, elevator controller software must be designed to achieve floor request response time requirements.

In contrast to time-driven software style, the event-driven software style has evolved largely to deal with the complexity arising from asynchrony, concurrency, and the inherent nondeterminism due to the two. The system must respond to asynchronous events in the external world, and the reaction must depend on the system state [SAK98]. Thus, elevator controller software must be responsive to user service requests that will occur *asynchronously* (floor push-button operations) and may occur *concurrently* (push-button operations occurring on different floors at the same time).

In the following integrated software–hardware design methodology, states and state transitions form the core processes, because the real-time environment is one of rapidly changing conditions, and state diagrams are effective for representing this environment. In addition, when the detailed software logic is developed for the elevator application, flowcharts are used because they are useful for portraying decision logic, which is endemic to this application. The flowchart is driven by the state diagram transitions. Both the generic and application-specific software state diagrams are shown in Figure 4.3. The generic software design is shown in Figure 4.8.

SOFTWARE FUNCTIONS

Input Processing State

Generic System

Input request i and service request time TS_i in Figure 4.8.

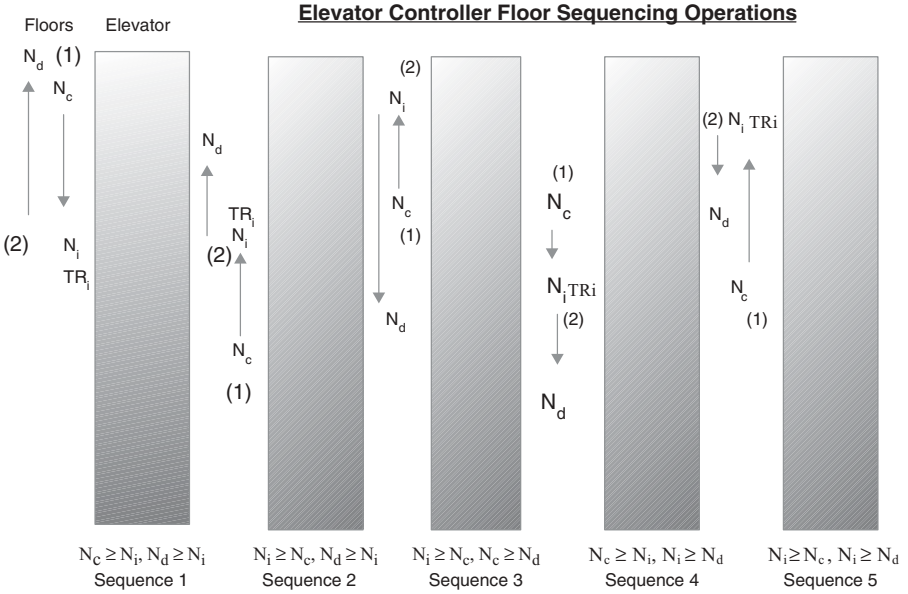


Figure 4.2 Elevator system. N_i, request floor; N_d, destination floor; N_c, current floor; TR_i, response time of request i.

Elevator System

Elevator floor sequencing controller identifies and processes floor requests in Figure 4.2.

Decision Analysis State

Generic System

Identify sequence of service requests i and i + 1 based on its priority PR_i and priority PR_{i+1}, respectively, and process them in this order in Figure 4.8.

Elevator System

Sequences elevator travel so that throughput is maximized and response time T_i is minimized, as shown in Figure 4.2. In contrast to the generic system, there is no priority in the elevator system; all floor requests are treated equally.

Computation State

Generic System

Compute response time TR_i for service request i, mean response time TR, and throughput for all service requests TP in Figure 4.8.

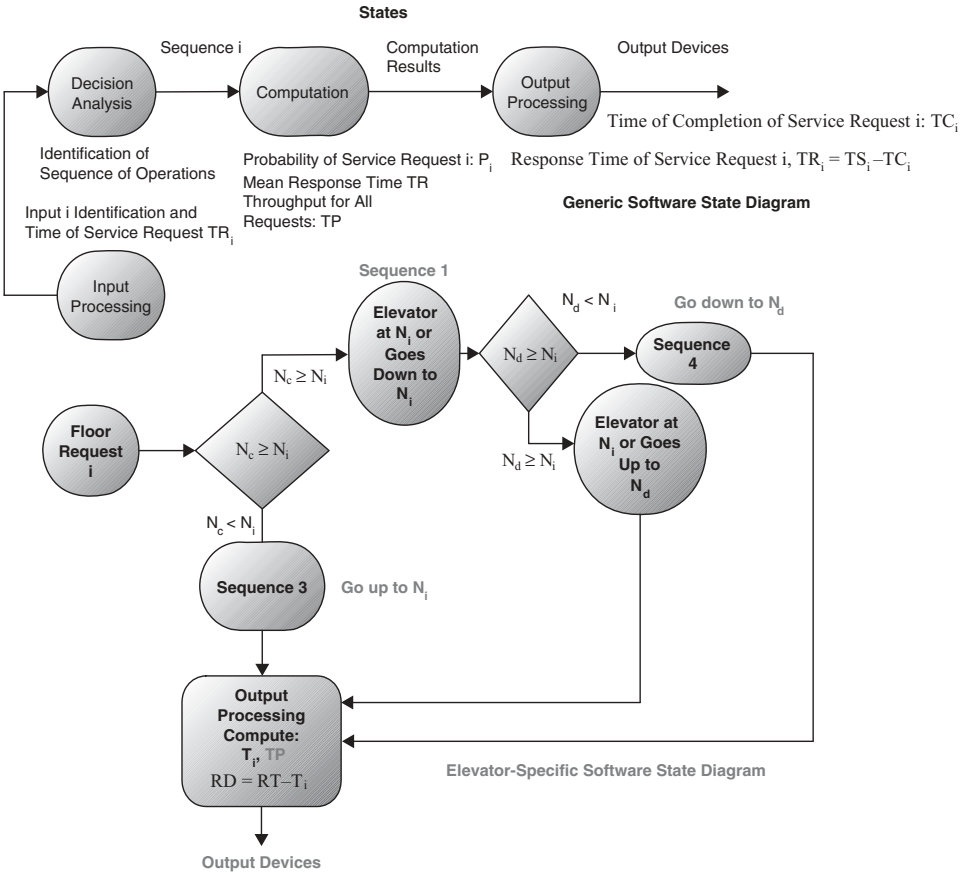


Figure 4.3 State diagrams. N_i, nearest request floor; N_c, current floor; N_d, destination floor associated with N_i; T_i: response time of request i; RD, response time difference with respect to required RT; TP, throughput.

Elevator System

Computations of response time metrics as the result of state transitions are based on comparison of floor locations in Figure 4.3.

Output Processing State

Generic System

Transfer results of decision analysis and computations to output devices in Figure 4.3.

Elevator System

Transfer results of computing performance metrics to output devices in Figure 4.3.

HARDWARE FUNCTIONS

Generic System

Microprocessor with sufficient speed (clock rate) to satisfy the response time requirement. If this requirement is not satisfied, increase the microprocessor speed in Figure 4.4.

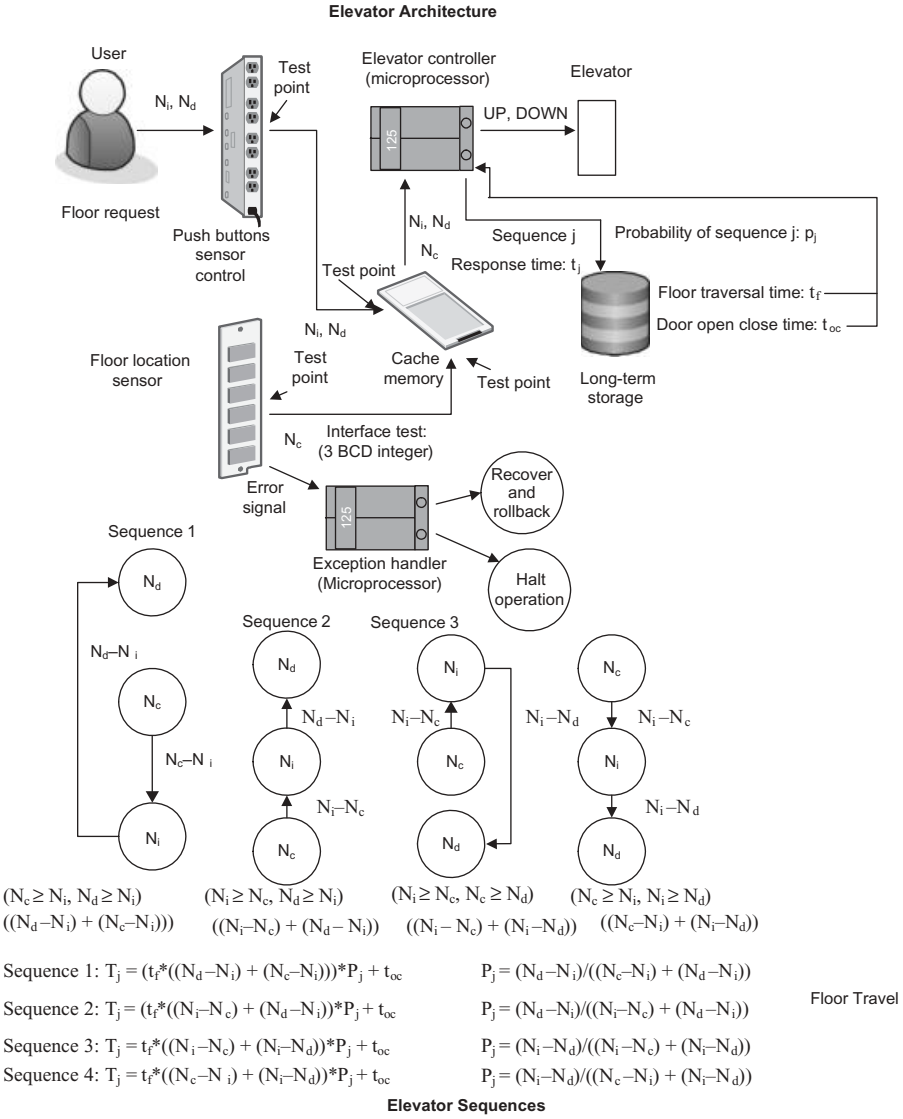


Figure 4.4 Elevator system architecture. N_i, request floor location; T_j, sequence j response time; T_f, single floor traversal time; N_c, current floor location; P_j, probability of sequence j; N_d, destination floor location; t_{oc}: door open/close time; BCD, binary-coded decimal.

Elevator System

Elevator floor sequencing controller with sufficient speed to satisfy floor request response time requirement. If this requirement is not satisfied, increase controller speed in Figure 4.4.

Generic System

I/O channels with sufficient transfer rate to keep up with real-time transaction input rate.

Elevator System

Elevator I/O channels with sufficient transfer rate to satisfy floor request response time and mean value and throughput requirements.

Generic System

Storage system with sufficient capacity to support the input, storage, and output of real-time transactions. Real-time systems do not have the luxury of inputting data when convenient for the microprocessor. These systems must input data as it arrives, with no loss of input, in Figure 4.4.

Elevator System

Many real-time designs impose hard real-time constraints on tasks. Thus, computing an upper bound of execution time of the software (e.g., maximum floor traversal time) is a critically important but difficult task. The difficulty arises particularly when the code is executed on processors with cache-based memory systems, which may be limited in capacity [UM03]. Therefore, the elevator cache must have sufficient capacity and speed to input and store floor requests, with no loss of floor requests, as shown in Figure 4.4.

Generic System

System bus with sufficient bandwidth to accommodate expected data transfer requirements, as shown in Figure 4.5.

Elevator System

Elevator system bus with sufficient bandwidth to achieve floor request response time, mean response time, and throughput requirements, as shown in Figure 4.6.

ELEVATOR SOFTWARE DESIGN

The purpose of the elevator-specific software design is to identify the floor travel sequences by comparing the values of the present floor location (N_c), request floor

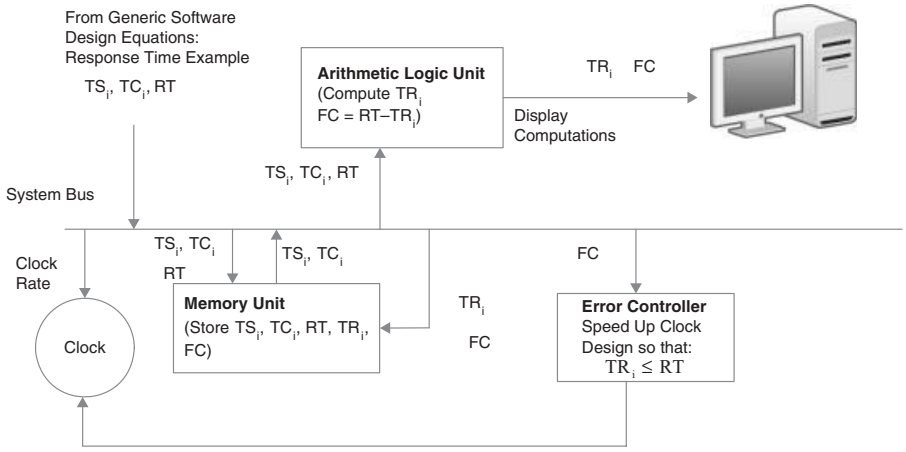


Figure 4.5 Generic hardware design and display. TS_i, time of service request i; TC_i, service request i completion time; TR_i, service request i response time; RT, required response time; FC, error control (used if $TR_i > RT$).

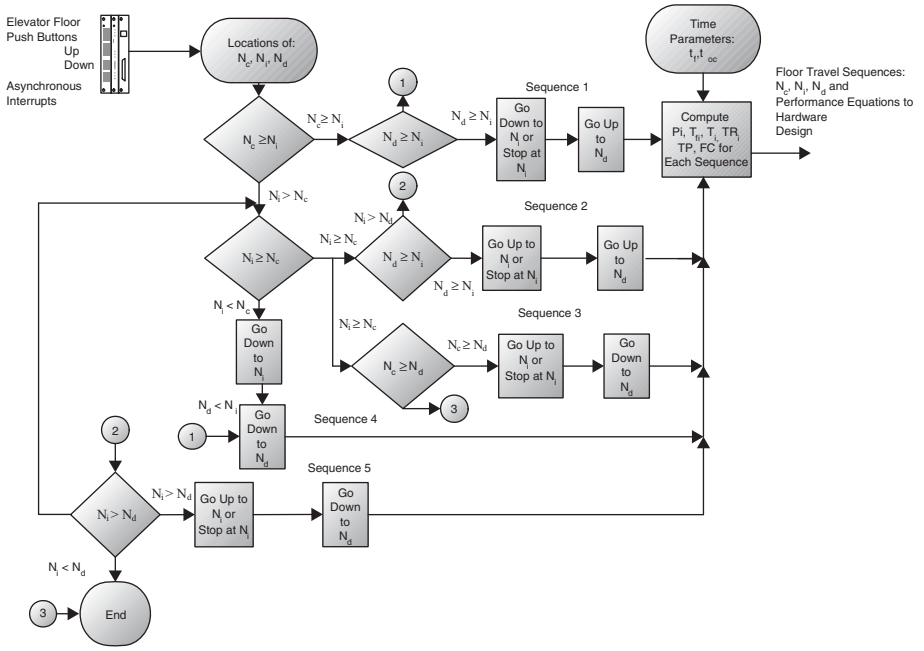


Figure 4.6 Elevator system software design. P_i, probability of completing service request i in required response time; T_{fi}, expected time of traversing all floors to respond a request i; T_i, expected time of traversing all floors to respond a request i plus opening and closing doors; TP, throughput; FC, response time feedback correction; t_f, time of traversing one floor; t_{oc}, time of opening and closing doors.

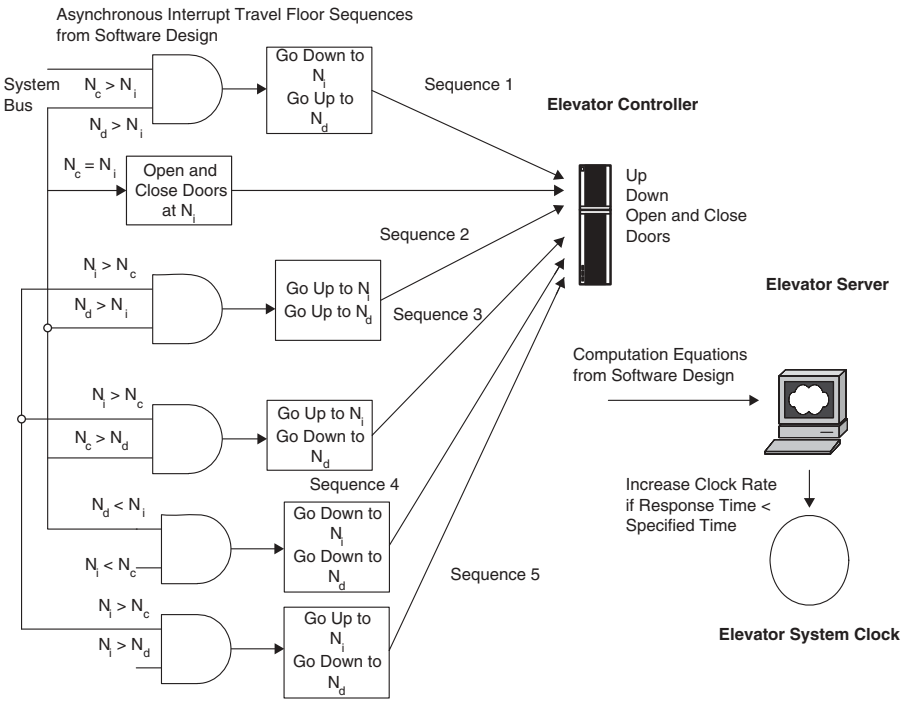


Figure 4.7 Elevator system hardware design. N_c, current floor; N_i, request floor; N_d, destination floor.

location (N_i), and the destination floor location (N_d), which are inputted by the elevator floor push buttons in Figure 4.6. In addition, the software design formulates the performance equations that are transferred to the hardware design in Figure 4.7 for implementation. According to Mok [MOK90], in real-time programs, the time of occurrence of events rather than the order of events is crucial in determining the outcome of a computation. However, both event order and time of occurrence are crucial in determining *system performance*. For example, the order of elevator floor traversals *and* their traversal times are important determinants of elevator system performance and must be included in the software design, as shown in Figure 4.6.

Selected Hardware Designs

Critical design functions are developed for both the generic and elevator systems. The purpose is to demonstrate how an integrated software–hardware design is achieved by mapping between software and hardware designs. Hardware-oriented design has to deal with more problems than software-based design, especially the progression of time [LU03], such as manipulating the clock rate to achieve the required response time in an elevator system, as shown in Figure 4.7. Therefore,

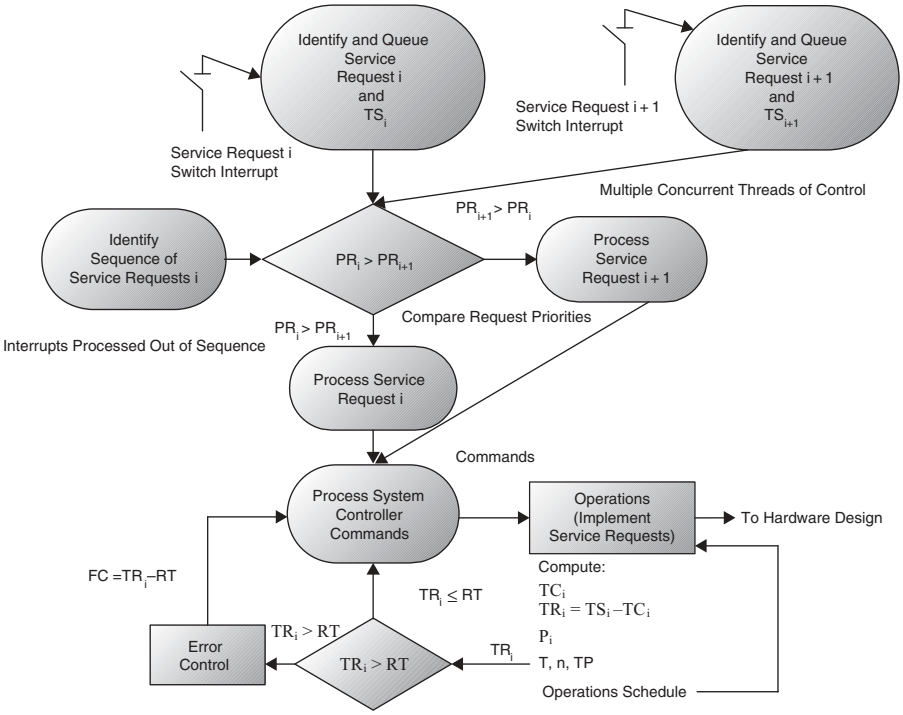


Figure 4.8 Generic software design. TS_i, time of service request i; TC_i, service request i completion time; TR_i, service request i response time; P_i, probability of service request i; RT, required response time; FC, error control; T, duration of service operations; n, number of responses required in operation time T; TP, throughput; PR_i, priority of request i.

the emphasis in the design process is on hardware design, but not neglecting the mapping between hardware and software designs.

Generic System

Develop control logic for decoding (i.e., identifying) input service requests in Figure 4.9 in order to demonstrate the mapping between generic software and hardware designs in Figures 4.8 and 4.5, respectively, where Figure 4.9 provides the decoder logic for generating service request interrupts. Table 4.1 shows the truth table corresponding to the design logic in Figure 4.9, where the bolded **1**s in the table correspond to the decoder outputs. Then, a second critical hardware function is designed—response time computation and display—if the response time requirement is not satisfied in Figure 4.7.

Elevator-Specific System

The elevator controller in the hardware design (Fig. 4.7) accepts the elevator floor sequences from the software design (Fig. 4.6) and uses digital logic to translate the

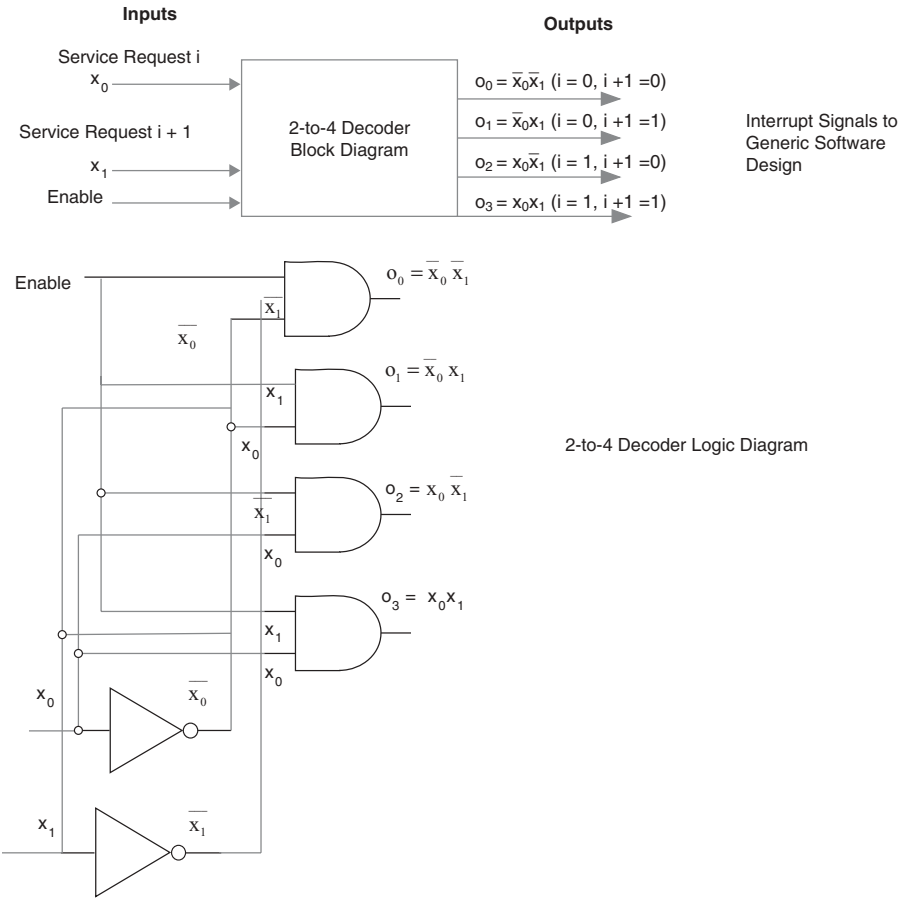


Figure 4.9 Input request decoder design.

Table 4.1 Decoder Truth Table for Two Inputs (Request i and Request i + 1) and Four Outputs (Request Interrupt Signals)

Inputs			Outputs			
E (Enable)	x_1 request (i + 1)	x_0 (request i)	d_3	d_2	d_1	d_0
1	0	0	**1**	0	0	0
1	0	1	0	**1**	0	0
1	1	0	0	0	**1**	0
1	1	1	0	0	0	**1**

sequences into elevator control commands: Up, Down, and Open and Close Doors. In addition, the elevator system is equipped with a server that implements the performance equations delivered by the software design. One of these equations is the response time error control function. In the event that a response time deficiency exists, the server uses the error control function to increase the clock rate. The increased clock rate, in turn, allows the floor travel time to be reduced to satisfy the response time requirement.

PERFORMANCE EQUATIONS DEVELOPMENT

Now define the variables, parameters, event sequences, and performance equations that are used in evaluating the performance of both generic and application-specific systems.

Definitions

Generic System

Time of service request i: TS_i

Time of completion of service request i: TC_i

Required response time: RT

Operational time: T

Response time of service request i, $TR_i = TS_i - TC_i$

Maximum response time service request i: TR_i (max)

Minimum response time service request i: TR_i (min)

Probability of completing service request i in required response time: P_i

System error if $TR_i > RT$

System error feedback correction: $FC = TR_i - RT$

Number of responses to service requests required in operational time T: n

Elevator-Specific System

The sequence of elevator system operations is complex with respect to the number and type of floor requests and the number of stops—with door openings and closings—over a given operational time. Therefore, this complexity is decomposed so that request floor N_i is considered to be the nearest request floor to the current elevator floor N_c, and destination floor N_d is considered to be the destination floor corresponding to N_i in a given elevator traversal. This formulation is reflected in the list below. By considering the traversals in Figure 4.2, the decomposition covers the possible floor traversal sequences.

Estimated by Uniformly Distributed Random Variable (Using Excel RAND Function)

Probability of floor request i: P_i

Request i floor (floor that is nearest to current location N_c of elevator): N_i

Current floor (current location of elevator): N_c

Destination floor associated with request i floor: N_d

Specified

Time of traversing one floor: t_f

Time of opening and closing doors: t_{oc} (these times are used at request floor N_i and again at destination floor N_d)

Number of floor requests processed in time T: n

Computed

Expected time of traversing all floors to respond to a request i: T_{fi}

Elevator response time = expected time of traversing all floors to respond to request i plus time of opening and closing doors: T_i

Maximum response time for service request i: T_i (max)

Minimum response time for service request i: T_i (min)

Required

Elevator response time: RT

Response time difference: $RD = RT - T_i$

Mean response time difference (MRD), computed over n floor requests

System error if $T_i > RT$

System error feedback correction: $FC = T_i - RT$

Elevator operational time: T

Event Sequences

Event sequences are generated in Figure 4.2 based on the number of distinct combinations of floor locations (N_i, N_c, N_d) and their travel directions. Note in the event sequences that if the elevator is already at the request floor ($N_c = N_i$), there is zero travel time from N_c to N_i. Also note, in Figure 4.2, that the relative locations of the elevator, the request floor, and the destination floor, are important in computing the elevator travel distances in the event sequences.

Sequence 1

(1) Elevator goes *down* from current floor N_c to request floor N_i, then (2) goes *up* from request floor N_i to destination floor N_d ($N_c \geq N_i$, $N_d \geq N_i$):

$$P_i = (N_d - N_i) / ((N_c - N_i) + (N_d - N_i)),$$
$$T_{fi} = t_f * ((N_d - N_i) + (N_c - N_i)) * P_i,$$
$$T_i = t_f * ((N_d - N_i) + (N_c - N_i)) * P_i + t_{oc}.$$

Sequence 2

(1) Elevator goes *up* from current floor N_c to request floor N_i, then (2) goes *up* from request floor N_i to destination floor N_d ($N_i \geq N_c$, $N_d \geq N_i$):

$$P_i = (N_d - N_i) / ((N_i - N_c) + (N_d - N_i)),$$
$$T_{fi} = t_f * ((N_i - N_c) + (N_d - N_i)) * P_i,$$
$$T_i = t_f * ((N_i - N_c) + (N_d - N_i)) * P_i + t_{oc}.$$

Sequence 3

(1) Elevator goes *up* from current floor N_c to request floor N_i, then (2) goes *down* from request N_i to destination floor N_d ($N_i \geq N_c$, $N_c \geq N_d$):

$$P_i = (N_i - N_d) / ((N_i - N_c) + (N_c - N_d)),$$
$$T_{fi} = t_f * ((N_i - N_c) + (N_c - N_d)) * P_i,$$
$$T_i = t_f * ((N_i - N_c) + (N_c - N_d)) * P_i + t_{oc}.$$

Sequence 4

(1) Elevator goes *down* from current floor N_c to request floor N_i, then (2) goes *down* from request floor N_i to destination floor N_d ($N_c \geq N_i$, $N_i \geq N_d$):

$$P_i = (N_i - N_d) / ((N_c - N_i) + (N_i - N_d)),$$
$$T_{fi} = t_f * ((N_c - N_i) + (N_i - N_d)) * P_i,$$
$$T_i = t_f * ((N_c - N_i) + (N_i - N_d)) * P_i + t_{oc}.$$

Sequence 5

(1) Elevator goes *up* from current floor N_c to request floor N_i, then (2) goes *down* from request floor N_i to destination floor N_d ($N_i \geq N_c$, $N_i \geq N_d$):

$$P_i = (N_i - N_d) / ((N_i - N_c) + (N_i - N_d)),$$
$$T_{fi} = t_f * ((N_i - N_c) + (N_i - N_d)) * P_i,$$
$$T_i = t_f * ((N_i - N_c) + (N_i - N_d)) * P_i + t_{oc}.$$

System Performance Equations

Both generic and elevator-specific performance equations are shown below. Later, these equations will be used to evaluate elevator system performance and to design tests of simulated performance.

Generic

Expected (mean) system response time, computed over n responses to service requests, accounting for probability of occurrence of response time:

$$TR = \frac{\sum_{i=1}^{n} (P_i)(TR_i)}{n}.$$

Elevator Specific

Mean time for elevator controller to service floor requests, accounting for probability of occurrence of response time:

$$TR = \frac{\sum_{i=1}^{n} (P_i)(T_i)}{n}.$$

Generic

Total expected operational time, accounting for the probability of occurrence of response time:

$$T = \sum_{i=1}^{n} (P_i)(TR_i).$$

Elevator Specific

Total expected elevator operational time over n operations, accounting for the probability of occurrence of response time:

$$T = \sum_{i=1}^{n} (P_i)(T_i).$$

Generic

Throughput (TP) = number of operations/operational time = n/T.

Elevator Specific

Number of service floor requests n processed by elevator controller during operational time T:

$$TP = n / T.$$

Generic

MRD for request i:

$$\text{MRD} = \frac{\sum_{i=1}^{n} \text{RD}_i}{n}.$$

Elevator Specific

Mean elevator response time difference for floor request i:

$$\text{MRD} = \frac{\sum_{i=1}^{n} \text{RD}_i}{n}.$$

REAL-TIME SYSTEM SIMULATED TESTING

This section is comprised of observations by other researchers of problems in real-time system testing and our responses to these problems.

Achieving Visibility of Operations

As complex devices such as elevator controllers are inserted into real-time systems, traditional testing methods may be inadequate. A difficult obstacle to thorough testing of real-time systems is achieving visibility into the operations of processing elements, such as the elevator server of the hardware design in Figure 4.7, while application software is executing, such as floor sequencing control, in the software design of Figure 4.6 [KIN98]. Resolve this problem by explicitly mapping Figure 4.6 computations into computation execution in Figure 4.7, and test the interaction in terms of performance results, as discussed in the next section.

Test Case Selection

Test case selection is designed to provide adequate coverage of system components by deriving test cases from software designs [EN08]; for example, the elevator software design in Figure 4.6. Test case selection is effective when software functions are mapped to test cases, such as floor sequence traversal sequences mapped to tests of sequence correctness. Test case selection can also be enhanced by using state diagrams to identify state transitions that must be tested [SHU04], such as the elevator travel state changes in Figure 4.3.

Verifying a Design

It has been noted that the application of simulation to verifying a design does not provide "total confidence" that the design is correct [UMR83]. Actually, no verification

method can provide "total confidence," but by replicating simulation tests a sufficiently large number of times, say 100 elevator floor traversals, verification error can be minimized.

Achieving Realism in Testing

The testing regimen approximates realism by mimicking the way the actual elevator system would perform with respect to floor traversal scenarios [ZHE04]. The testing of the elevator system is geared to the performance simulation results to be presented in the next section. The objective is to ascertain whether performance objectives such as required response time can be met. This is accomplished by simulating a specified number of floor requests that generate a series of elevator travel sequences. Based on these sequences, performance metrics are computed and compared with a specified performance. If there is a performance error, the test is repeated using a reduced floor traversal time, consistent with achievable performance of extant elevator systems. A key indicator of acceptable performance is that response time is satisfied for all floor requests.

Detecting Logical Errors

The characteristics of real-time systems impose specific requirements on the test system. The system must be capable of detecting logical as well as timing errors in the design [TIM93], for example the ability to detect incorrect elevator floor sequences (e.g., elevator goes up to the highest destination floor and attempts to go higher) and the ability to detect incorrect elevator timing computations (e.g., response time is computed to be negative).

Maximum Response Time Criterion

Maximum response time that occurs due to resource limitation [WED91], such as maximum elevator response time caused by the elevator not being available in a timely manner, is another important test criterion. An example of the test of this variable is shown in Figure 4.10 for elevator travel Sequence 1. Based on the test results, floor travel time would be reduced to 3 seconds in Figure 4.10 to obtain the required maximum response time of 60 seconds.

Complexity Caused by Interrupts

Another consideration in real-time testing is complexity caused by interrupts occurring in an asynchronous manner [PET07], such as elevator travel in the down direction being interrupted by a request to go in the up direction. Handle this situation by incorporating asynchronous interrupts into both the elevator software and hard-

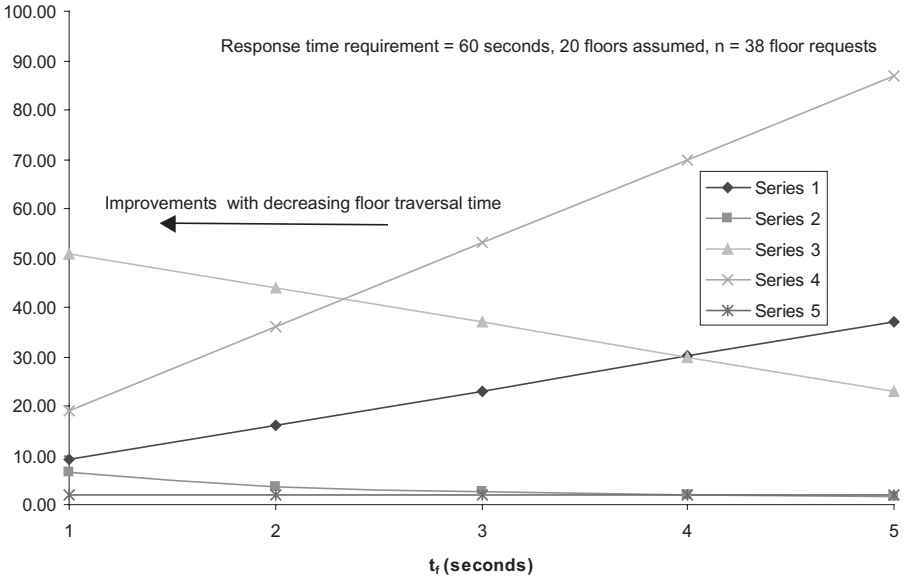


Figure 4.10 Elevator Sequence 1: Mean response time (TR), throughput (TP), MRD, maximum response time T_i (max), and minimum response time T_i (min) versus one elevator floor traversal time (t_f). Series 1: TR (seconds). Series 2: TP (requests per minute). Series 3: MRD, mean difference between required and achieved response time (seconds). Series 4: T_i (max) (seconds). Series 5: T_i (min) (seconds).

ware designs in Figures 4.6 and 4.7, respectively, and conducting performance simulations in this environment.

ELEVATOR SYSTEM PERFORMANCE RESULTS

Elevator system performance results are computed using performance metrics. These metrics are functions of elevator travel sequences and the comparison of sequences. As Figure 4.9 shows, performance improves with decreasing travel time for one floor. These results were generated by simulating testing of travel sequences 100 times. For each sequence, current floor, request floor, and destination floor locations were produced from uniformly generated random numbers, assuming there are 20 floors in the elevator system. Then, the floor location values were compared to produce the travel sequences. Next, using the sequences, various metrics were computed. Then, two sequences (1 and 4) were compared (see Fig. 4.2) to investigate whether there is a difference due to direction of elevator travel, for the same values of travel time for one floor. Indeed, as Figure 4.11 demonstrates, there are notable differences for throughput and mean difference between required and achieved response times. The lesson learned is that travel direction and distance is important in assessing performance.

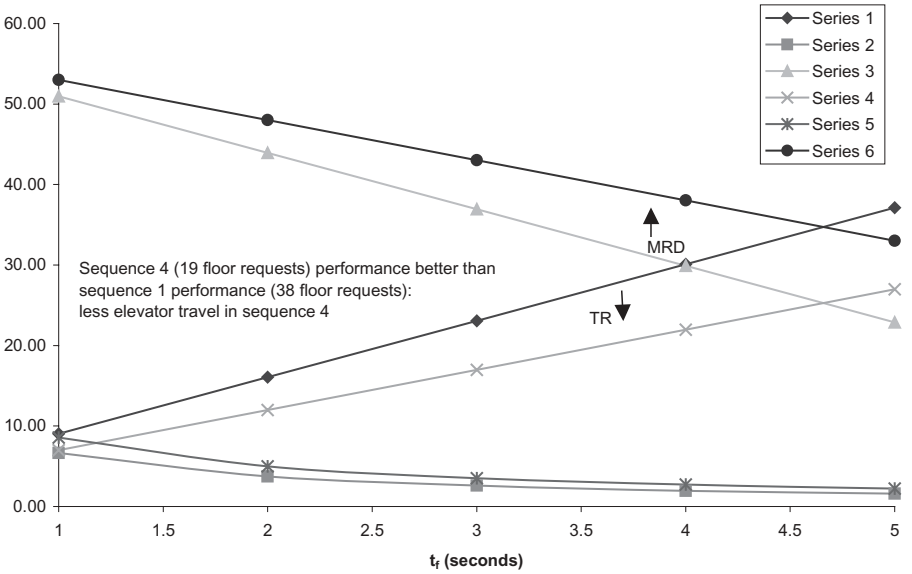


Figure 4.11 Elevator: Mean response time (TR), throughput (TP), and MRD versus one floor travel time t_f. Series 1: TR, Sequence 1. Series 2: TP, Sequence 1. Series 3: MRD, Sequence 1. Series 4: TR, Sequence 4. Series 5: TP, Sequence 4. Series 6: MRD, Sequence 4.

SUMMARY AND CONCLUSION

My aim was to develop an integrated and comprehensive design approach with the objective of providing engineers with a road map for improving real-time system design. My approach to real-time system design was to use models (e.g., elevator floor traversal sequences) that are appropriate for real-time system analysis, such as state diagrams, simulated testing, and event sequencing. Timing and schedule constraints were considered in order to analyze real-time system performance. I chose the elevator example because it presents many design challenges and everyone can relate to this system. I recognize that an abstract approach can only go so far in developing real-time hardware and software designs, and in testing these designs. Ultimately, the particular characteristics of the application must be considered.

By applying the above principles of real-time system design, an application-specific system such as an elevator system can be developed to satisfy response time requirements. The most important step in the development process is first, to represent the generic view of the application design. Then, using the generic design as a guide, develop the specific features of the application. This approach has the advantages of providing real-time system design abstractions that can be used for designing *any* real-time system, and at the same time providing sufficient specificity for designing application-specific systems.

Question for the Reader: In the elevator system design, why not model the complete, continuous scenario of elevator operations rather than dividing the operations into discrete sequences, such as those shown in Figure 4.2?

Answer: While at first glance this may be a reasonable thing to do in order to introduce more realism into the model, this approach would be counterproductive because excessive complexity induced by considering too large a chunk of a system at one time leads to errors in design and, consequently, to errors in the implemented system. The "divide-and-conquer" approach is a superior design paradigm.

REFERENCES

[AYA02] J. L. Ayala, A. G. Lomena, M. Lopez-Vallejo, and A. Fernandez, "Design of a pipelined hardware architecture for real-time neural network computations," *The 2002 45th Midwest Symposium on Circuits and Systems*, August 4–7, 2002, Volume 1, pp. 419–422.

[BAG97] A. Baganne, J. L. Philippe, and E. Martin, "Hardware interface design for real time embedded systems," *Proceedings. Seventh Great Lakes Symposium on VLSI*, March 13–15, 1997, pp. 58–63.

[BOA77] M. Boari, G. Grazia, and A. Bellman, "A methodological approach to the real-time software design and its use in a PCM toll office," *Proceedings of the IEEE*, 1977, 65(9), pp. 1335–1342.

[CAD98] R. Cardell-Oliver and T. Glover, "A framework for the test and verification of real-time systems," *IEE Colloquium on Real-Time Systems (Digest No. 1998/306)*, April 21, 1998, pp. 5/1–5/4.

[EN08] A. En-Nouaary and A. Hamou-Lhadj, "A boundary checking technique for testing real-time systems modeled as timed input output automata," *The Eighth International Conference on Quality Software*, August 12–13, 2008, pp. 209–215.

[GUP10] N. Gupta, S. K. Mandal, J. Malave, A. Mandal, and R. N. Mahapatra, "A hardware scheduler for real time multiprocessor system on chip," *23rd International Conference on VLSI Design*, January 3–7, 2010, pp. 264–269.

[HAR07] D. M. Harris and S. L. Harris, *Digital Design and Computer Architecture*. New York: Elsevier, 2007.

[KIN98] G. Walters E. King, R. Kessinger, and R. Fryer, "Processor design and implementation for real-time testing of embedded systems," *Proceedings of the 17th AIAA/IEEE/SAE Digital Avionics Systems Conference*, 1998, Volume 1, pp. B44/1–B44/8.

[KOY90] M. Koyamada and D. Iwado, "A stepwise approach to behavior design for real-time software," *Proceedings of the First International Conference on Systems Integration*, April 23–26, 1990, pp. 789–796.

[LU03] L. Bin, A. Monti, and R. A. Dougal, "Real-time hardware-in-the-loop testing during design of power electronics controls," *The 29th Annual Conference of the IEEE Industrial Electronics Society*, Volume 2, 2003, pp. 1840–1845.

[MOK90] A. K. Mok, "Real-time software design—from theory to practice," *IEEE Region 10 Conference on Computer and Communication Systems*, September, 1990, Hong Kong, pp. 394–398.

[MOO02] V. Mooney and D. Blough, "A hardware-software real-time operating system framework for SoCs," *Design & Test of Computers, IEEE*, 2002, 19, pp. 44–51.

[OST98] J. S. Ostroff and R. F. Paige, "Formal methods in the classroom: the logic of real-time software design," *Proceedings of Real-Time Systems Education III*, 1998, pp. 63–70.

[PET07] A. Pettersson, D. Sundmark, H. Thane, and D. Nystrom, "Shared data analysis for multitasking real-time system testing," *International Symposium on Industrial Embedded Systems*, July 4–6, 2007, pp. 110–117.

[PLA84] P. Bernhard, "Real-time execution monitoring," *IEEE Transactions on Software Engineering*, 1984, SE-10(6), pp. 756–764.

[SAK98] M. Saksena, "Real-time system design: a temporal perspective," *IEEE Canadian Conference on Electrical and Computer Engineering*, May 24–28, 1998, Volume 1, pp. 405–408.

[SEL03] B. Selic and L. Motus, "Using models in real-time software design," *IEEE Control Systems Magazine*, 2003, 23(3), pp. 31–42.

[SEL96] B. Selic and P. Ward, "The Challenges of Real-Time Software Design," Embedded Systems Programming, 1996, pp. 66–82.

[SHU04] S. Li, J. Wang, W. Dong, and Z.-C. Qi, "Property-oriented testing of real-time systems," *11th Asia-Pacific Software Engineering Conference*, November 30–December 3, 2004, pp. 358–365.

[SID06] S. H. Siddiquee and A. En-Nouaary, "Two architectures for testing distributed real-time systems,"*2nd Conference on Information and Communication Technologies*, 2006, Volume 2, pp. 3388–3393.

[TIM93] M. Timmerman, F. Gielen, and P. Lambrix, "A knowledge-based approach for the debugging of real-time multiprocessor systems," *Proceedings of the IEEE Workshop on Real-Time Applications*, May 13–14, 1993, pp. 23–28.

[UM03] U. Junhyung and K. Taewhan, "Code placement with selective cache activity minimization for embedded real-time software design," *International Conference on Computer Aided Design*, 2003, pp. 197–200.

[UMR83] Z. D. Umrigar and V. Pitchumani, "Formal verification of a real-time hardware design," *20th Conference on Design Automation*, June 27–29, 1983. pp. 221–227.

[WAN04] S. Wang, J. R. Merrick, and K.G. Shin, "Component allocation with multiple resource constraints for large embedded real-time software design," *Proceedings of the Real-Time and Embedded Technology and Applications Symposium*, 2004, pp. 219–226.

[WAN08] L. Wang, "Get real: real time software design for safety-and mission-critical systems with high dependability," *IEEE Industrial Electronics Magazine*, 2008, 2(1), pp. 31–40.

[WED91] H. F. Wedde, B. Korel, and D. M. Huizinga, "A critical path approach for testing distributed real-time systems," *Proceedings of the Twenty-Fourth Annual Hawaii International Conference on System Sciences*, January, 1991, Volume 2, pp. 400–407.

[ZHE04] L., Zhen, M., Kyte, and B. Johnson, "Hardware-in-the-loop real-time simulation interface software design," *Proceedings of the 7th International IEEE Conference on Intelligent Transportation Systems*, 2004, pp. 1012–1017.

Chapter 5

Network Systems

This chapter is dedicated to describing and analyzing the performance, reliability, maintainability, and availability of networks. With respect to the part of networks called the Internet, the chapter builds upon Chapter 6: Future Internet Performance Models. A smart grid application is used to illustrate network functional and performance requirements. The chapter covers the various types of networks, communication protocols, network services, and network architecture.

OVERVIEW

First, an overview of different types of networks is provided in order to give the reader a perspective on networks that will serve as a foundation for learning network details. In addition, because contemporary texts do not always explain the "why" of networks as opposed to the "how," this chapter will explain the rationale of each network concept.

Local Area Network

A local network provides processing and communication services to a community of users in a local area, typically within a corporate or residential geographical domain. Why not have these users communicate directly to the Internet? The reason is that some applications do not require access to the Internet. For example, users may need access to servers that are part of a corporate local area network. In addition, even if access to the Internet is ultimately required in the application, preliminary communication and processing may be necessary in the local area network prior to Internet access. For example, an electric utility may need to access smart meter readings in a local area network prior to communicating them over the Internet to various substations. Also, note that when possible, there is a performance advantage in communicating in a local area network as opposed to using the Internet because local networks employ higher speed communication lines and do not have to contend with the traffic congestion that is present on the Internet.

Computer, Network, Software, and Hardware Engineering with Applications, First Edition. Norman F. Schneidewind.
© 2012 the Institute of Electrical and Electronics Engineers, Inc. Published 2012 by John Wiley & Sons, Inc.

Wide Area Network

The most prominent of this type of network is the Internet. However, there are many private wide area networks that are part of corporate computer communication systems. Due to the geographic extent of these networks, communication services provided by communication carriers are necessary. This is not the case in local networks because the communication distances are sufficiently short that private communication channels (e.g., Ethernet) suffice. The performance of wide area networks is comparable to that of the Internet. Then, why not use the Internet directly? The reason, again, is a question of congestion, since the congestion on a private wide area network is much less than that on the public Internet.

Network Architecture

This aspect of networks is addressed at this point because many texts and articles use this model to explain network operations. It seems that the model is overused because it can appear to readers to be *the network* rather than a representation of network operations. The essence of the layered model is that corresponding layers between two computers in a network communicate, for example, between application layers. In actuality, the layers do not communicate. This is a software conception of how the major parts of a network interoperate. Actual communication is accomplished by a combination of hardware and software, as shown in Figure 5.1. This figure shows the function of each layer using the smart meter application as an example. The only *actual* communication that takes place in Figure 5.1 in the layered architecture is in the physical layer. In contrast to layered, virtual communication, the figure also shows the real communication between network computers.

NETWORK APPLICATION

To provide context for the various facets of network analysis that are presented in this chapter, a smart grid application is discussed. The smart grid is a network of computers and power devices that monitor and manage energy usage. Each energy producer—for example, a regional electrical company—maintains operational centers that receive usage information from collector devices placed throughout the served area (see the smart meter in Fig. 5.1). In a typical configuration, a neighborhood contains a single collector device that will receive periodic updates from each customer in the neighborhood via the Internet. The collector device reports usage readings to the operational centers using communication media such as the Internet.

Usage Reporting

The electric utilities manage transmission and perform billing based on smart meter readings and send this information to the database in Figure 5.1. The usage-reporting

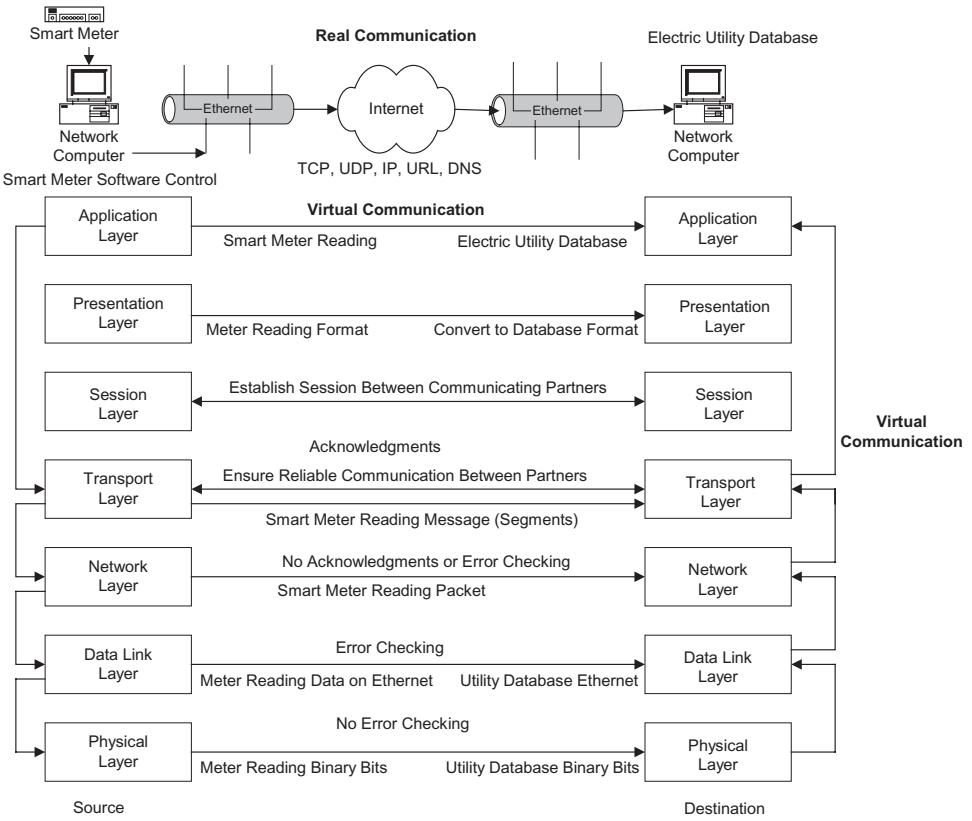


Figure 5.1 Network architecture.

device at each customer site is called a smart meter. It is a computerized replacement of the electrical meter attached to the exterior of many of our homes today. Each smart meter contains a processor, nonvolatile storage, and communication facilities. Although in many respects the smart meter's look and function is the same as its unsophisticated predecessor, its additional features make it more useful. Smart meters can track usage as a function of time of day, disconnect a customer via software, or send out alarms in case of problems. The smart meter can also interface directly with "smart" appliances to control them, for example, turn down the air conditioner during peak periods [MCD09]. Smart meters can collect a unique meter identifier, timestamp, usage data, and time synchronization every 15–60 minutes.

Data Requirements

In the United States, there are 338 million meters in operation. To bring the electricity grid into the digital era, every meter, and the millions of devices that connect to them, must be smart. Devices need to measure and transmit data, act on incoming

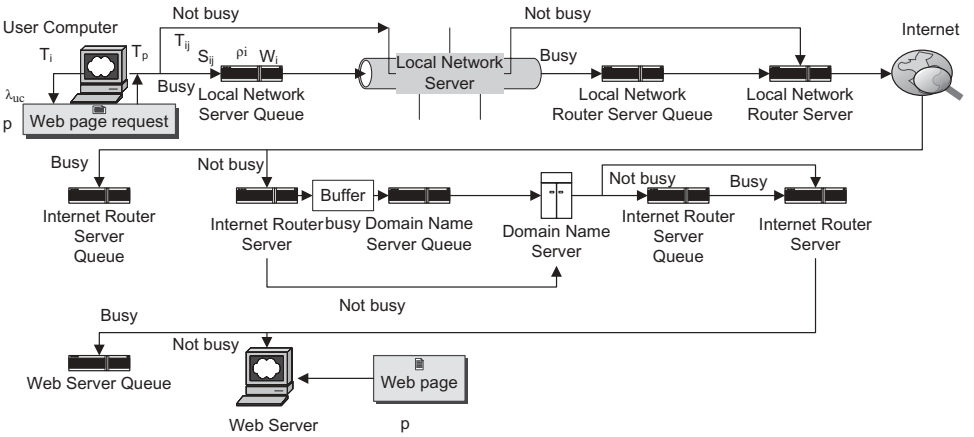


Figure 5.2 Network performance model. T_i, node i processing time; ρi, probability of node i being busy; λ_{uc}, user computer input rate; W_i: node i wait time; p, Web page packet request size or Web page size; T_p, packet input time; T_{ij}, link ij processing time; S_{ij}, link ij speed.

information, and handle innovative applications. This will require a network that can accommodate the sum of information that will be generated by the smart grid. For example, if the 338 million meters already deployed in the United States digitally reported the most basic electricity use information every 15 minutes, they would generate anywhere from 274 to 548 GiB of information every day (http://www.smartsynch.com/smartsynch_infrastructure.htm). Components such as those shown in Figure 5.2 would require this capacity.

NETWORK PROTOCOLS

Network protocols are rules of communication that govern how data are communicated in a network. Most of the protocols are used in the Internet due to the complexity of that network relative to local area networks. These protocols and ancillary items that are related to protocols will now be described.

Transmission Control Protocol (TCP) and Virtual versus Real Communication

This protocol operates in the Transport Layer in Figure 5.1 and is responsible for ensuring reliable end-to-end communication in the Internet. By "end-to-end," it is meant from network computer to network computer in Figure 5.1. While this objective sounds good, realistically, it cannot be achieved; no system can be failure free. An attempt is made at reliable communication by using acknowledgements. The receiver acknowledges to the sender that a "correct" message was received. The

message is assumed to be *incorrect* if an acknowledgement is not received within a specified time called the "time-out period." Since, as stated, no system is failure free, correct messages cannot be guaranteed. Rather, an attempt is made at correctness by appending error check data—which is computed over the message—to the message and transmitting this package to the receiver. The receiver, in turn, computes error check data over the received message, using the same algorithm that was used at the sender. The reliability of this process will be addressed in a later section. An interesting facet of TCP reliability is that a great deal of overhead is generated when the odds are that a high percentage of messages will be transmitted without error. This overhead injects additional congestion into the Internet, lowering the performance of the entire Internet. It seems that reliability–performance trade-offs were not considered in designing TCP. Also to be noted is that the Application Layer provides TCP with the Smart Meter Reading in Figure 5.1. Thus, in addition to the "horizontal" virtual communication, there is "vertical" virtual communication between layers. Of course, both "horizontal" and "vertical" virtual communications are fictitious; these are modeling artifacts. The only true communication is the "real communication" in Figure 5.1.

Also note in Figure 5.1 that the concept of vertical virtual communication occurs at both the source and destination. In the former, the Smart Meter Reading application data is sent from the Application Layer to TCP, where the data become a TCP message that is fortified with acknowledgement, error checking, and sequencing information. The last item is required because messages are actually comprised of smaller segments for efficiency in communication and processing (e.g., reduced buffer space at both source and destination). Each segment is assigned a sequence number because segments can get out of order when they are routed through the Internet due to different delay times experienced by the segments. Then, the segments are sent to the Network Layer where they are transformed into a series of packets in Figure 5.1, where the packets contain source and destination Internet Protocol (IP) addresses and routing information (e.g., IP address of next router). Local networks such as Ethernet—usually owned by user communities—are required at the source and destination in order to process and communicate data within the user communities, and to interact with the Internet. Therefore, Network Layer packets are "virtually transmitted" to the Data Link Layer. This layer's protocol provides error checking between source and destination local networks. Note that up to this point in the discussion of the layering approach the functionality in Figure 5.1 is implemented primarily in software. Now, at the Physical Layer, hardware is used to transmit binary bits from source to destination. No error checking is necessary in this layer because this function is performed in the Data Link Layer.

At the destination, the raw bits in the Physical Layer are checked for correctness by the Data Link Layer. Next, the packets that have been buried in the binary bits of the Physical Layer and the data of the Data Link Layer are recovered. The Network Layer also checks segment sequence, reassembling segments in the correct order, and thus recovering the TCP message. This is not the end of the story because the messages require error checking by the Transport.

User Datagram Protocol (UDP)

In contrast to TCP, the UDP does not attempt to ensure reliable communication. Operating in the Network Layer of Figure 5.1, it transmits data with no acknowledgements, thus providing higher performance compared with TCP. The term "datagram" could be confusing to the reader; "message" would suffice. To add to the confusion, "message" is used in TCP, wherein, in reality, both TCP and UDP transmit messages. However, Internet working groups designate this terminology because UDP transmits short messages, called datagrams, whereas TCP transmits messages comprised of several packets, where a packet is defined as data that have a header for an address and routing information, a body for the actual data, and a trailer for error check information.

Internet Protocol (IP)

This rule of communication is used in all Internet data transfers and is associated with the Network Layer in Figure 5.1. Interestingly, the TCP message is appended to the IP packet, where a packet is simply a set of binary bits that is transmitted in the Internet. Thus, in the literal sense, a TCP message is not transmitted in the Internet. Rather, it is the IP packet that is transmitted.

NETWORK SERVICES

Network services are network functions that provide services to users. In addition to their functions, services are distinguished from protocols by virtue of being affixed, by servers, to one or more points in local networks, as opposed to protocols that operate over communication channels between points in a network.

Domain Name Service (DNS)

The DNS can be mystifying to readers because the natural question arises: why can't my data be communicated in the Internet by using the name of my computer and the name of the resource I wish to access? The reason this is not feasible is that to access a resource in the Internet, an IP address is required. The reason for this requirement is that IP addresses provide generality in the Internet. That is, with each resource in the Internet having an IP address, which may be assigned permanently (e.g., Web server) or temporarily (e.g., duration of a network computer transaction), any resource can be accessed. While it could be possible to maintain tables of network computer names in order to access these resources, it would be inefficient because the names would vary in length and not all network computers would remain connected to the Internet over time. Thus, temporary assignment of an IP address for the duration of a transaction has proven effective. However, users do not want to remember IP addresses. It is more natural for them to deal with computer names.

Besides, as mentioned, IP addresses are only assigned temporarily. Therefore, DNS converts from a network computer name to an IP address when the computer accesses a resource (e.g., Web server) and performs the reverse conversion—from IP address to Uniform Resource Locator (URL)—when the Web server is accessed. The URL, a bureaucratic name if there ever was one, is the name of a Web server that is used by the network computer for accessing the Web server. Once the DNS converts network computer name to an IP address, the Web server uses it to respond to the Network Computer's request.

Web Site Services

These services go into action when users request Web pages on the Internet. Users are unaware of the many messages that transpire in the Internet when they access a Web page. In addition to the user's request, messages are required to perform DNS name-to-IP address translation and to establish a session between user and Web site. Thus, in assessing the user's performance experience on the Internet, many supporting "hidden messages" must be accounted for in addition to application message.

Session and Presentation Layer Services

Actually, these are nonservices because they are not needed by these layers! Then, why are they present in the architecture? The answer is that the international standards group included them because they believed these functions would be performed by distinct layers in the architecture. However, Internet architects assigned TCPs to session establishment by virtue of acknowledgements and they designated applications to format source data (e.g., user formatting of Web page requests) and services to format response data at the destination (e.g., Web site formatting of requested Web page). However, since the seven-layer architecture is the holy grail of networks, it is incumbent for book authors to include it.

NETWORK PERFORMANCE

In this section, network performance equations will be developed for each component shown in Figure 5.2. Later, relevant network performance data from Chapter 6 will be used in the equations to estimate the performance of extant computer networks.

Link Delay Times

These are the times required to transmit data on a link from the source point to the end point; for example, the delay time from the user computer to the local network queue in Figure 5.2. Thus, T_{ij} is link time, as computed below:

$$T_{ij} = p / S_{ij},$$

Table 5.1 Network Performance Parameters

Data item	Source	Value
Asymmetric digital subscriber line (ADSL) Internet communication channel speed	www.webopedia.com/	λ_{uc} = 640,000 bits per second
Local network (Ethernet) link speed	[HAM02]	S_{ij} = 100,000,000 bits per second
Local network router processing speed	http://arstechnica.com/hardware	S_i = 54,000,000 bits per second
Internet router link speed	www.highspeedrouter.com/	S_{ij} = 6,250,000 bits per second
Domain name server (DNS) processing speed	www.labnol.org/	S_i = 143,000 bits per second
DNS processing time	/www.labnol.org/	T_i = 0.007 seconds
Web server link speed	www.google.com	S_{ij} = 2,418,500 bits per second
Web page size	www.google.com	p = 96, 928 bits
User computer processing speed	[HAR07]	S_i = 2,000,000,000 bits per second
Local network server processing speed	Assume same speed as user computer	S_i = 2,000,000,000 bits per second
Internet router server processing speed	Cisco	S_i = 100,000,000,000 bits per second
Web server processing speed	http://www.info-techs.com/speedtest50.html	S_i = 12,439,000 bits per second

where p is the Web page request packet and Web page size and S_{ij} is speed of link ij. p is assumed to be exponentially distributed, with mean = 1000 bits for Web page request packet and mean = 96,928 bits for Web page size (see Table 5.1 for this information). The exponential distribution is justified on the basis of higher probability of small values of p and lower probability of large values. Values of p are generated by using the mean values in an exponential distribution, using a statistics program (e.g., Minitab).

Question for Reader: Why not use the mean values of Web page request packet and Web page size, rather than assume an exponential distribution and generate various values?

Answer: Single or mean values of p do not exist in real networks. Rather, in real networks, there exists a distribution of sizes, where the exponential is the most rationale distribution to use.

In addition to individual link delay times, it is also important to compute the mean of link delay, MT_L, and time over all links, N_L, to obtain a metric of *network* com-

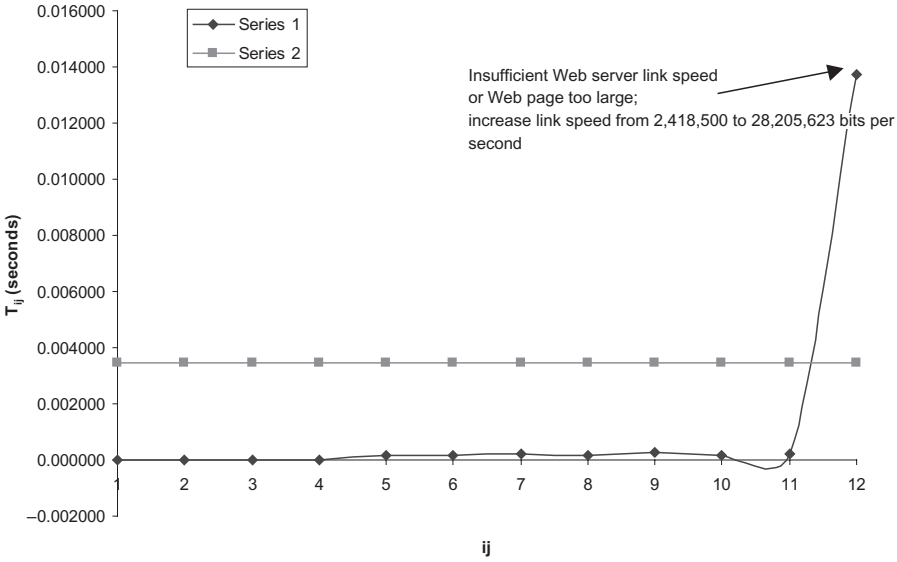


Figure 5.3 Link delay time T_{ij} versus link ij. Series 1: T_{ij}. Series2: Mean of T_{ij}.

munication performance that can be compared with individual link delay times to see which link delay times are excessive and in need of performance improvement by increasing link speed S_{ij}. This metric is computed as follows:

$$MT_L = \sum_{ij}^{N_L} T_{ij} / N_L.$$

Figure 5.3 puts the metric MT_L into action by identifying the Web server of exhibiting anomalous behavior: excessive link delay time attributed to the Web server, calling for an increase in its link speed. However, note that another contributor to excessive link delay is the large Web page. Unfortunately, it may be infeasible to reduce the size of the Web page. Therefore, the feasible option is to obtain a Web service that can provide the desired speed, where this speed is computed as follows, using the unchanged Web page size and the mean line delay:

$$S_{ij} = p / MT_L.$$

Figure 5.3 shows the increased Web service link speed designed to correct the performance deficiency.

Node Processing Times

Nodes in Figure 5.2 are any objects that are not a link (e.g., user computer). Thus, T_i is the processing time of node i, as computed below:

$$T_i = p / S_i,$$

where S_i is the processing speed of node i.

In addition to individual node processing times, it is also important to compute the mean processing time, MT_N, and time over all nodes, N_N, to obtain a metric of *network* processing performance, computed as follows:

$$MT_i = \sum_{i}^{N_N} T_i / N_N.$$

Then, individual node processing times can be compared with the mean to identify nodes that may be causing excessive processing time and, thus, are in need of processing speed increase.

Again, as was the case with link delay, Figure 5.4 demonstrates that the Web server is a bottleneck.

The remedies are to either increase the Web server processing speed or to decrease the Web page size. Therefore, again the performance problem can be solved by increasing the Web server processing speed, using the unchanged Web page size and the mean node processing time as follows:

$$S_i = p / MT_i.$$

Figure 5.4 shows the increased Web service processing speed designed to correct the performance deficiency. Note, however, that since both the increased Web service

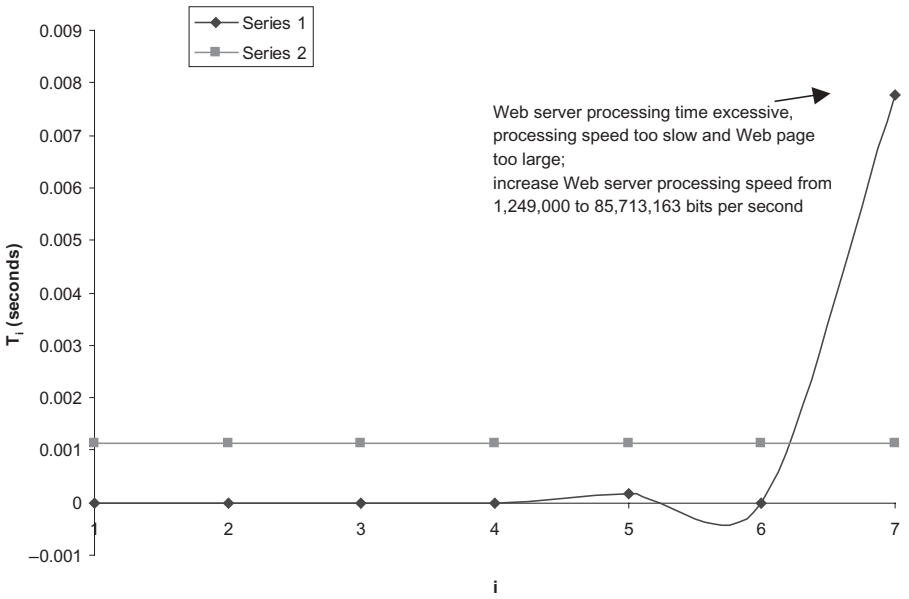


Figure 5.4 Node processing time T_i versus server node i. Series 1: T_i. Series 2: Mean of T_i.

link speed and node processing speed are significantly greater than the original values, it may not be practical to achieve these rates. Thus, it may be necessary to settle for improved Web server service but not to the extent suggested by Figures 5.3 and 5.4.

Node Probability of Being Busy

Note in Figure 5.2 that nodes such as local network server may or may not be busy. Being busy means that there are one or more items in a queue waiting to be processed. Thus, the probability of node i being busy, ρ_i, is related to the data input speed to node i, on link ij, S_{ij}, and to the processing speed of node i, S_i, as follows:

$$\rho_i = S_{ij} / S_i.$$

There is one exception to the application of this equation and that is the determination of ρ_i for the Domain Name Server (DNS). For the DNS, the Internet Router link speed, in Figure 5.2, which is the DNS link speed, is so much greater than the DNS processing speed (see Table 5.1) that it would be necessary to provide at buffer at the input of the DNS in Figure 5.2. This is done to prevent the DNS from being overrun by Internet traffic. $\rho_i = S_{ij}/S_i$ cannot be used because it would yield a value much larger than 1.0, which would indicate queue instability. However, this is not the case when the buffer is employed. To make the DNS operate in a stable manner $\rho_i = 0.8$ is assigned as the DNS busy metric.

Packet Input Time

Packet input time, T_p, is a driver of network operations that is needed to estimate its influence on wait time in a queue. Its influence is exerted because the rate of data input generated by the user computer, λ_{uc}, may cause the links and nodes down the line to be overwhelmed with data and, hence, increasing wait time. T_p is computed as follows:

$$T_p = p / \lambda_{uc},$$

where p is the packet size.

Figure 5.5 demonstrates the influence packet input time on wait time, in that wait time follows the pattern set by packet input time as a function of the node where the wait time occurs. The utility of this plot is to identify the node associated with anomalous high values, which in this case is the Domain Name Server (DNS), and to correct this deficiency by obtaining the services of a DNS provider that has a DNS with the requisite speed.

Node Wait Time

Node i wait time, W_i, can be estimated by considering that if node i probability of being busy, ρ_i. is 0, there are no items waiting for processing at node i. On the other

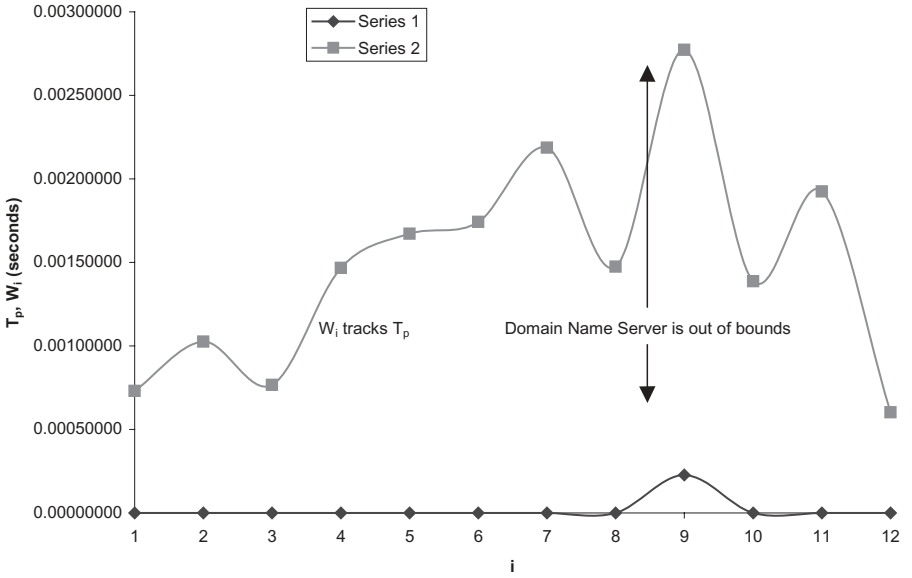


Figure 5.5 Packet input time T_p and node wait time W_i versus node i. Series 1: W_i. Series 2: T_p.

hand, if ρ_i is 1, the indication is that node i is in its maximum busy state and an item would have to wait to be processed. If the probability of node i being busy is $0 < \rho_i < 1$, it indicates the degree of busyness. Thus, on an *expected value* basis, node i wait time, W_i, is estimated as follows:

$$W_i = \rho_i T_i.$$

For example, if $\rho_i = 0$, then $W_i = 0$; if $\rho_i = 1$, then $W_i = 1$. That is, with $\rho_i = 0$, of course there is no waiting and $W_i = 0$. However, with $\rho_i = 1$, an item would have to wait for the previous item to be processed for a time T_i.

Node Processing Time plus Node Wait Time

Processing time alone does not tell the entire story of node performance. What is needed is to account for wait time, which could be significant. Therefore, the sum of these times, TW_i, is computed as follows:

$$TW_i = T_i + W_i = p / S_i + \rho_i T_i.$$

In addition, to provide a standard for evaluating the performance of individual nodes, the mean of TW_i is computed over N_n nodes as follows:

$$MTW_i = \frac{\sum_{i=1}^{N_n} TW_i}{N_n}.$$

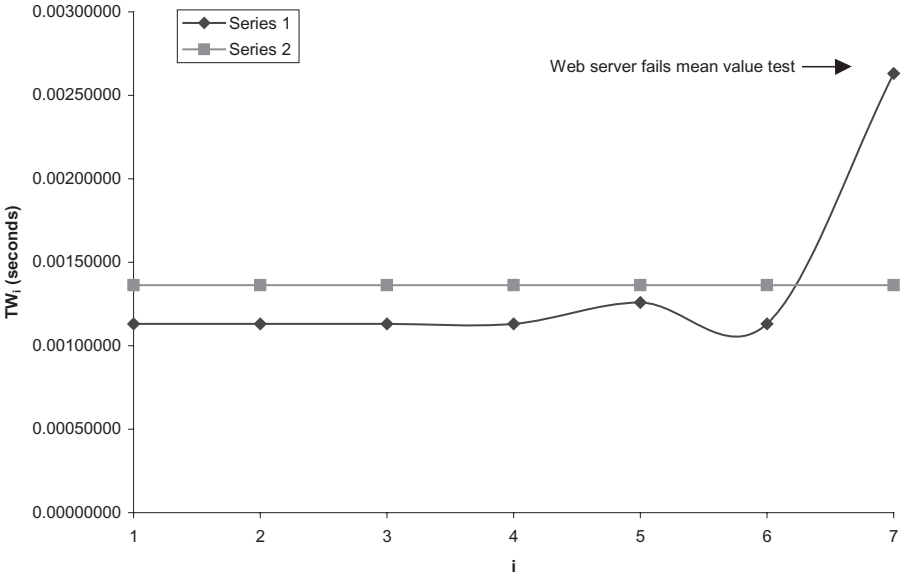


Figure 5.6 Node processing time plus wait time TW_i versus node i. Series 1: TW_i. Series 2: Mean of MTW_i.

The value of TW_i is compared with MTW_i to identify node processing times that may be excessive. This comparison is demonstrated in Figure 5.6, where the Web server is identified as failing to meet the mean value test. In these cases, there would be a need for node processing speed S_i to be increased.

Summation of Link Delay, Processing Time, and Wait Time

To obtain a comprehensive performance metric of an entire network, T_t, link delay, node processing time, and wait time are summed over number of links, N_L, and N_n, number of nodes in a network, as follows:

$$T_t = \sum_{ij}^{N_L} T_{ij} + \sum_{i}^{N_n} T_i + \sum_{i}^{N_n} W_i.$$

In addition, the mean MT_i of T_i is computed over all N_N nodes and N_L links as follows:

$$MT_i = \sum_{i,ij}^{N_N N_L} T_t.$$

It is appropriate to compare total network time T_t with the user expectation T_e to see whether the performance is meeting expectation. Furthermore, the relative error RE between expected and realized times is computed as follows:

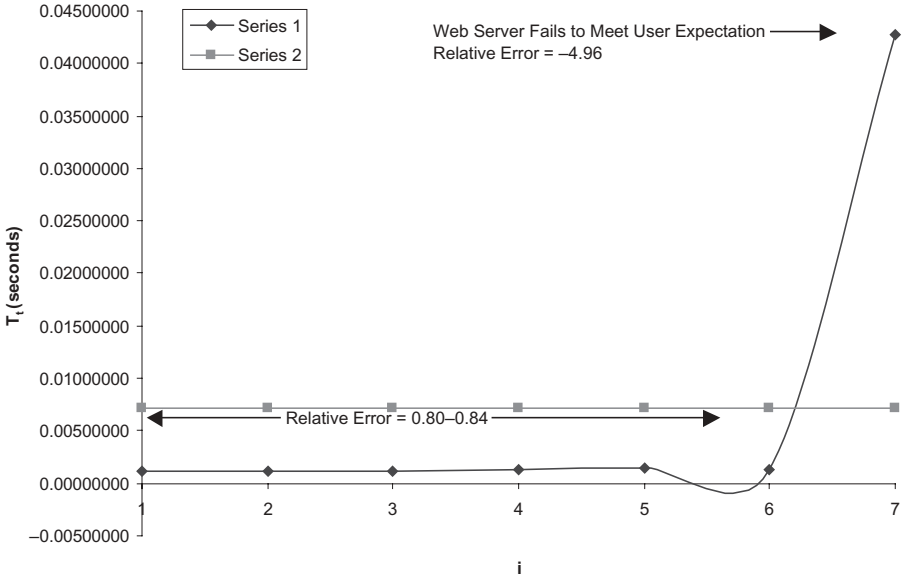


Figure 5.7 Node processing time plus wait time plus link delay T_t versus server node i. Series 1: T_t. Series 2: Mean of T_t.

$$RE = (T_e - T_t) / T_e,$$

where positive or zero values indicate that the user expectation of $T_t \leq T_e$ is satisfied and negative values indicate that the user expectation is not satisfied. By examining individual link delay, node processing time, and node wait time, the source of performance problems can be identified that could be excessive link delay or node processing time, or both. This comparison is performed in Figure 5.7, where it is shown that high RE is associated with the Web server failing to meet the total node processing, wait, and link delay time expectation.

Network Performance Parameters Data

The network performance parameter data that will be used in the network performance equations is documented in Table 5.1.

NETWORK RELIABILITY, MAINTAINABILITY, AND AVAILABILITY PREDICTION

In addition to performance, it is important to predict the reliability, maintainability, and availability that can be achieved in a network.

Reliability

The factors that govern reliability in a network are the following:

Both links and nodes must be used in prediction equations.

Because reliability is higher for small values of link delay and node processing time than for large values, the appropriate reliability function is the exponential.

The probability of a node being busy must be included in reliability prediction equations because when nodes are busy, not only are nodes busy but the connecting links are also busy because the data on the links must be delayed for processing until the nodes are no longer busy, thus exposing links to increased possibility of failure.

The failure rate λ is a random variable that is generated by using the Excel RAND function.

Thus, proceeding to use these factors in developing reliability prediction equations, the link failure rate, λ_{ij}, is computed as follows, applying the probability of node i being busy:

$$\lambda_{ij} = \rho_i \lambda.$$

The exponential distribution is put to work to predict link reliability, R_{ij}, where Tij is link delay:

$$R_{ij} = e^{-(\lambda_{ij} T_{ij})}.$$

Now node reliability is formulated in a manner similar to links. First, failure rate: as in the case of links, failure rate is the product of probability of node busy and the Excel RAND function, λ, generating difference values for this computation:

$$\lambda_i = \rho_i \lambda.$$

Then, the exponential distribution is called on to predict node reliability, R_i, applying node processing time T_i:

$$R_i = e^{-(\lambda_i T_j)}.$$

Maintainability

Maintainability is formulated by considering how the probability of maintenance actions can be estimated. The concept is that maintainability is a probability, and this probability is the ratio of the quantity of data processed by a given link or node to the total quantity of data processed at all links and nodes in the network. The quantity of data that is processed by each link and node is $p_{ij,i}$, the Web page request size for all links and nodes, except for the Web server and its associated link, where $p_{ij,i}$ is the Web page size. Thus, maintainability is predicted as follows, where N_L is the number of links and N_N is the number of nodes in the network:

$$M_{ij,i} = \frac{p_{ij,i}}{\sum_{ij,i}^{N_L N_N} p_{ij,i}}.$$

The primary purpose of Table 5.2 is to account for the links and nodes in the maintainability predictions. See Figure 5.2 as an aid in making this accounting. The Web page request packet and Web page sizes in Table 5.2 were generated from the aforementioned exponential distribution process.

Availability

Availability is important in all systems, including networks. It represents the fraction of time that a network is operational for useful work. The fraction of time that the network *is not* available is the time consumed in maintaining the system, and the fraction of time the network is *not being maintained* and doing useful work is when it is operating reliably. These fractions of times can be translated into corresponding probabilities in order to produce a general availability expression as follows:

Maintainability. The probability that the network *is not* available.

Reliability. The probability that the network *is* available.

Thus, link availability, A_{ij}, is predicted as follows:

$$A_{ij} = R_{ij} / (R_{ij} + M_{ij,i}),$$

and node availability, A_i, is predicted as follows:

$$A_i = R_i / (R_i + M_{ij,i}).$$

The results of combining reliability and maintainability into availability predictions are shown in Figure 5.8, where the link and node availabilities are almost identical so that only one availability plot is shown along with the required availability of 0.9800. This requirement means that the user expectation is that the network will be unavailable for not more than 2% of the scheduled operating time. The figure delineates the nodes and connecting links that satisfy the requirement and those that do not. The problem in the latter case is excessive maintainability. Since both link and node reliabilities are high, the remedy would be improved maintenance practices, such as preventive maintenance.

SUMMARY

This chapter has shown how to analyze and predict network performance, reliability, maintainability, and availability. In addition, using the foregoing tools, the reader learned how to identify anomalous performance and availability behavior, such as that exhibited by the Domain Name Server and Web server. Thus, the reader is then fortified with tools for correcting these deficiencies.

Table 5.2 Maintainability Data

Link	Node	p (Web page request or Web page size in bits)
User computer to local network server queue		467.29
	Local network server queue	467.29
Local network server queue to local network server		656.28
	Local network server	656.28
Local network server to local network router server queue		491.06
	Local network router server queue	491.06
Local network router server queue to local network router server		938.72
	Local network router server	938.72
Local network router server to internet router server queue		1,069.94
	Internet router server queue	1,069.94
Internet router server queue to internet router server		1,115.03
	Internet router server	1,115.03
Internet router server to domain name server queue		1,399.53
	Domain name server queue	1,399.53
Domain name server queue to domain name server		943.51
	Domain name server	943.51
Domain name server to internet router server queue		1,774.3
	Internet router server queue	1,774.3
Internet router server queue to internet router server		887.43
	Internet router server	887.43
Internet router server to Web server queue		1,231.98
	Web server queue	1,231.98
Web server queue to Web server		385.84
	Web server	385.84
Web page to Web server		33,181.3
	Web server	33,181.3
	Total	44,542.21

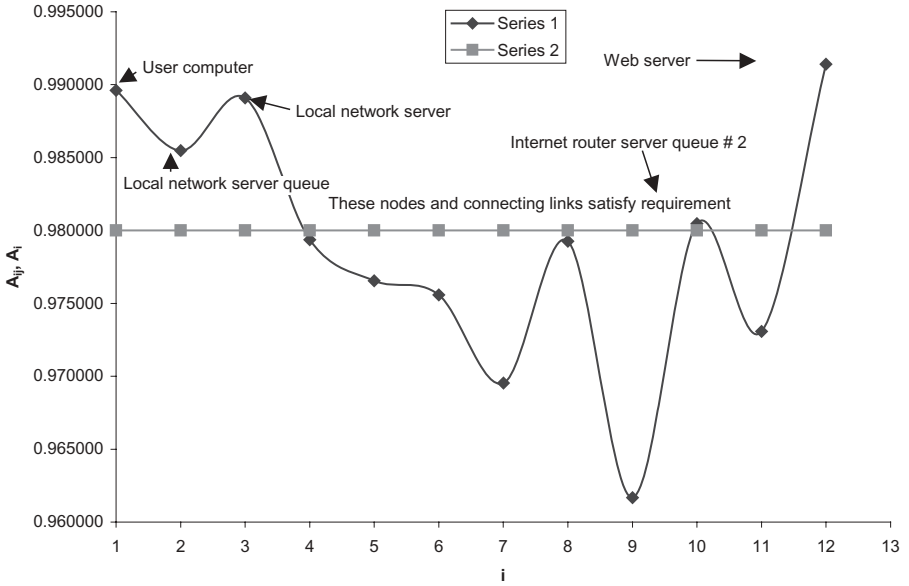


Figure 5.8 Link availability A_{ij} and node availability A_i versus node i. Series 1: A_{ij}, A_i. Required A_{ij}, A_i.

REFERENCES

[HAM02] Matt Hamblen, "10 Gig Ethernet: Speed Demon," Computer World: Networking & Internet, December 23, 2002.

[HAR07] David Money Harris and Sarah L. Harris, *Digital Design and Computer Architecture*. New York: Elsevier, 2007.

[MCD09] Patrick McDaniel, Stephen McLaughlin, "Security and privacy challenges in the smart grid," *IEEE Security and Privacy*, 2009, 7(3), pp. 75–77.

Chapter 6

Future Internet Performance Models

Having learned the fundamentals of computer design in Chapters 1 and 2, which apply to computers such as personal computers, you are ready to study a topic that is pervasive in the world of information technology—the Internet. Since there are few computer applications that do not use the Internet, I provide the reader with a perspective of the evolving Internet, using the present Internet as a baseline. The performance and reliability of a proposed future Internet—wired and wireless—is compared with the present Internet. Models of data transfer and queuing dynamics are used to make the performance comparison. These models consist of logic diagrams that express the sequence of data transfers in the Internet (e.g., local network to local network router) and queuing logic diagrams, supported by queuing equations (e.g., probability of local network queue busy). These models represent the steady-state behavior of the present and proposed future Internets. Computer programs are used to simulate the variability in queue behavior. The results are used to identify the major variables in Internet performance (e.g., Internet routing time as a major performance variable). Furthermore, the results are used to compare present Internet and proposed future Internet performance. Reliability analysis is performed by predicting cumulative Internet faults and failures and by analyzing the complexity of present and proposed Internet configurations as an indicator of reliability (i.e., number of points of failure in a configuration). Model results demonstrate significant increases in performance and reliability for the proposed Internet, attributed to the elimination of data transfer overhead (e.g., elimination of Domain Name Service) and simplified network configurations.

CHAPTER OBJECTIVES

One objective is to compare the performance and reliability of the present Internet with a proposed Internet of the future that could operate faster, more reliably, and with improved security, by eliminating the overhead induced by a multiplicity of protocols, intermediate networks, and interfaces that comprise the current Internet.

Computer, Network, Software, and Hardware Engineering with Applications, First Edition. Norman F. Schneidewind.
© 2012 the Institute of Electrical and Electronics Engineers, Inc. Published 2012 by John Wiley & Sons, Inc.

In order to illustrate the proposal, I developed analytic queuing models and simulation models for comparing the performance and reliability of the current versus the proposed Internet. The process starts by defining the network topology for present and future Internet configurations. This leads to identifying and defining the performance and reliability and variables of the model. In developing the prediction equations, the sequence of operations on the network—for example, an input request to the Internet—provides the basis for computing the performance and reliability of the present and proposed Internets.

To add realism to the models, I use publicly available performance and reliability data posted on the Internet. While the performance and reliability of present and proposed Internets are of interest, it is the *comparison* of the two that is my core objective that would demonstrate whether the proposed Internet is viable. Both wired and wireless Internets are included in the analysis, in both upload direction (i.e., request for Web page) and download direction (i.e., delivery of Web page). Based on extensive literature search, no one has proposed fundamental changes in the Internet configuration, as I propose. Rather, current research focuses on the present Internet configuration as a given, with proposals to improve quality of service, reliability, and so on, on the existing platform.

PROPERTIES OF THE PROPOSED FUTURE INTERNET

In today's Internet architecture, the Internet Protocol (IP), Internet addresses, and the Domain Name Service (DNS) implement core architectural principles that restrict the Internet's ability to adapt to improved performance and reliability requirements [GOE07]. In the future Internet, the current edge of the network (e.g., user computers and mobile devices) will often be just one hop to the Internet [FAI08]. That is, devices will be able to connect directly into the Internet, eliminating barriers such as local networks, local network routers, and domain name servers. The trend to connect more devices will also accelerate, facilitated by the increasing installation of Internet Protocol version 6 (IPv6). In the future, the Internet will connect vast numbers of tiny devices integrated into cell phones and other mobile devices [FAI08]. These devices may challenge the traditional understanding of network topology as a collection of networks and, instead, view the future Internet as a single unified network.

According to Gokhale et al. [SWA06], in a process-based Web server architecture, the server consists of multiple single-threaded processes, each of which handles one request at a time. In a thread-based architecture, the Web server consists of a single multithreaded process; each thread handles one request at a time. However, there is another Web server model—the one I use. This model uses multiple executing servers, each processing user requests concurrently.

The proposed future Internet is comprised of the following capabilities:

- Rather than using local networks, such as Ethernet, communication between user computers and Web servers and between mobile devices and Web servers would be direct, via Internet routers.

- Devices would be assigned permanent IP addresses, issued by the Internet authority, thus eliminating the need for name-to-IP address translation, thereby eliminating the need for Domain Name Systems. User computers and mobile devices would access a Web server by providing a Universal Resource Locator (URL) (Web site address) to the Internet service provider (Isp). The Isp, in turn, would look up the Web server IP address in its directory and append it to the IP packet. In case the IP address has not been recorded in the Isp directory, the Isp would broadcast a request to obtain the IP address.
- In order to provide increased security of data, every user computer and mobile device would have its own IP address, requiring the replacement of Internet Protocol version 4 (IPv4) with IPv6, in order to provide for a large address space. IPv6 does not provide any better (or worse) support for quality of service than IPv4, but it does have several important features that would enhance the performance and security of the future Internet, including larger address space, integrated security capabilities, easier configuration, and a simplified packet header format [MET03].
- Reliability would be increased because there would be fewer components that could fail and fewer single points of failure (e.g., elimination of local networks and Domain Name Systems). This is important because the Domain Name System is reputedly one of the main causes of failure in the Internet [PAR].
- Cyber security would be increased because there would be fewer components that could be attacked and if attacks do occur, resolution would be simplified by pinpointing the location of an attack by virtue of using a much simpler Internet configuration than is presently the case.

NETWORK USAGE DATA

In developing the Internet evaluation models, using queuing models, it is important to use real-world data, as advertised on the Internet and documented in Table 6.1. Some items in the table are descriptive to indicate the magnitude of wired and wireless Internet traffic and storage requirements. Other items are used to compute quantities that are used in queuing and simulation analyses.

QUEUING MODEL (PRESENT INTERNET SYSTEM)

In this section the various queuing model equations, computations, and plots [HIL01], using data from Table 6.1, are presented, encompassing upload of Web page requests and download of Web pages, for both wired and wireless technologies, for the present Internet system. The queuing models are based on a continuous timescale of user computer and mobile device Web page requests and corresponding Web server Web page deliveries in order to provide realistic portrayals of Internet performance and reliability that would not be feasible with a discrete, time-sampled approach ([JIN08], [TAK93]).

Table 6.1 Network Usage Data

Item	Reference	Period	Queue object	Quantity	Quantity computation
Mobile access point (descriptive)	[INS]	2008	Server	3,451,680	3,451,680 access points per year/12 months per year = **287,640 access points per month**
Worldwide mobile Subscribers (descriptive)	[ITU09]	2009	Input	4,197,544,101	4,197,544,101/365 days per year/24 hours per day/3600 seconds per hour = **133.10 subscribers per second**
Mobile spectrum usage (descriptive)	[ITU09]	2010	Input rate	1280–1720 MHz	
Worldwide mobile data traffic (descriptive)	[IVA09]	2009	Storage requirement	100,000 Tbit/month	100,000 Tbit/month/287,640 access points per month = **0.3477 access point**
Wireless backbone	[IVA09]	2009	Download processing rate	1.5–34 Mbit/s (uniform distribution assumed)	Mean = (34 – 1.5)/2 + 1.5 = **17.75 Mbit/s**
Asymmetric digital subscriber line, ADSL	www.webopedia.com/	2009	Web page download rate, Web packet request upload rate	1.5–9 Mbit/s (download), 0.016–0.640 Mbit/s (upload), 2–2.5 mile radius (uniform distribution assumed)	Mean = (9 – 1.5)/2 + 1.5 = **5.25 Mbit/s** (download) Mean = (0.640 – 0.016)/2 + 0.016 = **0.328 Mbit/s** (upload)
Local network (Ethernet) speed	[HAM02]	2002	Processing rate	100 Mbit/s <24 miles	

Local network router	arstechnica.com/hardware	November 5, 2007	Router speed	54 Mbit/s	
Internet router	www.highspeedrouter.com/	2009	Router speed	1–11.50 Mbit/s (uniform distribution assumed)	Mean = (11.50 – 1)/2 + 1 = **6.25 Mbit/s**
Personal digital assistant (PDA) data traffic	Wireless Communications Association International	2009	Wireless packet length per unit time	259.20 Mbit/month (assume packets transmitted in 1-second bursts)	259.20 megabits per month/30 days per month/24 hours per day/3600 seconds per hour*1,000,000 bits per megabit = **100 bits in 1 second**
Domain Name System (DNS)	Digital Inspiration Web site www.labnol.org/	2009	Processing time Processing rate	0.004–0.010 seconds processing time (assume Web page request packet length = 1000 bits)	Mean = (0.010 – 0.004)/2 + 0.004 = **0.007 seconds** processing time 1000 bits/0.007 seconds = **0.143 Mbit/s** processing rate
Web page size	www.google.com		Web page size	12,116 bytes = 96,928 bits	
Web server	www google.com	2009	Web page processing time Web page processing rate	0.040077 seconds processing time	96,928 bits/0.040077 seconds/1,000,000 bits per megabit = **2.4185 Mbit/s** for Web page processing rate
iPhone speed	http://www.apple.com/iphone/iphone-3gs/high-technology.html	2009	Wireless packet upload rate	7.2 Mbit/s	

Figure 6.1 shows the wired logic for upload of a packet that is requesting a Web page. It is assumed that Web pages are downloaded from a Web server to the user computer. Figure 6.1 also shows the wired logic of Web pages downloaded to the user computer. Note that in the case of Internet routers, Domain Name Systems, and Web servers, multiple servers are required in order for the probability of queue

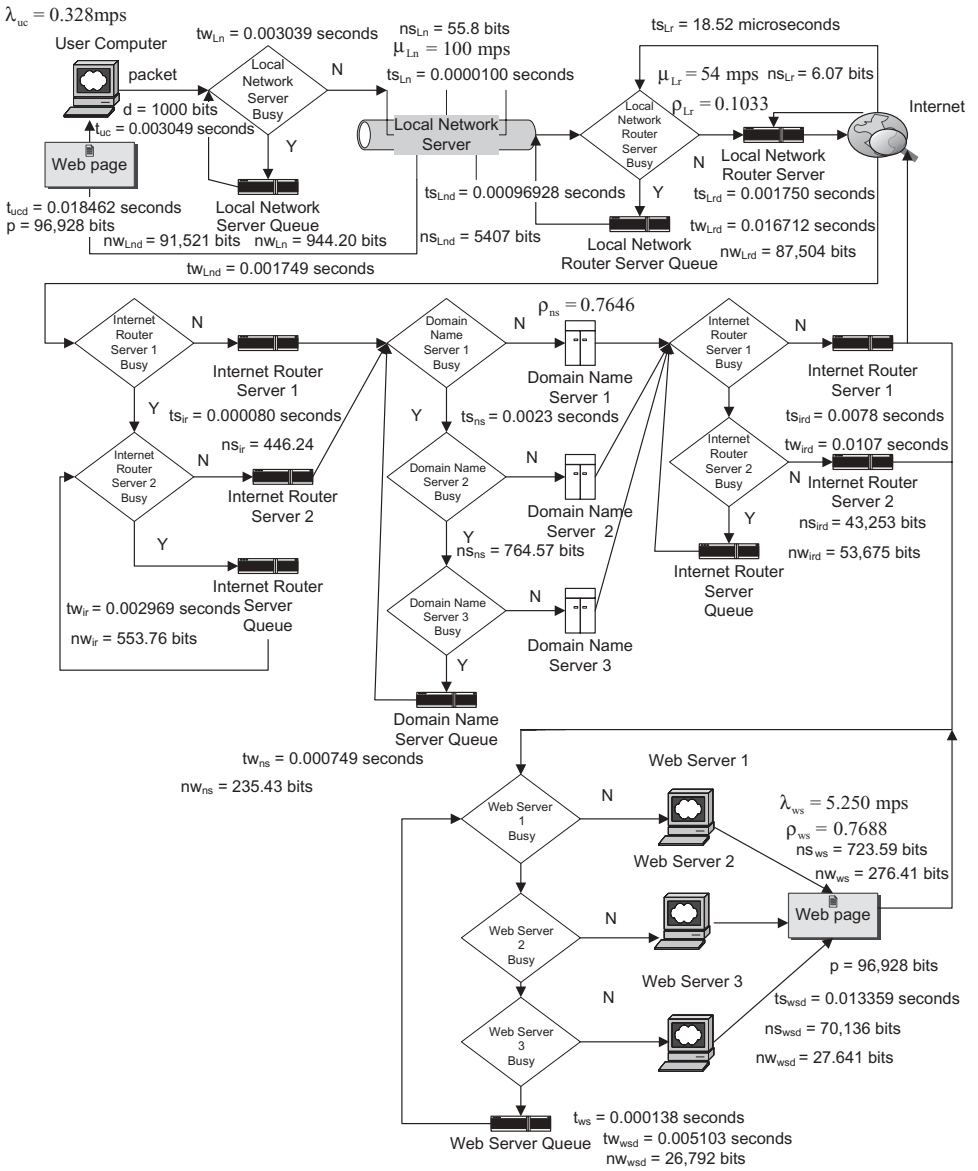


Figure 6.1 Present wired Internet queuing model.

busy <1 (i.e., server utilization <1). Otherwise, the queue systems would become unstable (i.e., the servers would become overwhelmed with traffic).

See the sections entitled "Present Internet Wired Logic Sequences for Upload and Download" and "Present Internet Wireless Backbone" for the explanations of the notations and quantities that appear on the queuing model figures.

The queuing models provide a *mean value analysis* of the wired and wireless performance. While important, mean values are not the whole story of Internet system performance. Since performance will vary considerably from the means, as a function of operating conditions in the Internet, this variation in Internet performance is estimated using simulation queuing models. These estimates are computed in a later section.

Present Internet Wired Logic Sequences for Upload and Download

This subsection contains the mean value equations and computations for the sequence of data transfer and processing operations that are required to upload a request for a Web page, as depicted in Figure 6.1, and to deliver a Web page to the user computer (download), as shown in Figure 6.1, for the *present* Internet wired system, organized by the components that comprise the system. The computations use the data in Table 6.1.

User Computer

Mean Packet Upload Time t_{uc}

$$t_{uc} = \frac{d}{\lambda_{uc}}, \tag{6.1}$$

where d is the packet size and λ_{uc} is the packet upload rate in Figure 6.1:

$$t_{uc} = \frac{1000 \text{ bits}}{0.328 \text{ Mbit/s}} = 0.003049 \text{ seconds.}$$

Web Server

Mean Web Page Download Time t_{ucd}

$$t_{ucd} = \frac{p}{\lambda_{ws}}, \tag{6.2}$$

where p is the Web page size and λ_{ws} is the Web page download rate in Figure 6.1:

$$t_{ucd} = \frac{96{,}928 \text{ bits}}{5.250 \text{ Mbit/s}} = 0.018462 \text{ seconds.}$$

Local Network

Single-server equations apply for the local network, shown in Figure 6.1, because only one server is required for the probability of queue being busy <1.

Probability of Queue Being Busy ρ_{Ln}

This probability is the ratio of the sum of the *packet arrival rate* λ_{uc} (upload) and the *Web page delivery rate* λ_{ws} (download) to the local network *packet service rate*, μ_{Ln}, shown in Equation 6.3. The result of this computation is shown in Figure 6.1:

$$\begin{aligned} \rho_{Ln} &= \frac{\lambda_{uc} + \lambda_{ws}}{\mu_{Ln}}, \\ \rho_{Ln} &= \frac{0.328 + 5.25 \text{ Mbit/s}}{100 \text{ Mbit/s}} = 0.055780. \end{aligned} \tag{6.3}$$

Mean Upload Processing Time ts_{Ln}

This is the mean time required for the local network in Figure 6.1 to process a packet of size d, using the local network processing rate μ_{Ln}, when the user computer requests a Web page to be uploaded, as given in Equation 6.4:

$$\begin{aligned} ts_{Ln} &= \frac{d}{\mu_{Ln}}, \\ ts_{Ln} &= \frac{1000 \text{ bits}}{100 \text{ Mbit/s}} = 0.000010 \text{ seconds}. \end{aligned} \tag{6.4}$$

Mean Upload Wait Time tw_{Ln}

This is the mean time a packet has to wait to be processed in the local network queue in Figure 6.1 when the user computer request for a Web page is uploaded, as given by the *packet upload time*, from Equation 6.1, and the local network *upload processing time*, computed in Equation 6.4:

$$\begin{aligned} tw_{Ln} &= t_{uc} - ts_{Ln}, \\ tw_{Ln} &= 0.003049 - 0.000010 \text{ seconds} = 0.003039 \text{ seconds}. \end{aligned} \tag{6.5}$$

Mean Download Processing Time ts_{Lnd}

This is the mean time, computed in Equation 6.6, required to process a Web page, of size p, processed at the local network processing rate μ_{Ln}, in the local network, when a Web page is downloaded in Figure 6.1:

$$\begin{aligned} ts_{Lnd} &= \frac{p}{\mu_{Ln}}, \\ ts_{Lnd} &= \frac{96{,}928 \text{ bits}}{100 \text{ Mbit/s}} = 0.00096928 \text{ seconds}. \end{aligned} \tag{6.6}$$

Mean Download Wait Time tw_{Lnd}

This is the mean time, computed in Equation 6.7, a Web page has to wait to be processed in the local network queue in Figure 6.1, when a Web page is downloaded, as given by the Web page download time, from Equation 6.2, and the local network processing time, computed in Equation 6.6:

$$\begin{aligned} tw_{Lnd} &= t_{ucd} - ts_{Lnd}, \\ tw_{Lnd} &= 0.018461 - 0.00096928 \text{ seconds} = 0.001749 \text{ seconds}. \end{aligned} \tag{6.7}$$

Mean Number of Packet Bits Being Processed in the Upload Direction ns_{Ln}

Equation 6.8 is the *probability of the local network being busy* from Equation 6.3 times the packet of length d, as shown in Figure 6.1:

$$\begin{aligned} ns_{Ln} &= \left(\frac{\lambda_{uc} + \lambda_{ws}}{\mu_{Ln}}\right) d, \\ ns_{Ln} &= \left(\frac{0.328 + 5.25 \text{ Mbit/s}}{100 \text{ Mbit/s}}\right)(1000 \text{ bits}) = 55.8 \text{ bits}. \end{aligned} \tag{6.8}$$

Mean Number of Packet Bits Waiting to be Processed in the Upload Direction nw_{Ln}

If the result computed in Equation 6.8 is subtracted from the packet length, the *number of packet bits waiting to be processed* can be computed in Equation 6.9, as shown in Figure 6.1:

$$\begin{aligned} nw_{Ln} &= d - ns_{Ln}, \\ nw_{Ln} &= 1000 - 55.8 \text{ bits} = 944.20 \text{ bits}. \end{aligned} \tag{6.9}$$

Mean Number of Web Page Bits Being Processed in the Download Direction ns_{Lnd}

In Equation 6.10, we multiply the Web page size p by the probability of the local network server being busy from Equation 6.3, as shown in Figure 6.1:

$$\begin{aligned} ns_{Lnd} &= \left(\frac{\lambda_{uc} + \lambda_{ws}}{\mu_{Ln}}\right)(p), \\ ns_{Lnd} &= \left(\frac{0.328 + 5.25 \text{ Mbit/s}}{100 \text{ Mbit/s}}\right)(96{,}928 \text{ bits}) = 5407 \text{ bits}. \end{aligned} \tag{6.10}$$

Mean Number of Web Page Bits Having to Wait to Be Processed in the Download Direction nw_{Lnd}

Equation 6.11 computes the number of Web page bits that are held up in the local network queue waiting to be processed for download that is equal to Web page size p minus the result from Equation 6.10, as shown in Figure 6.1:

$$\begin{aligned} nw_{Lnd} &= p - ns_{Lnd}, \\ nw_{Lnd} &= 96{,}928 - 5407 \text{ bits} = 91{,}521 \text{ bits}. \end{aligned} \tag{6.11}$$

Local Network Router

Single-server equations apply for the local network router, shown in Figure 6.1, because only one local network router server is required in order for the probability of queue being busy <1. Also, note that the local router equations are applied twice—once for routing the Web page request in Figure 6.1 and again for routing the downloaded Web page in Figure 6.1.

Probability of Local Network Router Being Busy ρ_{Lr}

This probability is the ratio of the sum of the *packet upload rate* λ_{uc} and the *Web server download rate* λ_{ws} to the local network router *packet routing rate*, μ_{Lr}, computed in Equation 6.12, as shown in Figure 6.1:

$$\begin{aligned} \rho_{Lr} &= \frac{\lambda_{uc} + \lambda_{ws}}{\mu_{Lr}}, \\ \rho_{Lr} &= \frac{0.328 + 5.25 \text{ Mbit/s}}{54 \text{ Mbit/s}} = 0.1033. \end{aligned} \tag{6.12}$$

Mean Upload Processing Time ts_{Lr}

This is the mean time required for the local network to process a packet, of size d, for routing in Figure 6.1, as given in Equation 6.13:

$$\begin{aligned} ts_{Lr} &= \frac{d}{\mu_{Lr}}, \\ ts_{Lr} &= \frac{1000 \text{ bits}}{54 \text{ Mbit/s}} = 18.52 \text{ μs}. \end{aligned} \tag{6.13}$$

Mean Upload Wait Time tw_{Lr}

This is the *mean time a packet has to wait to be routed* in the local network router queue in Figure 6.1, computed in Equation 6.14, as given by the *packet upload time*, from Equation 6.1, and the local network router *routing time*, computed in Equation 6.13:

$$\begin{aligned} tw_{Lr} &= t_{uc} - ts_{Lr}, \\ tw_{Lr} &= 3049 - 18.52 \text{ μs} = 3030.48 \text{ μs}. \end{aligned} \tag{6.14}$$

Mean Download Processing Time ts_{Lrd}

This is the mean time required to route a Web page of size p from a Web server to the user computer, as given in Equation 6.15 and shown in Figure 6.1:

$$\begin{aligned} ts_{Lrd} &= \frac{p}{\mu_{Lr}}, \\ ts_{Lrd} &= \frac{96{,}928 \text{ bits}}{54 \text{ Mbit/s}} = 0.001750 \text{ seconds}. \end{aligned} \tag{6.15}$$

Mean Download Wait Time tw_{Lrd}

This is the mean time, computed in Equation 6.16, a Web page has to wait before it can be routed in the download direction to the user computer, as shown in Figure 6.1. Equation 6.16 uses the Web page download time, t_{ucd}, computed in Equation 6.2 and the download processing time computed in Equation 6.15:

$$\begin{aligned} tw_{Lrd} &= t_{ucd} - ts_{Lrd}, \\ tw_{Lrd} &= 0.018462 - 0.001750 \text{ seconds} = 0.016712 \text{ seconds}. \end{aligned} \tag{6.16}$$

Mean Number of Packet Bits Being Processed in the Upload Direction for Routing ns_{Lr}

Equation 6.17 is equivalent to the product of the *probability of the local network router being busy*, from Equation 6.12, and the *packet size* d, as shown in Figure 6.1:

$$\begin{aligned} ns_{Lr} &= \frac{(\lambda_{uc})(d)}{\mu_{Lr}}, \\ ns_{Lr} &= \frac{(0.328 \text{ Mbit/s})(1000 \text{ bits})}{54 \text{ Mbit/s}} = 6.07 \text{ bits}. \end{aligned} \tag{6.17}$$

Mean Number of Packet Bits Waiting to be Processed in the Upload Direction for Routing nw_{Lr}

If the result computed in Equation 6.17 is subtracted from the packet length, the *number of packet bits waiting to be processed for routing* can be computed in Equation 6.18, as shown in Figure 6.1:

$$\begin{aligned} nw_{Lr} &= d - ns_{Lr}, \\ nw_{Lr} &= 1000 - 6.07 \text{ bits} = 993.93 \text{ bits}. \end{aligned} \tag{6.18}$$

Mean Number of Web Page Bits Being Processed for Routing in the Download Direction ns_{Lrd}

In Equation 6.19, compute the number of Web page bits being processed in the download direction by utilizing the Web page download rate λ_{ws}, Web page size p, and local network router processing rate μ_{1r}, as shown in Figure 6.1:

$$\begin{aligned} ns_{Lrd} &= \frac{(\lambda_{ws})(p)}{\mu_{lr}}, \\ ns_{Lrd} &= \frac{(5.25\ \text{Mbit/s})(96,928\ \text{bits})}{54\ \text{Mbit/s}} = 9424\ \text{bits}. \end{aligned} \tag{6.19}$$

Mean Number of Web Page Bits Waiting to Processed for Routing in the Download Direction nw_{Lrd}

This computation is made by subtracting Equation 6.19 from the Web page size p, producing Equation 6.20, as shown in Figure 6.1:

$$\begin{aligned} nw_{Lrd} &= p - ns_{Lrd}, \\ nw_{Lrd} &= 96,928 - 9424\ \text{bits} = 87,504\ \text{bits}. \end{aligned} \tag{6.20}$$

Internet Router

Probability of Internet Router Being Busy ρ_{ir}

This probability is the ratio of the sum of the *packet upload rate* λ_{uc} and the *Web page download rate* λ_{ws} to the Internet router *packet routing rate*, $s\mu_{lr}$, as shown in Equation 6.21, where s = 2 is the number of Internet router servers. Whenever there are multiple servers involved, this fact must be reflected in the total service rate. The Internet routers are shown in Figures 6.1–6.3.

$$\begin{aligned} \rho_{ir} &= \frac{(\lambda_{uc} + \lambda_{ws})}{s\mu_{ir}}, \\ \rho_{ir} &= \frac{(0.328 + 5.25\ \text{Mbit/s})}{(2)(6.25\ \text{Mbit/s})} = 0.4462. \end{aligned} \tag{6.21}$$

Mean Upload Processing Time ts_{ir}

This is the mean time required for the Internet router to route a packet in the upload direction, as given in Equation 6.22, where again, the computation must account for s = 2 servers, as shown in Figure 6.1:

$$\begin{aligned} ts_{ir} &= \frac{d}{s\mu_{ir}}, \\ ts_{ir} &= \frac{1000\ \text{bits}}{(2)(6.250\ \text{Mbit/s})} = 0.000080\ \text{seconds}. \end{aligned} \tag{6.22}$$

Mean Upload Wait Time tw_{ir}

This is the *mean time a packet has to wait to be routed* in the Internet router queue in the upload direction, as given by the *packet upload time*, from Equation 6.1, and the Internet router *processing time*, computed in Equation 6.23, as shown in Figure 6.1:

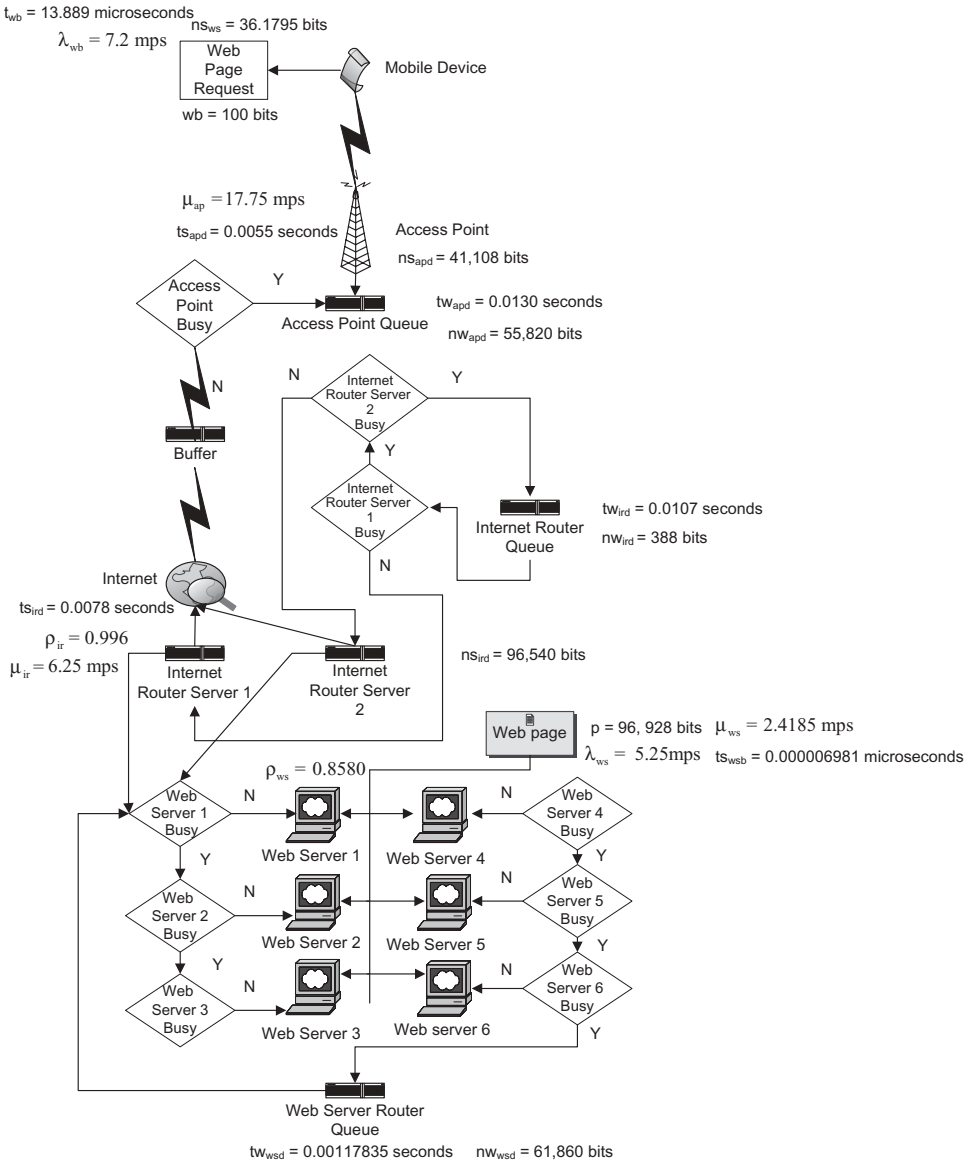


Figure 6.2 Present wireless queuing model (upload).

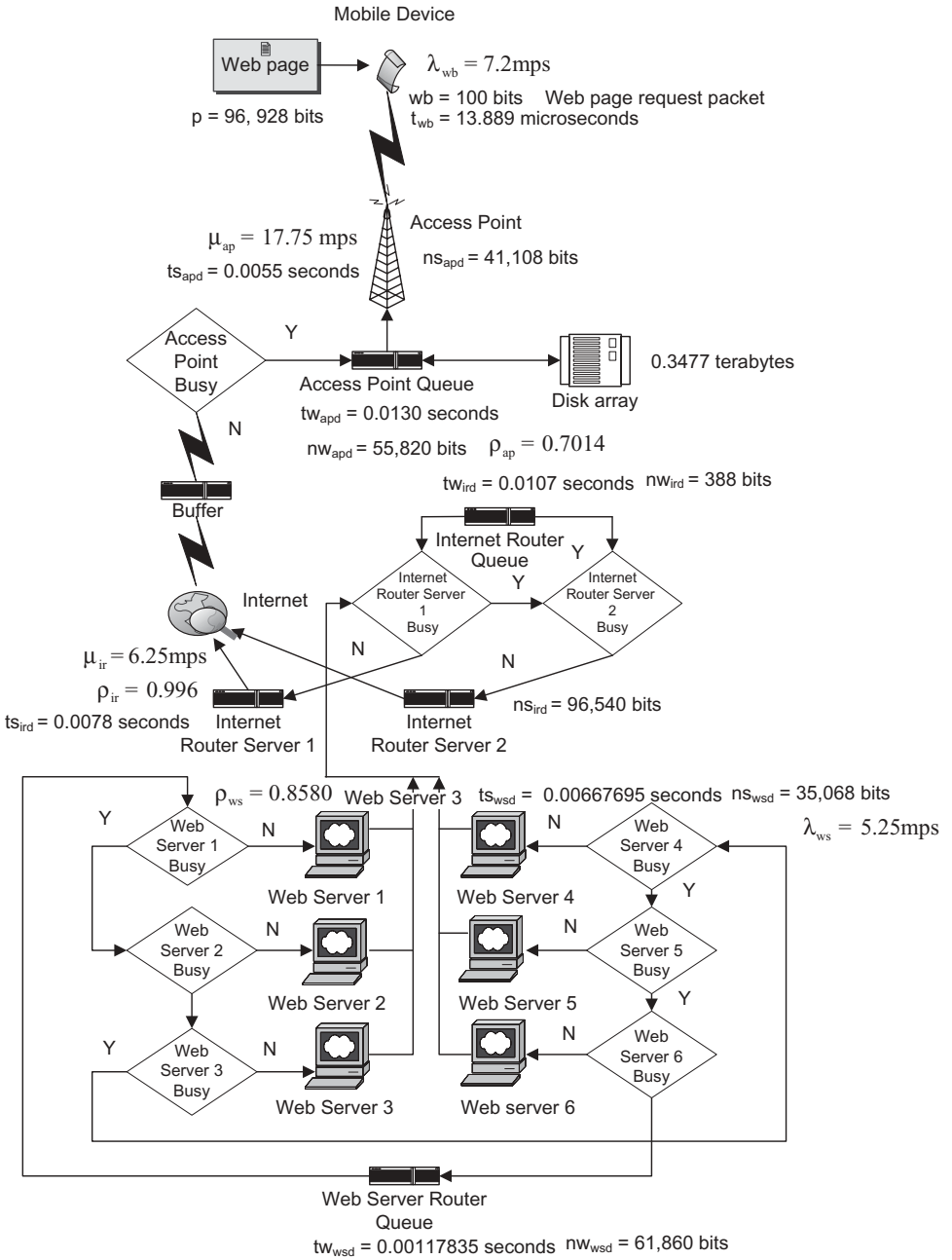


Figure 6.3 Present wireless queuing model (download).

$$\begin{aligned} tw_{ir} &= t_{uc} - ts_{ir}, \\ tw_{ir} &= 0.003049 - 0.000080 \text{ seconds} = 0.002969 \text{ seconds}. \end{aligned} \tag{6.23}$$

Mean Download Processing Time ts_{ird}

This is the mean time required for the Internet router to process a Web page for routing, using the Web page size p and the processing rate of s = 2 servers $s\mu_{ir}$ given in Equation 6.24, as shown in Figure 6.1:

$$\begin{aligned} ts_{ird} &= \frac{p}{s\mu_{ir}}, \\ ts_{ird} &= \frac{96{,}928 \text{ bits}}{(2)(6.250 \text{ Mbit/s})} = 0.0078 \text{ seconds}. \end{aligned} \tag{6.24}$$

Mean Download Wait Time tw_{ird}

Equation 6.25 computes the mean time a Web page, downloaded in a time t_{ucd}. must wait for routing, using the processing time ts_{ird}, computed in Equation 6.24, as shown in Figure 6.1:

$$\begin{aligned} tw_{ird} &= t_{ucd} - ts_{ird}, \\ tw_{ird} &= 0.018461 - 0.0078 \text{ seconds} = 0.0107 \text{ seconds}. \end{aligned} \tag{6.25}$$

Mean Number of Packet Bits Being Processed for Upload Routing ns_{ir}

Equation 6.26 is equivalent to the product of the *probability of the Internet router being busy*, from Equation 6.21, and the *packet size* d, as shown in Figure 6.1:

$$\begin{aligned} ns_{ir} &= \frac{(\lambda_{uc} + \lambda_{ws})(d)}{s\mu_{ir}}, \\ ns_{ir} &= \frac{(0.328 + 5.25 \text{ Mbit/s})(1000 \text{ bits})}{(2)(6.25 \text{ Mbit/s})} = 446.24 \text{ bits}. \end{aligned} \tag{6.26}$$

Mean Number of Packet Bits Waiting to be Processed for Routing in the Upload Direction nw_{ir}

If the result computed in Equation 6.26 is subtracted from the packet length d, the *number of packet bits waiting to be processed for routing in the upload direction* can be computed in Equation 6.27, as shown in Figure 6.1:

$$\begin{aligned} nw_{ir} &= d - ns_{ir}, \\ nw_{ir} &= 1000 - 446.24 \text{ bits} = 553.76 \text{ bits}. \end{aligned} \tag{6.27}$$

Mean Number of Web Page Bits Being Processed for Download Routing ns_{ird}

Equation 6.28 is equivalent to the product of the *probability of the Internet router being busy*, from Equation 6.21, and the Web page size p, for s = 2 servers as shown in Figure 6.1:

$$\begin{aligned} ns_{ird} &= \frac{(\lambda_{uc} + \lambda_{ws})(p)}{s\mu_{ir}}, \\ ns_{ird} &= \frac{(0.328 + 5.25 \text{ Mbit/s})(96,928 \text{ bits})}{(2)(6.25 \text{ Mbit/s})} = 43,253 \text{ bits}. \end{aligned} \tag{6.28}$$

Mean Number of Web Page Bits Waiting for Download Routing nw_{ird}

Equation 6.29 is computed by subtracting the result in Equation 6.28 from the Web page size p, as shown in Figure 6.1:

$$\begin{aligned} nw_{ird} &= p - ns_{ird}, \\ nw_{ird} &= 96,928 - 43,253 \text{ bits} = 53,675 \text{ bits}. \end{aligned} \tag{6.29}$$

Domain Name System (DNS)

Only the upload equations are computed because once the user computer has obtained an IP address from the DNS, it can be used for downloading a Web page. Note that because the DNS computations are mean values, a fraction of a packet (i.e., number of bits) would be computed for name-to-IP address translation, as opposed to an entire packet, which is the case in actual translations.

Probability of Domain Name System Being Busy ρ_{ns}

This probability is the ratio of *packet arrival rate* λ_{uc} to the DNS *user computer name to IP address translation rate*, $s\mu_{ns}$, shown in Equation 6.30, where s = 3 is the required number of DNS servers. This probability and the DNS servers are shown in Figure 6.1:

$$\begin{aligned} \rho_{ns} &= \frac{\lambda_{uc}}{s\mu_{ns}}, \\ \rho_{ns} &= \frac{0.328 \text{ Mbit/s}}{(3)(0.143 \text{ Mbit/s})} = 0.7646. \end{aligned} \tag{6.30}$$

Mean Processing Time ts_{ns}

This is the mean time required for a DNS in Figure 6.1 to do an address translation for a packet of size d, as given in Equation 6.31, where again, the computation must account for multiple servers:

$$ts_{ns} = \frac{d}{s\mu_{ns}},$$
$$ts_{ns} = \frac{1000 \text{ bits}}{(3)(143{,}000 \text{ bits/s})} = 0.0023 \text{ seconds}. \tag{6.31}$$

Mean Wait Time tw_{ns}

This is the *mean time a user computer Web page request packet* must wait in the DNS queue prior to name-to-IP address translation in Figure 6.1, computed in Equation 6.32, as given by the *packet upload time* from Equation 6.1, and the DNS *processing time*, as computed in Equation 6.31:

$$tw_{ns} = t_{uc} - ts_{ns},$$
$$tw_{ns} = 0.003049 - 0.0023 \text{ seconds} = 0.000749 \text{ seconds}. \tag{6.32}$$

Mean Number of Packet Bits Being Processed for Name Translation ns_{ns}

Equation 6.33 is equivalent to the product of the *probability of the DNS being busy*, from Equation 6.30, and the *packet size* d, as shown in Figure 6.1:

$$ns_{ns} = \frac{(\lambda_{uc})(d)}{s\mu_{ns}},$$
$$ns_{ns} = \frac{(0.328 \text{ Mbit/s})(1000 \text{ bits})}{(3)(0.143 \text{ Mbit/s})} = 764.57 \text{ bits}. \tag{6.33}$$

Mean Number of Packet Bits Having to Wait for Name Translation nw_{ns}

This quantity, computed in Equation 6.34, is the difference between packet size d and the number of bits being processed by the DNS from Equation 6.33, as shown in Figure 6.1:

$$nw_{ns} = d - ns_{ns},$$
$$nw_{ns} = 1000 - 764.57 = 235.43 \text{ bits}. \tag{6.34}$$

Web Server Processing

Some quantities involving Web pages were previously computed. This section provides computations for *Web server processing* (e.g., probability of server busy) for both the wired system (Fig. 6.1) and the wireless system upload Web page request (Fig. 6.2), and the wireless download Web page response (Fig. 6.3).

Probability of Web Server Being Busy ρ_{ws}

For wired systems, this probability is the ratio of the sum of the *Web page wired system request packet upload rate*, λ_{uc}, and the *Web page download rate*, λ_{ws}, to the

Web page processing rate, $s\mu_{ws}$, as shown in Figure 6.1. For wireless systems, this probability is the sum of the *Web page wireless system request rate*, λ_{wb}, and the *Web page download rate*, λ_{ws}, to the *Web page processing rate*, $s\mu_{ws}$, shown in Equation 6.35, where s is the number of Web servers. Note that in the case of the wireless system, twice as many servers (six) are required to maintain queue stability (i.e., $\rho_{ws} < 1.0$) than in the case of the wired system (three), due to the high *Web page wireless system request rate*, λ_{wb}, as shown in Figures 6.2 and 6.3:

$$\begin{aligned}
\rho_{ws} &= \frac{(\lambda_{uc} + \lambda_{ws})}{s\mu_{ws}} \text{ (wired system)}, \\
\rho_{ws} &= \frac{(\lambda_{wb} + \lambda_{ws})}{s\mu_{ws}} \text{ (wireless system)}, \\
\rho_{ws} &= \frac{(0.328 + 5.25 \text{ Mbit/s})}{(3)(2.4185 \text{ Mbit/s})} = 0.7688 \text{ (wired system)}, \\
\rho_{ws} &= \frac{(7.20 + 5.25 \text{ Mbit/s})}{(6)(2.4185 \text{ Mbit/s})} = 0.8580 \text{ (wireless system)}.
\end{aligned} \tag{6.35}$$

Wired and Wireless Systems Mean Upload Processing Time ts_{ws}

This is the mean time required for the Web servers to process requests for Web pages received from the user computers, as computed in Equation 6.36, for wired system packet size d and number of servers s = 3 (Fig. 6.1) and wireless system packet size wb and number of servers s = 6 (Fig. 6.2):

$$\begin{aligned}
&ts_{ws} = \frac{d}{s\mu_{ws}}, ts_{wsb} = \frac{wb}{s\mu_{ws}}, \\
&ts_{ws} = \frac{1000 \text{ bits}}{(3)(2,418,500 \text{ bits/s})} = 0.000138 \text{ seconds (three-server wired system)}, \\
&ts_{wsb} = \frac{100 \text{ bits}}{(6)(2.4185 \text{ Mbit/s})} = 0.000006891 \,\mu\text{s (six-server wireless system)}.
\end{aligned} \tag{6.36}$$

Wired and Wireless Systems Mean Download Processing Time ts_{wsd}

This is the mean time required by the Web servers to provide the Web pages requested by wired user computers (Fig. 6.1) and wireless mobile devices (Fig. 6.3), as computed in Equations 6.37 and 6.38, respectively, for Web pages of size p, again accounting for multiple servers:

$$\begin{aligned}
ts_{wsd} &= \frac{p}{s\mu_{ws}} = \frac{96,928 \text{ bits}}{3 * 2,418,500 \text{ bits/s for three-server wired system}} \\
&= 0.013359 \text{ seconds},
\end{aligned} \tag{6.37}$$

$$\begin{aligned}
ts_{wsd} &= \frac{p}{s\mu_{ws}} = \frac{96,928 \text{ bits}}{6 * 2,418,500 \text{ bits/s for six-server wireless system}} \\
&= 0.00667695 \text{ seconds}.
\end{aligned} \tag{6.38}$$

Wired and Wireless Systems Mean Upload Wait Time $\mathbf{tw_{ws}}$

This is the mean time a wired system user computer request for a Web page, and a wireless system mobile device request for a Web page, must wait in a Web server queue to be processed in Figures 6.1 and 6.2, respectively, as computed by Equation 6.39, using the Web page upload time from Equation 6.1 and the wired and wireless processing times from Equation 6.36:

$$\begin{aligned}&\text{Wired system: } tw_{ws} = t_{uc} - ts_{ws}\text{; wireless system: } tw_{wsb} = t_{uc} - ts_{wsb},\\ &tw_{ws} = 0.003049 - 000138 \text{ seconds} = 0.002911 \text{ seconds (wired system)},\\ &tw_{wsb} = 0.003049 - 0.000006891 \text{ seconds} = 0.003042 \text{ seconds (wireless system)}.\end{aligned} \tag{6.39}$$

Wired and Wireless Systems Mean Download Wait Time $\mathbf{tw_{wsd}}$

This is the mean time Web pages must wait in the Web server queue prior to being downloaded to user computers (Fig. 6.1) and mobile devices (Fig. 6.3), using the Web page download time from Equation 6.2 and the mean download processing times from Equations 6.37 and 6.38:

$$\begin{aligned}&tw_{wsd} = t_{ucd} - ts_{wsd},\\ &tw_{wsd} = 0.018462 - 0.013359 \text{ seconds} = 0.005103 \text{ seconds (wired system)},\\ &tw_{wsd} = 0.018462 - 0.00667695 \text{ seconds} = 0.00117835 \text{ seconds (wireless system)}.\end{aligned} \tag{6.40}$$

Wired System Mean Number of Web Page Request Bits Being Processed for Upload $\mathbf{ns_{ws}}$

Equation 6.41 is equivalent to the product of the *probability of a Web server being busy* (Eq. 6.35), Web page packet request size d, and s = 3 servers, in the wired system in Figure 6.1:

$$\begin{aligned}ns_{ws} &= \frac{(\lambda_{ws})(d)}{s\mu_{ws}},\\ ns_{ws} &= \frac{(5.25 \text{ Mbit/s})(1000 \text{ bits})}{(3)(2.4185 \text{ Mbit/s})} = 723.59 \text{ bits}.\end{aligned} \tag{6.41}$$

Wireless System Mean Number of Web Page Request Bits Being Processed for Upload $\mathbf{ns_{wsb}}$

Equation 6.42 is equivalent to the product of the *probability of a Web server being busy* (Eq. 6.35), and the wireless *Web page packet request size* wb, using s = 6 servers, in Figure 6.2:

$$ns_{wsb} = \frac{(\lambda_{ws})(wb)}{s\mu_{ws}} = \frac{(5.25 \text{ Mbit/s})(100 \text{ bits})}{(6)(2.4185 \text{ Mbit/s})} = 36.1795 \text{ bits}. \tag{6.42}$$

Wired and Wireless System Mean Number of Web Page Bits Being Processed for Download $\mathbf{ns_{wsd}}$

Equation 6.43 is equivalent to the product of the *probability of a Web server being busy* (Eq. 6.35) and the Web page size p. This equation applies to both the wired and wireless systems in Figures 6.1 and 6.2, respectively, noting the different number of servers used in the wired and wireless systems:

$$\begin{aligned} ns_{wsd} &= \frac{(\lambda_{ws})(p)}{s\mu_{ws}}, \\ ns_{wsd} &= \frac{(5.25\ \text{Mbit/s})(96{,}928\ \text{bits})}{(3)(2.4185\ \text{Mbit/s})} = 70{,}136\ \text{bits (wired system three servers)}, \\ ns_{wsd} &= \frac{(5.25\ \text{Mbit/s})(96{,}928\ \text{bits})}{(6)(2.4185\ \text{Mbit/s})} = 35{,}068\ \text{bits (wireless system, six servers)}. \end{aligned} \tag{6.43}$$

Wired System Mean Number of Web Page Bits Having to Wait for Upload Processing by a Web Server $\mathbf{nw_{ws}}$

This quantity is the difference between Web page packet request size d and the number of bits being processed by a Web server in the upload direction, which is computed in Equation 6.44. This equation applies to the wired system (Fig. 6.1):

$$\begin{aligned} nw_{ws} &= d - ns_{ws}, \\ nw_{ws} &= 1000 - 723.59\ \text{bits} = 276.41\ \text{bits}. \end{aligned} \tag{6.44}$$

Wireless System Mean Number of Web Page Bits Having to Wait for Upload Processing by a Web Server $\mathbf{nw_{wbd}}$

Equation 6.45 is the difference between wireless Web page packet request size wb and the number of bits being processed by a Web server in the upload direction, which is computed in Equation 6.41. This equation applies to the wireless system (Fig. 6.2):

$$\begin{aligned} nw_{wbd} &= wb - ns_{wsd}, \\ nw_{wbd} &= 100 - 72.359\ \text{bits} = 27.641\ \text{bits}. \end{aligned} \tag{6.45}$$

Wired and Wireless System Mean Number of Web Page Bits Having to Wait for Download Processing by a Web Server $\mathbf{nw_{wsd}}$

This quantity is the difference between Web page size p and the number of bits being processed by a Web server, which is computed in Equation 6.46. This equation applies to both the wired (Fig. 6.1) and wireless systems (Fig. 6.3), but note the different results due to the difference in processing time caused by difference in number of servers:

$$\begin{aligned} nw_{wsd} &= p - nc_{ws}, \\ nw_{wsd} &= 96{,}928 - 70{,}136 = 26{,}792\ \text{bits (wired system)}, \\ nw_{wsd} &= 96{,}928 - 35{,}068 = 61{,}860\ \text{bits (wireless system)}. \end{aligned} \tag{6.46}$$

Present Internet Wireless Backbone

In this section, the important contribution of data traffic generated by handheld devices, communicating via the wireless backbone with the Internet, is assessed. The importance of this data traffic, and its attendant storage requirement, can be seen by examining the high traffic rates documented in Table 6.1. See Figure 6.2 (upload) and Figure 6.3 (download) for the logic sequences.

Mobile Device

Mean Wireless Packet Upload Time t_{wb}

This is the mean time required to upload a wireless backbone packet of size wb at an upload rate of λ_{wb} in Figure 6.2:

$$\begin{aligned} t_{wb} &= \frac{wb}{\lambda_{wb}}, \\ t_{wb} &= \frac{100 \text{ bits}}{7.2 \text{ Mbit/s}} = 13.889 \text{ μs}. \end{aligned} \tag{6.47}$$

Access Point

Probability of Access Point Being Busy ρ_{ap}

This probability is the ratio of the *sum of wireless packet upload rate* λ_{wb} and *Web page download rate* λ_{ws} to the *access point processing rate* μ_{ap} in Figures 6.2 and 6.3:

$$\begin{aligned} \rho_{ap} &= \frac{(\lambda_{wb} + \lambda_{ws})}{\mu_{ap}}, \\ \rho_{ap} &= \frac{(7.2 + 5.25 \text{ Mbit/s})}{17.75 \text{ Mbit/s}} = 0.7014. \end{aligned} \tag{6.48}$$

Mean Access Point Processing Time in the Upload Direction ts_{ap}

This is the mean time required for the access point to process a wireless backbone packet of size wb at a processing rate of μ_{ap} in the upload direction in Figure 6.2:

$$\begin{aligned} ts_{ap} &= wb / \mu_{ap}, \\ ts_{ap} &= 100 \text{ bits}/17.75 \text{ Mbit/s} = 5.6338 \text{ μs}. \end{aligned} \tag{6.49}$$

Mean Access Point Queue Wait Time in the Upload Direction tw_{ap}

This is the mean time a wireless backbone packet—transmitted in the upload direction—must wait to be processed by the access point in Figure 6.2. Equation

6.50 uses the wireless packet upload time from Equation 6.47 and the access point processing time from Equation 6.49:

$$\begin{aligned} tw_{ap} &= t_{wb} - ts_{ap}, \\ tw_{ap} &= 13.889 - 5.6338\ \mu s = 8.2552\ \mu s. \end{aligned} \tag{6.50}$$

Mean Access Point Processing Time in the Download Direction ts_{apd}

This is the mean time required for the access point to process the Web page in the download direction, using the Web page of size p and the access point processing rate μ_{ap} in Equation 6.51, as shown in Figure 6.3:

$$\begin{aligned} ts_{apd} &= p/\mu_{ap}, \\ ts_{apd} &= 96{,}928\ \text{bits}/17.75\ \text{Mbit/s} = 0.0055\ \text{seconds}. \end{aligned} \tag{6.51}$$

Mean Access Point Queue Wait Time in the Download Direction tw_{apd}

This mean time, computed in Equation 6.52, using Web page download time t_{ucd} and the processing time computed in Equation 6.51, is the time the Web page must wait in the access point queue prior to being processed, as shown in Figure 6.3:

$$\begin{aligned} tw_{apd} &= t_{ucd} - ts_{apd}, \\ tw_{apd} &= 0.018461 - 0.0055\ \text{seconds} = 0.0130\ \text{seconds}. \end{aligned} \tag{6.52}$$

Mean Number of Wireless Packet Bits Being Processed in the Upload Direction by the Access Point ns_{ap}

This computation is made in Equation 6.53 by computing the probability of the access point being busy $(\lambda_{wb} + \lambda_{ws})/\mu_{ap}$, from Equation 6.48, and multiplying it by the size of the wireless packet size wb, as shown in Figure 6.2:

$$\begin{aligned} ns_{ap} &= \frac{(\lambda_{wb} + \lambda_{ws})(wb)}{\mu_{ap}}, \\ ns_{ap} &= \frac{(7.2 + 5.25\ \text{Mbit/s})(100\ \text{bits})}{17.75\ \text{Mbit/s}} = 70.14\ \text{bits}. \end{aligned} \tag{6.53}$$

Mean Number of Wireless Packet Bits Waiting to be Processed in the Upload Direction by the Access Point nw_{ap}

Equation 6.54 is computed by subtracting the number of wireless packet bits being processed in the upload direction, computed in Equation 6.53, from the wireless packet size wb, as shown in Figure 6.2:

$$\begin{aligned} nw_{ap} &= wb - ns_{ap}, \\ nw_{ap} &= 100 - 70.14\ \text{bits} = 29.86\ \text{bits}. \end{aligned} \tag{6.54}$$

Mean Number of Web Page Bits Being Processed in the Download Direction by the Access Point $\mathbf{ns_{apd}}$

This computation is made by computing the probability of the access point being busy $(\lambda_{wb} + \lambda_{ws})/\mu_{ap}$, derived from Equation 6.48, and multiplying it by the size of the Web page size p, as shown in Figure 6.3:

$$\begin{aligned} ns_{apd} &= \frac{(\lambda_{wb} + \lambda_{ws})(p)}{\mu_{ap}}, \\ ns_{apd} &= \frac{(7.2 + 0.328 \text{ Mbit/s})(96{,}928 \text{ bits})}{17.75 \text{ Mbit/s}} = 41{,}108 \text{ bits}. \end{aligned} \tag{6.55}$$

Mean Number of Web Page Bits Having to Wait for Processing in the Download Direction by the Access Point $\mathbf{nw_{apd}}$

Equation 6.56 computes the number of Web page bits having to wait to be processed by the access point by subtracting the Web page bits being processed in Equation 6.55 from the Web page size p, as shown in Figure 6.3:

$$\begin{aligned} nw_{apd} &= p - ns_{apd}, \\ nw_{apd} &= 96{,}928 - 41{,}108 \text{ bits} = 55{,}820 \text{ bits}. \end{aligned} \tag{6.56}$$

Internet Router

The Internet router computations in the succeeding sections are shown in Figure 6.2 (upload) and Figure 6.3 (download).

Probability of Internet Router Being Busy Processing Wireless Packet ρ_{ir}

Equation 6.57 expresses the probability that the Internet router will be occupied processing the wireless Web page request packet, transmitted at a rate λ_{wb} in the upload direction, plus being busy when the Web page is downloaded at a rate λ_{ws}, and the router processes at a rate μ_{ir}, and the number of routers s = 2, as shown in Figures 6.2 and 6.3:

$$\begin{aligned} \rho_{ir} &= \frac{(\lambda_{wb} + \lambda_{ws})}{(s)(\mu_{ir})}, \\ \rho_{ir} &= \frac{(7.2 + 5.25 \text{ Mbit/s})}{(2)(6.25 \text{ Mbit/s})} = 0.996. \end{aligned} \tag{6.57}$$

Mean Time a Wireless Packet Spends Being Processed for Routing by the Internet Router in the Upload Direction $\mathbf{ts_{ir}}$

Equation 6.58 computes this mean time by dividing the wireless packet of size wb by the Internet router processing rate $(s)(\mu_{ir})$ as shown in Figure 6.2:

$$\begin{aligned} ts_{ir} &= \frac{wb}{(s)(\mu_{ir})}, \\ ts_{ir} &= \frac{100 \text{ bits}}{(2)(6.25 \text{ Mbit/s})} = 8 \ \mu s. \end{aligned} \tag{6.58}$$

Mean Time a Wireless Packet Spends Waiting to be Processed for Routing by the Internet Router in the Upload Direction tw_{ir}

Equation 6.59 is computed by subtracting the processing time computed in Equation 6.58 from the wireless packet upload time t_{wb}, as shown in Figure 6.2:

$$\begin{aligned} tw_{ir} &= t_{wb} - ts_{ir}, \\ tw_{ir} &= t_{wb} - ts_{ir} = 13.889 - 8 \ \mu s = 5.889 \ \mu s. \end{aligned} \tag{6.59}$$

Mean Time a Web Page Requested by a Wireless Packet Spends Being Processed for Routing by the Internet Router in the Download Direction ts_{ird}

Equation 6.60 computes the mean processing time by dividing the Web page of size p by the Internet router processing rate $(s)(\mu_{ir})$,as shown in Figure 6.3:

$$\begin{aligned} ts_{ird} &= \frac{p}{(s)(\mu_{ir})}, \\ ts_{ird} &= \frac{96,928 \text{ bits}}{(2)(6.25 \text{ Mbit/s})} = 0.0078 \text{ seconds}. \end{aligned} \tag{6.60}$$

Mean Time a Wireless Packet Spends Waiting to be Processed for Routing by the Internet Router in the Download Direction tw_{ird}

This mean time is computed by subtracting the processing time, computed in Equation 6.60, from the Web page download time t_{ucd}, as shown in Figure 6.3:

$$\begin{aligned} tw_{ird} &= t_{ucd} - ts_{ird}, \\ tw_{ird} &= 0.018461 - 0.0078 \text{ seconds} = 0.0107 \text{ seconds}. \end{aligned} \tag{6.61}$$

Mean Number of Wireless Packet Bits Being Routed in the Upload Direction ns_{ir}

This quantity is computed in Equation 6.62 by multiplying the probability of the Internet router being busy, from Equation 6.57, by the wireless packet size wb, as shown in Figure 6.2:

$$\begin{aligned} ns_{ir} &= \frac{(\lambda_{wb} + \lambda_{ws})(wb)}{(s)(\mu_{ir})}, \\ ns_{ir} &= \frac{(7.2 + 5.25 \text{ Mbit/s})(100 \text{ bits})}{(2)(6.25 \text{ Mbit/s})} = 99.6 \text{ bits}. \end{aligned} \tag{6.62}$$

Mean Number of Wireless Packet Bits Having to Wait for Routing in the Upload Direction by the Internet Router nw_{ir}

This quantity is computed in Equation 6.63 by subtracting the number of bits being processed, computed in Equation 6.62, from the wireless packet size wb, as shown in Figure 6.2:

$$nw_{ir} = wb - ns_{ir} = 100 - 99.6 \text{ bits} = 0.4 \text{ bits} \tag{6.63}$$

Mean Number of Web Page Bits Being Processed in the Download Direction by the Internet Router ns_{ird}

Equation 6.64 is computed by multiplying the probability of the Internet router being busy, derived from Equation 6.57, by the Web page size p, as shown in Figure 6.3:

$$\begin{aligned} ns_{ird} &= \frac{(\lambda_{wb} + \lambda_{ws})}{(s)(\mu_{ir})}(p), \\ ns_{ird} &= \frac{(7.2 + 5.25 \text{ Mbit/s})}{(2)(6.25 \text{ Mbit/s})}(96,928 \text{ bits}) = 96,540 \text{ bits}. \end{aligned} \tag{6.64}$$

Mean Number of Web Page Bits Having to Wait for Routing in the Download Direction by the Internet Router nw_{ird}

This quantity is computed in Equation 6.65 by subtracting the number of Web page bits being processed for routing, computed in Equation 6.66, from the Web page size p, as shown in Figure 6.3:

$$\begin{aligned} nw_{ird} &= p - ns_{ird}, \\ nw_{ird} &= 96,928 - 96,540 \text{ bits} = 388 \text{ bits}. \end{aligned} \tag{6.65}$$

Domain Name System (DNS)

Only upload equations are computed because once the mobile device in Figure 6.2 has obtained an IP address from the DNS, it can be used for downloading a Web page. Because the wireless upload rate of the mobile device in Figure 6.3, $\lambda_{wb} = 7.2$ Mbit/s, is so much faster than the DNS processing rate, $\mu_{ns} = 0.143$ Mbit/s, a buffer is used at the access point to slow the mobile device rate to a value that the DNS can handle. This value, λ_{ns}, called the DNS input rate, is computed in Equation 6.66 by assuming that the probability of the DNS server busy $\rho_{ns} = 0.8$ (i.e., as long as a queue has $\rho \leq 0.8$, the queue is considered stable), and using the DNS processing rate μ_{ns}. Note that because the DNS computations are mean values, a fraction of a packet or Web page (i.e., number of bits) would be computed for translation:

$$\begin{aligned} \lambda_{ns} &= (\rho_{ns})(s)(\mu_{ns}), \\ \lambda_{ns} &= (0.8)(3)(0.143 \text{ Mbit/s}) = 0.3432 \text{ Mbit/s}. \end{aligned} \tag{6.66}$$

Mean Upload Time from Buffer of Wireless Packet to DNS ts_{ns}

Since the DNS input rate has been computed in Equation 6.66, now compute the wireless packet upload time, ts_{ns}, in Equation 6.67, using the wireless packet size wb. This is the upload time that results from using the DNS buffer in Figure 6.2:

$$\begin{aligned} ts_{ns} &= \frac{wb}{\lambda_{ns}}, \\ ts_{ns} &= \frac{100 \text{ bits}}{0.3432 \text{ Mbit/s}} = 291.38 \text{ μs}. \end{aligned} \tag{6.67}$$

Mean Processing Time for the DNS to Translate a Wireless Packet Name to an IP Address ts_{ns}

Equation 6.68 computes the name-to-IP address translation mean time for a wireless packet, using the wireless packet size, wb, number of DNS servers, s, and DNS processing rate, μ_{ns}:

$$\begin{aligned} ts_{ns} &= \frac{wb}{(s)(\mu_{ns})}, \\ ts_{ns} &= \frac{100 \text{ bits}}{(3)(0.143 \text{ Mbit/s})} = 233.10 \text{ μs}. \end{aligned} \tag{6.68}$$

Mean Time That a Wireless Packet Must Wait in the DNS Queue Prior to Name-to-IP Address Translation tw_{ns}

This mean time is computed in Equation 6.69 by subtracting the processing time, computed in Equation 6.68, from the upload time, computed in Equation 6.67, as shown in Figure 6.2.

$$\begin{aligned} tw_{ns} &= tu_{ns} - ts_{ns}, \\ tw_{ns} &= 291.38 - 233.10 \text{ μs} = 58.28 \text{ μs}. \end{aligned} \tag{6.69}$$

Mean Number of Wireless Packet Bits That Are Processed for Translation by the DNS ns_{ns}

The number of wireless packet bits that are processed for translation is computed in Equation 6.70 by multiplying the previously assumed *probability of the DNS being busy*, ρ_{ns}, by the wireless packet size wb. The result is shown in Figure 6.2:

$$\begin{aligned} ns_{ns} &= (\rho_{ns})(wb), \\ ns_{ns} &= (0.8)(100 \text{ bits}) = 80 \text{ bits}. \end{aligned} \tag{6.70}$$

Mean Number of Wireless Packet Bits That Are Waiting for Translation by the DNS nw_{ns}

The number of wireless bits waiting for translation is computed in Equation 6.71 by subtracting the number of bits being processed, computed in Equation 6.70, from the wireless packet size wb. The result is shown in Figure 6.2:

$$\begin{aligned} nw_{ns} &= wb - ns_{ns}, \\ nw_{ns} &= 100 - 80 \text{ bits} = 20 \text{ bits}. \end{aligned} \tag{6.71}$$

SUMMARY OF QUEUING MODEL COMPUTATIONS FOR PRESENT AND PROPOSED INTERNETS

Now that the queuing model computations have been made for the present wired and wireless Internets, it is time to summarize them in Table 6.2 (wired system) and in Table 6.1 (wireless system) in order to identify the critical performance variables. Also, the proposed wired and wireless Internet computations are shown in Tables 6.4 and 6.5, respectively. These computations are made in a later section entitled "Performance Analysis of Proposed Future Wired Internet," and are presented to contrast with the present systems mean value performance results.

Important results are shown at the bottom of each table: effective upload and download processing rates. The effective rate includes all the delays encountered in the various queues in the process of obtaining a Web page for the user computer or mobile device. These results are the following:

Table 6.2: Effective processing rate for present wired Internet: 1.7936 Mbit/s.

Table 6.3: Effective processing rate for present wireless Internet: 2.0697 Mbit/s.

Table 6.4: Effective processing rate for proposed wired Internet: 113.35 Mbit/s.

Table 6.5: Effective processing rate for proposed wireless Internet: 2.0853 Mbit/s.

Note that the effective rate of the proposed wired system is much greater than the effective rate of the present wired system because the former is not subject to local network, local router, and DNS delays. Also note that the effective rate of the proposed wireless system is marginally greater than the effective rate of the present wireless system because the former is not subject to the overhead introduced by the DNS. This reduction in overhead is not nearly as significant as the time saved by eliminating several components in the case of the proposed wired system.

SIMULATION QUEUING MODELS

Local Network: Present Wired System

In order to evaluate Internet performance, taking into account the variance in performance variables, such as the variability in upload and download times, a series of models is developed and designed to achieve this objective. The models include

Table 6.2 Summary of Queuing Computations for the Present Wired Internet System (Mean Values)

Variable	Component	Computation	Figure(s)
Packet upload time t_{uc}	User computer	0.003049 seconds	Figure 6.1
Web page download time t_{ucd}	Web server	0.018462 seconds	Figure 6.1
Probability of being busy ρ_{Ln}	Local network	0.055780	Figures 6.1 and 6.3
Upload processing time ts_{Ln}	Local network	0.000010 seconds	Figure 6.1
Upload wait time tw_{Ln}	Local network	0.003039 seconds	Figure 6.1
Download processing time ts_{Lnd}	Local network	0.0096928 seconds	Figure 6.1
Download wait time tw_{Lnd}	Local network	0.001749 seconds	Figure 6.1
Number of packet bits being processed in the upload direction ns_{Ln}	Local network	55.8 bits	Figure 6.1
Number of packet bits waiting to be processed in the upload direction nw_{Ln}	Local network	944.20 bits	Figure 6.1
Number of web page bits being processed in the download direction ns_{Lnd}	Local network	5407 bits	Figure 6.1
Number of web page bits having to wait to be processed in the download direction nw_{Lnd}	Local network	91,521 bits	Figure 6.1
Probability of being busy ρ_{Lr}	Local network router	0.1033	Figures 6.1 and 6.3
Upload processing time ts_{Lr}	Local network router	18.52 μs (0.00001852 seconds)	Figure 6.1
Upload wait time tw_{Ln}	Local network router	3039 μs (0.003039 seconds)	Figure 6.1
Download processing time ts_{Lnd}	Local network router	0.00096928 seconds	Figure 6.1
Download wait time tw_{Lnd}	Local network router	0.001749 seconds	Figure 6.1
Number of packet bits being processed in the upload direction for routing ns_{Lr}	Local network router	6.07 bits	Figure 6.1

Number of packet bits waiting to be processed in the upload direction for routing nw_{Lr}	Local network router	993.93 bits	Figure 6.1
Number of web page bits being processed for routing in the download direction ns_{Lrd}	Local network router	9424 bits	Figure 6.3
Number of web page bits waiting to processed for routing in the download direction nw_{Lrd}	Local network router	87,504 bits	Figure 6.1
Probability of being busy ρ_{ns}	Domain Name System	0.7646	Figure 6.1
Upload translation time ts_{ns}	Domain Name System	0.0023 seconds	Figure 6.1
Upload wait time for translation tw_{ns}	Domain Name System	0.00749 seconds	Figure 6.1
Number of packet bits being translated in the upload direction ns_{ns}	Domain Name System	764.57 bits	Figure 6.1
Number of packet bits having to wait for translation in the upload direction nw_{ns}	Domain Name System	235.43 bits	Figure 6.1
Probability of being busy ρ_{ir}	Internet router	0.4462	Figures 6.1 and 6.3
Upload processing time ts_{ir}	Internet router	0.000080 seconds	Figure 6.1
Upload wait time tw_{ir}	Internet router	0.002969 seconds	Figure 6.1
Download processing time ts_{ird}	Internet router	0.0078 seconds	Figure 6.3
Download wait time tw_{ird}	Internet router	0.0107 seconds	Figure 6.3
Number of packet bits being processed for routing in the upload direction ns_{ir}	Internet router	446.24 bits	Figure 6.1
Number of packet bits waiting to be processed for routing in the upload direction nw_{ir}	Internet router	553.76 bits	Figure 6.1
Number of web page bits being processed for routing in the download ns_{ird}	Internet router	43,253 bits	Figure 6.1

(*Continued*)

Table 6.2 (*Continued*)

Variable	Component	Computation	Figure(s)
Number of web page bits waiting for routing in the download direction nw_{ird}	Internet router	53,675 bits	Figure 6.1
Probability of being busy ρ_{ws}	Web server	0.7688	Figure 6.1
Upload processing time ts_{ws}	Web server	0.000138 seconds	Figure 6.1
Upload wait time tw_{ws}	Web server	0.002911 seconds.	Figure 6.2
Download processing time ts_{wsd}	Web server	0.013359 seconds	Figure 6.1
Download wait time tw_{wsd}	Web server	0.005103 seconds	Figure 6.1
Number of web request packet bits being processed in the upload direction ns_{ws}	Web server	723.59 bits	Figure 6.1
Number of web request packet bits waiting to be processed in the upload direction nw_{ws}	Web server	276.41 bits	Figure 6.1
Number of web page bits being processed for download routing ns_{wsd}	Web server	70,136 bits	Figure 6.1
Number of web page bits waiting for download routing nw_{wsd}	Web server	26,792 bits	Figure 6.1
	Total delay time	0.054599 seconds	
	Web request packet length + Web page length	1000 + 96,928 bits = 97,928 bits	
	Effective wired upload/download processing rate	97,928 bits/0.054599 seconds =1 .7936 Mbit/s	

Table 6.3 Summary of Queuing Computations for the Present Wireless Internet System (Mean Values)

Variable	Component	Computation	Figure(s)
Wireless packet upload time t_{wb}	Mobile device	13.889 μs (0.000013889 seconds)	Figures 6.2 and 6.3
Probability of access point being busy ρ_{ap}	Access point	0.7014	Figure 6.3
Access point processing time in the upload direction ts_{ap}	Access point	5.6338 μs (0.0000056338 seconds)	Figure 6.4
Access point queue wait time in the upload direction tw_{ap}	Access point	8.2552 μs (0.0000082552 seconds)	Figure 6.4
Access point processing time in the download direction ts_{apd}	Access point	0.0055 seconds	Figures 6.2 and 6.3
Access point queue wait time in the download direction tw_{apd}	Access point	0.0130 seconds	Figures 6.2 and 6.3
Number of wireless packet bits being processed in the upload direction ns_{ap}	Access point	70.14 bits	Figure 6.4
Number of wireless packet bits having to wait to be processed in the upload direction nw_{ap}	Access point	29.86 bits	Figure 6.4
Number of web page bits being processed in the download direction ns_{apd}	Access point	41,108 bits	Figures 6.2 and 6.3
Number of web page bits having to wait for processing in the download direction nw_{apd}	Access point	55,820 bits	Figures 6.2 and 6.3
Time to translate wireless packet name to IP address tu_{ns}	Domain Name System	291.38 μs (0.00029138 seconds)	Figure 6.4
Time wireless packet waits for name-to-IP address translation tw_{ns}	Domain Name System	58.28 μs (0.00005828 seconds)	Figure 6.4

(*Continued*)

Table 6.3 (*Continued*)

Variable	Component	Computation	Figure(s)
Number of wireless packet bits that are processed for translation ns_{ns}	Domain Name System	80 bits	Figure 6.4
Number of wireless packet bits that are waiting for translation nw_{ns}	Domain Name System	*20 bits*	Figure 6.4
Probability of being busy processing wireless packet ρ_{ir}	Internet router	0.996	Figures 6.2 and 6.3
Time a wireless packet spends in being processed for routing router in the upload direction ts_{ir}	Internet router	8 μs (0.000008 seconds)	Figure 6.1
Time a wireless packet spends waiting to be processed for routing in the upload direction tw_{ir}	Internet Router	5.889 μs (0.000005889 seconds)	Figure 6.4
Time a wireless packet spends being processed for routing in the download direction ts_{ird}	Internet router	0.0078 seconds	Figures 6.2 and 6.3
Time a wireless packet spends waiting to be processed for routing the download direction tw_{ird}	Internet router	0.0107 seconds	Figures 6.2 and 6.3
Number of wireless packet bits being processed in the upload direction ns_{ir}	Internet router	99.6 bits	Figure 6.4
Number of wireless packet bits having to wait for routing in the upload direction nw_{ir}	Internet router	*0.4 bits*	Figure 6.4
Number of web page bits being processed in the download direction ns_{ird}	Internet router	96, 540 bits	Figure 6.2

Table 6.3 (*Continued*)

Variable	Component	Computation	Figure(s)
Number of web page bits having to wait for routing in the download direction nw_{ird}	Internet router	388 bits	Figures 6.2 and 6.3
Probability of being busy ρ_{ws}	Web server	0.8580	Figures 6.2 and 6.3
Upload processing time ts_{ws}	Web server	0.00006891 seconds	Figure 6.4
Upload wait time tw_{ws}	Web server	0.002980 seconds	Figure 6.4
Download processing time ts_{wsd}	Web server	0.0013359 seconds	Figure 6.1
Download wait time tw_{wsd}	Web server	0.005103 seconds	Figure 6.5
Number of web request packet bits being processed in the upload direction ns_{ws}	Web server	723.59	Figure 6.1
Number of web request packet bits waiting to be processed in the upload direction nw_{ws}	Web server	276.41	Figure 6.1
Number of web request packet bits being processed in the download direction ns_{wsd}	Web server	35,068 bits	Figure 6.3
Number of web request packet bits waiting to be processed in the download direction nw_{wsd}	Web server	61,860	Figures 6.2 and 6.3
	Total delay time	0.046879137 seconds	
	Web request packet + web page	100 + 96,928 bits = 97,028 bits	
	Effective wireless upload/download processing rate	97,028 bits/0.046879137 seconds = 2.0697 Mbit/s	

Table 6.4 Summary of Queuing Computations for the Future Wired Internet System (Mean Values)

Variable	Component	Computation	Figure
Packet upload time t_{uc}	User computer	0.000050 seconds	Figure 6.15
Web page download time t_{ucd}	Web server	605.8 μs (0.000006058 seconds)	Figure 6.15
Probability of being busy ρ_{ir}	Internet router	0.0900	Figure 6.15
Upload processing time ts_{ir}	Internet router	0.5 μs (0.0000005 seconds)	Figure 6.15
Upload wait time tw_{ir}	Internet router	49.5 μs (0.0000495 seconds)	Figure 6.15
Download processing time ts_{ird}	Internet router	46.464 μs (0.000046464 seconds)	
Download wait time tw_{ird}	Internet router	557.336 μs (0.0000557336 seconds)	Figure 6.15
Number of packet bits being processed for upload routing ns_{ir}	Internet router	640 bits	Figure 6.17
Number of packet bits waiting to be processed for routing in the upload direction nw_{ir}	Internet router	360 bits	Figure 6.17
Number of web page bits being processed for download routing ns_{ird}	Internet router	2055 bits	
Number of web page bits waiting for download routing nw_{ird}	Internet router	94,873 bits	Figure 6.15
Probability of being busy ρ_{ws}	Web server	0.02120	Figure 6.17
Upload processing time ts_{ws}	Web server	0.11779 μs (≈0 seconds)	Figure 6.15
Upload wait time tw_{ws}	Web server	49.88 μs (0.00004988 seconds)	Figure 6.15
Download processing time ts_{wsd}	Web server	11.417 μs (0.000011417 seconds)	Figure 6.15
Download wait time tw_{wsd}	Web server	594.383 μs (0.000594383 seconds)	Figure 6.15
Number of web request packet bits being processed in the upload direction ns_{ws}	Web server	20.21 bits	Figure 6.17

Table 6.4 (*Continued*)

Variable	Component	Computation	Figure
Number of web request packet bits waiting to be processed in the upload direction nw_{ws}	Web server	978.80 bits	Figure 6.15
Number of web page bits being processed in the download direction ns_{wsd}	Web server	2055 bits	
Number of web page bits waiting to be processed in the download direction nw_{wsd}	Web server	94,873 bits	Figure 6.15
	Total delay time	0.0008639356 seconds	
	Web request packet length + Web page length	1000 + 96,928 bits = 97,928 bits	
	Effective wired upload/download processing rate	97,928 bits/ 0.0008639356 seconds = 113.35 Mbit/s	

each component of the Internet, where Figure 6.4—the local network and router present system simulation model—is the first of these. Now, develop the equations that will be implemented in computer programs (C++) for each of the performance variables, using the mean values previously computed. The mean values and associated probabilities are used to simulate the exponential distribution of the various Internet prediction equations shown below. As opposed to mean values, the prediction equations permit the variation in queuing model variables to be evaluated. The local network and router simulation equations are shown in Figure 6.4.

The probabilities referred to below are generated by using the Excel random number function RAND. The simulation program checks for the possibility of queue being busy. If it is *not* busy, zero time is assigned for the wait time, rather than using the wait time equations below. We can use the simulation program to monitor queue status for the steady-state condition (i.e., wait time does not become excessive). If steady state is not reached, as the load increases, it would be indicative of a poorly performing Internet (e.g., inefficient routing).

Local Network Processing Times

Starting with the packet upload processing time (Ts_{Ln}) that we assume is exponentially distributed, with probability $P(Ts_{Ln})$ and mean processing rate μ_{Ln}, Equation

Table 6.5 Summary of Queuing Computations for the Future Wireless Internet System (Mean Values)

Variable	Component	Computation
Wireless packet upload time t_{wb}	Mobile device	13.889 μs (0.000013889 seconds)
Probability of access point being busy ρ_{ap}	Access point	0.7014
Access point processing time in the upload direction ts_{ap}	Access point	5.6338 μs (0.0000056338 seconds)
Access point queue wait time in the upload direction tw_{ap}	Access point	8.2552 μs (0.0000082552 seconds)
Access point processing time in the download direction ts_{apd}	Access point	0.0055 seconds
Access point queue wait time in the download direction tw_{apd}	Access point	0.0130 seconds
Number of wireless packet bits being processed in the upload direction ns_{ap}	Access point	70.14 bits
Number of wireless packet bits having to wait to be processed in the upload direction nw_{ap}	Access point	29.86 bits
Number of web page bits being processed in the download direction ns_{apd}	Access point	41,108 bits
Number of web page bits having to wait for processing in the download direction nw_{apd}	Access point	55,820 bits
Probability being busy processing wireless packet ρ_{ir}	Internet router	0.9960
Time a wireless packet spends in being processed for routing in the upload direction ts_{ir}	Internet router	8 μs (0.000008 seconds)
Time a wireless packet spends waiting to be processed for routing in the upload direction tw_{ir}	Internet router	5.889 μs (0.000005889 seconds)
Time a wireless packet spends being processed for routing in the download direction ts_{ird}	Internet router	0.0078 seconds
Time a wireless packet spends waiting to be processed for routing in the download direction tw_{ird}	Internet router	0.0107 seconds
Number of wireless packet bits being processed in the upload direction ns_{ir}	Internet router	99.6 bits

Table 6.5 (*Continued*)

Variable	Component	Computation
Number of wireless packet bits having to wait for routing in the upload direction nw_{ir}	Internet router	*0.4 bits*
Number of web page bits being processed in the download direction ns_{ird}	Internet router	96,540 bits
Number of web page bits having to wait for routing in the download direction nw_{ird}	Internet router	388 bits
Probability being busy processing wireless packet ρ_{ws}	Web server	0.8580
Time a wireless packet spends in being processed in the upload direction ts_{wsb}	Web server	0.000006891 seconds
Time a wireless packet spends waiting to be processed in the upload direction tw_{wsb}	Web server	0.003042 seconds
Time a wireless packet spends being processed in the download direction ts_{wsd}	Web server	0.0013359 seconds
Time a wireless packet spends waiting to be processed in the download direction tw_{wsbd}	Web server	0.005103 seconds
Number of wireless packet bits being processed in the upload direction ns_{wsb}	Web server	72.359 bits
Number of wireless packet bits having to wait for processing in the upload direction nw_{ws}	Web server	27.641 bits
Number of web page bits being processed in the download direction ns_{wsd}	Web server	701 bits
Number of web page bits having to wait for processing in the download direction nw_{wsd}	Web server	96,227 bits
	Total delay time	0.0465294580 seconds
	Web request packet + Web page	100 + 96,928 bits = 97,028 bits
	Effective wireless upload/download processing rate	97,028 bits/0.0465294580 seconds = 2.0853 Mbit/s

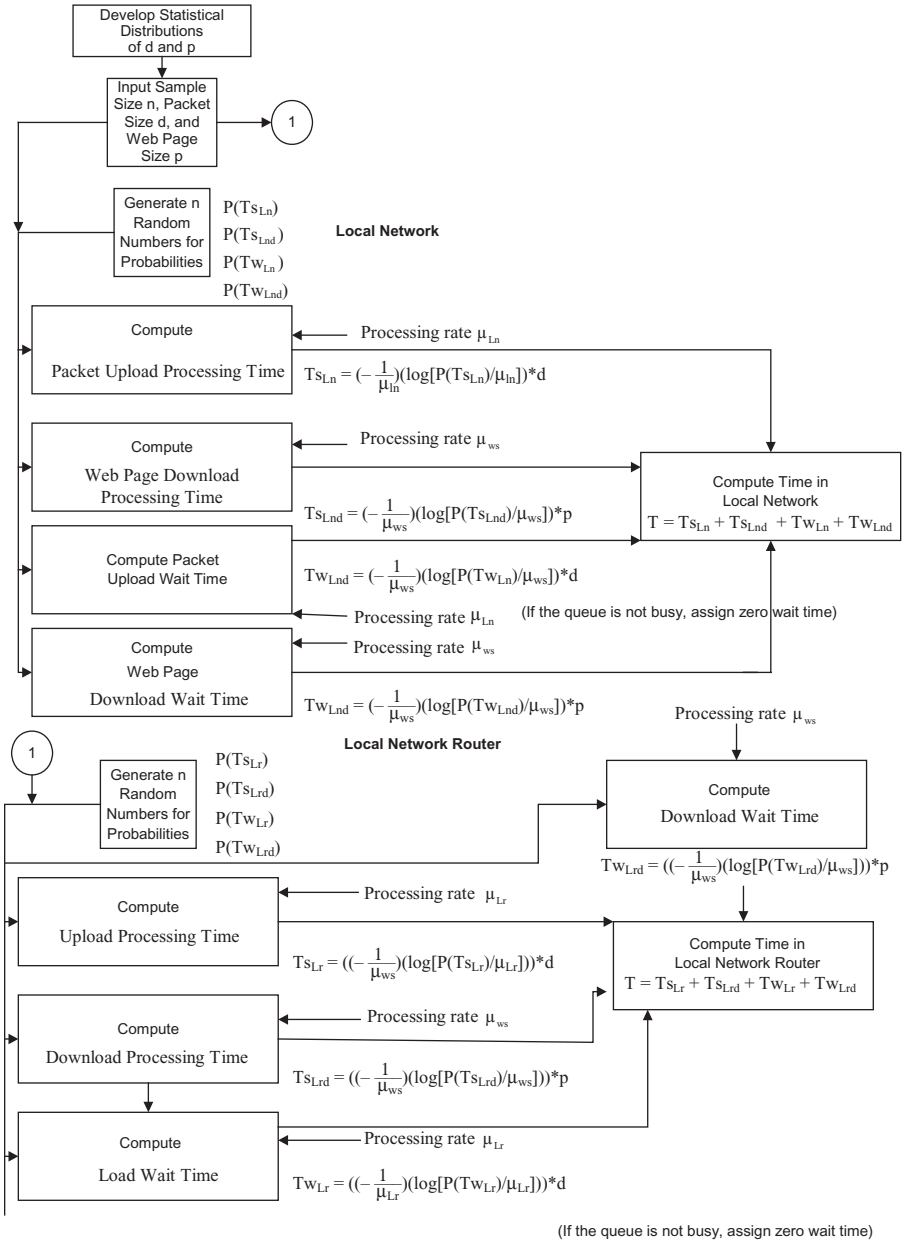


Figure 6.4 Local network and router present wired system simulation model.

6.72 is formulated. Then, solve for Ts_{Ln} in Equation 6.73, which is the upload time per packet, by including the packet size d in the formulation. The assumption of exponential distribution is based on the likelihood that there will be significantly more short times than long times.

$$P(Ts_{Ln}) = (\mu_{Ln})e^{-(\mu_{Ln})Ts_{Ln}}, \tag{6.72}$$

$$Ts_{Ln} = \left(\left(-\frac{1}{\mu_{Ln}}\right)(\log[P(Ts_{Ln})/\mu_{Ln}])\right) * d. \tag{6.73}$$

In a similar vein, develop Equation 6.74 for the Web page download processing time Ts_{Lnd}, with probability $P(Ts_{Lnd})$, mean processing rate μ_{ws}, and Web page size p:

$$Ts_{Lnd} = \left(\left(-\frac{1}{\mu_{ws}}\right)(\log[P(Ts_{Lnd})/\mu_{ws}])\right) * p. \tag{6.74}$$

Local Network Wait Times

Since wait times should follow the pattern of processing times, again assume the exponential distribution for wait times. Thus, Equation 6.75 is produced for the upload wait time, using the probability $P(Tw_{Ln})$:

$$Tw_{Ln} = \left(\left(-\frac{1}{\mu_{Ln}}\right)(\log[P(Tw_{Ln})/\mu_{Ln}])\right) * d. \tag{6.75}$$

Lastly, in a similar fashion, the download wait time Tw_{Lnd} is developed in Equation 6.76:

$$Tw_{Lnd} = \left(\left(-\frac{1}{\mu_{ws}}\right)(\log[P(Tw_{Lnd})/\mu_{ws}])\right) * p. \tag{6.76}$$

Local Network Wait Times Time in System

Finally the time packets and Web pages in the system, waiting and being processed, is computed in Equation 6.77 by adding Equations 6.73–6.76.

$$T = Ts_{Ln} + Ts_{Lnd} + Tw_{Ln} + Tw_{Lnd}. \tag{6.77}$$

Packet Lengths Being Processed and Waiting for Processing by Local Network

Similar to the situation for service and wait times, the distribution of packet bits being processed in the upload direction can be estimated by assuming an exponential distribution (i.e., high probability of small packet lengths and low probability of large packet lengths). The distribution of packet length bits is generated by statistical software, such as Minitab, using the mean ns_{Ln}, computed in Equation 6.8.

In an analogous fashion the distribution of packet length bits having to wait to be processed in the upload direction is generated by statistical software, using the mean nw_{Ln}, computed in Equation 6.9.

Also account for the distribution of Web page length bits that are processed in the download direction generated by statistical software, using the mean ns_{Lnd}, computed in Equation 6.10. Correspondingly, generate the distribution of Web page length bits that must wait to be processed in the download direction, using a mean of nw_{Lnd}, computed in Equation 6.11.

Local Network Router: Present Wired System

Similar to the approach used for the local network, equations for processing time, wait time, and time in the system are developed for the local network router, using different service and wait time probabilities, mean local network router processing rate μ_{Lr}, and mean Web page download processing rate μ_{ws}. These equations are shown in Figure 6.1.

Processing Times

Upload processing time, using packet size d:

$$Ts_{Lr} = \left(\left(-\frac{1}{\mu_{Lr}}\right)(\log[P(Ts_{Lr})/\mu_{Lr}])\right) * d. \tag{6.78}$$

Download processing time, using Web page size p:

$$Ts_{Lrd} = \left(\left(-\frac{1}{\mu_{ws}}\right)(\log[P(Ts_{Lrd})/\mu_{ws}])\right) * p. \tag{6.79}$$

Wait Times

Upload wait time, using packet size d:

$$Tw_{Lr} = \left(\left(-\frac{1}{\mu_{Lr}}\right)(\log[P(Tw_{Lr})/\mu_{Lr}])\right) * d. \tag{6.80}$$

Download wait time, using Web page size p:

$$Tw_{Lrd} = \left(\left(-\frac{1}{\mu_{ws}}\right)(\log[P(Ts_{Lrd})/\mu_{ws}])\right) * p. \tag{6.81}$$

Time in System

Add Equations 6.78–6.81 to obtain time spent in local network router:

$$T = Ts_{Lr} + Ts_{Lrd} + Tw_{Lr} + Tw_{Lrd}. \tag{6.82}$$

Packet Length Bits Being Routed and Waiting for Routing by Local Network Router

Once again, the distribution of packet length bits is generated using statistical software—this time for the local network router, assuming an exponential distribution. To generate the upload packet length bits being routed, use the mean ns_{Lr} from Equation 6.17. Correspondingly, generate the distribution of the packet length bits that must wait to be routed in the upload direction by using the mean nw_{Lr} from Equation 6.18.

Also account for the distribution Web page bits that are routed in the download direction generated by statistical software, using the mean ns_{Lrd} computed in Equation 6.19. Correspondingly, generate the distribution of Web page bits that must wait to be routed in the download direction, using the mean nw_{Lrd} computed in Equation 6.20.

Internet Router: Present Wired System

As before, processing time, wait time, and time in system are computed using the simulation program and the Internet router upload processing rate μ_{ir} and the Web page download processing rate μ_{ws}. The processing logic is shown in Figure 6.5.

Processing Times

Upload processing time, using packet size d:

$$Ts_{ir} = \left(\left(-\frac{1}{\mu_{ir}}\right)(\log[P(Ts_{ir})/\mu_{ir}])\right) * d. \tag{6.83}$$

Download processing time, using Web page size p:

$$Ts_{ird} = \left(\left(-\frac{1}{\mu_{ws}}\right)(\log[P(Ts_{ird})/\mu_{ws}])\right) * p. \tag{6.84}$$

Wait Times

Upload wait time using packet size d:

$$Tw_{ir} = \left(\left(-\frac{1}{\mu_{ir}}\right)(\log[P(Tw_{ir})/\mu_{Lr}])\right) * d. \tag{6.85}$$

Download wait time, using Web page size p:

$$Tw_{ird} = \left(\left(-\frac{1}{\mu_{ws}}\right)(\log[P(Tw_{ird})/\mu_{ws}])\right) * p. \tag{6.86}$$

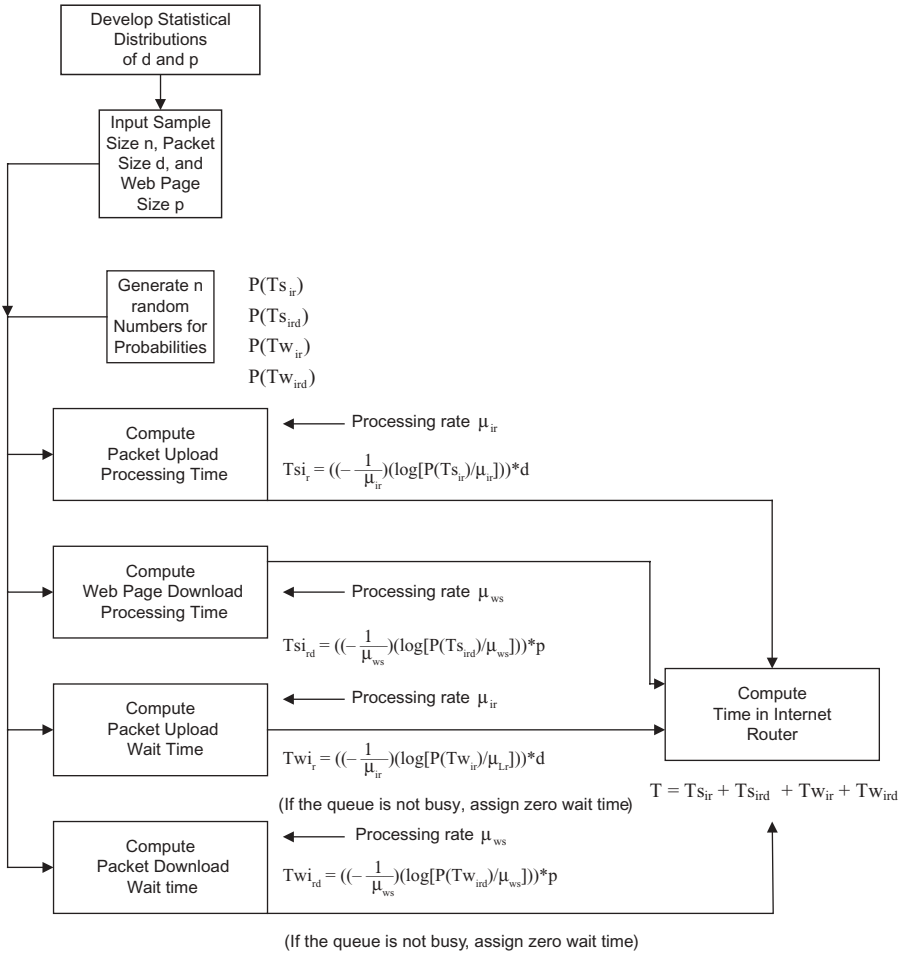


Figure 6.5 Internet router present wired system queuing model.

Time in System

Add Equations 6.83–6.86 to obtain time spent in Internet router:

$$T = Ts_{ir} + Ts_{ird} + Tw_{ir} + Tw_{ird}. \tag{6.87}$$

Packet Lengths Being Routed and Waiting for Routing by the Internet Router

Using the mean ns_{ir} of the number of packet bits being processed for routing in the upload direction from Equation 6.26, statistical software is again called upon to generate an exponential distribution of these data. Similarly, generate the distribution of number of packet bits waiting for routing in the upload direction using the mean

nw_{ir} from Equation 6.27. Continuing in this vein, the corresponding download distributions are generated using the mean values ns_{ird}, from Equation 6.28, and nw_{ird}, from Equation 6.29.

Queue Efficiency

Because it is germane to simulation model analysis, we introduce the concept of queue efficiency. In addition to the various time, packet length, and Web page size variables, queue efficiency should be assessed for each Internet component (e.g., wired Internet router). Do this by computing an efficiency metric, recognizing that the queue count generated in the simulation models accounts for both upload and download traffic, where n is the number of Web page requests, nq is the queue count (count of requests being processed plus requests waiting to be processed) for both upload and download directions, and nd is the number of upload and download data transfers in a sequence of Web page requests by an Internet component (e.g., Internet router). For example, nd = 2 for an Internet router because it is involved in both upload and download data transfers, whereas nd = 1 for a DNS because it is only involved in upload data transfer. Then, queue efficiency qe can be computed in Equation 6.88:

$$\text{qe} = \frac{(\text{Total number of upload and download Web data transfers}) - (\text{Queue count})}{\text{Total number of upload and download Web data transfers}},$$

$$\text{qe} = \frac{(\text{nd*n}) - \text{nq}}{\text{nd*n}} = 1 - \frac{\text{nq}}{\text{nd*n}}. \tag{6.88}$$

This metric can be used to pinpoint strengths and weaknesses in the ability of Internet systems to process queue traffic.

Domain Name System (DNS): Present System

Consistent with the foregoing approaches, processing time, wait time, and time in the system are developed for the DNS, using the appropriate processing and wait time probabilities and the mean DNS processing rate μ_{dn}. The processing logic is shown in Figure 6.6.

Processing Time

$$\text{Ts}_{ns} = \left(\left(-\frac{1}{\mu_{dn}}\right)(\log[P(\text{Ts}_{dn})/\mu_{dn}])\right) * \text{d, using packet size d.} \tag{6.89}$$

Wait Time

$$\text{Tw}_{ns} = \left(\left(-\frac{1}{\mu_{dn}}\right)(\log[P(\text{Tw}_{dn})/\mu_{dn}])\right) * \text{d, using packet size d.} \tag{6.90}$$

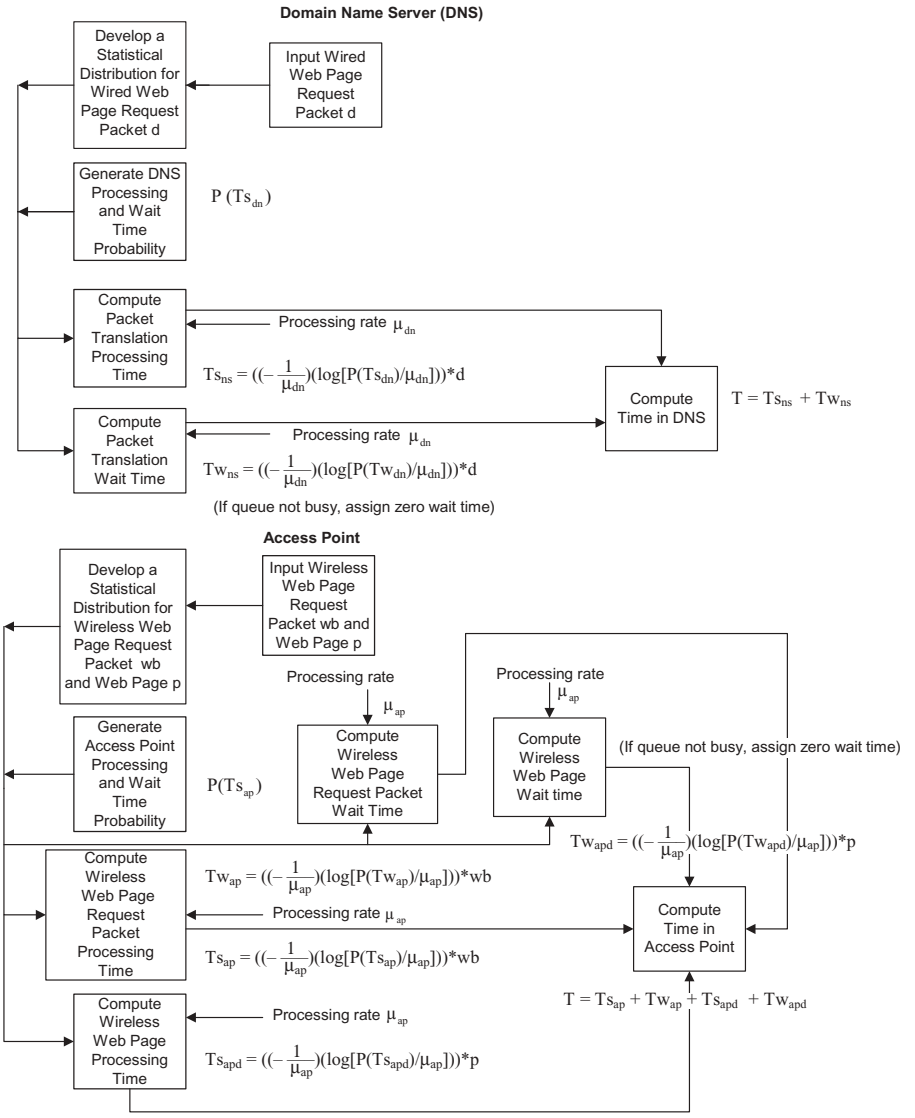


Figure 6.6 DNS and access point present system simulation queuing model.

Time in System

Add Equations 6.89 and 6.90 to obtain time spent in the DNS:

$$T = Ts_{ns} + Tw_{ns}. \tag{6.91}$$

Packet Bits Translated from Name-to-IP Address by the DNS and Waiting for Translation

Using the mean ns_{ns}, from Equation 6.33, of the number of packet bits being translated, statistical software is again called upon to generate an exponential distribution of these data. Similarly, we generate the distribution of number of packet bits waiting for translation, using the mean nw_{ns} from Equation 6.34.

Access Point: Present Wireless System

For the wireless system, access point processing and wait times are computed using the logic in Figure 6.6, using the access point processing rate μ_{ap}.

Processing Times

This upload time accounts for the processing required to process Web packet size wb, using the processing rate μ_{ap} in Equation 6.92:

$$Tsap = \left(\left(-\frac{1}{\mu_{ap}}\right)(\log[P(Ts_{ap})/\mu_{ap}])\right) * wb. \tag{6.92}$$

Similarly, Equation 6.93 accounts for the time required in the download direction to process Web page size p:

$$Tsapd = \left(\left(-\frac{1}{\mu_{ap}}\right)(\log[P(Ts_{apd})/\mu_{ap}])\right) * p. \tag{6.93}$$

Wait Tines

Correspondingly, Equation 6.94 computes the wait time in the upload direction and Equation 6.95 computes the wait time in the download direction:

$$Twap = \left(\left(-\frac{1}{\mu_{ap}}\right)(\log[P(Tw_{ap})/\mu_{ap}])\right) * wb, \tag{6.94}$$

$$Twapd = \left(\left(-\frac{1}{\mu_{ap}}\right)(\log[P(Tw_{apd})/\mu_{ap}])\right) * p. \tag{6.95}$$

Time in System

Equation 6.96 computes the time in the system, accounting for both upload and download processing and wait times:

$$T = Tsap + Tsapd + Twap + Twapd. \tag{6.96}$$

Wireless Packet Bits Processed by Access Point and Waiting for Processing

Using the mean ns_{ap}, from Equation 6.53, of the number of packet bits being processed, statistical software is once again called upon to generate an exponential distribution of these data. Similarly, generate the distribution of number of packet bits waiting for processing, using the mean nw_{ap} from Equation 6.54.

Web Server Processing: Wired and Wireless

Continuing the generation of processing time, wait time, and time in system, equations are developed for the Web servers, using the appropriate processing and wait time probabilities, the mean Web server processing rate μ_{ws}, the wired Web page request packet size d, the Web page size p, and the wireless Web request packet size wb. The processing logic is shown in Figure 6.7.

Processing Times

Wired upload, using wired packet size d:

$$Ts_{ws} = \left(\left(-\frac{1}{\mu_{ws}}\right)(\log[P(Ts_{ws})/\mu_{ws}])\right) * d. \tag{6.97}$$

Wired and wireless download, using web page size p:

$$Ts_{wsd} = \left(\left(-\frac{1}{\mu_{ws}}\right)(\log[P(Ts_{wsd})/\mu_{ws}])\right) * p. \tag{6.98}$$

Wireless upload, using wireless packet length wb:

$$Ts_{wsb} = \left(\left(-\frac{1}{\mu_{ws}}\right)(\log[P(Ts_{wsb})/\mu_{ws}])\right) * wb. \tag{6.99}$$

Wait Times

The corresponding wait times are developed as follows:

$$Tw_{ws} = \left(\left(-\frac{1}{\mu_{ws}}\right)(\log[P(Tw_{ws})/\mu_{ws}])\right) * d \text{ (wired upload)}, \tag{6.100}$$

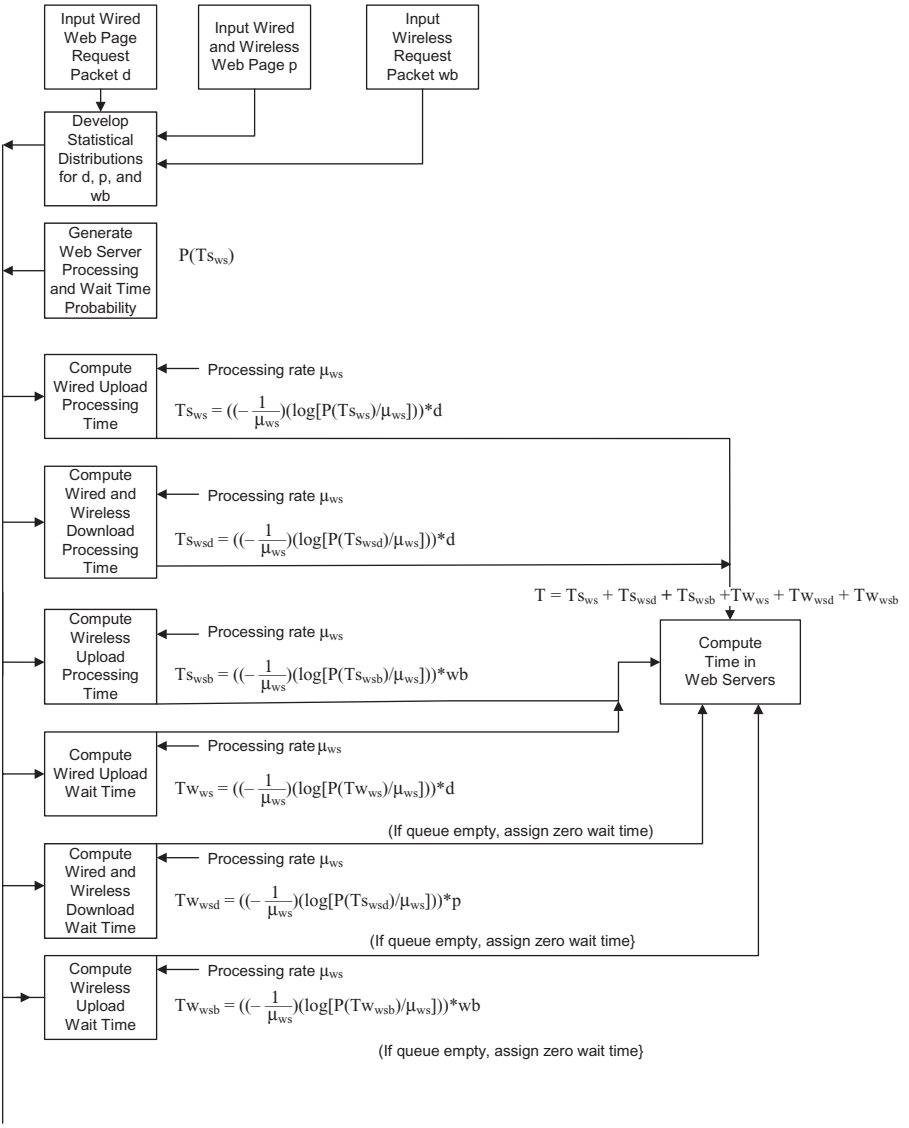


Figure 6.7 Web server simulation queuing model (wired and wireless).

$$\mathrm{Tw}_{\mathrm{wsd}} = \left(\left(-\frac{1}{\mu_{\mathrm{ws}}}\right)(\log[\mathrm{P}(\mathrm{Tw}_{\mathrm{wsd}})/\mu_{\mathrm{ws}}])\right) * \mathrm{p}\ \text{(wired and wireless download)}, \tag{6.101}$$

$$\mathrm{Tw}_{\mathrm{wsb}} = \left(\left(-\frac{1}{\mu_{\mathrm{ws}}}\right)(\log[\mathrm{P}(\mathrm{Tw}_{\mathrm{wsb}})/\mu_{\mathrm{ws}}])\right) * \mathrm{wb}\ \text{(wireless upload)}. \tag{6.102}$$

Time in System

Time in the system for all three cases is computed by adding Equations 6.97–102:

$$T = Ts_{ws} + Ts_{wsd} + Ts_{wsb} + Tw_{ws} + Tw_{wsd} + Tw_{wsb}. \quad (6.103)$$

Web Page Lengths Being Processed and Waiting for Processing by the Web Servers

Using the mean ns_{ws}, from Equation 6.41, of the number of Web page bits being processed in the download direction, the statistical software is again called upon to generate an exponential distribution of these data. Similarly, we generate the distribution of number of Web page bits waiting for processing in the download direction, using the mean nw_{ws}, from Equation 6.44.

Summary of Simulation Model Computations

The main benefit of the C++ simulation model results is to provide a comparison of the performance and reliability of the present Internet with the proposed future Internet. Thus, the following figures contrast the present with future Internet performance; the latter is based on equations, queuing model diagrams, and simulation logic diagrams that have been developed. In addition, we present additional results that illustrate key performance characteristics of the present Internet. For example, it is of interest to identify when the various Internet components achieve stability as a function of number of uploads and downloads. This is illustrated in Figure 6.8, where time in the present Internet system stabilizes (i.e., reaches steady state) after $n = 26$ uploads and downloads. Thus, in this example, we would not be confident of dependable performance until $n > 26$. Now, the shape of the plots is determined by the assumption of exponentially distributed queue processing and wait times. Other distributions could be assumed, which could result in different patterns, but the important point is the efficacy of the modeling methodology.

Another interesting comparison is between the wired and wired present Internet systems, again using time in the system as the basis of comparison. Figure 6.9 shows an example in which the wireless has better performance (i.e., shorter time in the system) because it is not encumbered by the overhead induced by the local network and router required in the wired system (i.e., the wireless system access point is much faster).

Queue efficiency computations identify the Internet components that are efficient with respect to processing Internet traffic—both upload and download—and those that are inefficient, thus highlighting smooth traffic flows and bottlenecks, respectively. For example, Figure 6.10 shows that the proposed future wired Internet is more efficient than the present wired Internet. The reason for this, as shown in Figure 6.10, is the fact that queue count dominates the computation of queue efficiency in Equation 6.87, thus resulting in lower queue count and higher queue efficiency for the proposed future Internet components. By examining Figures 6.10

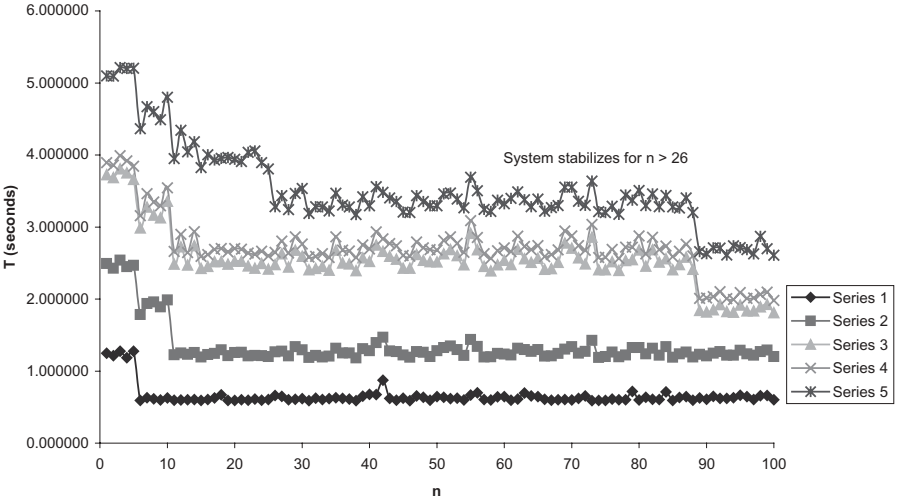


Figure 6.8 Time in present wired Internet system T versus number uploads and downloads n. Series 1: Local network. Series 2: Local network + local network router. Series 3: Local network + local network router + Internet router. Series 4: Local network + local network router + Internet router + DNS. Series 5: Local network + local network router + Internet router + DNS + web server.

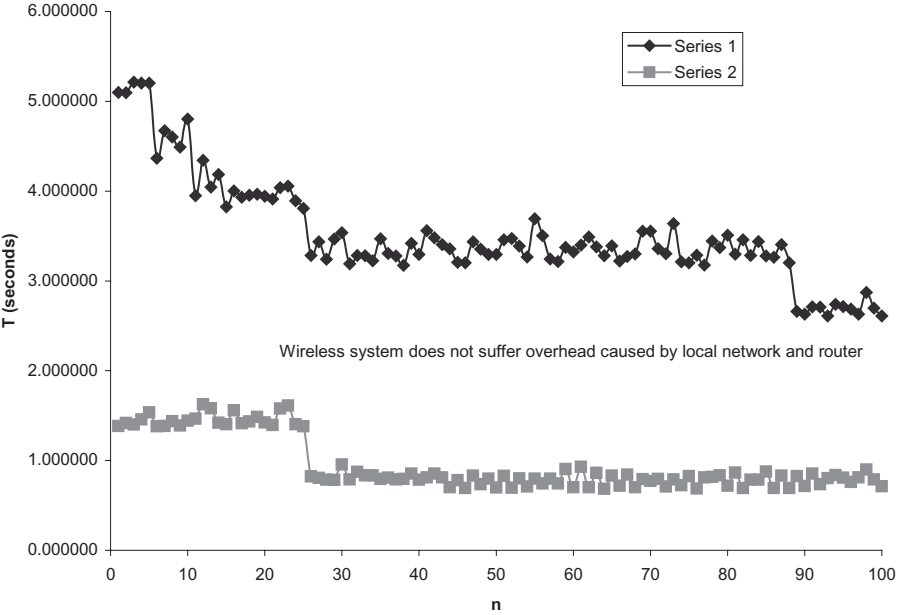


Figure 6.9 Time in present Internet system T versus number of uploads and downloads n. Series 1: Wired system. Series 2: Wireless system.

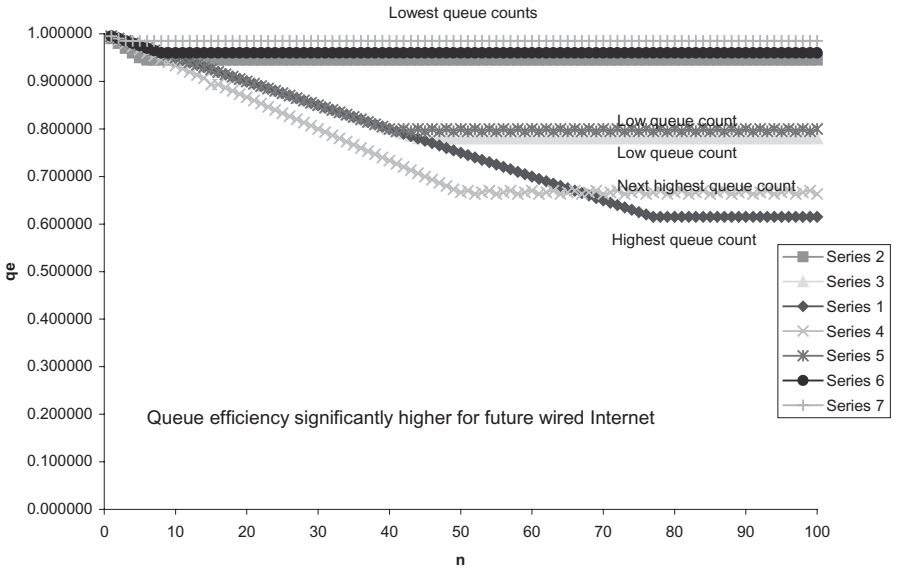


Figure 6.10 Queue efficiency qe versus number of uploads and downloads. Series 1: Present wired DNS. Series 2: Present wired local network router. Series 3: Present wired Internet router. Series 4: Present Web server. Series 5: Present wireless access point. Series 6: Future wired Internet router. Series 7: Future wired Web server.

and 6.11, you can see that there is an inverse association between queue efficiency and queue count (i.e., small queue count leads to large queue efficiency).

Finally, we look at the patterns of data that wait to be processed by Internet components. The rationale for this assessment is queue processing efficiency from the perspective of quantity of data waiting to be processed (i.e., stability) in Figure 6.12. This figure tells us that download processing of Web pages will eventually deteriorate, and become unstable, as queue count increases. Figure 6.12 provides another perspective: aided by the standard deviation that has been computed for each component, we see that the wireless Web server has the highest standard deviation, meaning that its distribution of queuing data has the most variability. This is due to the great variety of mobile devices that connect to the Internet, resulting in the Web server being less able to process data efficiently than when responding to wired system requests.

INTERNET RELIABILITY ANALYSIS

Reliability is the probability of fault-free operation of an Internet component for a specified time in a specified environment. A fault is an instance of any unanticipated Internet component output (e.g., incorrect routing of a Web page request) caused by errors in the component. Faults are the results of errors, which are design or coding flaws created inadvertently by hardware designers and programmers [NEU93].

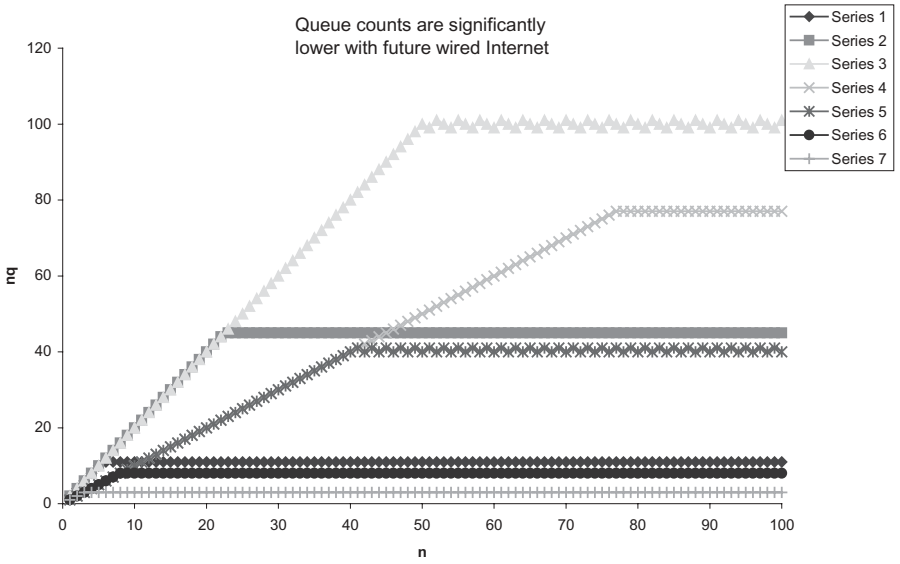


Figure 6.11 Queue count nq versus number of uploads and downloads n. Series 1: Present wired local network router. Series 2: Present wired Internet router. Series 3: Present wired Web server. Series 4: Present wired DNS. Series 5: Present wireless access point. Series 6: Future wired Internet router. Series 7: Future wired Web server.

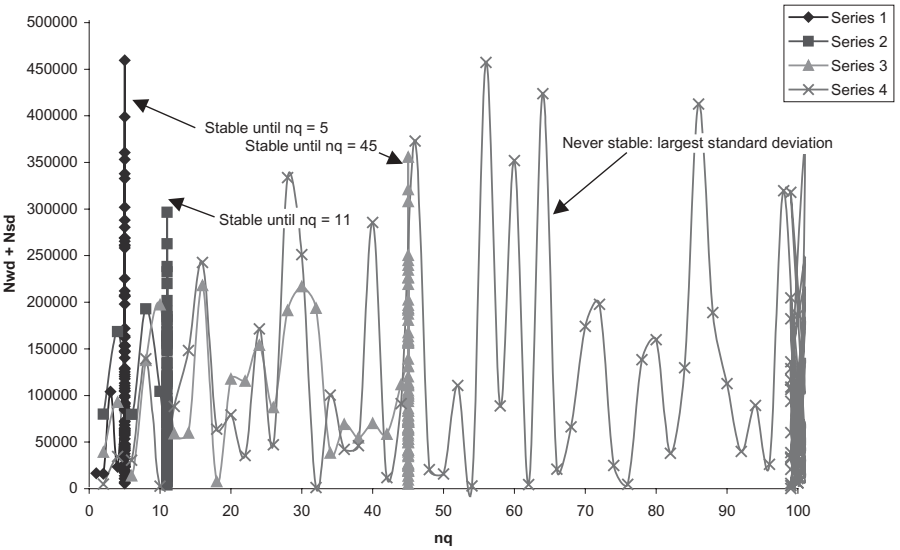


Figure 6.12 Number of download bits waiting for processing and being processed Nwd + Nsd versus queue count nq. Series 1: Present wired local network; standard deviation = 102301. Series 2: Present wired local network router; standard deviation = 64914. Series 3: Present wired internet router; standard deviation = 76006. Series 4: Present wireless web server; standard deviation = 108851.

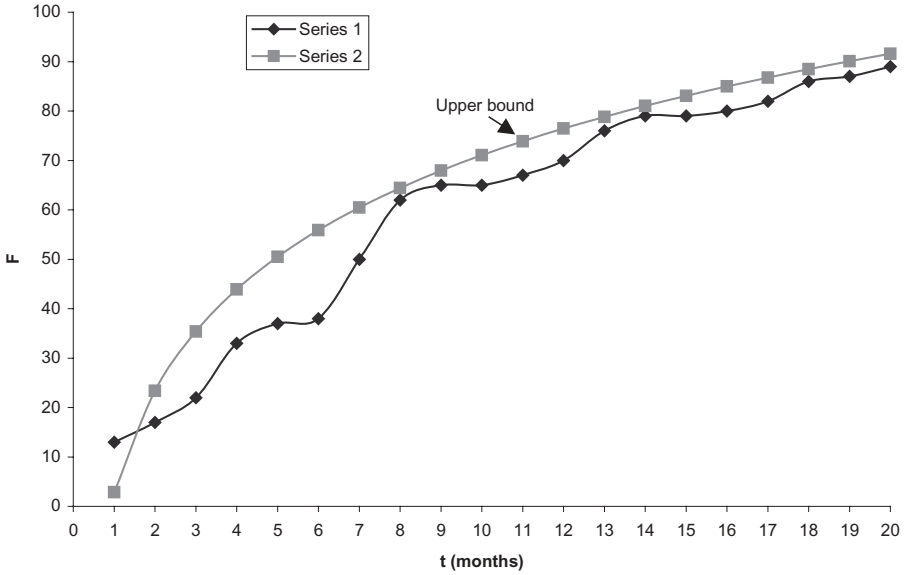


Figure 6.13 Cumulative number of Internet faults and failures, F, versus time t. Series 1: Empirical. Series 2: Predicted. MRE = 01920; $R^2 = 0.9447$.

Reliability modeling involves using fault data (e.g., records of faults occurring per testing time of an Internet component) to fit some parameters of a prediction model, and then using this model to predict reliability of Internet components at a future time. We can then determine when components will be ready for release to the Internet, based on predicting when the number of faults and failures will drop below a specified threshold [FOO95]. Predicting the reliability of components means we can estimate how many faults the Internet user is likely to encounter per unit time.

Because Internet fault and failure data do not correspond to a theoretical distribution (we made a test of the Poisson distribution of failures that failed), we fit a regression curve against the cumulative empirical data [FINK98] in Figure 6.13 that yields a reasonable fit of the upper bound by virtue of $R^2 = 0.9447$ and mean relative error (MRE) = 0.1920. This source of fault and failure data [FINK98] does not break the data down by component. Thus, Figure 6.13 represents the *total* present Internet reliability in terms of cumulative faults and failures.

When assessing Internet reliability—both present and proposed systems—it is important to note the number of servers comprising a component in each part of the Internet (e.g., number of Web servers comprising a Web site) because with multiple servers, a degree of redundancy is provided, such that if one server fails, another can be used. This fact is used in the reliability prediction equation to be developed.

Local Network and Router

Local network failure data were obtained from Kalyanakrishnam et al. [KAL99], involving Windows NT Local Area Network (LAN) systems. For a sample size of

1298, mean uptime = 354.6 hours and mean downtime = 1.97 hours, yielding a reliability = 0.9945 (354.6/(354.6 + 1.97)). Local network router failure data were obtained from reports about Cisco routers. For the Cisco 6500 router, 23 of the 805 routers failed, for a reliability 1 – (23/805) = 0.9714 [PAG]. The local network and router comprise a single network because there is only a single router server, as shown in Figure 6.1. Thus, this network, lacking redundancy, has no backup capability if a failure occurs.

Internet Router

An Internet router reliability estimate based on 104 router outages out of a total of 1616 outages yields reliability = 1 – (104/1616) = 0.9356, as reported in the Interdomain Border Gateway Protocol data collection (01/98~11/98) [LAB98]. The Internet router uses two servers, thus providing a degree of redundancy against failures, as shown in Figure 6.1.

Domain Name System (DNS)

The Domain Name System (DNS) is a ubiquitous part of everyday computing, translating human-friendly machine names to numeric IP addresses. Most DNS research has focused on server-side infrastructure, with the assumption that the aggressive caching and redundancy on the client side are sufficient. However, through systematic monitoring, the authors found that client-side DNS failures are widespread and frequent, degrading DNS performance and reliability [PAR]. In support of this finding, we collected the following failure data:

Number of successful DNS accesses: 5268. Number of failed DNS accesses: 153 (Cricket Liu's Advisor, http://www.infoblox.com/services/cl_cookbook_5.14.cfm). Reliability = 5268/(5268 + 153) = 0.9718.

Successfully answered queries: 37,973. Failure responses: 348 (http://www.daemon.be/maarten/dns.html). Reliability = 37,973/(37,973 + 348) = 0.9909.

Total queries: 4,547,577. Total replies: 3,893,205. [PAP]. Reliability = 3,893,205/4,547,577 = 0.8561.

Web Server

The reliability for Web applications can be defined as the probability of failure-free Web operation completions (i.e., successful completion of upload Web request and download Web page delivery). We define Web failures as the inability to correctly obtain or deliver information, such as documents or computational results, requested by Web users. This definition conforms to the standard definition of failures being the behavioral deviations from user expectations [IEE90]. Based on this definition, we can consider the following failure sources:

Web server failures that prevent the delivery of requested information to Web users.

Web site content failures that prevent the acquisition of the requested information by Web users because of problems such as missing or inaccessible files.

Applying these definitions, we obtained failure data in terms of failures per hit, where a hit is a successful access of Web server, from the School of Engineering, Southern Methodist University, Dallas, Texas, USA (errors per hit = 0.09091) and Unix desktop computers accessing Web servers (errors per hit = 0.0466). These data yield reliability = (1 – 0.09091) = 0.90909 and (1 – 0.0466) = 0.9534, respectively [TIA04].

In our models, the number of Web servers varies depending on whether they are used in a wired system (three servers) in Figure 6.1 or wireless system (six servers) in Figure 6.3. In both cases, reliability would be significantly improved over a single server system.

Access Point

Given their convenience of user mobility, wireless networks are increasingly being considered as the platform of choice for various applications. Critical applications, such as health monitoring systems and so on, require the network to continue to function even in the presence of faults. Unfortunately, current wireless networks are notoriously prone to a number of problems, such as the loss of connectivity due to user mobility combined with network failures, which makes it difficult to guarantee their reliability. Today's users are mostly content with their ability to access wired networks conveniently from mobile devices, even if the access is unreliable. However, as wireless networks become ubiquitous and start to support more critical applications, users will expect wireless networks to provide the same guarantees of reliability as their wired counterparts. Furthermore, providing wireless networks with a certain degree of reliability will lead to more opportunities for wireless carriers to provide applications that can be run satisfactorily on mobile devices. Some authors [GAN03] propose the signal-to-noise ratio as the metric to identify access point failures. Unfortunately, they do not provide actual failure data that can be used in our study.

In a study of digital cellular systems [TIP02], call blocking probability (i.e., the chance that due to heavy wireless traffic, calls will be blocked) was estimated. The blocking probabilities range from a minimum of 0.0385 to a maximum of 0.226, yielding from reliability = (1 – 0.0385) = 0.9615 to reliability = (1 – 0.226 = 0.774), respectively.

In our model, only a single access point is used because only a single mobile device accesses the wired network, via the access point, as shown in Figure 6.3. Of course, in the real Internet, there are many mobile devices and access points that would impose additional load on the Internet. However, our goal is not to model the total Internet, which would be infeasible. Rather, our objective is *compar*e the present Internet with the proposed Internet. In each case, our access point configuration is the same.

Present Wired Internet System

Now use the above reliabilities to predict both the *present* and *proposed* Internet component reliability R_c in Equation 6.104 [SHO83], where R is the reported server reliability (e.g., 0.9945 for local network), 1 – R is the server unreliability, and n is the number of servers for a given component. Then, to obtain the Internet system reliability R_s in Equation 6.105, the component reliabilities are multiplied, reflecting the fact that Internet components operate in series. To apply R_c to the entire Internet would be incorrect because each component comprises a separate configuration of servers, where the components operate in series. Thus, R_c is applied to each component, and then R_s is applied to the entire Internet.

These equations take into account the use of multiple servers for some Internet components in our model.

$$R_c = 1-(1-R)^n, \tag{6.104}$$

$$R_s = \prod_{i=1}^{N} R_c. \tag{6.105}$$

We proceed by first listing the server reliabilities, which were described above, for the wired system and the number of servers that a given component uses, where for components that have multiple reported server reliabilities, only the minimum and maximum values are used in order to provide a range of Internet reliability predictions:

Local network: 0.9945, n = 1 server:

$$R_c = R = 0.9945.$$

Local network router: 0.9714, n = 1 server:

$$R_c = R = 0.9714.$$

Domain Name Server (DNS): 0.8561, 0.9718, 0.9909, n = 3 servers:

$$R_c = 1-(1-0.8561)^3 = 0.99702 \text{ (minimum)},$$
$$R_c = 1-(1-0.9909)^3 = 0.99999 \text{(maximum)}.$$

Internet router: 0.9356, n = 2 servers:

$$R_c = 1-(1-0.9356)2 = 0.99585.$$

Web server: 0.9091, 0.9534, n = 3 servers:

$$R_c = 1-(1-0.9091)3 = 0.99925 \text{(minimum)},$$
$$R_c = 1-(1-0.9534)3 = 0.99990 \text{(maximum)}.$$

Then, applying equation R_s to the components in series:

$$R_s = 0.9945 * 0.9714 * 0.99702 * 0.99585 * 0.99925 = 0.95846 \text{ (minimum)},$$
$$R_s = 0.9945 * 0.9714 * 0.99999 * 0.99585 * 0.99990 = 0.96194 \text{ (maximum)}.$$

Thus, the reliability of the present wired system is predicted to be between 0.95846 and 0.96194 .

Present Wireless Internet System

To obtain the reliability estimates for the wireless system, we need only to factor in the access point reliability estimates (0.774, 0.9615) to the results for R_s computed above, as follows:

$$R_s = 0.95846 * 0.774 = 0.74184 \text{ (minimum)},$$
$$R_s = 0.96194 * 0.9615 = 0.92491 \text{ (maximum)}.$$

Thus, present wireless system reliability is predicted to range between 0.74184 and 0.92491. These results, combined with the total Internet cumulative faults and failures in Figure 6.13, provide a comprehensive picture of present Internet reliability. Figure 6.13 has the desirable feature of predicting an upper bound on total Internet cumulative faults and failures. Thus, we are assured that it is highly unlikely that reliability will be any worse than the upper bound. The overall picture that emerges suggests that considerable improvement in reliability is needed, particularly with regard to the wireless system.

Proposed Wired Internet System

The equations for R_c and R_s are now applied to the proposed wired Internet system. Recall that the proposed wired system does not include local network, local network router, and DNS. Therefore, the following component reliabilities are used:
Internet router: 0.9356, n = 2 servers:

$$R_c = 1 - (1 - 0.9356)2 = 0.99585.$$

Web server: 0.9091, 0.9534, n = 3 servers:

$$R_c = 1 - (1 - 0.9091)3 = 0.99925,$$
$$R_c = 1 - (1 - 0.9534)3 = 0.99990,$$
$$Rs = 0.99585 * 0.99925 = 0.99510 \text{ (minimum)},$$
$$Rs = 0.99585 * 0.99990 = 0.99575 \text{ (maximum)}.$$

Note the improvements over the present wired Internet system: 0.95846 → 0.99510 (3.83% increase, minimum) and 0.96194 → 0.99575 (3.51% increase, maximum) (Figure 6.14).

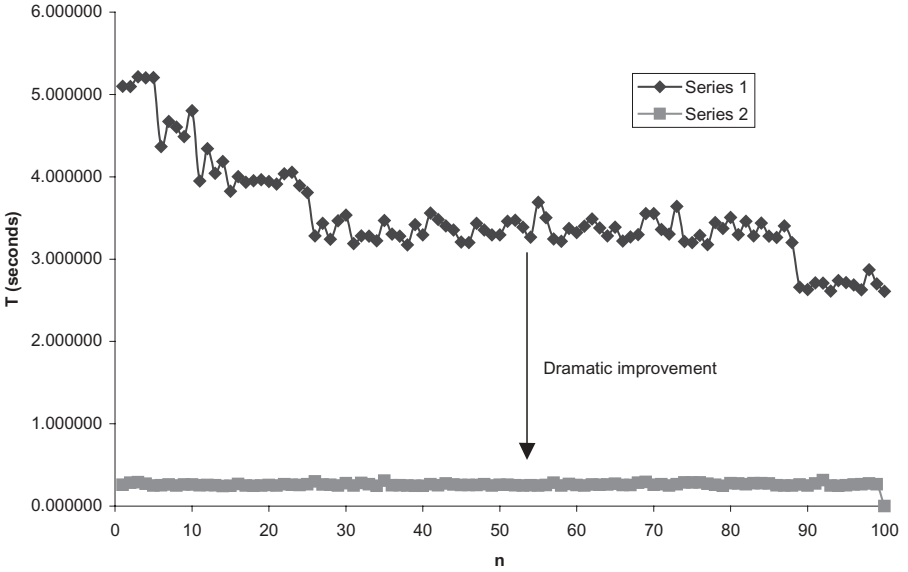


Figure 6.14 Time in system T versus number of uploads and downloads n. Series 1: Present Internet wired system. Series 2: Proposed Internet wired system.

Proposed Wireless Internet System

The equations for R_c and R_s are now applied to the proposed wireless Internet system. Again, recall that the proposed wireless system does not include local network, local network router, and DNS. However. It does require an access point. Therefore, the following component reliabilities are used:
Access Point: 0.774, 0.9615, n = 1 server:

$$R_c = R = 0.774, 0.9615.$$

Internet router: 0.0.9356, n = 2 servers:

$$R_c = 1-(1-0.9356)2 = 0.99585.$$

Web server: 0.9091, 0.9534, n = 6 servers:

$$R_c = 1-(1-0.9091)6 = 0.99999 \text{ (minimum)},$$
$$R_c = 1-(1-0.9534)6 = 0.99999 \text{ (maximum)},$$
$$Rs = 0.774 * 0.99585 * 0.99925 = 0.77021 \text{ (minimum)},$$
$$Rs = 0.9615 * 0.99585 * 0.99990 = 0.95714 \text{ (maximum)}.$$

Thus, the reliability of the proposed wireless Internet system is predicted to be between 0.77021 and 0.95714. However, note the improvements over the present

wireless Internet system: 0.74184 → 0.77021 (3.82 % increase, minimum) and 0.92491 → 0.95714 (3.48 % increase, maximum). Thus, the relative reliability improvements are approximately the same for the wired and wireless systems, with the former yielding the greater absolute improvement.

PERFORMANCE ANALYSIS OF PROPOSED FUTURE WIRED INTERNET

As has been demonstrated, the proposed Internet would achieve improved performance and reliability because its configuration would have fewer components, thus reducing performance overhead, single points of failure, and number of components that could fail. The configuration would be comprised of only a user computer, Internet routers, and Web servers for wired systems. No DNS would be required because each user computer and mobile device would be supplied with a permanent IP address, using IPv6. For wireless systems, the access point is still necessary to provide for mobile device access to the Internet.

In order to provide a realistic model of the proposed future Internet, projected future Internet speeds of Web page request packet upload rate λ_{uc} = 20 Mbit/s and Web page download rate λ_{ws} = 160 Mbit/s were obtained from the following source: http://www.livescience.com/technology /070522_cable_modem.html.

Also, I found that the future Internet routing speed is projected as μ_{ir} = 1000 Mbit/s [BAN]. These data are used to produce revised computations, using the equations below.

In addition, it is appropriate to use different probabilities of processing and wait times than were used for the present Internet because these probabilities change with changing operating conditions. The proposed future wired Internet system queuing model, using the following performance computations, is shown in Figure 6.15.

User Computer

Mean Upload Packet Time t_{uc}

This mean time is computed in Equation 6.106 by dividing the Web page request packet of size d by the packet upload rate λ_{uc}:

$$t_{uc} = d/\lambda_{uc} = 1000 \text{ bits}/20 \text{ Mbit/s} = 50 \text{ μs}. \tag{6.106}$$

Mean Web Page Download Time t_{ucd}

This mean time is computed in Equation 6.107 by dividing the Web page size p by the packet upload rate λ_{uc}:

$$t_{ucd} = p/\lambda_{ws} = 96{,}928 \text{ bits}/160 \text{ Mbit/s} = 605.8 \text{ μs}. \tag{6.107}$$

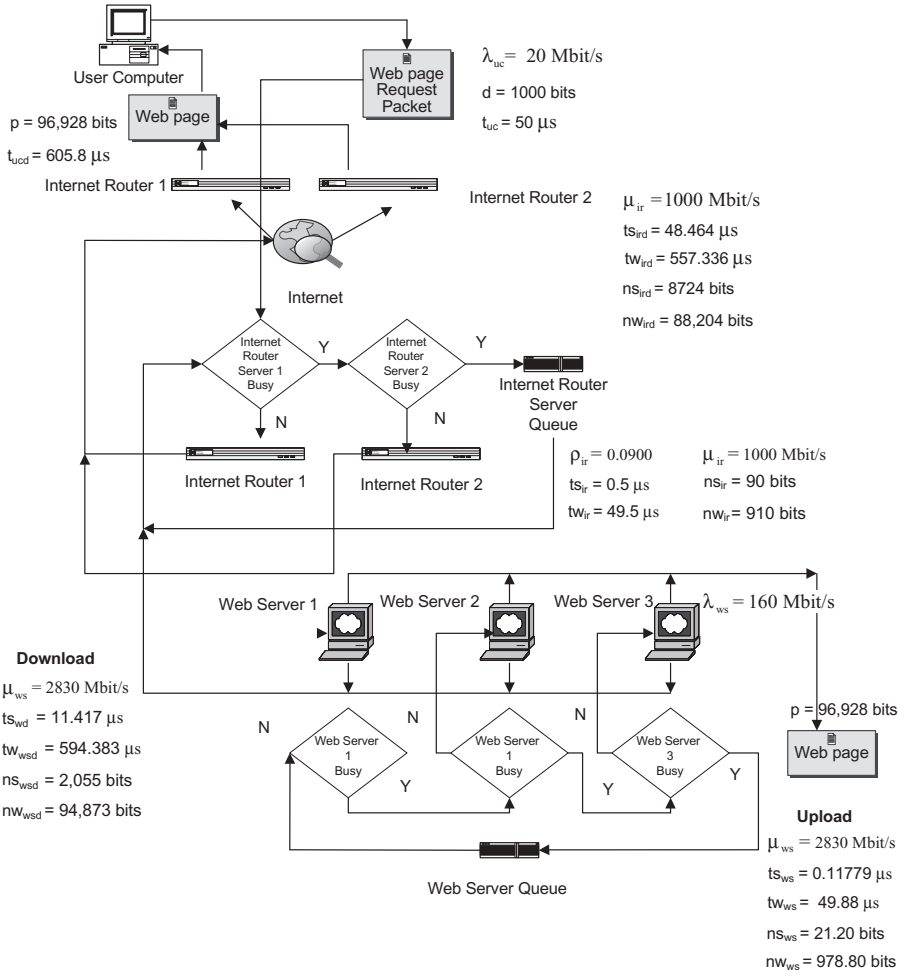


Figure 6.15 Future wired Internet system queuing model.

Wired System Internet Router

Probability of Being Busy ρ_{ir}

This probability is computed in Equation 6.108 by dividing the *sum of packet upload rate* λ_{uc} and *Web page download rate* λ_{ws} by the Internet router processing rate μ_{ir}, accounting for the number of servers s:

$$\rho_{ir} = \frac{(\lambda_{uc} + \lambda_{ws})}{(s)(\mu_{ir})} = \frac{(20 + 160 \text{ Mbit/s})}{(2)(1000 \text{ Mbit/s})} = 0.0900. \tag{6.108}$$

Mean Upload Service Time ts_{ir}

This mean time is computed in Equation 6.109 by dividing the Web page request packet length d by the Internet router processing rate μ_{ir}, accounting for number of servers s:

$$t_{sir} = \frac{d}{(s)(\mu_{ir})} = \frac{1000 \text{ bits}}{(2)(1000 \text{ Mbit/s})} = 0.5 \ \mu s. \tag{6.109}$$

Mean Upload Wait Time tw_{ir}

This mean time is computed in Equation 6.110 by subtracting the upload service time computed in Equation 6.109 from the packet upload time t_{uc}:

$$tw_{ir} = t_{uc} - ts_{ir} = 50 - 0.5 \ \mu s = 49.5 \ \mu s \tag{6.110}$$

Mean Download Service Time ts_{ird}

This mean time is computed in Equation 6.111 by dividing the Web page length p by the Internet router processing rate μ_{ir}, accounting for number of servers s:

$$ts_{ird} = \frac{p}{(s)(\mu_{ir})} = \frac{96{,}928 \text{ bits}}{(2)(1000 \text{ Mbit/s})} = 48.464 \ \mu s. \tag{6.111}$$

Mean Download Wait Time tw_{ird}

This mean time is computed in Equation 6.112 by subtracting the download service time computed in Equation 6.111 from the Web page download time t_{ucd}, computed in Equation 6.107:

$$tw_{ird} = t_{ucd} - ts_{ird} = 605.8 - 48.464 \ \mu s = 557.336 \ \mu s. \tag{6.112}$$

Mean Number of Bits Being Processed in the Upload Direction ns_{ir}

This quantity is computed in Equation 6.113 by multiplying the probability of Internet router queue being busy, computed in Equation 6.108, by the Web page request packet size d:

$$ns_{ir} = \frac{(\lambda_{uc} + \lambda_{ws})(d)}{s\mu_{ir}} = \frac{(20 + 160 \text{ Mbit/s})(1000 \text{ bits})}{(2)(1000 \text{ Mbit/s})} = 90 \text{ bits}. \tag{6.113}$$

Mean Number of Bits Waiting for Processing in the Upload Direction nw_{ir}

This quantity is computed in Equation 6.114 by subtracting the number of bits being processed in the upload direction, computed in Equation 6.115 from the Web page request packet size d:

$$nw_{ir} = d - ns_{ir} = 1000 - 90 \text{ bits} = 910 \text{ bits}. \tag{6.114}$$

Mean Number of Bits Being Processed in the Download Direction ns_{ird}

This quantity is computed in Equation 6.115 by multiplying the probability of Internet router queue busy, computed in Equation 6.108, by the Web page size p:

$$ns_{ird} = \frac{(\lambda_{uc} + \lambda_{ws})(p)}{s\mu_{ir}} = \frac{(20 + 160 \text{ Mbit/s})(96{,}928 \text{ bits})}{(2)(1000 \text{ Mbit/s})} = 8724 \text{ bits}. \tag{6.115}$$

Mean Number of Bits Waiting For Processing in the Download Direction nw_{ird}

This quantity is computed in Equation 6.116 by subtracting the number of bits being processed in the download direction, computed in Equation 6.115 from the Web page size p:

$$nw_{ird} = p - ns_{ird} = 96{,}928 - 8724 \text{ bits} = 88{,}204 \text{ bits}. \tag{6.116}$$

The statistical program is used to generate the distribution of wired system Internet router number of bits being processed and waiting for processing, using the above mean values.

Wired System Web Server Processing

The probability of queue busy, using the IBM System x3250 M2 4194 Web server processing rate μ_{ws} = 2830 Mbit/s, is computed in Equation 6.117:

$$\rho_{ws} = \frac{(\lambda_{uc} + \lambda_{ws})}{s\mu_{ws}} = \frac{(20 + 160 \text{ Mbit/s})}{(3)(2830 \text{ Mbit/s})} = 0.02120. \tag{6.117}$$

Mean Upload Packet Processing Time ts_{ws}

This mean time is computed in Equation 6.118 by dividing the Web page request packet size, d, by the processing rate, accounting for number of servers, s:

$$ts_{ws} = \frac{d}{s\mu_{ws}} = \frac{1000 \text{ bits}}{(3)(2830 \text{ Mbit/s})} = 0.11779 \text{ μs}. \tag{6.118}$$

Mean Download Web Page Processing Time ts_{wd}

This mean time is computed in Equation 6.119 by dividing the Web page size, p, by the processing rate, again accounting for number of servers, s:

$$ts_{wd} = \frac{p}{s\mu_{ws}} = \frac{96{,}928 \text{ bits}}{(3)(2830 \text{ Mbit/s})} = 11.417 \text{ μs}. \tag{6.119}$$

Mean Web Page Download Time t_{ucd}

This is the mean time required to download a Web page of size, p, using the Web page download rate, λ_{ws}, in Equation 6.120:

$$ts_{wd} = \frac{p}{s\mu_{ws}} = \frac{96{,}928 \text{ bits}}{(3)(2830 \text{ Mbit/s})} = 11.417 \text{ μs}. \tag{6.120}$$

Mean Web Page Upload Wait Time tw_{ws}

This mean time is computed in Equation 6.121 by subtracting the upload processing time computed in Equation 6.119 from the Web page request packet upload time, t_{uc}:

$$tw_{ws} = t_{uc} - ts_{ws} = 50 - 0.11779 \text{ μs} = 49.88 \text{ μs}. \tag{6.121}$$

Mean Web Page Download Wait Time tw_{wsd}

Using Equation 6.122, compute the mean Web page download wait time by subtracting the download processing time computed in Equation 6.119 from the Web page download time computed in Equation 6.120:

$$tw_{wsd} = t_{ucd} - ts_{wsd} = 605.8 - 11.417 \text{ μs} = 594.383 \text{ μs}. \tag{6.122}$$

Mean Number of Web Page Bits Being Processed in the Upload Direction ns_{ws}

This quantity is computed in Equation 6.123 by multiplying the probability of Web server queue busy, computed in Equation 6.117 by the Web page request packet size d:

$$ns_{ws} = \frac{(\lambda_{uc} + \lambda_{ws})(d)}{s\mu_{ws}} = \frac{(20 + 160 \text{ Mbit/s})(1000 \text{ bits})}{(3)(2830 \text{ Mbit/s})} = 21.20 \text{ bits}. \tag{6.123}$$

Mean Number of Web Page Bits Being Processed in the Download Direction ns_{wsd}

This quantity is computed in Equation 6.124 by multiplying the probability of Web server queue busy, computed in Equation 6.117 by the Web page size p:

$$ns_{wsd} = \frac{(\lambda_{uc} + \lambda_{ws})(p)}{s\mu_{ws}} = \frac{(20 + 160 \text{ Mbit/s})(96{,}928 \text{ bits})}{(3)(2830 \text{ Mbit/s})} = 2055 \text{ bits}. \tag{6.124}$$

Mean Number of Web Page Bits Waiting for Processing in the Upload Direction nw_{ws}

This quantity is computed in Equation 6.125 by subtracting the number of bits being processed in the upload direction ns_{ws}, computed in Equation 6.124, from the Web page request packet size d:

$$nw_{ws} = d - ns_{ws} = 1000 - 21.20 \text{ bits} = 978.80 \text{ bits}. \tag{6.125}$$

Mean Number of Web Page Bits Waiting for Processing in the Download Direction nw_{wsd}

This quantity is computed in Equation 6.126 by subtracting the number of bits being processed in the download direction ns_{wsd}, computed in Equation 6.125 from the Web page size p:

$$nw_{wsd} = p - ns_{wsd} = 96,928 - 2055 \text{ bits} = 94,873 \text{ bits}. \quad (6.126)$$

Again, statistical software is called upon to generate the distribution of wired system Web server bits being processed and waiting to be processed, using the above mean values.

COMPARISON OF PRESENT AND FUTURE WIRED INTERNET PERFORMANCE

Time in System

As can be seen by comparing the present wired Internet time in system with the proposed wired Internet time in system in Table 6.6, the latter's performance is dramatically better than the former. Table 6.6 contains the means and standard deviations of the performance times of the two systems. The significant performance advantage of the proposed Internet system, which is not burdened with local router and DNS overhead, is readily apparent, since both the means and standard deviations are lower. In addition, because the DNS in the present Internet only has to deal with upload traffic, its time in system is the lowest.

Table 6.6 Performance Comparison of Present and Proposed Future Wired Internets

	Present Internet		Proposed future Internet	
Performance metric	Mean (seconds)	Standard deviation (seconds)	Mean (seconds)	Standard deviation (seconds)
Wired local network time in system	0.655720	0.140187	Does not apply	Does not apply
DNS time in system	0.178284	0.007516	Does not apply	Does not apply
Wired Internet router time in system	1.194493	0.224831	0.258607	0.029701
Wired Internet Web server time in system	0.797331	0.283792	0.000814	0.000115
Total time in wired system	3.522262	0.584658	0.259421	0.029705

Table 6.7 Packet and Web Page Length Summary (Processed and Waiting to Be Processed)

Component	Present wired Internet (coefficient of variation)	Proposed future wired Internet (coefficient of variation)
Local network (upload)	0.9180	Does not apply
Local network (download)	0.8985	Does not apply
Local network router (upload)	0.9399	Does not apply
Local network router (download)	0.7516	Does not apply
DNS (upload)	0.9822	Does not apply
Internet router (upload)	0.7903	0.9510
Internet router (download)	0.6940	0.9059
Web server (upload)	0.7774	0.9623
Web server (download)	0.7382	0.8047

Internet Data Traffic

In addition to the time in system performance metric, it is important to include a metric that measures the variation in data flow. This metric is the coefficient of variation (standard deviation/mean) of the sum of bits being processed and bits waiting to be processed for each component in Table 6.7. This metric is computed because there is a great deal of variation in data flows, thus it is appropriate to normalize the standard deviation by the mean in order to obtain a representative picture of variation across the components. We see that the proposed wired network has consistently higher variation. This result is attributed to the fact that the proposed wired system has much higher Web request packet upload and Web page download rates, thus generating greater variation in data flows. The implication of this result is that higher data transfer rates achieved in the proposed wired system comes at a cost—lower stability of data traffic in the Internet.

COMPARISON OF PRESENT AND FUTURE WIRELESS INTERNET PERFORMANCE

As a reader exercise, for the proposed wireless Internet system, develop the equations for the mean values and performance prediction equations similar to what was done for the proposed Internet wired system. In addition, produce a future wireless Internet system simulation queuing model similar to Figure 6.15. Use the mean values contained in Table 6.8. In order to compare the present with the proposed wireless system, document the logic of the proposed wireless system, for both the upload and download directions. Notice that in contrasting the present wireless systems in Figure 6.2 (upload) and Figure 6.3 (download) with your proposed system, the difference is that the latter does not require the services of the DNS. In addition, all the mean value equations and the equations for the distribution of

Table 6.8 Performance Comparison of Present and Proposed Future Wireless Internets

	Present Internet		Proposed future Internet	
Performance metric	Mean (seconds)	Standard deviation (seconds)	Mean (seconds)	Standard deviation (seconds)
Wireless access point time in system	0.165121	0.044345	Does not apply	Does not apply
Wireless DNS	0.178284	0.007516	Does not apply	Does not apply
Wireless Internet router time in system	1.194089	0.225939	0.516090	0.032931
Wireless Internet Web server time in system	0.790009	0.281181	1.157510	0.233608
Total time in wireless system	2.151653	0.406204	1.852345	0.257725

processing and wait times remain the same, although the results of the distribution value equations will change based on different sets of probabilities of those distributions.

By eliminating the DNS from the present wireless simulation queuing model in the upload direction in Figure 6.2, you can produce the future wireless simulation queuing model. The components that would be included in the simulation queuing model are: access point, Internet router, and Web server, both upload and download. As in the case of the present *wireless* system, depicted in Figures 6.2 and 6.3, twice as many Web servers are required to maintain queue stability, compared with the present and proposed *wired* systems.

Eliminating the burden of DNS overhead on the proposed wireless system improves total system performance, as shown in Figure 6.16. However, not all advantages would necessarily accrue to the proposed wireless Web server because it could suffer from a higher queue load, which could be caused by a higher probability of queue being busy, compared with the present Web server. This result is demonstrated in Figure 6.17, where even the present wired system is superior to the proposed wireless system with respect to queue efficiency. As has been suggested, the result is caused by a higher queue load in the wireless system, which in turn is the result of high Internet activity generated by wireless devices.

Time in System

As was the case for the wired systems, we see in Table 6.8 that the proposed future wireless Internet has superior performance with respect to the time metrics, again as the result of not having to contend with the local network, local network router, and DNS overhead.

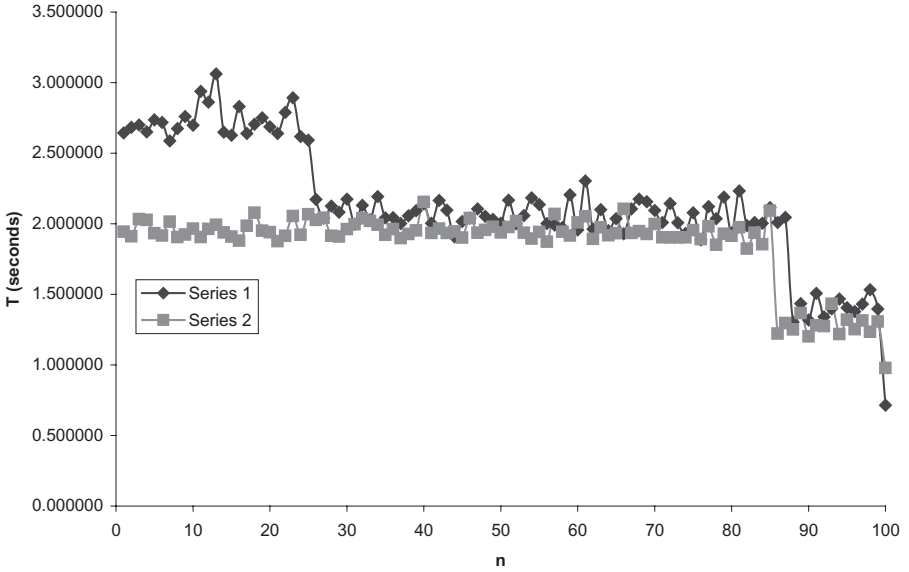


Figure 6.16 Time in system T versus number of uploads and downloads n. Series 1: Total present wireless system. Series 2: Total future wireless system.

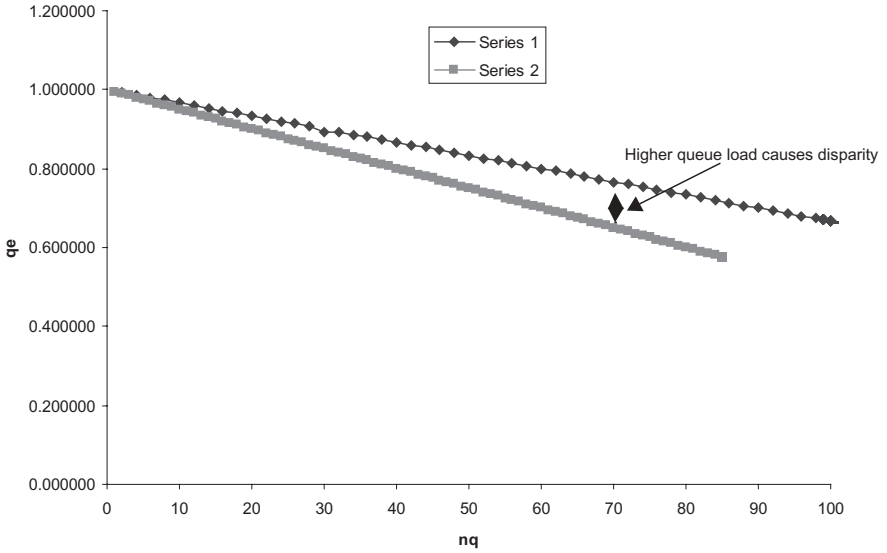


Figure 6.17 Queue efficiency qe versus queue count nq. Series 1: Present Internet wired Web server; probability of queue busy = 0.7688. Series 2: Future Internet wireless Web server; probability of queue busy = 0.8580 (i.e., higher queue load).

Table 6.9 Packet and Web Page Length Summary (Processed and Waiting to be Processed)

Component	Present wireless Internet (coefficient of variation)	Proposed future wireless Internet (coefficient of variation)
Access point (upload)	0.7849	0.6970
Access point (download)	0.7191	0.6554
DNS (upload)	0.9822	Does not apply
Internet router (upload)	1.1548	0.7611
Internet router (download)	0.9247	0.7643
Web server (upload)	0.7731	0.6774
Web server (download)	1.0310	0.7303

Internet Data Traffic

Again, as in the case of comparing present and proposed future *wired* systems, the Web page length variations of the present and proposed *wireless* system are compared in Table 6.9, where we see the advantage of the future Internet, again as the result of not being burdened by the variation in local network, local network router, and DNS data traffic.

SUMMARY

By eliminating the local network and its supporting router, and eliminating DNS name-to-IP address translation by virtue of providing every user computer and mobile device with its own IP address, both wired and wireless performance are predicted to be improved, as measured by time in the system. It is also useful to model data flows in the present and proposed Internets to gauge the relative variation in traffic across components. This process spotlights excessive component variation, for example, the present wireless Internet router in the upload mode in Table 6.9 that is reduced in the future Internet. It is recognized that major organizational changes (e.g., elimination of domain name administration) and technical changes (e.g., elimination of local networks and routers) are required in order to realize this vision of the future Internet. However, if the Internet were being built from scratch, it should closely resemble this proposal.

A performance metric that was not covered is the delay time among communicating devices. This is particularly the case for wireless devices, where both human-made and nature-made interference can have a significant effect on Internet performance [XYL99]. This factor should be included in enhanced future Internet proposals.

REFERENCES

[BAN] J. Bannister, J. Touch, P. Kamath, and A. Patel, "An Optical Booster for Internet Routers," University of Southern California, Information Sciences Institute, Los Angeles, California, U.S.A., http://www.isi.edu/touch/pubs/hipc2001

[FAI08] G. Fairhurst, B. Collini-Nocker, and L. Caviglione, "FIRST: Future Internet—a role for satellite technology," *International Workshop on Satellite and Space Communications, IWSSC 2008*, October 1–3, 2008, pp. 160–164.

[FINK98] R. A. Fink, "Reliability modeling of freely-available Internet-distributed software," *Proceedings of the Fifth International Software Metrics Symposium*, November 20–21, 1998, pp. 101–104.

[FOO95] M. A. Foody, "When is software ready for release?" *UNIX Review*, 1995, 13, pp. 35–41.

[GAN03] R. Gandhi, "Tolerance to access-point failures in dependable wireless local area networks," *The Ninth IEEE International Workshop on Object-Oriented Real-Time Dependable Systems*, October 1–3, 2003, pp. 136–136.

[GOE07] M. Siekkinen, V. Goebel, T. Plagemann, K.-A. Skevik, M. Banfield, and I. Brusic, "Beyond the future Internet—requirements of autonomic networking architectures to address long term future networking challenges," *11th IEEE International Workshop on Future Trends of Distributed Computing Systems*, March, 2007, pp. 89–98.

[HAM02] M. Hamblen, "10 Gig Ethernet: speed demon," Computer World: Networking & Internet, December 23, 2002.

[HIL01] F. S. Hillier and G. J. Lieberman, *Introduction to Operations Research*, 7th ed. New York: McGraw Hill, 2001.

[IEE90] IEEE, "IEEE Standard Glossary of Software Engineering Terminology," Number STD 610.12-1990, IEEE, 1990.

[INS] In-Stat Market Research Corporation, www.instat.com.

[ITU09] International Telecommunication Union, "New ITU ICT development index compares 154 countries." Press Release, http://www.itu.int/newsroom/press_releases/2009/07.html, March 2, 2009.

[IVA09] F. Ivanek, "Mobile backhaul," *IEEE Microwave Magazine*, 2009, pp. 10–20.

[JIN08] S. Jin, W. Yue, and N. Tian, "Performance analysis of self-similar traffic for future service-oriented Internet," *14th European Wireless Conference*, June 22–25, 2008, pp. 1–5.

[KAL99] M. Kalyanakrishnam, Z. Kalbarczyk, and R. Iyer, "Failure data analysis of a LAN of Windows NT based computers," *Proceedings of the 18th IEEE Proceedings of the Symposium on Reliable Distributed Systems*, 1999, pp. 178–187.

[LAB98] C. Labovitz and A. Ahuja, "Experimental study of Internet stability and wide-area backbone failure." Merit Network, Inc., 1998.

[MET03] C. Metz, "Moving toward an IPv6 future", *IEEE Internet Computing*, May/June, 2003, p. 25.

[NEU93] A. M. Neufelder, *Ensuring Software Reliability*. New York: Marcel Dekker, 1993.

[PAG] R. A. Page, Jr., M. Spence, and E. Tamson, "Customer loyalty and LAN failure: a positive relationship?" PDi Articles, Performance Dimensions International.

[PAP] V. Pappas, D. Massey, A. Terzis, and L. Zhang, "A comparative study of the DNS design with DHT-based alternatives," Computer Science Department, UCLA; Computer Science Department, Colorado State University; and Computer Science Department, Johns Hopkins University.

[PAR] K. Park, V. S. Pai, L. Peterson and Z. Wang, "CoDNS: improving DNS performance and reliability via cooperative lookups," Department of Computer Science, Princeton University.

[SHO83] M. L. Shooman, *Software Engineering: Design, Reliability, and Management*. New York: McGraw-Hill, 1983.

[SWA06] S. S. Gokhale, P. J. Vandal, and J. Lu, "Performance and reliability analysis of Web server software architectures," *12th Pacific Rim International Symposium on Dependable Computing*, December, 2006, pp. 351–358.

[TAK93] H. Takagi, *Queueing Analysis (Volume 3: Discrete-Time Systems)*. Amsterdam: North-Holland, 1993.

[TIA04] J. Tian, S. Rudraraju, and Li Zhao, "Evaluating web software reliability based on workload and failure data extracted from server logs," *IEEE Transactions on Software Engineering*, 2004, 30(11), pp. 754–769.

[TIP02] D. Tipper, T. Dalhberg, H. Shin, and C. Charnsripinyo, "Providing fault tolerance in wireless access networks," *IEEE Communications Magazine*, 2002.

[XYL99] G. Xylomenos and G.C. Polyzos, "Internet protocol performance over networks with wireless links," *IEEE Network*, 1999, 13(4), pp. 55–63.

Chapter 7

Network Standards

The primary objective of this chapter is to provide evidence of the fact that while there is a myriad of network standards, they are lacking in some fundamental properties as predictions of reliability, maintainability, and availability. For without these crucial properties being included in network standards, you would question the utility of existing standards. Existing standards do a reasonable job of specifying speed, range, signal properties, wireless device mobility, and compatibility requirements. However, to a large extent these properties are expressed sans user perspective. By user perspective I mean, for example, if a standard specifies a signal-to-noise ratio in the abstract, what does this mean in concrete terms for the user? Will the user be able to access a Web server, via the Internet, when desired, and reliably retrieve a Web page from the server within, say 5 seconds? Given the abstract nature of standards, my goal is to equip the reader with practical methods for designing and evaluating network standards with the goal of increasing user productivity in their use of computer networks.

DESIRABLE PROPERTIES OF NETWORK STANDARDS

First, the properties that are desirable in a network standard will be discussed and, second, a comparison will be made between desired properties and existing standard properties. Third, improvements designed to bring existing standards into conformance with desired properties will be identified. The reader may be surprised to learn that common requirements such as reliability specifications and the means for testing reliability are largely absent from current standards. Building on the foundation of network principles learned in Chapter 5, desirable properties of network performance, reliability, maintainability, and availability will be specified along with test procedures to ensure compliance with the specifications. These specifications will be formulated and used as a baseline for judging the utility of existing standards from the *user's perspective*.

Computer, Network, Software, and Hardware Engineering with Applications, First Edition. Norman F. Schneidewind.
© 2012 the Institute of Electrical and Electronics Engineers, Inc. Published 2012 by John Wiley & Sons, Inc.

Network Efficiency

The first desirable property to be addressed is network efficiency, defined as: (total time a packet spends in a network to achieve the user's goal)/(packet input time).

From Chapter 5, you learned that the numerator, T_t, is obtained as the summation of link delay, T_{ij}, node processing time, T_i, and wait time, W_i, summed over number of links, N_L, and nodes, N_n, in a network. The numerator is $T_t = \sum_{ij}^{N_L} T_{ij} + \sum_{i}^{N_n} T_i + \sum_{i}^{N_n} W_i$, and the denominator is $T_p = p/\lambda_{uc}$, where p is the Web page request packet size and λ_{uc} is the user computer packet input rate. Thus, efficiency E, which we desire to be as high as feasible, is defined as follows:

$$E = \left(\sum_{ij}^{N_L} T_{ij} + \sum_{i}^{N_n} T_i + \sum_{i}^{N_n} W_i \right) (\lambda_{uc}) / p.$$

Thus, you can see that what was initially formulated as *time efficiency* has become *data efficiency* because the above ratio is the quantity of data transmitted in a network to achieve the user's goal to the quantity of data inputted by the user; in other words, the number of bits outputted per input bit. The network performance evaluation model for making the efficiency test is shown in Figure 7.1. Figure 7.2 shows that only the Web Server Queue node is in conformance with the specification by virtue of having the smallest Web page request packet size, p, compared with the other nodes. Recall from Chapter 5 that p is generated by an exponential statistical routine. When this routine was applied in Chapter 5, it generated a small value of p

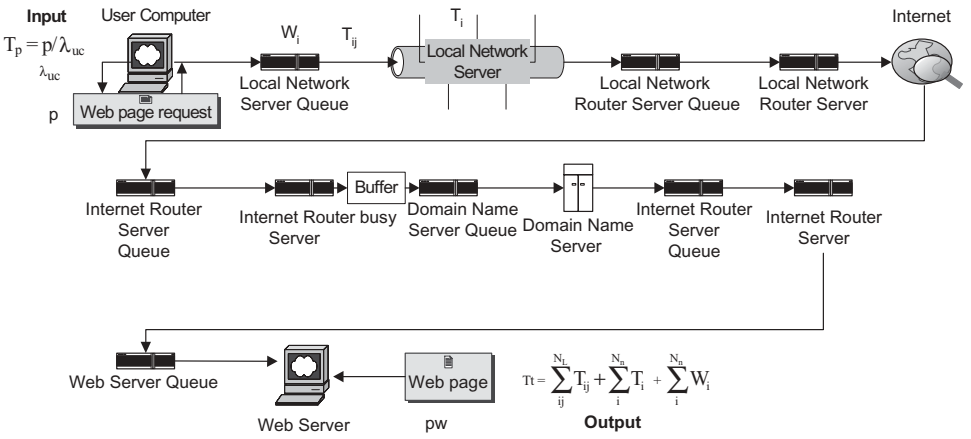


Figure 7.1 Network performance evaluation model. λ_{uc}, user computer input rate; p, Web page packet request size; T_{ij}, link ij processing time; W_i, node i wait time; T_p, packet input time; T_i, node i processing time; pw, Web page size. Efficiency = Output/Input = T_t/T_p.

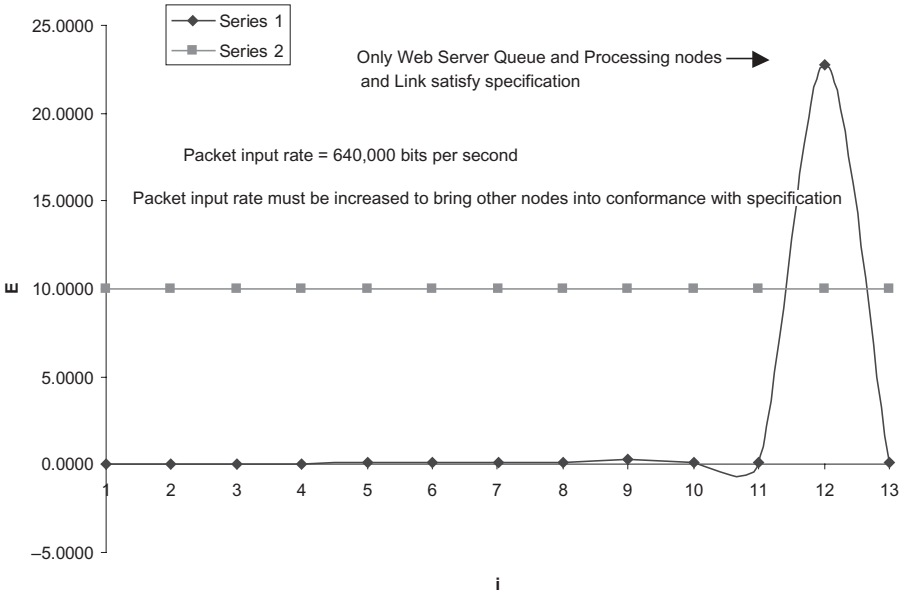


Figure 7.2 Network efficiency E versus node i. Series 1: E. Series 2: Specified E.

for the Web server queue (see Table 5.2, Chapter 5). This means that the result in Figure 7.2 is only one instance of computing efficiency. To obtain a more representative result you would repeat the process of generating p and computing values of E, say 100 times, and compute the mean value of E. Also, it is important to note, in Figure 7.2, that the packet input rate λ_{uc} must be increased to bring the other nodes into conformance with the specification.

Problem: What is the equation required to ensure that all nodes and associated links will satisfy the efficiency specification?

Solution: Solve the efficiency equation for λ_{uc}, the user computer input rate from Figure 7.1, as shown below, for *all* of the nodes and associated links, setting E to your desired value, and use the values of p from Table 5.2 in Chapter 5 for *each computation* of λ_{uc}. Use the *maximum* value of λ_{uc} as your solution. Show a plot of λ_{uc} versus node identification i:

$$(\lambda_{uc}) = (E * p) / \sum_{ij}^{N_L} T_{ij} + \sum_{i}^{N_n} T_i + \sum_{i}^{N_n} W_i.$$

Using an E = 10.0, the largest value of λ_{uc} = 950,118,764 bits per second. The required plot is shown in Figure 7.3. The *maximum* value of λ_{uc} corresponds to the User Computer and Local Network Server Queue nodes and the link between them (see node and link identifications in Fig. 7.4).

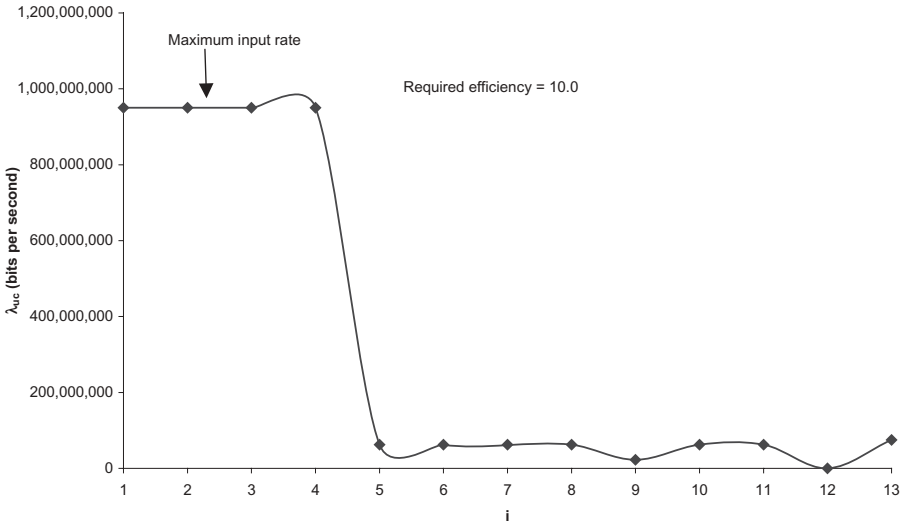


Figure 7.3 User Computer input rate λ_{uc} versus node or link i.

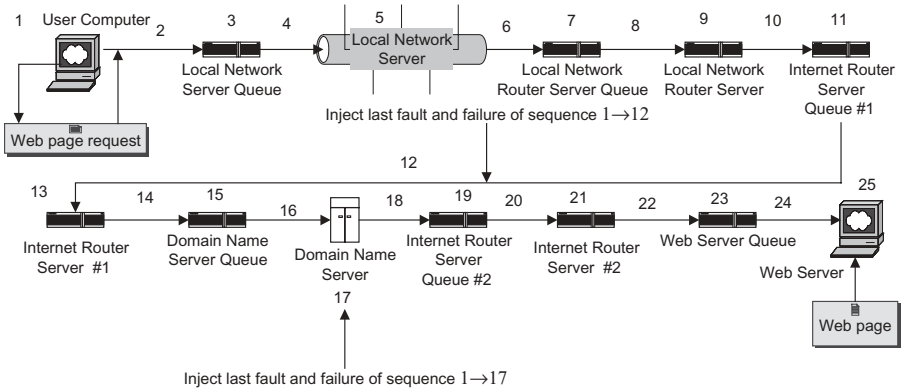


Figure 7.4 Reliability simulation process diagram.

RELIABILITY PREDICTION PROCESS

The reason reliability prediction is important is that national and international regulations may require that reliability specifications be achieved in network systems [MAR10]. The approach to reliability prediction is to simulate the injection of faults and failures into a replica of a computer network. Faults such as garbled data on a link, attributed to a noisy communication channel, cause failures such as the inability to transmit data between two nodes. Injection of faults and failures is simulated by randomly selecting links and nodes to be injected. This is accomplished by using the Excel RAND function (uniformly distributed random numbers between 0 and

1) to generate random numbers that will identify where injection is to take place. Specifically, since there is a total of 25 nodes and links, counting queues, in Figure 7.1, RAND is multiplied by 25 and then rounded to obtain the identification of injection location. Injection locations are identified in Figure 7.4, which will be used for simulating network reliability. Note that the Internet icon in Figure 7.1, which is included for clarity of presentation, is not shown in Figure 7.4 because Internet nodes and links are obviously components of the Internet; to show the Internet icon would involve double counting of nodes and links. In addition, the buffer in Figure 7.1, which is included for clarity of presentation, is not carried over into Figure 7.4 because the buffer is part of the Domain Name Server, which, again, would result in double counting.

Table 7.1 is provided to document the results of the node and link identification process, where, for example, the link Internet Router Server Queue → Internet Router Server is identified as number 12. Therefore, this link receives a fault and failure injection in Figure 7.4.

Estimation of Failure Rate

Continuing the process of reliability prediction, failure rate must be estimated. This will be an interesting exercise in failure rate estimation because the failure rate will be estimated for each sequence of the reliability simulation. The rationale for sequences is that end-to-end transmission and processing of data in a network is comprised of subsets of the total end-to-end chain called sequences. One of the principles of computer engineering is to reduce the complexity of analysis by decomposing a system into its component parts. If this is not done, the complexity of large systems overwhelms the engineer, leading to errors in analysis. To illustrate the sequence prediction process, the *first* estimate pertains to the *last* fault and failure of the *first* sequence injected at point 12 in Figure 7.4; the *second* estimate pertains to the *last* fault and failure of the *second* sequence injected at point 17, and so on. Duplicate sequences that may be generated by the random designation of injection points are not repeated because this would bias the results in favor of repeated sequences.

The failure rate λ pertaining to each simulation test of a *sequence* of fault and failure injection is estimated by summing the node and link times T_i over the N nodes and links, from node 1 to the last node or link of the sequence, N, where there has been an injection, and dividing it into the number of failures occurring over the sequence. For example, for injection at point 12 in Figure 7.4, T_i and n_i are summed from 1 to N = 12, where n_i is the number of failures *expected* in node or link i. The sequence failure rate equation follows:

$$\lambda = \frac{\sum_{i=1}^{N} n_i}{\sum_{i=1}^{N} T_i}.$$

Table 7.1 Simulated Reliability Testing Data

Sequence (node 1 to designated last link or node)	Name of *last* link or node in sequence	Sequence time $\sum_{i=1}^{N} T_i$ (seconds)	Sequence number of failures $\sum_{i=1}^{N} n_i$	Sequence failure rate λ (failures per second)	Mean sequence reliability
$1 \rightarrow 12$ Sequence 1	Internet Router Server Queue #1 to Internet Router Server #1	0.00037643	0.00071903	1.91014845	0.99999950
$1 \rightarrow 17$ Sequence 2	Domain Name Server	0.00088427	0.00104017	1.176295912	0.99999930
$1 \rightarrow 6$ Sequence 3	Local Network Server to Local Network Router Server Queue	0.00001695	0.00030732	**18.12701883**	0.99999802
$1 \rightarrow 15$ Sequence 4	Domain Name Server Queue	0.00060038	0.00091967	1.531828617	0.99999854
$1 \rightarrow 19$ Sequence 5	Internet Router Server Queue #2	0.00131913	0.00116259	0.881333876	0.99999800
$1 \rightarrow 11$ Sequence 6	Internet Router Server Queue #1	0.00019802	0.00063770	3.220345624	0.99999654
$1 \rightarrow 24$ Sequence 7	Web Server Queue to Web Server	0.01803274	0.00145202	0.080521348	0.99999841

1 → 4 Sequence 8	Local Network Server Queue to Local Network Server	0.00001180	0.00020174	**17.10034815**	0.99999451
1 → 18 Sequence 9	Domain Name Server to Internet Router Server Queue #2	0.00131911	0.00106875	0.810203027	0.99998897
1 → 21 Sequence 10	Internet Router Server #2	0.00146112	0.00125116	0.856303528	0.99998357
1 → 1 Sequence 11	User Computer	0.00000023	0.00008512	**364.3322654**	0.99999955
1 → 2 Sequence 12	User Computer to Local Network Server Queue	0.00000491	0.00009970	**20.3201531**	0.99999703
1 → 5 Sequence 13	Local Network Server	0.00001204	0.00027378	**22.73376925**	0.99999557
1 → 10 Sequence 14	Local network Router Server to Internet Router Server Queue #1	0.00019801	0.00056255	2.840988169	0.99996549
1 → 13 Sequence 15	Internet Router Server #1	0.00037644	0.00079729	2.117963322	0.99969476
1 → 20 Sequence 16	Internet Router Server Queue #2 to Internet Router Server #2	0.00146111	0.00115333	0.789351264	0.99976384

The determination of failure occurrence is performed by again employing the RAND function for estimating the number of failures, n_i, at each node and link, using a uniformly distributed number between 0 and 1. Of course, it is recognized that in the real world the number of failures must be an integer value. However, as stated, n_i is an *expected* value, justified by the fact that over a *large* number of operations in the real-world network, an integer number of failures would occur over the nodes and links. If the mean of these integer values were computed, the result would be the *expected* fractional value.

Reliability Prediction

Reliability is based on the *sequence* failure rate estimated above and the node or link times, Ti, from node 1 to the node or link where the last fault and failure injection occurs in a *sequence* at N. Thus, for each simulation test, there will be predictions from 1 to N. Again, the exponential distribution is used because there is a high probability of short node and link times and low probability of long times. Then, the sequence i reliability, R_i, is formulated as follows:

$$R_i = e^{(-\lambda T_i)}.$$

Since there will be many values of reliability in a sequence—one for each node or link—the mean value is computed in order to generate an overall sequence reliability metric. The mean values are tabulated in Table 7.1. This table will be used to identify possible low sequence reliability values that would be indicative of low values of node and link reliabilities.

Analysis of Table 7.1 reveals that since all of the mean sequence reliability values are very high, the prediction is that there should be no problem with reliability per se in actual operation. However, note that some of the failure rates are relatively high (bolded). In particular, this is the case for sequences associated with local network components. A possible reason for this is that local network components operate faster than Internet components. The higher speed can result in failures occurring at a higher rate. Thus, you can see that Table 7.1 is valuable in pinpointing reliability weak spots in a network. Notice that Table 7.1 results are consistent with the results in Figure 7.2, where only the Web Server satisfies the efficiency requirement. That is, both network performance—as measured by efficiency—and reliability are better at the service end of a network than at the input request end, suggesting that network standards should focus on local networks.

Maintainability Prediction

Recalling from Chapter 5 that maintainability was formulated as a probability, and this probability was the ratio of the quantity of data processed by a given link or node to the total quantity of data processed at all links and nodes in the network. Now, since reliability has been predicted using network entities called sequences, maintainability will also be predicted using sequences in order that availability,

which is a function of reliability and maintainability, will have consistent inputs for its prediction in a later section. Thus, maintainability will be formulated as a ratio of the quantity of data processed by a given *sequence* of nodes and associated links (e.g., Local Network Server Queue and the link between it and the User Computer) to the total quantity of data processed at all nodes and associated links in a network. Note that nodes and their associated links process the same quantity of data. Therefore, the maintainability of sequence i, M_i, is predicted as follows, where p_i is the quantity of data transmitted or processed in a node and the associated link, N_i is the number of nodes and associated links in sequence i, and N is the total number of nodes and associated links in a network:

$$M_i = \frac{\sum_{i=1}^{N_i} p_i}{\sum_{i=1}^{N} p_i}.$$

Availability Prediction

Similar to the formulation of availability in Chapter 5, the availability of sequences represents the probability that the set of nodes and links that comprise a sequence will be available for operational use when needed. Equivalently, availability is the proportion of operational time that maintenance is *not* being performed on a sequence (i.e., the sequence is operating reliably). Thus, availability, A_i, of a sequence is predicted as follows:

$$A_i = R_i/(R_i + M_i).$$

Figure 7.5 shows the results of applying the availability prediction equation, results that are opposite to those obtained for failure rate in Table 7.1 that showed local network sequences with relatively high failure rates, whereas Figure 7.5 shows that local network sequences are the only ones that satisfy the availability requirement. What accounts for the discrepancy? Checking Table 5.2 of Chapter 5, which records Web page request and Web page size, we see that local network components have smaller sizes that are the primary driver of maintenance actions. The lesson to be learned from this exercise is that multiple dimensions of network quality must be evaluated. If any one is deficient, it is a signal that network quality should be improved. In this case, local network sequences would be subject to further testing to discover and remove additional faults.

Storage Capacity Prediction

Due to the fact that there is a great deal of data transmitted and processed in a network, storage requirements must be predicted. Since other metrics, such as availability, have been predicted on the basis of node and link sequences, consistency

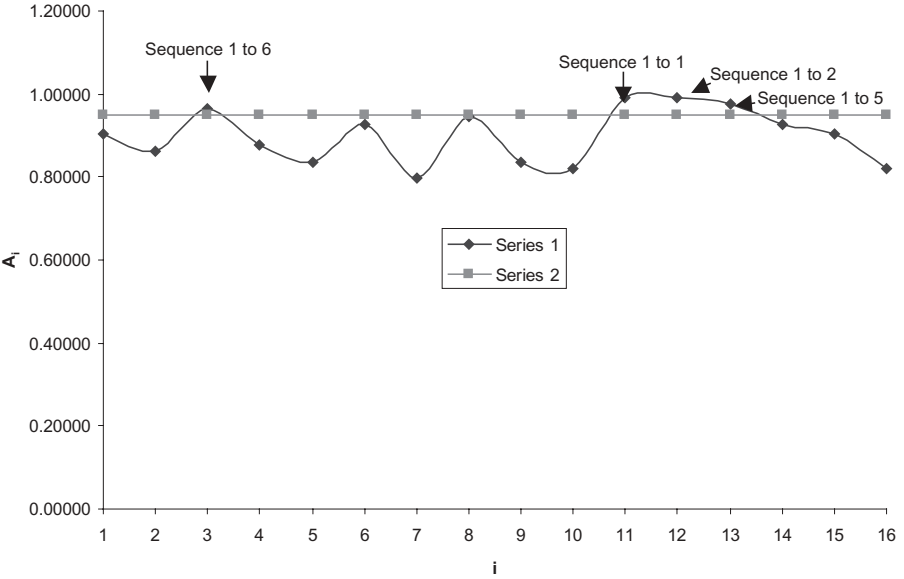


Figure 7.5 Sequence i availability A_i versus sequence i. Series 1: A_i. Series 2: Required A_i. Only local network nodes and links satisfy requirement.

requires that storage requirements predictions use the same approach. Thus, storage capacity is predicted by noting that data is injected into the network (i.e., sequences) in Figure 7.1 at a rate determined by the User Computer input rate, λ_{uc}. This sequence input rate multiplied by the sequence operational time, $\sum_{i=1}^{N} T_i$, generates the predicted sequence i storage capacity, S_i, and computed over the N nodes and links of the sequence, as follows:

$$S_i = (\lambda_{uc})\left(\sum_{i=1}^{N} T_i\right).$$

Figure 7.6 shows the results of predicting sequence storage capacity requirements, where the utility of the prediction is to delineate the maximum storage requirement, which in this case occurs in the Local Network Router Server Queue. This result is due to the relatively heavy traffic load in the local network. As a minimum, the Local Network Router Server Queue should be designed to accommodate this much data (in this case, about 12,000 bits).

Software Compatibility Standards Issue

An issue of great concern in network standards is whether various software systems that are required in a network are compatible, meaning that, for example, user

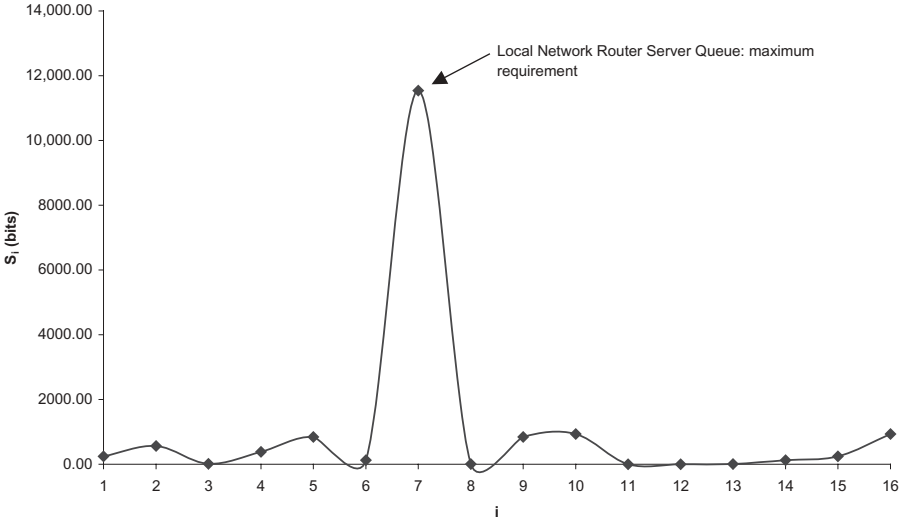


Figure 7.6 Sequence storage capacity S_i versus sequence i.

requests for Web pages in Figure 7.1 can traverse all links and nodes to the Web Server and return without disruption caused by software incompatibilities. Specifically, this refers to compatibility among the user operating system, Web browser, Ethernet local network protocol, router software—both local network and Internet—Internet Protocol (IP), Domain Name Server software, Web Server, database management system, and network security software. My motivation for including this issue is to identify for the reader improvements that would make standards more valuable for the *user*. There is no standards issue more important from the *user standpoint* than software compatibility because: (1) historically, software, due to its complexity, has caused more problems than hardware and (2) unfortunately, network standards do not address the software compatibility issue. This means that network users must insist on receiving compatibility information when considering purchase of network software from vendors.

EXISTING STANDARDS

At this point, existing standards will be reviewed to see to what extent, if any, these standards address performance, reliability, maintainability, availability, and software compatibility.

There are a lot of different network standards that the majority of computers use. There are standards for both physical hardware and for signaling. For example, IEEE 802.11g is a wireless networking standard. It includes specifications for the type of radio that is used, how strong the signal can be amplified, a standard set of encryption schemes, and so on.

Another standard is Ethernet, also known as IEEE 802.3. This is a standard for hardwired networks. It defines what types of wiring can be used, transmission power requirements, connector styles, and so on.

There are also protocols. As you learned in Chapter 5, Transmission Control Protocol (TCP)/IP is a protocol that operates at the transport layer of the seven-layer network. Note that TCP/IP is a protocol that interfaces with local network protocols such as Ethernet.

Today, virtually all networking standards are "open" standards, administered by a standards organization or industry group. Open standards are more popular than proprietary ones in the computer industry, and this is particularly the case for networking. In fact, the few technologies that have no accepted open standard have been losing ground to those with open standards, particularly in the areas of wireless local area and home networks (http://uk.answers.yahoo.com/question/index?qid=20091014025636AAOqcDy).

Open standards are useful for helping to mitigate the problem of software compatibility because with open standards, software developers can incorporate compatibility into standards. However, there is no assurance that individual developers will address all compatibility issues.

WIRELESS STANDARDS

As voice and data in wired networks increasingly converge to use the Internet, similar convergence is happening with wireless access networks [CHA07]. Many different wireless network standards have been developed or are under development for metropolitan area networks (MANs), local area networks (LANs), and personal area networks (PANs).

The wireless access networks are diverse but the major standards may be classified as belonging to either a group of public land mobile networks (PLMNs) owned by cellular phone operators or to another group of wireless networks under the IEEE 802 family of standards. The frequency spectrum used by these wireless systems includes both unlicensed and licensed bands. The cellular networks and systems are diverse, and efforts to standardize them include the 3G Wireless in the International Mobile Telecommunication 2000 (IMT-2000) standard.

While the cellular networks have been moving from voice networks toward the Internet packet network, the family of IEEE 802 wireless networks is attempting to achieve the higher quality that is required in voice and other real-time applications. Different wireless network systems have good technological reasons to exist. There are different power requirements, distance ranges, data rates, and carrier frequencies. Different systems are therefore needed to optimize the performance and cost according to different requirements. Note that there is no mention of reliability and other important metrics in this list!

The most widely implemented wireless network standards fall into two major groups. One group of wireless networks is the PLMN family of cellular networks. Another group of wireless networks are the IEEE802 family of standards.

PLMNs

The major wireless systems under the PLMN family are primarily operated by the cellular telecom service providers. Cellular systems are usually designed with maximum cell range exceeding 10 km, where a cell is a wireless geographical area that has access to an access point, which has, in turn, access to the Internet. However, the peak data rates may only be realized in favorable channel conditions, such as in those areas close to the base station, where a base station contains a transmitter and antenna for transmitting mobile device signals. Note that given the erratic channel conditions, reliability should be predicted under these conditions to have a useful standard.

Multimedia Services

One key issue in providing multimedia services over a wireless network is the quality-of-service (QoS) support in the presence of changing network connectivity. The concern here is user mobility and shared, noisy, highly variable, and limited wireless communication links. Most wireless standard organizations are revising existing standards or making new specifications to provide more bandwidth or QoS-related parameters and interfaces to meet requirements from highly demanded multimedia applications, such as wireless video phone and multimedia message systems [GAN04].

IEEE802 WIRELESS NETWORKS

An important group of wireless networks is the IEEE802 family of standards. PAN distance ranges are 10 m, for example the 802.15 Bluetooth standard. LAN distance ranges that are within 100 m are the 802.11 Wireless LAN standards with data rates of 11, 55, and 100 Mbit/s. MAN distance ranges are 3–8 km. While range is an important network standard attribute, it is meaningless if not accompanied by specifying the reliability that would be achieved at these ranges!

The 3G Wireless networks, which provide wireless access to global and metropolitan area data networks, are standardized according to the 3G Wireless requirements specified in IMT-2000. The IMT-2000 3G Wireless goals are summarized below [CHA07]. The purpose of describing this standard is to indicate what is currently feasible in this class of important wireless networks and what needs to be improved by more mature standards.

1. ***Enable global roaming.*** Allow a mobile device to be used anywhere in the world, without changing network cards. A noble objective, but currently infeasible because of differing wireless technologies in different parts of the world. This is a generalization of the software compatibility problem.
2. ***Use Standardized Interfaces.*** Use the same interface between mobile devices and applications across mobile device developers.

3. ***Support Multimedia Services.*** This requirement has evolved into a very mature set of services, given the extensive use of, for example, cameras, social networking, and Web site access via mobile devices.
4. ***Have Minimum Data Rates.*** A minimum of 144 kbps in a vehicular environment, 384 kbps in a pedestrian environment, and 2 Mbit/s in an indoor office environment; these specifications are not particularly useful because they are arbitrary with no justification provided. The performance methodology presented in this chapter should be used to *quantitatively* estimate these requirements. This example illustrates the deficiency in some network standards: specifying a requirement, while neglecting to provide a rationale.
5. ***Operate in Multiple Environments.*** Indoor, outdoor, vehicular, and satellite; this specification should be tested by subjecting the wireless system to operate in these environments and noting whether there is equal performance and availability across the environments.

A more advanced wireless network is 4G, which is designed to operate at 50–250 MBit/s. Among other capabilities, 4G supports TV broadcast and interoperates with the wired Internet.

Limited Range Wireless Network

It is instructive to consider a limited range wireless network because the network standard is decidedly different from its long-range cousin due to a different market objective. The example is the Bluetooth wireless network.

The Bluetooth network has no network infrastructure other than the nodes (i.e., mobile device) [CHA07]. A Bluetooth network, called a piconet, consists of one master node and up to seven slave nodes within the radio frequency range of about 10 m. Adjacent piconets may interconnect with each other through nodes in overlapping regions of the separate piconets to form a larger network. Bluetooth provides packet switching links. The total data rate is in the1 Mbit/s range.

Bluetooth provides rapid ad hoc connections without cable and without line-of-sight requirement. It uses small form factor, low power, and low cost devices. The use of low power enables longer battery life applications such as a personal data assistant (PDA). Applications include phones, pagers, modems, headsets, notebook computers, handheld personal computers, and digital cameras.

The salient issue in standardizing a network such as Bluetooth is to test it in the environment described above to ascertain whether connectivity, performance, and availability can be achieved in a limited range environment.

Spectrum Considerations

Signal interference in the available spectrum, particularly in wireless systems, is a network standards issue. The degree of interference that is tolerable in various geographical areas, using specified network hardware and software, should be specified

by signal-to-noise (S/N) ratio. Increasing the S/N ratio will increase the range of the wireless system.

TEST BED FOR TESTING NETWORKS

Having discussed a number of network-recommended performance and availability metrics and having reviewed existing network standards, it is now appropriate to show the reader how a test bed could be deployed to perform tests designed to ensure the networks adhere to proposed and existing standards. It is important that the test bed be automated [HOD99], as portrayed in Figure 7.7, where the test measurements are instrumented.

First, the important network requirements that would be subject to testing are listed. These are shown in the network test bed in Figure 7.7.

Compatibility of a local network, wired and wireless systems, with the Internet. Test software records a compatibility result if the signal is received. An incompatible result is recorded if the signal is not received.

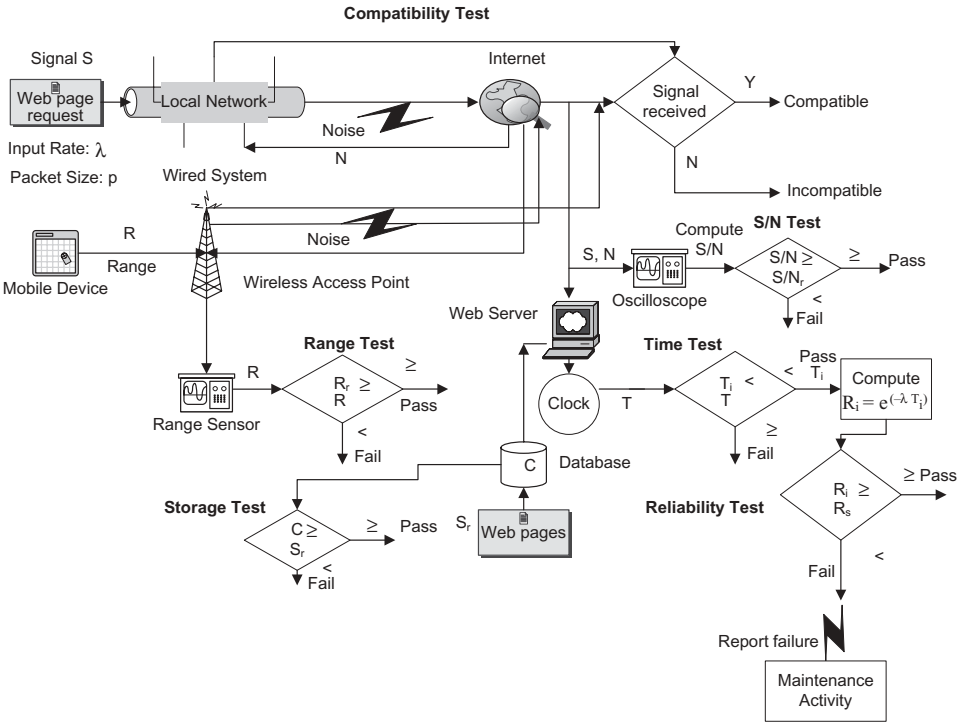


Figure 7.7 Network test bed. $T_i = p/\lambda$, required time; R_r, required range; S/N_r, required signal/noise; C, storage capacity; $R_i = e^{(-\lambda T_i)}$, required reliability; R_s, specified stability; S/N, actual signal/noise; S_r, storage requirement.

Time required to request a Web page from Web server. Test software computes required time and compares it with clock time.

Range required by mobile device in wireless network. A sensor attached to access point records the range between the mobile device and the access point. Test software compares the actual range with the received range.

Reliability is assessed following a successful time test by using test software that computes the required reliability, using the successful time obtained from the previous test, and compares it with the specified reliability. If the reliability test fails, the failure is reported to the maintenance activity [BAL89], as shown in Figure 7.7.

S/N ratio is tested by propagating the signal and noise to an oscilloscope where signal and noise are measured. The S/N ratio is computed and test software is used to compare the required ratio with the ratio actual generated in the network.

The *storage requirement* test is conducted with test software by comparing the database capacity with the Web page storage requirement.

The standards tests in Figure 7.7 are the major ones that can be quantified. Additional, important tests, such as the ability of a mobile device to roam and achieve connectivity, the use of compatible interfaces, and the ability of wireless networks to operate in multiple environments could also be tested in the test bed.

SUMMARY

The main point I wish to leave with the reader is that there are many more crucial factors involved in obtaining satisfaction in using networks than those factors contained in extant network standards. From the review of existing network standards, you can see that dwelling on speed, for example, is certainly not the whole story in assessing network standards. From the user's perspective, equally, if perhaps more, important are factors such as reliability, maintainability, and availability that are not *quantified* in existing standards reports [LEE06]. In addition, while standards developers may assume that the products to which their standards apply are reliable [SIE00], there is no guarantee of reliability without the type of testing shown in Figure 7.7. Therefore, it is important for the user acquiring networks to ascertain whether the network vendor has specified these crucial factors. Furthermore, the engineer charged with designing networks should include these crucial factors in the specifications and establish a test system, such as the one described in this chapter, for verifying that the specifications can be achieved.

REFERENCES

[BAL89] R. Ballart and Y.-C. Ching, "SONET: now it's the standard optical network," *IEEE Communications Magazine*, 1989, 27(3) pp. 8–15.

[CHA07] H. A. Chan, "Comparing wireless data network standards," *AFRICON*, 2007, pp. 1–15.

[GAN04] A. Ganz, Z. Ganz, and K. Wongthavarawat, *Multimedia Wireless Networks: Technologies, Standards, and QoS*. Upper Saddle River, NJ: Prentice Hall, 2004.

[HOD99] J. Hodges and J. Visser, "Accelerating wireless intelligent network standards through formal techniques," *IEEE 49th Vehicular Conference*, Volume 1, July 1999, p. 737.

[LEE06] C.-S. Lee, N. Morita, et al. "Next Generation Network Standards in ITU-T," *The 1st International Workshop on Broadband Convergence Networks*, 2006, pp. 1–15.

[MAR10] P. Mariño, F. Poza, and M. Á. Domínguez, "Instrumentation for an urban series-PHEV bus with onboard-based sensors and automotive network standards," *IEEE Transactions on Instrumentation and Measurement*, 2010, 59(7) pp. 1900–1910.

[SIE00] T. M. Siep, I. C. Gifford, R. C. Braley, and R. F. Heile, "Paving the way for personal area network standards: an overview of the IEEE P802.15 Working Group for Wireless Personal Area Networks," *IEEE Personal Communications*, 2000, 7(1), pp. 37–43.

Chapter 8

Network Reliability and Availability Metrics

Having been introduced to the basics of reliability and availability in Chapters 5 and 7, it is time to turn to developing a detailed, quantitative modeling methodology for predicting these variables in order to provide the reader with the tools that are needed to support complex network development.

Today, standalone computer applications are rare. Almost all applications involve a network—the Internet in particular. Models are important for analyzing the reliability and availability of networks. Therefore, in this chapter, you will learn how a model is developed for predicting the probability of failure, reliability, and availability in a network comprised of nodes, links, and subnetworks. This chapter provides a foundation for Chapter 15: "Mobile Device Engineering." In addition to developing the quantitative models, a template, or road map, is provided for modeling network reliability and availability. The process starts by defining the network topology and subnetwork configurations. This leads to identifying and defining the parameters and variables of the model. In developing the prediction equations, you define the sequence of operations on the network—for example, an input request to the Internet—that provides the basis for computing the reliability and availability of nodes, links, subnetworks, and network. Predicted failure and fault correction times are used to predict revised probability of failure, reliability, and availability that result from the correction process. These results are used to compute changes in these metrics that occur due to failure and fault correction. In addition, you examine the possibility of employing alternate network communication and processing paths to increase reliability and assess whether the increase in reliability is warranted by the increase in cost. With respect to model validity, you will find that reliability predictions for the network yield very low error values with respect to the actual network reliability (i.e., reliability computed from actual failure data).

Computer, Network, Software, and Hardware Engineering with Applications, First Edition. Norman F. Schneidewind.
© 2012 the Institute of Electrical and Electronics Engineers, Inc. Published 2012 by John Wiley & Sons, Inc.

INTRODUCTION

Network Metrics

Dependability of a system is the ability to deliver services that can be trusted. Dependability can be divided into three properties: availability, reliability, and maintainability. It is expected that a dependable system will be operational when needed (availability), that the system will keep operating correctly while being used (reliability), and that the correction of failure and faults will leave the network in a stable state (maintainability). This chapter covers reliability, availability, and maintainability by developing prediction equations for these metrics for network nodes, links, and subnetworks.

There are various perspectives on network reliability, all of which are useful. The perspective that is relevant depends on the characteristic to be emphasized, as follows, where reliability is defined by various researchers and comments are made concerning the relevance of the definitions to this chapter's perspective on reliability:

- Reliability is the ability of the network to provide communication in the event of a failure of a component in the network, such as a node or link, and it depends on the reliability of both hardware and software. Historically, failures were primarily due to hardware malfunctions. In current networks, many failures are due to fiber cable cuts, software faults, and malicious attacks [MED00]. Such failures can drop a significant number of existing connections. Thus the network should have the ability, with low delay, to detect and correct failures and the faults that cause the failures. The model in this chapter predicts the time required to correct failures and faults.
- Reliability is the maintenance of connectivity between nodes via their interconnected links, as shown in Figure 8.1. In this model, connectivity refers to the availability of a path from a source node to a destination node, for example, between nodes a and c, via link a, c, in Figure 8.1 [MEN].
- Reliability in interconnected networks is defined as survivability. That is, the network will not crash in the face of abnormal events. Reliability analysis depends on probability models of the failure rate, operating time duration, and severity of faults in hardware and software [NIC04]. In this chapter's model, reliability is cast in terms of probability of failure and associated reliability of the nodes and links in a subnetwork.
- Reliability is the probability of no permanent critical system failures during operating time t [ATH05]. Operating time is a key parameter in this chapter's model.

Software Dimension

It is claimed by some researchers that network hardware reliability is a mature field, and that there has not been an equal maturation of network software reliability

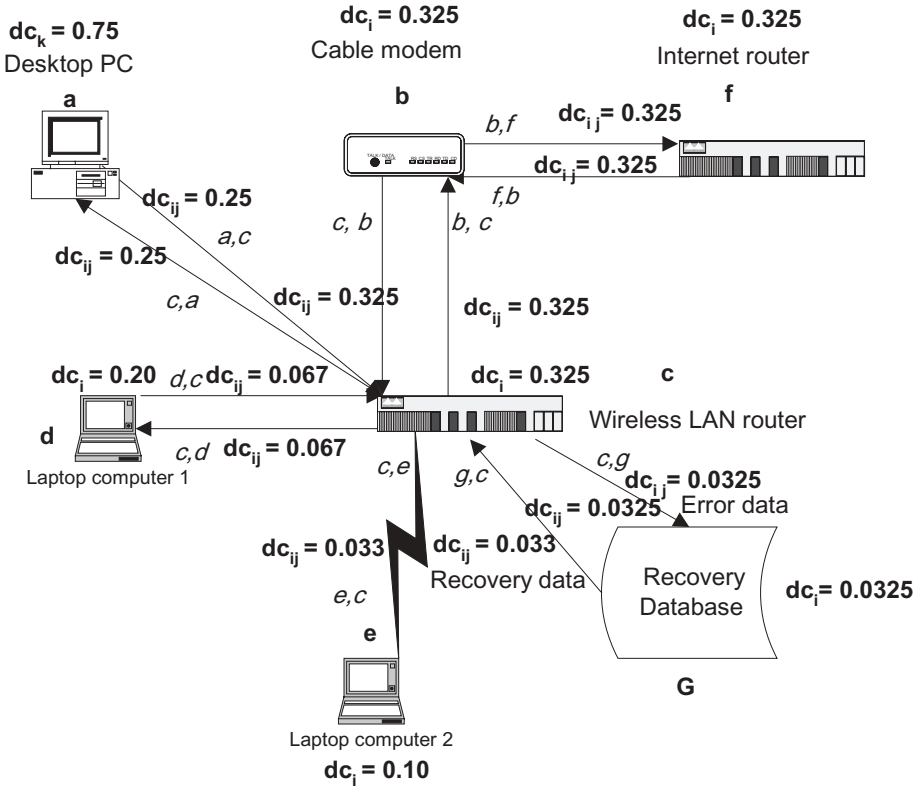


Figure 8.1 Network topology. dc_k, duty cycle of primary node; dc_i, duty cycle of nonprimary node; dc_{ij}, duty cycle of link.

[BEN07], despite the wealth of software reliability models and measurements [MUS04]. They claim that network software reliability quantification remains an open issue for a number of reasons. These include: confusion as to what to measure, when to measure it, and how to measure it. However, network users are not interested in software versus hardware reliability. They are just interested in enjoying reliable networks! Therefore, this chapter does not distinguish between hardware and software reliability in its model. Instead, this model uses failure rate and derivative metrics that include both hardware and software failures.

Model Tasks

The tasks that are required to develop this model were inspired by Chirivella et al. [CHI01], and are comprised of the following:

1. Define the network topology in Figure 8.1.
2. Define the subnetworks in Figures 8.2–8.4.

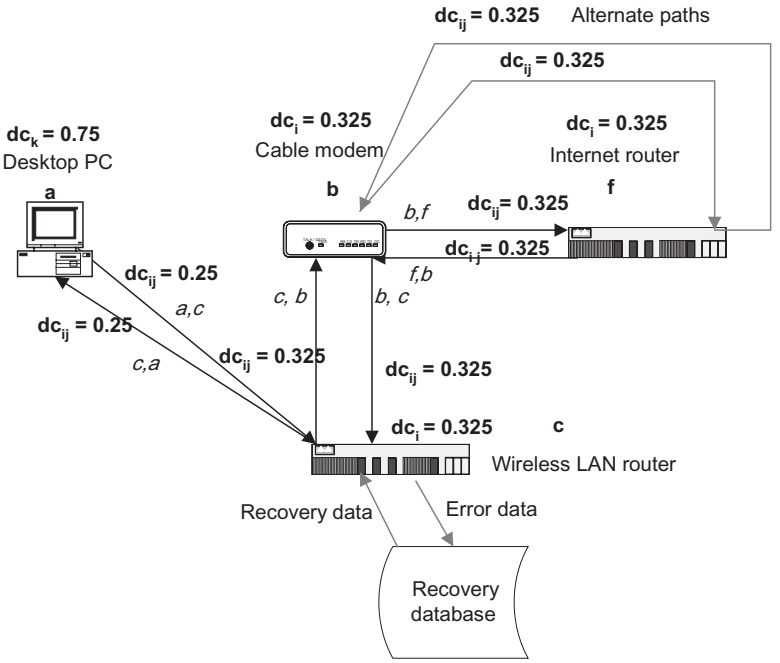


Figure 8.2 Subnetwork 1. dc_k, duty cycle of primary node; dc_i, duty cycle of nonprimary node; dc_{ij}, duty cycle of link.

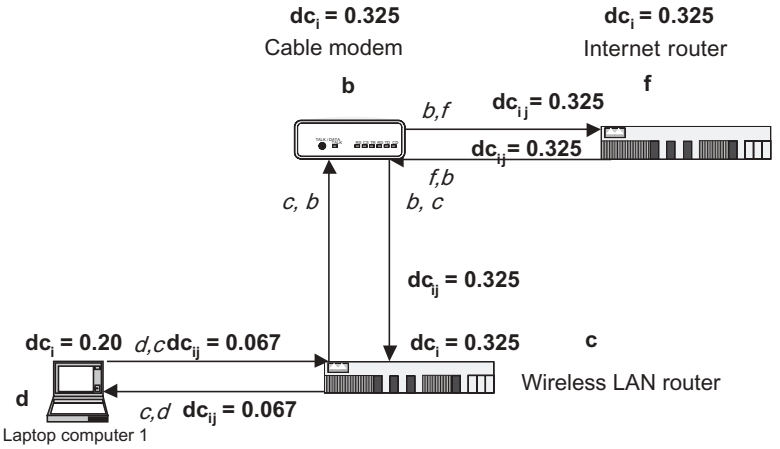


Figure 8.3 Subnetwork 2. dc_k, duty cycle of primary node; dc_i, duty cycle of nonprimary node; dc_{ij}, duty cycle of link.

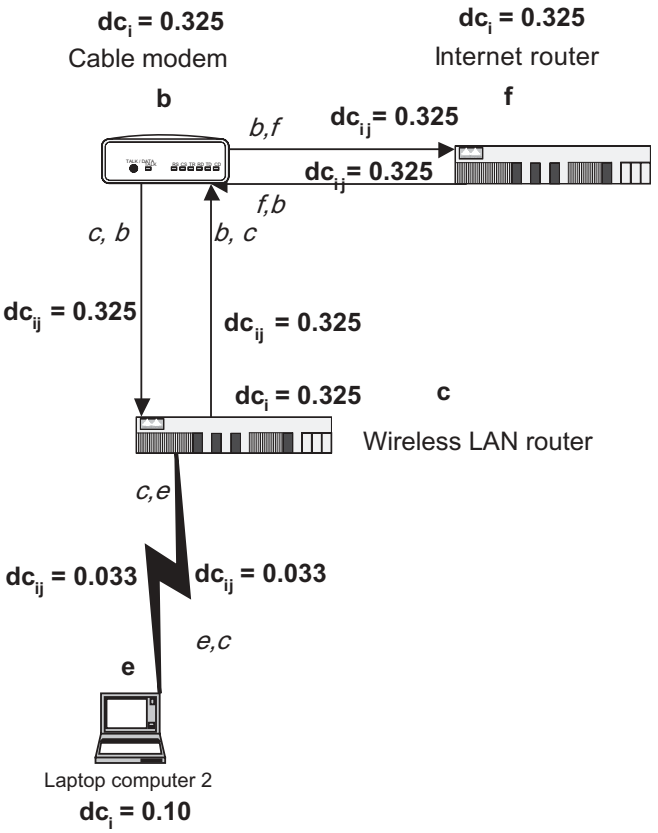


Figure 8.4 Subnetwork 3. dc_k, duty cycle of primary node; dc_i, duty cycle of nonprimary node; dc_{ij}, duty cycle of link.

3. Define network components, parameters, and variables, as follows:

Node and Link

Successful Operation of a Node. The node must be able to communicate with all nodes over the links to which it is connected [SHO02]. This definition is adopted by predicting the reliability and availability of the subnetworks in which a given node, and its connected links, must communicate.

Path. The sequence of nodes and links in a subnetwork representing transaction paths to and from the Internet (e.g., user's request for an Internet Web page, issued in a local network, and response provided by the Web site).

Successful Operation between a Pair of Nodes. One or more paths are operating correctly between the nodes [SHO02]. In this chapter's model,

it is not *any* path that is relevant but the *specific* path necessary to implement a network transaction.

Many terms are defined in the following sections. The reader need not focus on the definitions at this time. The terms are placed in here so that when studying the equations that use the terms, you will have one place to refer to the definitions, if necessary.

k: primary node: node that governs the operation of the network (e.g., desktop PC)

i: nonprimary node (e.g., cable modem)

Failure Rate and Failure Counts

Failure. One or more nodes cannot communicate with each other, either because there are no physical links between them, or because the Internet router in Figure 8.1 cannot select a route to reach the destination node [CHI01]. In addition, there can be failures internal to nodes, such as an operating system failure in a desktop computer.

f(t): network failure rate

$F(t_k)$: failure count at primary node k

$F(t_i)$: failure count at nonprimary node i

$F(t_{ij})$: failure count on link i, j

$F(t_s)$: failure count on subnetwork

$F(t_n)$: failure count on network

x: expected number of failures to occur in time t

M(f(t)): mean network failure rate

$M(t_k)$: mean failure count at primary node k

$M(t_i)$: mean failure count at nonprimary node i

$M(t_{ij})$: mean failure count on link i, j

$MR(t_k)$: revised mean failure count at primary node k based on failure correction

$MR(t_i)$: revised mean failure count at nonprimary node i based on failure correction

$MR(t_{ij})$: revised mean failure count on link i, j based on failure correction

Network Times

t: network operating time

t_k: primary node k operating time

t_i: nonprimary node i operating time

t_{ij}: operating time of link i, j

t_s: subnetwork operating time

tc_k: mean fault and failure correction time of primary node k

tc_i: mean fault and failure correction time of nonprimary node i

tc_{ij}: mean fault and failure correction time of link i, j

Duty Cycles

dc_k: duty cycle of node k = fraction of time t that node k is operational

dc_i: duty cycle of node i = fraction of available time t that node i is operational

dc_{ij}: duty cycle of link i, j = fraction of available time t that link i, j is operational

Probabilities

$P(F(t_k))$: probability of failure at primary node k

$P(F(t_i))$: probability of failure at nonprimary node i

$P(F(t_{ij}))$: probability of failure on link i, j

$P(F(t_s))$: probability of failure on subnetwork

$P(F(t_n))$: probability of failure on network

$PR(F(t_k))$: revised probability of failure at primary node k based on failure correction

$PR(F(t_i))$: revised probability of failure at nonprimary node i based on failure correction

$PR(F(t_{ij}))$: revised probability of failure on link i, j based on failure correction

$PR(F(t_s))$: revised probability of failure on subnetwork based on failure correction

$PR(F(t_n))$: revised probability of failure on network based on failure correction

Reliabilities

$R(t_k)$: reliability of primary node k

$R(t_i)$: reliability of nonprimary node i

$R(t_{ij})$: reliability of link i, j

$R(t_s)$: reliability of subnetwork

$R(t_n)$: reliability of network

$RR(t_k)$: revised reliability of primary node k based on failure correction

$RR(t_i)$: revised reliability of nonprimary node i based on failure correction

$RR(t_{ij})$: revised reliability of link i, j based on failure correction

$RR(t_s)$: revised reliability of subnetwork based on failure correction

$RR(t_n)$: revised reliability of network based on failure correction

Availabilities

$A(t_k)$: availability of primary node k

$A(t_i)$: availability of nonprimary node i

$A(t_{ij})$: availability of link i, j

$A(t_s)$: availability of subnetwork

$A(t_n)$: availability of network

$RA(t_k)$: revised availability of primary node k based on failure correction

$RA(t_i)$: revised availability of nonprimary node i based on failure correction

$RA(t_{ij})$: revised availability of link i, j based on failure correction

$RA(t_s)$: revised availability of subnetwork based on failure correction

$RA(t_n)$: revised availability of network based on failure correction

Faults and Failures Corrected

$N(t_k)$: number of faults and failures corrected in primary node k

$N(t_i)$: number of faults and failures corrected in nonprimary node i

$N(t_{ij})$: number of faults and failures corrected in link i, j

Remaining Faults and Failures

$r(t_k)$: number of faults and failures remaining in primary node after correction process

$r(t_i)$: number of faults and failures remaining in nonprimary node after correction process

$r(t_{ij})$: number of faults and failures remaining in link node after correction process

$r(t_s)$: number of faults and failures remaining in subnetwork after correction process

$r(t_n)$: number of faults and failures remaining in network after correction process

p: priority of failure and fault correction

4. Select metrics that quantify the reliability and availability characteristic that you want to study: Use probability of failure and actual and predicted reliability and availability.
5. Compute probability of failure for nodes, links, subnetworks, and network.
6. Use probability of failure to prioritize the order in which failure and faults are corrected on subnetworks.
7. Predict reliability and availability for subnetworks and network.
8. Predict failure and fault correction times for nodes and links.

9. Use correction times to predict remaining failures for subnetworks and network.
10. Use remaining failures to revise predictions of reliability and availability for subnetworks and network.
11. Determine whether the revised predictions satisfy the reliability and availability specifications.
12. If the specifications are not satisfied, additional testing is required to correct more failures and faults.
13. Determine whether the use of alternate network paths would increase reliability and availability to the extent that the cost of additional paths would be justified.

MODEL DEVELOPMENT

Table 8.1 shows how the duty cycle assignments for the example network topology shown in Figure 8.1 are obtained, starting with the primary nodes a, d, and e that are the drivers for nonprimary node and link duty cycle assignments. The data are illustrative only. For example, link a, c is assumed to be active one-third of the time that node a is active. Different data would apply to other topologies and applications. If you have data from an existing system, use it! Otherwise, you must make assumptions. You could vary the assumptions to see how sensitive network solutions are to the assumptions.

Actually, there is only one physical connection between pairs of nodes in Figure 8.1, but I show two links to account for the two-way flow of data. You can assume that the data flows between pairs of nodes are equal because the link speeds are equal in the two directions. Thus, the duty cycles for these links are equal. Figures 8.2–8.4 show subnetworks 1, 2, and 3, respectively, configured from Figure 8.1. It is these subnetworks that provide the platforms for the models to be described and analyzed.

Node and Link Operating Times

Operating time provides an accurate measure of fault discovery, is easy to measure, and reflects the time during which faults are discovered [DIS01]. Since operating times will be needed in the computation of failure rates in the next section, they are computed here for the primary nodes, nonprimary nodes, and links in Equations 8.1–8.3, respectively, where the duty cycles in Table 8.1 are multiplied by the *network* operating time:

$$t_k = t^*dc_k, \tag{8.1}$$

$$t_i = t^*dc_i, \tag{8.2}$$

$$t_{ij} = t^*dc_{ij}. \tag{8.3}$$

Table 8.1 Duty Cycle Assignments

		Link duty cycle						
		Node						
Node	Node duty cycle	a	b	c	d	e	f	g
a	**0.75**			0.75/3 = **0.225**				
d	**0.20**			0.20/3 = **0.067**				
e	**0.10**			0.10/3 = **0.033**				
c	0.225 + 0.067 + 0.033 = **0.325**	0.75/3 = **0.225**	0.225 + 0.067 + 0.033 = **0.325**		0.20/3 = **0.067**	0.10/3 = **0.033**		0.10 * 0.325 = **0.0325**
b	0.225 + 0.067 + 0.033 = **0.325**			225 + 0.067 + 0.033 = **0.325**			0.225 + 0.067 + 0.033 = **0.325**	
f	0.225 + 0.067 + 0.033 = **0.325**		0.225 + 0.067 + 0.033 = **0.325**					
g	0.10 * 0.325 = **0.0325**			0.10 * 0.325 = **0.0325**				

Failure Rates and Failure Counts

Using software and hardware failure data from the Computer Emergency Response Team (CERT) Web site [MOO01], based on a prominent router vendor's experience, the failure rate function in Equation 8.4 was fitted with these data, using network operating time t. Then, using Equations 8.1–8.3, the failure counts at primary nodes k, nonprimary nodes i, and links i, j, are computed in Equations 8.5–8.7, respectively:

$$f(t) = 0.0868e^{35t}, \tag{8.4}$$

$$F(t_k) = f(t)*t_k, \tag{8.5}$$

$$F(t_i) = f(t)*t_i, \tag{8.6}$$

$$F(t_{ij}) = f(t)*t_{ij}. \tag{8.7}$$

Probabilities of Failure

One of the measures of reliability is the probability of incurring failures at nodes and on links and subnetworks. We assume that the probability of failure is governed by the Poisson distribution. This assumption is justified because although there is a reason for failures, from the user's perspective, failures appear to occur at random (i.e., Poisson distribution [MUS87]).

First, the mean number of failures must be estimated for primary node, nonprimary node, and link, in Equations 8.8–8.10, respectively. Then, using these equations, the probabilities of failure at the primary node, nonprimary node, and link, are computed in Equations 8.11–8.13, respectively, where x = f(t)*t is the expected number of failures, based on the failure rate from Equation 8.4:

$$M(t_k) = \frac{\sum F(t_k)}{n_k}, \tag{8.8}$$

where n_k is the number of failure counts recorded for primary node

$$M(t_i) = \frac{\sum F(t_i)}{n_i}, \tag{8.9}$$

where n_i is the number of failure counts recorded for nonprimary node

$$M(t_{ij}) = \frac{\sum F(t_{ij})}{n_{ij}}, \tag{8.10}$$

where n_{ij} is the number of failure counts recorded for link

$$P(F(t_k)) = \frac{(M(t_k))^x e^{-M(t_k)}}{x!}, \tag{8.11}$$

$$P(F(t_i)) = \frac{(M(t_i))^x e^{-M(t_i)}}{x!}, \tag{8.12}$$

$$P(F(t_{ij})) = \frac{(M(t_{ij}))^x e^{-M(t_{ij})}}{x!}. \tag{8.13}$$

Reliabilities

You can use the Weibull distribution to model reliability because it has the flexibility of representing decreasing, constant, or increasing reliability over operating time, governed by the values of parameter α [LLO62]. You estimate the value α based on minimizing the mean relative error (MRE) between predicted and actual reliabilities. Error computations will be described later.

Reliabilities are the "bottom line" of quality in that they predict the probability that a node, link, subnetwork, or network will survive for a time greater than operating time t. The computation of the primary node, nonprimary node, and link reliabilities using Weibull distribution reliabilities [LLO62] are shown in Equations 8.14–8.16, respectively:

$$R(t_k) = e^{-(f(t)t_k^{\alpha})}, \tag{8.14}$$

$$R(t_i) = e^{-(f(t)t_i^{\alpha})}, \tag{8.15}$$

$$R(t_{ij}) = e^{-(f(t)t_{ij}^{\alpha})}. \tag{8.16}$$

Now using these reliabilities, compute the subnetwork reliability in Equation 8.17, based on a series configuration (nodes and links are connected in series in the subnetworks). These subnetworks are shown in Figures 8.2–8.4, where only the nodes and links that are relevant to the operation of the primary node are shown (i.e., the primary node a is *not directly* concerned with the recovery database node g in Fig. 8.1). Then, the network reliability is predicted in Equation 8.18, again using a series configuration:

$$R(t_s) = \prod_{k,i,ij} (R(t_k))(R(t_i))(R(t_{ij})), \tag{8.17}$$

$$R(t_n) \prod_{s} R(t_s). \tag{8.18}$$

In Figure 8.2, the subnetwork reliability is equal to the reliability of the path (a, c, b, f) in the upload direction, and the reverse path (f, b, c, a) in the download direction. Since these path reliabilities are equal, reliabilities are only computed for the upload direction—input request transaction to the Internet, reflecting the typical scenario of request to an Internet Web server.

Fault and Failure Correction

In formulating the fault and failure correction process, we assume the following: (1) one-to-one relationship between faults and failures and (2) the times required to

correct faults and failures are exponentially distributed (i.e., high probability of small correction times and low probability of large correction times). While it is true that a fault could spawn multiple failures, this is the minority case. These correction times are computed in Equations 8.19–8.23, for primary node, nonprimary node, link, subnetwork, and network, respectively, by using the mean number of failures divided by the mean failure rate M(f(t)):

Based on assumption (1), the correction rate equals the failure rate f(t) from Equation 8.4. Using assumption (2), the correction time probabilities in Equations 8.24–8.28 are based on the exponential distribution for primary node, nonprimary node, link, subnetwork, and network, respectively, where tc_k, tc_i, tc_{ij}, tc_s, and tc_n are the corresponding mean correction times:

$$tc_k = M(t_k)/M(f(t)), \tag{8.19}$$

$$tc_i = M(t_i)/M(f(t)), \tag{8.20}$$

$$tc_{ij} = M(t_{ij})/M(f(t)), \tag{8.21}$$

$$tc_s = M(t_s)/M(f(t)), \tag{8.22}$$

$$tc_n = M(t_n)/M(f(t)), \tag{8.23}$$

$$p(t_k) = f(t)e^{(-f(t)*tc_k)}, \tag{8.24}$$

$$p(t_i) = f(t)e^{(-f(t)*tc_i)}, \tag{8.25}$$

$$p(t_{ij}) = f(t)e^{(-f(t)*tc_{ij})}, \tag{8.26}$$

$$p(t_s) = f(t)e^{(-f(t)*tc_s)}, \tag{8.27}$$

$$p(t_n) = f(t)e^{(-f(t)*tc_n)}. \tag{8.28}$$

Once the correction time probabilities have been obtained, the next step is to compute the expected number of faults and failures that can be corrected. These quantities are computed by using correction time probabilities and the corresponding failure counts, for primary node, nonprimary node, link, subnetwork, and network, in Equations 8.29–8.33, respectively:

$$N(t_k) = p(t_k)*F(t_k), \tag{8.29}$$

$$N(t_i) = p(t_i)*F(t_i), \tag{8.30}$$

$$N(t_{ij}) = p(t_{ij})*F(t_{ij}), \tag{8.31}$$

$$N(t_s) = p(t_s)*F(t_s), \tag{8.32}$$

$$N(t_n) = p(t_n)*F(t_n). \tag{8.33}$$

Now that the number of faults and failures that can be corrected has been estimated, the remaining number of faults and failures are computed for primary node, nonprimary node, and link in Equations 8.34–8.36, respectively. Now, not all faults and failures are of equal priority for correction. An example of a serious one is a communication failure on links that connect the primary node to the Internet. An example of a nonserious failure is a transient failure, such as a file that does not initially show

on the desktop screen of the primary node but does appear after a refresh. The computation of the priority code is based on the *relative* value of the mean number of failures in the subnetworks. The priority code, pc, is applied to the nodes and links that comprise a subnetwork, as shown in Equations 8.34–8.36, to account for the *relative* priority of correcting remaining faults and failures.

The subnetwork remaining failures in Equation 8.37 is computed by summing remaining failures over primary nodes, nonprimary nodes, and links. Then, the network remaining failures in Equation 8.38 is computed by summing the remaining failures over subnetworks:

$$r(t_k) = (F(t_k) - N(t_k))*pc, \tag{8.34}$$

$$r(t_i) = (F(t_i) - N(t_i))*pc, \tag{8.35}$$

$$r(t_{ij}) = (F(t_{ij}) - N(t_{ij}))*pc, \tag{8.36}$$

$$r(t_s) = \sum_{k,i,ij} (r(t_k)) + (r(t_i)) + (r(t_{ij})), \tag{8.37}$$

$$r(t_n) = \sum_{n} r(t_s). \tag{8.38}$$

Revising Probabilities of Remaining Failures Based on Fault and Failure Correction

Once the remaining failures have been estimated, the revised probability of remaining failures for the primary node, nonprimary node, link, subnetwork, and network can be predicted by first computing the mean remaining failures in Equations 8.39–8.43 and substituting these values in Equations 8.44–8.48, respectively, and using x as the expected number of failures in the Poisson distribution of remaining failures:

$$MR(t_k) = \frac{\sum r(t_k)}{n}, \tag{8.39}$$

$$MR(t_i) = \frac{\sum r(t_i)}{n}, \tag{8.40}$$

$$MR(t_{ij}) = \frac{\sum r(t_{ij})}{n}, \tag{8.41}$$

$$MR(t_s) = \frac{\sum r(t_s)}{n}, \tag{8.42}$$

$$MR(t_n) = \frac{\sum r(t_n)}{n}, \tag{8.43}$$

$$PR(r(t_k)) = \frac{(MR(t_k))^x e^{-MR(t_k)}}{x!}, \tag{8.44}$$

$$PR(r(t_i)) = \frac{(MR(t_i))^x e^{-MR(t_i)}}{x!}, \tag{8.45}$$

$$PR(r(t_{ij})) = \frac{(MR(t_{ij}))^x e^{-MR(t_{ij})}}{x!}, \tag{8.46}$$

$$PR(r(t_s)) = \frac{(MR(t_s))^x e^{-MR(t_s)}}{x!}, \tag{8.47}$$

$$PR(r(t_n)) = \frac{(MR(t_n))^x e^{-MR(t_n)}}{x!}. \tag{8.48}$$

Revising Reliabilities Based on Fault and Failure Correction

Since the failure count has been reduced by the correction process, it is necessary to revise the reliabilities of the primary nodes, nonprimary nodes, and links, using the remaining failures failure rates, $rf(t_k)$, $rf(t_i)$, and $rf(t_{ij})$, and Weibull distribution, as shown in Equations 8.49–8.51, respectively:

$$RR(t_k) = e^{-(rf(t_k)t_k^{\alpha}}, \tag{8.49}$$

where primary node failure rate $rf(t_k) = r_k(t)/t_k$ (remaining primary node failures/primary node operating time);

$$RR(t_i) = e^{-(rf(t_i)t_i^{\alpha}}, \tag{8.50}$$

where nonprimary node failure rate $rf(t_i) = r_i(t)/t_i$ (remaining nonprimary node failures/nonprimary node operating time); and

$$RR(t_{ij}) = e^{-(rf(t_{ij})t_{ij}^{\alpha}}, \tag{8.51}$$

where link failure rate $rf(t_{ij}) = r_{ij}(t)/t_{ij}$ (remaining link failures/link operating time).

Next, the revised subnetwork reliability can be predicted in Equation 8.52 as follows, using Equations 8.49–8.51 and a series configuration:

$$RR(t_s) = \prod_{k,i,ij} (RR(t_k))(RR(t_i))(RR(t_{ij})). \tag{8.52}$$

Then, by using Equation 8.52 and a series configuration, the revised network reliability is predicted in Equation 8.53:

$$RR(t_n) = \prod_{s} RR(t_s). \tag{8.53}$$

Availability Analysis

To predict availability of a new system, probabilistic models need to be formulated [MIL98], and you need to account for the downtime attributed to fault detection, isolation, and correction [EIS]. However, while these approaches are valid, it is easier to define availability as the expected fraction of time that a system is functioning acceptably [MUS04], or alternatively, as the fraction of time that the network delivers proper service (i.e., it is *not* engaged in correcting faults) during its operating time [ATH05].

In this chapter's network model, $t_k/M(t_k)$ is the mean time to failure for primary nodes, computed from the operating time, t_k, and the mean number of failures, $M(t_k)$, and tc_k is the mean fault correction time. Thus, using these quantities, the availability of the primary node is computed in Equation 8.54. Similarly, the availability of the nonprimary nodes and the links are computed in Equations 8.55 and 8.56, respectively. Next, the subnetwork availabilities are computed in Equation 8.57 as the product of primary node, nonprimary node, and link availabilities, using a series configuration. Finally, the network availability is computed in Equation 8.58 as the product of the subnetwork availabilities.

You must also account for revised availabilities, once faults and failures have been corrected, by using the mean remaining failure counts. This is accomplished for the primary node, nonprimary node, link, subnetwork, and network in Equations 8.59–8.63, respectively, using the means of remaining failures (MR) that result from fault correction actions:

$$A(t_k) = \frac{\frac{t_k}{M(t_k)}}{\frac{t_k}{M(t_k)} + tc_k} = \frac{1}{1 + ((tc_k / t_k)M(t_k))}, \tag{8.54}$$

$$A(t_i) = \frac{\frac{t_i}{M(t_i)}}{\frac{t_i}{M(t_i)} + tc_i} = \frac{1}{1 + ((tc_i / t_i)M(t_i))}, \tag{8.55}$$

$$A(t_{ij}) = \frac{\frac{t_{ij}}{M(t_{ij})}}{\frac{t_{ij}}{M(t_{ij})} + tc_{ij}} = \frac{1}{1 + ((tc_{ij} / t_{ij})M(t_{ij}))}, \tag{8.56}$$

$$A(t_s) = \prod_{k,i,ij} A(t_k)A(t_i)A(t_{ij}), \tag{8.57}$$

$$A(t_n) = \prod_{s} A(t_s), \tag{8.58}$$

$$RA(t_k) = \frac{\frac{t_k}{MR(t_k)}}{\frac{t_k}{MR(t_k)} + tc_k} = \frac{1}{1 + ((tc_k / t_k)MR(t_k))}, \tag{8.59}$$

$$RA(t_i) = \frac{\frac{tc_i}{MR(t_i)}}{\frac{tc_i}{MR(t_i)} + tc_i} = \frac{1}{1 + ((tc_i / t_i)MR(t_i))}, \tag{8.60}$$

$$RA(t_{ij}) = \frac{\frac{t_{ij}}{MR(t_{ij})}}{\frac{t_{ij}}{MR(t_{ij})} + tc_{ij}} = \frac{1}{1 + ((tc_{ij} / t_{ij})M(t_{ij}))}, \tag{8.61}$$

$$RA(t_s) = \prod_{k,i,ij} RA(t_k)RA(t_i)RA(t_{ij}), \tag{8.62}$$

$$RA(t_n) = \prod_{s} RA(t_s). \tag{8.63}$$

PROBABILITY OF FAILURE ANALYSIS RESULTS

Typically, systems are unstable as they boot up when many processes and applications are invoked concurrently; later, the systems stabilize, hence the reason for the decreasing probability of failure over operating time in Figures 8.5 and 8.6. Figure 8.5 shows the ranking of probability of failure for the subnetworks, using the mean failure count. The value of this figure is that it identifies the order in which failure and fault correction should take place for the subnetwork, according to the priority code.

Figure 8.6 shows that the revised probability of failure for the network does not become favorable (i.e., crosses the original probability of failure curve) until operating time $t > 17$. The implication is that the network must be operated for a considerable time before the effect of failure and fault correction occurs. This concept is reinforced in Figure 8.7, where the changes in probability of failure between original and revised are plotted for the subnetworks and network. Again, there is considerable delay before the changes occur in the favorable direction.

FAULT AND FAILURE CORRECTION ANALYSIS RESULTS

This analysis is directed toward answering the question: Are there correction time anomalies among the nodes and links such that a priority ranking for fault and failure correction should be established? In looking at Figure 8.8, the answer is "yes"

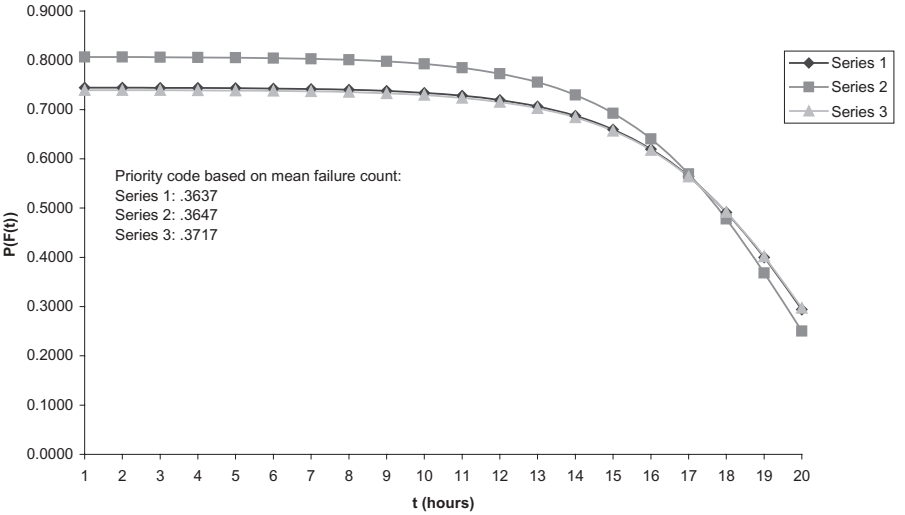


Figure 8.5 Original subnetwork probability of failure P(F(t)) versus operating time t. Series 1: subnetwork 1, mean failure count = 0.2953. Series 2: subnetwork 2, mean failure count = 0.2149. Series 3: subnetwork 3, mean failure count = 0.3018.

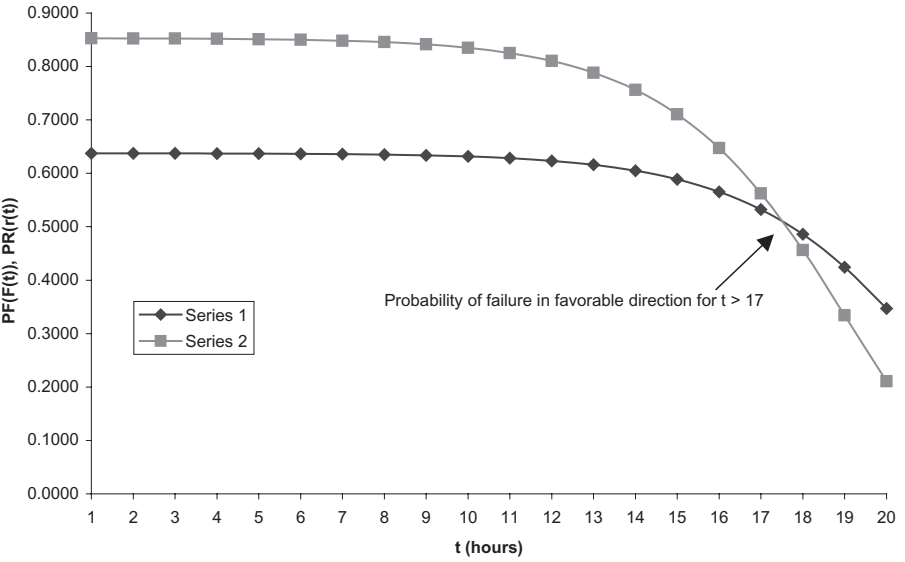


Figure 8.6 Original probability of failure P(F(t)) and revised probability of failure PR(r(t)) versus operating time t. Series 1: P(F(t)): network. Series 2: PR(rt)): network.

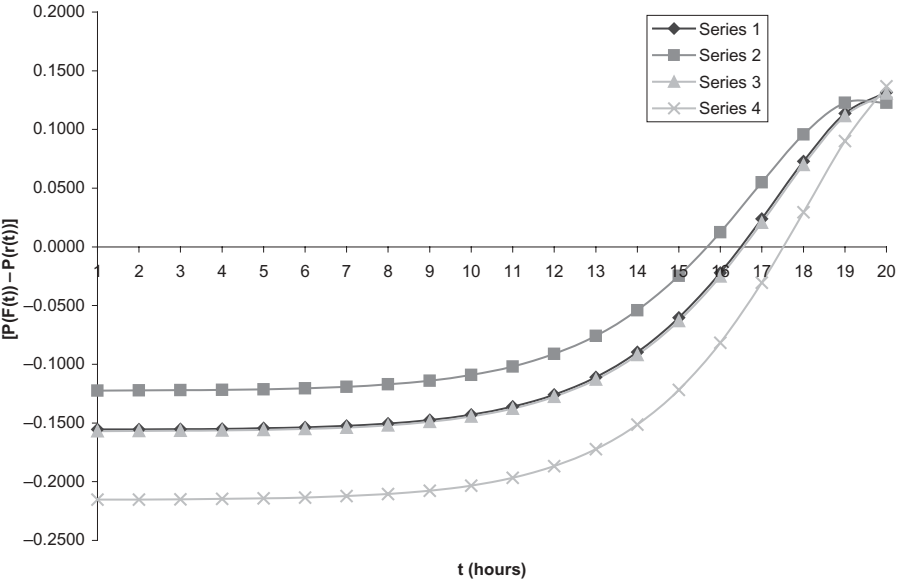


Figure 8.7 Change in probability of failure [P(F(t)) – P(r(t))] versus operating time t. Series 1: subnetwork 1: change in favorable direction for t > 16. Series 2: subnetwork 2: change in favorable direction for t > 15. Series 3: subnetwork 3: change in favorable direction for t > 16. Series 4: network: change in favorable direction for t > 17.

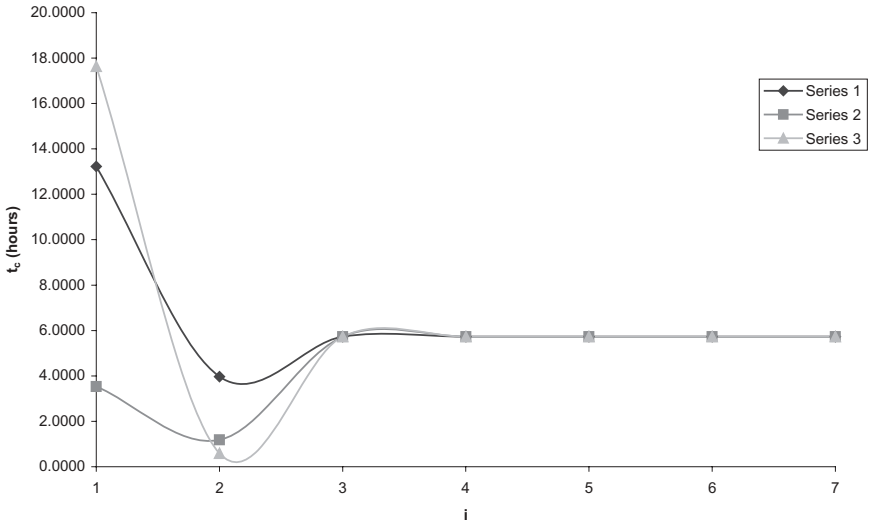


Figure 8.8 Subnetwork failure and fault correction time tc versus node and link identifier i. Series 1: subnetwork 1: i = 1: node a; i = 2: link ac; i = 3: node c; i = 4: link cb; i = 5: node b; i = 6: link bf, node f. Series 2: subnetwork 2: i = 1: node d; i = 2: link dc; i = 3: node c; i = 4: link cb; i = 5: node b; i = 6: link bf, node f. Series 3: subnetwork 3: i = 1: node e; i = 2: link ec; i = 3: node c; i = 4: link cb; i = 5: node b; i = 6: link bf, node f.

because there is considerable variation in the correction times for the primary node (i = 1) and nonprimary node (i = 2), and connecting link (i = 3), whereas the correction time stabilizes for the remainder of the subnetworks. The reason for the anomalies is that the software that is used in the home, such as desktop and laptop computers, is usually more difficult to debug than, for example, a cable modem, wherein a reset will usually clear the failure.

REMAINING FAILURES ANALYSIS RESULTS

An important metric for judging the reliability of a network system is predicted remaining failures. After all, predicted remaining failures of subnetworks and network represent residual problems that signal the need for further failure and fault correction. Figure 8.9 shows the relative effectiveness of the correction effort. The process has been most effective for subnetwork 2 and less effective for subnetworks 1 and 3, and the network.

RELIABILITY ANALYSIS RESULTS

After remaining failures have been predicted and plotted in Figure 8.9, as the result of failure and fault correction, reliability predictions, before and after failure and fault correction, can be analyzed in Figures 8.10 and 8.11, respectively, to identify

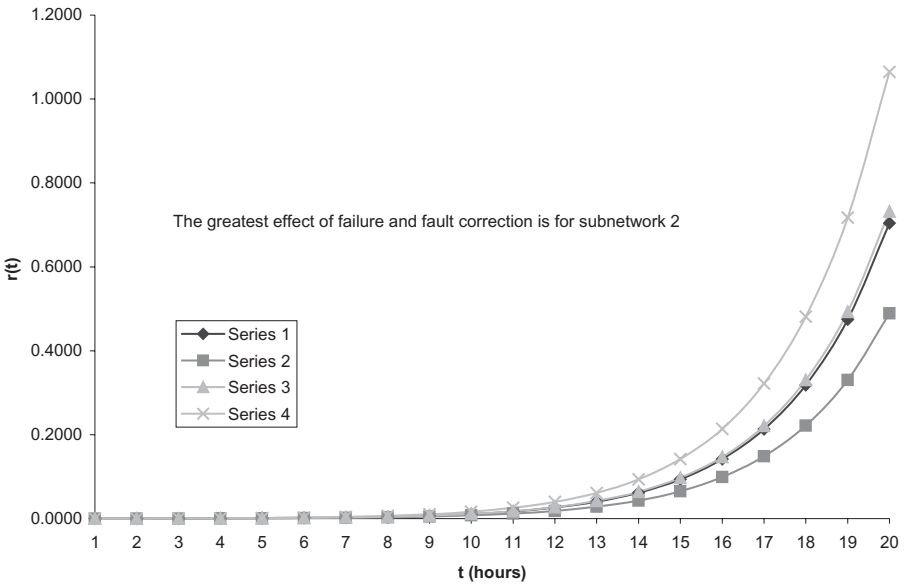


Figure 8.9 Remaining failures r (t) versus operating time t. Series 1: Subnetwork 1. Series 2: Subnetwork 2. Series 3: subnetwork 3. Series 4: Network.

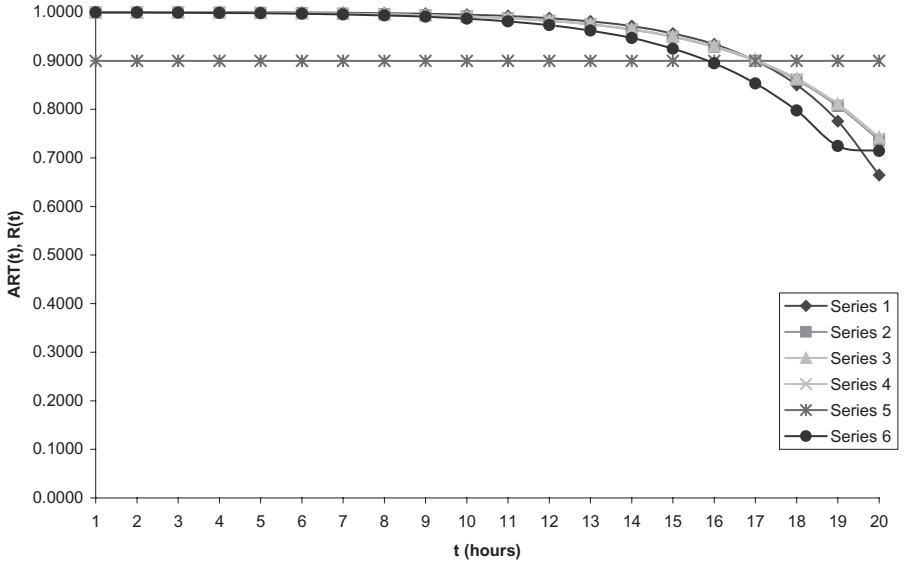


Figure 8.10 Original actual reliability AR(t) and predicted reliability R(t) versus operating time t. Series 1: AR(t). Series 2: R(t), subnetwork 1, fails specification for t > 17. Series 3: R(t), subnetwork 2, fails specification for t > 17. Series 4: R(t), subnetwork 3, fails specification for t > 17. Series 5: Specified reliability = 0.9000. Series 6: R(t), network, mean squared error (MSE) = 0.0140, fails specification for t > 16.

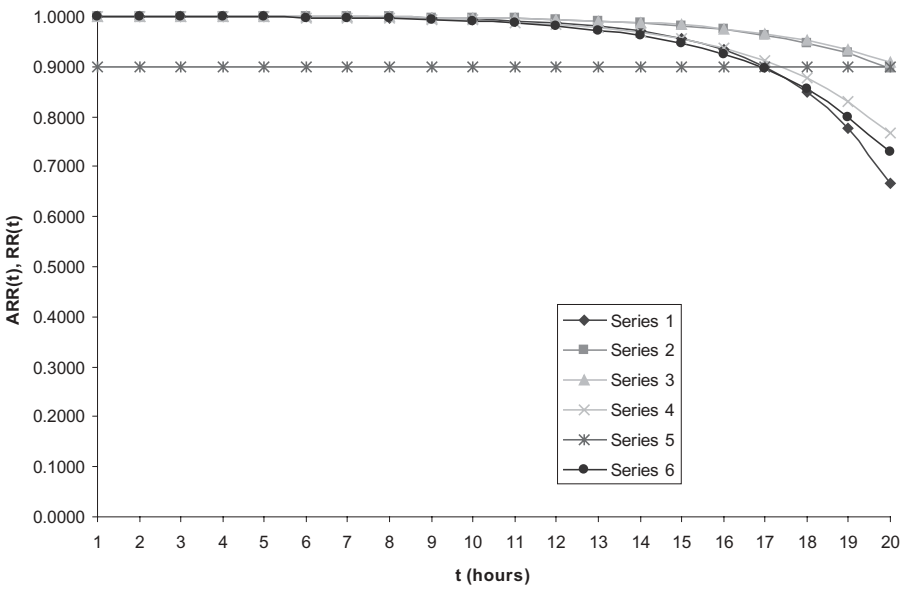


Figure 8.11 Revised actual reliability ARR(t) and predicted reliability RR(t) versus operating time t. Series 1: ARR(t). Series 2: RR(t), subnetwork 1, satisfies specification for all t. Series 3: RR(t), subnetwork 2, satisfies specification for all t. Series 4: RR(t), subnetwork 3, fails specification for t > 17. Series 5: specified reliability = 0.9000. Series 6: network, MSE = –0.0030, fails specification for t > 17.

improvements in reliability for the subnetworks. Comparing the figures, you can see that improvements in reliability occur for subnetworks 1 and 2, no improvement for subnetwork 3, and minor improvement in the network, as the result of failure and fault correction. Thus, the correction process has proved partially beneficial. Subnetwork 3 must be subjected to further fault correction. Also note that the predictions for the network yield very low MREs with respect to the actual network reliability. The value of Figures 8.10 and 8.11 is that network administrators can determine whether failure and fault correction efforts are likely to succeed.

AVAILABILITY ANALYSIS RESULTS

Figure 8.12 demonstrates that none of the subnetworks and the network satisfies the availability requirement. Therefore, action would be taken to correct failures and faults and then revise the availability predictions. The predictions are revised in Figure 8.13, where it is demonstrated that the failure and fault correction process has been very effective because now all subnetworks and the network satisfy the availability requirement.

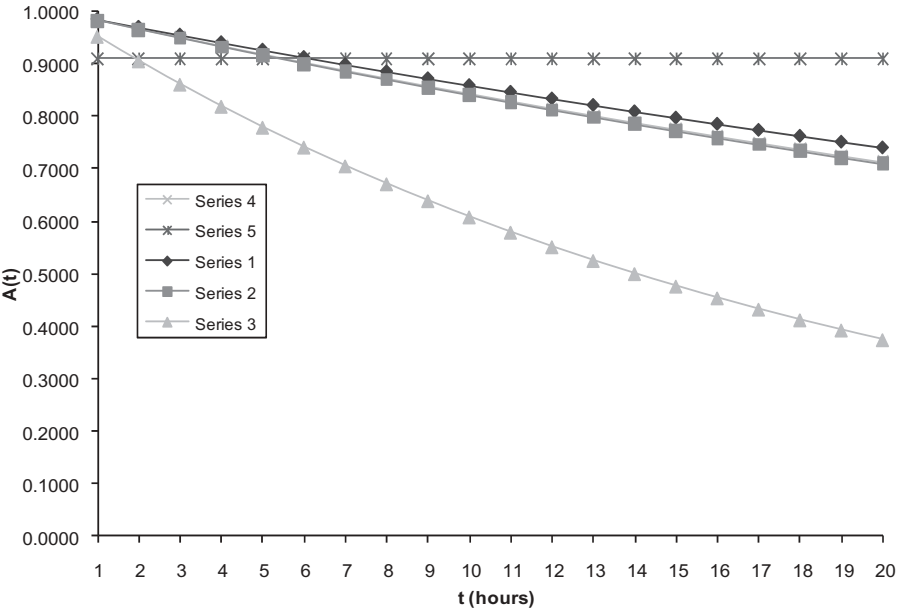


Figure 8.12 Original availability A(t) versus operating time t. Series 1: Subnetwork 2: fails requirement for t > 6. Series 2: Subnetwork 3: fails requirement for t > 5. Series 3: Network: fails requirement for t > 2. Series 4: Subnetwork 1: fails requirement for t > 5. Series 5: Specified availability = 0.9100.

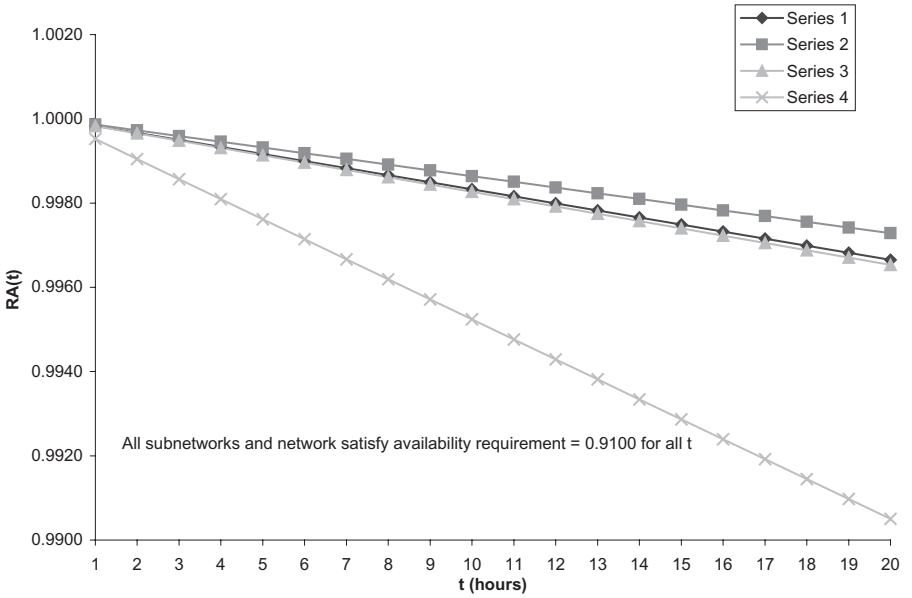


Figure 8.13 Revised availability RA (t) versus operating time t. Series 1: Subnetwork 1. Series 2: Subnetwork 2. Series 3: Subnetwork 3. Series 4: Network.

ANOTHER PERSPECTIVE ON PROBABILITY OF FAILURE

In this section a simplistic equation is developed for the probability of failures in a subnetwork or network, using a binomial distribution that is a function of x, a specified number of nodes and links that could fail in a subnetwork or network. In Equation 8.64, the constant probability of a node or link failing is p = 1/n, where n is the number of nodes and links in a subnetwork or network. This formulation assumes that nodes and links fail independently and that the probability of failure p is constant. Thus, the probability of x failures, P(x), is expressed in Equation 8.68:

$$P(x) = \frac{n!}{(n-x)!}(p^x)((1-p)^{n-x}). \tag{8.64}$$

While, admittedly, this is a crude formulation, it is useful for obtaining a rough cut of the reliability of a subnetwork or network when individual node and link failure data are not available. Even absent these data, Equation 8.64 provides the likelihood that x number of nodes and links is likely fail, and the values of x where P(x) will be a maximum. For example, the results in Figure 8.14 show that as the

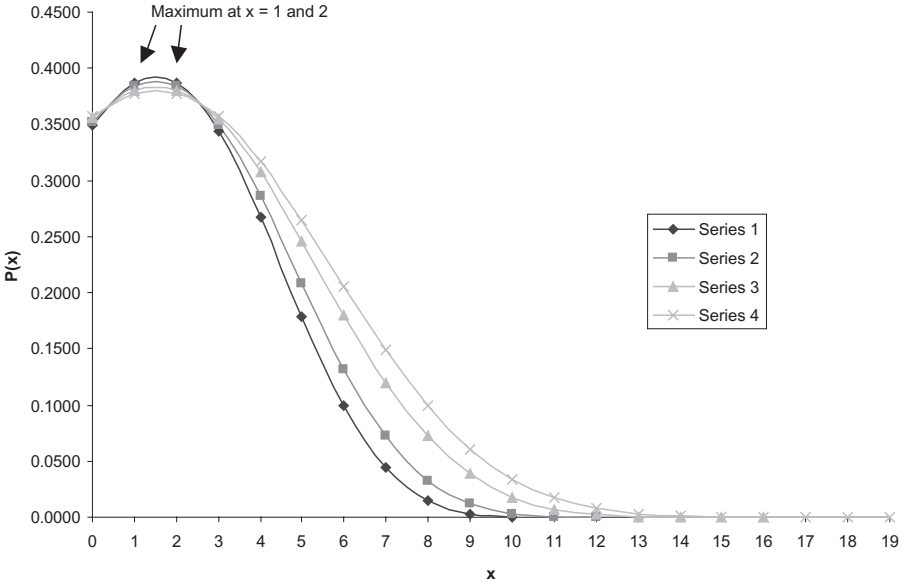


Figure 8.14 Probability of x failures, P(x), versus x. Series 1: Subnetwork 1. Series 2: Subnetworks 1 and 2. Series 3: Subnetworks 1, 2, and 3. Series 4: Network.

nodes and links are aggregated into subnetworks, and subnetworks are aggregated into the network, the maximum probability of failure always occurs at one and two failures. Thus, network users would only have to prepare for a small number of failures.

Problem for Reader: Why are the probabilities of failure in Figures 8.6 and 8.14 so significantly different?

Answer: Notice that in Figure 8.6, for the Poisson distribution, the probability of failure is a function of operating time while in Figure 8.14, for the binomial distribution, the probability of failure is a function of number of failures—specified number of link and node failures. In the case of Figure 8.6, the probability of failure is large because the network is exposed to long operating times—up to 20 hours. Furthermore, the probability of failure is also driven by the failure rate postulated in Equation 8.4, whose source is an Internet router company that reported a variety of network hardware and software failures. During this prolonged exposure, there are opportunities for faults to wreck havoc on the system. Contrast this situation with Figure 8.14, where the probability of failure is much smaller because the probability pertains to a link or node failing—an occurrence rare compared to *any* type of failure in Figure 8.6.

MEASURING PREDICTION ACCURACY

Of course, it is important to measure the accuracy of predictions so that you can see whether the models are validated. A frequently used measure is the relative error (RE) of reliability predictions and the MRE [FEN97]. To illustrate the process, first estimate the actual reliability, AR(t), for a subnetwork, in Equation 8.65, by summing the *original* failure counts F(t) over the number of nodes and links in the network, N, in the numerator, and then summing these counts over the number of operating time periods, n, in the denominator. Then compute the RE in Equation 8.66, using the *original* predicted reliabilities R(t) and actual reliabilities AR(t). Next, compute the MRE of the RE. It is also important to assess prediction accuracy after failure and fault correction, using remaining failures r(t) in Equation 8.67, as the actual remaining failure count, and then compute the *revised* RE RRE(t) in Equation 8.68, using the *revised* reliability predictions RR(t). Finally, compute the MRE of the revised RE:

$$\mathrm{AR}(t) = 1 - \frac{\sum_{i=1}^{N} \mathrm{F}(t)}{\sum_{i=1}^{n}\sum_{i=1}^{N} \mathrm{F}(t)}, \tag{8.65}$$

$$\mathrm{RE}(t) = (\mathrm{AR}(t) - \mathrm{R}(t))/\mathrm{AR}(t), \tag{8.66}$$

$$\mathrm{ARR}(t) = 1 - \frac{\sum_{i=1}^{N} \mathrm{r}(t)}{\sum_{i=1}^{n}\sum_{i=1}^{N} \mathrm{r}(t)}, \tag{8.67}$$

$$\mathrm{RRE}(t) = (\mathrm{ARR}(t) - \mathrm{RR}(t))/\mathrm{ARR}(t). \tag{8.68}$$

METHODS FOR IMPROVING RELIABILITY

The network should preserve connectivity in the presence of failures (i.e., fault tolerance in router subnetworks) [MEN]. One way to implement fault tolerance is to provide redundant units that can replace failed units. This approach can extend the mean lifetime of fault-free operation [KAI95]. Rather than switch in a fault-free unit, a network can achieve equivalent fault tolerance by providing alternate paths for data in the event of a router or link failure. This approach quickly plays to the strength of routers: detecting network failures and routing data round them. Data are routed between any two subnetworks on the lowest cost or shortest time-path basis. Redirectors exist for different network protocols: sending, receiving, and processing routing updates. Redirectors calculate a forwarding table from the available routing information, including destination subnetwork interface on which data are bound for

the destination subnetwork. During normal router operation, the forwarding table indicates only the best path to a destination subnetwork. When a link or router fails, routers exchange routing information to learn alternate paths. The period of time for the routers to detect the link failure and discover new routes to all available subnetworks is referred to as convergence time. Generally, convergence occurs within 1 minute [HEW93]. An example of this principal is shown in Figure 8.2, for subnetwork 1, where alternate paths would be provided by the Internet service provider between the cable modem and Internet router.

When an alternate path is provided on the link b, f in Figure 8.2, assuming equal reliabilities on the single and alternate paths, the alternate path *original* link reliability $APR(t_{ij})$ is the parallel reliability shown in Equation 8.69, where $R(t_{ij})$ is the *original* single-path reliability [LYU96]. Thus, use Equation 8.69 to see whether significant improvement in reliability is obtained for subnetwork 1 by using an alternate path. Then, the subnetwork *original* reliability, comprised of the reliability on link b, f, from Equation 8.69, and the reliabilities of the primary ($R(t_k)$) and nonprimary ($R(t_i)$) nodes, are computed in Equation 8.70:

$$APR(t_{ij}) = 2\,R(t_{ij}) - R(t_{ij})^2, \tag{8.69}$$

$$APR_s(t) = \prod_{ij,k,i} (APR(t_{ij}))(R(t_k))(R(t_i)). \tag{8.70}$$

In addition to the original reliabilities in Equations 8.69 and 8.70, the *revised* reliabilities resulting from failure and fault correction are computed in Equations 8.71 and 8.72 for the alternate path provided by link b, f and the subnetwork, respectively.

$$APRR(t_{ij}) = 2\,RR\,(t_{ij}) - RR\,(t_{ij})^2, \tag{8.71}$$

$$APR_s(t) = \prod_{ij,k,i} (APRR(t_{ij}))(RR(t_k))(RR(t_i)). \tag{8.72}$$

The first test of reliability improvement is shown in Figure 8.15 where, for *subnetwork 1*, the revised alternate path and single path subnetwork reliabilities satisfy the requirement for all values of operating time. The second test is shown in Figure 8.16, where original and revised alternate path *link* and single-path *link* reliabilities satisfy the reliability specification for all values of operating time, with the alternate path configurations providing the higher reliability. The expense incurred by using an alternate path would be justified for a mission-critical application but, perhaps, not for a commercial application.

Network reliability can also be improved by dividing the network into subsets that have high interaction and connectivity *within* a subnetwork. Subnetworks are then interconnected, thus providing isolation of network domains that are likely to experience high failure rates due to high interaction. This concept is shown in Figures 8.2–8.4 for subnetworks 1, 2, and 3, respectively, where the subnetworks have been created from the total network in Figure 8.1.

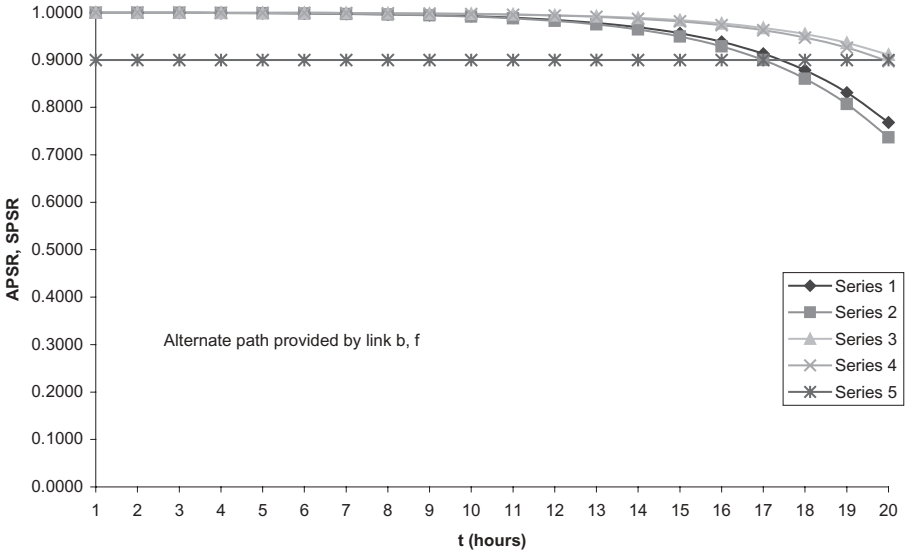


Figure 8.15 Alternate path subnetwork 1 reliability (APSR) and single path subnetwork reliability SPSR versus operating time t. Series 1: Original APSR, fails specification for t > 17. Series 2: Original SPSR, fails specification for t > 17. Series 3: Revised APSR, satisfies specification for all values of t. Series 4: Revised SPSR, satisfies specification for all values of t. Series 5: Specified reliability = 0.9000.

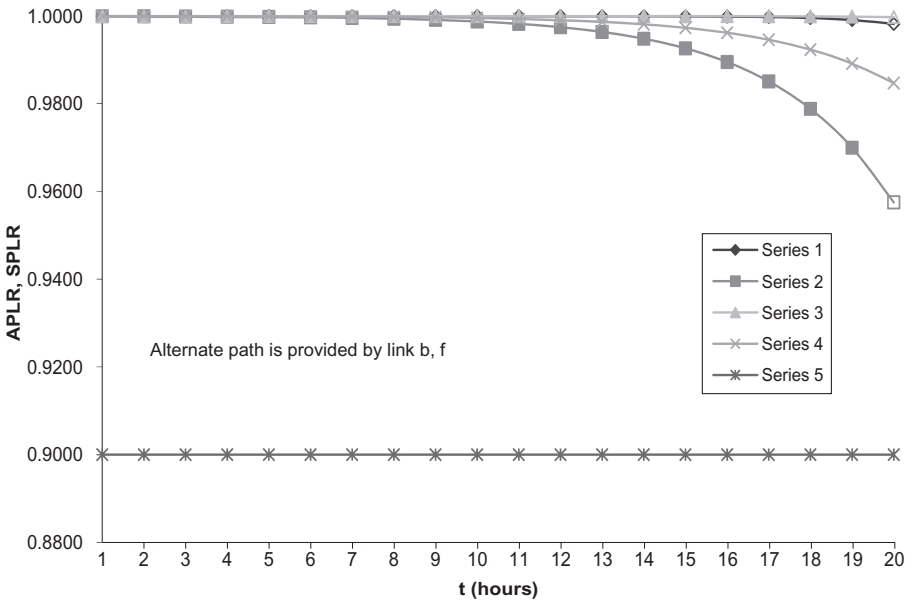


Figure 8.16 Alternate path link reliability (APLR) and single path link reliability (SPLR) versus operating time t. Series 1: Original APLR, specification satisfied for all values of t. Series 2: Original SPLR, specification satisfied for all values of t. Series 3: Revised APLR, specification satisfied for all values of t. Series 4: Revised SPLR, specification satisfied for all values of t. Series 5: Reliability requirement = 0.9000.

Table 8.2 Subnetwork Metrics and Results

		Subnetwork						
		1		2		3		
Metric	Purpose	Original	Revised	Original	Revised	Original	Revised	Action
Original probability of failure (Fig. 8.5)	Determine priority for failure correction	Highest priority		Lowest priority		Middle priority		Apply priority code to failure correction
Original and revised probability of failure (Fig. 8.6)	Compare original and revised probability of failure							Change in favorable direction for $t > 17$
Change in probability of failure (Fig. 8.7)	Compare changes in probability of failure		Change in favorable direction for $t > 16$		Change in favorable direction for $t > 15$		Change in favorable direction for $t > 16$	Correct additional failures
Failure and fault correction time (Fig. 8.8)	Detect anomalies in failure and fault correction time		Considerable variation in the correction times for the primary node and nonprimary node, and connecting link		Considerable variation in the correction times for the primary node and nonprimary node, and connecting link		Considerable variation in the correction times for the primary node and nonprimary node, and connecting link	Focus test effort on primary node and connecting link

(*Continued*)

Table 8.2 (*Continued*)

		Subnetwork						
		1		2		3		
Metric	Purpose	Original	Revised	Original	Revised	Original	Revised	Action
Remaining failures (Fig. 8.9)	Identify residual problems in network		Moderate effect of failure and fault correction		Greatest effect of failure and fault correction	Moderate effect of failure and fault correction		Focus test effort on subnetworks 1 and 3
Original actual and predicted reliability (Fig. 8.10)	Identify operating times when subnetworks fails specification	Fails specification for $t > 17$		Fails specification for $t > 17$		Fails specification for $t > 17$		Predict revised reliability
Revised actual and predicted reliability (Fig. 8.11)	Identify subnetwork(s) that fail specification		Satisfies specification for all t		Satisfies specification for all t		Fails specification for $t > 17$	Reboot subnetwork 3 for $t > 17$
Original predicted availability (Fig. 8.12)	Identify operating times when subnetworks fail specification	Fails requirement for $t > 5$		Fails requirement for $t > 6$		Fails requirement for $t > 5$		Determine whether revised availabilities will meet specification

Revised predicted availability (Fig. 8.13)	Identify subnetworks that satisfy availability requirement		Subnetwork satisfies availability requirement for all t		Subnetwork satisfies availability requirement for all t		Subnetwork satisfies availability requirement for all t	None
Probability of x failures (Fig. 8.14)	Determine distribution of failures in subnetworks	Maximum probability for x = 1 and 2		Maximum probability for x = 1 and 2		Maximum probability for x = 1 and 2		Focus test effort on one and two failure occurrences
Alternate path subnetwork reliability (Fig. 8.15)	Determine whether it is advantageous to use alternate path subnetwork	Fails specification for t > 17 on subnetwork alternate and single paths	Satisfies specification for all values of t on subnetwork alternate and single paths					None
Alternate path link reliability (Fig. 8.16)	Determine whether it is advantageous to use alternate path link	Specification satisfied for all values of t on link for alternate and single paths	Specification satisfied for all values of t on link for alternate and single paths					None

Table 8.3 Network Metrics and Results

Metric	Purpose	Original	Revised	Action
Original and revised probability of failure (Fig. 8.6)	Determine when the effect of correcting failures occurs	Probability of failure in favorable direction for $t < 17$	Probability of failure in favorable direction for $t > 17$	Improve test process for earlier operating time
Probability of failure change (Fig. 8.7)	Compare changes in probability of failure	Change in favorable direction for $t > 17$	Change in favorable direction for $t > 17$	Correct additional failures
Remaining failures (Fig. 8.9)	Identify residual problems in network		Minimal effect of failure correction	Focus test effort on network
Original actual and predicted reliability (Fig. 8.10)	Identify operating times when network fails specification	Fails specification for $t > 16$		Predict revised reliability
Revised actual and predicted reliability (Fig. 8.11)	Determine whether network fails specification		Fails specification for $t > 17$	Reboot network for $t > 17$
Original predicted availability (Fig. 8.12)	Identify operating times when network fails specification	Fails requirement for $t > 2$		Determine whether revised availability will meet specification
Revised predicted availability (Fig. 8.13)	Determine whether network satisfies availability requirement		Satisfies availability requirement all t	None
Probability of x failures (Fig. 8.14)	Determine distribution of failures in network	Maximum probability for $x = 1$ and 2		Focus test effort on one and two failures

Another method for improving reliability is to store recovery data in a database. For example, when a problem is detected in a network, a recovery action is executed by the wireless router in Figure 8.1 that is associated with the malfunctioning component to guide the recovery procedure. A database in Figure 8.1 stores network state data (e.g., node and node communication history) that is used to recover lost data after a crash. Changes in the network are broadcast to dependent components through the database's publish mechanism in order to initiate recovery. When a failure is detected, the defect is repaired, and the system continues running with minimum disturbance to other processes [HER07].

SUMMARY OF RESULTS

A large number of metrics, analyses of metric results, and explanatory plots have been used in modeling network reliability and availability. Therefore, it is necessary to summarize the highlights in Tables 8.2 and 8.3 for the subnetworks and network, respectively. The most important part of the tables is the action taken in response to the metric results. The actions indicate what users can do to improve the reliability and availability of their networks.

SUMMARY

Based on the network configuration diagrams, mathematical formulations and corresponding plots, and analysis results summaries, a practical template has been demonstrated for modeling and analyzing the reliability and availability of networks—nodes, links, and subnetworks. The specific numerical results that were obtained were for illustrative purposes. However, the template, or road map, could be used for different network topologies, parameters (i.e. duty cycle), and variables (i.e., failure rate).

REFERENCES

[ATH05] E. Athanasopoulou, P. Thakker, and W. H. Sanders, "Evaluating the dependability of a LEO satellite network for scientific applications," *Second International Conference on the Quantitative Evaluation of Systems (QEST'05)*, 2005, pp. 95–104.

[BEN07] S. Benlarbi and D. Stortz, "Measuring software reliability in practice: an industry case study," *The 18th IEEE International Symposium on Software Reliability (ISSRE '07)*, 2007, pp. 9–16.

[CHI01] V. Chirivella, R. Alcover, and J. Duato, "Accurate reliability and availability models for direct interconnection networks," *International Conference on Parallel Processing (ICPP '01)*, 2001, p. 517.

[DIS01] D. Donovan, C. Dislis, R. Murphy, S. Unger, C. Kenneally, J. Young, and L. Sheehan, "Incorporating software reliability engineering into the test process for an extensive GUI-based network management system", *Proceedings in 12th International Symposium on Software Reliability Engineering*, Volume, Issue, November 27–30, 2001 pp. 44–53.

[EIS] I. Eisenberger and F. Maiocco, "A preliminary deep space station operational availability model," JPL Deep Space Network Progress Report 42–21.

[FEN97] F. Norman, *Fenton and Shari Lawrence Pfleeger, Software Metrics: A Rigorous & Practical Approach*, 2nd ed. PWS Publishing Company, 1997.

[HER07] J. N. Herder, H. Bos, B. Gras, P. Homburg, and A. S. Tanenbaum, "Failure resilience for device drivers," *37th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, Volume, Issue, June 25–28, 2007, pp. 41–50.

[HEW93] Hewlett-Packard, "Improving Network Availability," Update July 1993.

[KAI95] N. Gaitanis, P. Kostarakis, and A. Paschalis, "Totally self checking reconfigurable duplication system with separate internal fault indication," *Proceedings of the Fourth Asian Test Symposium*, November 23–24, pp. 316–321.

[LLO62] D. K. Lloyd and M. Lipow, *Reliability: Management, Methods, and Mathematics*. Englewood Cliffs, NJ: Prentice-Hall, Inc., 1962.

[LYU96] M. R. Lyu (ed.), *Handbook of Software Reliability Engineering*. Los Alamitos, CA: IEEE Computer Society Press; New York: McGraw-Hill Book Company, 1996.

[MED00] D. Medhi and D. Tipper, "Multi-layered network survivability—models, analysis, architecture, framework and implementation: an overview," *Proceedings of the DARPA Information Survivability Conference and Exposition (DISCEX'2000)*, Hilton Head, South Carolina, 2000.

[MEN] V. B. Mendiratta, "A Simple ATM Backbone Network Reliability Model," Bell Labs, Lucent Technologies, Naperville, Illinois, USA, e-mail: veena@lucent.com.

[MIL98] M. R. Wilson, Bell Laboratories, "The quantitative impact of survivable network architectures on service availability," IEEE Communications Magazine, 1998.

[MOO01] D. Moore, G. Voelker, and S. Savage, "Inferring Internet denial of service activity," *Usenix Security Symposium*, 2001.

[MUS04] J. D. Musa, *Software Reliability Engineering: More Reliable Software, Faster, and Cheaper*, 2nd ed. Authorhouse, 2004.

[MUS87] J. D. Musa, A. Iannino, and K. Okumoto, *Software Reliability: Measurement, Prediction, Application*. New York: McGraw-Hill, 1987.

[NIC04] D. M. Nicol, W. H. Sanders, and K. S. Trivedi, "Model-based evaluation: from dependability to security," *IEEE Transactions on Dependable and Secure Computing*, 2004, 1(1).

[SHO02] L. Martin, *Shooman, Reliability of Computer Systems and Networks: Fault Tolerance, Analysis, and Design*. New York: John Wiley & Sons, Inc, 2002.

Part Three

Software Engineering

Chapter 9

Programming Languages

This chapter is designed to provide the reader with valuable information, analyses, and evaluations of programming languages. This is a vital topic because, after all, all the computer hardware and system design tools in the world will not produce an implement application without programming languages to support the implementation of software! An outstanding feature of this chapter are the models for estimating the reliability, maintainability, and availability of computer programs. This feature does not exist in other texts. In addition, the reader is led through various methodologies for designing programs, supported by graphical presentations that render the methodologies understandable.

INTRODUCTION

Programming languages are programmers' most basic tools. With appropriate programming languages one can drastically reduce the cost of building new applications as well as maintaining existing ones. There have been many advances in programming languages technology. The main driving force was and will be to better express programmers' ideas. Therefore, research in programming languages is an endless activity and the core of computer science. New language features, new programming paradigms, and better compile-time and run-time mechanisms can be foreseen in the future [COM09]. This chapter will discuss programming language issues and show the reader how languages can be evaluated and improved.

DESIRABLE PROPERTIES OF A PROGRAMMING LANGUAGE

A convenient way to think about desirable properties in a programming language is to think about how the brain solves a problem. First, there should be a minimum of syntax that has little relevance for how humans solve problems. The reason for this

Computer, Network, Software, and Hardware Engineering with Applications, First Edition. Norman F. Schneidewind.
© 2012 the Institute of Electrical and Electronics Engineers, Inc. Published 2012 by John Wiley & Sons, Inc.

criterion is that such excess baggage is a distraction to problem solving. The C++ language will be used as the principle language for illustrating characteristics that are beneficial for problem solving and those characteristics that are detrimental to this cause. I will couple the application of C++ to solving a problem in searching a Web site for desired information with the principles of the design process.

What is Design?

Let us consider the question: what is design? Design is the process of making decisions about an abstract representation of a system. The word "abstract" is used because at this stage the system does not exist. It is a concept in the brain that is later translated to a more concrete representation, such as a drawing, model, or mathematical equation. Design involves making trade-offs among various design alternatives [REI99]. For example, one alternative for searching a Web site is to serially—in a brute-force fashion—search for the desired content. A second alternative is to use pointers that have been organized to map to various subject matters. Yet a third alternative is to combine the second alternative with organizing the subject matter in chronological sequence, on the presumption that the user is interested in seeing the latest content first. Note that C++ is irrelevant to the evaluation of these *design* alternatives! This is an important point that you should be aware of: too many books confuse design issues with programming language characteristics. The correct process is to first select the most appropriate design, independent of programming languages. Then choose a programming language whose characteristics are most representative of the selected design. Another design principle according to some authors is avoiding details in the initial design process [REI99]. While it is true that it is unwise to become mired in details at an early stage in design, thus losing sight of the big picture, it is equally detrimental to the design process if important details are considered too late in the process. For example, if the desirability of presenting Web content in chronological order were put off until the design is almost complete, it would be very difficult to include this important feature when the design is almost complete.

System Decomposition into Components

A very worthwhile design principle is to decompose a system into its constituent components [REI99]. Doing so allows the designer to not become overwhelmed by the complexity of the system, thus leading to errors in design. For example, the Web search problem would be decomposed into *search request interpretation*, *search mechanization in the Web database*, and *Web page pointer management*. Along with decomposition, an important issue is the number of components and their hierarchy [REI99]. Again, using the Web search example, three components and a hierarchy according to the above sequence, seem appropriate.

Form of Design Presentation

Another consideration is the form in which the design should be presented [REI99]. This is particularly the case with respect to the purchaser and user of the system. The purchaser of a Web system server may be primarily interested in cost and search time, while the user would want to know the details of formatting a request and browser specifications, and so on.

Functional-Oriented versus Data-Oriented Design

There are two major design approaches: functional oriented and data oriented [REI99]. The emphasis in the former is the functions that must be executed to solve a problem; in the latter, the focus is on the data that must be processed to obtain a solution. For example, in the Web search application, functional-oriented design would involve identifying *search request interpretation*, *search mechanization in the Web database*, and *Web page pointer management* as procedures that must be executed to locate the user's desired Web page. In contrast, in the data-oriented design approach, the search request data, the data in the Web database, and the data related to pointers would be the focus of Web search processing. It is important to note that each approach would arrive at the same result, but with different performances, depending on the relative performance of computing (procedure-oriented) and communication (data-oriented) resources at the Web site. These design alternatives are shown in Figure 9.1.

Object-Oriented Design

Another design methodology, one touted by its advocates of solving the entire world's problems, is object-oriented design. This approach is based on the premise that systems are comprised of entities called objects that possess state, data that identify the object, and can perform actions, accompanied by state transitions. An example of a state is a Web server that is in the state of searching for a Web page. An example of identifying data is the manufacturer of the Web server. An example of an object action is a Web server object performing the action of delivering a Web page to the requester when the server is in the state of having located the desired page, as shown in Figure 9.1.

In this design methodology, objects are members of classes. Classes are entities that are the parents of objects. Classes have the same data attributes as classes and perform the same actions, but at a higher level. For example, a *generic* Web server could be a class, and *specific* Web servers manufactured by companies A, B, and C would be object members of the class.

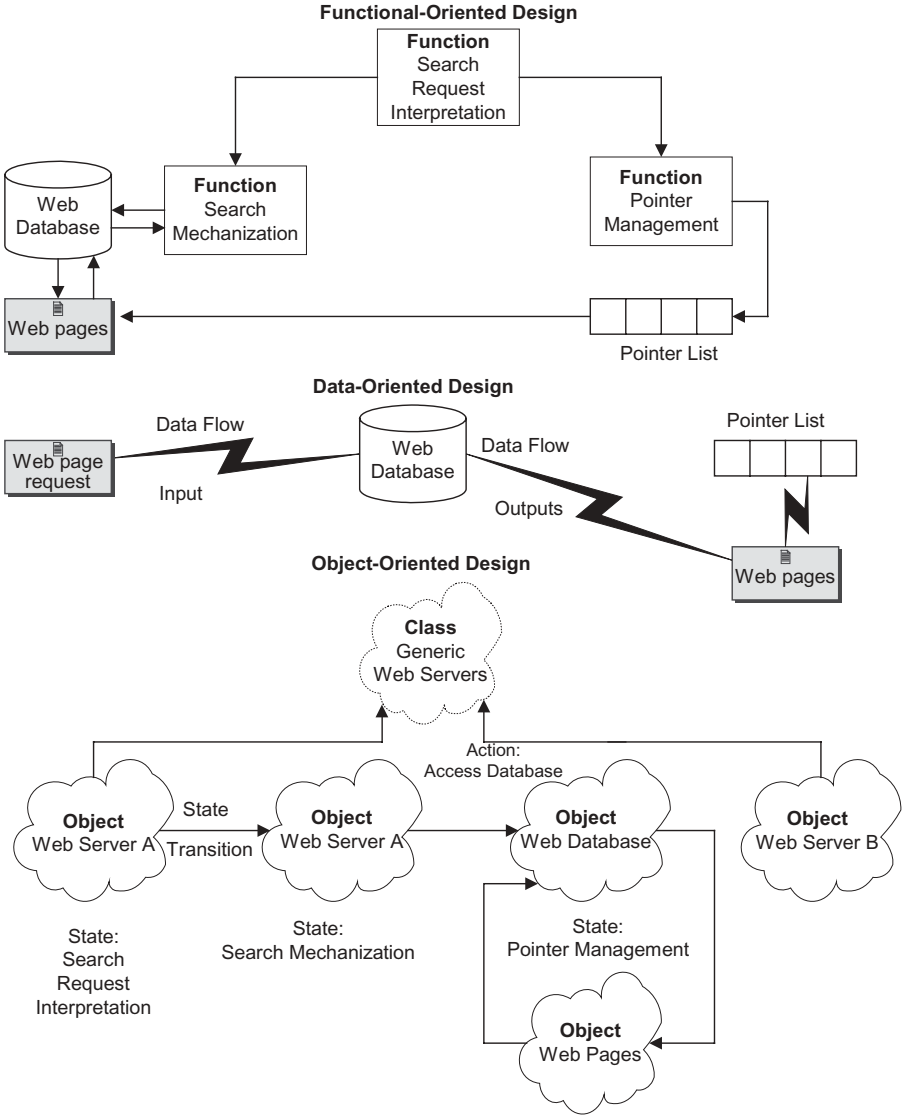


Figure 9.1 Design alternatives.

Analysis of the Design Alternatives

It is tempting to examine Figure 9.1 and think that one alternative is superior to the others, when in fact each can be used to advantage in a coordinated design approach. The functional approach aids the engineer and programmer in identifying calling sequences that can be used in C++, for example. That is, the functions identify the second-level program functions (search mechanization and pointer mana-

gement) that will be called by the top-level program function (search request interpretation).

Then, examining the data-oriented approach, this supports our use of data resources in implementing the design in C++. Code will be needed for implementing data flow between components in Figure 9.1. Note that the functional- and data-oriented approaches are complementary because data flow must be mechanized in order to accomplish the functional requirements. An important advantage of the data flow approach, compared with the other two methods, is that inputs and outputs are specified [REI99]. This is crucial because computer programs involve more than computation: they require input data to perform computations and the computation must produce output data.

Taking the analysis to the object-oriented level, this approach identifies state transitions. State transitions are an important way for organizing a computer program. For example, the Web server can be programmed to sequence through the state transitions in Figure 9.1. Additionally, because our computer program may have to handle multiple Web servers, the relationship between classes and objects is helpful because the computer code that implements the Web server generic class can be reused by multiple Web server objects by only changing the object name. Reusing software is important because the effort and time of program development are reduced and programming errors are reduced!

All design methods should provide for placeholders in order to implement interfaces between subsystems of a system, for example, between user Web request subsystem and local network subsystem[REI99]. This is a common technique in computer programming for reserving space in a program for code that will be determined at a future time. An example in C++ is to name an interface function, but leave the details for a future time when interface requirements have been determined.

Problem Representativeness in Programming Languages

One of the most intriguing aspects of this process is the fact that information is lost in transitioning from brain thought to a model of the system. For example, if it is desired to add two quantities, this operation is easily understood in our brain. For example, we “know” that the quantities are integer. We also know the length and precision of the quantities. Unfortunately, C++ and other languages do not know any of this to begin with and must be told every bit of minutia! Aggravating this problem is the fact that each compiler has its own syntax rules that do not always follow the C++ standard.

A METHOD FOR ANALYZING COMPUTER PROGRAM RELIABILITY

The reliability of a computer program is tremendously important but unfortunately is often overlooked in programming textbooks. In this chapter you will be introduced

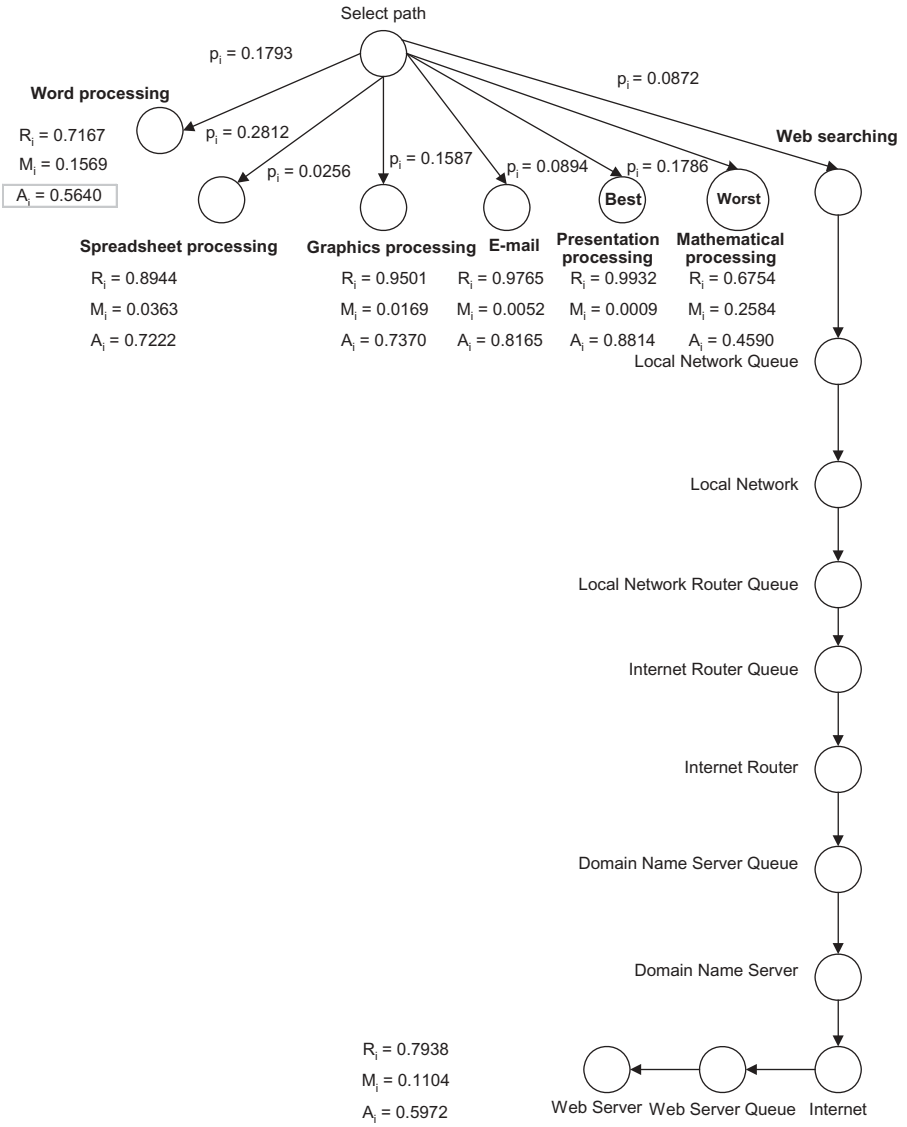


Figure 9.2 Program-directed graphs. R_i, path i reliability; M_i, path i maintainability; A_i, path i availability; p_i, path i probability.

to analyzing reliability as a function of the structure and complexity of the computer code. A program such as the Web server application can be conceptualized as a directed graph of logic in Figure 9.2, whose structure and complexity can be analyzed, leading to the estimation of program reliability. Interestingly, the directed graph of program logic does not correspond to any of the design methodologies already discussed despite the fact that a primary aspect of most problems is decision

making. None of the three design approaches already discussed represent the decisions made in a computer program. In contrast, the directed graph does an excellent job in this regard because decisions, represented by nodes in a graph, are concerned with the probability of deciding which logic paths to execute in a computer program, where a path represents a sequence of instructions (edges in a graph) to be executed to achieve a requirement (e.g., transmit a Web page request to a Web server). Once a decision is made, the execution of the selected path may encounter one or more faults, leading to one or more failures. Failure occurrence, in turn, provides the data for estimating reliability. Note that this design approach, like the data-oriented method, specifies input data (input to the first node on a path) and output data (output from the last node on a path).

Path probabilities and, hence, selected paths are dependent on input data. For the purpose of path probability assessment, input data is characterized by the probability of selecting path i, p_i, based on the frequency, f_i, of input type i:

$$p_i = \frac{f_i}{\sum_{i=1}^{n} f_i},$$

where n is the number of input types in the program.

For a *new* program, although the number of input data types n is known, the frequency of the types f_i is unknown; this factor would become known only after the program has been executed for a considerable period of time, but to estimate reliability, it is required that p_i be computed *now*. Therefore, f_i is generated from random numbers using our old friend Excel RAND function.

As stated earlier, paths are comprised of sequences of instructions. For programs that we write, we can estimate the number of instructions required. However, these days, the majority of software is that provided by software vendors where we have no idea of the sizes. Therefore, we can resort to using the total estimated number of instructions over all programs, N. How do we know the value of N? We do not know its value, but this need not concern us because whatever value we choose will lead us to the estimation of *relative* values of path reliability. Our interest is in estimating path reliability on a relative basis so that maintenance actions and, hence, availability can be estimated accordingly.

The estimated *expected* number of instructions executed on path i, N_i, is estimated as follows, where N is specified as 1000:

$$N_i = p_i * N.$$

Once the number of instructions on a path has been estimated, the number of instructions that are expected to fail, N_f, is estimated as follows:

$$N_f = (N_i) * (r_i/N),$$

where r_i is the estimated failure rate of path i, for N number of instructions, using the RAND random number generator. However, what is needed is the failure rate per instruction, r_i/N.

Then, using the above formulations, path i unreliability, UR_i, is estimated as the ratio of the number of failed instructions, Nf, on a path to the total number of failed instructions over all paths in the program:

$$UR_i = \frac{N_f}{\sum_{i=1}^{n} N_f}.$$

Then the R_i, reliability of path i, is estimated as follows:

$$R_i = 1 - UR_i.$$

The resultant unreliabilities and reliabilities are annotated on the directed graph of path logic in Figure 9.2. As you can see in this *particular* example that depends upon the roll of the dice in random number generation, the path reliabilities are low, suggesting that major maintenance actions would be required on a *relative* basis. The term *relative* is used because, as mentioned earlier, it is the *relative* reliability that is significant for signaling the for-maintenance action, once the software is implemented.

MODELING PATH MAINTAINABILITY AND AVAILABILITY

It is reasonable to suggest that path maintainability is proportional to path unreliability on the basis that the greater the unreliability, the greater the need for maintenance action. Also, we can see that the frequency of maintenance actions, m_i, is an additional determinate of the probability of maintenance action (i.e., maintainability of path i, M_i). Thus, combining these two factors, we have:

$$M_i = m_i * Ur_i.$$

Now, how is m_i determined? Well, for one thing, it should have the same ordering as unreliabilities. For example, the highest value of m_i should be associated with the highest value of UR_i because, naturally, the higher the unreliability, the greater the need for maintenance. Second, we do not know in advance of software implementation the frequency of maintenance activity. Therefore, m_i will be estimated by generating uniformly distributed random numbers between 0 and 1 and associate them with unreliability on an ordered basis. The resultant values of M_i are annotated on Figure 9.2 to provide visibility of the probable need for maintenance, by path, on a relative basis.

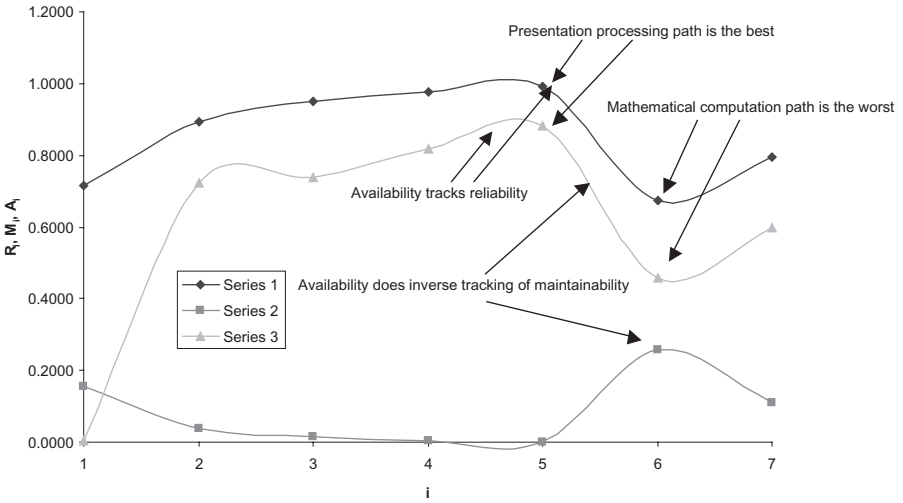


Figure 9.3 Path i reliability R_i (Series 1) maintainability M_i (Series 2), and availability A_i (Series 3) versus path i.

Now that both reliability and maintainability have been estimated for each path, availability, A_i, of path i is estimated as follows:

$$A_i = R_i/(R_i + M_i),$$

which expresses the fraction of software execution time during which there is a reliable operation. Availability results, by path, are keyed to paths in Figure 9.2.

Because the Web Processing function is the focus of our analysis, the details of its path are shown in Figure 9.2, whereas the other functions do not have expanded paths.

Another view of the results is provided by Figure 9.3, where you can see that availability mirrors reliability because availability reflects good operational time, which is the time when the software operates reliably. In contrast, availability has an inverse relationship with maintainability because the operational time lost to the maintenance activity is a loss of the availability of the software. Furthermore, Figure 9.3 is useful because it identifies the most reliable and least reliable paths. This information can be used to prioritize testing, allocating the greatest effort to the least reliable paths.

EXECUTING TEST SCENARIOS

It is necessary to test paths and combinations of paths that are called scenarios [REI99]. Scenarios represent sequences of path executions, where the scenario input has been defined, the corresponding computation specified, and the resulting output defined. Thus, scenarios are the mechanism for validating a computer program (i.e., demonstrating that a program does what it is supposed to do [BAG97]).

IMPLEMENTING COHESION AND COUPLING

Cohesion means that, for example, paths should contain only the code that is relevant to the path (e.g., Web searching should not contain word processing code). Coupling refers to the maximization of the independence of paths. That is, there should be only the minimum amount of interaction among paths. Following this principle reduces faults and failures when software is updated in the future. Of course some interaction is necessary; for example, when a document being created with a word processor requires data from a Web site. However, if access to the Web site were to be implemented by code in the document, as opposed to using a browser, problems would arise in maintaining the document if the Internet location of the Web site should change.

DETAILED ANALYSIS OF A PROGRAMMING LANGUAGE

It is difficult for the practitioner to know what design approach and programming language is best for his or her application because there are many advocates for a particular approach and language to the exclusion of other alternatives. The software field is plagued by faddism, where design models are proclaimed to be the only way to implement software, only to be discarded when the next fad arrives! What is needed is a balanced approach because there are properties of various design alternatives that can be combined to support the software implementation of a given problem. Therefore, this section is dedicated to providing the reader with a practical road map for developing software solutions. I begin by describing the entities that should comprise a design with the rationale given for each entity.

Program Objective

This is the most important part of the design approach [PRA02]: a clear and succinct statement of the program's requirement; for example, to retrieve publications of "Schneidewind software reliability model" in 5 seconds.

Objects

This entity is the basis of a design because all problems involve objects, whether it is a human user, database, mathematical equation, Web server, Web page, and so on.

Classes

A class is a set of objects, such as all Web servers. Some authors advocate making class the focus of design [REI99]. This seems strange because it is objects that are the active entities in a problem; classes, as the name implies, are classifications of

objects. Thus, I assign a class to the role of classifying objects. This is useful because the same Web searching code can be used, for example, for a variety of Web servers, by classifying each specific Web server object as a member of the generic Web server class.

Functions

Functions, for example mathematical functions, do not receive the attention they merit in contemporary programming language texts [PRA02]. In these books, functions are described as C and C++ functions, the modules of these languages.

Decisions

Decisions are the meat of many problems, for example determining whether a Web searching algorithm has found the requested information. Again, decisions are given the short shrift in many texts. Decisions can be represented by a directed graph or by an old-fashioned flow chart, heaven forbid!

Input Data

Objects and functions cannot operate in a vacuum. Input data, such as the user's specification of the desired information required from a Web site, must be specified to the search algorithm. The origin of input data is specified.

Output Data

Output data refers to the data that the output units in a computer system will provide. The destination of output data is specified.

Control

Program control is necessary because certain operations, such as iteration, must be terminated, for example, the termination of Web searching when the desired Web page has been found in Figure 9.4.

Units

These are physical computer system entities, such as a graphics display device. Generally, units are only specified in special-purpose computing, such as in space applications, where specific hardware is assigned to processing specific software. In general computer processing, such as Web searching, unit specifications are not

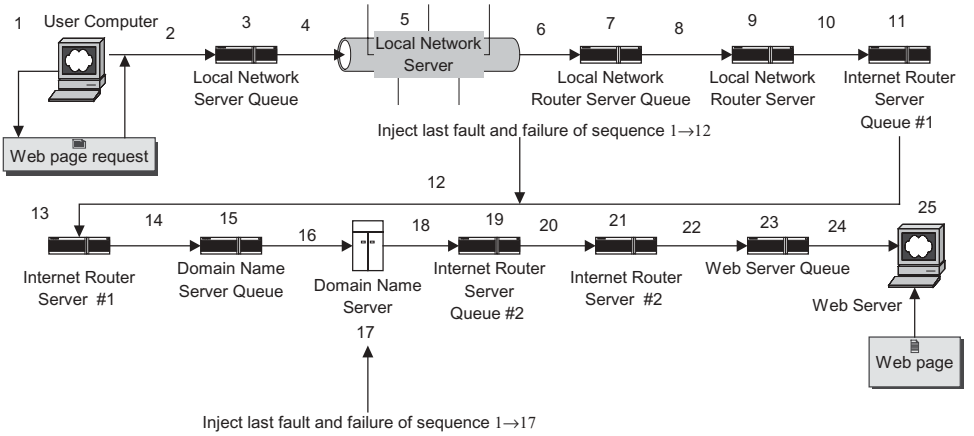


Figure 9.4 Reliability simulation process diagram.

necessary because the user is only concerned with results, not the particular hardware that produces the results.

Support Requirements

In addition to these specifications, it is important to keep in mind the need to maintain the software in the future based on errors that occur and on the need for enhancements that users require [PRA02]. Incidentally, the cost of software maintenance is the largest of all software costs, including the cost of development.

Now, the above entities of software design will be illustrated by producing Table 9.1, which will document each entity for the Web searching problem, and Figure 9.4, which shows the logic for Web searching. An important additional factor, which contributes to high quality software, is the need to produce good documentation. Document the software for other people to read, who may not be as familiar with the details as you are! Poor or nonexistent documentation has been the bane of the software field. Do not contribute to this chaos!

PROGRAM LANGUAGE CHARACTERISTICS

Structure

This section is dedicated to describing program language structure, using C++ as the example. My purpose is to not only describe, but, in addition, evaluate the efficacy of the structures so that you can apply the structures with minimum difficulty. First, what is meant by program structure? Structure is required to control the

Table 9.1 Program Objective: Perform Web Search for Desired Information

Object	Class	Function	Decision	Input Data	Output Data
User	People	Provide search request	Is returned information the desired information?	Brain activity: need information	Search request to Internet object
Web Server A	Web Server	Web search		Web page request from user	Desired Web page to user
Web page on Web Server A	Web page	Contains archived information			
Web Server A search algorithm	Search algorithm		Is this the desired Web page?	Search command from Web Server A	Desired Web page to Web Server A
Web Server A database	Web server database	Stores archived information		Search criteria from search algorithm	Retrieves desired Web page
Iteration control of Web site searching	C++ class of *while* instructions	Implements *while* program control	*Not* desired Web page?	Web page request and Web page	Found Not found

Table 9.2 Definition of Conditions

Operator	Meaning	Figure 9.4 example
=	Equal	Search information correct
!=	Not equal	Search information correct *not* correct
<	Less than	Web page *not* found
>	Greater than	Web page *not* found
<=	Less than or equal to	Web page found
>=	Greater than or equal to	Web page found

program statements to execute and the order of executing them [HAN97]. These structures are sequential, selection, and repetition, using iteration control. Sequential structures are sequences of statements that do not involve decision making. The best example is a sequence of arithmetic statements programmed to solve an equation. Selection involves designing statements to make decisions, such as the decision by the user in Figure 9.4 concerning whether the desired information has been obtained from the Web site. Decisions are typically implemented with the *if* statement. Repetition, controlled by iteration, is implemented with the *while* statement in Figure 9.4.

Conditions

Conditions refer to the outcome of comparing quantities. The meaning of the condition operators, geared to Figure 9.4, is defined in Table 9.2.

Logical Operators

There are some conditions that cannot be handled by the operators in Table 9.2. These are the Boolean operators *and (&&)*, *or (||)*, and *not(!)*. Using Figure 9.4 as an example, suppose the user desires information by subject S *and* that the date D of the information be greater than d. Then the condition can be written as follows; statements must end in a semicolon and "//" indicates a comment:

S && (d > D) = **true**; // if S and (d > D) are true, the result is true = 1, if the result is not true, it is false = 0, so that the result can be checked by comparing with constants 1 and 0.

As another example, suppose the subject is still S *and* either (date d1 = D1) *or* (date d2 = D2) is required. Then, these conditions are written as:

(S && ((date d1 = D1) || (d2 = D2))) = **true**; // notice the liberal use of parentheses that renders the code readable.

As the third example, suppose subject S is still desired but the date d = D is to be *excluded* in the search:

(S && (d ! = D)) = true // != signifies *not equal*.

Important Variable Types

There are various types of variables that can be used in programming languages. Among these are types *bool* (i.e., Boolean), *integer*, and *double* (i.e., floating point). The variable type *bool* is used to keep track of events in a program's execution [HAN97]. For example, in Figure 9.4, suppose there is a variable of type *bool* called *search information correct*. Then, when the *if* statement is executed, *search information correct = true*, if the correct information found, and *search information correct = false*, if the correct information *not* found. Variable types *integer* and *double* are used in arithmetic operations to signify operands that have no fractional part and operands that do have a fractional part respectively.

Detailed Design Example

Now, in Figure 9.5, a detailed software design is implemented for the Web search problem in Figure 9.4. This exercise will illustrate some very interesting design aspects that are not covered well in programming texts. One aspect is that for a design to be meaningful, it must not be limited to abstractions, such as classes! Instead, for a design to be understood, there must be a combination of physical entities and program abstractions. For example, in Figure 9.5 the designer should not limit the design to proclaiming an input specification. Instead, it is critical to identify the input device: keyboard, hard disk file, memory stick, and so on. The reason is that C++ and other languages are very finicky about such details because there are different types of commands pertinent to different devices. Thus, where decisions about physical components must be made, I ask questions in Figure 9.5 to stimulate decision making. Note that I do not pose questions concerning Internet resources, such as the brands of Internet browser, Web server, and search engine because these components are outside the scope of this *particular* C++ program. The browser and search engine are human user choices and the search engine vendor specifies the server(s) to use on a particular search. Also, notice the need for "housekeeping" declarations in Figure 9.5, such as specifying the search request type, where "type" refers to format, not the type of request. Additional C++ syntax is shown in Figure 9.5 in connection with the *if* statement. If the condition is false (e.g., the search has been unsuccessful), the program branch executed is called *else*.

Rather than attempt to write one program to cover all the logic in Figure 9.5, experience has shown that when the programmer includes too much logic in a single program, the programmer is overwhelmed by complexity and programming errors grow exponentially. Therefore, the total logic should be divided into digestible pieces [HON96]. Thus, three C++ programs: #1 for user specification of Web search requirement, #2 for user analysis of Web search results, and #3 for Web server search process are shown below. A major purpose of this presentation is to show the reader that there is a great deal of housekeeping that must accompany the meat of a program, if the program is to compile (translate from C++ statements to machine language that can be executed on a computer). It is imperative to understand that

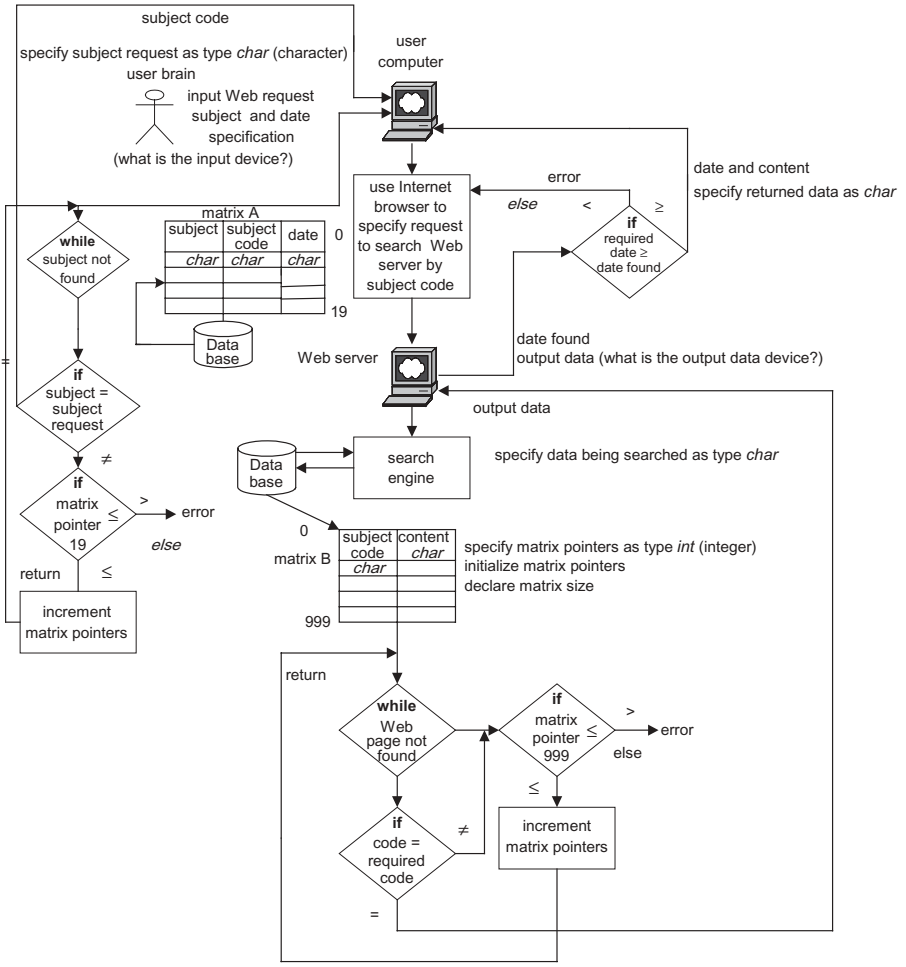


Figure 9.5 Web search detailed design.

both the logic of the program and the housekeeping directives must be correct for the program to work. This characteristic of software development is one of the reasons that software is so costly and error prone.

Note that by convention, the first index value for an array or matrix is equal to zero, hence the initialization of the matrix pointers in the following programs to zero. Also note that matrix sizes must be assumed. This means that if a search exhausts a matrix without finding the desired item, an error must be signaled. In addition, Figure 9.5 shows that for both the user and Web site, a database is used for search request and search content, respectively. The computer codes for these functions are not included in the following programs because this processing would be handled by separate software in database management systems. In addition, the

code for the interaction between Internet browser and Web server for specifying user-specified subject code, as shown in Figure 9.5, is included in the programs.

Program #1

```
// program specifying Web search request (good idea
to state purpose of program in first comment)

# include <iostream> // specify input output library

# include <math.h> // specify math library

using namespace std; // allows C++ to allocated
computer space for names

using std::cout; // specify standard screen output

using std::cin; // specify keyboard input

main() // beginning of main code, this is required in
every program

{ // opening bracket is needed at start of code

const char* format_string; // pointer to type char
for processing alphanumeric data

int i, j, imax // declare matrix A pointers and
maximum pointer value as type integer

char A [20] [20] [20]; // declare matrix A as three
columns and 20 rows and type char that contains
subject, subject code, and date

char subject; // declare search request subject

char code; declare located subject code storage
area

char date; declare storage area for search request
content date

i, j, k = 0 // initialize matrix pointers

imax = 19; // initialize maximum value of matrix A i
pointer

cout << endl; // start screen output on a new line

cout << "search request subject ="; // tell user to
input search request subject from keyboard to screen,
the = sign means that input is expected after it

cin >> subject ; // request subject inputted

cout << "input required date of search results"; //
input required date of search results from keyboard
to screen

cin >> date; // required date of search results
inputted
```

```
while (subject != A [i]) // while subject request not
found, continue with search
{
if (subject = A [i]) // test for finding request
subject
{
code = A [j]; // if request subject found, store
located subject code for use by user computer
date = A [k]; // store specified date of search
results for use by Web server
else
{
if ( i < = imax) // test for subject pointer being
less than or equal to maximum value
{
i = i + 1; // if subject pointer has not reached
maximum value, increment all matrix pointers
j = j + 1;
k = k + 1;
}
else
cout << “error: subject not found”; // tell user that
request subject not found
}
return 0; // return to the operating system
} // executable code ends here
```

Program #2

```
// program for analyzing Web search results
(good idea to state purpose of program in first
comment)
# include <iostream> // specify input output library
# include <math.h> // specify math library
using namespace std; // allows C++ to allocated
computer space for names
using std::cout; // specify standard screen output
main() // beginning of main code, this is required in
every program
{ // opening bracket is needed at start of code
```

```
FILE *fp;//pointer to type FILE, C++ requires
pointers to files that are read or written, files
usually stored on hard disk or stick memory, fp
points to the file command in the next statement

fp = fopen ("c:/search results.txt", "w"); // file
for writing search results data output in text
format, the location is given for this file to be
written, signified by "w"

const char* format_string; // pointer to type char

char date; // declare date of search results in
alphanumeric format

char requireddate; // declare required date of search
results in alphanumeric format

if (date > = requireddate)

{

fprintf (fp, "%s%\n", "search results date", (char)
20); // write quoted text in c:/search results.txt
disk file, "s" specifies writing quoted text, 20
characters long, "\n" specifies line feed for next
line to be written

fprintf (fp, "%c%\n", date, (char) 8); // date of
search result written in c:/search results.txt disk
file, "c" means date is in alphanumeric format, 8
characters long, "\n" specifies line feed for next
line to be written

fprintf (fp, "%s%\n", "search results content",
(char) 20); // write quoted text in c:/search
results.txt disk file, "s" specifies writing quoted
text, 20 characters long, "\n" specifies line feed
for next line to be written

fprintf (fp, "%c%\n", date, (char) 1000); // search
results content written in file c:/search results.txt
disk file, "c" means content is in alphanumeric
format, 1000 characters long, "\n" specifies line
feed for next line to be written

}

else

{

cout << "search results date incorrect"; // tell user
that search results date incorrect

}

return 0; // return to the operating system

} // executable code ends here
```

Program #3

```
// program for searching Web database (good idea to
state purpose of program in first comment)

# include <iostream> // specify input output library

# include <math.h> // specify math library

using namespace std; // allows C++ to allocated
computer space for names

main() // beginning of main code, this is required in
every program

{ // opening bracket is needed at start of code

const char* format_string; // pointer to type char
for processing alphanumeric data

int i, j, imax // declare matrix B pointers and
maximum value of subject code pointer as type
integer

char B [20] [20]; // declare matrix B as two columns
and 20 rows and type char

char subject; // declare search request subject code
that was inputted by Web browser

char content; declare located content storage area

i, j = 0 // initialize matrix pointers

imax = 999; // initialize maximum value of matrix B i
pointer

while

{(subject ! = B [i]) // continue search while subject
code not found

if (B [i] = subject) // test for finding subject code

{

content = B [j] ; // if subject code found, store
content in Web server

}

else

{

if ( i < = imax) // test for subject code pointer
being less than or equal to maximum value

{i = i + 1; // increment matrix B pointers, if
subject code not found

j = j + 1;

}
```

```
else
{
cout << "error: subject code not found"; // tell Web
server that request subject not found
}
return 0; // return to the operating system
} // executable code ends here
```

EVALUATION OF PROGRAMMING LANGUAGES

Factors to consider when evaluating a programming language are compile time, execution time, understandability of error messages, and availability and quality of help information. Some versions of C++, for example, produce unintelligible error message and their help information is minimal. These are important considerations that significantly affect your productivity. To avoid these pitfalls, download free copies of programming systems and test them against the above criteria for the same program. Then avoid using systems that produce more noise than signal!

One researcher investigated program length, programming effort, runtime efficiency, memory consumption, and reliability [PRE00]. However, the validity of the analysis was compromised by using several programmers in the tests, rather than one, thus introducing programmer skill variability into the mix. Nevertheless, there are some valuable aspects of this experiment that you should note. One is programming effort: is the effort you expend in understanding and using the language reasonable? Another good point is reliability: does the program produce the correct, predetermined result. Of course. You must be careful that a perceived incorrect result is not due to your programming errors! The other factors—program length, runtime efficiency, and memory consumption—are of little consequence, given the speed and memory capacity of contemporary microcomputers. It is surprising that the author did not evaluate compile time, execution time, understandability of error messages, and availability and quality of help information.

Visual Language Alternative

There have been numerous studies that have looked at the learning styles of engineering students. These learning style preferences are consistent across populations. What these studies have found is that engineering students tend to be more visual in their learning styles. However, since most programming languages taught in introductory courses are text based, a disconnect occurs between what is being taught and how these students prefer to learn [BUC09].

Because many text-based languages use syntax that incorporates many English terms, students often resort to using the models they have developed for the natural language use of these terms. However, this poses a significant problem for some terms because the model for how the word is used in natural language differs from

how it is used in a programming language. For example, in natural language, the term "while" has a slightly different meaning than it does in programming usage. In natural language, "while" implies that as soon as the condition is no longer satisfied, the activity will cease. In a programming language, the conditional statement associated with the "while" is only checked once during an iteration. This can cause students problems if they believe that as soon as the condition is met, the loop will exit [BUC09].

A promising avenue for the reader to explore to address these issues is the use of graphical programming languages. Graphical programming languages allow the user to create programs by connecting together graphical icons representing different functions, similar to flowcharts. Using these languages should help students learn better from visual presentations [BUC09]. For, example, graphical programming languages such as Simulink and Hypersignal, and others, have been coming into use recently for rapid prototyping of digital signal processing algorithms. Using such languages amounts to dragging functional blocks from libraries and connecting them to form a block diagram, which is also a program [AMI00].

Question for the Reader: Based on what you have learned about programming language characteristics, what characteristic do you think is the most important?

Answer: If a programming language is not *representative* of the problem to be solved, programs that are produced using this language could be loaded with bugs! For example, a program for doing numerical computation should have a library of mathematical software (e.g., sine function) that the programmer could invoke rather than having to program these functions, thus saving a significant amount of time and avoiding programming errors.

SUMMARY

The reader has been exposed to many aspects of evaluating programming languages that are not covered in contemporary texts. Among these aspects are lack of coherent compiler and execution error messages and help aids. Techniques have been presented for testing a set of programming languages against a specified program in order to identify the language that is best for the user. Armed with these tools, the reader will be able to combine previously learned computer hardware design skills and knowledge with programming languages to develop computer-based systems.

REFERENCES

[AMI00] N. Amir, "The role of graphical programming languages in teaching DSP," *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing*, Volume 6, 2000, p. 3514.

[BAG97] D. J. Bagert, "A software engineering-oriented concepts of programming languages course," *Proceedings of 27th Annual Conference on Frontiers in Education: Teaching and Learning in an Era of Change*, Volume 2, 1997, p. 699.

[BUC09] G. Bucks and W. Oakes, "Work in progress—using graphical programming languages in the introductory programming course," *39th IEEE Frontiers in Education Conference*, October, 2009, pp. 1–2.

[COM09] *International Multiconference on Computer Science and Information Technology*, October 12–14, 2009, p. 633.

[HAN97] J. R. HANLY, *Essential C++ for Engineers and Scientists*. Reading, MA: Addison-Wesley Longman, Inc., 1997.

[HON96] J. W. HONCHELL and T. L. ROBERTSON, "Is the role of applied programming languages changing," *Proceedings of 26th Annual Conference Frontiers in Education Conference*, Volume 2, 1996, pp. 791–794.

[PRA02] S. PRATA, *C. Primer Plus*, 4th ed., Indianapolis, IN: SAMS Publishing, 2002.

[PRE00] L. PRECHELT, "An empirical comparison of seven programming languages," *IEEE Computer*, 2000, 33(10), pp. 23–29.

[REI99] S. P. REISS, *A Practical Introduction to Software Design with C++*. New York: John Wiley and Sons, Inc., 1999.

Chapter 10

Operating Systems

The aim of this chapter is to expose the reader to important facets of operating system (OS) analysis and design that are missing from contemporary texts. Among these are quantitative analyses of reliability, performance attributes such as memory management efficiency, optimization of time-slice allocation to processes, and deadlock detection and prevention. To lay the foundation for these analyses, OS issues and OS architecture are discussed so that the reader will understand why there is a focus on certain facets of OS behavior, such as the difference in computing environments between general-purpose and real-time systems. The dramatically different OS performance differences between these environments are highlighted.

OPERATING SYSTEM ISSUES

Operating systems (OSs) have become increasingly complex and thus very expensive and time consuming to develop, maintain, and debug. Two difficulties are inherent in maintaining any large complex system: a large investment is required to add new features rapidly enough to meet time-to-market requirements and another is the effort required to preserve compatibility with prior versions. Continuing to patch existing OSs in ad hoc ways to accommodate tomorrow's needs (e.g., high reliability) is not cost-effective. Interactions within the OS and between the OS and application programs are very complex [HAM95].

The challenge to OSs designers is to deliver to applications the performance available now only from dedicated hardware, combined with the ease of sharing resources and data among multiple applications [AND92]. This issue is fascinating because the original objective of OSs was to efficiently manage multiple applications in a complex computing environment. Unfortunately, OSs have grown to the extent that their excess baggage can slow application execution to a crawl!

The importance of OSs has motivated the development of this chapter, which is designed to provide the reader with methodologies for analyzing and estimating

Computer, Network, Software, and Hardware Engineering with Applications, First Edition. Norman F. Schneidewind.
© 2012 the Institute of Electrical and Electronics Engineers, Inc. Published 2012 by John Wiley & Sons, Inc.

the performance and reliability of these systems, with the goal of mitigating the problems described above.

OS ARCHITECTURE

OS architecture is comprised of processes (i.e., a process is a program in execution), interprocess communication (e.g., instruction and data communication on a bus), virtual memory (a fancy name for disk drive memory as opposed to random access memory [RAM] devices), hierarchical file system, and access control to system resources [REI04]. Three approaches have evolved for organizing structure and are shown in Figure 10.1. The monolithic architecture considers the OS as being comprised of modules wherein any module can call another module. The modules are controlled by the supervisor so that they can have access to the hardware. All module calls must be made under supervisor control. This organization is obviously complex and leads to high overhead. In response to this problem the microkernel architecture was developed [REI04]. This architecture simplifies OS functions by centralizing functions in the microkernel. This strategy reduces the complexity of OS design because the numerous interactions that must transpire in the monolithic case can be centralized in the microkernel in the microkernel case. A similar situation happens in the client–server architecture, wherein simplicity is achieved by using a bus for communication among system resources. However, as will be shown in the reliability section. There is a price to pay for reducing architectural complexity because reducing interactions among resources can lead to dependence on a single resource (e.g., microkernel) for managing controlling OS actions. This dependency can lead to reduced reliability.

OS PERFORMANCE EVALUATION

An OS performance attribute is its ability to switch among various programs while also allocating resources to these programs. This switching function is executed by the supervisor and microkernel in Figure 10.1, which switch among OS modules, application programs, and computer hardware as the need for these resources arises during system operation. Switching speed and time are important performance metrics. Switching speed can be computed in two ways: one, by switch operation i, S_i, using the number of programs switched on switch operation i, n_i; and the second, computed over the number of programs n. In both cases, since we do not know a priori the probability of making a switch i, the probability, p_i, must be included in the equations. Thus, switching speed, S, and time for switching to program i, T_i, can be estimated by considering the number of programs, n, that must be switched for a given user's operation, as follows:

$$S_i = (p_i n_i)/Ti,$$

$$S = \sum_{i=1}^{n} ((p_i n_i)/T_i)/n.$$

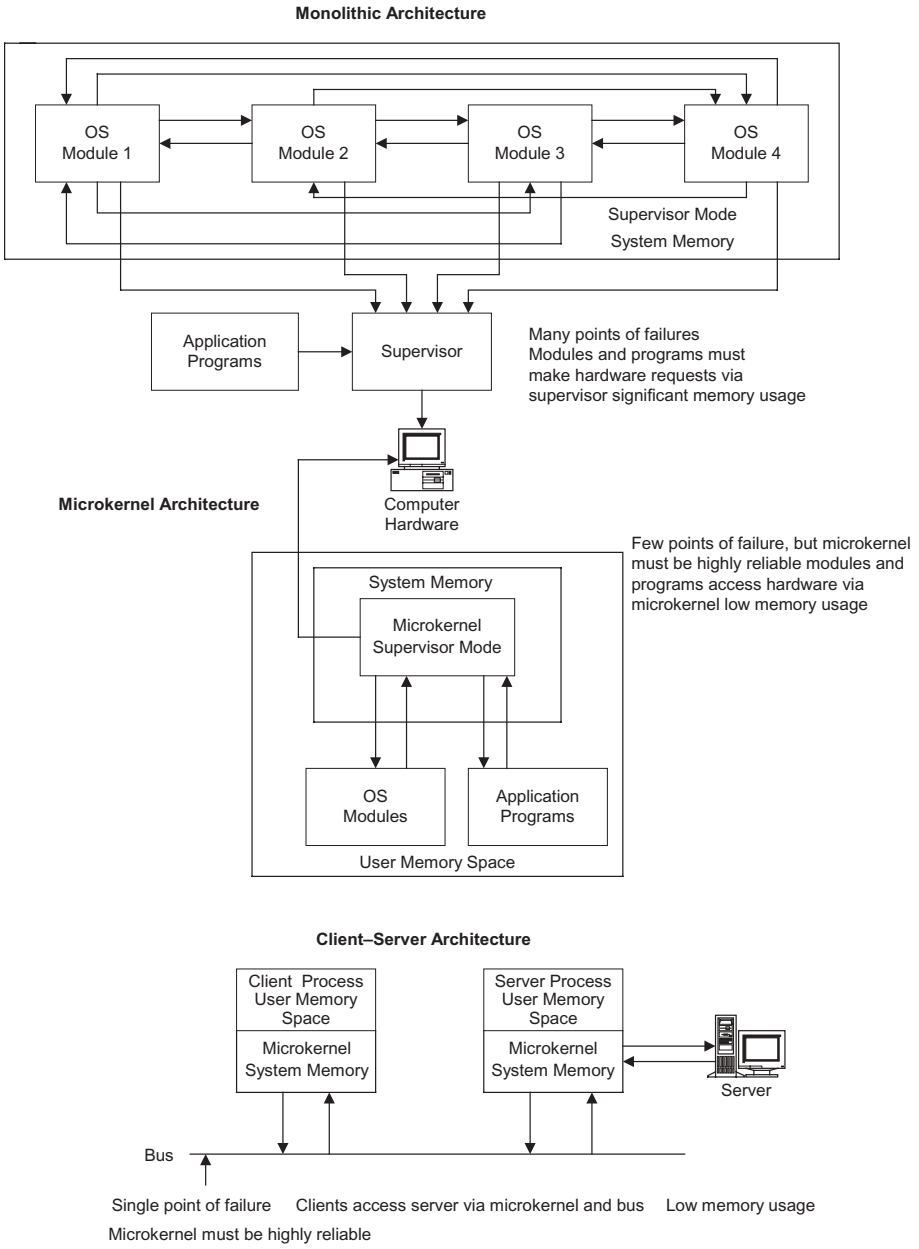


Figure 10.1 Operating system (OS) architectures.

Since n_i and T_i are not known in advance of using an OS, their values must be simulated by using the Excel RAND function (uniformly distributed random numbers between 0 and 1) for n = 100 switch operations in order to achieve computational validity. After executing the RAND function, both n_i and S_i are multiplied by 10 in order to produce practical values. Since the relative values of S_i will not be affected by the choice of multiplication factor, readers are free to choose whatever value is practical in their application.

Furthermore, since S is the mean of S_i, the standard error of the mean $S_i\sqrt{n}$ can be computed, which when combined with the mean $\left(S \pm 3 * S_i / \sqrt{n}\right)$ provides confidence intervals for S_i. If any values of S_i fall outside the mean plus or minus three $S_i\sqrt{n}$, it is indicative of switching spends that are unlikely to be achieved. Thus, the user can predict in advance of OS usage the bounds on this performance metric. Figure 10.2 shows how the bounds on switching speed can be analyzed.

OS RELIABILITY EVALUATION

Another important OS metric that we can relate to switching actions is reliability. Again, prior to using an OS to manage our computer operations, it is possible to estimate reliability by randomly injecting faults into the switching operation. Thus, the reliability, R_i, of switch operation i, is estimated by noting that the *expected* number of faults that occur on switch operation i is the product of the probability

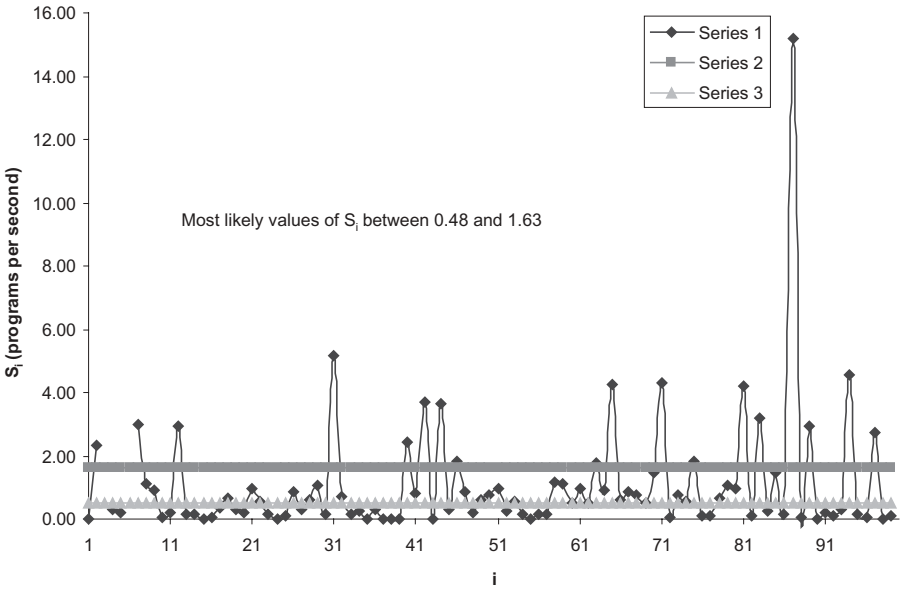


Figure 10.2 Switch operation i speed S_i versus switch operation i. Series 1: S_i. Series 2: Mean S_i + 3∗(standard error of mean) = 1.63. Series 3: Mean S_i – 3*(standard error of mean) = 0.48.

of the switch operation, p_i, and the number of faults f_i. Then, this term is related to the total number of *expected* faults over n switching operations to produce the unreliability of switch operation i. Finally, unreliability is subtracted from 1 to produce reliability. As in the case of performance, confidence intervals can be developed for reliability so that the likely achievable range of reliability can be estimated:

$$R_i = 1 - \frac{p_i f_i}{\sum_{i=1}^{n} p_i f_i}.$$

Figure 10.3 shows the probable range of switching reliability that is likely to be achieved in practice. In addition to the foregoing quantitative reliability assessment, it is important to evaluate reliability on a quantitative basis, based on the competing architectures shown in Figure 10.1. Although the monolithic architecture is considered inefficient [REI04], it can continue operation with reduced capability because communication between surviving OS modules and application programs can continue in the face of one or more failed OS modules. In contrast, the microkernel architecture, while compact and efficient, is highly dependent on its namesake for reliable communication because all traffic flow must be managed by the microkernel. A similar situation occurs with the client–server architecture because all communication must take place on the bus. Thus, the lesson to be learned is that the relationship

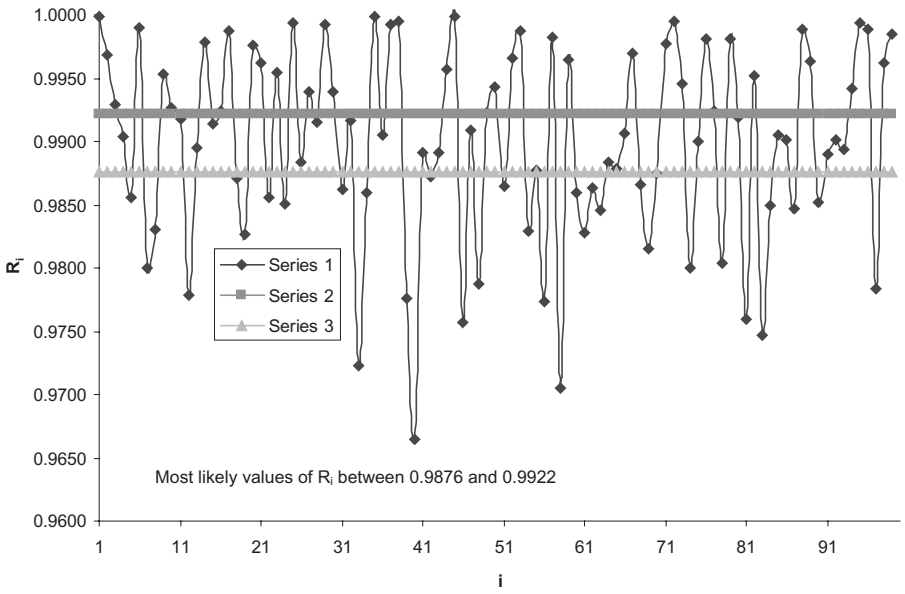


Figure 10.3 Switch operation i reliability R_i versus switch operation i. Series 1: R_i. Series 2: Mean R_i + 3*(standard error of mean) = 0.9922. Series 3: Mean R_i – 3*(standard error of mean) = 0.9876.

between architectural efficiency and reliability is a subtle one. That is, efficient architectures can be relatively unreliable!

OS CHARACTERISTICS

There are a number of characteristics of an OS that determine its ability to effectively manage system resources. The first is the scheduling algorithm. Scheduling refers to the way processes are assigned to run on the available central processing units (CPUs), since there are typically many more processes running than there are available CPUs. This assignment is carried out by software known as a scheduler and dispatcher.

The scheduler is concerned mainly with:

Throughput. The number of processes that complete their execution per time unit.

Turnaround. The total time between submission of a process and its completion.

Response Time. The amount of time it takes from when a request was submitted until the first response is produced.

Fairness. Equal CPU time to each process (or more generally appropriate times according to each process' priority).

In practice, these goals often conflict (e.g., throughput vs. latency), thus a scheduler will implement a suitable compromise.

In real-time environments, such as mobile devices for automatic control in industry (e.g., robotics), the scheduler must also ensure that processes can meet deadlines; this is crucial for keeping the system stable.

Long-Term Scheduler

The long-term scheduler decides which jobs or processes are to be admitted to the ready queue; that is, when an attempt is made to execute a program, its admission to the set of currently executing processes is either authorized or delayed. Thus, this scheduler dictates what processes are to run on a system, and the degree of concurrency to be supported at any one time (i.e., number of processes are to be executed concurrently), and how the split between input/output (I/O)-intensive and CPU-intensive processes is to be handled. This is used to make sure that real-time processes get enough CPU time to finish their tasks [STA04].

Mid-Term Scheduler

The mid-term scheduler temporarily removes processes from the main memory and places them on the secondary memory (such as a disk drive), or vice versa. This is

commonly referred to as "swapping out" or "swapping in," respectively. The mid-term scheduler may decide to swap out a process which has not been active for some time, or a process which has a low priority, or a process which is taking up a large amount of memory, in order to free up main memory for other processes, swapping the process back in later when more memory is available [STA04].

Short-Term Scheduler

The short-term scheduler decides which of the ready, in-memory processes are to be executed next (allocated a CPU) following a clock interrupt, an I/O interrupt, or OS call. Thus, the short-term scheduler makes scheduling decisions more frequently than the long-term or mid-term schedulers. A scheduling decision will be made when there is completion of an event, signaled by an interrupt, or periodically. This scheduler can be preemptive, implying that it is capable of forcibly removing processes from a CPU when it decides to allocate that CPU to another, higher priority process, or nonpreemptive, in which case the scheduler is unable to "force" processes off the CPU [STA04].

Dispatcher

Another component involved in scheduling function is the dispatcher. The dispatcher is the module that gives control to the process selected by the short-term scheduler. This function involves the following:

Switching among processes

Jumping to the proper location in a program to start its execution

The dispatcher should be as fast as possible, since it is invoked during every process switch

Scheduling Efficiency

Scheduling efficiency is an important OS performance metric. It is formulated by considering that the number of programs that have been queued, waiting for service, as the result of switch operation i, nq_i, related to the number of programs that have been scheduled as a result of switch operation i, n_i, measures scheduling efficiency, E_s, because programs waiting in a queue decrease the scheduling rate. Thus, E_s is computed as follows:

$$E_s = nq_i/n_i.$$

Then, the number of programs queued, waiting for service, is equal to the program input speed to the queue, S_i, as a result of switch operation i, times the switch operation i wait time, tw_i:

$$nq_i = S_i * tw_i.$$

To compute tw_i, note that the probability, ρ_i, of a queue being busy, as the result of switch operation i, is defined as:

$$\rho_i = tw_i/T_i,$$

where T_i is the switch operation i time. This equation reflects the fact that the higher the lower the wait time for a given switch time, the lower the wait time.

Now, combining the foregoing equations, the number of programs in the queue generated by switch operation i is the following:

$$nq_i = S_i * \rho_i * T_i.$$

Last, scheduling efficiency, E_s, is computed by using the last equation as follows:

$$E_s = (S_i * \rho_i * Ti)/ni.$$

Note that the probability of the queue being busy is unknown when the OS is designed. Therefore, this parameter must be estimated using the Excel RAND function.

By computing efficiency over 100 programs and then computing its mean, the value 0.2635 or 26.35% is produced. The utility of this analysis is that scheduling efficiency can be estimated during design, in advance of implementation, and increased if warranted by the estimated value. In this example, an increase is needed, which would be accomplished by increasing the switching speed, S_i.

Annoying Messages

An important distraction to user computing productivity is the plethora of annoying messages concerning, for example, never-ending notification of security settings and offers to update software, which various vendors seem compelled to present to the user whether the user is interested or not. In some cases, the messages can be turned off. In other cases, it is very difficult or impossible to turn them off. The problem is that rather than the default mode being "no messages," the default mode is "maximum messages"! Thus, it is important for the prospective customers of OS and application software to give the system a trial run before purchasing.

SCHEDULING ALGORITHMS

OS scheduling algorithms can be classified into two major categories related to the computing environment. One pertains to personal computer processing where program execution is triggered by user actions, typically with a mouse or Internet browser. In this case, there is really no need for a scheduler because program invocation is preordained by user actions. The more interesting and challenging case is the service computing environment: Web servers, file servers, e-mail processing, and so

on. In this environment, program invocation should be triggered by interrupts. The reason is that in order to not waste time, the OS should only service a program when the program needs servicing (e.g., a Web server receives a search request). However, before rushing to conclude that an interrupt-driven scheduler is sufficient to achieve efficient use of resources, consider the possibility that once a program gains control of the microcomputer, it may execute for a long time, thus preventing other programs from executing for a prolonged period of time. Then, how can this problem be resolved? A solution is to allocate a time slice to a program once it has signaled the need for service via an interrupt. The time slice is the amount of execution time allocated to a program. In a generalized computing environment, such as Web searching, all users have equal priority; thus, each program is allocated the same fixed time slice.

Contrariwise, in a real-time environment wherein deadlines must be met, the order and size of time slices is the order of deadlines. That is, the program with the first deadline receives the next time slice whose length is equal to the difference between the deadline time and the current time (times are determined by the microcomputer clock). In both cases, the scheduler performs interrupt-driven [WAN06] time-slice *allocation*, and switching logic *controls* the execution of programs, as shown in Figure 10.5. While this strategy may result in some programs not meeting their deadlines because a given program has control of the microcomputer until it meets its deadline, it is a sound strategy because at least there is assurance that the given program will meet its deadline.

Next, the generalized computer environment scheduling process will be formulated, using the following definitions and equations:

TS_i: length of time slice for switch action i

NI_i: number of instructions executed during time TSi

Since in a generalized computing environment this quantity is unknown a priori, it is estimated by using random number generation multiplied by a practical factor, say 10,000.

n_i: number of programs allocated time slices as a result of switch operation i

CR: microcomputer clock rate (1/CR = time of clock pulse)

It is assumed that one instruction is executed per clock pulse.

Typical values of CR are 2 and 4 GHz, yielding (1/CR) = 0.5 and 0.25 ns, respectively.

Using the above definitions, the length of the time slice is formulated as follows:

$$TS_i = ((1/CR) * NI_i)/n_i.$$

Figure 10.4 shows the result of the time-slice analysis wherein two factors drive the length to decreasing quantities: one is that, of course, as the number of programs that must be serviced increase, the length of the slice, necessarily, decreases. The second factor, the microprocessor speed (clock rate), may not be so obvious. With

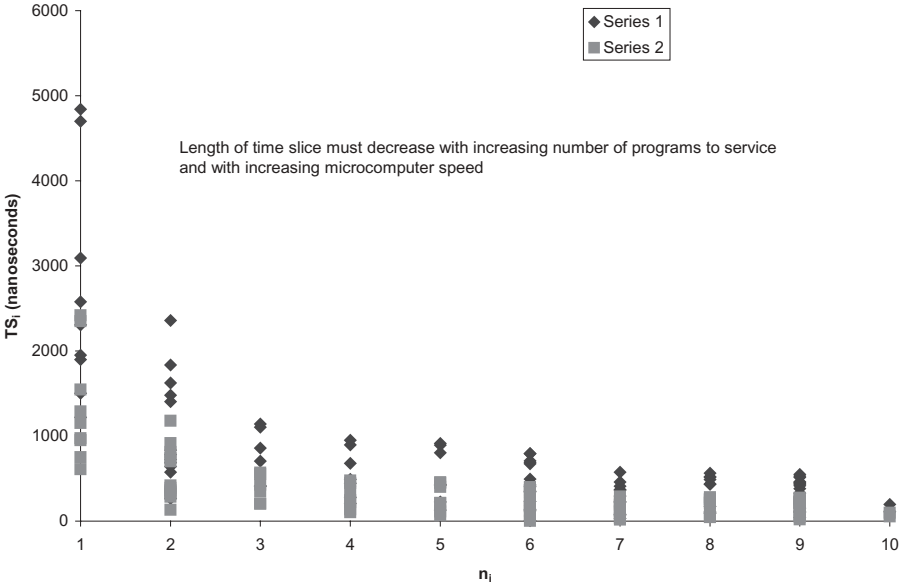


Figure 10.4 Time-slice length of switch action i, TS_i, versus number of programs switch action i, n_i. Series 1: Microcomputer speed = 2 GHz. Series 2: Microcomputer Speed = 4 GHz.

a higher speed, less time is required per instruction. Therefore, the time slice decreases.

Since for real-time systems the time-slice length strategy is highly dependent on deadlines that evolve with unpredictable events in real-time that cannot be predicted at OS design time, the next deadline, TD_i, associated with switch action i, is estimated by considering it to be randomly distributed, using the RAND function and a multiplication factor of 10 to make the estimates realistic. Time slice TS_i is estimated by the difference between the next deadline and the previous deadline, TD_{i-1}, associated with switch action i – 1, as follows:

$$TS_i = TD_i - TD_{i-1}.$$

Since, as stated, assigning a time slice to one program may cause other programs to miss their deadlines, it is necessary to estimate this blocking delay, TB_i: the difference between the deadline associated with switch action i + 1 and the deadline associated with switch action i, as follows:

$$TB_i = TD_{i+1} - TD_i.$$

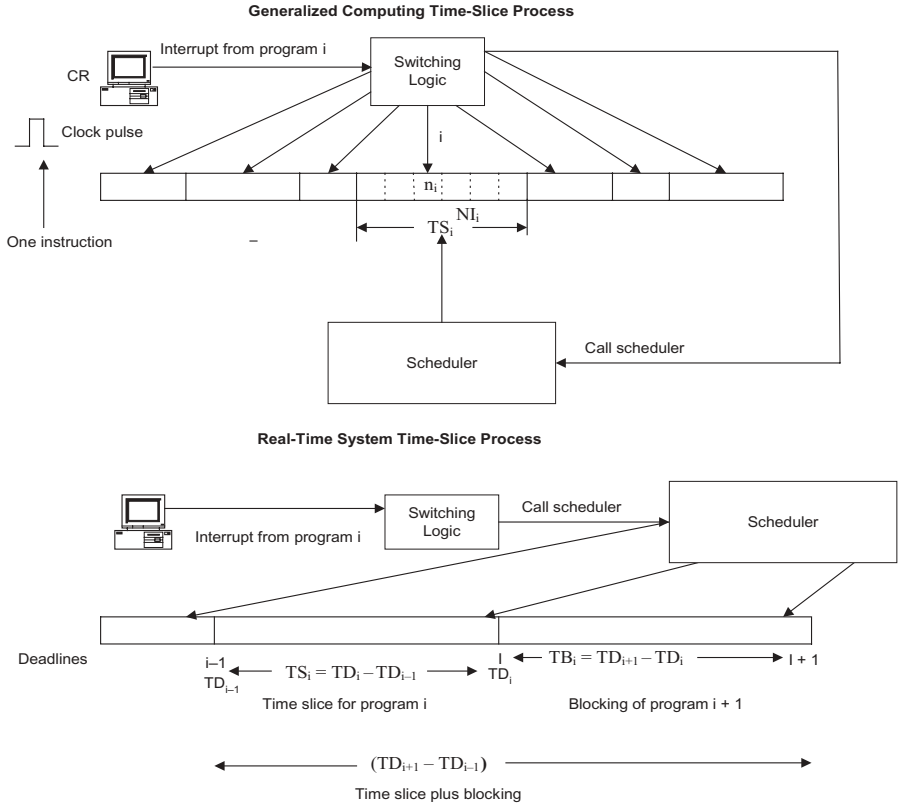
A measure of real-time scheduler scheduling efficiency, RT_i, is the fraction of time between deadline i + 1 and deadline i – 1 that is consumed by blocking time, computed as follows:

$$RT_i = (TD_{i+1} - TD_i)/(TD_{i+1} - TD_{i-1}).$$

To obtain an overall estimate of the scheduler efficiency and blocking delay over n switching actions, the means of TS_i, TB_i, and RT_i are computed. These values are: 0.0993, 0.1003, and 0.5057 seconds, respectively. The result suggests that with more time spent in being blocked, over 50%, as opposed to productive computing (time slice), the scheduler efficiency should be improved. An appropriate solution would be a very fast microcomputer. While significant blocking may still occur, the time lost to blocking would be significantly reduced. Thus, in advance of scheduler implementation, it is possible to estimate the penalty incurred by using a scheduling policy that assigns time slices equal to the deadline requirements.

The logic of time slicing in generalized computing is developed in Figure 10.5 along with the real-time system scheduling.

PD_i



$RT_i = (TD_{i+1} - TD_i)/(TD_{i+1} - TD_{i-1})$: scheduling efficiency

Figure 10.5 Time-slice process. TS_i, length of time slice for switch action i; n_i, number of programs allocated time slices as a result of switch operation i; NI_i, number of instructions executed during time TS_i; CR, microcomputer clock rate.

MEMORY MANAGEMENT

If all the memory requirements of all the programs that an OS must manage could fit into the main memory, there would be no need for memory management. However, as the size of programs continues to grow due to increasing user requirements, memory requirements expand exponentially. Thus, memory management has become a major component of contemporary OSs. This section contains an important quantitative treatment of memory management that is missing from the mostly qualitative coverage of OS texts. Consider the following definitions of memory management, as related to switching actions, which *trigger* memory accesses:

M: size of main memory (e.g., RAM) that is required by programs

P: fixed page size used in memory accesses triggered by switch operations, where a page is a subset of M, designed to allow only the instructions and data that are required for a given program's memory requirement to be a resident in M. This concept permits multiple processes, each of which has memory requirements, to be active at the same time. Note that the use of fixed size pages does not utilize memory as efficiently as variable size pages (i.e., wasted space when a page does not fit in M). However, because variable size pages are difficult to implement, OS designers opt for fixed size pages.

N: number of page transfers from secondary storage to M required by a program's operations, triggered by switch action i:

$$N = M/P.$$

PT: total paging time generated by n switching actions (n programs):

$$PT = \sum_{i=1}^{n} T_i N = \sum_{i=1}^{n} (T_i)(M/P),$$

where T_i is the time of switch action i (i.e., page transfer) and $PT_i = (T_i)(M/P)$ is the page transfer time per single program (i.e., switch action i).

PR: paging rate PR = 1/PT

C: page cost C = P*c, where c is the cost per megabyte

Now, our objective is to achieve a relatively high benefit–cost ratio, BC, consistent with minimizing the page transfer time per single program. Doing this provides a reasonable balance between BC and performance:

$$BC = PR/(P * c).$$

Figures 10.6 and 10.7 show how this balance is achieved, wherein Figure 10.6 documents the minimum single program page transfer time and corresponding page size. Then this information is used in Figure 10.7 to identify the "reasonable balance" BC. Last, Figure 10.8 provides the reader with a pictorial view of the mechanics of memory management.

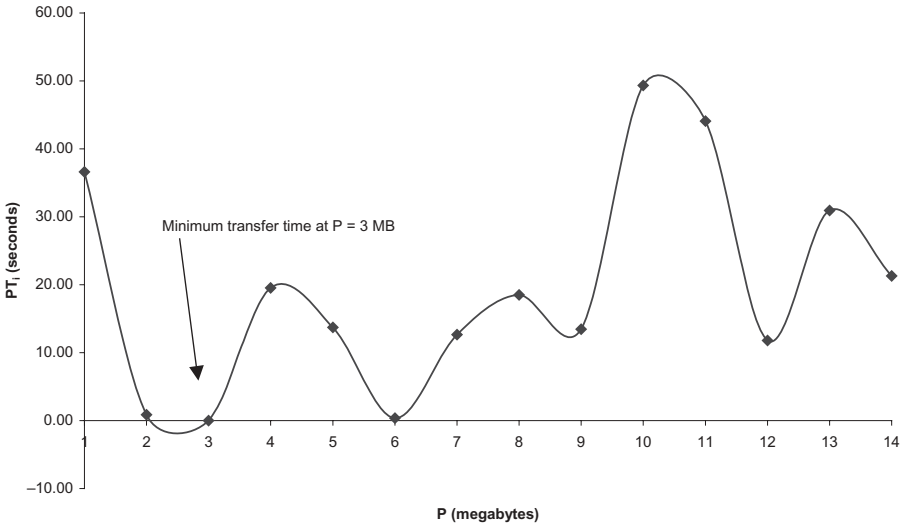


Figure 10.6 Page transfer time for single program (switch action) PT_i versus page size P.

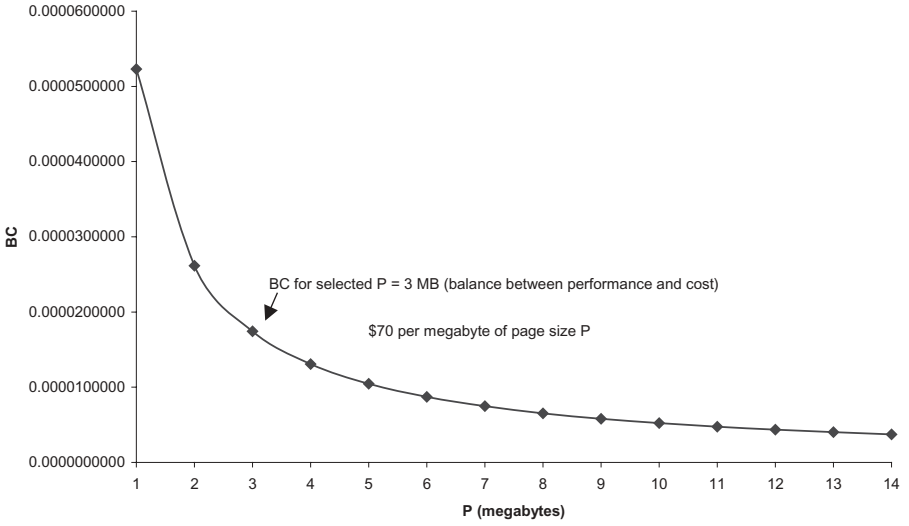


Figure 10.7 Paging benefit–cost ratio BC versus page size P.

DEADLOCK ANALYSIS AND PREVENTION

A deadlock is a situation in which two computer programs sharing the same resource are preventing each other from accessing the resource, resulting in both programs ceasing to function.

The earliest computer OSs ran only one program at a time. All of the resources of the system were available to this one program. Later, OSs ran multiple programs

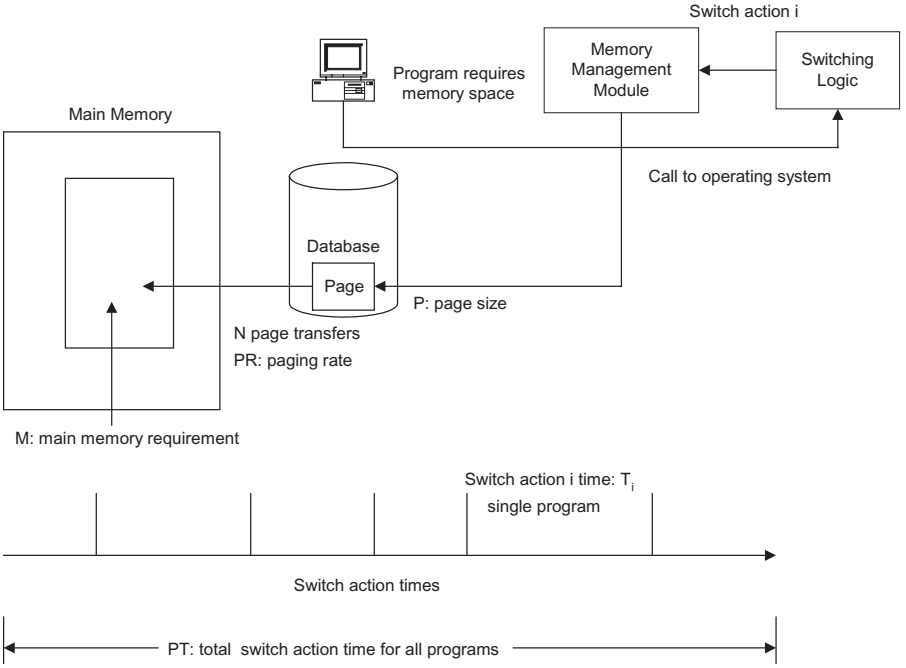


Figure 10.8 Paging operation.

at once, interleaving them. Programs were required to specify in advance what resources they needed so that they could avoid conflicts with other programs running at the same time. Eventually some OSs offered dynamic allocation of resources. Programs could request further allocations of resources after they had begun running. This led to the problem of the deadlock. Here is the simplest example:

Program 1 requests resource A and receives it.

Program 2 requests resource B and receives it.

Program 1 requests resource B and is queued up, pending the release of B.

Program 2 requests resource A and is queued up, pending the release of A.

Now, neither program can proceed until the other program releases a resource. The OS cannot know what action to take. At this point, the only alternative is to abort (stop) one of the programs. Learning to deal with deadlocks has had a major impact on the development of OSs. A solution to the deadlock problem is to allocate all the resources the program needs to complete its processing [REI04]. While this solution may prevent other programs from executing for a prolonged period of time, it does have the advantages of being relatively simple to implement, thus avoiding program failures, and of guaranteeing that at least one program will run to completion. Preventing deadlock is difficult if the OS allows for programs to execute concurrently. Note that this does not mean simultaneous execution; rather, it refers to two or more programs executing during a period of time allocated by the scheduler.

It is possible to estimate the probability, PD_i, of deadlock in a concurrent computing environment, when switch action i triggers execution of program i. The first factor that governs this probability is the probability of N_i, the number of computer resources (e.g., main memory) that are concurrently invoked by switch action i, related to the total number of resources, N. Thus, this probability is Ni/N.

The second factor that must be considered is the probability of n_i programs being invoked concurrently by switch action i, related to the total number of programs, n, invoked over all switch actions. Thus, this probability is:

$$\frac{n_i}{\sum_{i=1}^{n} n_i}.$$

The third and last factor is the number of complete computer systems (i.e., processor, memory, and all peripheral devices), N_s. The probability of deadlock is inversely proportional to N_s because the greater the number of computer systems, the lower the resource conflicts that cause deadlocks. Putting these factors together, the probability of deadlock is estimated as follows:

$$PD_i = \left((N_i / N) * \left(\frac{n_i}{\sum_{i=1}^{n} n_i} \right) \right) / N_s.$$

As we would expect, Figure 10.9 shows that the probability of deadlock increases with the number of concurrent programs and decreases with the number of available computer systems. Thus, this type of estimate is useful for planning resource utilization to avoid deadlocks: moderate concurrency coupled with the availability of several computer systems.

DISTRIBUTED OSS

The development of distributed OSs was partly motivated by a desire to escape from the limitations of centralized OSs, which has the disadvantage of centralizing resource allocation management, such as memory management, with attendant failure vulnerability (i.e., single point of failure) and lowered performance (i.e., slowdown caused by all programs competing for the attention of an OS function, for example, a scheduler). Hence, the distributed OS was developed, which distributes the processing load across processing elements [THU79]. These processing elements have their own interconnected memory and I/O units, thus achieving modularity of design [THU79]. The performance penalty for achieving greater autonomy of resource management is the time delay incurred when elements communicate via messages. Also, distributed systems virtually eliminate deadlocks by

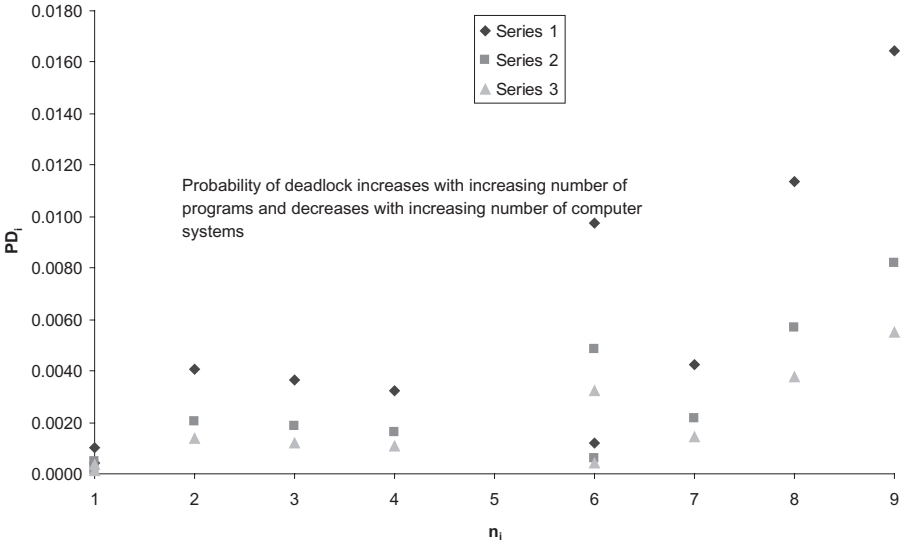


Figure 10.9 Probability of deadlock by switch action i, PD_i, versus number of concurrent programs executing by switch action i, n_i. Series 1: number of computer systems = 1. Series 2: number of computer systems = 2. Series 3: number of computer systems = 3.

virtue of using several autonomous computing elements, each equipped with all the resources needed to execute several program concurrently.

VIRTUAL OSS

Virtualization enables installation and running of multiple virtual machines on the same computer system. The OS that communicates directly with the hardware is known as the host OS, whereas virtual OSs have all the features of a real OS, but they run on virtual machines inside the host OS. A virtual machine is separated from the host computer hardware and it runs in emulation mode (i.e., software emulates hardware operations). The performance of a virtual OS running on the same computer system as the host OS depends on the performance of the host OS [MAR10]. The benefit of virtual OSs is the isolation that they provide from faults occurring in other virtual OSs and in the host OS. Thus, a high degree of reliability can be achieved. With virtual OSs based on time-slice allocation, during which time a given virtual OS has exclusive use of hardware resources, performance improvements ca also be achieved.

SUMMARY

The reader has been shown that it is important to estimate OS performance and reliability in advance of acquiring these systems by simulating the operating conditions under which an

OS would function. While the actual performance and reliability may differ from the estimates, nevertheless, statistical confidence intervals can bound the estimates such that deviations in operation from the estimates are very unlikely. Thus, the engineer who designs systems, of which the OS is a part, can anticipate OS performance and reliability in advance of committing time, effort, and funds to system implementation.

Question for the Reader: What OS characteristics would be appropriate for a computer system that is to control space flights?

Answer: It should have the characteristics of a real-time OS, meaning that it is imperative to meet deadlines (e.g., meeting launch schedule) by allocating time slices to programs in accordance with the logic shown in Figure 10.5.

REFERENCES

[AND92] T. E. Anderson, "The case for application-specific operating systems," *Proceedings of the Third Workshop on Workstation Operating Systems*, 1992, pp. 92–94.

[HAM95] D. Hamilton, "Are we answering the right research questions for commercial operating systems," *Proceedings. of the Fifth Workshop on Hot Topics in Operating Systems*, May 4–5, 1995, pp. 142–144.

[MAR10] G. Martinovic, J. Balen, and S. Rimac-Drlje, "Impact of the host operating systems on virtual machine performance," *Proceedings of the 33rd International Convention of MIPRO*, May 24–28, 2010, pp. 613–618.

[REI04] E. D. Reilly (ed.), *Operating Systems, Concise Encyclopedia of Computer Science*. Chichester: John Wiley & Sons, Ltd, 2004.

[STA04] W. Stallings, *Operating Systems Internals and Design Principles*, 5th ed.. Upper Saddle River, NJ: Prentice Hall, 2004.

[THU79] K. J. Thurber and G. M. Masson, *Distributed-Processor Communication Architecture*. Toronto: Lexington Books, 1979.

[WAN06] J. Wang, H. Zhao, P. Li, Z. Liu, J. Zhao, and W. Gao, "The mechanism and performance comparison of two wireless sensor network operating system kernels," *Sixth International Conference on Information Technology: New Generations*, 2006, pp. 1492–1446.

Chapter 11

Software Reliability and Safety

Having laid a foundation of reliability principles in Chapter 8, the reader is now prepared to study important applications of reliability, such as the risk to system safety of unreliability. Thus, the objective of this chapter is to develop and illustrate a software reliability risk profile that supports system safety. Understand that there is more to safety than reliability. However, it is clear that achieving reliability goals will support safety. The problem to be addressed is the development and analysis of a profile of software reliability risk metrics designed to measure the risk of software *not* meeting requirements with respect to reliability, time to failure, and remaining failures. If these goals are not achieved, catastrophic failures could occur that would jeopardize the mission. This problem is important because while there are many papers and texts about various reliability prediction models, there is inadequate attention to evaluating and responding to the risk to the mission of predictions that fail to achieve reliability goals.

RISK EVALUATION

During project development, risk is any threat to the development and delivery of a reliable product. The primary goal of software developers is the production of reliable systems that meet the needs of the user. To meet the goal of reliable software, developers focus on particular risks, including reliability risks [GOT01]. Risk evaluation is performed because the operation of software may not go according to plan. Risk evaluation is essential for spacecraft software. Spacecraft software is particularly critical, because its failure can directly jeopardize the mission (e.g., software's role in Ariane V's demise [ARI96], and as the most probable cause of loss of the Mars Polar Lander [JPL00]). Thus, it is important to develop metrics that can quantify risk and to consider the consequences of software operations that deviate from plans [CHI96].

Computer, Network, Software, and Hardware Engineering with Applications, First Edition. Norman F. Schneidewind.
© 2012 the Institute of Electrical and Electronics Engineers, Inc. Published 2012 by John Wiley & Sons, Inc.

First, I define risk as used in the analysis: Software risk is a measure of the probability that faults and failures will occur in the development of software [CHI94]. Thus, you can see that risk is related to reliability. Since testing is a key element of project development, the objective is to minimize the probability of reliability risk by testing the software to the extent that the predicted reliability exceeds the reliability goal during operation.

Then, I posit questions from the user's perspective related to concerns about risk.

What are the questions users might ask about the risk of using their software? Here are some of the major concerns:

Will the software satisfy my reliability goals?

Will the software operate without failure during my planned mission?

Will there be residual faults and failures after testing that would jeopardize the mission?

The risk evaluation process focuses on these concerns.

To provide for the evaluation of risk, each of the risk metrics must have a goal. These goals are the following:

Reliability. Predicted reliability must exceed *specified reliability* for the planned duration of the mission.

Time to Failure. Predicted time to failure must exceed the planned *duration* of the mission.

Remaining Failures. Predicted remaining failures must be *less* than a specified number of failures.

In addition, the *test time* when remaining failures have been reduced to an acceptable value is identified.

The consequences of not achieving these goals are the following: required reliability is not attained during a mission; the software fails during a mission because the time to failure is too short; and residual faults in the software lead to failures during a mission.

In addition to specifying the goals, the degree of risk computed by the risk metrics is quantified. In order to quantify risk, reliability prediction equations are developed. These equations and the corresponding risk metrics use National Aeronautics and Space Administration (NASA) Space Shuttle flight software failure data, related to orbital trajectory calculations. Because the Shuttle is a safety critical system, using failure data from this system is appropriate for illustrating modeling for achieving high reliability goals. A number of plots are made showing how, for example, risk varies with test time. If the plots indicate that the degree of risk would endanger the safety of the mission, corrective action is taken, for example, predict the amount of test time that would be required to reduce risk and achieve reliability goals.

OBJECTIVE

My objective is to develop and illustrate a software reliability risk profile that supports reducing uncertainty in achieving user reliability goals [GUP08]. The motivation is to address this aspect of risk that needs more attention. To illustrate, consider the following list of risks as presented in a research paper [ROP00]:

scheduling and timing risks,

system functionality risks,

subcontracting risks,

requirement management risks,

resource usage and performance risks, and

personnel management risks.

I wonder what happened to reliability risk! Therefore, I am motivated to develop the following reliability risk criteria:

Criterion 1. The reliability $R(t_s)$ predicted to be achieved for test time t_s must exceed the specified reliability R. The concept is that we must have confidence that the software satisfies reliability goals before it is released for operational usage. The specified reliability is made an increasing function of test time based on the premise that the reliability goal should increase as more faults are corrected with increased test time.

The risk of Criterion 1 is measured by:

$$\text{Risk } 1 = (R - R(t_s))/R = 1 - (R(t_s)/R).$$

If $R(t_s) < R$, the risk is positive and undesirable; otherwise, it is zero or negative and favorable.

Criterion 2. The predicted time to failure T must exceed mission duration t_m. The concept is that we want to be assured that that the mission can be completed with no failures.

The risk of Criterion 2 is measured by:

$$\text{Risk } 2 = (t_m - T)/t_m = 1 - (T/t_m).$$

If $T < t_m$, the risk is positive and undesirable; otherwise, it is zero or negative and favorable.

Criterion 3. The failures predicted to remain after the software is tested for a time t_s, $r(t_s)$, must *not* exceed r_c, where r_c is a specified critical value. It is also important that no residual failures remain when the software is released for operational usage.

The risk of Criterion 3 is measured by:

$$\text{Risk } 3 = (r_c - r(t_s))/r_c = 1 - (r(t_s)/r_c).$$

If $r(t_s) > r_c$, the risk is negative and undesirable; otherwise, it is zero or positive and desirable.

SOFTWARE RELIABILITY PROFILE IMPLEMENTATION

Now, each of the profile criterions will be implemented, using the NASA Space Shuttle Operational Increment OI8 (software release) as the source of failure data. These data are shown in Table 11.1. In order to implement the profile, the prediction equations of the Schneidewind software reliability model (SSRM) [SCH97, IEE08] will be used. Other extant models [LYU96] could also be used.

Criterion 1 (Reliability Risk)

Equation 11.1 is used to implement Criterion 1, where α, β, and s are parameters related to failure rate, estimated from the data in the Table 11.1. The long test times are due to the fact that the software for a given release (e.g., OI8A) is included in subsequent releases and undergoes additional testing in the combined software configuration. The procedure is to predict reliability $R(t_s)$ as a function of test time t_s and compare it with specified reliability, R, in order to predict Risk 1. If Risk $1 = 1 - (R(t_s)/R$ is positive, the prediction is less than the required reliability, and there is a risk of mission failure; otherwise, Risk 1 predicts a safe mission. In addition, note that predicted reliability increases with increasing test time in accordance with the concept that additional testing will remove additional faults.

$$R(t_s) = e^{-\left[\frac{\alpha}{\beta}\left[e^{-\beta(t_s-s+1)}-e^{-\beta(t_s-s+2)}\right]\right]}, \tag{11.1}$$

Table 11.1 NASA Space Shuttle OI8A Failure Data

Test time (days)	Number of failures	Cumulative failures
56	1	1
104	1	2
119	1	3
402	1	4
412	1	5
3077	1	6
4896	1	7
Model parameters		
Alpha	Beta	Xs–1
0.8747	0.0650	0
Initial failure rate	Rate of change of failure rate	Number of failures in range 1, s–1
s = 2	Starting time for parameter estimation	

where α, β, and s are failure rate parameters estimated from the Shuttle failure data in Table 11.1.

The reason for the low failure count is that the Shuttle software is highly reliable.

It is important to assess risk related to requirements early in the software development cycle so that corrective action can be taken before reliability problems are frozen in the software [APP05]. In addition, testing should be conducted as soon as possible to provide a *quantitative* assessment of risk sufficiently early to take corrective action, such as looking for potential software errors that could be generated by risky requirements. Risk-based reliability prediction is accomplished by specifying reliability R as a function of mission duration t_m, based on the premise that higher values of specified reliability should correspond to higher values of planned mission duration. Then, the software is tested to see how well predicted reliability matches the specified values, as shown in Figure 11.1.

Just predicting reliability does not tell the whole story about Criterion 1. We are also interested in how much test time is likely to be required to achieve the reliability goal. Thus, Equation 11.2 is produced by substituting R for $R(t_s)$ in Equation 11.1 and solving for t_s. Equation 11.2 is used to predict the test time t_s required to achieve the reliability requirement R, using the Shuttle continuous software testing regimen in the Shuttle simulators and in flight, as shown in Figure 11.2. In addition, in order to ensure a safe mission, it is required that testing continue for a time $t_s > t_m$. That is, testing under simulated operational conditions should continue for a duration longer than the planned mission duration:

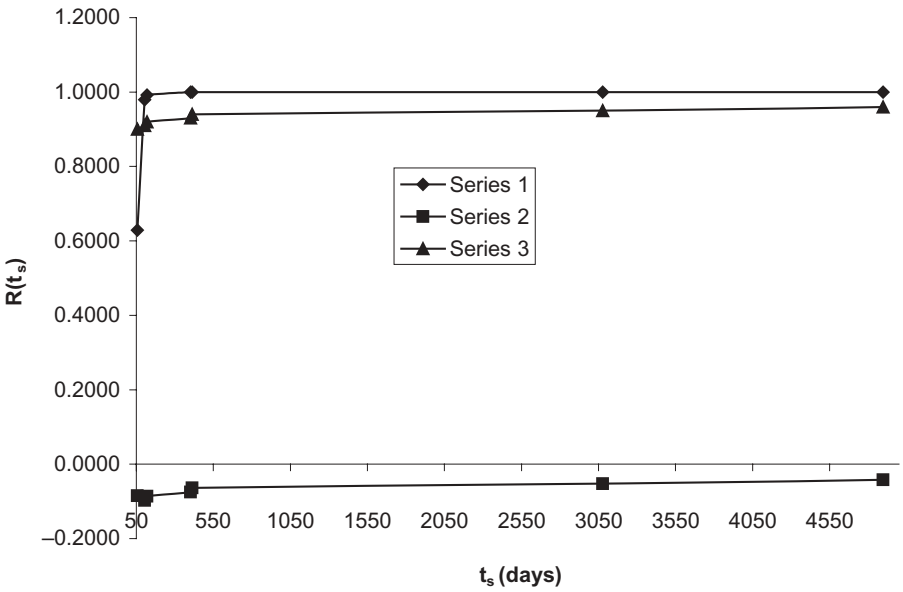


Figure 11.1 NASA Space Shuttle (OI8A) (SSRM): predicted reliability $R(t_s)$ and specified reliability R versus test time t_s. Series 1: $R(t_s)$. Series 2: Reliability risk. Series 3: R.

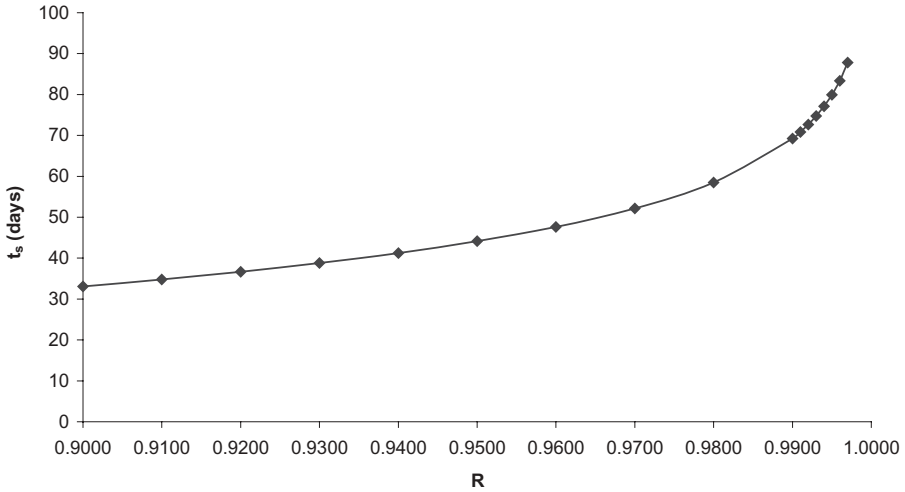


Figure 11.2 NASA space shuttle (OI8A): SSRM: predicted time required to achieve specified reliability R, t_s versus R.

$$t_s = \left[\left(-\frac{1}{\beta}\right)\log\left[\frac{\left(-\frac{\beta}{\alpha}\right)(\log R)}{\left(1-e^{-\beta}\right)}\right]\right] + (s-1). \tag{11.2}$$

The first step in the procedure for evaluating reliability risk is to plot Equation 11.1 as a function of test time and the specified reliability R in accordance with Risk 1. The purpose is to see whether the reliability objective has been achieved. As can be seen in Figure 11.1, it is achieved once there has been sufficient testing at 104 days when faults have been removed to the extent that the reliability goal can be achieved. Once this has occurred, missions can be launched for durations $t_m < t_s = 104$ days.

The second step is to see how fast risk can be decreased by achieving higher reliability by increasing test time. Figure 11.1 attests to the strategy of achieving the reliability risk goal by increasing test time.

The third step is evaluate the cost of testing, using test time as the surrogate for cost, to identify the value of achieved reliability where the cost becomes prohibitive. We see in Figure 11.2 that $R > 0.9800$ would result in exorbitant cost of testing (i.e., relatively large test times). Thus $R = 0.9800$, requiring $t_s = 58$ days of test time, is a reasonable objective that balances safety against cost.

Criterion 2 (Time to Failure Risk)

To address the risk posed by this criterion, predict the time to next failure as a function of given number of *cumulative failures* F(T), and relate it to Risk 2:

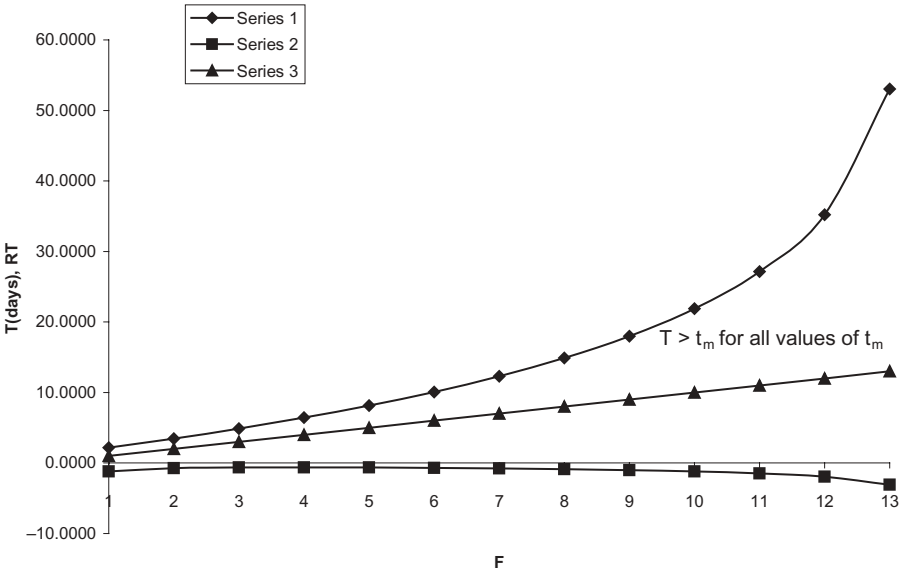


Figure 11.3 NASA space shuttle (OI8A) predicted time to failure, T, and time to failure risk, RT, versus number of failures F. Series 1: T. Series 2: RT. Series 3: Mission duration: t_m (days).

$RT = (t_m - (T)/t_m = 1 - (T)/t_m)$ to see which mission durations constitute the highest risk. To do this, the time to next failure(s) is needed in Equation 11.3 [SCH97]:

$$T = -\frac{1}{\beta}\log\left[1-(F(T)+X_{s-1})\left(\frac{\beta}{\alpha}\right)\right]+s-1 \text{ for } (F(T)+X_{s-1})\left(\frac{\beta}{\alpha}\right)<1, \tag{11.3}$$

where F is the given number of *cumulative failures* and X_{s-1} is the number of failures in the range 1, s–1. Figure 11.3 demonstrates that all mission durations are safe to fly because in all cases, $T > t_m$, where the range of t_m = 1, 13 days.

Criterion 3 (Remaining Failures Risk)

This criterion uses Equation 11.4 [SCH97] to predict the number of failures remaining, $r(t_s)$, after the software has been tested for a time t_s. This prediction provides an assessment of residual faults in the software as a function of test time, leading to the identification of test time required to predict the risk associated with the remaining failures criterion. This time is predicted as 41 days in Figure 11.4 for a criterion of one remaining failure. That is, a minimum test time of 41 days is required to ensure that the remaining failures criterion is satisfied:

$$r(t_s) = \frac{\alpha}{\beta}\left[\exp\left(-\beta\left(t_s-(s-1)\right)\right)\right]. \tag{11.4}$$

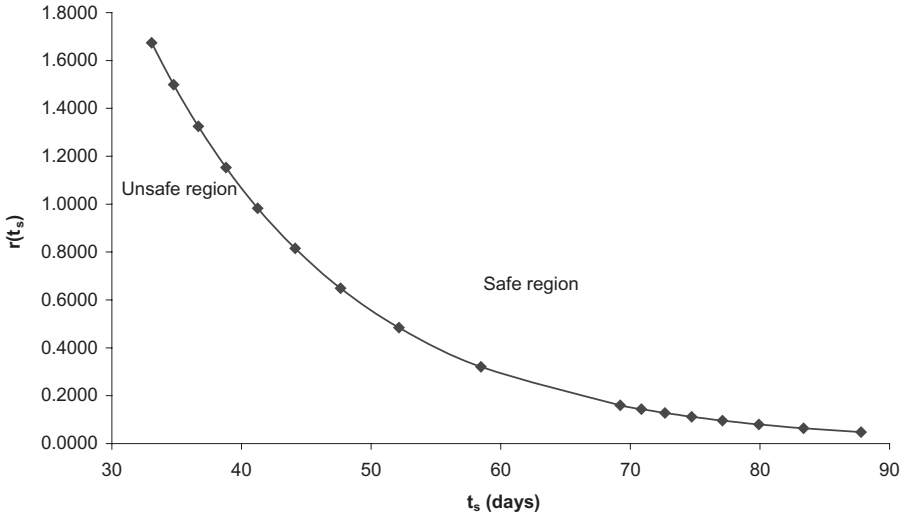


Figure 11.4 SSRM: predicted remaining failures $r(t_s)$ versus test time t_s.

Table 11.2 Summary of Risk Prediction Results

Risk criterion	Figure	Result
Reliability	Figure 11.1	$R(t_s) > R = 0.9797$ at $t_s = 104$ days
Reliability	Figure 11.2	Cost of testing high for $t_s > 58$ days and $R > 0.9800$
Time to failure	Figure 11.3	$T > t_m$ for all values of t_m
Remaining failures	Figure 11.4	Safe region: $t_s > 41$ days for $r_c = 1$

t_s, test time; $R(t_s)$, predicted reliability; R, specified reliability; T, predicted time to failure; t_m, mission duration; r_c, remaining failures criterion.

Question for Reader: Why is the remaining failures criterion zero remaining failures rather than one?

Answer: As can be seen in Figure 11.4, predicted remaining failures decreases asymptotically with increasing test time. Thus, it would require an infinite amount of test time to achieve zero remaining failures. In addition, from a practical standpoint, no software of any consequence is error free. If it appears error free because no errors have been experienced, either the program has not been executed long enough or the code where the errors are hiding has not been executed! Thus, a criterion of "one" is a practical goal.

Summary of Prediction Results

The purpose of Table 11.2 is to assemble the prediction results corresponding to the risk criteria in one place so that the summary results can be identified. Based on the results, the following are the key findings:

Reliability requirement is achieved at R = 0.9797 at t_s = 104 days.

Cost-effective predicted reliability is approximately 0.9800.

Test software for a minimum of 41 days to achieve the remaining failures requirement.

Test for a maximum of 58 days to avoid excessive cost of testing.

All planned mission durations are safe.

In summary, one must test for 104 days to achieve the reliability objective, even though it is not cost-effective. This decision gives a higher priority to safety than to cost.

Risk Control

It is insufficient to predict risk. In addition, risk control and mitigation is necessary for developing and implementing risk resolution plans (i.e., action to take if risk goal not achieved), monitoring risk status (measuring current risk and comparing it with planned risk), and correcting deviations from the plan [KHA09 , RUZ03]. For example, there could be errors in predicting reliability, time to failure, and remaining failures, resulting in inaccurate assessments of the conditions for safe missions. Monitoring risk involves recording the *actual future time to failure* during test and operation and comparing with predicted values. Then the mean relative error (MRE) is computed. For example, an MRE in excess of ±20% could be considered justification for discarding the current model and evaluating others [LYU96]. Risk mitigation can be implemented by refining predictions to improve their accuracy by using additional failure data generated from future tests and operations, designed to improve the accuracy of risk criteria computations.

Another consideration in risk control is mapping failures to their causes [FEA04]. To illustrate, the failures recorded for the NASA Space Shuttle software release OI8A in Table 11.1 spans the range of category 1, mission-threatening failures, to category 3, minor failures; workarounds are available for the latter. Thus, in the examples, since reliability predictions are based on these data, the predictions are *representative* of *typical* failure scenarios (e.g., *time to failure* predictions can produce a mix of category 1–3 *time to failure* predictions).

CONCLUSIONS

It is beneficial for risk analysis to focus on reliability because, after all, if expected reliability cannot be achieved, the software would be useless no matter what other qualities it may possess. Mission success can be measured by predicting the extent to which predicted reliability exceeds specified reliability. Other reliability-related metrics are *time to failure* and *remaining failures*. We would have confidence in the safety of the mission if *predicted time to failure* exceeds *planned mission duration* and *predicted remaining failures* are less than a *specified critical value.*

REFERENCES

[APP05] K. Appukkutty, H. H. Ammar, and K. G. Popstajanova, "Software requirement risk assessment using UML," *The 3rd ACS/IEEE International Conference on Computer Systems and Applications*, 2005, p.112.

[ARI96] Ariane 5 Inquiry Board, "ARIANE 5 Flight 501 Failure," 1996.

[CHI94] C. Chittister and Y. Y. Haimes, "Assessment and management of software technical risk," *IEEE Transactions on Systems, Man and Cybernetics*, 1994, 24(2), pp. 187–202.

[CHI96] C. Chittister and Y. Y. Haimes, "Systems integration via software risk management," *IEEE Transactions on Systems, Man and Cybernetics, Part A: Systems and Humans*, 1996, 26(5), pp. 521–532.

[FEA04] M. S. Feather, "Towards a unified approach to the representation of, and reasoning with, probabilistic risk information bout software and its system interface," *15th International Symposium on Software Reliability Engineering*, November 2–5, 2004, 391–402.

[GOT01] D. Gotterbarn, "Enhancing risk analysis using software development impact statements," *Proceedings of the 26th Annual NASA Goddard Software Engineering Workshop*, Greenbelt, MD, 2001, pp. 43–51.

[GUP08] D. Gupta and M. Sadiq, "Software risk assessment and estimation model," *International Conference on Computer Science and Information Technology*, August 29–September 2, 2008, Singapore, 2008 pp. 963–967.

[IEE08] IEEE/AIAA P1633™, "Recommended Practice on Software Reliability," June 2008.

[JPL00] JPL Special Review Board, "Report on the Loss of the Mars Polar Lander and Deep Space 2 Missions", March 2000, JPL D-18709, Jet Propulsion Laboratory, California Institute of Technology. 2000.

[KHA09] S. Khan, "An approach to facilitate software risk identification," *2nd International Conference on Computer, Control and Communication*, February 17–18, 2009, Karachi, pp. 1–5.

[LYU96] M. R. Lyu (ed.), *Handbook of Software Reliability Engineering*. Los Alamitos, CA: IEEE Computer Society Press; New York: McGraw-Hill Book Company, 1996.

[ROP00] J. Ropponen and K. Lyytinen, "Components of software development risk: how to address them? A project manager survey," *IEEE Transactions on Software Engineering*, 2000, 26(2), pp. 98–112.

[RUZ03] R. Xu, L. Qian, and X. Jing, "CMM-based software risk control optimization," *IEEE International Conference on Information Reuse and Integration*, October 27–29, 2003, pp. 499–503.

[SCH97] N. F. Schneidewind, "Reliability modeling for safety critical software," *IEEE Transactions on Reliability*, 1997, 46(1), pp. 88–98.

Part Four

Integration of Disciplines

Chapter 12

Integration of Hardware and Software Reliability

The objective of this chapter is to integrate hardware and software into a unified reliability model by using several reliability models in order to identify an appropriate integrated model, supported by failure data from several real-world projects. Several system configurations are evaluated, including series, parallel, and series–parallel. The Weibull distribution, because of its ability to model various failure rate patterns, is useful for identifying the reliability properties of each of the configurations. A reliability benefit–cost ratio, with cost based on the number of series and parallel components, is useful for evaluating model predictions. As a by-product of the modeling process, several reliability relationships are revealed that might be intuitively obvious, but are dramatized by quantitative analysis. For example, increasing the degree of hardware parallelism will not produce the desired reliability if hardware and software failure rates are excessive. In this situation, the only recourse to achieving acceptable reliability is testing to correct faults. In addition, in a component-based system, component failure rates must be extremely low in order to prevent the failure of even a single component that could bring the system down. This chapter uses several reliability principles covered in Chapter 8. The reader may want to refer to Chapter 8 because topics such as series and parallel reliability configurations and Poisson and Weibull distributions are used in this chapter.

INTRODUCTION

Objectives

The primary objective is to show the reader how to integrate hardware and software reliability into a single system reliability model. The reason for this is typically, hardware and software are treated as disparate entities in reliability analysis, when in fact they are intimately related. For example, an error in software causes a divide overflow, leading to a hardware divide overflow interrupt, which, in the user's view,

Computer, Network, Software, and Hardware Engineering with Applications, First Edition. Norman F. Schneidewind.
© 2012 the Institute of Electrical and Electronics Engineers, Inc. Published 2012 by John Wiley & Sons, Inc.

is a system failure. A second objective is to evaluate various software–hardware configurations such as series, series–parallel, and pure parallel in relation to cost, so that the reader can see which configuration produces the most favorable reliability benefit–cost (BC) relationship. Third, the analysis is supported with failure data from real-world projects in order to see how these data affect efforts to achieve high system reliability by using redundancy, for example.

A factor in achieving high reliability is that the quality of a product design is dependent on the quality of the process in which it is inserted. Changes in process evolve and this evolution should be taken into account when designing a product for high reliability [BON98]. For example, achievable product reliability is a function of software testing methodologies (e.g., testing by software function versus testing by program path). While this is true, process is beyond the scope of this chapter because there is no information available on the relationship between product and process quality.

Definitions

As an aid in understanding the development of reliability models, the following terms are defined:

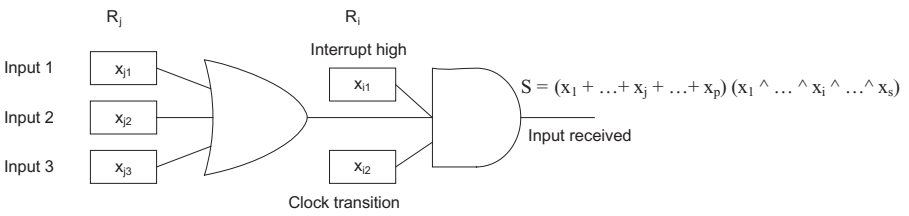
Series Component **(x_i).** Part of a series, or series–parallel, configured computer system (see Fig. 12.1), hardware or software [MUS87].

Parallel Component **(x_j).** Part of a parallel, or series–parallel, configured computer system (see Fig. 12.1), hardware or software [MUS87].

Operational Mode. The operating characteristics of a component [MUS87].

Operational modes can have different requirements and reliabilities [MUS87].

Requirements include both functional (e.g., performance) and nonfunctional (e.g., reliability) specifications [MUS87].



$$R_S = R_s\, R_p = \left(\prod_{i=1}^{s} R_i\right)\left(1 - \left(\prod_{j=1}^{p} (1 - R_j)\right)\right)$$

R_S: system reliability

Figure 12.1 Series–parallel reliability.

Concurrent System. Comprised of components that execute during the same scheduled operational time, but not simultaneously (e.g., computer and applications software) [MUS87].

Sequential System. Comprised of components that execute at different scheduled operational times (e.g., input data component execution followed by database management system execution) [MUS87].

Functional Logic

Reliability is related to the functions that are performed in a system. For example, if the output S in Figure 12.1 of a series–parallel hardware and software system is "1," it means that the input has been received and that system reliability of this event is R_S. The success of the input received function in Figure 12.1 of a series–parallel hardware and software system is given in Equation 12.1:

$$S = (x_1 + \ldots + x_j + \ldots + x_p)(x_1 \wedge \ldots \wedge x_i \wedge \ldots \wedge x_s), \tag{12.1}$$

where x_j and $x_i = 0$ or 1.

RELIABILITY LOGIC

Reliability logic refers to the ways in which reliability is computed for configurations of series, parallel, and series–parallel hardware and software components. This entails considering the way components are connected, as in Figure 12.1, which provides the logic for the number of terms and the operators (OR, AND) in the reliability equations. In addition, the reliability models for individual components (e.g., exponential, Weibull) are integrated with the connection logic.

Series–Parallel Configuration

In viewing the following reliability equations, refer to Figure 12.1 for a pictorial view of a series–parallel configuration, where components can execute sequentially (inputs followed by interrupt and clock transition) or concurrently (interrupt and clock transition during scheduled operating time).

The reliability of a series configuration with s components, each with reliability R_i, is computed in Equation 12.2:

$$R_s = \prod_{i=1}^{s} R_i. \tag{12.2}$$

The reliability of a parallel configuration with p components, each with reliability R_j, is computed in Equation 12.3. The rationale of this equation is that parallel configuration reliability is equal to 1 minus the parallel configuration *unreliability*:

$$R_p = 1 - \left(\prod_{j=1}^{p} (1 - R_j) \right). \tag{12.3}$$

Then using Equations 12.2 and 12.3, and assuming *series system* reliability R_S, it is computed in Equation 12.4:

$$R_S = R_s R_p = \left(\prod_{i=1}^{s} R_i \right) \left(1 - \left(\prod_{j=1}^{p} (1 - R_j) \right) \right). \tag{12.4}$$

Assuming series system reliability provides a conservative, or worst-case, computation, with the assurance that the system reliability will be no worse than predicted by Equation 12.4.

When the failure rate λ is constant mean value (i.e., exponentially distributed time t between failures), the reliabilities in Equations 12.2–12.4 are computed based on the exponential distribution in Equation 12.5:

$$R(t) = e^{(-\lambda t)}. \tag{12.5}$$

In hardware reliability, Equation 12.5 is used during the operational phase (neither burn-in nor wear out phases) when operating time is available [MIC05]. This corresponds to the Poisson failure count model in Equation 12.6 that is used when you want to predict the probability of x failures occurring, with a failure rate λ, for an operating time t. If you set x = 0 in Equation 12.6, you arrive at Equation 12.5:

$$P(x, t) = \frac{(\lambda t)^x e^{(-\lambda t)}}{x!}. \tag{12.6}$$

Equation 12.6 can also be used for evaluating the effect of failure rate of various components on probability of failure and to identify the number of failures x when the probability of failure becomes negligible. The latter analysis can be used to advantage in determining how long to test components (i.e., terminate testing when x failures have occurred and their faults removed).

Question for Reader: Can you think of an assumption that governs the structure of reliability Equations 12.2–12.4?

Answer: There is the assumption of independence of faults that cause failures, thus allowing component reliabilities to be multiplied. However, this may not be the cases because faults can be dependent. For example, one fault can mask another. The masked fault cannot be detected until the masking fault is removed [LYU96]. In such cases, the faults and failures and resultant component reliabilities are not independent. However, by multiplying component reliabilities, the salvation from this problem is that the assumption of independence leads to lower, and, hence, conservative reliability predictions.

Reliability When n Out of N Components Fail

In the preceding discussion of reliability, notice that component reliability is not taken into account—computations of reliability are at the system level. If N, the number of components in a system, is available, you can use Equation 12.7—the binomial distribution—to predict the system reliability R_S, based on n of the N components operating *without* failure, each with a reliability R. The component reliabilities, in turn, can be formulated on an exponentially distributed operating time basis by using Equation 12.8:

$$R_S = \left(\frac{N!}{n!(N-n)!}\right)\left(R^n\right)\left((1-R)^{(N-n)}\right), \tag{12.7}$$

$$\text{where } R = e^{(-\lambda t)}. \tag{12.8}$$

To predict the operating time t corresponding to specified reliability R and mean failure rate λ, use Equation 12.9, which is obtained by solving Equation 12.8 for t. This equation is useful for predicting the duration of operating time that is feasible for an application with a specified reliability. If the time does not satisfy the operational requirement, it means that the specified reliability would have to be reduced to meet the requirement:

$$t = (-\log R)/\lambda. \tag{12.9}$$

The likelihood of processor failure during a long-running application that uses multiple processors increases with the number of processors, and the failure of a single processor can crash the entire system. Detecting faults and recovering from faults is thus a major concern in using these systems [CAR95]. On the one hand, based on the reliability of individual components, $R = e^{(-\lambda t)}$, reliability will decrease for a long-running application. On the other hand, if each processor (component) runs the same application, system reliability R_S in Equation 12.7 will increase as the number of processors, n, that *do not* fail increases. The net effect on system reliability depends on values of failure rate, λ, operating time, t, and n.

Cost Considerations

You should not evaluate the reliability of various computer configurations ignoring cost. For example, in configurations that involve parallel redundancy in order to increase reliability, there would be additional cost incurred compared with a series configuration. The penalty for using parallelism to achieve reliability improvement is the additional processors that are required. For a serial–parallel configuration comprised of one processor to communicate with s serial components (e.g., input–output, memory) and p processors to communicate with p parallel components, the total number of processors is c = p + 1. Since the cost of processors would be equal

for a given configuration, the cost is proportional to c, so that the benefit of increased system reliability R_S can be related to the cost c by the BC in Equation 12.10:

$$BC = R_S/c = R_S/(p+1). \tag{12.10}$$

If the configuration is pure series, p = 0 and c = 1; if it is pure parallel, c = p = N, number of components. A cautionary note is that since c = 1 for the series configuration, BC = R_S would look very favorable. However, the system reliability R_S must also satisfy the specified reliability requirement R. That is, $R_S \geq R$. Thus, first, the reliability requirement must be satisfied. Then, BC can be computed.

RELIABILITY ANALYSIS RESULTS

Series–Parallel Configurations

The disk failure rates used in the analysis of series–parallel configurations are shown in Table 12.1.

Figure 12.2 shows that only the pure parallel configuration satisfies the reliability requirement. While the reliability of the series configuration is poor, it does provide the worst case, so that you can be assured that reliability would be no worse than this case. Finally, you can see that the series–parallel configuration does not provide a significant advantage over the series configuration.

BC Considerations

Now, when the BC relationship is applied in Figure 12.3, the superiority of the series configuration for all values of operating time is evident. However, for a mission-critical application, operating for prolonged periods, and cost is a minor

Table 12.1 Disk Failure Rates (The Computer Failure Data Repository [CFDR], Carnegie Mellon University)

			Failures per day per disk			
Type of cluster	From day	To day	Days	Failures	Number of disks	Failure rate
			T	f	d	λ
High performance	37,104	38,838	1734	1263	3406	0.000214
High performance	37,987	38,899	912	14	520	0.000030
Internet server	38,000	38,031	31	465	26734	0.000561
Internet server	38,231	38,808	577	667	39039	0.000030
Internet server	38,353	38,687	334	346	3734	0.000277

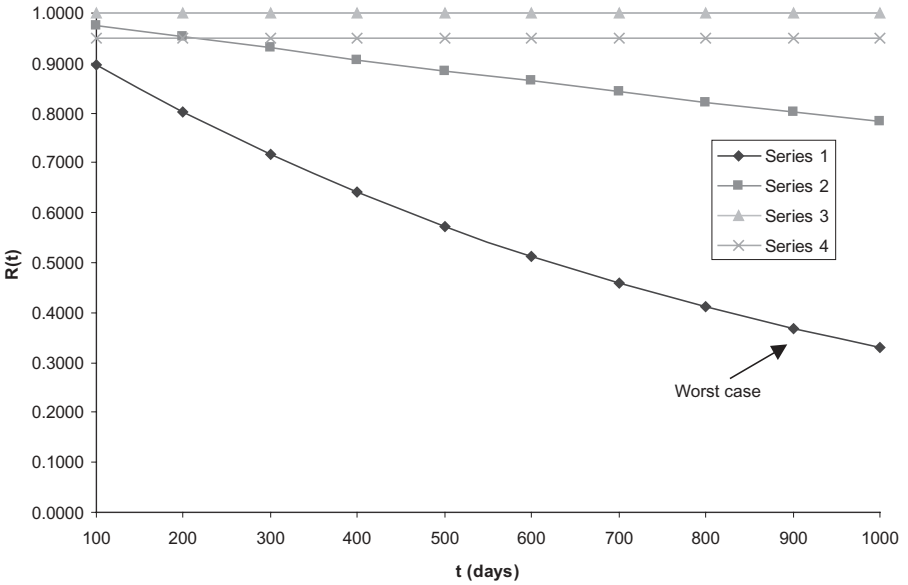


Figure 12.2 Disk reliability R(t) versus operating time t. Series 1: 5 disks connected in series, 1 processor required. Series 2: 2 disks connected in series, 3 disks connected in parallel, 4 processors required. Series 3: 5 disks connected in parallel, 5 processors required. Series 4: required reliability = 0.9500 (only parallel configuration satisfies reliability requirement).

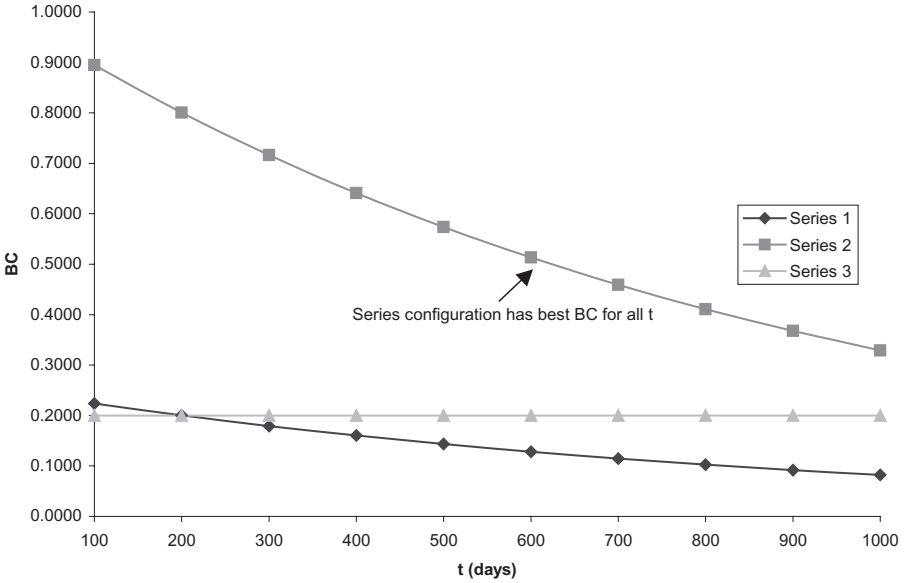


Figure 12.3 Reliability benefit–cost ratio BC for disk configurations versus operating time t. Series 1: Series–parallel configuration (2 disks connected in series, 3 disks connected in parallel, 4 processors required). Series 2: Series configuration (5 disks connected in series, 1 processor required). Series 3: Parallel configuration (5 disks connected in parallel, 5 processors required).

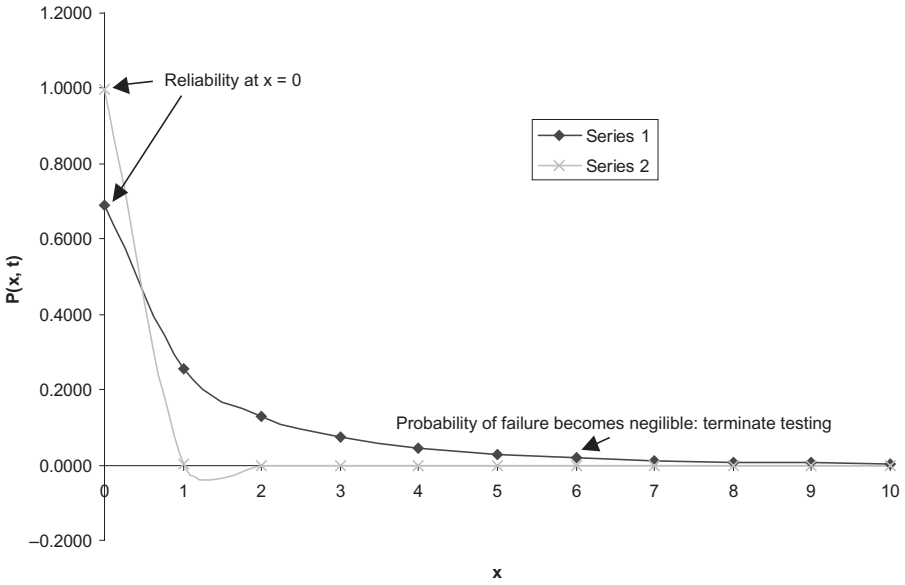


Figure 12.4 Disk probability of failure P(x, t) versus number of failures x. Series 1: failure rate = 0.370816 failures per day. Series 2: failure rate = 0.004271 failures per day.

consideration (e.g., nuclear power plant control), you would select the parallel configuration.

Probability of Failure

It is important to study the effects of failure rate and number of failures on probability of failure. In addition, since testing can be expensive, you want guidance for determining how long to test. Both of these issues are illustrated in Figure 12.4, where you can see that failure rate dramatically affects probability of failure, and at x = 6 failures detected, you could stop testing.

Figure 12.5 uses Equation 12.9 to estimate the operating time t that can be achieved for specified values of reliability R(t) for five disk systems with different failure rates. You can see that t decreases with increasing R(t) and failure rate. This *type* of figure could be employed to estimate the operating time that could be achieved for *any* hardware or software component whose reliability is described by the exponential function.

Component Reliability Analysis

Advances in multiprocessor technology have made possible the design of highly flexible parallel multiprocessor memory systems, such as the Los Alamos National

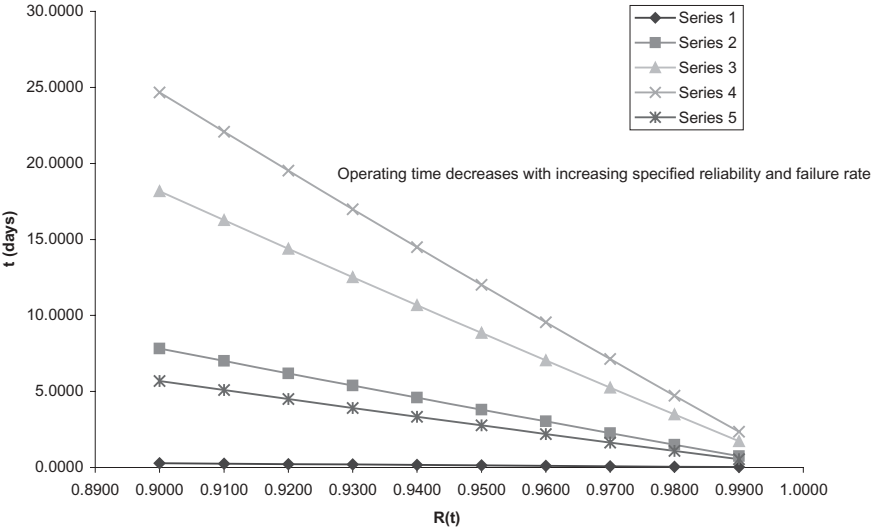


Figure 12.5 Disk operating time t versus required reliability R(t). Series 1: failure rate = 0.378016 failures per day. Series 2: failure rate = 0.013462 failures per day. Series 3: failure rate = 0.005798 failures per day. Series 4: failure rate = 0.004271 failures per day. Series 5: failure rate = 0.018532 failures per day.

Table 12.2 Component Failure Data (Los Alamos National Laboratory's ASC Q Supercomputer)

Component type failure rate	Memory failures per hour	Cache failures per hour	Parity failures per hour	CPU failures per hour	Hardware failures per hour	Total failures per hour
λ	0.0047	0.0061	0.0065	0.0075	0.0096	0.0127
Number of components	N	22				
Operating time	t	1.00 hour				
Operating time	t	3.50 hours				
Operating time	t	4.50 hours				

Laboratory computer documented in Table 12.2. High reliability is required for these systems because a small degradation in a component (processor or memory) can be catastrophic by significantly lowering the overall system reliability. High reliability of these systems has been commonly achieved by utilizing redundancy [CHO02]. Therefore, the component-based reliability relationships are investigated, including redundancy that applies when individual component reliabilities are predicted and the results put into a larger framework of generating system reliabilities. This

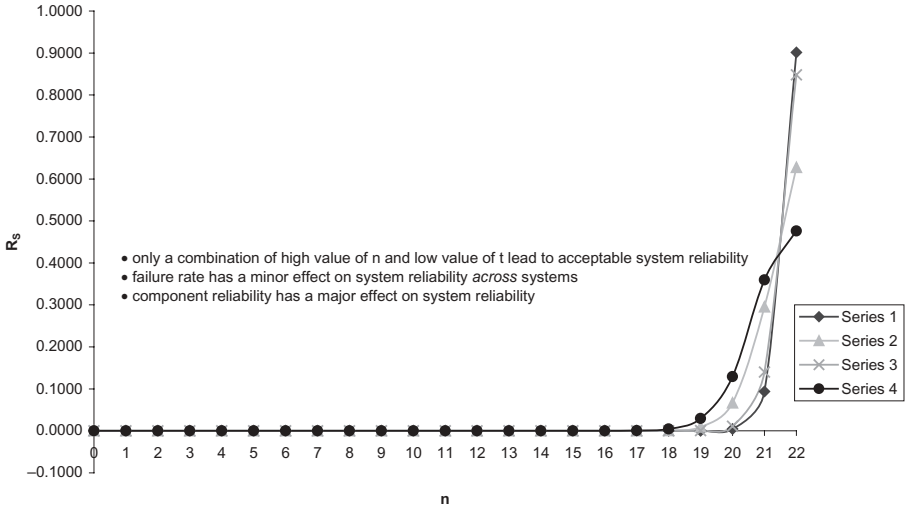


Figure 12.6 System reliability R_S versus number of components that do not fail n. Series 1: R_S, memory failures, failure rate = 0.0047 failures per hour, operating time t = 1 hour, component reliability = 0.9953. Series 2: R_S, memory failures, failure rate = 0.0047 failures per hour, operating time t = 4.5 hours, component reliability = 0.9790. Series 3: R_S, CPU failures, failure rate = 0.0075 failures per hour, operating time t = 1 hour, component reliability = 0.9925. Series 4: R_S, CPU failures, failure rate = 0.0075 failures per hour, operating time t = 4.5 hours, component reliability = 0.9668.

analysis is informative because you can study the effects on system reliability of failure rate, component reliability, operating time, and number of components that *do not* fail. The data that were used to support this analysis are shown in Table 12.2. As can be seen in Figure 12.6, when very few components fail (i.e., n is large) and operating time is low, these are conditions for producing acceptable system reliability (i.e., approximately 0.9000).

Another perspective on component reliability evaluation can be obtained by using the Poisson probability of failure that was introduced in Equation 12.6, but this time rather than number of failures, the focus is on number of failed components n in Equation 12.11:

$$P(n, t) = \frac{(\lambda t)^n e^{(-\lambda t)}}{n!}. \tag{12.11}$$

The purpose of this examination is to determine whether there is a significant probability of multiple failed components. Using the same components, failure rates, and operating times that were explored in Figure 12.6, you can produce Figure 12.7, revealing that for both memory and central processing unit (CPU) components, the probability of multiple failed components is negligible. Therefore, the prospects are good of achieving high reliability in this multiple component system.

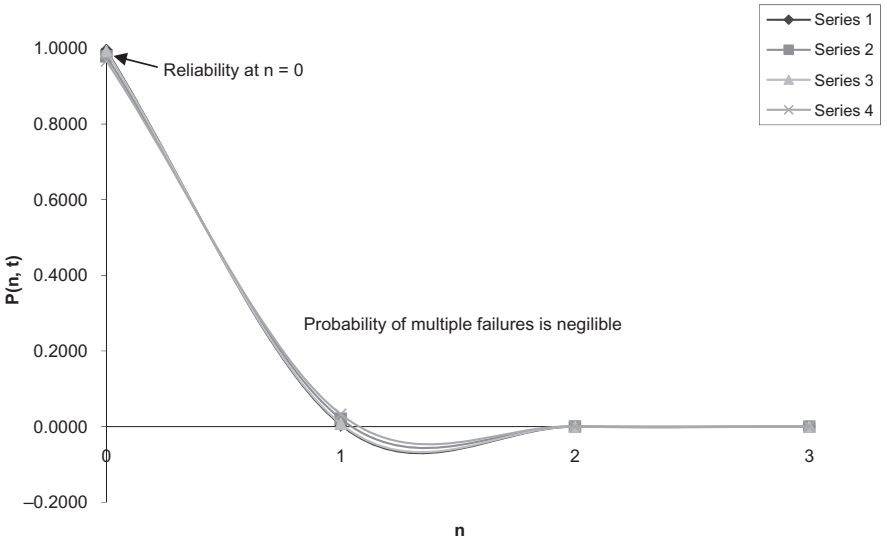


Figure 12.7 Probability of components failing at time t, P (n, t) versus n. Series 1: P(n, t): memory failures, failure rate = 0.0047 failures per hour, t = 1 hour. Series 2: P(n, t): memory failures, failure rate = 0.0047 failures per hour, t = 4.5 hours. Series 3: P(n, t): CPU failures, failure rate = 0.0075 failures per hour, t = 1 hour. Series 4: P(n, t): CPU failures, failure rate = 0.0075 failures per hour, t = 4.5 hours.

Assessing Reliability Model Predictive Accuracy

If you possessed historical *computed* reliability data for the systems tabulated in Tables 12.1 and 12.2, you could compute the error between the prediction models and the historical reliability data in order to assess the predictive validity of the models. Lacking this information, you can make a qualitative assessment as follows: If redundancy is used to improve hardware reliability, then a model with parallelism is most appropriate. If the major concern is to predict worst-case reliability, the series model should be used. If component failure data are available, the n out of N model is the most appropriate. Complementing these models is the Poisson probability of failure model that provides an additional quality perspective by predicting the probability of a specified number of failures occurring. This information can be used to determine how long to test (i.e., stop testing when a number of failures have been detected corresponding to a low value of probability of detection).

COMBINED HARDWARE–SOFTWARE RELIABILITY ANALYSIS

One approach to reduce the complexity of systems and, hence, render them suitable for reliability modeling is decomposition. To deal with the complexity of integrated

modeling, the functions of a computer system are successively divided by a functional decomposition method. The decomposition of a function into subfunctions stops when the smallest subfunctions cannot be divided further or dividing the subfunctions further will be of no interest. When the smallest subfunction is achieved, the next step is to represent the implementation of the function in terms of the hardware function, software function, and some form of interaction [PUR99]. Unfortunately, this is usually difficult to do because the functionality information to support decomposition is not available.

The classical reliability models that considered only hardware are no longer relevant. Software, its operations, and resultant failures, are at least as important as hardware failures. Interestingly, the author [PUR99] uses user-perceived reliability and availability data rather than data recorded against the hardware and software. This is a useful practice because who else is better to judge whether a system is up or down than the user [WOO95]? While you might like to use this concept, user-perceived reliability and availability data are generally not available for most projects.

Another approach is to provide strong partitioning of applications, which means that the boundaries among applications are well defined and protected so that operations of an application will neither be disrupted nor corrupted by erroneous behavior of another application [RUS99]. Each application is allocated to a single partition, providing computational and memory resources and the means to access devices [ISL06]. Strong partitioning improves the reliability of individual applications and the system as a whole.

Combining Hardware and Software Reliability

In the case of a system with a real-time operational mode, such as the Los Alamos CRAY-1 computer, performance is affected by such factors as interprocess communication, sequence of operations, and processor scheduling policies. On the other hand, the reliability of the system is affected by random hardware and software failures. In the event of the failure of some components, a real-time system must still continue to function, and a subset of its time-critical tasks must meet the deadline [LSL92]. To respond to the continuous operation requirement, one of the hardware–software models that will be explored provides parallel hardware redundancy combined with software components in series. Note that software redundancy is infeasible because the same fault will reside in all copies of the software, but you can mitigate the risk of software failure by testing for a time to assure high reliability. To aid this investigation, use the example hardware and software failure data from the CRAY-1 computer in Table 12.3. These data will be used in the Weibull reliability model in a later section.

System Validation

Validation of computer system reliability during the development of the system is an important activity. The validation process provides: (1) a measure of the ability

Table 12.3 Hardware–Software Failure Data (Los Alamos National Laboratory)

CRAY-1 reporting period	Total failures	Hardware failures	Fraction software failures	Software failures	Hardware failure rate	Software failure rate
t	f	h = f – (s ∗ fs)	fs	s = f ∗ fs	λ	λ
1	40	13.88	0.6529	26.12	0.69	1.31
2	38	37.51	0.0128	0.49	1.88	0.02
3	29	19.66	0.3219	9.34	0.98	0.47
4	23	19.47	0.1534	3.53	0.97	0.18
5	19	5.60	0.7055	13.40	0.28	0.67
6	15	7.90	0.4733	7.10	0.40	0.35
7	17	5.89	0.6533	11.11	0.29	0.56
8	19	10.42	0.4516	8.58	0.52	0.43
9	24	18.67	0.2219	5.33	0.93	0.27
10	25	3.80	0.8479	21.20	0.19	1.06
11	31	19.90	0.3581	11.10	0.99	0.56
12	35	24.80	0.2914	10.20	1.24	0.51
13	33	31.15	0.0561	1.85	1.56	0.09
14	34	23.62	0.3052	10.38	1.18	0.52
15	35	27.22	0.2224	7.78	1.36	0.39
16	33	22.14	0.3290	10.86	1.11	0.54
17	24	1.20	0.9500	22.80	0.06	1.14
18	28	23.07	0.1761	4.93	1.15	0.25
19	27	4.56	0.8313	22.44	0.23	1.12
20	28	1.48	0.9471	26.52	0.07	1.33
21	31	11.89	0.6163	19.11	0.59	0.96
Totals	588	333.85		254.15		

20 days in reporting period.

of a system to detect, locate, and recover from errors; (2) confidence in a system before it is deployed; and (3) feedback during the development stage for improving the design and implementation of a system. Fault injection has been recognized as one of the best approaches for evaluating the behavior and performance of complex systems. There are several advantages in adopting the fault injection approach for evaluating these systems. These advantages include: (1) the effects of faults can be determined when executing programs; (2) the overhead of algorithms that are used to recover from faults can be evaluated; (3) the effects of *additional* faults occurring during the recovery process can be studied; and (4) reliability models can be refined by utilizing data, such as the distribution of faults in the hardware and software [KNA95]. These methods are powerful, but in order to use them, you need access to software code that would allow you to do fault injection.

In real-time systems, hardware and software interact to accomplish a specific task. The presence of both hardware and software causes difficulties in validating real-time systems. A common obstacle is the lack of formal methods (e.g., correctness proofs) that can be used to validate both hardware and software [HSI99]. However, a method that you can apply to both hardware and software is *predictive validity* (e.g., mean squared error [MSE] between actual [historical] and predicted reliability [LYU96]). MSE has the advantageous property that it effectively measures the variance between actual and predicted values and is useful for comparing the prediction accuracy of various reliability models.

Structure of a Software Application

The structure of a software application may be defined as a collection of components comprising the application and the interactions among the components. A component could be a single function, a class, an object, or a collection of these. The interactions among the components may be procedure calls, client–server protocols, links between distributed databases, or synchronous and asynchronous communication among components [GOK05]. These software components are integrated with hardware components to form a unified suite of components that can be subjected to reliability evaluation.

Reliability evaluation is useful and important in designing computer systems, while at the same time it is also difficult. The difficulty becomes significant when the model combines hardware, software, and their interactions, due to the difference in failure behavior between hardware and software [PUR99]. Despite this difficulty, the analysis now shifts to investigate one of the major objectives: the possibility of developing a unified hardware–software reliability model (i.e., a system reliability model not limited to hardware *or* software).

Hardware and Software Failure Relationships

You can consider hardware and software failure relationships to be based on the following reasoning: It is extremely unlikely that hardware and software failures would occur simultaneously. If they did, it would be a coincidence rather than cause and effect. For example, an error in the software that causes the program to take a wrong branch, would not, in itself, result in a hardware failure. Another example is when there is a memory failure and, subsequently, the software "fails" in attempting to access the defective memory. But the failure should be charged to the hardware and not to the software. Now, it is possible for a permanent hardware failure to render the software inoperable [KAN96], but this is not the fault of the software. The failure should be charged against the hardware. The consequence is that the *availability* of the software would be decreased.

Assessing Predictive Validity

In order to assess predictive validity of hardware, software, and system reliability predictions, the corresponding actual (i.e., historical) reliability computations are required in Equations 12.12–12.14, respectively, over the scheduled operating time T. Once these values have been computed, mean relative error, with respect to the corresponding predictions, can be computed:

$$R_{ah} = 1 - \frac{h_t}{\sum_{t=1}^{T} h_t}, \tag{12.12}$$

where h_t is the actual number of hardware failures;

$$R_{as} = 1 - \frac{s_t}{\sum_{t=1}^{T} s_t}, \tag{12.13}$$

where s_t is the actual number of software failures;

$$R_{af} = 1 - \frac{f_t}{\sum_{t=1}^{T} f_t}, \tag{12.14}$$

where f_t is the actual number of system failures.

Weibull Reliability Model

Due to the great variation in both hardware and software failure counts in Table 12.3, a flexible failure function is needed to represent these phenomena. One of the most widely used distributions for reliability is the Weibull failure distribution [SHO83]. It has the flexibility of allowing for constant, increasing, and decreasing failure rate functions. Thus, given the variability in hardware and software failure rates in Table 12.3, it is a good candidate for predicting the reliability of the CRAY-1 computer. The reliability R(t) at operating time t is given in Equation 12.15, where λ is the failure rate and α is the shape parameter (i.e., the parameter that governs the shape of the reliability function) [LLO62]:

$$R(t) = e^{-(\lambda t^{\alpha})}. \tag{12.15}$$

The parameters of the Weibull distribution are estimated according to Lloyd and Lipow [LLO62] in Equations 12.16 and 12.17, where n is the number of failure counts:

$$\lambda = \frac{n}{\sum_{i=1}^{n} t_i^{\alpha}}, \tag{12.16}$$

$$\alpha = \frac{n}{\lambda \sum_{i=1}^{n} t_i^{\alpha} \log t_i - \sum_{i=1}^{n} \log t_i}. \tag{12.17}$$

However, trying to solve Equations 12.16 and 12.17 is not practical because in order to solve for λ in Equation 12.16, α is required, but to solve for α in Equation 12.17, λ is required. A practical approach is to use the reliability function, Equation 12.15, to solve for α, given values of λ and R(t), for a specified value of t. Now, solving Equation 12.15 for α results in Equation 12.18:

$$\alpha = \left[\frac{-\log(R(t))}{\lambda} \right]. \tag{12.18}$$

However, notice the constraint on the maximum value of R(t) that can be achieved to avoid trying to take the log of a negative quantity: $R(t) < e^{-\lambda}$ because $\log R(t) = \lambda$. Therefore, set the limit on R(t) according to $R(t) < e^{-\lambda}$ and substitute this value in Equation 12.18 and solve for α. Since the hardware and software failure rates λ and failure time t (reporting period) are given in Table 12.3, you have all the information needed to estimate the parameter α.

Weibull Model Results

If Equation 12.15 does not yield adequate hardware–software predictive reliability, compared with actual hardware–software reliability, parallel and combined series–parallel reliability models can be brought into play to provide hardware redundancy, thereby increasing both hardware and system reliability. Figure 12.7 shows these concepts, where predicted system reliability is considerably below actual system reliability and there is a large MSE difference between the two reliabilities. Using five hardware components in parallel, with software in series, while significantly reducing the prediction error, does not result in predicted system reliability approximating actual reliability. Therefore, in order to raise reliabilities to desirable levels, hardware, software, and system failure rates must be reduced. This issue will be addressed in the next section.

Solving Equation 12.15 for failure rate $\lambda(t)$, for values of operating time t and mean value of parameter α, allows you to estimate the failure rate required to achieve specified reliability R(t) in Equation 12.19. This estimate is made for hardware, software, and system reliability:

$$\lambda(t) = (-\log(R(t))/(t^{\alpha}). \tag{12.19}$$

Table 12.4 CRAY-1 Failure Rates

	Mean failures per day		
Required reliability	Hardware	Software	System
No requirement	0.794874	0.605126	1.400000
0.8000–0.9900	0.107179	0.106992	0.107185

Applying Equation 12.19, the mean value of failure rates for CRAY-1 hardware, software, and system required to achieve the specified reliability values are tabulated in Table 12.4. Surprisingly, there is negligible difference in failure rates among hardware, software, and system, but significant reductions when compared with the failure rates where no reliability requirement is specified. The reason for this is that when there is a reliability goal, efforts to reduce faults and subsequent failures are focused, such as testing to bring reliability into conformance with the specification. Whether an organization would opt to achieve these reliability levels would depend on the mission reliability requirement and the cost of testing to remove faults to the extent that required failure rate reduction would be achieved. The lack of distinction between hardware and software failure rates may be explained by the fact that being a super computer, the CRAY-1 possesses both complex hardware and software, contributing approximately equally to the generation of failures.

It is also important to estimate the operating time t that could be achieved for a specified reliability R(t) and mean value of parameter α in the Weibull model, by solving Equation 12.15 for t. The result is Equation 12.20:

$$t = \left[\left(\frac{1}{\lambda}\right)(\log R(t))\right]^{\left(\frac{1}{\alpha}\right)}. \tag{12.20}$$

Figure 12.8 shows the results of applying Equation 12.20 to the CRAY-1 computer data, where you can identify the maximum operating times that can achieved at specified values of reliability. The utility of this figure is that it shows the predicted spread, and maximums of operating times, that could be achieved for a computer and its applications.

SUMMARY AND CONCLUSIONS

Using data from several real-world projects, evaluations were conducted with several hardware, software, and system reliability models. The major result is that if the project failure data are significant, no amount of parallelism will salvage a reliability disaster. Faults must be removed and failure rates reduced for the systems to come into conformance with reliability specifications.

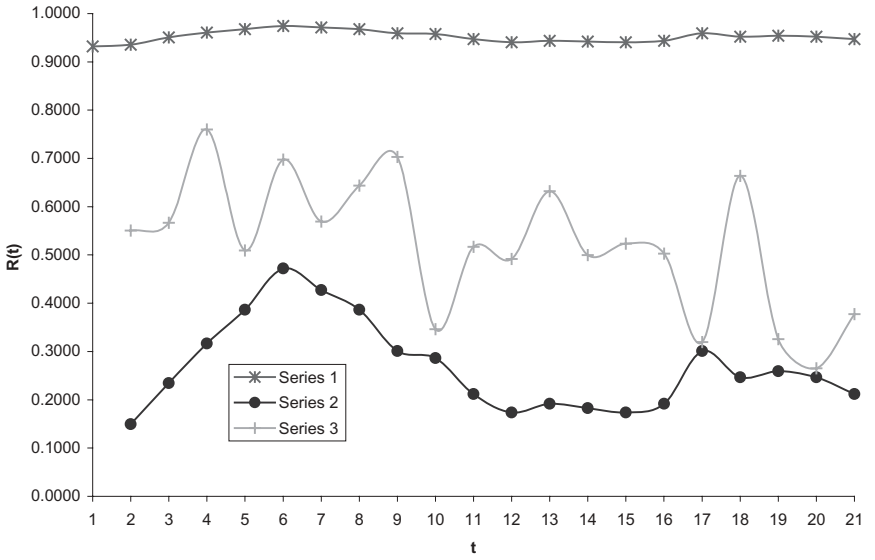


Figure 12.8 CRAY-1 reliability R(t) versus operating time t, using Weibull distribution for predictions. Series 1: Actual system reliability. Series 2: Predicted hardware–software system reliability, no parallel hardware components, MSE = 0.4952. Series 3: Predicted hardware–software system reliability, five parallel hardware components, MSE = 0.0146.

Although not a result, it was noted that only hardware is subject to parallel-based reliability improvement because using software redundancy is no help because the same faults will be repeated in multiple copies of the software.

It was found that the reliability BC, where cost is based on number of configuration components, is a good tool for deciding on a series–parallel configuration that provides both required reliability at a reasonable cost. This metric can be related to operating time so that it is possible to see when one configuration (e.g., pure parallel) becomes superior to another (e.g., series).

The probability of failure metric is useful because it allows you to identify the accumulated number of failures where the probability of additional failures becomes negligible and testing can be terminated.

An important consideration of the user community is how long a system can be operated at specified values of reliability. Using various values of reliability, corresponding failure rates, and solving the reliability equation for operating time, the community can predict the operating times that could be achieved. When both hardware and software failure data are available, this prediction identifies the maximum operating time and corresponding reliability that can be achieved by hardware, software, and system.

It was seen that when component failure data are available so that system reliability can be predicted as a function of number of components and their failure rates, no components can be allowed to fail in order to achieve acceptable reliability.

Even a single component failure would put the system down. The solution to this problem is to use very high reliability software and hardware components, combined with hardware redundancy, in the system design. As the analysis showed, this problem is mitigated by the fact that the probability of multiple component failures at the same operating time is negligible.

In accordance with the major objective of integrating hardware and software into a system model, the Weibull distribution was chosen for this purpose because it has the flexibility of modeling various failure rate patterns. While the Weibull distribution is useful for showing how parallelism can improve system reliability, it did not match *actual reliability* very well; other models may provide better accuracy.

REFERENCES

[BON98] A. Bondavalli, A. Fantechi, D. Latella, and L. Simoncini, "Towards a discipline of system engineering: validation of dependable systems," *Proceedings of Computer Security, Dependability and Assurance: From Needs to Solutions*, 1998. pp. 144–165.

[CAR95] J. Carreira, H. Madeira, and J. G. Silva, "Assessing the effects of communication faults on parallel applications," *Proceedings of the International Computer Performance and Dependability Symposium*, April 24–26, 1995, pp. 214–223.

[CHO02] M. Choi, N. Park, and F. Lombardi, "Hardware-software co-reliability in field reconfigurable multi-processor-memory systems," *Proceedings International of the Parallel and Distributed Processing Symposium*, 2002, pp. 138–151.

[GOK05] S. S. Gokhale and M. R.-T. Lyu, "A simulation approach to structure-based software reliability analysis," *IEEE Transactions on Software Engineering*, 2005, 31(8), pp. 643–656.

[HSI99] P.-A. Hsiung, "Hardware-software coverification of concurrent embedded real-time systems," *Proceedings of the 11th Euromicro Conference on Real-Time Systems*, 1999, pp. 216–223.

[ISL06] S. Islam, R. Lindstrom, and N. Suri, "Dependability driven integration of mixed criticality SW components," *Ninth IEEE International Symposium on Object and Component-Oriented Real-Time Distributed Computing (ISORC'06)*, 2006, pp. 485–495.

[KAN96] K. Kanoun and M. Borrel, "Dependability of fault-tolerant systems—explicit modeling of the interactions between hardware and software components," *2nd International Computer Performance and Dependability Symposium (IPDS '96), IEEE Computer Society*, 1996, pp. 252–261.

[KNA95] N. A. Kanawati, G. A. Kanawati, and J. A. Abraham, "Dependability evaluation using hybrid fault/error injection," *Proceedings of the International Computer Performance and Dependability Symposium*, April, 1995, pp. 224–233.

[LLO62] D. K. Lloyd and M. Lipow, *Reliability: Management, Methods, and Mathematics*. Englewood Cliffs, NJ: Prentice-Hall, Inc., 1962.

[LSL92] S. M. R. Islam and H. H. Ammar, "Performability of integrated SW-HW components of real-time parallel and distributed systems," *IEEE Transactions on Reliability*, 1992, 41(3), pp. 352–362.

[LYU96] M. R. Lyu, (ed.), *Handbook of Software Reliability Engineering*. Los Alamitos, CA: IEEE Computer Society Press; New York: McGraw-Hill Book Company, 1996.

[MIC05] S. E. Michalak, K. W. Harris, N. W. Hengartner, B. E. Takala, and S. A. Wender, "Predicting the number of fatal soft errors in Los Alamos National Laboratory's ASC Q supercomputer," *IEEE Transactions on Device and Materials Reliability*, 2005, 5(3), p. 329.

[MUS87] J. D. Musa, A. Iannino, and K. Okumoto, *Software Reliability: Measurement, Prediction, Application*. New York: McGraw-Hill, 1987.

[PUR99] Y. Purwantoro and S. Bennett, "Decomposition technique for integrated dependability evaluation of hardware-software systems using stochastic activity networks," in *25th Euromicro Conference (EUROMICRO '99)*-Volume 2, 1999, p. 2142.

[RUS99] J. Rushby, "Partitioning in Avionics Architectures: Requirements, Mechanisms, and Assurance," NASA/CR-1999-209347, SRI International, Menlo Park, California, 1999.

[SHO83] M. L. Shooman, *Software Engineering: Design, Reliability, and Management*. New York: McGraw-Hill, 1983.

[WOO95] A. P. Wood, "An analysis of client/server outage data," *Proceedings of the International Computer Performance and Dependability Symposium*, April 24–26, 1995, pp. 295–304.

Part Five

Applications

Chapter 13

Applying Neural Networks to Software Reliability Assessment

While you have studied many reliability concepts—both software and hardware—in Chapters 8, 11, and 12, this material was based on traditional models. In this chapter, new models are studied based on concepts from the field of neural networks that are used to assess the reliability of software, employing cumulative failures, reliability, remaining failures, and time to failure metrics. In addition, the risk of not achieving reliability, remaining failures, and time to failure goals are assessed. The purpose of the assessment is to compare a criterion, derived from a neural network model, for estimating the parameters of software reliability metrics, with the method of maximum likelihood estimation. The neural network method proved superior for all the reliability metrics that were assessed by virtue of yielding lower prediction error and risk. Considerable adaptation of the neural network model was necessary to be meaningful for the software reliability assessment application—only inputs, functions, neurons, weights, activation units, and outputs were required to characterize this application.

INTRODUCTION

Neural networks have attracted a great deal of attention from researchers because they have many advantages over other models. For example, they have the ability to learn. Given sample data, a neural network can learn rules from these sample data with or without a teacher. They have the capability to adapt weights to changes in the surrounding environment. That is, a neural network trained to operate in a specific environment can be retrained to deal with minor change in the operating environmental conditions [WON08].

Computer, Network, Software, and Hardware Engineering with Applications, First Edition. Norman F. Schneidewind.
© 2012 the Institute of Electrical and Electronics Engineers, Inc. Published 2012 by John Wiley & Sons, Inc.

Neural networks have been successfully applied to many fields, such as pattern recognition [FUK98], system identification [CHU90], and intelligent control [NAR92]. Software engineering areas including risk analysis [NEU02], cost estimation [TAD05], reliability estimation [KAR96], and reusability characterization [BOB93]. However, they have not been applied as extensively to help programmers find bugs [WON07, WON08]. Since neural networks operate on the principle of learning, no model is specified a priori [KAR96], meaning the models are evolved by learning.

Neural networks are comprised of the following components [KAR96]:

- ***Models of Neurons.*** Characteristics of the processing units used in neural networks
- ***Models of Interconnection Structure.*** Topology of the network and strength of interconnections that encode network knowledge
- ***Learning Algorithm.*** Steps involved in computing or assigning neural connection weights in the network

Biological neurons are single cells capable of crude computation. Neurons are stimulated by one or more inputs and generate outputs that are sent to other neurons. Outputs are dependent on the strength of inputs and the nature of input connections. Some connections excite neurons and increase output; others inhibit neuron output [MAS93]. Neurons are connected together with weighted connections following a specified structure. Each neuron has an activation function that describes the relationship between its input and output [WON08]. Neural network learning is normally accomplished through an adaptive procedure, known as a learning algorithm [WON08]. The architecture of a generic neural network is shown in Figure 13.1.

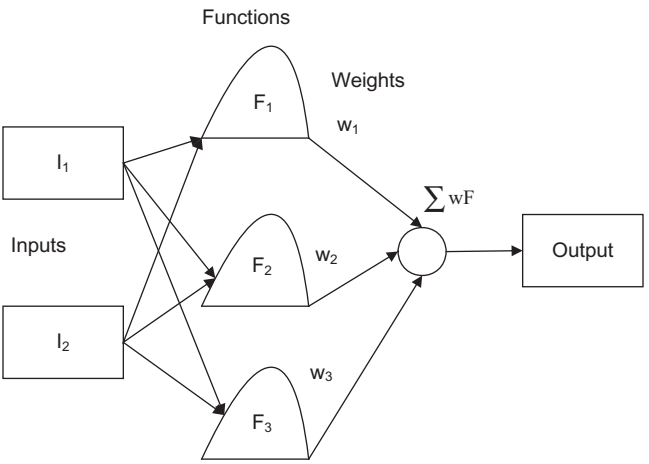


Figure 13.1 Generic neural network.

Back Propagation Algorithm

Back propagation (BP), or propagation of error, is a common method of teaching neural networks how to perform a given task. It is similar to feedback in a control system that adjusts the input to achieve the desired output. The algorithm can calculate the desired output for any given input. An important application of BP in neural networks is fault localization s[LEE99, WAS93]. It is a learning method, and is an implementation of the gradient descent (ascent) learning rule. Gradient descent (ascent) refers to computing the rate of change of a function to find where the rate of change is minimum. For example, finding the time of testing software where the rate of change between reliability and test time is a minimum, thereby achieving a balance between improving reliability, by localizing faults, and the cost of testing. Once this rule is learned for one software system, it can be applied to subsequent software systems.

NEURAL NETWORKS APPLIED TO FAULT LOCALIZATION

Fault localization is the most expensive activity in program debugging. Traditional ad hoc methods can be time consuming and ineffective because they rely on programmers' intuitive guesswork, which may be neither accurate nor reliable. A better solution is to utilize a systematic and statistically well-defined method to automatically identify code that should be examined for possible fault locations. A statistical method can be used to identify the coverage of each executable statement and the execution result (success or failure) for each test case. A record is constructed for each executable statement and a statistic is computed to determine the likelihood of the corresponding statement containing bugs. Statements with a higher likelihood of bugs are more likely to contain bugs and should be examined before those with a lower likelihood [WON07].

A typical neural network has a feed-forward structure that can be trained to learn the input–output relationship from a set of data. For example, the input is the program statement coverage of a test case and the output is the corresponding statement execution result (success or failure). After the network is trained, a test case with only one statement covered is used as an input to compute the likelihood of the corresponding statement containing bugs. The larger the output, the greater the likelihood of statement bugs. Statements are then ranked in descending order based on their likelihood of containing bugs. Programmers examine these statements from the top of the rank, one by one, until the first statement containing the bugs is identified.

In fault localization, the output of a given input can be defined as a binary value of 0 or 1, where 1 represents a program failure on this input and 0 represents a successful execution. With this definition, the output of each input is known because you know exactly whether the corresponding program execution fails or succeeds. Moreover, two similar inputs can produce different outputs because the program

execution may fail on one input but succeed on another input. Thus, learning algorithms that cannot adapt to the environment are inappropriate for fault localization. Therefore, neural networks using *adaptable* learning algorithms are better candidates for solving the fault localization problem.

NEURAL NETWORKS APPLIED TO SOFTWARE RELIABILITY ASSESSMENT

Another approach to software reliability improvement, in addition to fault localization, is to adapt neural network concepts to reliability prediction. The idea is to use gradient descent or ascent, depending on the nature of the activation function in Figure 13.2 (i.e., relationship between inputs and outputs). In effect, the network is trained to use the gradient method to identify the test time when the marginal reduction in failures and faults (benefit) is just balanced by the marginal increase in test time (cost).

Cumulative Failures

Software reliability, as measured by cumulative failures during testing, is illustrated in Figure 13.2. The idea is to embody the neuron with the processing power to aggregate the weighted failure counts x_i in the test time intervals i, such that the

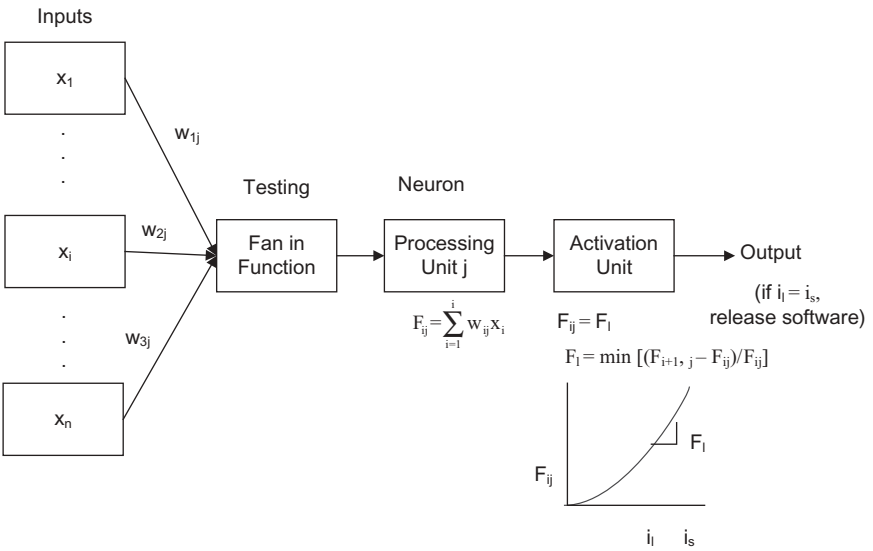


Figure 13.2 Neural network process function. x_i, number of failures in interval i; w_{ij}, severity of failures in interval i for software system j; F_l, cumulative failures limit; F_{ij}, cumulative failures Activation Function for test interval i and software system j; i_s, scheduled test time; i_l, test time at F_l.

actual cumulative failures for a software system j, F_{ij}, is compared with the limit value, F_l, where the limit value is equal to the slope of the curve in Figure 13.2 at test time i_l. If $F_{ij} \leq F_l$, the software is released because at this test time i, marginal benefit equals marginal cost; otherwise, it is subjected to further testing. Equation 13.1 shows the computation of cumulative failures and Equation 13.2 shows the computation of weights, using the failure severity code:

$$F_{ij} = \sum_{i=1}^{i} w_{ij} x_i, \tag{13.1}$$

$$w_{ij} = 1 - \frac{s_{ij}}{s_m}, \tag{13.2}$$

where s_{ij} is the severity code of x_i for software system j and s_m is the maximum value of the severity code (minimum severity). The limit F_l is computed in Equation 13.3, where the limit is the minimum rate of change over successive test intervals i. The value of i corresponding to F_l is the amount of test time required to achieve the reliability objective. If this value of i, i_l, is less than or equal to the schedule test time i_s, release the software system; otherwise, continue testing. Note that in order for this policy to make sense, the faults causing the failures that have been detected must be corrected:

$$F_l = \min[F_{i+j,j} - F_{ij}]. \tag{13.3}$$

In order to test the validity of Equation 13.3 as a criterion of a benefit–cost limit for cumulative failures, the equation for *predicted* cumulative failures is needed in order to see whether F_l is capable of identifying the amount of test time that should be used to accurately estimate the parameters of the prediction model. The predicted cumulative failures will be compared with the actual cumulative failures (unweighted) in Figure 13.3. The prediction equation from the Schneidewind software reliability model (SSRM) [SCH97] for test interval i is shown in Equation 13.4:

$$F(i) = (\alpha/\beta)\left[1 - e^{-\beta(i-s+1)}\right] + X_{s-1}, \tag{13.4}$$

where α and β are failure rate parameters, s governs how much failure data are used in parameter estimation, and X_{s-1} is the observed failure data in the range (s–1), i.

Figure 13.3 shows how the neural network criterion limit of Figure 13.2 and Equation 13.3 can be applied to identify the test interval i that is optimal for terminating testing and releasing the software system. This is the test interval when the rate of change of actual cumulative failures is minimum. In other words, this is the point in test time when diminishing returns in finding and correcting faults has been reached. The results of an experiment to test the validity of the neural network criterion limit are shown in Figure 13.4. The experiment was conducted by predicting cumulative failures for a National Aeronautics and Space Administration (NASA) Space Shuttle software system j = OI6, using SSRM. This model has a parameter s that identifies the first interval of test failure data that is used in estimating model

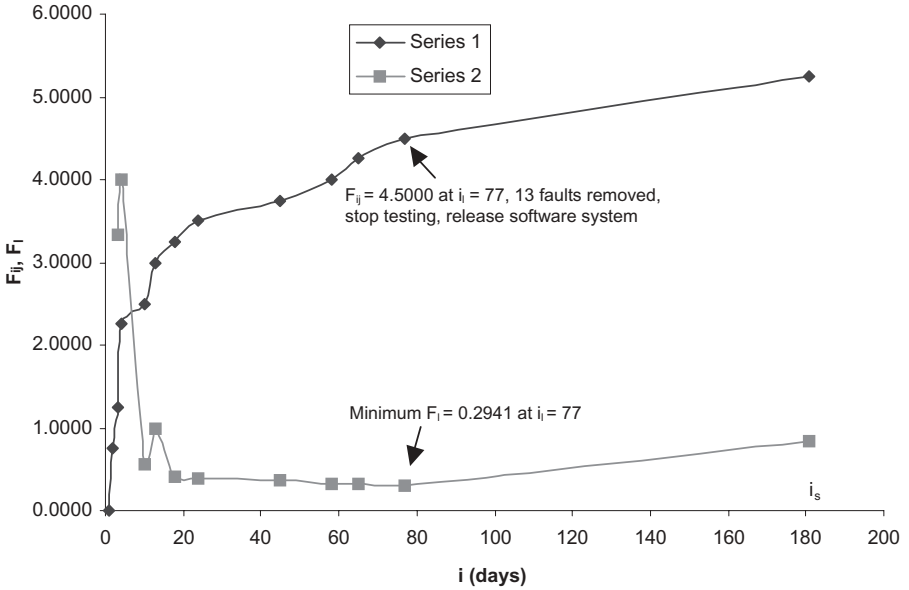


Figure 13.3 NASA space shuttle OI6: actual cumulative failures F_{ij} and criterion limit ($F_l \times 5$) versus test interval i. Series 1: F_{ij}. Series 2: F_l.

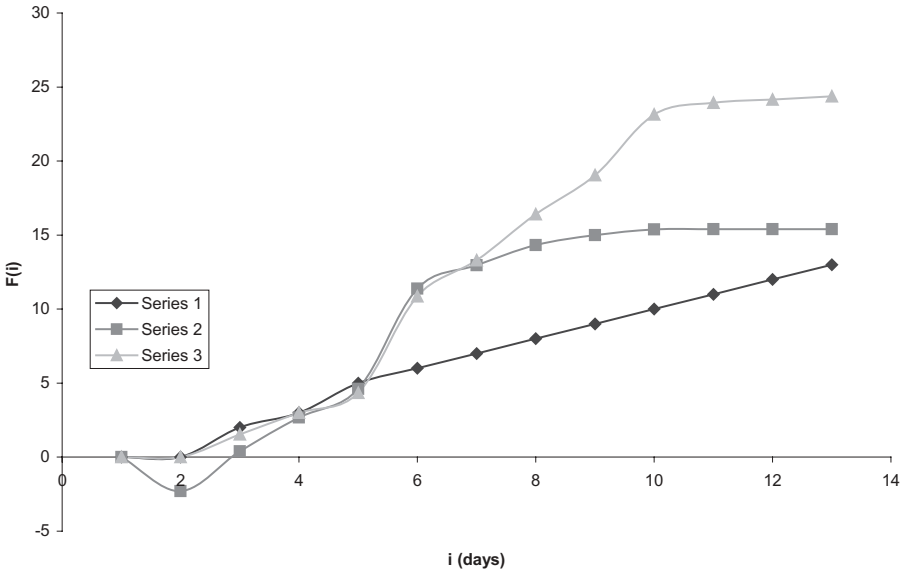


Figure 13.4 NASA Space Shuttle OI6: cumulative failure F(i) versus test interval i. Series 1: Actual F(i). Series 2: Predicted F(i), using neural network criterion for s = 12, MSE = 16.6306. Series 3: Predicted F(i), using parameter evaluation method for s = 2, MSE = 74.1375. s, first test interval of failure data used in predicted F(i) parameter estimation.

parameters. Two criteria were used in selecting s: one is based on the neural network criterion limit identified in Figure 13.3 as i = 77 that corresponds to s = 13 (the 13th failure count interval); the second is based on the maximum likelihood estimation (MLE) method of parameter estimation [SCH07] that yielded s = 2. The mean squared error between the actual and predicted cumulative failures was computed for the two methods. As Figure 13.4 demonstrates, the neural network criterion limit provides much better prediction accuracy.

Reliability

A second validity test was conducted by experimenting with the reliability activation function in test interval i, R_i (i.e., R_i output produced when input failure counts x_i occur). Unlike the case of cumulative failures, using weights does not apply because reliability is not an additive function. Start by computing the actual reliability and its reliability limit criterion, R_l, in Equations 13.5 and 13.6, respectively:

$$R_i = 1 - \left(\frac{x_i}{\sum_{i=1}^{i} x_i} \right), \tag{13.5}$$

$$R_l = \min[(R_{i+1} - R_i) / R_i]. \tag{13.6}$$

As in the case of cumulative failures, the equation for predicted reliability is needed in order to compare it with actual reliability from Equation 13.5, and to ascertain whether Equation 13.6 provides an effective criterion for identifying the optimal amount of test time. Predicted reliability, as obtained from SSRM, is shown in Equation 13.7, where the parameters have been defined previously.

In Figure 13.5 you see that the reliability criterion limit R_l from Equation 13.6 is associated with the maximum actual reliability R_i at a test time i_l equal to the total scheduled test time i_s. This test time corresponds to the reliability parameter s = 13 that will be used in subsequent reliability evaluations.

The superiority of the neural network reliability criterion limit in the early stages of testing is demonstrated in Figure 13.6, where this method produces a prediction lower error, with respect to actual reliability, than in the case of the parameter evaluation method. However, the latter method does have an advantage in yielding higher reliability in the later stages of testing. Thus, in choosing reliability prediction models, it would be prudent to evaluate more than one model because a given model may not be superior for all test times.

Another important formulation of reliability is shown in Equation 13.8, where the concept is to predict reliability at the end of the mission duration, t_m. This is done by predicting reliability for the test time i plus the mission duration $(i + t_m)$, assuming the system becomes operational immediately after the completion of test time i. The concept is to subject the system to increasing values of mission duration

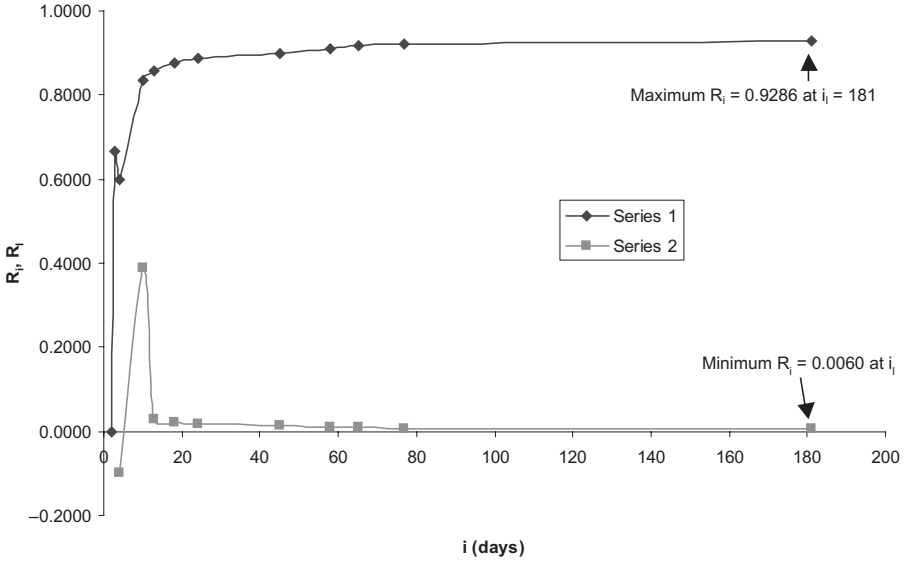


Figure 13.5 NASA space shuttle OI6: actual reliability R_i (Series 1) and reliability criterion limit R_I (Series 2) versus text time i.

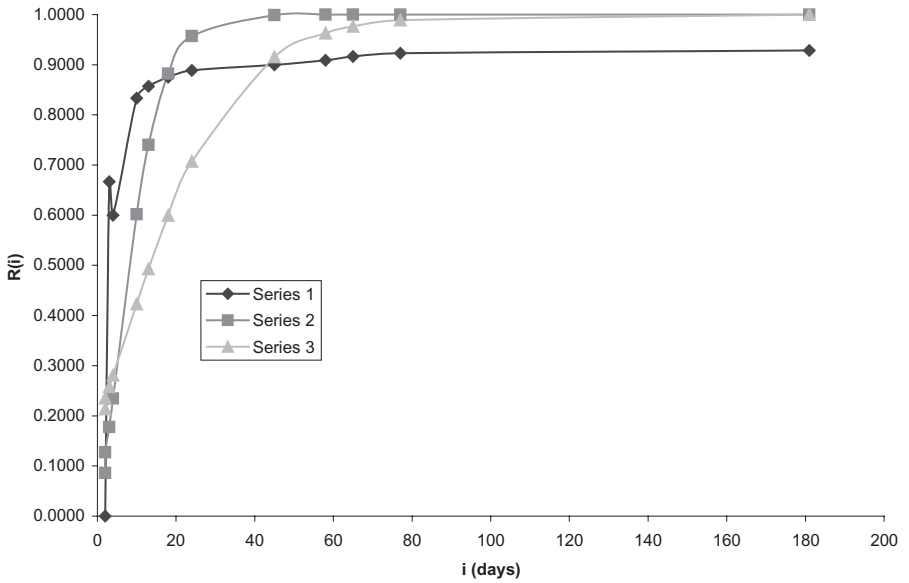


Figure 13.6 NASA space shuttle OI6: reliability R(i) versus test time i. Series 1: Actual R(i). Series 2: Predicted R(i), using neural network criterion for s = 13, MSE = 0.0414. Series 3: Predicted R(i), using parameter evaluation method for s = 2, MSE = 0.0626. s, first test interval of failure data used in predicted R(i) parameter estimation.

in order to identify the maximum mission duration (i.e., mission duration where predicted reliability no longer achieves specified reliability):

$$R(i) = e^{-\left[\frac{\alpha}{\beta}\left[e^{-\beta(i-s+1)}-e^{-\beta(i-s+2)}\right]\right]}, \tag{13.7}$$

$$R(i+t_m) = e^{-\left[\frac{\alpha}{\beta}\left[e^{-\beta(i+t_m-s+1)}-e^{-\beta(i+t_m-s+2)}\right]\right]}. \tag{13.8}$$

Reliability Risk

Risk is a major issue in software reliability assessment because there is a probability (i.e., risk) that the predicted reliability of a software system, as given by Equation 13.8, will not achieve specified reliability, R, at the end of the mission. Thus, reliability risk, RR, is computed in Equation 13.9:

$$RR = (R - R(i+t_m))/R = 1 - (R(i+t_m))/R, \tag{13.9}$$

where R is specified reliability. The greater the relative difference between specified and predicted reliabilities in Equation 13.9, the greater the risk. The best result is when RR goes negative (i.e., predicted reliability > specified reliability). Figure 13.7 demonstrates that the neural network criterion method involves lower reliability risk

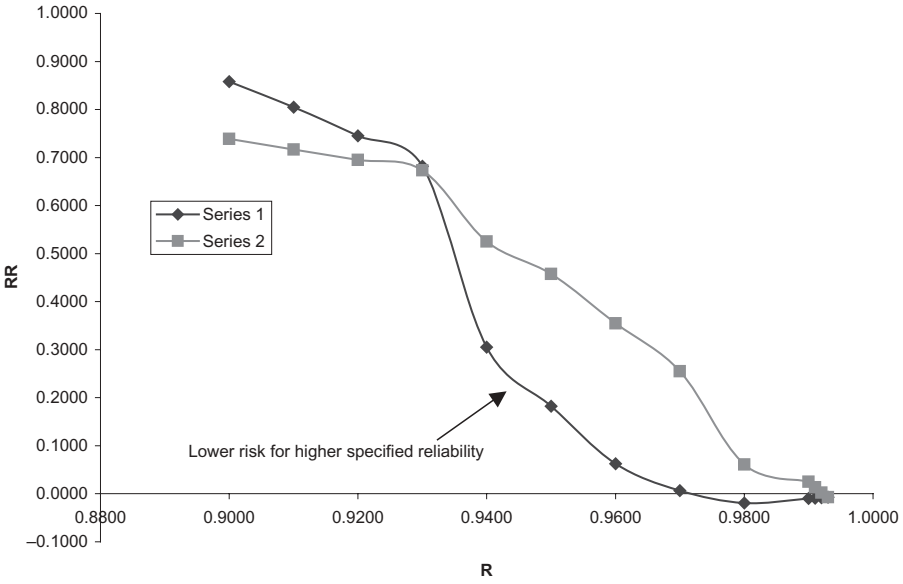


Figure 13.7 NASA space shuttle OI6: reliability risk RR versus specified reliability R. Series1: RR, using neural network criterion for s = 13. Series 2: RR, using parameter evaluation method for s = 2. s, first test interval of failure data used in RR parameter estimation.

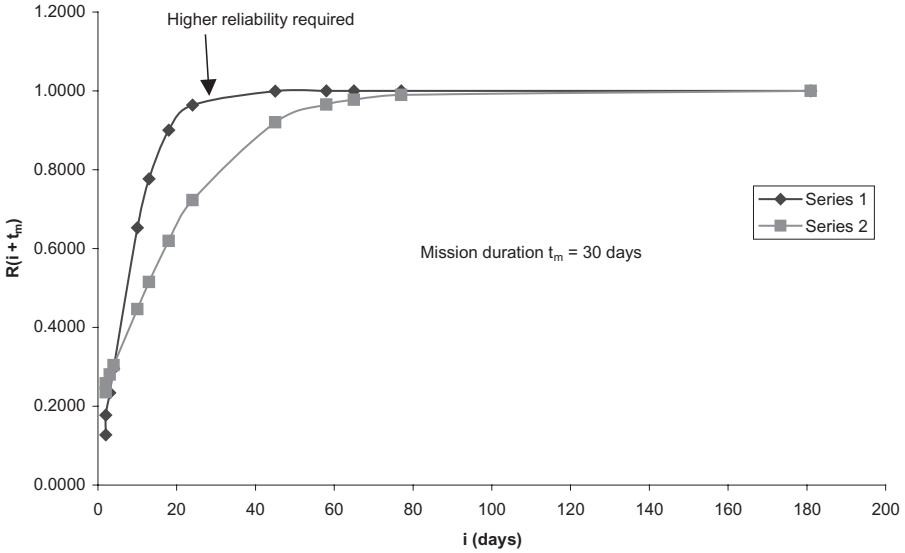


Figure 13.8 NASA space shuttle OI6: reliability required to achieve mission duration $R(i + t_m)$ versus test time i. Series 1: $R(i + t_m)$, using neural network criterion for s = 13. Series 2: $R(i + t_m)$, using parameter evaluation method for s = 2. s, first test interval of failure data used in predicted $R(i + t_m)$ parameter estimation.

at higher values of reliability. This is important in a mission-critical application, such as the Shuttle flight software, which requires high reliability at low risk.

It is also important to compare the neural network and parameter evaluation methods with respect to the reliability required to achieve the mission duration. Figure 13.8 provides an interesting contrast between the methods because although the required reliability produced by the parameter evaluation method is less, for the given mission duration, this would not be desirable for a mission-critical application where the reliability must be high. Thus, it is important to evaluate such results in the context of the application: for a commercial application, where the cost of achieving reliability is critical, the parameter evaluation method would be the choice, but not in a mission-critical application.

It is also of interest to predict the test time i_R required to achieve specified reliability R. This quantity is predicted in Equation 13.10 by solving Equation 13.7 for i, where R(i) becomes the specified reliability R:

$$i_R = [(-1/\beta)\log[(-\beta/\alpha)\log R/(1-\exp(-\beta))]] + \beta(s-1). \qquad (13.10)$$

Figure 13.9 vividly shows that the neural network criterion is superior because its use requires significantly less test time to achieve specified reliability. Thus, on balance, considering the software reliability results shown in Figures 13.5–13.9, the neural network criterion is the better choice, particularly for the mission-critical application that has been evaluated.

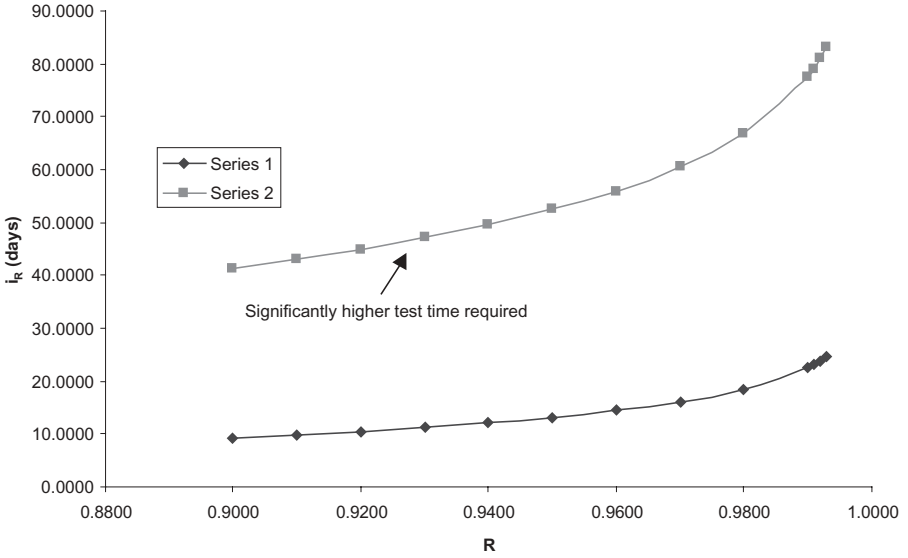


Figure 13.9 NASA space shuttle OI6: test time required to achieve specified reliability i_R versus specified reliability R. Series 1: i_R, using neural network criterion for s = 13. Series 2: i_R, using parameter evaluation method for s = 2. s, first test interval of failure data used in i_R parameter estimation.

Remaining Failures

Another important reliability metric is remaining failures. The reason for the importance of this metric is that remaining failures represent residual problems buried in the software code that could emerge when least expected—during operation. Thus, it behooves us to include this metric in our arsenal of software reliability tools. Note that remaining failures, expressed in Equation 13.11 (SSRM), is a predicted quantity because you have no way of knowing the *actual* number of remaining failures. But this begs the question of how to evaluate the neural network and parameter estimation methods that were employed previously by comparing the method prediction errors, using actual failure values. The solution is to approximate remaining failures by using the *known* remaining failures, as shown in Equation 13.12, where X_s is the total number of failures reported at the scheduled test time interval i_s and x_i is the number of failures in test interval i. Since these are failure counts, it is appropriate to weigh remaining failures in Equation 13.12:

$$r(i) = \frac{\alpha}{\beta}[\exp(-\beta(i-(s-1)))], \tag{13.11}$$

$$r_{ij} = w_{ij}\left(X_S - \sum_{i=1}^{i} x_i\right). \tag{13.12}$$

Then the neural network criterion r_l can be computed by the usual process, with the proviso that since remaining failures is a decreasing function, as opposed to the increasing functions of cumulative failures and reliability, Equation 13.13 has been formulated appropriately:

$$r_l = \min[(r_{i,j} - r_{i+1,j})/r_{i+1,j}]. \tag{13.13}$$

In addition, as in the case of reliability, predict the reliability risk using Equation 13.14 (SSRM):

$$rr(i) = 1 - (r(i)/r_c), \tag{13.14}$$

where r_c is a specified number of remaining failures. Values of $r(i) < r_c$ will render rr(i) positive, and, hence, yield decreasing risk.

Figure 13.10 again demonstrates the superiority of the neural network criterion for parameter evaluation by producing a significantly lower prediction error with respect to the actual remaining failures. More evidence of this result is afforded by Figure 13.11 that shows, for a specified remaining failures $r_c = 1$, that the risk is lower (i.e., more positive) for the neural network criterion. Furthermore, by using this criterion, the risk trends positive much earlier in test time.

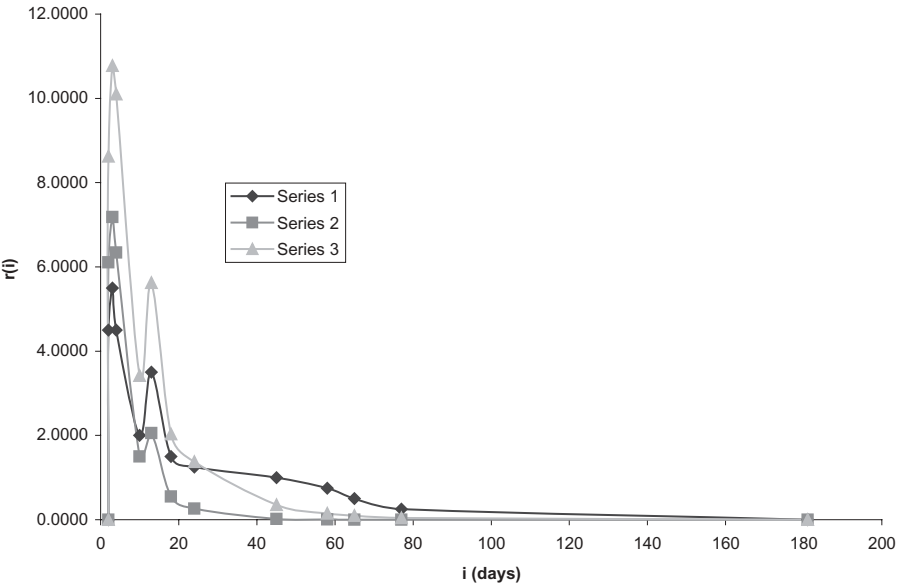


Figure 13.10 NASA space shuttle OI6: remaining failures r(i) versus test time i. Series 1: actual r(i). Series 2: predicted r(i), using neural network criterion for s = 8, MSE = 1.1423. Series 3: predicted r(i), using parameter evaluation method for s = 2, MSE = 4.4606.

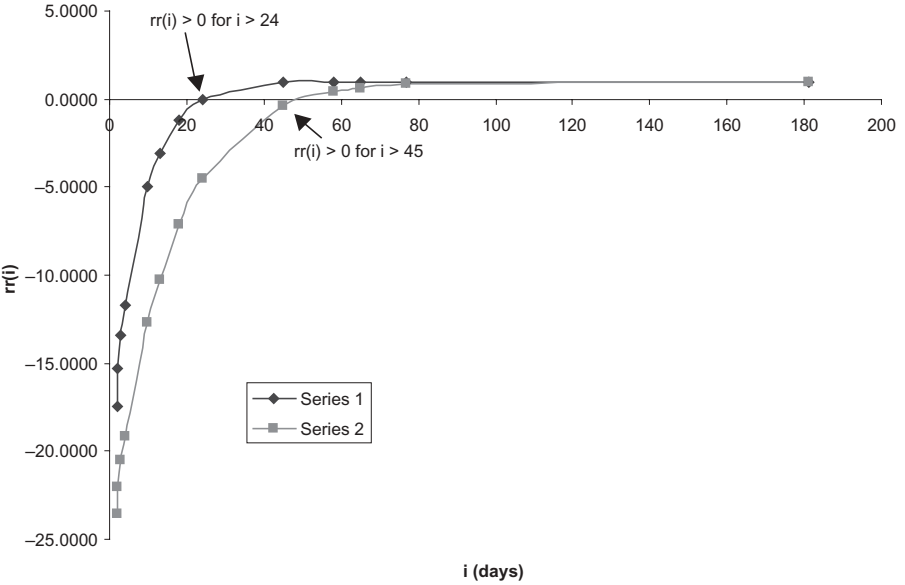


Figure 13.11 NASA space shuttle OI6: remaining failures risk rr(i), for $r_c = 1$, versus test time i. Series 1: rr(i), using neural network criterion for s = 8. Series 2: rr(i), using parameter evaluation method for s = 2.

Time to Next Failure

Time to next failure is also an important software reliability metric because if the predicted value is less than the mission duration, it could be disastrous for the mission. Therefore, preceding as before, first find the neural network criterion t_l by using the actual time to failure, t_i, as illustrated in Equation 13.15. The prediction metric is shown in Equation 13.16, using SSRM as the source:

$$t_l = \min[(t_i - t_{i+1}) / t_i], \tag{13.15}$$

$$T(i) = \log\left[\frac{\alpha}{(\alpha - \beta(F + X_{s,i}))/\beta}\right] - (i - s + 1) \text{ for } \alpha > \beta(F + X_{s,i}), \tag{13.16}$$

where F is the specified number of failures (usually one) to use in predictions and $X_{s,i}$ is the observed failure count in the range s, i, and i is the failure count interval when the prediction is made.

As in the case of remaining failures, there is a risk associated with the time to failure metric because, as mentioned earlier, a prediction less than the mission duration poses a risk. This relationship is expressed in Equation 13.17, where $T(i) < t_m$ represents risk in the risk criterion metric RCM T(i). When $T(i) \geq t_m$, the risk function in Equation 13.17 is negative (i.e., favorable):

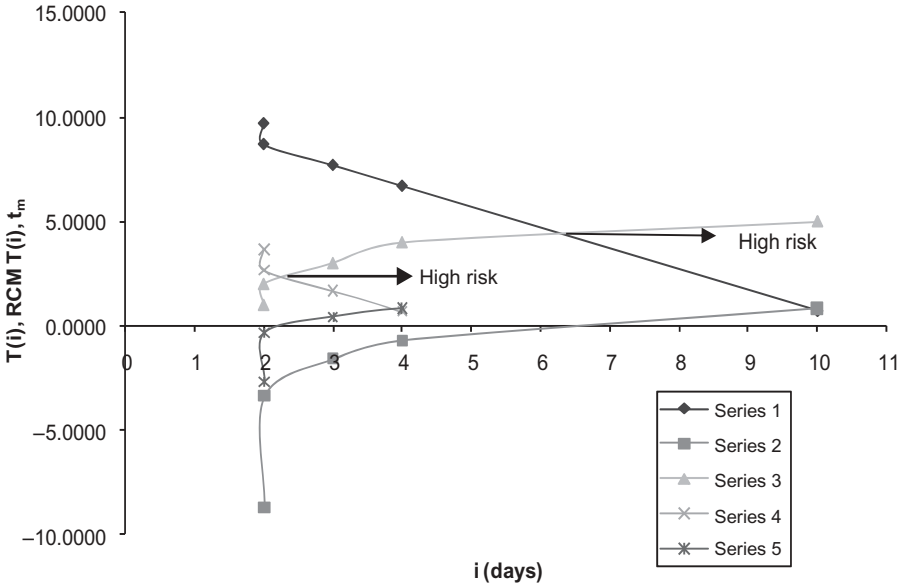


Figure 13.12 Predicted time to failure T(i), risk criterion metric RCM T(i), and mission duration t_m versus time i. Series 1: T(i), using neural network criterion for s = 8. Series 2: RCM T(i), using neural network criterion for s = 8. Series 3: t_m. Series 4: T(i), using parameter evaluation method for s = 2. Series 5: RCM T(i), using parameter evaluation method for s = 2.

$$\text{RCM T(i)} = \frac{t_m - T(i)}{t_m} = 1 - \frac{T(i)}{t_m}. \tag{13.17}$$

Figure 13.12 shows that the safer (i.e., lower risk) alternative is the one produced by the neural network criterion: the RCM is more negative and this metric goes positive at a longer time. The implication is that the software system could be operated safely for a longer time, using the neural network criterion.

Mean Time to Failure

The mean time to failure (MTTF) is the *expected* value of *predicted time to failure* and is valuable for characterizing *time to failure* across various time intervals i. It can be conveniently predicted in Equation 13.18 by using the interval i, and then calling upon the *predicted cumulative failures* F(i) from Equation 13.4:

$$\text{MTTF} = i / F(i). \tag{13.18}$$

Further evidence of the superiority of neural network criterion is provided by Figure 13.13, wherein MTTF is higher for this method. The importance of this result is that MTTF is well understood in the software industry and is typically used to characterize the reliability of software systems [MUS87].

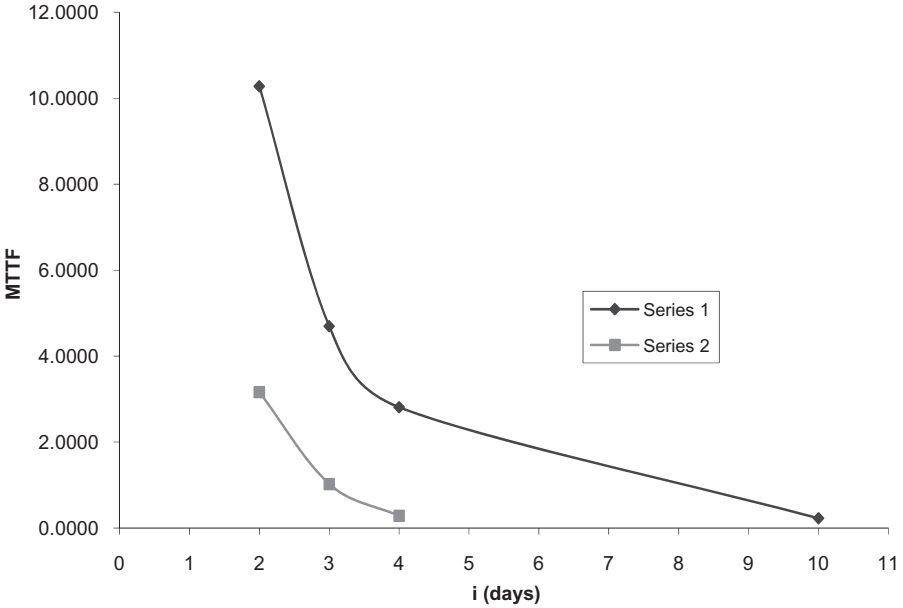


Figure 13.13 NASA space shuttle OI6: Predicted mean time to failure MTTF versus time i. Series 1: MTTF, using neural network criterion for s = 8. Series 2: MTTF, using parameter evaluation method for s = 2.

Problem for Solution by Reader: Using Equations 13.11 and 13.14 for predicted remaining failures and remaining failures risk, respectively, compute the risk for specified remaining failures $r_c = 2$. Then plot the following four risk curves, with respect to test time i, which has been used on previous plots, on one figure:

risk for $r_c = 1$, using neural network criterion with $\alpha = 1.0895$ and $\beta = 0.1250$

risk for $r_c = 1$, using parameter evaluation method with $\alpha = 1.5953$ and $\beta = 0.0650$

risk for $r_c = 2$, using neural network criterion with $\alpha = 1.0895$ and $\beta = 0.1250$

risk for $r_c = 2$, using parameter evaluation method with $\alpha = 1.5953$ and $\beta = 0.0650$

Interpret the results: compare the four curves and indicate which factors lead to the greatest risk.

Solution: Figure 13.14 shows the solution with the greatest risk factors indicated. The neural network criterion leads to the lowest risk because the risk function is more positive, and becomes more positive sooner, than using the parameter evaluation criterion.

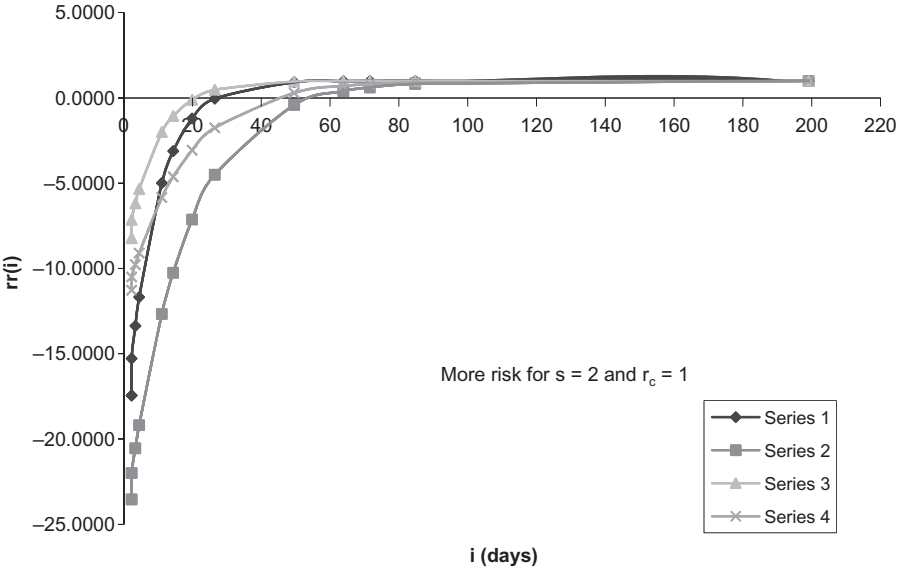


Figure 13.14 NASA space shuttle OI6: remaining failure risk rr(i) versus test time i. Series 1: $r_c = 1$, using neural network criterion for s = 8. Series 2: $r_c = 1$, using parameter evaluation method for s = 2. Series 3: $r_c = 2$, using neural network criterion for s = 8. Series 4: $r_c = 2$, using parameter evaluation method for s = 2.

SUMMARY

The basic concepts of neural networks, as exhibited in Figures 13.1 and 13.2, proved helpful in formulating the software reliability assessment problem. However, some properties of neural networks, such as networks learning from a teacher and others [MAS93], proved to be obscure and of little practical value for the reliability problem that was analyzed. On the positive side, a surprising and enlightening result is that for all software reliability prediction metrics, the neural network prediction criterion was superior to the traditional reliability model parameter estimation method.

REFERENCES

[BOB93] G. Boetticher and D. Eichmann, "A neural network paradigm for characterizing reusable software," *Proceedings of the 1st Australian Conference on Software Metrics*, Sydney, Australia, 1993, pp. 41–54.

[CHU90] S. R. Chu, R. Shoureshi, and M. Tenorio, "Neural networks for system identification," *IEEE Control Systems Magazine*, 1990, 10(3), pp. 31–35.

[FUK98] K. Fukushima, "A neural network for visual pattern recognition," *Computer*, 1998, 21(3), pp. 65–75.

[KAR96] N. Karunanithi and Y. K. Malaiya, "Neural networks for software reliability engineering," in M. R. Lyu (ed.), *Handbook of Software Reliability Engineering*. Los Alamitos, CA: IEEE Computer Society Press; New York: McGraw-Hill Book Company, 1996.

[LEE99] C. C. Lee, P. C. Chung, J. R. Tsai, and C. I. Chang, "Robust radial basis function neural networks," *IEEE Transactions on Systems, Man, and Cybernetics: Part B Cybernetics*, 1999, 29(6), pp. 674–685.

[MAS93] T. Masters, *Practical Neural Network Recipes in C++*. San Diego, CA: Academic Press, 1993.

[MUS87] J. D. Musa, A. Iannino, and K. Okumoto, *Software Reliability: Measurement, Prediction, Application*. New York: McGraw-Hill, 1987.

[NAR92] K. S. Narendra and S. Mukhopadhyay, "Intelligent control using neural networks," *IEEE Control System Magazine*, 1992, 12(2), pp. 11–18.

[NEU02] D. E. Neumann, "An enhanced neural network technique for software risk analysis," *IEEE Transactions on Software Engineering*, 2002, 28(9), pp. 904–912.

[SCH07] N. F. Schneidewind, "Predicting shuttle software reliability with parameter evaluation," *Innovations in Systems and Software Engineering*, 2007.

[SCH97] N. F. Schneidewind, "Reliability modeling for safety critical software," *IEEE Transactions on Reliability*, 1997, 46(1), pp. 88–98.

[TAD05] N. Tadayon, "Neural network approach for software cost estimation," *Proceedings of the International Conference on Information Technology: Coding and Computing*, Las Vegas, Nevada, April, 2005, pp. 815–818.

[WAS93] P. D. Wasserman, *Advanced Methods in Neural Computing*. New York: Van Nostrand-Reinhold, 1993.

[WON07] W. E. Wong, L. Zhao, and Y. Qi, "Fault localization using BP neural networks," *Proceedings of SEKE*, 2007.

[WON08] W. E. Wong; S. Yan; Y. Qi; and R. Golden, "Using an RBF neural network to locate program bugs," *19th International Symposium on Software Reliability Engineering*, November 10–14, 2008, pp. 27–36.

Chapter 14

Web Site Design

Given the importance of Web systems in contemporary society, it behooves us to contribute to improving their reliability. This chapter is just such a contribution. Much valuable research on Web systems focuses on performance evaluation, failing to recognize that, in addition, reliability should be considered. For example, if Web client-to-Web server access time is short, while the system is up, the performance loses meaning if there is considerable downtime. You can model the reliability of Web systems from the bottom up by developing component reliability prediction equations for Web server, Web client, and the communication channels that interconnect them. Then, the component models are integrated to produce total system reliability models. Support your modeling efforts with real-world failure data. The prediction equations identify weak spots in component and system reliability that assist organizations in identifying corrective actions, such as fault removal, in order to achieve reliability goals.

INTRODUCTION

Background

The paradigm of Web services has been gathering significant momentum in both academia and industry in recent years. This paradigm transforms the Internet from a repository of data into a repository of services. Simply put, a Web service is a programmable Web application that is universally accessible through standard Internet protocols [FER03]. Web services opens a new cost-effective way of engineering systems to quickly develop and deploy Web applications by dynamically integrating other independently published Web services [HOL02]. However, it is not clear that this new model of Web services provides any measurable increase in reliability [PAR90]. Thus, this is a motivation for this chapter to show the reader how the reliability of Web services could be improved.

The essential feature of *dynamically* configured Web services poses new challenges for Web system reliability. In a traditional system, all of its components and

Computer, Network, Software, and Hardware Engineering with Applications, First Edition. Norman F. Schneidewind.
© 2012 the Institute of Electrical and Electronics Engineers, Inc. Published 2012 by John Wiley & Sons, Inc.

their relationships are decided before the system runs. Therefore, each component can be thoroughly tested, and the interactions among the components can be fully examined, before the system starts to execute. Web services dynamically locate and assemble distributed Web services in an Internet setting. More precisely, when a system requires a Web service component, the system will search Web services providers to choose the optimal Web service that fulfills the requirements [GOL04]. The challenge to reliability of this approach is that these components may not have been subjected to rigorous reliability testing.

Web System Reliability Approach

While there is much coverage of the performance characteristics of Web systems, there has not been equal attention to the contribution of nonfunctional characteristics such as reliability that plays an important role in the selection of Web services by users [ZO07]. My objective is to improve nonfunctional properties, such as reliability, by developing and analyzing comprehensive Web system reliability models. To set the stage for Web system reliability models development, some characteristics of Web systems that influence the design of Web-based models are described below.

Fault-Tolerant Web Systems

The Web Service–Fault-Tolerance Mechanism is an implementation of the classic N-version redundancy model for Web services that can easily be applied to systems with minimal change. The Web services are implemented in different redundant versions. The voting mechanism, which decides whether a component has failed, and, thus, requires replacement, is conducted in the client program (i.e., user) [CHA07]. The problem with this is that while it will work for hardware, it will not work for software because a fault in one version will be a fault in another version!

Web System Communication

In Web services, standard communication protocols and simple client–server requests for Web pages are needed to facilitate service performance because standardization simplifies interoperability [CHA08]. It is necessary to ensure the reliability of Web system communication and the interconnected components. Indeed, the use of networks, such as wireless to access Internet resources such as Web servers, causes failures and degradation of the communication links between Web clients and Web servers. In the Internet, problems such as the decrease of transmission speed due to competing Web client access to Web servers, the decrease in processing performance in Internet routers, and the degradation of the communication lines may occur. Also, a decrease in quality of communication may be caused by changing distances and locations between Web clients and Web servers [NAR05].

Robotic Web Services

Because Web services, implemented in robots, are moving, the distance between clients and servers is likely to vary, and the position of clients and servers relative to the signal will also change. This may affect the quality of communication lines. If a client notices that the electric signal worsens, the client can move to find a place where the electric signal can be received better. The solution is to achieve reliable messaging technology of Web services, combined with a standard for the recovery of failed Web services [NAR05].

Cyclomatic Complexity Analysis of Web System Reliability

The authors exploit the idea of cyclomatic complexity to cover the number of independent paths interconnecting Web clients with Web servers, where cyclomatic complexity is the number of independent paths (i.e., no additional paths can be created from existing paths) in a directed graph representation of a system. Thus, this process focuses on the most likely communication paths, and still maintains the dynamic nature of Web surfing (i.e., communication paths can change rapidly) [WAN03]. This approach is very good, but Web system path data are needed to support its implementation. These data may not be available.

Therefore, based on the above Web system characteristics, you can develop reliability prediction models for assessing the software, hardware, and system quality of a Web system. In performing this assessment, be cognizant of the importance of quality of service [LAK05]. Quality of service is dependent on the nature of Web service client, communication links, and Web server interactions. In order to understand the myriad of failures that can occur in a Web system, for example, on the client side, it is instructive to consider the properties of an XHTML Web page and its associated tree structure. This is advantageous because you can obtain a sense of the types of failures that could occur in constructing a Web page by a Web server. A partial XHTML tree structure is shown in Figure 14.1 [MAC09]. This diagram provides a visual perspective of Web page syntax, which is not always easy to understand in a *linear text* format. Note that errors in Web page design, in any path, could lead to failure in Web page processing by the Web server.

Web Services State Transitions

In composing Web services, the usual assumption is that invocations of Web service operations are independent (i.e., the invocation of a given Web service does not depend on the invocation of another Web service). This assumption, however, does not hold in practice because the service requirements impose ordering on the invocation of operations. Therefore, the use of state machines to model the order of Web service operations is appropriate [HWA07]. In the spirit of this advice, you can use

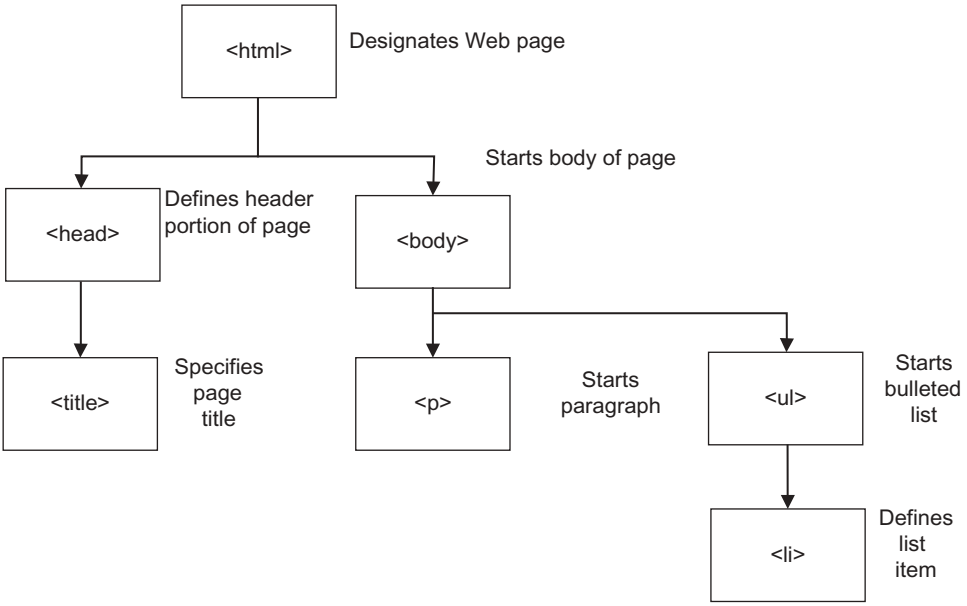


Figure 14.1 XHTML tree structure.

the Web System State Transition Diagram in Figure 14.2, and the supporting transition information in Table 14.1.

Measures for predicting reliability are calculated with the aid of system configuration descriptions, as shown in Figure 14.2 [ALA02]. As shown in Figure 14.2, system configuration descriptions denote the sequence of interactions. In order to obtain the state transition probabilities that will be needed later in predicting total Web system reliability, Table 14.1 presents the state transitions involved when a Web client interacts with a Web server, as portrayed in Figure 14.2. The state transition probabilities shown in this figure are developed in Table 14.3 in a later section. In developing Figure 14.2, note that a Web transaction consists of a client resolving a Web server name to the corresponding Internet Protocol (IP) address—browser accessing the Domain Controller in Figure 14.2—establishing a Transmission Control Protocol (TCP) connection to the Web server, and downloading the object of interest, using Hypertext Transfer Protocol (HTTP) [PAD05]. In addition to failures due to interactions between client and server, failures in the disk storage unit nodes, such as Web servers, account for a significant number of failures [SCH071].

Web Server Proxy

Web server proxy is a well-developed scheme for improving the performance of Web browsing. Users' requests can be supported by a proxy, instead of the processing

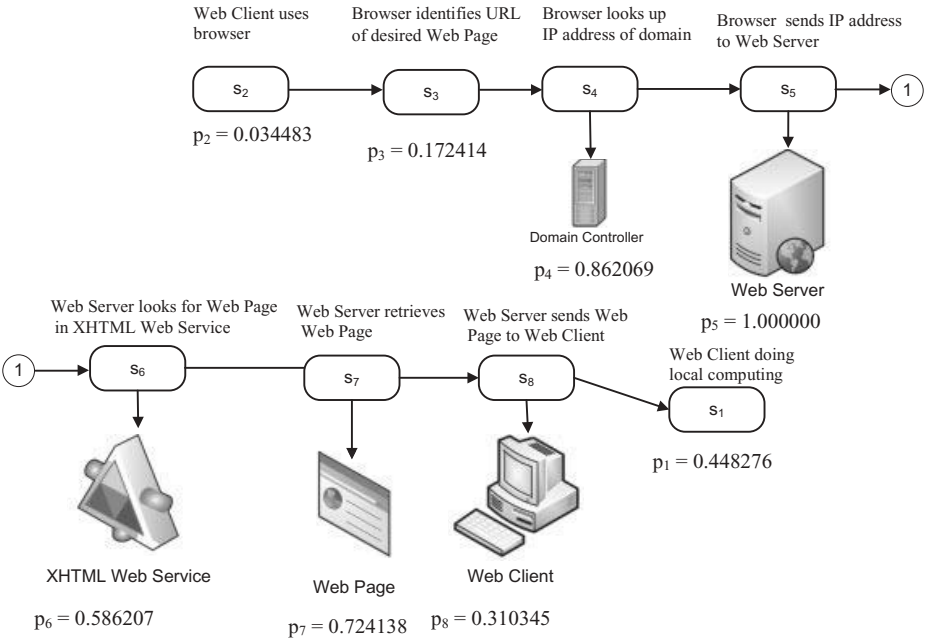


Figure 14.2 Web system state transition diagram. URL, Uniform Resource Locator; p_s, probability of state transition = probability of next state s.

Table 14.1 Web State Transitions

Current state	Next state	Transition trigger
Web Client doing local computing s_1	Web client uses browser s_2	Web client needs Web Page
Web Client uses browser s_2	Browser identifies URL of desired Web Page s_3	Browser locates URL name in Internet list
Browser identifies URL of desired Web Page s_3	Browser looks up IP address of domain s_4	Browser accesses Domain Controller
Browser obtains IP address of domain s_4	Browser sends IP address to Web Server s_5	Automatic state change in browser
Browser sends IP address to Web Server s_5	Web server looks for Web page in XHTML Web Service s_6	Web Server receives request from Web Client
Web Server looks for Web Page in XHTML Web Service s_6	Web Server retrieves Web Page s_7	Web server finds Web Page on XHTML Web Service
Web Server retrieves Web Page s_7	Web Server sends Web page to Web Client s_8	Web Server has found requested Web Page

being performed by a Web server. In this instance, a proxy is a computer that performs ancillary services on behalf of the Web server. These services are, for example, identifying the initial search location in the Web database and formatting output for the user. Performance studies show that a proxy is very effective in reducing the response time of Web accesses [SHE04]. While this is true, the models in this chapter are based on the Web client directly accessing the Web server to obtain a page. Also, Web system service and reliability could be improved by transitioning to another Web site or page in the event of failure of a given Web site [DHA08].

Web Server Failure Data

Web server failure data cannot be found in abundance, which is an understatement! Vendors are not anxious to reveal their reliability problems. Therefore, we have to settle for failure data from computers that *could* function as Web servers, such as the data described below. Actually, the particular data that are used are not important as long as they are representative of the Web environment. What is important are the characteristics of models that predict Web service reliability. The available data are used for explanatory purposes; any representative data could be used.

One of the hardest problems in future high-performance computing (HPC) installations, such as Web servers, will be avoiding, coping with, and recovering from failures. HPC requires the simultaneous use and control of hundreds of thousands or even millions of processing, storage, and networking elements. With this large number of elements involved, element failure will be frequent, making it increasingly difficult for applications to make progress. The success of HPC computing will depend on the ability to provide high reliability, supported by representative failure data. The available data sets cover computer outages in HPC clusters, as well as failures in storage systems [SCH07].

The data obtained were collected during 1995–2005 at Los Alamos National Laboratory (LANL) and covers 22 high-performance computing systems, including a total of 4750 computing systems and 24,101 processors at two sites. The data contain an entry for any failure that occurred during the 9-year time period. The data cover all aspects of system failures: software failures, hardware failures, failures due to operator error, network failures, and failures due to environmental problems (e.g., power outages). Hardware is the single largest component, with 50% of all failures assigned to this category. Software is the second largest contributor, with 20% of all failures at both sites attributed to software. Failure rate varies widely across systems, from 10 failures per year per system to 1180 failures per year per system. Note that a failure rate of 1180 failures per year per system means that a Web server application will fail and require recovery action more than three times per day, thus causing considerable disruption for Web clients. One might wonder what causes the large differences in failure rates across the different systems. The main reason for these differences is that the systems vary widely in size. Thus, the failure rate of a system grows proportional to the number of processors in the system (i.e., size) [SCH07].

WEB SERVER RELIABILITY ANALYSIS

The architecture of a Web Server has a profound impact on its performance and reliability. One of the architectural characteristics of a Web server is its processing method, which describes the type of process that is used to support Web Server operations [GOK06]. While this is true, unfortunately, the available Web Server failure data do not include information on processing architecture.

As the authors attest [NIC05], it is common to use statistical modeling theory for the evaluation of Web-based system reliability. Keying on this idea, let us use various statistical metrics to compute and predict reliability for illustrative Web Servers, using the system, software, and hardware failure data shown in Table 14.2. Note that the numbers of software (20%) and hardware (50%) failures do not add to the number of system failures. The reason for this is that there are other types of failures that are not identified in the Los Alamos failure data. In addition, it is unusual to have a higher percentage of software failures compared with hardware failures. The apparent reason is the complexity of supercomputer hardware configurations.

The probability distribution of choice is the Weibull, as elaborated and justified in the next section. Recall that you were introduced to this reliability distribution in Chapter 12. Based on the patterns of failure data tabulated in Table 14.2, the Weibull distribution proved appropriate for predicting system, software, and hardware reliability.

Weibull Failure Distribution

One of the most widely used distributions for predicting reliability is the Weibull failure distribution [LYU96]. It has the flexibility of allowing for constant, increasing, and decreasing hazard functions (i.e., instantaneous failure rate), as demonstrated by the hazard function in Equation 14.1 [LLO62]:

$$\text{Hazard function: } h(i) = \alpha\lambda(i)^{(\alpha-1)}, \tag{14.1}$$

where α is a shape parameter, i is the system identification in Table 14.2, and λ is a scale parameter.

The parameter λ can also be considered to be the failure rate.

Equation 14.2 represents the probability p(i) of system i failing. This equation is flexible because it can portray various patterns of probability of failure across systems, depending on the values of α and λ [LLO62]:

$$p(i) = \alpha\lambda i^{(\alpha-1)} e^{-\lambda i^{\alpha}}. \tag{14.2}$$

For the exponentially distributed pattern of failure data in Table 14.2, the Weibull reliability in Equation 14.3 is advantageous to use [LLO62]:

$$R(i) = e^{-(\lambda i^{\alpha})}. \tag{14.3}$$

Table 14.2 Software and Hardware Failure Data (1995–2005, LANL)

System ID	**System** failures per year	**Software** failures per year	**Hardware** failures per year	Actual reliability	Weibull **system** failure rate (failures per day, λ)	Weibull **system** shape parameter (α)	Weibull **software** failure rate (failures per day, λ)	Weibull **software** shape parameter (α)	Weibull **hardware** failure rate (failures per day, λ)	Weibull **hardware** shape parameter (α)
1	10	2	5	0.9982	0.0274		0.0055		0.0137	
2	10	2	5	0.9982	0.0274	0.0040	0.0055	0.0354	0.0137	1.3320
3	10	2	5	0.9982	0.0274	0.0025	0.0055	0.0224	0.0137	0.8404
4	80	16	40	0.9857	0.2192	0.0003	0.0438	0.0019	0.1096	0.6617
5	550	110	275	0.9020	1.5068	0.0002	0.3014	0.0003	0.7534	0.5694
6	380	76	190	0.9323	1.0411	0.0001	0.2082	0.0004	0.5205	0.5116
7	1180	236	590	0.7897	3.2329	0.0006	0.6466	0.0002	1.6164	0.4710
8	1150	230	575	0.7950	3.1507	0.0004	0.6301	0.0002	1.5753	0.4408
9	120	24	60	0.9786	0.3288	0.0002	0.0658	0.0005	0.1644	0.4174
10	120	24	60	0.9786	0.3288	0.0002	0.0658	0.0004	0.1644	0.3983
11	120	24	60	0.9786	0.3288	0.0002	0.0658	0.0004	0.1644	0.3825
12	20	4	10	0.9964	0.0548	0.0006	0.0110	0.0037	0.0274	0.3699
13	90	18	45	0.9840	0.2466	0.0001	0.0493	0.0007	0.1233	0.3576
14	120	24	60	0.9786	0.3288	0.0002	0.0658	0.0004	0.1644	0.3475
15	110	22	55	0.9804	0.3014	0.0002	0.0603	0.0007	0.1507	0.3387
16	150	30	75	0.9733	0.4110	0.0002	0.0822	0.0005	0.2055	0.3306
17	100	20	50	0.9822	0.2740	0.0001	0.0548	0.0005	0.1370	0.3237
18	140	28	70	0.9750	0.3836	0.0002	0.0767	0.0003	0.1918	0.3172
19	350	70	175	0.9376	0.9589	0.0001	0.1918	0.0002	0.4795	0.3113
20	700	140	350	0.8752	1.9178	0.0002	0.3836	0.0002	0.9589	0.3060
21	40	8	20	0.9929	0.1096	0.0003	0.0219	0.0018	0.0548	0.3015
22	60	12	30	0.9893	0.1644	0.0003	0.0329	0.0006	0.0822	0.2968

The parameters of the Weibull distribution are estimated according to reference [LL062] in Equations 14.4 and 14.5, where n is the number of systems:

$$\lambda = \frac{n}{\sum_{j=1}^{n} i_j^{\alpha}}, \tag{14.4}$$

$$\alpha = \frac{n}{\lambda \sum_{j=1}^{n} i_j^{\alpha} \log i_j - \sum_{j=1}^{n} \log i_j}. \tag{14.5}$$

However, trying to solve Equations 14.4 and 14.5 is not practical because in order to solve for λ in Equation 14.4, α is required, but to solve for α in Equation 14.5, λ is required. A practical approach is to use the reliability function, Equation 14.3 to solve for α, given values of λ and R for a specified value of i.

Now, solving Equation 14.3 for α results in Equation 14.6:

$$\alpha = \frac{\log\left[\frac{-\log(R(i))}{\lambda}\right]}{\log(i)}. \tag{14.6}$$

However, notice the constraint on the maximum value of R(i) that can be achieved to avoid trying to take the log of a negative quantity: $R(i) < e^{-\lambda}$ because $\log R(t) = \lambda$. Therefore, set the limit on R(i) according to $R(i) < e^{-\lambda}$, and substitute this value in Equation 14.6 and solve for α.

In order to examine the validity of reliability predictions using Equation 14.3, the actual reliability $R_a(i)$, based on historical failure data, is computed in Equation 14.7, where $f_s(i)$ is the number of software failures for system i, computed over n systems:

$$R_a(i) = 1 - \frac{f_s(i)}{\sum_{i=1}^{n} f_s(i)}. \tag{14.7}$$

The error between the predicted and actual reliabilities is computed using the mean relative error that is computed as: mean*((actual – predicted)/actual) [FEN97].

Factoring in Probability of State Transitions

Now, although the reliability analysis that was presented is appropriate, it is only relevant when, according to Table 14.1, there is a state transition that causes a given node (e.g., Web Server) in the Web system in Figure 14.2 to become active (e.g., Web server looks for Web page). Therefore, to predict *total* Web system reliability,

Table 14.3 Web State Transition Probabilities

Current state	Next state	Transition probability = Probability of next state
Web Client doing local computing s_1	Web Client uses browser s_2	p_2 (1) = 0.034483 w (2) = 0.008333
Web Client uses browser s_2	Browser identifies URL of desired Web Page s_3	p_3 (1) = 0.172414 w (3) = 0.041667
Browser identifies URL of desired Web Page s_3	Browser looks up IP address of domain s_4	p_4 (2, 3, 4) = 0.862069 w (4) = 0.208333
Browser obtains IP address of domain s_4	Browser sends IP address to Web Server s_5	p_5 (4) = 1.000000 w (5) = 0.241667
Browser sends IP address to Web Server s_5	Web Server looks for Web Page in XHTML Web Service s_6	p_6 (6, 7) = 0.586207 w (6) = 0.141667
Web Server looks for Web Page in XHTML Web Service s_6	Web Server retrieves Web Page s_7	p_7 (7, 8) = 0.724138 w (7) = 0.175000
Web Server retrieves Web Page s_7	Web Server sends Web Page to Web Client s_8	p_8 (6, 5, 2) = 0.310345 w (8) = 0.075000
Web Server sends Web Page to Web Client s_8	Web Client doing local computing s_1	p_1 (1) = 0.448276 w (1) = 0.108333

each *node and link system reliability* (systems of Table 14.2) must be multiplied by the *weighted* probability of state transition $w(i)\ p_s(i)$, where $p_s(i)$ is the unweighted probability and w(i) are weights that sum to one. Then, the values of $w(i)\ p_s(i)\ R(i)$ are summed to predict total system reliability R_s. The result is Equation 14.8:

$$R_s = \sum_{i=1}^{n} w(i)\ p_s(i)\ R(i). \tag{14.8}$$

A random number generator was coded in C++ to produce random numbers from which probabilities of state transitions were derived. These probabilities and weights are shown in Table 14.3. Note that these probabilities are for the purpose of *illustrating* the computation of Web system reliability. Other probabilities could be used in other situations.

Using the logic of Figure 14.2—Web System State Transition Diagram—and the state transition information in Table 14.3, the Web Client and Server Interactions is constructed in Figure 14.3. With the probabilities of state transitions appended, this figure will be used to predict total Web system reliability, as given by Equation 14.8.

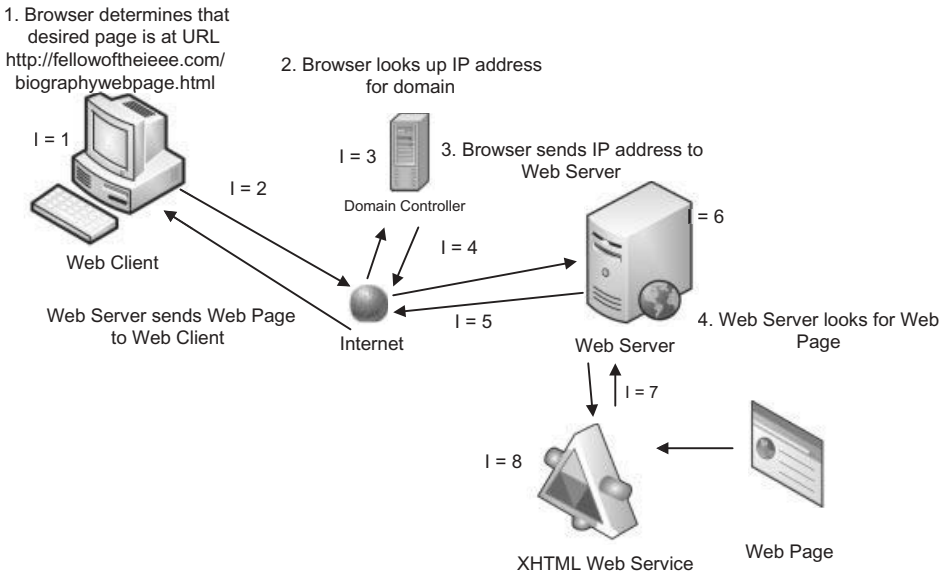


Figure 14.3 Web client and Web server interactions. i, node or link identification.

Reliability Analysis Based on Web Systems

The purpose of Figure 14.4 is to identify which Web systems are able to satisfy the Web Server reliability requirement at the software, hardware, and system levels, and to compute the prediction accuracy of software, hardware, and system with respect to actual reliability. We see that only systems 1 and 2 satisfy the requirement and that software has the best prediction accuracy as judged by the mean relative error (MRE). This means that additional debugging of the faults in the system and hardware is necessary, recognizing that system failures include user and computer operator errors. Traditionally, user and operator errors have not been analyzed because the information may not be available. Since these errors could be significant contributors to unreliability, they should be tracked by using user and computer operator logs. Note that these systems are the ones from the LANL, where the failure data are documented in Table 14.2, and the Web Server system is depicted in Figure 14.2.

It is also important to track the hazard function (i.e., instantaneous failure rate) produced by Web servers in Figure 14.5 to determine whether there is any anomalous behavior (i.e., sudden jumps in hazard function) that would jeopardize reliability. As Figure 14.5 shows, indeed, there are cases of hardware and system showing sudden jumps in hazard function, thus reinforcing the finding from Figure 14.4 that hardware and system are candidates for additional fault removal.

Another reliability metric of interest is the probability of failure of Web server systems shown in Figure 14.6, with software demonstrating the lowest probability, once the probability of failure reaches steady state. Recall that the reliability plots

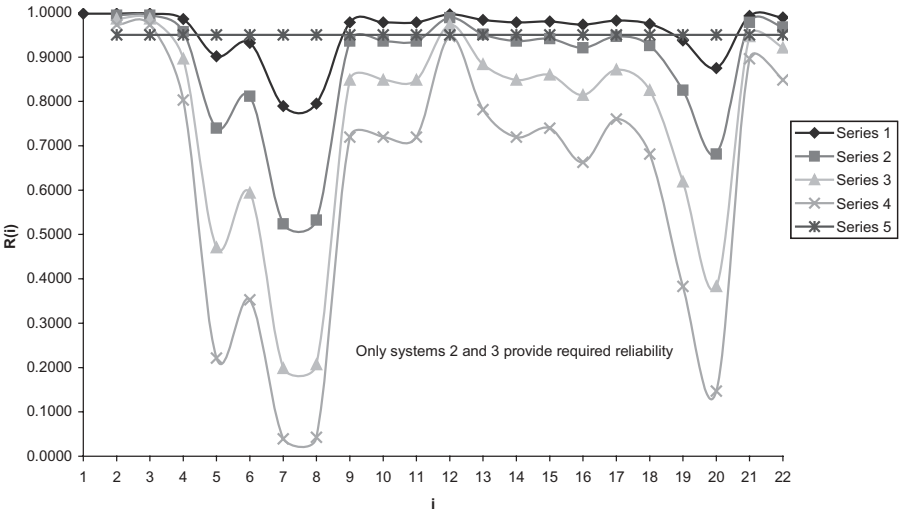


Figure 14.4 Web server reliability R(i) versus system identification i. Series 1: Actual reliability. Series 2: Predicted software reliability, MRE = 0.0849. Series 3: Predicted hardware reliability, MRE = 0.2225. Series 4: Predicted system reliability, MRE = 0.3608. Series 5: Required reliability = 0.9500.

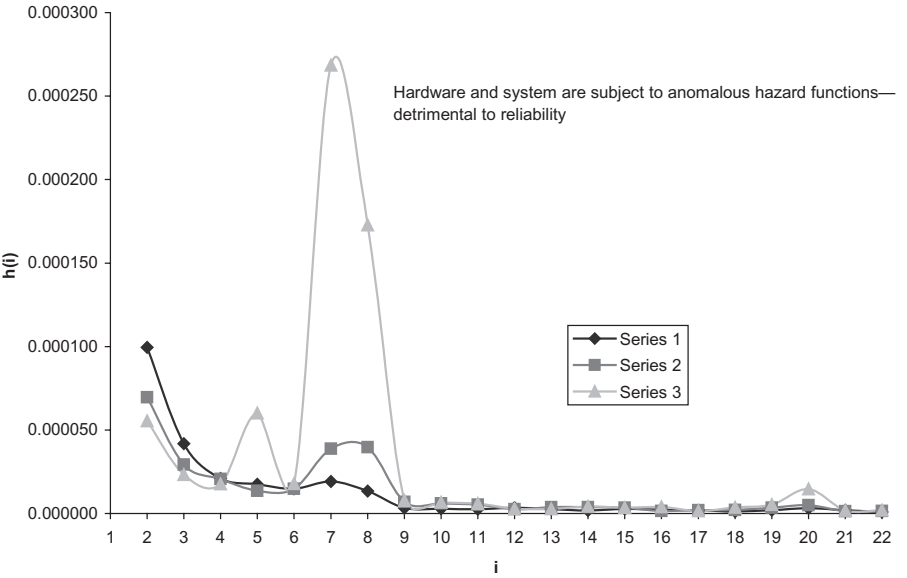


Figure 14.5 Web server hazard function h(i) versus system identification i. Series 1: Software. Series 2: Hardware. Series 3: System.

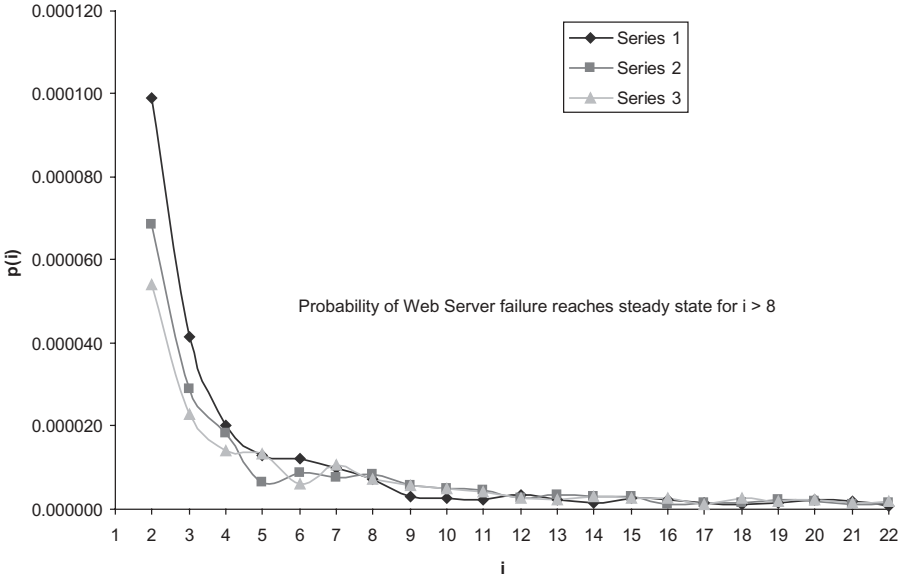


Figure 14.6 Probability of Web server failure p(i) versus system identification i. Series 1: Software. Series 2: Hardware. Series 3: System.

in Figure 14.4 showed that only systems 2 and 3 satisfied the reliability requirement, but according to Figure 14.6, it might be better to select system 9 as the Web Server because at this point, the probability of failure has stabilized. This would be a false choice because reliability is the primary metric; other metrics are of secondary importance. Since the probabilities of failure are relatively small for systems 2 and 3 in Figure 14.6, these systems would remain our choice for Web server.

When predicted reliabilities do not satisfy the required reliability, as is the case in Figure 14.4, we can solve Equation 14.3 for λ to predict the required failure rate λ that is necessary to achieve required reliability R(i). The result is Equation 14.9. Statistical testing and reliability analysis can be used effectively to assure quality for Web applications [KAL01]. Therefore, Equation 14.9 helps us determine how much testing must be conducted to remove faults in order to reduce the failure rate λ to a value that will achieve the required reliability, R(i):

$$\lambda = -\frac{\log R(i)}{i^{\alpha}}. \tag{14.9}$$

Web Server Reliability Analysis Based on Operating Time

Up to this point we have used LANL data and made reliability predictions across these systems. Now, we focus on using data and making predictions as a function of Web server *operating time*. Failure phenomena of Web server systems depends

on their workload characteristics. As a result, the number of user Web sessions strongly affects the failure rate of Web servers [FUJ09]. While this is true, *operating time* is a better metric of workload than number of sessions because it represents the *continued*, not periodic, use of Web facilities. Therefore, to start the analysis, Table 14.4 is presented showing the *operating time-oriented* data for system, software, and hardware failure rates, where 20% of the failures are contributed by software and 50% by hardware. Typically, software accounts for a larger proportion of failures than hardware. However, in the case of Los Alamos supercomputers, the hardware configurations are very complex. This complexity contributes a disproportionate share of failures.

Note that in using *operating time*, all equations that have been developed remain the same except that *operating time* variable t is substituted for system identification i.

Next, predict the *operating time-oriented* reliability of software, hardware, and system and compare these predictions to the actual reliability by computing the MRE, similar to our previous analysis of the system-oriented reliabilities. Figure 14.7 shows that none of the reliability metrics—software, hardware, system—achieve the required reliability. This means that, again, we must call upon Equation 14.9 to find the reduced failure rates that would allow the required reliability to be achieved. Figure 14.8 shows the dramatic reduction in *system* failure rate required to bring predicted reliabilities into conformance with required reliability. The implication of this result is that a massive reduction in Web server faults must occur through comprehensive testing.

WEB CLIENT RELIABILITY ANALYSIS

The logic for developing the client-side Web probability of failure model is to consider that, with a historical error rate of n errors per Web page operation, N number of operations on the Web page, and an assumed exponential decrease in reliability, as n and N increase, Equation 14.10 is produced reflecting the logic of n Web page errors occurring over N Web page operations. Admittedly, there are no data to prove the behavior of Equation 14.10. However, it seems reasonable that, as n and N increase, the complexity of the Web page increases at an exponential rate, reflected in an exponentially decreasing reliability $R_c(n, N)$.

The overall failure rate for a given server or a given client can be noticeable. Failure rates in excess of 2% are not uncommon. The failure rate varies considerably across servers and clients. About 30% of the failures can be traced to Domain Name Server (DNS) problems, and most of the rest are due to the inability of the client to establish a TCP connection to the remote Web server. (Note that the DNS lookup accesses are included in Fig. 14.2.) Client-side problems account for the overwhelming majority of DNS lookup failures, whereas server-side problems are the dominant cause of TCP connection failures [PAD05]. Therefore, in predicting client-side *probability of failure*, based on the above failure history, assume various values of n—as much as 2% in Figure 14.9—to see how sensitive the result is to the size of n, for given

Table 14.4 Failure Data Related to Operating Time (1995–2005, LANL)

Year operating time	**System** failures per year	**Software** failures per year	**Hardware** failures per year	Actual reliability	Weibull **system** failure rate (failures per day, λ)	Weibull **system** shape parameter (α)	Weibull **software** failure rate (failures per day, λ)	Weibull **software** shape parameter (α)	Weibull **hardware** failure rate (failures per day, λ)	Weibull **hardware** shape parameter (α)
1	380	76	190	0.9312	1.0411		0.2082		0.5205	
2	680	136	340	0.8769	1.8630	0.0010	0.3726	0.0005	0.9315	0.0016
3	30	6	15	0.9946	0.0822	0.0034	0.0164	0.0095	0.0411	0.0162
4	70	14	35	0.9873	0.1918	0.0016	0.0384	0.0021	0.0959	0.0055
5	580	116	290	0.8950	1.5890	0.0012	0.3178	0.0008	0.7945	0.0019
6	390	78	195	0.9294	1.0685	0.0011	0.2137	0.0012	0.5342	0.0018
7	1190	238	595	0.7846	3.2603	0.0022	0.6521	0.0012	1.6301	0.0008
8	1150	230	575	0.7919	3.1507	0.0035	0.6301	0.0028	1.5753	0.0014
9	120	24	60	0.9783	0.3288	0.0019	0.0658	0.0034	0.1644	0.0041
10	130	26	65	0.9765	0.3562	0.0010	0.0712	0.0021	0.1781	0.0095
11	140	28	70	0.9747	0.3836	0.0023	0.0767	0.0025	0.1918	0.0036
12	20	4	10	0.9964	0.0548	0.0074	0.0110	0.0135	0.0274	0.0448
13	60	12	30	0.9891	0.1644	0.0042	0.0329	0.0064	0.0822	0.0092
14	110	22	55	0.9801	0.3014	0.0025	0.0603	0.0049	0.1507	0.0100
15	100	20	50	0.9819	0.2740	0.0014	0.0548	0.0038	0.1370	0.0085
16	105	21	52.5	0.9810	0.2877	0.0029	0.0575	0.0060	0.1438	0.0095
17	90	18	45	0.9837	0.2466	0.0023	0.0493	0.0060	0.1233	0.0104
18	110	22	55	0.9801	0.3014	0.0029	0.0603	0.0057	0.1507	0.0117
19	70	14	35	0.9873	0.1918	0.0036	0.0384	0.0047	0.0959	0.0122

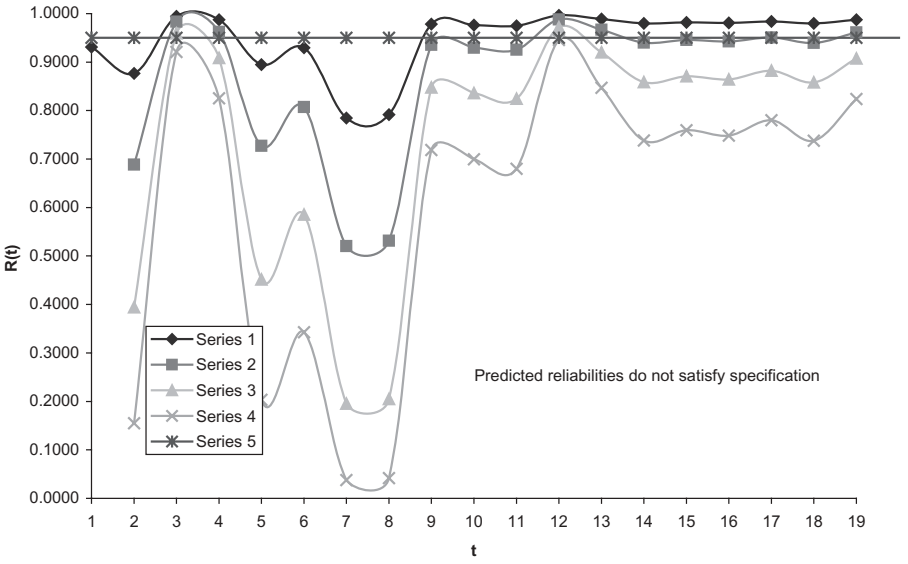


Figure 14.7 Web server reliability R(t) versus operation time t. Series 1: Actual reliability. Series 2: Predicted software reliability, MRE = 0.0895. Series 3: Predicted hardware reliability, MRE = 0.2332. Series 4: Predicted system reliability, MRE = 0.3744. Series 5: Specified reliability = 0.9500.

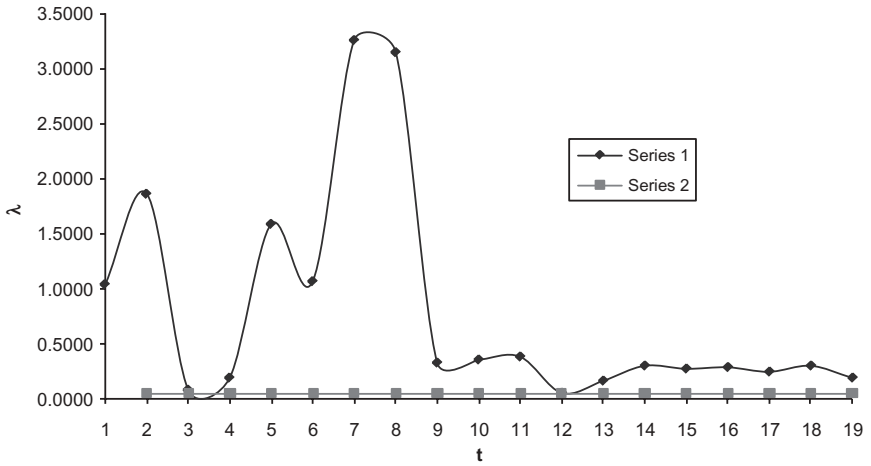


Figure 14.8 Web server system failure rate λ versus operating time t. Series 1: failure rate for predicted reliability. Series 2: failure rate required to achieve required reliability = 0.9500.

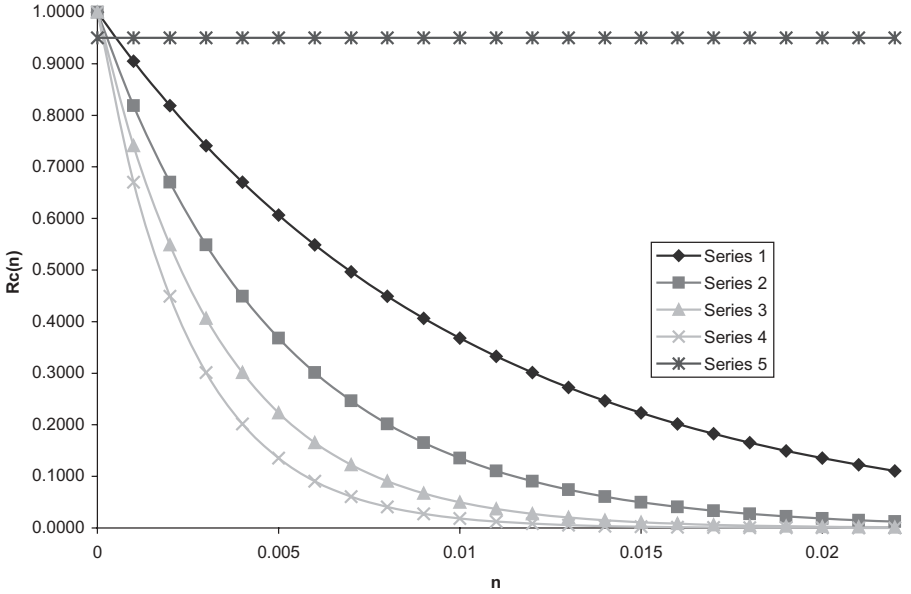


Figure 14.9 Web client reliability $R_c(n)$ versus Web page error rate n. Series 1: N = 100. Series 2: N = 200. Series 3: N = 300. Series 4: N = 400. Series 5: Required reliability = 0.9500. N, number of Web page operations.

values of N. It is also of interest to see how reliability changes as a function of n as reliability is increased by reducing the number of errors on a Web page. Therefore, the rate of change of reliability with respect to n is predicted in Equation 14.11:

$$R_c(n,N) = e^{(-Nn)}, \tag{14.10}$$

$$\frac{dR_c(n,N)}{d(n)} = -NR_c(n,N). \tag{14.11}$$

Calling on Equation 14.10 in Figure 14.9, we are able to determine whether the Web client meets the reliability requirement for various values of number of errors n and number of operations N. As can be seen, this is not the case. Therefore, considerable debugging of client software and hardware is necessary to achieve the required reliability. In order to determine the error rate n that would be required to achieve the required reliability for a given value of N, manipulate Equation 14.10 to produce Equation 14.12:

$$n = -\frac{\log R_c(n,N)}{N}. \tag{14.12}$$

It is evident in Figure 14.10 that an excessive number of Web page operations is bad news for reliability because the rate of change of reliability increases in the negative direction as the number of Web operations increases.

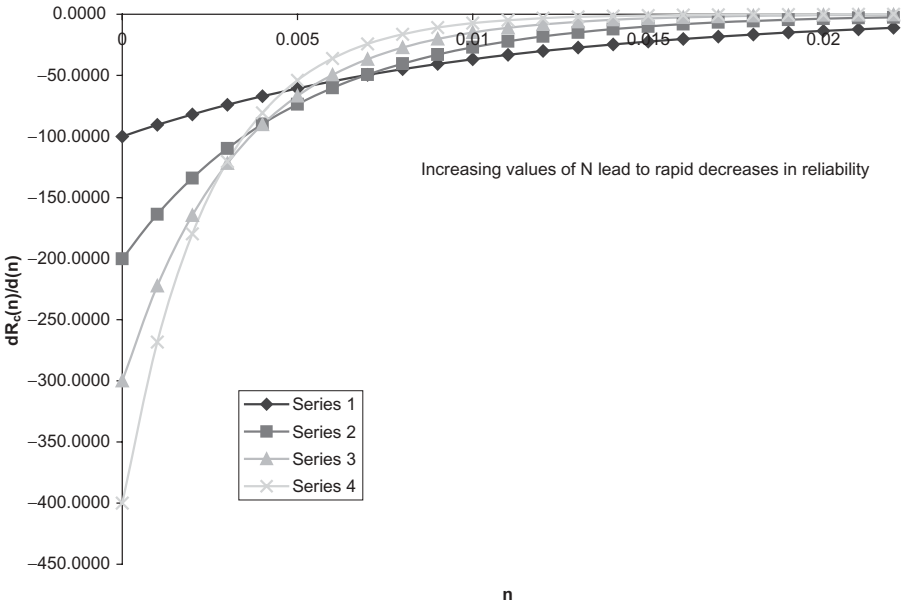


Figure 14.10 Web client rate of change of reliability $dR_c(n, N)/d(n)$ versus error rate n. Series 1: N = 100. Series 2: N = 200. Series 3: N = 300. Series 4: N = 400. N, number of Web page operations.

COMMUNICATION RELIABILITY ANALYSIS

The last element in the Web system model to be subjected to reliability analysis is the communication among Web system elements in Figure 14.2. The Web system error rate is defined as the frequency with which errors or noise are introduced into communication channels. Error rate may be measured in terms of erroneous bits received per bits transmitted B. The distribution of errors is usually nonuniform, with a higher probability of small message size B and a lower probability of large message size. Thus, use the exponential distribution to represent the error rate in Equation 14.13, where λ is the communication channel error rate in megabits per second (Mbit/s), b_m is the mean error rate, and B_M is the maximum bandwidth in megabits per second assumed available to the Web system. Given the exponential decay in error rate in Equation 14.13, the reliability, $R_{cc}(t)$, of the communication channel in Equation 14.14 is expected to degrade exponentially with operating time t:

$$\lambda \quad (b_m)e^{-\left(\frac{B}{B_M}\right)}, \tag{14.13}$$

$$R_{cc}(t) \quad e^{(-\lambda t)}. \tag{14.14}$$

As Figures 14.11 and 14.12 attest, client and server are restricted in obtaining required communication reliability to a bandwidth of 30 Mbit/s (Fig. 14.11) and an

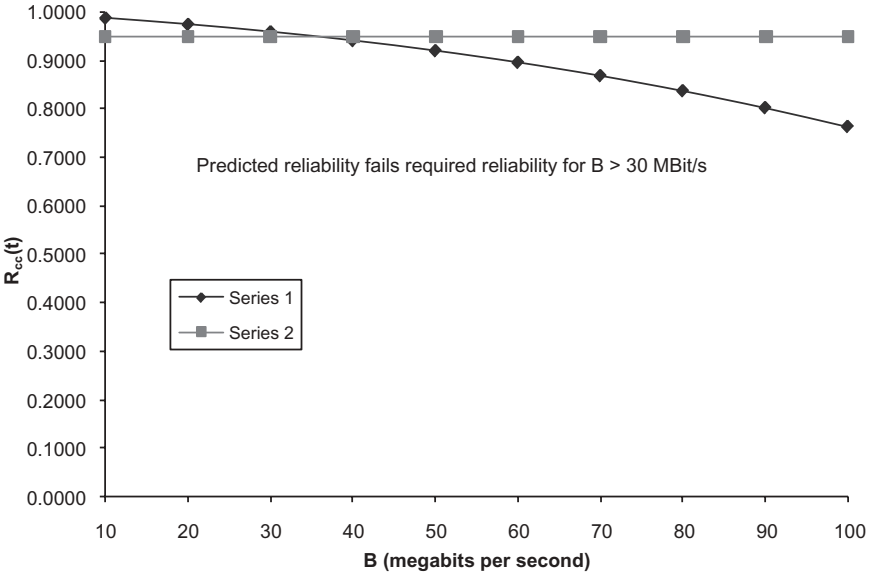


Figure 14.11 Web communication channel reliability $R_{cc}(t)$ versus bandwidth B. Series 1: Communication channel reliability. Series 2: Required reliability = 0.9500.

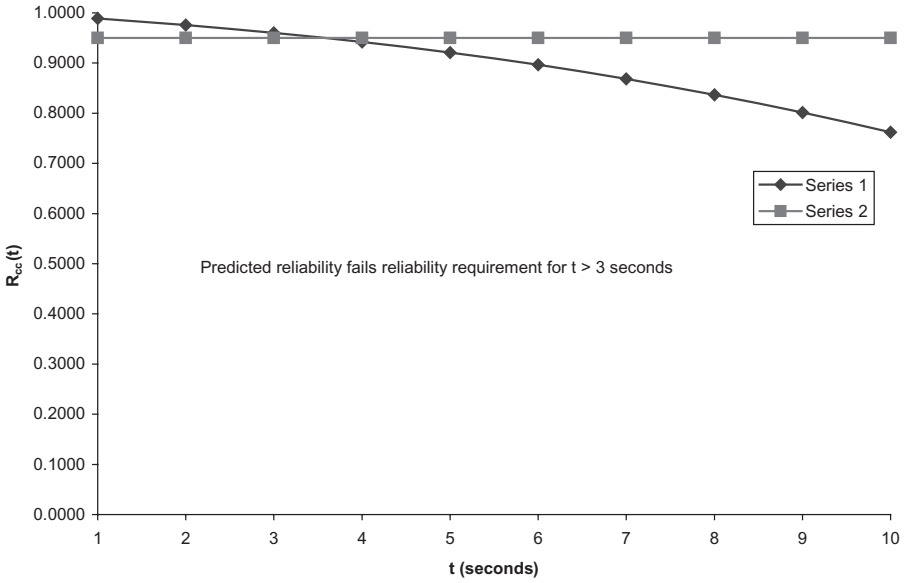


Figure 14.12 Web communication channel reliability $R_{cc}(t)$ versus operating time t. Series 1: Predicted reliability. Series 2: Required reliability = 0.9500.

operating time of 3 seconds (Fig. 14.12). If higher reliability is required, Web system providers and users would have to invest in higher reliability communication facilities. While reliability performance is obviously not outstanding, the situation is not quite so dire because operating time refers to the time required to complete single communication functions, not all the Web system functions illustrated in Figure 14.2.

TOTAL SYSTEM RELIABILITY ANALYSIS

Individual Web components can be used to form value-added total Web services. The value of total Web services is directly influenced by the reliability of individual components [YAN06]. Following this dictum, invoke the total system reliability Equation 14.8 to predict total Web system reliability in Figure 14.13. We see that required reliability is satisfied for only a limited range of operating time. Furthermore, by including client, server, and communication component reliabilities in Figure 14.13, we are able to prioritize the components for reliability improvement, yielding the result that the server component is the first in line for reliability improvement. In addition, Figure 14.13 provides us with the increase in system reliability necessary to achieve the reliability goal for each value of operating time.

Question for Reader: Based on what you have learned in this chapter, what process could you use to choose among existing Web services in terms of performance and reliability?

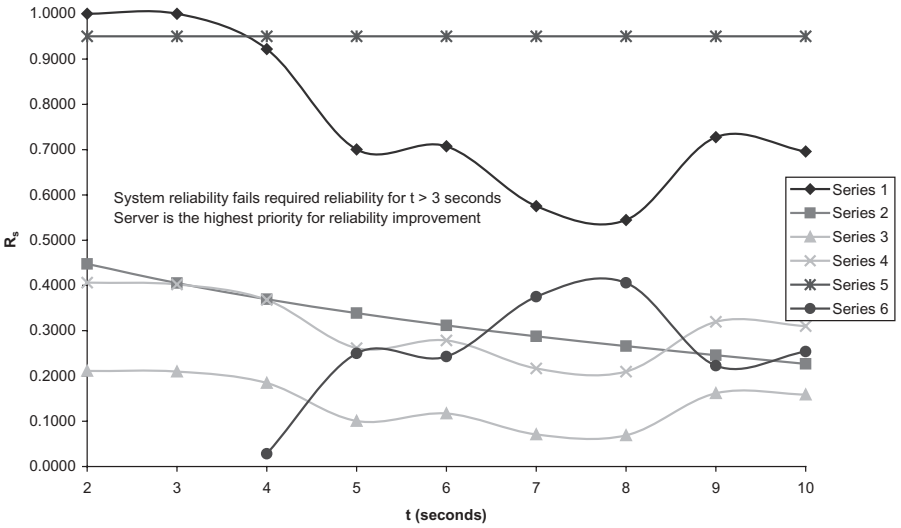


Figure 14.13 Total Web system reliability R_s versus operating time t. Series 1: System. Series 2: Client. Series 3: Server. Series 4: Communication channel. Series 5: Required reliability = 0.9500. Series 6: Required system reliability improvement.

Answer: Probably the most important performance factor that was not covered in the chapter is the relevance of search results to the user's information needs. It was not covered because search relevance cannot be generalized. It is highly personable and only has meaning for the Web client. Thus, you could evaluate the relevance, coupled with response time, for various searches important to you, over various Web services, and compare the results.

With respect to reliability, you could repeatedly access a group of Web systems in rapid succession for the same search request, over an observed operating time, and record any failures to provide search results. If failure counts are obtained, the data would be used in the *system actual* reliability, Equation 14.7, to compute the reliability of each Web system. The result would be one basis for choosing a Web system.

SUMMARY AND CONCLUSIONS

In order to obtain a comprehensive and valid assessment of Web system reliability and related metrics it is necessary to decompose the system into its component parts, predict component reliabilities, and then do an integrative analysis to produce total system reliability predictions. The reason for this is that there are different failure properties for Web client, Web server, and the interconnected communication channels. This process includes the following steps:

1. Identify Web page operations so that the number of ways Web clients and servers could fail can be ascertained.
2. Identify states and state transition probabilities so that components reliabilities can be properly weighted to produce total system reliabilities.
3. Determine whether each component and the system satisfy the reliability requirement.
4. When reliability requirements are not achieved, compute the failure rates required to bring components into conformance with reliability requirements.
5. Use rate of change of Web client and server-predicted reliability, with respect to error rate, to identify the number of Web page operations that cause reliability degradation.
6. Study the effects of increasing bandwidth and operating time on communication channel reliability.
7. Integrate component reliabilities into total system reliability predictions and compute the reliability improvement necessary to achieve the reliability goal.

The above process not only yields important Web system reliability predictions, but, in addition, allows the researcher and practitioner to understand how all the pieces of the reliability picture fit together, thus supporting reliability analyses.

REFERENCES

[ALA02] V. S. Alagar and O. Ormandjieva, "Reliability assessment of WEB applications," *26th Annual International Computer Software and Applications Conference*, 2002, p. 405.

[CHA07] Pat P. W. Chan, Michael R. Lyu, and Miroslaw Malek, "Reliable web services: methodology, experiment and modeling," *IEEE International Conference on Web Services (ICWS 2007)*, 2007.

[CHA08] Pat Pik Wah W. Chan and Michael R. Lyu, "Dynamic web service composition: a new approach in building reliable web service," *22nd International Conference on Advanced Information Networking and Applications*, 2008, pp. 20–25.

[DHA08] Sanjeev Dhawan, and Rakesh Kumar, "Analyzing performance of web-based metrics for evaluating reliability and maintainability of hypermedia applications," *Third International Conference on Broadband Communications, Information Technology & Biomedical Applications*, 2008, pp. 376–383.

[FEN97] Norman F. Fenton and Shari Lawrence Pfleeger, *Software Metrics: A Rigorous & Practical Approach*, 2nd ed. London: PWS Publishing Company, 1997.

[FER03] C. Ferris and J. Farrell, "What are web services?," *Communications of the ACM*, 2003.46(6), p. 31.

[FUJ09] Toshiya Fujii and Tadashi Dohi, "Statistical failure analysis of a web server system," *International Conference on Availability, Reliability and Security*, 2009, pp. 554–559.

[GOK06] Swapna S. Gokhale, Paul J. Vandal, and Jijun J. Lu, "Performance and reliability analysis of web server software architectures," *12th Pacific Rim International Symposium on Dependable Computing*, 2006, pp. 351–358.

[GOL04] N. Gold, C. Knight, A. Mohan, and M. Munro, "Understanding service-oriented software," *IEEE Software*, 2004, pp. 71–77.

[HOL02] P. Holland, "Building web services from existing application," *eAI Journal*, 2002, pp. 45–47.

[HWA07] San-Yih Hwang, Ee-Peng Lim, Chien-Hsiang Lee, and Cheng-Hung Chen, "On composing a reliable composite web service: a study of dynamic web service selection," *IEEE International Conference on Web Services*, 2007, pp. 184–191.

[KAL01] Chaitanya Kallepalli and Jeff Tian, "Measuring and modeling usage and reliability for statistical web testing," *IEEE Transactions on Software Engineering*, 2001, 27(11), pp. 1023–1036.

[LAK05] Neila Ben Lakhal, Takashi Kobayashi, and Haruo Yokota, "A failure-aware model for estimating and analyzing the efficiency of web services compositions," *11th Pacific Rim International Symposium on Dependable Computing*, 2005, pp. 114–124.

[LLO62] David K. Lloyd and Myron Lipow, *Reliability: Management, Methods, and Mathematics*. Englewood Cliffs, NJ: Prentice-Hall, Inc., 1962.

[LYU96] Michael R. Lyu (ed.), *Handbook of Software Reliability Engineering*. IEEE Computer Society Press and McGraw-Hill Book Company, 1996.

[MAC09] Matthew MacDonald, *Creating a Web Site*, 2nd ed. Sebastopol, CA: O'Reilly Media, Inc., 2009.

[NAR05] Masahiko Narita, Makiko Shimamura, and Makoto Oya, "Reliable protocol for robot communication on Web services," *International Conference on Cyberworlds*, 2005, pp. 210–220.

[NIC05] Leticia Davila-Nicanor and Pedro Mejia-Alvarez, "Reliability evaluation of web-based software applications," *Sixth Mexican International Conference on Computer Science*, 2005, pp. 106–112.

[PAD05] Venkata N. Padmanabhan, Sriram Ramabhadran, and Jitendra Padhye, "Client-Based Characterization and Analysis of End-to-End Internet Faults," Technical Report MSR-TR-2005-29, Microsoft Research, March 2005.

[PAR90] D. L. Parnas, A. J. V. Schouwen, and S.P. Kwan, "Evaluation of safety-critical software," *Communications of the ACM*, 1990, 33(6), pp. 636–648.

[SCH07] Bianca Schroeder and Garth A. Gibson, "Understanding failures in petascale computers," Computer Science Department, Carnegie Mellon University, Journal of Physics: Conference Series 78 (2007) 012022, IOP Publishing.

[SCH071] Bianca SCHROEDER and Garth A. GIBSON, "Disk failures in the real world: what does an MTTF of 1,000,000 hours mean to you?," *FAST'07: 5th USENIX Conference on File and Storage Technologies*, San Jose, CA, February 2007, pp. 14–16.

[SHE04] Bo SHENG and Farokh B. BASTANI, "Secure and reliable decentralized peer-to-peer web cache," *18th International Parallel and Distributed Processing Symposium (IPDPS'04)*, Vol. 1, 2004, p. 54b.

[WAN03] Wen-Li WANG and Mei-Huei TANG, "User-oriented reliability modeling for a web system," *14th International Symposium on Software Reliability Engineering*, 2003, p. 293.

[YAN06] Yanping YANG, Qingping TAN, Yong XIAO, Jinshan YU, and Feng LIU, "Exploiting hierarchical CP-nets to increase the reliability of web services workflow," *2006 International Symposium on Applications and the Internet*, 2006, pp. 116–122.

[ZO07] Hangjung ZO, Derek L. NAZARETH, and Hemant K. JAIN, "Measuring reliability of applications composed of web services," *40th Annual Hawaii International Conference on System Sciences*, 2007, p. 278c.

Chapter 15

Mobile Device Engineering

Issues in mobile network reliability, performance, and context and network awareness are examined. Based on mobile phone failure data reported in the literature, reliability models for assessing mobile network reliability are explored from two perspectives: by type of failure and by category of failure recovery action. This chapter builds on the foundation provided in the Chapter 8. Furthermore, the operational time corresponding to specified reliability values are predicted. Based on these calculations, you could conclude that current mobile networks are unable to provide highly reliable service for more than a few months of operation. In addition, a novel signal-to-noise ratio is developed and computed, and applied to assessing mobile network stability. Where data were not available, such as in issues involving context (i.e., environment in which mobile device is operational, such as a wireless hot spot) and network awareness (i.e., mobile device having the intelligence to recognize its operational environment), I have indicated with diagrams how mobile networks could respond to changes in both context and network awareness.

INTRODUCTION

The chapter's objective is to discuss a number of issues in mobile computing, such as risks of operating mobile devices, the problem of maintaining adequate power in a mobile network, mobile device software reliability, context-aware and network-aware mobile computing, and mobile device performance. Because the mobile environment involves many software and hardware components and technologies, it is important to address many relevant issues. Thus, for each of these issues, where appropriate, a quantitative approach is used for making assessments of the need for mobile device improvement. Reliability is an example of where the quantitative approach is applied, which uses failure data reported in the literature to develop several quantitative assessments of mobile network reliability, based on types of failures and responses to the failures. In other cases, such as context-aware mobile computing, where there is no quantitative data relating reliability and performance to the context of the mobile environment, a qualitative analysis is provided.

Computer, Network, Software, and Hardware Engineering with Applications, First Edition. Norman F. Schneidewind.
© 2012 the Institute of Electrical and Electronics Engineers, Inc. Published 2012 by John Wiley & Sons, Inc.

Major Risks Posed By Mobile Devices

Security

While mobile devices are productivity-enhancing tools, they bring new security threats to the enterprise. A security breach on the device can be expensive to the user organization. The increasing numbers of mobile users and the explosion of Internet connectivity have demolished the concept of a "fixed" information perimeter for organizations. A company network protected only by a central firewall is no longer adequate. Users frequently travel outside the perimeter, where they can expose confidential data and risk attacks. Mobile devices are at risk of carrying viruses and other malware. These information corruptions may be released into the network [TRE]. In order to counteract security threats, a network firewall, lightweight encryption, intrusion detection, and antimalware software should be employed, as shown in Figure 15.1.

Intrusion Detection

Of particular concern in the protection of mobile networks is intrusion detection because if intrusion is successful, it could disrupt an entire network. Intrusion detection techniques sense intrusions while they are acting on an information system. Existing intrusion detection techniques fall into two major categories: signature recognition and anomaly detection [DEL04, ESA98]. Signature recognition techniques match entities in an information system with signatures of known entity

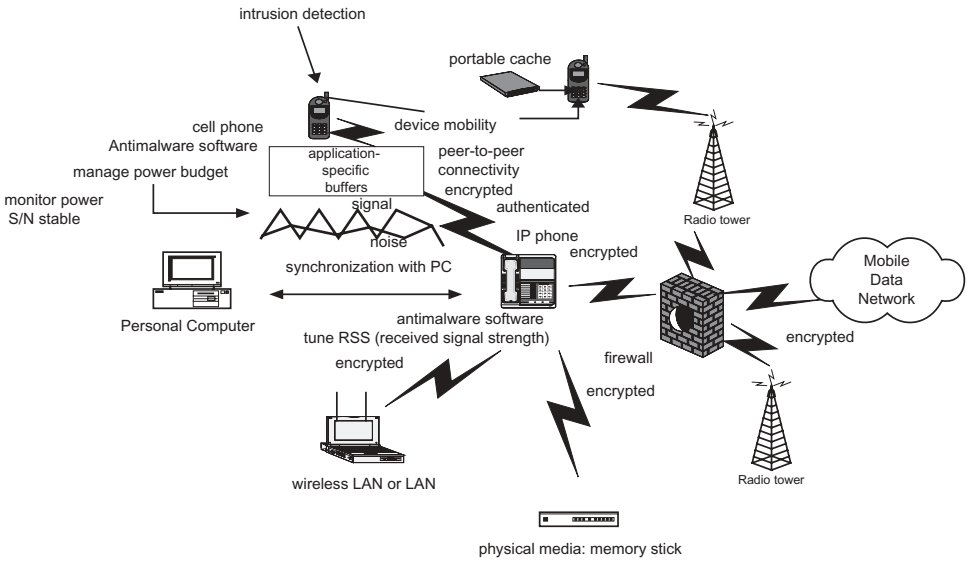


Figure 15.1 Mobile device connectivity for security and performance.

intrusions and signal an intrusion when there is a match. For example, consider the entities: user, file, program, host, network, and so on. Signature recognition techniques establish a profile of the entity's normal behavior, for example, the files a user is authorized to access. Then, anomaly detection compares the observed behavior of the entity with the profile, and signals intrusions when the entity's observed behavior deviates significantly from its profile, for example, when the user is accessing unauthorized files [YE02]. A successful intrusion increases the noise in a mobile network and, thus, lowers the signal-to-noise ratio (S/N).

Power Loss

Another important risk of operating mobile devices is the challenge of power management and potential loss of power. These devices are increasingly being used in multimedia streaming-type applications, common examples being on-demand movie streaming and video conferencing. In spite of technological advances, battery life still remains a major limitation of portable devices. The main power consuming components of a mobile device are: central processing unit (CPU), display, and network interface. Running multimedia applications further aggravates the situation, because these programs are both CPU and network intensive. However, while the CPU and network may benefit from managing the power budget (see Fig. 15.1), displays need to be on at all times and thus limits the possibilities for saving power without severely impacting the user experience [COR06]. The risk of power loss can be mitigated by the use of power monitoring, as shown in Figure 15.1.

MOBILE DEVICE RELIABILITY

Software Mobile Network Products

In mobile ad hoc networks, wireless media have limited and variable ranges, as distinguished from wired media. Each mobile device moves in an arbitrary manner and routes are subject to frequent breakage [QIN03]. In software mobile network products, often the failure rate decreases after installation, eventually reaching a steady state. The time it takes for a product to reach its specified reliability depends on different product parameters. Stabilization time is the operating time during which specified reliability is achieved [SAU06]. In mobile devices, achieving stabilization is a function of parameters, such as quality of communication between mobile devices and between mobile devices and mobile network, as represented by S/N stability (i.e., $S/N >> 1$), in Figure 15.1.

Radio frequency (RF) interference, large-scale path loss, and fading cause adverse channel conditions by reducing the S/N of the wireless communications. When the S/N is lower than a certain threshold, the bit error rate of the wireless communication rises over the acceptable limit, thus disrupting the wireless connection. Therefore, the key to maintaining wireless communication under adverse channel conditions is to provide as high an S/N as possible [WAN07]. Furthermore,

this important parameter can be related to reliability, as will be shown later. Reliability becomes even more important as new critical applications emerge for mobile phones (e.g., robot control, traffic control, and telemedicine). In such scenarios, a phone failure affecting the application could result in a significant loss or hazard (e.g., a robot performing uncontrolled actions) [CIN07]. Thus, device mobility, designed to make devices less dependent on particular locations and resources, is essential, as illustrated in Figure 15.1.

Wireless Communication

Wireless communication must be maintained under adverse channel conditions. Wireless channel conditions are inherently more vulnerable than those of wireline communications due to the existence of problems such as multiple access contention, RF interference, large-scale signal path loss, and signal fading. Industrial environments make these problems worse due to large obstructions and possible electromagnetic interferences (EMI). An example is the EMI from electric welding or an electric motor that can last for hours or even days. Wireless local area networks (WLANs) require much higher reliability than wired local area networks (LANs) for office or home use. Most office or home wired LANS allow a few seconds or even minutes of adverse channel conditions. They just need to back off or shut down until the channel condition recovers and then retransmit. However, WLANs do not have the luxury of delay or shut down. Delay or shut down would cause deadlines to be missed, which result in poor reliability and performance [WAN07].

Reported Failure Data

In this section, failure data reported in the literature are analyzed. Failure data were obtained from the analysis of failure reports posted between January 2003 and March 2006. There were a total of 533 reports. Phone models from many major vendors are represented: Motorola, Nokia, Samsung, Sony Ericsson, LG, Kyocera, Audiovox, HP, Blackberry, Handspring, and Danger [CIN07]. Twenty-two point three percent (22.3%) of failure reports are from smartphones, although smartphones represented only 6.3% of the market share in 2005. This is attributed to the fact that smartphones: (1) have more complex architecture than voice-centric mobile phones and (2) are open for users to download and install third-party applications or develop their own applications, which results in high failure rates.

Data for this study were obtained from publicly available Web forums, where users post information on their experiences in using handheld devices. Symbian operating system (OS)-based smartphone failure data were collected from 25 phones (in Italy and the United States) over a period of 14 months [CIN07]. Key findings indicate that: (1) the majority of OS kernel failures are due to memory access violation errors (56%) and memory management problems (18%) and (2) users experience a failure (freeze or self-shutdown) every 11 days, on average [CIN07].

Failure Types

The following failure types are the way vendors classify failures [CIN07]:

- ***Freeze (Lock-Up or a Halting Failure).*** The device's output becomes constant, and the device does not respond to the user's input.
- ***Input Failure.*** User inputs have no effect on device behavior (e.g., device keys do not work).
- ***Output Failure.*** The device, in response to an input sequence, delivers an output sequence that deviates from the expected one. Examples include inaccuracy in battery charge indicator, ring or music volume different from the configured one, and event reminders going off at wrong times.
- ***Self-Shutdown (Silent Failure).*** The device shuts down itself, and no service is delivered to the user.
- ***Unstable Behavior (Erratic Failure).*** The device exhibits erratic behavior without any input from the user, (e.g., backlight flashing and device self-activation of applications).

Recovery Actions

A disruption due to the failure of one of the participating (e.g., mobile device) or intermediary (e.g., cellular network) systems typically results in the user having to restart the application, often at significant expense to both the user and to the service provider For mobile users accessing digital cellular networks, such disruptions occur frequently, as the wireless link is much less reliable than wired connections [VAN03]. Therefore, it is important to discuss and evaluate actions to recover from a device failure. Recovery actions are classified as follows [CIN07]:

- ***Service the Phone.*** The user has to bring the phone to a service center for assistance. Often, when the failure is firmware related (computer programming instructions that are stored in a read-only memory unit rather than being implemented through software), the recovery consists of either a master reset (all the settings are reset to the factory settings and the user's content is removed from the memory) or a firmware update (i.e., uploading a new version of the firmware).
- ***Reboot.*** The user turns off the device and then turns it on to restore the correct operation (a temporary corrupted state is cleaned up by the reboot).
- ***Remove Battery.*** Battery removal is mainly performed when the phone freezes. In this case, the phone often does not respond to the power on/off button. Battery removal can clean up a permanent corrupted state; however, this is a crude way to invoke power management. Improved power management is needed in mobile devices to increase their utilization [YUK03].

- ***Wait an Amount of Time.*** Often it is sufficient to wait for a certain amount of time to let the device deliver the expected service.
- ***Repeat the Action.*** Repeating the action is sometimes sufficient to get the phone working properly (i.e., the problem was transient).
- ***Optimistic Message Logging.*** In optimistic message logging, the task of logging mobile device messages is assigned to a centralized mobile station, so that in the event of a mobile device failure, the device may be able to recover the message from the centralized mobile station. A number of message-logging algorithms have been proposed to support fault tolerance of mobile computing systems. However, little attention has been paid to the optimistic message logging scheme. Optimistic message logging has a lower failure-free operation cost compared to other logging schemes [PAR02].
- ***Automated Failure Data Logger.*** There is still little understanding of how and why mobile phones fail or of the methods and techniques needed to gain such understanding. A well-established methodology to evaluate the reliability of operational systems and to identify its bottlenecks is *field failure data analysis*. However, today's smartphones do not have a means to detect and collect failures. A solution is the automated failure data logger. Upon failure detection, the logger gathers useful information, such as the phone's activity, the list of running applications, and error conditions in system and application modules. The technique has been implemented in Symbian OS smartphones. The main objective of the logger is to detect and record the occurrences of freezes and reboots. It is important to detect the status of the phone during a failure. For example, assume that a phone freezes when a text message is received. It is important to answer questions such as: (1) do we know that a text message was being received? (2) do we know whether some module failed? and (3) are we aware of other applications running during the failure that may have contributed to the freeze)? [WAN07]

Failure Severity

Failure severity is classified according to the user perspective and defines severity levels corresponding to the difficulty of the recovery action(s) [CIN07]:

- ***High.*** A failure is considered to be high severity when recovery requires the assistance of service personnel.
- ***Medium.*** A failure is considered to be of medium severity when the recovery requires reboot or battery removal.
- ***Low.*** A failure is considered to be of low severity if the device operation can be reestablished by repeating the action or waiting for a certain amount of time.

All failures occurring during emergency calls (e.g., 911) are high severity.

Table 15.1 Failure Frequency and Recovery Action Distribution: Fraction of Total Number of Failures

			Recovery Action					
Failure type	Severity	Service phone	Reboot	Remove battery	Wait for response	Repeat operation	Unreported	Totals
Freeze	Medium	0.0365	0.0236	0.0901	0.0429	0	0.0601	0.2532
Input failure	High	0.0064	0.0064	0.0021	0	0.0064	0.0086	0.0299
Output failure	Low	0.0687	0.088	0.0043	0.0064	0.0579	0.1373	0.3604
Self-shutdown	High	0.0665	0	0.0215	0.0043	0	0.0773	0.1696
Unstable behavior	High	0.0687	0.0172	0.0021	0.0021	0.0064	0.088	0.1845
Totals		0.2468	0.1352	0.1201	0.0557	0.0707	0.3713	

Table 15.1 shows the distribution of failures and recovery actions. Despite their high occurrence, output failures are low severity, since repeating the action is often sufficient to restore the device to the correct operation. On the other hand, self-shutdown and unstable behavior are high-severity failures because they must be corrected by servicing the phone or removing the battery. Phone freezes are medium severity, since rebooting only occurs in 2.36% of the total number of failures. While input failures are high severity because device keys do not work, their frequency of occurrence is low.

From the recovery action perspective, it should be noted that reboots are an effective way to recover from output failures (8.80% of the total number of failures). This indicates that output failures are often due to a temporary software corrupted state, which is cleaned up by the reboot. This is also confirmed by the fact that repeating the action is often sufficient to restore a correct device operation. Freezes are usually recovered by pulling out the battery (9.01%), even if a significant number of them (4.29%) are recovered by simply waiting for the phone to respond. This may indicate that a certain fraction of battery removals and reboots in response to freezes is due to impatient users. In general, this leads to the conclusion that freezes are more annoying than output failures.

Additionally, failure occurrences can be associated with the user activity at the time of the failure (not shown in Table 15.1). In particular, 13% of failures occur during voice calls, 5.4% while creating, sending, and receiving text messages, 3.6% while using Bluetooth, and 2.4% when manipulating images. Finally, several reports provide insight into the failure causes. There are indications of loss of memory data, incorrect use of the device resources, bad handling by software of indexes and pointers to objects, and incorrect management of buffer sizes.

Table 15.2 Expected Number of Failures, Failure Rates, and Recovery Rates

		Recovery Action						
Failure type	Service phone	Reboot	Remove battery	Wait for response	Repeat operation	Unreported	n Totals	Failure rate λ
Freeze	19.45	12.58	48.02	22.87	0.00	32.03	134.96	9.64
Input	3.41	3.41	1.12	0.00	3.41	4.58	15.94	1.14
Output	36.62	46.90	2.29	3.41	30.86	73.18	193.27	13.80
Self-shutdown	35.44	0.00	11.46	2.29	0.00	41.20	90.40	6.46
Unstable behavior	36.62	9.17	1.12	1.12	3.41	46.90	98.34	7.02
Totals	131.54	72.06	64.01	29.69	37.68	197.90	n	
Recovery rate	9.40	5.15	4.57	2.12	2.69	14.14		

Further elaboration of failure types and corresponding recovery actions are documented in Table 15.2, showing expected number of failures and failure rates for the various failure categories.

RELIABILITY CALCULATIONS

Probability of Failure

Using the data from Tables 15.1 and 15.2, and assuming that failures occur according to a Poisson distribution, we are able to calculate several interesting reliability metrics. The Poisson distribution is most often used in situations where the probability of the next state in a process is only dependent on the present state—the so-called memoryless systems. For example, when the probability of the next failure is only dependent on the present state (e.g., loss of battery power) of the mobile phone when a failure occurs [MUS87]. This is a reasonable assumption because, for example, prior calls by the mobile device should have no effect on the current failure probability of the device. Then the probability of x mobile phone failures occurring during operating time t, is given by Equation 15.1:

$$P(x,t) = \frac{e^{-\lambda t}(\lambda t)^x}{x!}, \tag{15.1}$$

where λ is the failure rate. Note that when x = 0, Equation 15.1 yields the classical reliability expression $R(t) = e^{-\lambda t}$.

Using the fraction of failures in each category in Table 15.1 and the 533 total failure reports that are available, you can compute the *expected* number of failures shown in Table 15.2 by multiplying the fraction of failures by 533. In addition, the

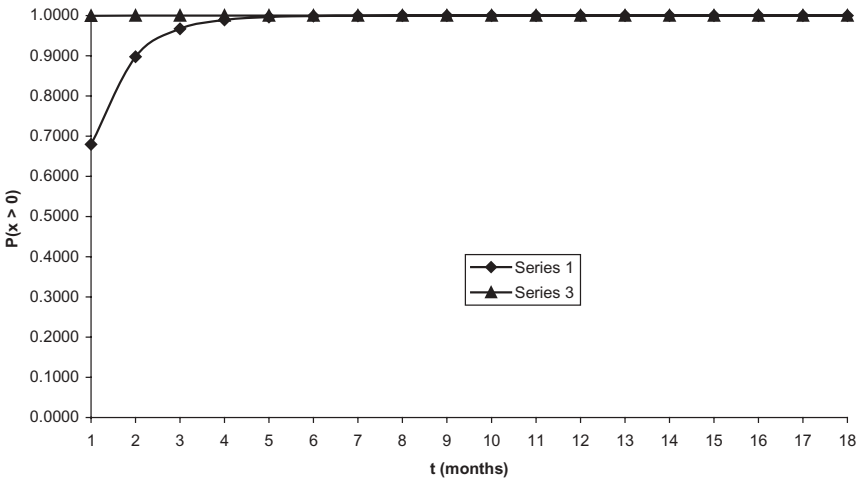


Figure 15.2 Mobile phone: probability of one or more failures P(x > 0) versus operating time t. Series 1: Input failure, high severity; expected number of failures = 15.94; failure rate = 1.14 failures per month. Series 3: Unstable behavior, high severity; expected number of failures = 98.74; failure rate = 7.02 failures per month.

failure rate was computed for each failure type and recovery action by $\lambda = n/t$, where n is the "Totals" column and row of Table 15.2 and t is equal to 14 months—the length of time during which the failure data was collected. With the failure rate in hand, you can compute the probability of one or more failures during the operational time for each failure type (i.e., *unreliability* at operational time t). This is done in Equation 15.2:

$$P(x > 0, t) = 1 - \left[\frac{e^{-\lambda t}(\lambda t)^x}{x!} \right]. \tag{15.2}$$

The result is shown in Figure 15.2, where two of the high severity failure categories are plotted. The figure indicates that over the life of a mobile phone—reports indicate that phones are discarded every 18 months—it is highly unlikely that there would be failure-free service for these failure categories. The same result was obtained for the other failure categories but it was infeasible to include them in the same figure. The results suggest that, given the fact that memory violations are the cause of the majority of the failures, vendors should provide better protection against memory violations, such as validity checks on memory access to ensure that the correct area of memory is being accessed.

Figure 15.3 shows an application of Equation 15.1 applied to the input failure category where an improvement in reliability is achieved by switching from a failed phone to a nonfailed phone in the backup network. This process is only possible in the case of an organization with multiple cell phone users at various locations, such that a user with a working phone can take over communication from a user with a failed phone.

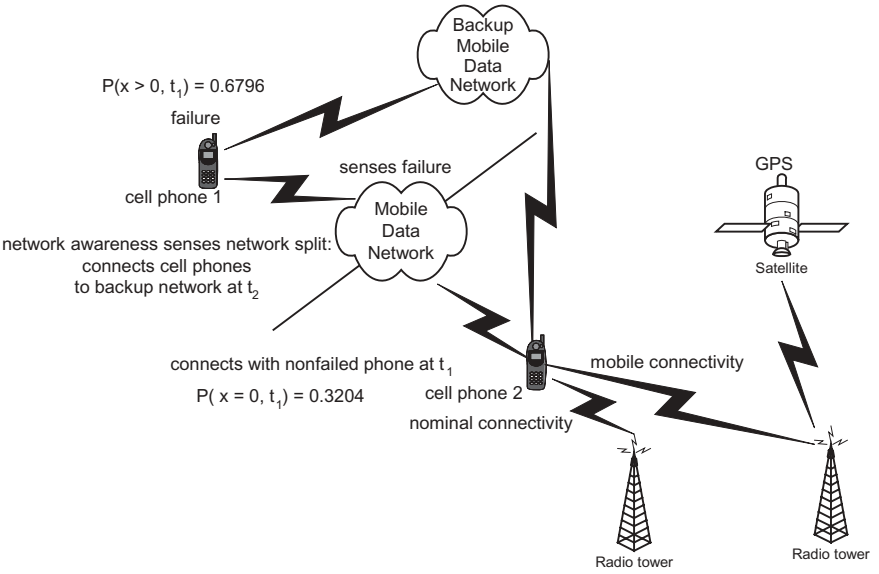


Figure 15.3 Context- and network-aware mobile data network. P(x, t), probability of x number of input failures in operational time t; t_1, cell phone 1: failure in 1 hour, cell phone 2: no failure; t_2, network split. Improvement in reliability = 0.3592. Cell phone 2 exhibits context awareness and mobile connectivity to obtain GPS location information.

Another perspective of failure characteristics is shown in Figure 15.4, where interestingly, you can identify the best customer strategy, based on using the expected number of failures and failure rates corresponding to the recovery action categories. Figure 15.4 shows that for mobile phone usage time of less than 4 months, the best customer action is to wait for a response from the phone. For extended usage (i.e., t > 4), none of the alternatives would be more advantageous than the others.

Stabilization Time

It is of interest to compute the operating time *during* which a specified reliability requirement is achieved. This is the stabilization time mentioned earlier that increases with decreasing failure rate for a specified reliability. This time is computed by solving $R(t) = e^{-\lambda t}$ for t, as follows:

$$t = \frac{-LN(R(t))}{\lambda}, \tag{15.3}$$

where R(t) is now the specified reliability.

Again using the high severity types of failures, Equation 15.3 is computed in Figure 15.5, where you can see that for *low failure rate, input failures*, the specified reliability is achieved for a *longer* stabilization time than for *high failure rate,*

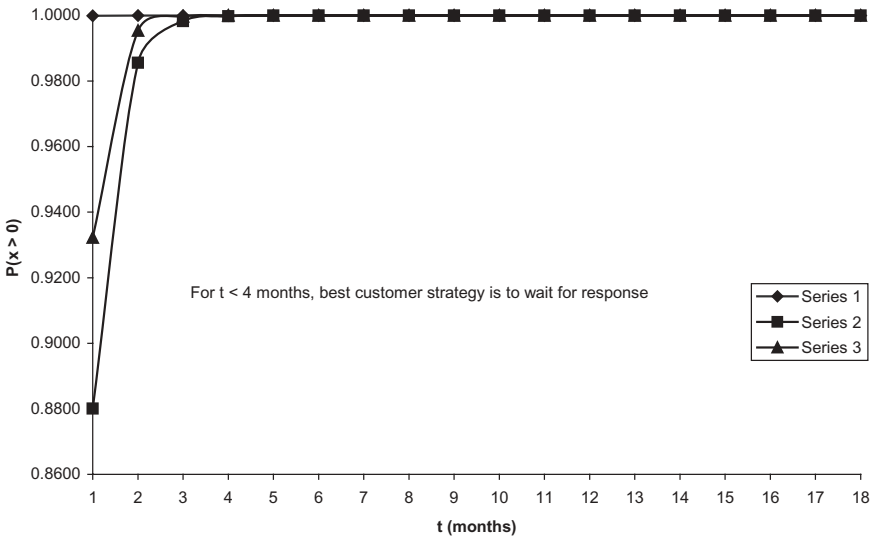


Figure 15.4 Mobile phone: probability of one or more failures related to recovery action P (x > 0) versus operational time t. Series 1: Service phone, expected number of failures = 131.54, failure rate = 9.40 failures per month. Series 2: Wait for response, expected number of failures = 29.69, failure rate = 2.12 failures per month. Series 3: Repeat operation, expected number of failures = 37.68, failure rate = 2.69 failures per month.

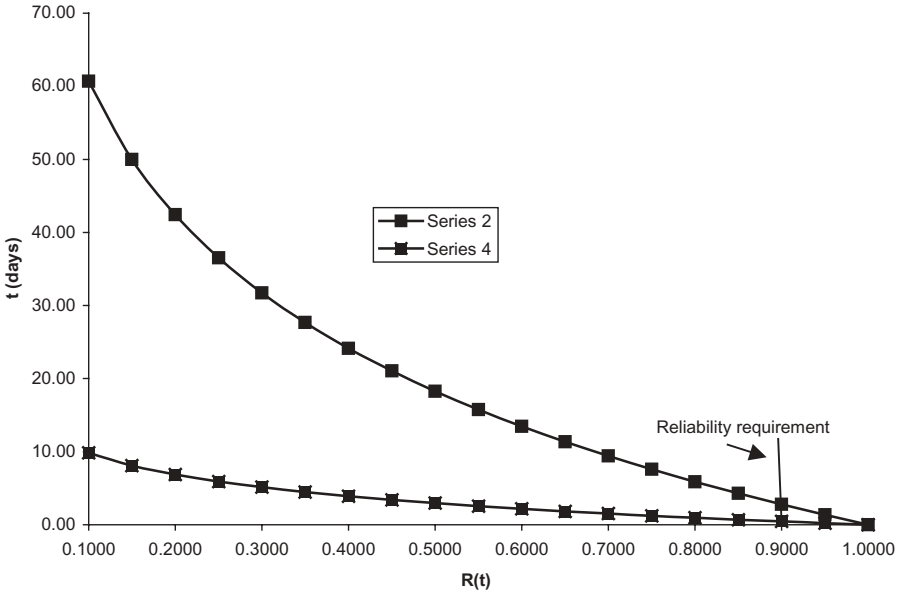


Figure 15.5 Mobile phone: stabilization time t versus specified reliability R(t) for failure types. Series 2: Input failure, high severity, failure rate = 1.14 failures per month, meets reliability requirement for 2.78 days of operation. Series 4: Unstable behavior, high severity, failure rate = 7.02 failures per month, meets reliability requirement for 0.45 days of operation.

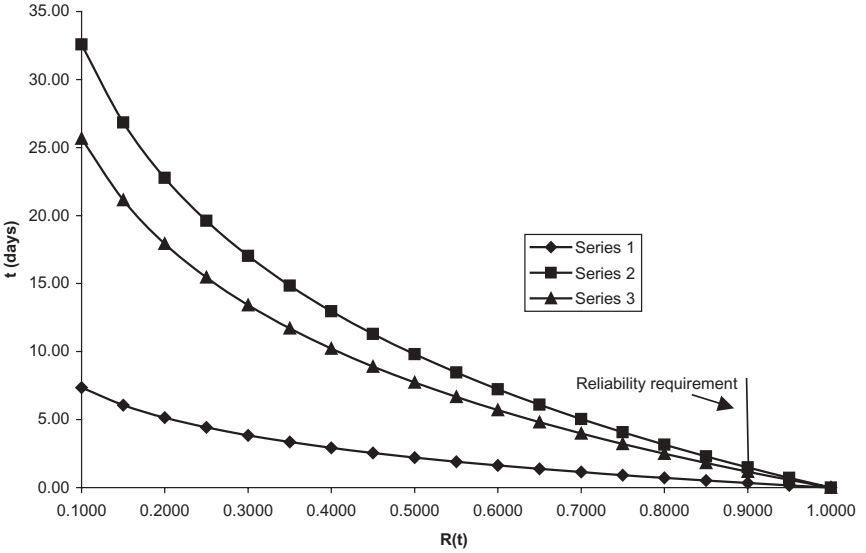


Figure 15.6 Mobile phone: stabilization time t versus specified reliability R(t) for recovery actions. Series 1: Service phone, failure rate = 9.40 failures per month, meets requirement for 0.34 days of operation. Series 2: Wait for response, failure rate = 2.12 failures per month, meets requirement for 1.49 days of operation. Series 3: Repeat operation, failure rate = 2.69 failures per month, meets requirement for 1.17 days of operation.

unstable behavior failures. Figure 16.6 presents the corresponding information for recovery actions, where the finding of Figure 15.4 is confirmed: the best customer strategy is to wait for a response. Doing so would result in the longest stabilization time of the various recovery actions.

MOBILE DEVICE CONTEXT AWARENESS

Information about the user's environment offers new opportunities to improve personalized applications. Such applications constantly need to monitor the environment (e.g., connectivity to access points, as identified by sending test signals)—called context—to allow the application to react according to this context. Context awareness is especially interesting in mobile scenarios where the context of the application is highly dynamic in which the application must deal with the constraints of presentation (e.g., small display screen) and communication restrictions (e.g., noisy signal propagation conditions) [HOF03]. With regard to location awareness, most WLANs positioning systems use received signal strength (RSS) as important information to estimate the location of a mobile station (see Fig. 15.1). RSS can be obtained at the access points or at the mobile device [YEU07].

A user who is moving with his or her mobile device is not permanently connected to a network, as depicted in Figure 15.1. In the case of WLAN, Bluetooth, or other wireless connections, if a user gets out of range of access points, the user is switched to other access points. Since permanent connections are not guaranteed, an application cannot rely on remote servers. This discontinuity of network connections has to be taken into account when designing an architecture for mobile devices. But the network connection is not the only difference of mobile versus nonmobile applications. Mobile devices are much more personal, meaning that the user of a mobile device seldom changes [HOF03]. This characteristic is a benefit in disguise because, unlike the case of nonmobile users, there is only a single user who can be the source of user-injected errors.

Regarding the need to save energy, a mobile device can be turned off, or some features can be disabled when they are not required. While this saves energy, a deactivated sensor cannot sense any information about the context and, therefore, the context cannot be determined until the sensor is turned on again [HOF03].

On the basis of these special characteristics of a mobile device scenario, the following requirements for an architecture to support context awareness on mobile devices have been identified:

- ***Lightweightness.*** The framework has to take into account the restrictions on limited processing power. For example, current IEEE 802.11 power saving schemes provide limited savings for Voice over Internet Protocol (VoIP) wireless traffic. A novel scheme named Adaptive Microsleep (AMS) can be applied to solve this problem. AMS is well suited for power saving on mobile VoIP devices by adapting to power needs. For example, when power requirements are low, the amount of time that devices spend in a low power sleep state is increased, but doing so without introducing additional delays that would noticeably deteriorate voice quality [CHA07].
- ***Extensibility.*** Since available sensors (i.e., a mobile device part that can sense information, such as connectivity with an access point) and extension slots are limited, it is not possible for a single device to sense all context information. Therefore, the architecture should support connections to sensors that are the most important for a given application.
- ***Robustness.*** The architecture has to guard against disconnections of remote sensors (e.g., sensor associated with access point).
- ***Context Sharing.*** Provide a mechanism for sharing context information (i.e., information about wireless environment [S/N]) with other mobile devices.

The *context architecture* comprises the following types of context:

- ***Time.*** The current time, as provided by the system clock of the mobile device.
- ***Location.*** Represents the current (physical) position of the mobile device using Global Positioning System (GPS) coordinates. This context is typically set by an adaptor, which reads a GPS receiver. This is illustrated in Figure 15.3 where cell phone 2 obtains GPS-provided location information.

- ***Device Identification.*** Consists of an identifier, which should be unique, and a device type, which can be used to distinguish between different types of devices, such as desktop PCs, laptops, and personal digital assistants (PDAs).
- ***User.*** Identifies the current user of the device [HOF03].
- ***Network.*** Contains information about the available network connection types of the device (e.g., access point, wired network). This context can provide additional information, such as the likelihood of the abort of a connection and the connection available bandwidth.
- Operating Context *(*OC*).* This is defined by the device hardware and software, the target user, and other characteristics, such as the communications carrier. For example, consider the following two OCs:
  - OC1: target device = Nokia N90 (defines device hardware and software), target user = subscribers of carrier A.
  - OC2: target device = Nokia N90 (defines device hardware and software), target user = subscribers of carrier B.

Note that both OCs are for the same physical device.

Interoperability of Mobile Devices with Other Computing Infrastructures

In the near future, personal mobile devices will become ingredients of other infrastructures, such as electric power grids. Computing techniques have been devised to enhance mobile devices so that their interoperability with grid infrastructures will be achieved by employing Personal Augmented Computing Environment (PACE). PACE characteristics include (1) collaborative mobile device visualization (e.g., electric utility and customer meter reading), (2) context-aware methods for mobile devices to achieve efficient utilization of grid resources (e.g., electric utility mobile device senses power outage and invokes backup power supply), and (3) integration of mobile devices and environmental infrastructures (e.g., amalgamation of electric utility and customer mobile devices with electric utility substation to achieve efficient power usage) [LUO07]. This development is important because it facilitates the production of common power utility software and common customer software, thus achieving software portability [MIK07].

Context-Aware Migratory Service

Due to the vagaries of context aware services, a model is needed of service interaction in ad hoc networks, based on the concept of context-aware migratory service. Unlike a regular service that always executes on the same node, a context-aware migratory service is capable of migrating to different nodes in the network in order to effectively accomplish its task. For example, a mobile device lacking connectivity to an access point can migrate to another access point. The service migration is

context aware because it is triggered by context changes of the nodes in the ad hoc network [RIV07]. An example is cell phone mobility in Figure 15.1, triggered by sensing an intrusion.

When failures occur, the mobile network has to try to find other devices to execute the mobile programs. If it is successful in finding such a device, it will transfer program control to that device [KUN]. You can consider the probability $P(x > 0)$ of one or more failures x of the failing device versus the probability of $x = 0$ failures $P(x = 0)$ of the nonfailing device, and compute the improvement in reliability. This process is illustrated in Figure 15.3.

NETWORK-AWARE APPLICATIONS

A network-aware application attempts to adjust its resource demands in response to network performance variations. In most current network-aware applications, changes in network environments refer to changes in the following parameters of network quality: bandwidth, which is the minimum link capacity among all the links from a source mobile device to a destination mobile device; throughput, which is measured in rate of data transfer [CAO04]; and reliability. When these parameters decrease, the network-aware application slows the utilization of resources to reflect the decrease in performance. Contrariwise, when parameters increase, resource utilization is increased.

MOBILE DEVICE PERFORMANCE

User-Perceived Response Time

Today, most personal mobile devices are multimedia enabled and support a variety of concurrently running applications, such as audio and video players, word processors, and Web browsers. Media-processing applications are often computationally complex. As a result, the user-perceived application response times are often poor when multiple applications are concurrently executed. By using application-specific buffering techniques, as shown in Figure 15.1, the workload of these applications can be "shaped" to fit the available processor bandwidth [CHA06].

Mobile Phone Performance Assessment

Performance is an important quality attribute of a software system but it is not always considered when mobile phone software is designed. Furthermore, software evolves and these changes can negatively affect performance. New requirements could introduce performance problems and the need for a different design. Performance assessment is a way to highlight design flaws or inefficiencies. Periodic performance assessments can help to discover potential bottlenecks [DEL04]. For example, in Figure 15.1, a potential bottleneck to accurately locate the Internet Protocol (IP)

phone is the RSS. This parameter can be tuned to avoid a bottleneck by adjusting receiver sensitivity in the phone.

Storage Capabilities

Mobile computing devices with several networking interfaces have become commonplace (e.g., text messaging, Internet Web sites). Networked data storage facilities greatly extend their use. The storage architecture for such devices is a critical performance factor. A two-level structure is used in which one component, the mobile memory cache, moves when the device is mobile [MAP07], as illustrated in Figure 15.1. In addition, there is a fixed location secondary storage component that is capable of storing large amounts of mobile network data.

Signal-to-Noise Ratio

As mentioned previously, and as shown in Figure 15.1, S/N is an important performance attribute of a mobile network. Data from mobile device vendors about actual S/N are not available for this analysis; however, surrogates in Equation 15.4 are based on the ratio of reliability (R(t) = signal) to unreliability (U(t) = noise):

$$S/N = R(t)/U(t). \tag{15.4}$$

Using this metric, you can see in Figure 15.7 that the mobile phone customer would not enjoy a good S/N for more than 4 months of usage.

Problem: Is there a limitation to computing S/N as shown above, and if so, what is the limitation? Answer the question by formulating an equivalent equation for S/N.

Solution: To obtain the answer, compute S/N as follows:

$$\frac{S}{N} = \frac{R(t)}{U(t)} = \frac{R(t)}{1-R(t)} = \frac{1}{\dfrac{1}{R(t)}-1}.$$

Thus, S/N is only a function of reliability (i.e., signal) instead of signal *and* noise! However, given the lack of signal and noise data, this is the best we can do.

SUMMARY AND CONCLUSIONS

In conformance with the chapter objective, a variety of mobile device issues have been addressed that differ dramatically from those of wired networks. Where data

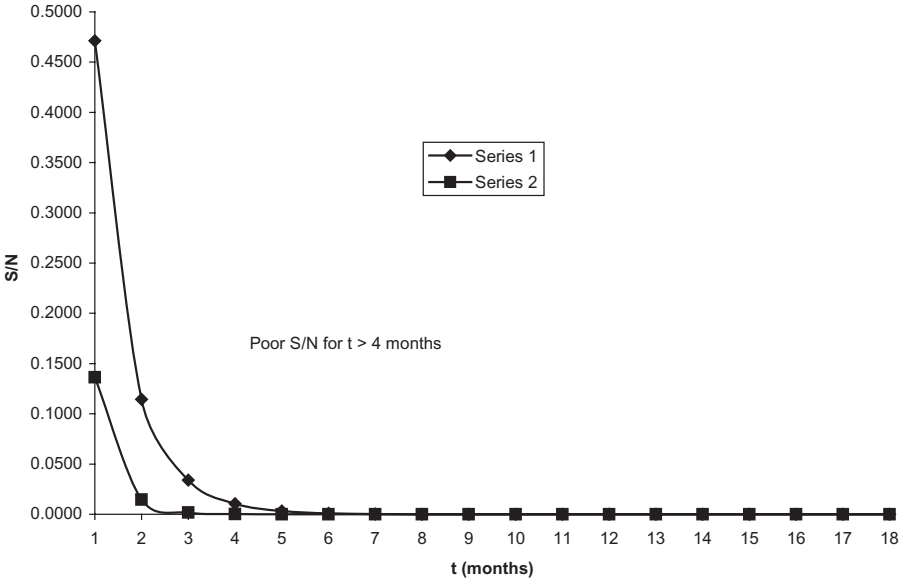


Figure 15.7 Mobile phone: signal-to-noise ratio S/N versus operational time t. Series 1: Input failure, high severity, failure rate = 1.14 failures per month. Series 2: Wait for response, failure rate = 2.12 failures per month.

were available, as in the case of reliability, quantitative methods were employed to assess reliability. A conclusion based on this analysis is that mobile device reliability is only satisfactory for the first few months of operation. If significant advances are to be made in reliability, it will be necessary to make improvements in both hardware and software reliability, particularly as it relates to memory management.

Where quantitative data were not available for issues such as context and network awareness in mobile networks, it was shown how mobile networks can adapt to changing conditions, such as a network split in Figure 15.3, using a network-awareness approach. There is an urgent need for further research centered on collecting and analyzing reliability and performance data related to context and network awareness because these applications represent the greatest potential for improvement in mobile networks, supporting, for example, the intelligent use of resources in an electric grid network.

REFERENCES

[CHA06] Samarjit Chakraborty, Balaji Raman, "Application-specific workload shaping in multimedia-enabled personal mobile devices," *Proceedings of the 4th International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS'06)*, 2006, pp. 4–9.

[CHA07] Shuvo Chatterjee, Shuvo Chatterjee, Shuvo Chatterjee, Shuvo Chatterjee, Dietrich Falkenthal, Dietrich Falkenthal, Dietrich Falkenthal, Dietrich Falkenthal, Tormod Ree, Tormod Ree, Tormod Ree, Tormod Ree, "Exploring adaptive power saving schemes for mobile VoIP devices in IEEE 802.11 networks," *Second International Conference on Digital Telecommunications (ICDT'07)*, 2007, p. 13.

[COR06] R. Cornea, A. Nicolau, N. Dutt, "Software annotations for power optimization on mobile devices," *Proceedings of the Design Automation & Test in Europe Conference* Volume 1, 2006, p. 148.

[CIN07] M. Cinque, D. Cotroneo, Z. Kalbarczyk, and R. K. Iyer, "How do mobile phones fail? A failure data analysis of Symbian OS smart phones," *37th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN'07)*, 2007, pp. 585–594.

[DEL04] Christian Del Rosso, "The process of and the lessons learned from performance tuning of a product family software architecture for mobile phones," *Eighth Euromicro Working Conference on Software Maintenance and Reengineering (CSMR'04)*, 2004, p. 270.

[ESA98] T. Escamilla, *Intrusion Detection: Network Security beyond the Firewall*. New York: John Wiley & Sons, 1998.

[HOF03] Thomas Hofer, Wieland Schwinger, Mario Pichler, Gerhard Leonhartsberger, Josef Altmann, Werner Retschitzegger, "Context-awareness on mobile devices - the hydrogen approach," *36th Annual Hawaii International Conference on System Sciences (HICSS'03)—Track 9*, January 2003, p. 292a.

[KUN] Christian P. Kunze, Sonja Zaplata, Mirwais Turjalei, and Winfried Lamersdorf, "Enabling context-based cooperation: a generic context model and management system," Distributed Systems and Information Systems, Computer Science Department, University of Hamburg, Hamburg, Germany, http://vsis-www.informatik.uni-hamburg.de/getDoc.php/publications/314/BIS2008_kztl08.pdf

[LUO07] X. Luo, "PACE: augmenting personal mobile devices with scalable computing," *Seventh IEEE International Symposium on Cluster Computing and the Grid (CCGrid '07)*, 2007, pp. 875–880.

[MAP07] Glenford Mapp, Dhawal Thakker, David Silcott, "The design of a storage architecture for mobile heterogeneous devices," *International Conference on Networking and Services (ICNS'07)*, 2007, p. 41.

[MIK07] Tommi Mikkonen, *Programming Mobile Devices: An Introduction for Practitioners*. Hoboken, NJ: John Wiley & Sons, Ltd., 2007.

[MUS87] John D. Musa, Anthony Iannino, and Kazuhira Okumoto, *Software Reliability: Measurement, Prediction, Application*. New York: McGraw-Hill, 1987.

[PAR02] Taesoon Park, Namyoon Woo, Heon Y. Yeom, "An efficient optimistic message logging scheme for recoverable mobile computing systems," *IEEE Transactions on Mobile Computing*, 2002, 1(4), pp. 265–277.

[QIN03] Liang Qin, Thomas Kunz, "Increasing packet delivery ratio in DSR by link prediction," *36th Annual Hawaii International Conference on System Sciences (HICSS'03)—Track 9*, 2003, p. 300a.

[RIV07] Oriana Riva, Tamer Nadeem, Cristian Borcea, and Liviu Iftode, "Context-aware migratory services in ad hoc networks," *IEEE Transactions On Mobile Computing*, 2007, 6(12).

[SAU06] Vibhu Saujanya Sharma, Pankaj Jalote, "Stabilization time—a quality metric for software products," *17th International Symposium on Software Reliability Engineering (ISSRE'06)*, 2006, pp. 45–51.

[TRE] http://us.trendmicro.com/imperia/md/content/us/pdf/products/enterprise/mobilesecurity/wp05_tmms_080211us.pdf

[VAN03] Debra VanderMeer, Anindya Datta, Kaushik Dutta, Krithi Ramamritham, Shamkant B. Navathe, "Mobile user recovery in the context of Internet transactions," *IEEE Transactions on Mobile Computing*, 2003, 2(2), pp. 132–146.

[WAN07] Qixin Wang, Xue Liu, Weiqun Chen, Lui Sha, and Marco Caccamo, "Building robust wireless LAN for industrial control with the DSSS-CDMA cell phone network paradigm," *IEEE Transactions on Mobile Computing*, 2007, 6(6), pp. 706–719.

[YE02] Nong Ye, Syed Masum Emran, Qiang Chen, Sean Vilbert, "Multivariate statistical analysis of audit trails for host-based intrusion detection," *IEEE Transactions on Computers*, 2002, 51(7), pp. 810–820.

[YEU07] Wilson M. YEUNG, Joseph K. NG, "Wireless LAN positioning based on received signal strength from mobile device and access points," *13th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA 2007)*, 2007, pp. 131–137.

[YUK03] Yukikazu NAKAMOTO, "The next generation software platform for mobile phones," *Sixth IEEE International Symposium on Object-Oriented Real-Time Distributed Computing (ISORC'03)*, 2003, p. 46.

Chapter 16

Signal-Driven Software Model for Mobile Devices

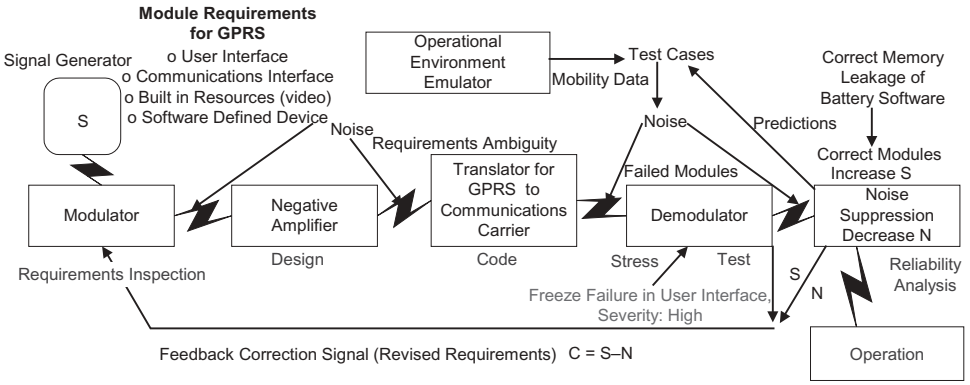
There is a paucity of software models that deal with mobile devices. Therefore, the motivation for this chapter is to build on the mobile device material learned in Chapter 15 and apply it to the development of important mobile device software models. Issues in mobile device reliability are explored, using a signal-driven software model for mobile devices. Based on mobile phone failure data reported in the literature, the model is implemented in two dimensions: by type of failure and by type of failure recovery action. Based on these calculations, it can be concluded that current mobile devices are unable to provide highly reliable service for more than a few months of operation. In addition, a novel signal-to-noise ratio (S/N) representation of reliability is developed and applied to the failure and recovery action data. Having discovered that S/N influences test effectiveness, it can be shown that S/N can be used to prioritize software modules for testing.

INTRODUCTION

You may ask: what is so important about signal-driven software models? How do they differ from plain old input-driven software models? Well, there are significant differences because mobile devices operate in a hostile communications environment, confronted by adverse atmospheric conditions and physical barriers to signal propagation. Thus, software models for mobile devices must take these conditions into account.

The key issues in mobile computing include mobility-related reliability and testing problems, such as loss or degradation of wireless connections, high latency wireless networks, and low quality connections (e.g., caused by network failures) [POU06]. To address the reliability issue, a signal-driven software model for mobile devices is developed and shown in Figure 16.1. In the physical mobile device system, signal strength is critical to effective communication. Mobile devices use

Computer, Network, Software, and Hardware Engineering with Applications, First Edition. Norman F. Schneidewind.
© 2012 the Institute of Electrical and Electronics Engineers, Inc. Published 2012 by John Wiley & Sons, Inc.



S/N = Signal/Noise = # Correct Modules/# Failed Modules

Reliability = 1–(N/S)

Figure 16.1 Signal-driven software model for mobile devices.

received signal strength to estimate the location of other devices and stations [YEU07]. Analogously, signal strength in a software model is important because, with high *signal* strength S, representing the number of correct software modules, and low *noise* strength N, representing the number of failed software modules, reliability will be increased, as shown in Figure 16.1, by *decreasing* the ratio N/S. This formulation of signal-to-noise ratio (S/N) may seem strange because traditionally, signal and noise are represented by electrical signals. This is appropriate for hardware that can process signals represented by voltage, but, for software, a representation is needed that pertains to the correctness of the software production process.

Mobile Device Software Reliability and Testing

In software products, often the failure rate decreases after installation, eventually reaching a steady state. The time it takes for a product to reach its steady-state reliability depends on different product parameters. Stabilization time is the time taken after installation for the reliability of the product to asymptotically approach a constant value [SAU06]. In mobile devices, achieving stabilization is a function of parameters, such as quality of communication between mobile devices and between mobile devices and mobile network, as represented by S/N. Interestingly, S/N does not have to be limited to measurement in physical systems. It can be used as a measure of software reliability in Figure 16.1.

Reliability becomes even more critical as new applications emerge for mobile phones (e.g., robot control, traffic control, and telemedicine). In such scenarios, a device failure affecting the application could result in a significant loss or hazard (e.g., a robot performing uncontrolled actions) [CIN07]. The challenges to reliability are amplified because of the problems with current-generation wireless technologies.

Whether it be due to signal dead zones, environmental conditions, a crowded concentration of mobile devices, or simply a device going offline to save data charge costs, mobile wireless devices do not have the same communication reliability as their wired counterparts [LAR07].

Testing Challenges

Users have high expectations for the reliability of the software on mobile devices. Users require devices to be reliable and stable. They will not be comfortable with a mobile device that crashes and loses personal data. This requires the device manufacturer and software vendors to guarantee the high quality of their products. Testing is their most important tool. The testing of mobile devices is difficult because the environment is complex**.** To be effective, the execution of tests should interact with end users, wireless signals**,** and the wireless network. The diversity of mobile devices reduces the reusability of test cases. The devices are highly resource constrained in terms of processing and communication ability and in memory capacity; test plans must recognize these constraints.

The testing approach must be highly nonintrusive to the mobile device environment in order that test results reflect realistic operating conditions. Also, device behavior is highly interactive. The devices constantly accept activations from users and send responses back for the user to take further action. Since it is difficult to predict a user's actions, many of the usage scenarios are difficult to automate.

The development of software for mobile computing devices is very difficult due to the limited computational resources of these devices (i.e., a great deal of functionality must be squeezed into a small memory space). This highly compact functionality must be reflected in the testing strategy (i.e., testing must be performed in the constrained memory of the mobile device, not in the software development platform). Furthermore, mobile device tasks are susceptible to errors because changes in network connectivity and locations may lead to sudden and unpredictable changes. A change in mobile network and mobile device location may imply movement away from the servers currently in use, and toward new ones. For example, a handheld computing device with a short-range radio link, such as IEEE 802.11b or Bluetooth, carried across the floors of an office building, may have access to different resources, such as printers and directory information for visitors, on each floor. Therefore, to construct reliable application software, the developer must test it in the operating environment of the mobile devices [SAT03]. However, it is impractical to physically visit all the places where the device may operate. Therefore, it is necessary to emulate the operational environment as shown in Figure 16.1.

Another testing challenge is to include the number of active users connected to mobile networks. This is an important aspect that affects the reliability of the connection and the performance of the device, as perceived by the user. More active users lead to fewer available communication timeslots, which decreases the throughput per user and, as a result, the latency increases [HOL06]. Latency is defined as

the time required for the data signal to be transmitted through the communications medium [ROU04]. Latency is the reciprocal of data rate that is tabulated for mobile devices in Table 16.1. A queuing model—not covered in this chapter—could be developed, using message transmission rate as queue arrival rate, and a variable number of active users, in order to estimate latency during performance testing.

MOBILE DEVICE CHARACTERISTICS

Mobile devices have unique characteristics that must be taken into consideration when modeling reliability and testing. These characteristics are the following [FIT07]:

Static versus Dynamic

The static part of the mobile device is its hardware; the dynamic part that can respond to changing operating conditions, is its software. The industry is developing a software-defined device that can be dynamically defined in real time. The software-defined device provides needed functionality (e.g., short-range to long-range communication). The specification of requirements for such a device is suggested in Figure 16.1.

Interfaces

Interfaces have been a major source of failures in computer systems because the joining of disparate components of a system is a complex process, subject to many failures, for example the interface between a mobile device and a mobile network. Thus, interfaces represent the major software modules to be developed by the mobile device process in Figure 16.1: user interface (e.g., user keying of mobile device), communication interface (e.g., mobile device to cellular communication), and built-in resources (e.g., interface between mobile device built-in video reception and its display).

Communication Systems

Various mobile device characteristics are tabulated in Table 16.1. Assembling these data helps us to identify a mobile device technology with a relatively low reliability rating, appropriate for applying a reliability model for worst-case analysis—General Packet Radio Service (GPRS). The reliability analysis of GPRS has general applicability because it is used in several applications. An important characteristic of mobile devices is that a given device may communicate with more than one communications carrier [COM]. Thus, in Figure 16.1 we show GPRS requirements being translated to software code compatible for operating with a communications carrier.

Table 16.1 Mobile Device Communication Characteristics

Generation	Type	Data rate	Network access	Number of time slots per user	Network communication	Error characteristics	Reliability rating	Application
2G	CSD	9.6 kbps	TDMA	1	Circuit switching	No error correction	Low	Cell phone
2G	HSCSD	9.6 kbps	TDMA	1	Circuit switching	Full error correction	High	Cell phone
2G	HSCSD	38.4 kbps	TDMA	1	Circuit switching	Partial error correction	Moderate	Cell phone
2G	HSCSD	57.6 kbps	TDMA	4	Circuit switching	No error correction	Low	Cell phone
2.5G	GPRS	171.2 kbps	TDMA	8	Packet switching	Poor performance in fully loaded cells	**Low**	Cell phone, e-mail, Web browsing
2.5G	EDGE	473.6 kbps	TDMA	8	Modulation	Unknown	Unknown	Cell phone, e-mail, Web browsing
3G	UMTS	384 kbps	WCDM	8	Packet switching	Unknown	Unknown	Cell phone, e-mail, Web browsing

CSD, circuit switched data; HSCSD, high-speed circuit switched data; EDGE, enhanced data rates for GSM evolution; UMTS, universal mobile telecommunications system; TDMA, time division multiple access; WCDM, wide band code division multiplexing.

In order to understand how mobile devices fail and the consequent recovery actions, the following definitions are provided in the succeeding sections, followed by elaborations of failure and recovery action characteristics [CIN07].

Failure Types

- ***Freeze (Lock-Up or a Halting Failure).*** The device's output becomes constant, and the device does not respond to the user's input.
- ***Input Failure.*** User inputs have no effect on device behavior (e.g., device keys do not work).
- ***Output Failure.*** The device, in response to an input sequence, delivers an output sequence that deviates from the expected one. Examples include inaccuracy in battery charge indicator, ring or music volume different from the configured one, and event reminders going off at wrong times.
- ***Self-Shutdown (Silent Failure).*** The device shuts itself down, and no service is delivered at the user.
- ***Unstable Behavior (Erratic Failure).*** The device exhibits erratic behavior without any input inserted by the user (e.g., backlight flashing and self-activation of applications). Unstable behavior can be caused by programming errors induced by the trend toward integration of complete systems on a chip that requires the placement of larger and larger chips into complex and small mobile devices [ZAN93].

Recovery Actions

User-initiated actions to recover from a device failure can be classified according to the following categories:

- ***Service the Phone.*** The user has to bring the phone to a service center for assistance. Often, when the failure is firmware related, the recovery consists of either a master reset (all the settings are reset to the factory settings and the user's content is removed from the memory) or a firmware update (i.e., uploading a new version of the firmware). Firmware is software instructions embodied in a read-only memory as opposed to using a read–write memory. Problems are fixed by substituting malfunctioning components (e.g., screen, keypad, firmware) or by replacing the entire device with a new one.
- ***Reboot (Reset the Mobile Device).*** The user turns off the device and then turns it on to restore the correct operation (a temporary corrupted state is cleaned up by the reboot). Related to reboot is a panic event. A panic event represents a nonrecoverable error condition signaled to the mobile device operating system kernel by either the user or by applications. Information associated with a panic event is delivered to the operating system kernel, which decides on the recovery action (e.g., system reboot).

- ***Remove Battery.*** Battery removal is mainly performed when the phone freezes. In this case, the phone often does not respond to the power on/off button. Battery removal can clean up a permanent corrupted state (e.g., corrupted memory contents); however, this is a crude way to invoke power management. Improved power management is needed in mobile devices to increase their utilization [YUK03]. Battery problems can be mitigated by using a power-saving technique that increases the amount of time devices spend in a low power sleep state, but doing so without introducing additional delays that would noticeably degrade performance [CHA07].
- ***Wait for a Response.*** Often it is sufficient to wait for a certain amount of time to let the device deliver the expected service.
- ***Repeat the Action.*** Repeating the action is sometime sufficient to get the mobile device working properly (i.e., the problem was transient).

Failure Severity

Failure severity is classified according to the user perspective and defines severity levels corresponding to the difficulty of the recovery action(s).

- ***High.*** A failure is considered to be of high severity when recovery requires the assistance of service personnel.
- ***Medium.*** A failure is considered to be of medium severity when the recovery requires reboot or battery removal.
- ***Low.*** A failure is considered to be of low severity if the device operation can be reestablished by repeating the action or waiting for a certain amount of time.

Failure Characteristics

Mobile device failure characteristics are compiled in Table 16.1. Despite their high occurrence, output failures are low severity, since repeating the action is often sufficient to restore the device to correct the operation. On the other hand, self-shutdown and unstable behavior are considered to be high-severity failures because they must be corrected by servicing the phone or removing the battery. Phone freezes are medium severity, since rebooting occurs only in 2.36% of the freeze failures. While input failures are high severity when device keys do not work, their frequency of occurrence is low.

Additionally, failure occurrences can be associated with the user activity at the time of the failure. In particular, 13% of failures occur during voice calls, 5.4% while creating, sending, and receiving text messages, 3.6% while using Bluetooth, and 2.4% when manipulating images. Finally, the history of mobile device usage indicates that there are memory leaks (i.e., loss of data in mobile device memory), incorrect use of the device resources (e.g., excessive activation of wireless com-

munication links by the user), bad handling by software of pointers to mobile device instructions and data, and incorrect management of buffer sizes (e.g., too little memory space allocated to buffers, resulting in buffer overflow).

Recovery Action Characteristics

From the recovery action perspective, reboots are an effective way to recover from output failures (8.80% of the total number of failures). This indicates that output failures are often due to a temporary software corrupted state, which is cleaned up by the reboot. This is also confirmed by the fact that repeating the action is often sufficient to restore a correct device operation. Freezes are usually recovered by pulling out the battery (9.01%) or recovered by simply waiting for the phone to respond (4.29%). This may indicate that a certain fraction of battery removals and reboots in response to freezes is due to frustrated user actions.

Probability of Failure and Recovery Action

The empirical failure probability for different types of failures and the probability of recovery action, given a failure, are shown in Table 16.2 [YUK03]. These probabilities allow us to estimate both the signal (number of correct modules) and noise (number of failed modules) in Figure 16.1 and, hence, the reliability of a mobile device. For the purpose of illustration, assume that the data in Table 16.2 apply to

Table 16.2 Probability of Failure and Corresponding Recovery Action

		Probability of recovery action given a failure						
		Recovery action						
Failure type	Severity	Service phone	Reboot	Remove battery	Wait for response	Repeat operation	Unreported	Probability of failure
Freeze	Medium	0.0365	0.0236	0.0901	0.0429	0	0.0601	0.2532
Input failure	High	0.0064	0.0064	0.0021	0	0.0064	0.0086	0.0299
Output failure	Low	0.0687	0.088	0.0043	0.0064	0.0579	0.1373	0.3626
Self-shutdown	High	0.0665	0	0.0215	0.0043	0	0.0773	0.1696
Unstable behavior	High	0.0687	0.0172	0.0021	0.0021	0.0064	0.088	0.1845
	Totals							
Probability of recovery action		0.2468	0.1352	0.1201	0.0557	0.0707	0.3713	0.9998

the GPRS mobile device whose communication characteristics were defined in Table 16.1. The reason for this is that, as mentioned previously, GPRS has a relatively low reliability rating (see Table 16.1). Thus, it would be interesting to focus on this device.

MOBILE DEVICE RELIABILITY MODEL

Focus on Failure Type

Using the mobile device empirical probabilities of failure and recovery in Table 16.2, construct a simple model of reliability based on the S/N shown in Figure 16.1, focusing on failure type. This analysis will allow you to understand the relationship between failures, recovery actions, and reliability. Assuming the probabilities P_{ij} of a recovery action of type j, given a failure of type i, are independent, the probability of a failure of type i is estimated in Equation 16.1. This equation sums the probability of a failure of type i over all recovery actions n. Regarding the assumption of independence, we have no reason to believe, for example, that rebooting, as the result of a freeze failure, depends on servicing the device:

$$P_i = \sum_{j=1}^{n} P_{ij}. \tag{16.1}$$

Failure severity is reflected in the model according to the following severity codes that were defined earlier: severity *high*, code = 3; severity *medium*, code = 2; and severity *low*, code = 1. Then, the expected number of failed modules of failure type i, where M is the total number of modules in a mobile device, is given by Equation 16.2; this is the noise factor N in Figure 16.1:

$$N_i = P_i sM. \tag{16.2}$$

The total number of modules M must be equal to the number of correct modules S_i (signal) plus the number of failed modules N_i (noise). Thus the signal S, based on failure type i, is computed in Equation 16.3:

$$S_i = M - N_i = (1 - P_i s)M. \tag{16.3}$$

As indicated in Figure 16.1, reliability R_i of a mobile device when failures of type i occur is related to the inverse of the S/N (i.e., unreliability). Using this fact and Equations 16.2 and 16.3, produce Equation 16.4:

$$R_i = 1 - (N_i/S_i) = 1 - (P_i sM/(1 - P_i s)M), \text{ for } R_i \geq 0. \tag{16.4}$$

Again, using Equations 16.2 and 16.3, the S/N, based on failure type i, is computed as follows:

$$S_i/N_i = (1 - P_i s)M/P_i sM = (1 - P_i s)/P_i s. \tag{16.5}$$

It is also important to estimate the feedback control signal C_i in Figure 16.1 that refers to the difference between number of correct modules (S_i) and number of failed modules (N_i) for failure type i. This feedback would be used to revise mobile device requirements for the purpose of driving N_i to 0 and C_i to = S_i. This idea is implemented for failure type i in Equation 16.6. However, if C_i is negative, it indicates that there is more noise than signal and that the software is in need of significant software development process improvement to reduce failures of type i.

$$C_i = S_i - N_i. \tag{16.6}$$

Focus on Recovery Action Type

Again, using the mobile device empirical probabilities of failure and recovery, P_{ij}, in Table 16.2, construct another part of the reliability model based on the S/N shown in Figure 16.1, focusing on recovery action type j. In this case, estimate P_j in Equation 16.7 as the probability of failure across the failure types i associated with a given recovery action type j:

$$P_j = \sum_{i=1}^{f} P_{ij}, \tag{16.7}$$

where f is the number of failure types.

Whereas previously the interest was in assessing reliability as a function of failure type, now the focus is on reliability as a function of recovery action type. This assessment identifies which recovery actions produce the highest reliability. Analogues to the failure type analysis, develop the reliability and S/N equations for recovery type as follows.

Since the expected recovery action is a probabilistic function of the failure types, compute a weighted sum of the probabilities P_{ij}, weighed by the failure severity code s_{ij}. The idea, as in the case of failure types, is to represent failure severity in the computation of expected number of failed modules, where now severity reflects both failure type and recovery type. Thus, the expected number of failed modules of recovery type j, where M is the total number of mobile device modules, and f is the total number of failures, is given by Equation 16.8:

$$N_j = M\sum_{i=1}^{f} P_{ij}s_{ij}, \text{ for } j = 1, \ldots, n. \tag{16.8}$$

In order to compute the signal, use the fact that M must equal the sum of correct modules S_j and failed modules N_j. Thus the signal S_j, the number of correct modules, based on recovery action j, is computed in Equation 16.9 by using Equation 16.8:

$$S_j = M - N_j = M\left(1 - \sum_{i=1}^{f} P_{ij}s_{ij}\right). \tag{16.9}$$

Reliability R_j, based on recovery action type j, is related to the inverse of the S/N, as follows:

$$R_j = 1 - (N_j / S_j) = 1 - \left(\frac{M \sum_{i=1}^{f} P_{ij} s_{ij}}{(M - N_j)} \right), \text{ for } R_j \geq 0. \tag{16.10}$$

In addition, the S/N, based on recovery type j and reliability R_j, from Equation 16.10, is computed as follows:

$$S_j/N_j = 1/(1 - R_j). \tag{16.11}$$

Again it is important to estimate the difference that exists between correct modules and failed modules. This feedback would be used to revise requirements for the purpose of driving the noise N_j to 0 and C_j to be equal to the signal S_j. This idea is implemented in Equation 16.12 for recovery action type j in Equation 16.12. However, if C_j is negative, it indicates that there is more noise than signal and that the recovery method j is dysfunctional and that other recovery options should be considered, for example central server monitoring of the health of the mobile device.

$$C_j = S_j - N_j. \tag{16.12}$$

Model Limitations

In revising software requirements, it is important to recognize that there are both explicit and implied requirements [MCC02]. In the case of mobile devices, this is a tricky issue because there is no direct developer–customer relationship (i.e., developers produce for a mass mobile device market). Almost all requirements are implicit (e.g., able to connect to a mobile network on demand) as opposed to explicit requirements, such as the obvious one of having power when the device is turned on. This issue illustrates the fact that there are aspects of mobile device development that cannot be quantified in a model, such as the one in Figure 16.1. For example, noise in the figure, representing requirements ambiguity, and quantified as number of failed modules, may capture power failure but not unsatisfactory Web search results.

Another limitation of the model is the absence of workload in Figure 16.1. Measurements show that software reliability results cannot be considered representative unless the system workload is taken into account [IYE85]. For example, the reliability of a mobile device will decrease nonlinearly with the amount of interactive processing (e.g., number of simultaneous mobile network connections). This property could be simulated but is difficult to address in an analytical model.

Question for Reader: It was stated above that S/N would not be an appropriate metric for evaluating Web search results if noise is represented by number

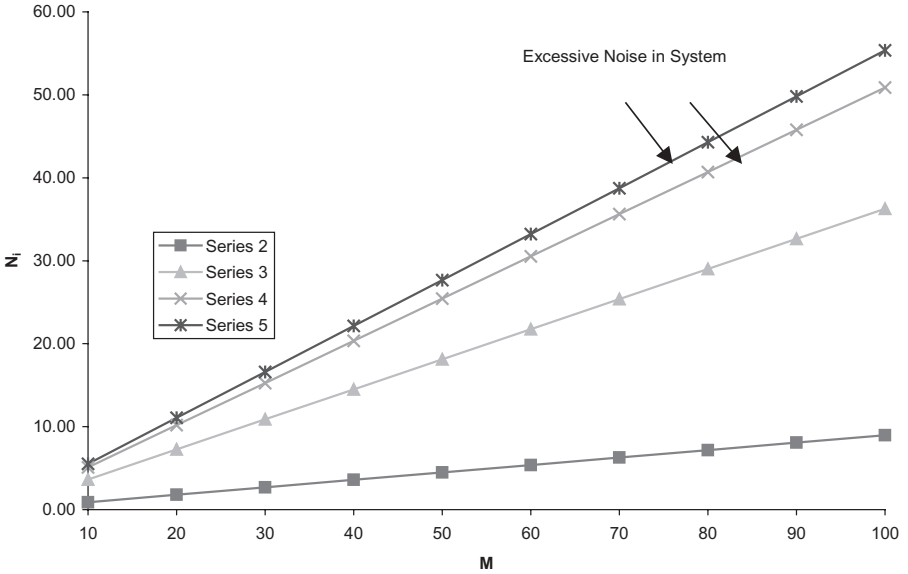


Figure 16.2 Noise N_i (number of failed modules) versus total number of modules M. Failure types: Series 2: Input: high severity; Series 3: Output: low severity; Series 4: Self-shutdown: High severity, freeze: medium Severity; Series 5: Unstable behavior: high severity.

of failed modules. That being the case, is there an appropriate metric that uses S/N for evaluating Web search results?

Answer: Signal could be represented by the number of successful Web search results and noise could be represented by the number of unsuccessful Web search results. The Web search application objective would be to maximize the S/N.

Failure-Type Estimation Results

Figure 16.2 identifies major noise contributors: *unstable behavior* and *self-shutdown* that the noise suppression process in Figure 16.1 needs to emphasize in order to improve the S/N. In Figure 16.3, the S/N indexes reliability (e.g., high S/N yields high reliability). Only two failure types are shown because the others have negative reliability (i.e., noise exceeds signal). Thus, S/N can be used to rank the reliability of mobile device software.

Figure 16.4 shows that the feedback signal $C_i = S_i - N_i$ is negative in Figure 16.1 for *freeze* and *unstable behavior* failure types. Therefore, negative feedback is needed to correct modules because in these cases there are more failed modules than correct modules. Although positive feedback is also important, the modules with failure types associated with negative feedback should receive priority attention.

Figure 16.3 Signal-to-noise ratio S/N versus probability of failure P_i for failure type i.

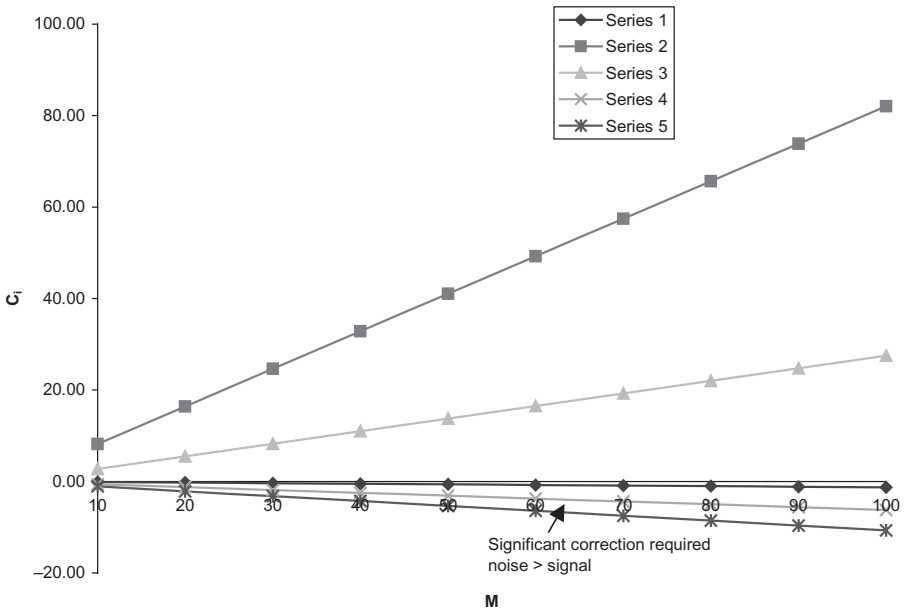


Figure 16.4 Correction signal C_i (number of modules) versus total number of modules M. Failure types: Series 1: Freeze, medium severity; Series 2: Input, high severity; Series 3: Output, low severity; Series 4: Self-shutdown, high severity; Series 5: Unstable behavior, high severity.

EXPECTED NUMBER OF FAILURES AND FAILURE RATE ANALYSIS

In order to extend the modeling effort into the mobile device operating time domain, use the expected number of failures and failure rate data, organized by failure type and recovery type, in Table 16.3 [YUK03]. These data permit more than one dimension to be represented in the model. The dimension presented so far has been static, confined to, for example, reliability as a function of module count. There has been no accounting of mobile device operating time. Unfortunately, the distribution of failures over time is not available. Only the expected number of failures shown in Table 16.3 and the operating time t = 14 months, which were used to compute the failure rates λ, were available. Thus, the model is limited to using a simple time-based reliability model, like the one based on the classical exponential distribution. A justification of the model is that it is conservative because it does not exhibit reliability growth. In fact, it shows just the opposite—reliability decreasing with operating time. In addition, exponentially distributed failure times reflect a high probability of short times and a low probability of long times. Furthermore, my aim is to compare reliabilities by failure type and recovery action type, and not to accurately predict reliability for particular types. Therefore, any shortcoming in the model will occur for all failure and recovery action-type predictions. Thus, predict reliability over a specified operating time of the mobile device t, using Equation 16.13:

$$R(t) = e^{-\lambda t}. \tag{16.13}$$

Table 16.3 Expected Number of Failures and Failure Rates

	Recovery action							
Failure type	Service phone	Reboot	Remove battery	Wait for response	Repeat operation	Unreported	Totals	Failure type failure rate (λ)
Freeze	19.45	12.58	48.02	22.87	0.00	32.03	134.96	9.64
Input failure	3.41	3.41	1.12	0.00	3.41	4.58	15.94	1.14
Output failure	36.62	46.90	2.29	3.41	30.86	73.18	193.27	13.80
Self-shutdown	35.44	0.00	11.46	2.29	0.00	41.20	90.40	6.46
Unstable behavior	36.62	9.17	1.12	1.12	3.41	46.90	98.34	7.02
Totals	131.54	72.06	64.01	29.69	37.68	197.90		
Recovery action recovery rate	9.40	5.15	4.57	2.12	2.69	λ	Failures per month	

Operating Time

It is of interest to compute the operating time during which a specified reliability requirement R(t) is to be achieved. This time is equal to the mission duration that can be achieved for a specified R(t). This time is computed by solving R(t) in Equation 16.13 for t as follows:

$$t_m = \frac{-LN(R(t))}{\lambda}. \tag{16.14}$$

You can see that for a specified reliability requirement R(t), the larger the failure rate λ, the shorter the mission duration that can be achieved.

Results Based on Failure Rate Analysis

Figure 16.5 is interesting because it shows that only one failure type—Input—has acceptable reliability, and, then, only at low operating times. The other types require drastic reductions in failure rate by eliminating software faults to qualify as acceptable. This result is reinforced by Figure 16.6, which shows failure type Input being the only type that achieves the required mission duration of one month.

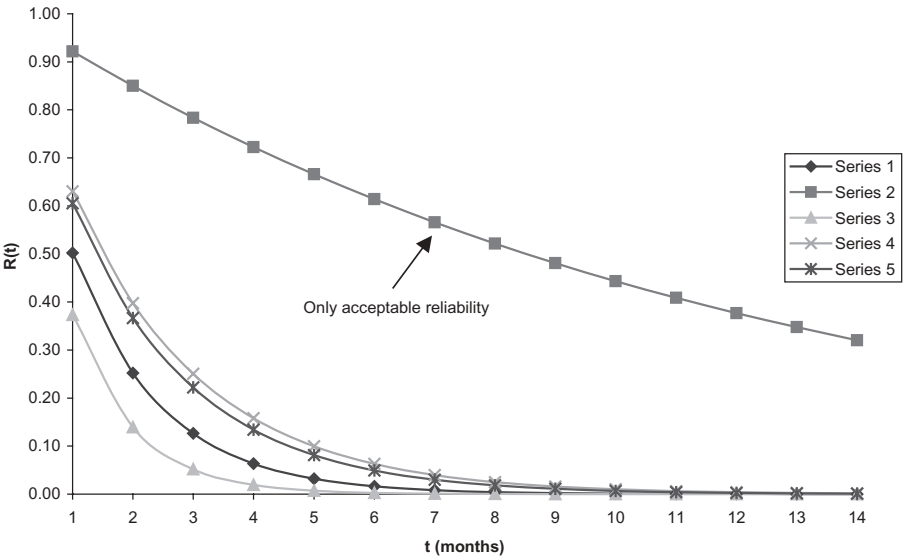


Figure 16.5 Reliability R(t) versus operating time t. Failure types: Series 1: Freeze, failure rate = 0.69 failures per month; Series 2: Input, failure rate = 0.08 failures per month; Series 3: Output, failure rate = 0.99 failures per month; Series 4: Self-shutdown, failure rate = 0.46 failures per month; Series 5: Unstable behavior, failure rate = 0.50 failures per month.

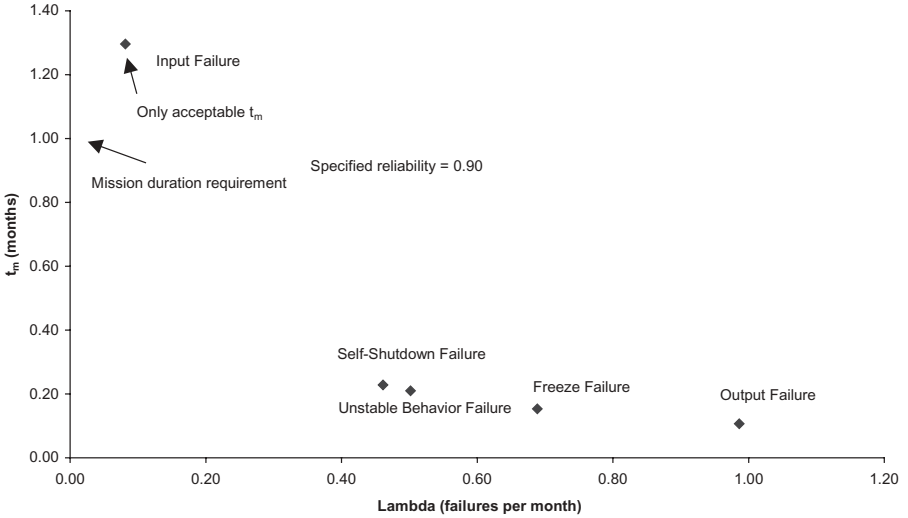


Figure 16.6 Mission duration t_m versus failure rate lambda.

MOBILE DEVICE TESTING EFFECTIVENESS

Testing and reliability have a synergistic relationship, as shown in Figure 16.1. That is, module device failure data generated from test results drive reliability model analysis and the analysis highlights the parts of the software that deserve priority in testing, and reliability predictions influence the selection of test cases. The signal and noise relationships can be used to quantify test effectiveness. Test effectiveness of failure type i, E_i, is defined as the ratio of the change in noise ΔN_i (i.e., number of corrected failed modules) to the total number of modules M. Test effectiveness is expressed in Equation 16.15:

$$E_i = \frac{\Delta N_i}{M}. \tag{16.15}$$

To compute ΔN_i, define N_i^* as the reduced noise accomplished through testing of failure type i (i.e., reduced number of failed modules):

$$\Delta N_i = N_i - N_i^*. \tag{16.16}$$

Recalling from Equation 16.5 the computation of S/N, express N_i^* as shown in Equation 16.17:

$$N_i^* = (S_i P_i s)/(1 - P_i s). \tag{16.17}$$

Then Equation 16.16 is reformulated for failure type i, substituting Equation 16.2 for N_i and Equation 16.17 for N_i^*, as follows:

$$\Delta N_i = N_i - N_i^* = \left(MP_i s - (S_i P_i s)/(1 - P_i s)\right). \tag{16.18}$$

Finally, Equation 16.15 is recomputed in Equation 16.19 to obtain the final form of test effectiveness by failure type:

$$E_i = (MP_i s - (S_i P_i s)/(1 - P_i s))M. \tag{16.19}$$

Note that $E_i > 0$ corresponds to a small signal S_i and $E_i < 0$ corresponds to a large signal. The first case reflects the fact that large gains in noise reduction would be achieved through testing if the number of correct modules, due to eliminating failures of type i, is *already* small. The second case reflects the fact that small gains in noise reduction would be achieved through testing if the number of correct modules is *already* large. Thus, test effectiveness can be used to prioritize modules for testing: the higher the value of E_i (low signal), the higher the priority of modules for testing.

Using similar reasoning for recovery action types and calling on Equations 16.8–16.10, test effectiveness for recovery action type j is computed in Equation 16.20:

$$E_j = \frac{\Delta N_j}{M} = \left(N_j - N_j^*\right)/M = \left(\sum_{i=1}^{f} P_{ij} s_{ij}\right) - (S_j(1 - R_j)/M), \text{ for } R_j \geq 0. \tag{16.20}$$

In this case, note that $E_j > 0$ corresponds to a small signal S_j and $E_j < 0$ corresponds to a large signal. The first case reflects the fact that large gains in noise reduction would be achieved through testing if the number of correct modules, due to recovery action j, is *already* small. The second case reflects the fact that small gains in noise reduction would be achieved through testing if the number of correct modules, due to recovery action j, is *already* large. Thus, again, test effectiveness can be used to prioritize modules for testing: the higher the value of E_j (low signal), the higher the priority of modules for testing.

Test Time

Related to test effectiveness is the duration of test necessary to achieve that effectiveness. Estimate this time based on the reduction in number of failed modules ΔN_i achieved by test effectiveness E_i, for failure rate λ_i and failure type i, which is tabulated in Table 16.3. Thus, test duration t_i is estimated for failure type i, and number of failures f, using Equation 16.15, in Equation 16.21 that expresses the fact that test time is equal to the number of failed modules that are corrected divided by the failure rate. Test duration serves as a test stopping rule:

$$t_i = (fE_i M)/\lambda_i, \text{ for } E_i > 0, \tag{16.21}$$

where $E_i M = \Delta N_i$ and f is the number of failures per failed module.

A similar equation is formulated for recovery action types:

$$t_j = (fE_jM)/\lambda_j, \text{ for } E_j > 0, \tag{16.22}$$

It is recognized that this formulation of test time may understate the time required to identify all mobile device hazards. To adequately evaluate the reliability of a mobile device, the analyst must stress it to identify both hardware and software failures. Using failure data, such as that in Tables 16.1–16.3, the analyst can run realistic tests that stress the hardware and software to fail by using the test times given by Equations 16.21 and 16.22 as baselines, and gradually increasing them until the device fails [STA97]. Applying a stress test to a mobile device is shown in Figure 16.1.

FAILURE TYPE AND RECOVERY ACTION TYPE RESULTS

Signal-to-Noise Ratio

Figure 16.7 shows the plots of S/N failure type and S/N recovery action type along with the S/N limit = 1 (i.e., number of correct modules = number of failed modules). Failure types below the limit should be investigated to identify the cause of excessive failures in the mobile device software development process. Correspondingly, recovery action types below the limit need attention to identify why the recovery software is not able to provide effective recovery.

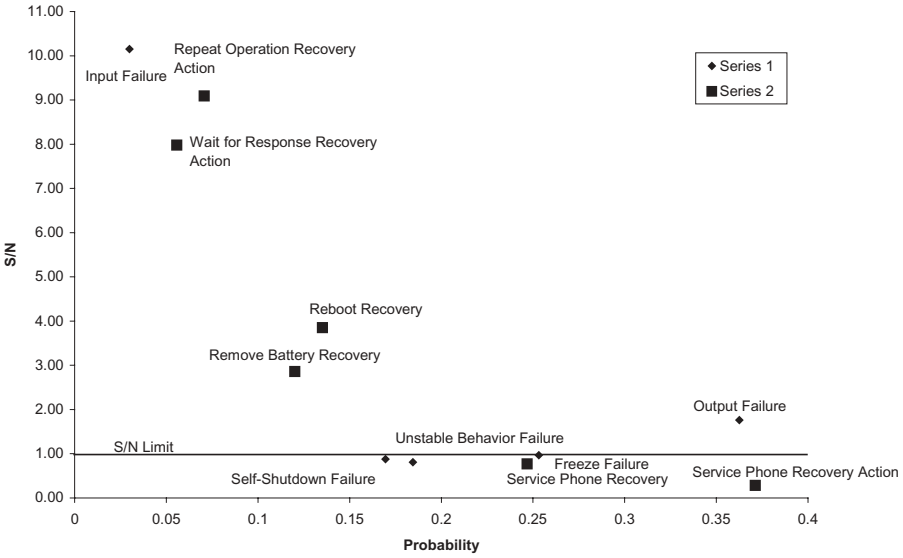


Figure 16.7 Signal-to-noise ratio S/N versus probability of failure and probability of recovery action. Series1: Failure type S/N. Series 2: Recovery action type S/N.

Reliability

Figure 16.8 shows, as Figure 16.7 had shown, that one of the best user recovery actions is to wait for a response. This strategy showed the second best S/N in Figure 16.7 and the highest reliability in Figure 16.8 (the other recovery action reliabilities are all lower, but are not shown). However, recovery action reliability lags the input failure reliability and never achieves the reliability limit. This result indicates that these example mobile devices need improved reliability even *after* responding to a failure.

Mission Duration

Assume that an acceptable mission duration for the mobile device user is 1 month to achieve a specified reliability of 0.90. Then Figure 16.9 demonstrates that the only situation in which this could occur is when the mobile device is subject to an input failure. All other failures would result in unacceptable mission duration at the specified reliability. Also note that all recovery actions are deficient with respect to achieving the desired mission duration after failures occur. Thus, mobile device manufacturers should improve the quality of their recovery action software. In

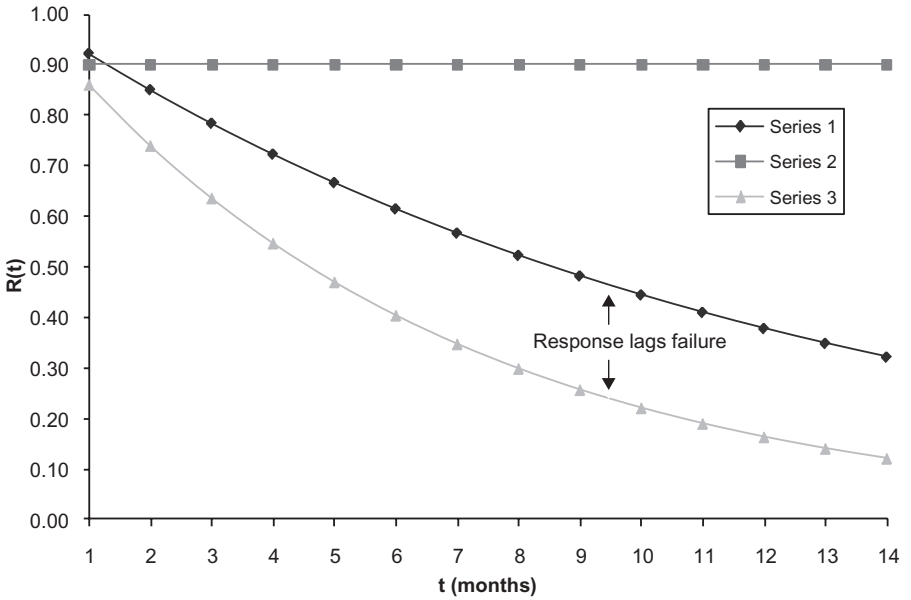


Figure 16.8 Reliability R(t) versus operating time t. Series 1: Input failure type, failure rate = 0.08 failures per month. Series 2: Reliability limit. Series 3: Wait for response recovery action, failure rate = 0.15 failures per month.

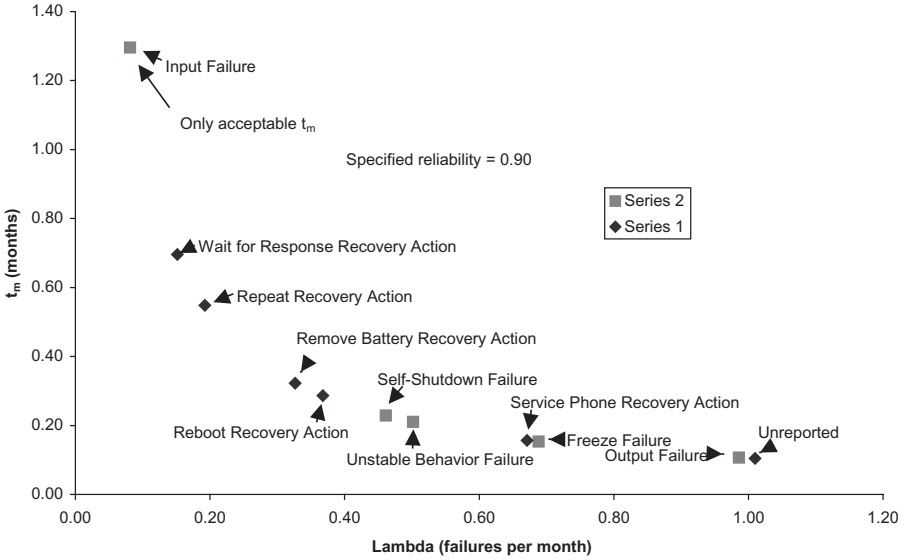


Figure 16.9 Mission duration t_m versus failure rate lambda. Series 1: Recovery action type. Series 2: Failure type.

particular, users should not have to wait for a response in order to recover from a failure. Response to failures should be so effective that users should be unaware of this strategy! The best way of avoiding user frustration is by *designing in* higher quality.

Test Effectiveness

Figure 16.10 demonstrates, perhaps counterintuitively, that test effectiveness *increases* with *lower* S/N. The reason is that lower S/N means higher noise, which represents many failed modules that are subject to correction. Figure 16.10 allows one to identify the test effectiveness for a given number of modules that are being tested for a mobile device. The other two failure types are not shown because their test effectiveness are negative and do not plot well on the same graph.

Test Time

Test time is modeled as a two-phase sequence: first, test cases are based on type of failure (e.g., freeze) followed by test cases that are based on recovery action (e.g., remove battery). In the first phase, test time *increases* with *decreasing* S/N (i.e., many failed modules compared with the number of correct modules), as shown in

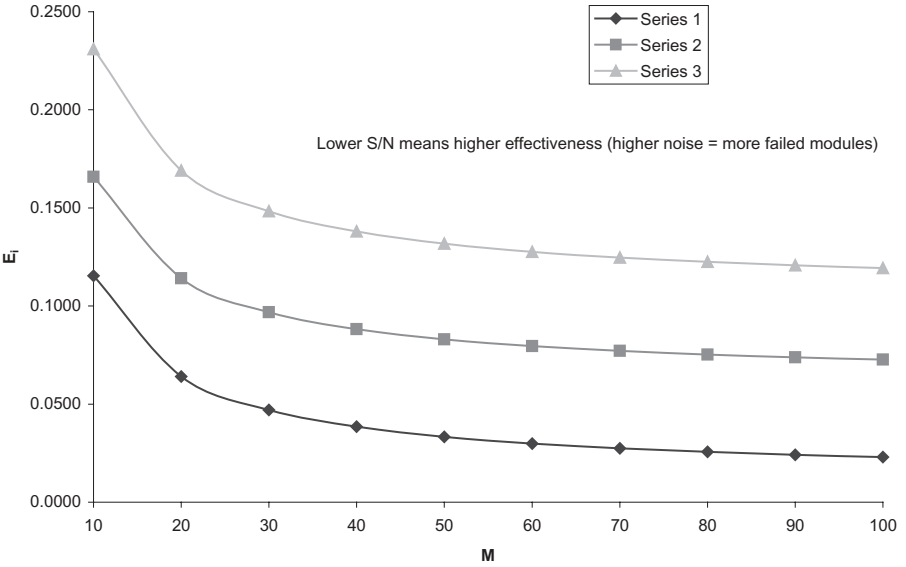


Figure 16.10 Test efficiency E_i for failure type i versus total number of modules M. Series 1: Freeze failure, S/N = 0.97. Series 2: Self-shutdown failure, S/N = 0.88. Series 3: Unstable behavior failure, S/N = 0.81.

Figure 16.11. In the second phase, the focus of testing and debugging is recovery actions. Recovery action testing takes less time than failure type testing because many bugs have already been removed by failure type testing by the time recovery action testing takes place. In addition, recovery type testing is based on test cases under the control of the tester (e.g., try removing the battery, and see what happens). In the case of failure type testing, the tester is at the mercy of the operating environment of the mobile device (e.g., failures caused by noise in the communications network). Thus, failure type testing takes more time. Another characteristic of both failure type and recovery action type testing is that decreasing reliability necessitates increasing test time, as shown in Figure 16.11.

SUMMARY OF RESULTS

Whichever measure was being analyzed, whether noise, S/N, reliability, minimum acceptable operating time (mission duration), or test effectiveness, and whether the focus was failure type or recovery action type, the sample of mobile devices did not, in general, meet requirements. Since this is a large, representative, and recent sample, the results suggest that mobile devices should be improved so that they are really usable by customers. While it is true that users discard mobile devices on average every 18 months [CIN07], results indicate that severe reliability problem will prevail short of 18 months. For example, see Figure 16.5.

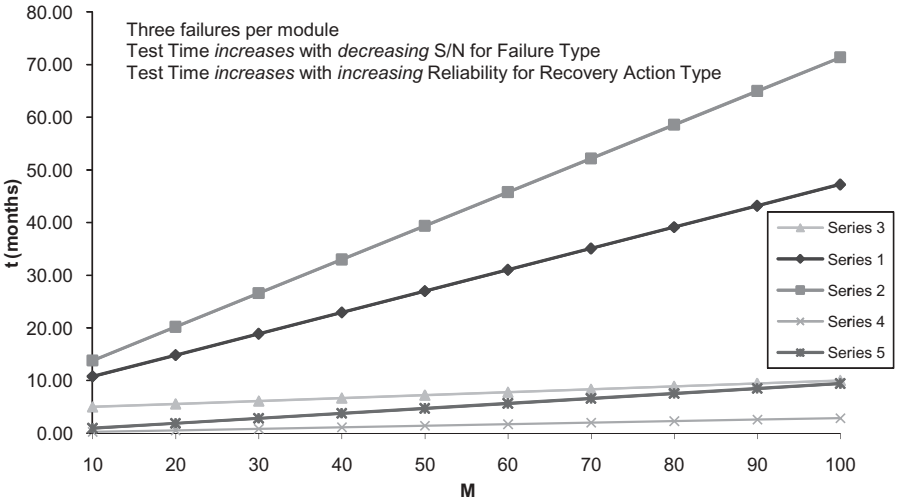


Figure 16.11 Test time t for failure type and recovery action type versus total number of modules. Series 1: Self-shutdown failure, S/N = 0.88. Series 2: Unstable behavior failure, S/N = 0.81. Series 3: Freeze failure, S/N = 0.97. Series 4: Remove battery recovery action, reliability = 0.65. Series 5: Wait for response recovery action, reliability = 0.88.

FUTURE MOBILE DEVICE DEVELOPMENTS AND RESEARCH

Grid Computing

In the near future, personal mobile devices will become ingredients of other infrastructures, such as the electric grid. Computing techniques have been devised to enhance mobile devices so that their interoperability with other mobile devices and electric grid infrastructure will be improved, such as the Personal Augmented Computing Environment (PACE). PACE characteristics include (1) collaborative visualization using display clusters composed by mobile devices, for example, electric utility and customer mobile electric meter reading devices collaborating in the presentation of meter readings, and (2) context-aware methods for mobile devices to achieve efficient utilization of grid resources, for example, intelligent mobile meter readers being aware of, and communicating with, electric substations [LUO07]. Future research will be directed toward creating software development models for mobile devices to communicate with the electric grid in a collaborative processing mode.

Context-Aware Migratory Service

Because a mobile process can involve context-aware migratory tasks (e.g., sudden need for the mobile device to move with the user [context aware] and connect to a

hot spot [migratory]) and heterogeneous mobile devices (e.g., smartphones, meter readers), the mobile device model must account for context and migration. When failures occur, the mobile network has to try to find another mobile device to execute the current process. If it is successful in finding such a device, it will transfer the process to this device [KUN]. Unlike a stationary service that always executes on the same node, a context-aware migratory service is capable of migrating to different nodes in the network in order to effectively accomplish its task. Thus, the interaction between a user application and a migratory service can continue uninterrupted, except for small delays generated by the migration process. This model provides two advantages. First, when a node becomes unsuitable for hosting a service, the user application does not need to perform any new service discovery because the current service can automatically migrate to a node that is qualified for accomplishing the current task (e.g., when an electric grid substation fails, the mobile meter reader can be automatically connected to an operational substation). Second, the migratory service incorporates all the state information (e.g., power usage reported at the failed substation before it failed) necessary to resume the interaction with the user when the migration to a different node has completed [RIV07].

REFERENCES

[CHA07] Shuvo Chatterjee, Dietrich Falkenthal, and Tormod Ree, "Exploring adaptive power saving schemes for mobile VoIP devices in IEEE 802.11 networks," *Second International Conference on Digital Telecommunications (ICDT'07)*, 2007, p. 13.

[CIN07] M. Cinque, D. Cotroneo, Z. Kalbarczyk, and R. K. Iyer, "How do mobile phones fail? A failure data analysis of Symbian OS smart phones," in *37th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN'07)*, 2007, pp. 585–594.

[COM] http://www.comp.nus.edu.sg/~damithch/df/device-fragmentation.htm

[FIT07] Frank H. P. Fitzek and Frank Reichert (eds.) *Mobile Phone Programming and Its Application to Wireless Networking*. Dordrecht, The Netherlands: Springer, 2007.

[HOL06] Magnus Holmqvist, Gunnar Stefansson, "Mobile RFID—a case from Volvo on innovation in SCM," in *Proceedings of the 39th Annual Hawaii International Conference on System Sciences (HICSS'06) Track 6*, 2006, p. 141a.

[IYE85] R. K. Iyer, D. J. Rossetti, "Effect of system workload on operating system reliability: a study on IBM 3081," *IEEE Transactions on Software Engineering*, 1985, 11(12), pp. 1438–1448.

[KUN] Christian P. Kunze, Sonja Zaplata, Mirwais Turjalei, and Winfried Lamersdorf, "Enabling context-based cooperation: a generic context model and management system" Distributed Systems and Information Systems, Computer Science Department, University of Hamburg, Hamburg, Germany, http://vsis-www.informatik.uni-hamburg.de/getDoc.php/publications/314/BIS2008_kztl08.pdf

[LAR07] Henry Larkin, "Data representations for mobile devices," *13th International Conference on Parallel and Distributed Systems—Volume 1 (ICPADS'07)*, 2007, pp. 1–6.

[LUO07] X. Luo, "PACE: augmenting personal mobile devices with scalable computing," *Seventh IEEE International Symposium on Cluster Computing and the Grid (CCGrid '07)*, 2007, pp. 875–880.

[MCC02] Steve McConnell, "Real quality for real engineers," *IEEE Software*, 2002, 19(2), pp. 5–7.

[POU06] Gilda Pour, Nivedita Laad, "Enhancing the horizons of mobile computing with mobile agent components," *5th IEEE/ACIS International Conference on Computer and Information Science and 1st IEEE/ACIS International Workshop on Component-Based Software Engineering, Software Architecture and Reuse (ICIS-COMSAR'06)*, 2006, pp. 225–230.

[RIV07] Oriana Riva, Tamer Nadeem, Cristian Borcea, and Liviu Iftode, "Context-aware migratory services in ad hoc networks," *IEEE Transactions On Mobile Computing*, 2007, 6(12).

[ROU04] T. Ruohonen, L. Ukkonen, M. Soini, L. Sydänheimio, and M. Kivikoski, "Quality and reliability of GPRS connections," in *Proceedings of Consumer Communications and Networking Conference,* Las Vegas, 5–8 January, 2004, pp. 268–272.

[SAT03] Ichiro Satoh, "A testing framework for mobile computing software," *IEEE Transactions on Software Engineering*, 2003, 29(12), pp. 1112–1121.

[SAU06] Vibhu Saujanya Sharma, Pankaj Jalote, "Stabilization time—a quality metric for software products," *17th International Symposium on Software Reliability Engineering (ISSRE'06)*, 2006, pp. 45–51.

[STA97] Donald Staab, Eugene R. Hnatek, "Diagnosing IC failures in a fast environment," *IEEE Design and Test of Computers*, 1997, 14(3), pp. 70–75.

[YEU07] Wilson M. Yeung, Joseph K. Ng, "Wireless LAN positioning based on received signal strength from mobile device and access points," *13th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA 2007)*, 2007, pp. 131–137.

[YUK03] Yukikazu Nakamoto, "The next generation software platform for mobile phones," *Sixth IEEE International Symposium on Object-Oriented Real-Time Distributed Computing (ISORC'03)*, 2003, p. 46.

[ZAN93] Enrico Zanoni and Paolo Pavan, "Improving the reliability and safety of automotive electronics," *IEEE Micro*, 1993, 13(1), pp. 30–48.

Chapter 17

Object-Oriented Analysis and Design Applied to Mathematical Software

Object-oriented (O-O) methods are highly touted in the literature as the solution to the world's software reliability problems. While this *may* be true, there seems to be little supporting evidence. Also, based on O-O project results in the literature, you would wonder how well earlier methods, such as structured analysis and design, would have fared. There is a natural relationship between O-O attributes and the modeling of physical systems, such as the software for controlling a nuclear reactor. However, such a relationship is not obvious for modeling mathematical software, such as programs designed to predict software reliability. The rationale for using mathematics as the basis of comparison with O-O methods is that the solution of mathematical equations is a common computer application; indeed, it was the reason the first computers were developed. While some O-O diagrams are useful for providing high-level visibility of computer program structure, in the main, prediction equations, coupled with a directed graph representation of the computer program, are better tools for modeling mathematical software. Thus, it is important for the reader to learn for which applications O-O methods can be applied and for which applications O-O methods would be misapplied.

INTRODUCTION

It is assumed that the reader has a basic understanding of probability and statistics. Where this is not the case, the following reference will be helpful: David M. Levine, Patricia P. Ramsey, and Robert K. Smidt, *Applied Statistics for Engineers and Scientists* (New York: Prentice-Hall, 2001).

Computer, Network, Software, and Hardware Engineering with Applications, First Edition. Norman F. Schneidewind.
© 2012 the Institute of Electrical and Electronics Engineers, Inc. Published 2012 by John Wiley & Sons, Inc.

Definitions

First, definitions are presented to assist the reader in understanding the sections that follow.

Object. In a computer program, any entity that can execute in a computer or can support execution, such as an interrupt from an input device and the software instructions that support interrupt processing.

Class. A classification of objects, such as *program interrupts*, where an interrupt from an input device is an *object*.

Inheritance. A property of object-oriented design that allows an object to acquire the properties of its class (e.g., the object *input device interrupt* inherits the property *jump to first interrupt processing instruction* from the class of *interrupts*).

Directed Graph. A graph whose edges are ordered pairs of nodes. That is, each edge is preceded by a node and followed by another node. A directed graph can be used to represent a computer program, where edges represent program branches and nodes represent program statements.

Unified Modeling Language (UML). Standardized notation and set of diagrams for supporting object-oriented (O-O) analysis and design.

Activity Diagram. In the UML, activity diagrams are used to describe the operational step-by-step workflows of software components in a system. An activity diagram shows computer program control flows.

Sequence Diagram. A sequence diagram in UML is an interaction diagram that shows how software processes (e.g., computer code for computing a mathematical function) operate with one another and in what order.

State Diagram. A diagram that shows states (e.g., a computer program is processing an interrupt) and transitions between states (e.g., transition from the state of processing an interrupt to the state of returning to the interrupted program).

Cyclomatic Complexity Metric. (Number of edges – number of nodes) + 1 in a directed graph. Cyclomatic complexity can be used to represent the number of independent paths in a computer program, where an independent path is one that cannot be formed by combining other paths in the directed graph.

Information Hiding. A software design technique that "hides" system details (e.g., disk format) in the interface between modules rather than in modules, so that system changes will only affect the interface, and hopefully increase software reliability, as a result.

Procedure. Portion of software code (e.g., subroutine) within a software program

Encapsulation. In O-O programming, encapsulation is the inclusion within a program object of all the resources needed for the object to function. For

example, the object *interrupt processing* would include instructions for processing an interrupt, pointer to the first interrupt processing instruction in memory, and the return address of the instruction that would have been executed if the interrupt had not occurred.

CAN O-O METHODS BE APPLIED TO MATHEMATICS?

To see whether O-O methods can be applied to mathematical software, experiments are conducted to compare this approach with using equations and directed graph representations of computer program code to see whether O-O analysis, design, and language are applicable in general to mathematical models and, in particular, to developing the mathematics for software reliability models. It is recognized that there are many applications of O-O methods other than mathematical models, but this is such a fundamental computer application that it is expected that *any* design method would do well.

The perfect mathematical programming environment would automatically transform systems of equations into efficient symbolic and numerical programs. It would select solution routines that have good convergence properties for the given problem. It would also formulate equations from problem specifications. Although it is easy to imagine such an automatic environment, it is more realistic to assume that the user will interact with the system, supplying information to help it choose the right algorithms and transformations. Such a system is ObjectMath, which is a high-level programming environment with a modeling language based on the computer algebra language Mathematica. The ObjectMath language augments Mathematica with classes and other O-O language constructs. However, ObjectMath focuses on mathematical modeling rather than on O-O programming [FRI95]. While this development is impressive, it is focused on the mathematical modeling of physical objects rather than on mathematical software reliability, which is the application that is evaluated for O-O applicability in this chapter.

O-O Approach versus Mathematics

The ease of mapping real-world objects to the O-O model, enabling software reuse and the support of various tools, have led to its wide acceptance [SEN07]. While it is easy to see that the O-O approach is compatible with developing, let us say, an elevator system or Web site, it is not clear how this approach applies to developing mathematical functions. In the case of a Web site, there are activities (e.g., user access to a Web server) so that an activity diagram would apply. Also, in the case of an elevator system, there are the activities of passengers accessing and riding in elevators. Interestingly, even in applications such as elevator systems, there can be limitations to the O-O approach. For example, although the UML sequence diagrams are capable of representing sequential interactions (e.g., only a single elevator floor request at a time), they are not capable of properly representing concurrent interac-

tions (e.g., elevator passengers on different floors concurrently pressing up and down buttons). Therefore, it is necessary to have a model capable of representing both sequential and concurrent interactions between objects [RYA06].

Now, in the case of equations, would activity diagrams and state diagrams apply? Would it make sense to consider an equation as an object? To pursue answers to these questions, the O-O approach will be used to model facets of the software reliability models and compare the result with the mathematical model approach.

In explicating the mathematical model approach, you can use the directed graph representation of the mathematical model. This approach has the advantages of showing iteration, computing cyclomatic complexity metrics from the directed graph, and identifying the key parts of the program to test [MCC76]. Interestingly, there seems to be no comparable features of the O-O approach. For example, *iteration* in O-O is defined as: an operation that permits all parts of an object to be accessed in a well-defined order [BOO94]. Note that repetition is not explicitly mentioned. This definition does not ring true for writing a mathematical program in which we simply want to repeat the execution of an equation.

Proponents of O-O claim many benefits. Unfortunately, these claims are not accompanied by a discussion of disadvantages. An obvious one is that O-O is highly abstract, and based on experience in teaching this model, students find the abstractions difficult to grasp. In fact, some researchers employ UML without stating why they are using it [SEL04]. The O-O approaches suffer because they are too syntax oriented (e.g., emphasis on UML diagramming techniques) and lack a proper and simple semantic foundation (e.g., mathematical equations that communicate the meaning of the application). A precise description and common understanding of the semantics, as well as the relations between the various UML diagrams for the description of software systems, is missing [BRO01].

As claimed, the benefits of O-O analysis and design specifically include the following [GRA] (comments added where the claim is challenged):

- Required changes are localized and unexpected interactions with other program modules are unlikely. (How is this different from information hiding and modular design?)
- Inheritance makes O-O systems more extensible, contributing to more rapid development. (Equations have variables and parameters that can be made extensible by changing the data used in the variables and parameters.)
- Object-based design is suitable for distributed, parallel, or sequential implementation. (Equations can be used in any of the aforementioned implementations)
- Objects correspond more closely to the products and processes in the conceptual worlds of the designer and user, leading to greater traceability of product and process. (This can be accomplished by effective software management requiring traceability among software products and the process steps that produce them; also, mathematical models do not preclude the use of objects.)

- Shared data areas are protected, reducing the possibility of unexpected modifications or other update anomalies; this is an operating and security system property, not a property of the design paradigm. (Independent of the design paradigm, data can be protected by access controls and encryption.)
- O-O provides various views of a software system that are useful for understanding and maintaining the code [SAL04]. (While this is true, equations provide an excellent view of its implemented software, which is useful for debugging.)
- O-O can be effective for reengineering from a software system designed with procedures to an O-O perspective to provide better code visibility [ZOU02].
- O-O can be used for transforming the states of legacy software (i.e., software that, while old, must be maintained because it is still valuable to the using organization) to O-O software, thus providing greater clarity of system states [ZOU021].

ELEMENTS OF A REQUIREMENT

In analysis and design, models are built to seek an understanding of the requirements or to specify the systems to be built. To be useful, the model should be abstract (does not contain unnecessary details), complete (captures all relevant aspects), unambiguous (meaning is clearly expressed), and well integrated (the various parts fit together to form a coherent whole) [KGU96]. Although a worthy statement of the objectives of analysis and design, it is a tall order because it is difficult to *not* include unnecessary details and at the same time capture all relevant aspects. It is difficult, particularly in the requirements phase of a project, to know what is unnecessary and what is relevant. The following is an approach to identifying the elements of a requirement for the purpose of making a requirement understandable.

Object. The focus of attention (e.g., software reliability).

Function. A function is the task that the object must achieve (e.g., software reliability [object] must achieve its specification [task] during test and operating time). In programming languages, a function is a subroutine that can, if required, return a single value to the caller (the part of the program which invoked the function). The strength of functions lies in the fact that they are programs within a program. Functions are written for two major reasons: (1) to provide frequently used operations that can be accessed by *many programs* or from *many points* within a *single program* and (2) to modularize complex programs and make the maintenance and understanding of such programs easier. In C++, a function is a named, independent section of code that performs a specific task and optionally returns a value to the calling program. User-defined functions are functions that programmers create for specialized tasks.

Limit. Constraint imposed on a function (e.g., software reliability must exceed 0.9500 for all operating times).

Parameter. A model numerical factor estimated from data (e.g., software reliability parameters estimated from failure data).

Variable. A model predictor specified in a function (e.g., predictor of software reliability).

Equation. Mathematical implementation of a function: relationship among variables and parameters (e.g., $\text{reliability} = R(t) = \int_T^\infty (p(t)dt)$, where p(t) is a probability density function.

Model. Representation of objects, functions, limits, parameters, variables, and equations.

Requirement Implementation

Programming Language Statements. Statements that implement a model on a computer (e.g., C++ statements).

Data. Historical data (e.g., failure counts in time intervals) for estimating model parameters and for computing actual model quantities, based on historical failure data (e.g., actual reliability).

Iteration. Repetition of an operation (e.g., reading failure data from a file).

Decision Operations. Control program flow (e.g., processing failure data dependent on its value).

EXAMPLE OF COMPARING O-O WITH MATHEMATICAL APPROACHES

The *mathematical concept* of a *function* expresses dependence between two or more quantities, one of which is known and the other which is produced. A function associates a single output to each input element drawn from a fixed set. In Equation 17.1, $P(x_t, t)$ is a function of x_t and t.

A *variable* assumes values based on a function, such as x_t and t in Equation 17.1. The term usually occurs in opposition to *parameter*, which is a symbol for a nonvarying value, such as λ in Equation 17.1.

You can use Equation 17.1 to see an example of how a Poisson failure occurrence model and its associated function, reliability, would be implemented with the two approaches. In developing the implementation approaches, each facet of the failure model is defined and analyzed in order to illustrate how well the two approaches apply.

In the Poisson distribution of failure occurrence, $P(x_t, t)$, in Equation 17.1, λ is the failure rate, x_t is failure count at test or operating time t, and t is the time of failure occurrence:

$$P(x_t, t) = \frac{(\lambda t)^{x_t} e^{-(\lambda t)}}{x_t!}. \tag{17.1}$$

In order to compute Equation 17.1, the failure rate λ must be computed in Equation 17.2:

$$\lambda = \frac{\sum_{i=1}^{n} x_t}{t_n}, \tag{17.2}$$

where t_n is the last failure time.

Now, reliability, with exponentially distributed operating times t, can be obtained from Equation 17.1 by setting $x_t = 0$. Reliability R(t) is shown in Equation 17.3:

$$R(t) = e^{-\lambda t}. \tag{17.3}$$

Comparing O-O and Mathematical Definition of Terms

Table 17.1 contains definitions and comparisons of terms from the O-O and mathematical domains. Multiple definitions are valid because the appropriate definition depends on the *context* of the application. For example, mathematical terms could be cast in the context of developing a failure model, such as Equation 17.1.

O-O CONCEPTS APPLIED TO POISSON FAILURE MODEL

At this stage in the comparison, the O-O representation of the Poisson failure model is developed by first defining the model objects and then showing how the UML diagrams can be used to model the elements. In showing these diagrams, it is not suggested that all of them are needed to model mathematical software (e.g., Poisson failure model). Rather, the goal is to illuminate the various perspectives that the diagrams provide and determine which are the best for a mathematical application.

Objects

Objects have two characteristics: state and behavior [BOO94] (e.g., Poisson failure model *object* is executed [state] and the result is stored [behavior]). Objects also possess attributes (e.g., failure rate, failure count, and failure time).

Activity Diagram

The purpose of the activity diagram is to model the procedural flow of actions in a system. [DOU98]. Activity diagrams can be used to model the activities associated

Table 17.1 Comparison of O-O and Mathematical Terms

Term	O-O definition	O-O application	Mathematical definition	Mathematical application
Function	1. Method, operation [DOU98] 2.An input/output mapping resulting from an object's behavior [BOO94]	Encapsulate a process under one name [ULL06]	Dependence between two quantities (Wikipedia encyclopedia)	1. Expresses dependence between two or more quantities (Wikipedia encyclopedia) 2. Predictor of software failure $P(x_t, t)$
Variable	Language element (Wikipedia encyclopedia)	Evolves dynamically over time	Assumes values based on a function	x_t: failure count t: time of failure occurrence
Parameter	Attribute of object	Equation fixed elements in C++	1. Nonvarying quantity 2. A model numerical factor estimated from data (e.g., failure rate λ)	Poisson failure model parameter computed from failure data: λ: failure rate
Object	Has state, behavior, and identity [BOO94]	Organizing components that can be connected to form a complex system	Has state, behavior, and identity [BOO94]	The focus of attention: Poisson failure model
State	One of the possible conditions in which an object may exist [BOO94]	Change condition of a system based on events	Properties that a system displays at a given time	State changes from property "no failure count input" to property "failure count input" at a specified time
Behavior	How an object reacts to state changes [BOO94]	Identify the characteristics of an object [BOO94]	How an object reacts to state changes [BOO94]	Poisson failure model Object is executed when its state changes from "no failure count input" to "failure count input" at a specified time
Class	The classification of objects	Shows the relationships of classes, objects, and methods in a system	The classification of objects	The class of probability distributions of which Poisson is a member (object)
Method	The operations performed on the object of a class [BOO94]	Shows what operations exist in a system and how they are performed	The operations performed on the object of a class [BOO94]	Compute the function $P(x_t, t)$ (object) of the class probability distribution

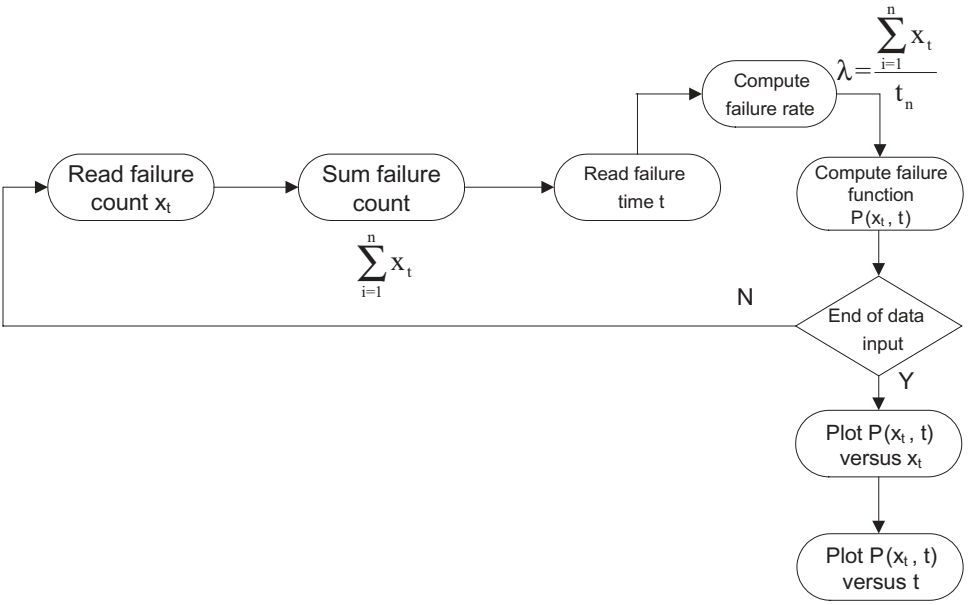


Figure 17.1 Activity diagram for Poisson failure model. t_n, last failure time.

with implementing a function, such as the Poisson failure model. The activity diagram for the Poisson failure model is shown in Figure 17.1. Notice that in addition to modeling the flow of producing the function, the diagram also includes decision activities such as determining when all of the input has been read.

Sequence Diagram

Sequence diagrams show the sequence of operations between objects and the sequence of program steps that are required to implement a model [DOU98]. The sequence diagram for the Poisson failure model is shown in Figure 17.2. While activity diagrams are one dimensional, sequence diagrams provide both the sequence of model operations on data and the sequence of steps that implement the model operations.

State Diagram

The state of an object represents the results of its behavior [BOO94]. For example, once failure count data have been read in Figure 17.1, this is a trigger (event) for the Poisson failure model (object) to store the failure count (action), and go to the "sum failure count" state. This is called a state transition. Each state transition con-

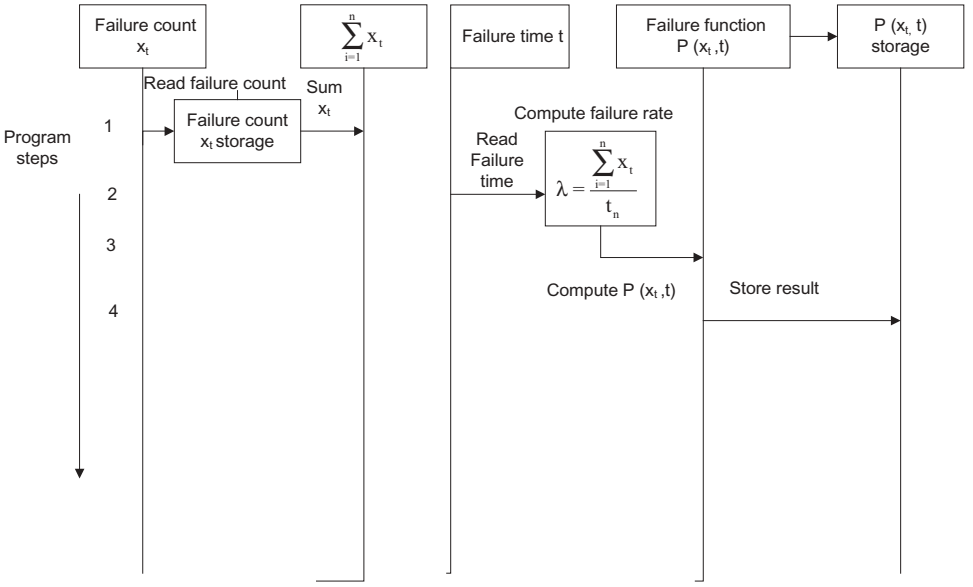


Figure 17.2 Poisson model sequence diagram. t_n, last failure time.

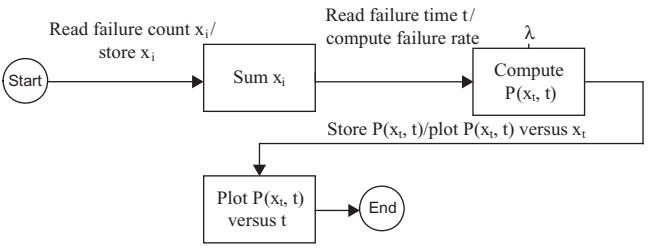


Figure 17.3 Poisson model state diagram.

nects two states [BOO94]. Before constructing the state diagram in Figure 17.3, it is useful to identify the model states, events, actions, and state transitions, as shown in Table 17.2.

Class Diagram

A class diagram shows the relationships among classes, objects of a class, and the methods (operations) performed on the classes. When a class is declared, it is identified by name, attributes, and methods. According to Eden [EDE02], the absence of variable symbols is one of the major shortcomings of class diagrams. However, this does not have to be the case, as shown in Figure 17.4, where mathematical symbols

Table 17.2 Poisson Failure Model State, Events, Actions, and State Transitions

State	Event/Action	State transition
start	Read failure count x_i/store x_i	Sum x_i
Sum x_i	Read failure time t/compute failure rate	Compute $P(x_t, t)$
Compute $P(x_t, t)$	Store $P(x_t, t)$/plot $P(x_t, t)$ versus x_t	Plot $P(x_t, t)$ versus t
end		

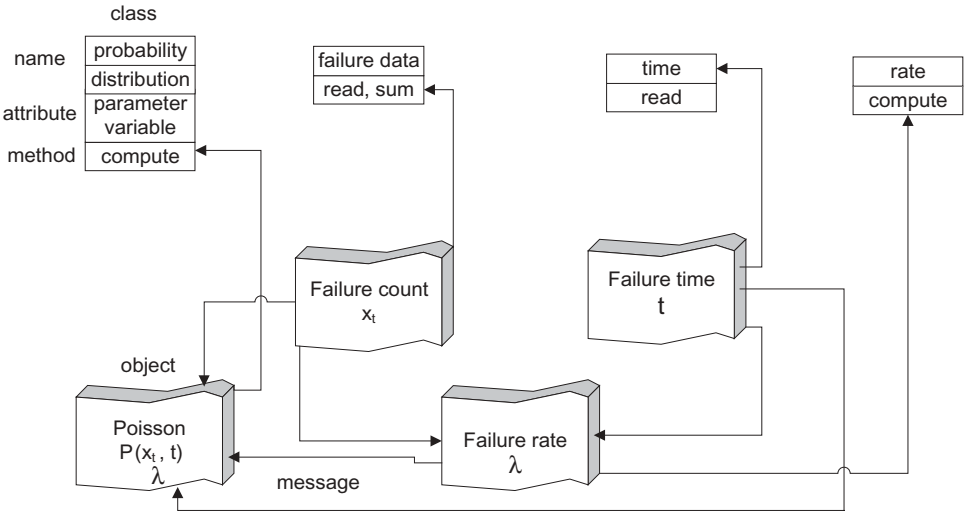


Figure 17.4 Poisson failure model class diagram.

for the Poisson distribution have been added. The primary usefulness of this class diagram is to provide a template for using various objects (probability functions) and their attributes (*variables* and *parameters*) in the same probability distribution class in an O-O programming language.

Class

A set of objects that have common attributes [BOO94]. For example, the class of probability distributions is comprised of the objects Poisson, exponential, normal, uniform, and so on.

Method

An operation on an object that is part of the declaration of a class [BOO94]: the read failure count x_i operation performed on the Poisson failure model class diagram

in Figure 17.4. Once the classes have been identified in Figure 17.4, the interfaces with objects and between objects can be specified. Then you would identify the services the objects are to perform (e.g., read, sum), and messages they may send (e.g., Object Failure Time t $\rightarrow$ Object Failure rate λ) and receive (e.g., Object Failure rate $\lambda \leftarrow$ Object Failure Time t) in Figure 17.4 [HOF97]. The relationship between objects must be designed with great care, because they determine how well the run-time program will perform [GAM95]. For example, in Figure 17.4, an efficient implementation is obtained by the objects *failure count* and *failure time* feeding the object *failure rate*. Then, the failure rate λ is computed. Last, objects *failure count*, *failure time*, and *failure rate* feed object *Poisson* for computing $P(x_t, t)$.

Summary of O-O Diagrams Properties

Based on an analysis of the O-O diagrams, the activity and sequence diagrams were the most useful for designing the Poisson failure model function in C++ (see Appendix for the code). The reason is that these diagrams portray the sequence of activities in the code that are necessary to compute the function.

APPLYING MATHEMATICAL MODELING TO THE POISSON FAILURE MODEL

A modeling method has two major components: a model (e.g., Poisson failure model) and a procedure (e.g., the steps below in implementing the Poisson failure model). The model consists of the underlying concepts (e.g., failure occurrences distributed according to a Poisson process) and associated notation (e.g., equations and C++ syntax). The procedure consists of a number of steps (e.g., failure data identification) required to construct the model [KGU96]. In the mathematical modeling approach, the equations suggest the steps to implement the program. In the O-O approach, the sequence diagram (see Fig. 17.2) can be used to identify program steps. However, the mathematical modeling approach has an advantage because in all of the O-O references, there is no mention about the nasty details in writing computer code of items such as iteration control, variable types, array bounds, and sequence of computer code fragments, all of which can have a significant effect on correctness of program execution. The identification of these coding details flow more naturally from mathematical expressions. Furthermore, in the O-O paradigm, data are sometimes relegated to an obscure role that has nothing to do with the collection and processing of raw data (e.g., amassing failure data and identifying its statistical distribution). For example, in *Real-Time UML: Developing Efficient Objects for Embedded Systems* [DOU98, p.310], the author describes data collection as assembling primitive data attributes that may be structured in a myriad of ways, including stacks, queues, lists, vectors, and a forest of trees, to the exclusion of discussing raw data collection and processing.

MATHEMATICAL MODELING DESIGN APPROACH EXAMPLE

The objective of this section is to show the reader the details of implementing software, using the National Aeronautics and Space Administration (NASA) Space Shuttle flight software as an example. The following approach is comprised of two synchronized program development activities: Identify the several phases and steps in program implementation and, in addition, construct a directed graph of the program logic consisting of nodes (program functions While, If, Else, Set, Read, Write, Store, and Compute) and edges (Transfer Control, Iteration Control, and Return). The directed graph will serve as the vehicle for expressing C++ program logic, and, in addition, allow you to identify the key paths to test based on the cyclomatic complexity metric [MCC76].

Failure Data Identification Phase

Identify the number of failure counts that occur at test time t, x_t. The NASA Space Shuttle flight software OI6 failure data are used as an example, and identifies the times t when the failures occurred. These failure data (x_t) were obtained from the Shuttle contractor and are organized by the number of days (t) since the software was released by the contractor to NASA. The data are listed in Table 17.3 along with the test paths used to debug the C++ program.

Table 17.3 Shuttle OI6 Failure Data

Failure time	Failure count	Factorial	Poisson failure model	Reliability	
t	x_t	$x_{t!}$	$P(x_t, t)$	R(t)	Test path
56	0	1	0.8658	0.8658	E
71	2	2	0.0139	0.8330	A,B,C,D,F,G
104	1	1	0.2048	0.7651	E
105	0	1	0.7632	0.7632	E
119	2	2	0.0345	0.7362	A,B,C,D,F,G
293	1	1	0.3548	0.4704	E
382	1	1	0.3678	0.3741	E
525	1	1	0.3499	0.2589	E
711	1	1	0.2935	0.1604	E
1355	1	1	0.1066	0.0306	E
1748	1	1	0.0500	0.0111	E
1951	1	1	0.0331	0.0066	E
2307	1	1	0.0157	0.0026	E
5438	1	1	0.0000	0.0000	E

Failure rate $\lambda = 0.002574$.

C++ Program Logic Development Phase

This section is dedicated to developing the logic of the C++ program that is used to implement the Poisson failure model. The steps that follow correspond to the logic in Figure 17.5.

While 1 Component
Node 1: *While* there is more failure data x_t
Edge 1, 2: *Transfer Control*
Node 2: *Store* x_t
Edge 2, 3: *Transfer Control*
Node 3: *If* **not** reached end of failure data x_t input
Edge 3, 1:*Iteration Control*, *Return* to *While* 1
Edge 3, 4: *Transfer Control* to *While* 4 Component

While 4 Component
Node 4: *While* there is more failure time data t
Edge 4, 5: *Transfer Control*
Node 5: *Store* t
Edge 5, 6: *Transfer Control*
Node 6: *If* **not** reached end of failure time data t input
Edge 6, 4: *Iteration Control*, *Return* to *While* 4
Edge 6, 7: *Transfer Control* to *While* 7 Component

While 7 Component
Node 7: *While* there are more failure counts x_t
Edge 7, 8: *Transfer Control*
Node 8: *Compute* cumulative x_t
Edge 8, 9: *Transfer Control*
Node 9: *Compute* failure rate λ = cumulative x_t/t_n (last failure time)
Edge 9, 10: *Transfer Control*
Node 10: *Write* failure rate λ
Edge 10, 11: *Transfer Control* to *While* 11 Component

While 11 Component
Node 11: *While* there is more failure data x_t
Edge 11, 12: *Transfer Control*
Node 12: *If* $x_t \leq 1$
Edge 12, 12a: *Transfer Control*

Independent paths:
Number of edges, e = 24
Number of nodes, n = 18
Cyclomatic complexity:
e – n + 1= 7:
Number of independent paths

A. 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18
B. 1, 2, 3, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18
C. 1, 2, 3, 4, 5, 6, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18
D. 1, 2, 3, 4, 5, 6, 7, 8, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18
E. 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 12a, 16, 17, 18
F. 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 14, 15, 16, 17, 18
G. 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 11

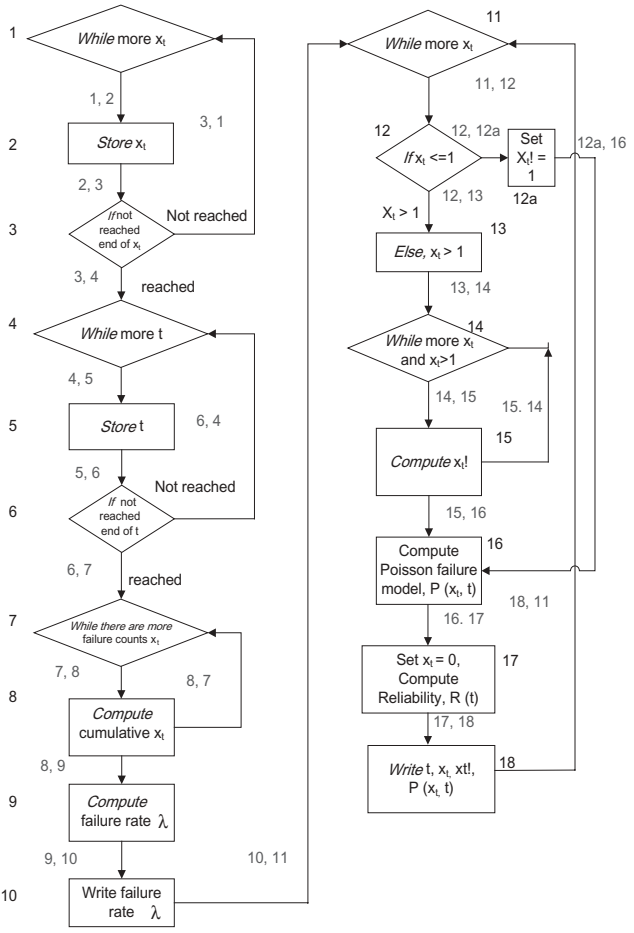


Figure 17.5 Directed graph of Poisson failure model.

Node 12a: *Set* factorial $x_t! = 1$

Edge: 12, 13: *Transfer Control*

Node 13: *Else*, case for $x_t > 1$

Edge 13, 14: *Transfer Control to While* 14 Component

While 14 Component

Node 14: *While* there is more failure data $x_t > 1$

Edge: 14, 15 *Transfer Control*

Node 15: *Compute* factorial $x_t!$, *Iteration Control*, *Return* to *While* 14

Edge 15, 16: *Transfer Control*

Node 16: *Compute* Poisson failure model, $P(x_t, t)$

Edge 16, 17: *Transfer Control*

Node 17: *Set* $x_t = 0$ in $P(x_t, t)$ and *Compute* Reliability, $R(0, t)$

Edge 17, 18: *Transfer Control*

Node 18: *Write* failure time data t, failure data x_t, factorial $x_t!$, Poisson failure model, $P(x_t, t)$, and Reliability R(t), *Iteration Control*, *Return* to *While* 11 Component

Identifying Independent Paths and Evaluating Program Test Coverage

Cyclomatic complexity is computed as $cc = e - n + 1$, where e is the number of edges (branches) and n is the number of nodes (statements) in the directed graph representation of a program. In the directed graph of Figure 17.5 there are $e = 24$ edges and $n = 18$ nodes, so $cc = 7$. This is equal to the number of *independent* paths [MCC76]. An independent path is one that cannot be constructed from other paths. This is why the last independent path G in Figure 17.5 does not repeat the logic produced by other paths. Note that independent path E corresponds to $x_t \leq 1$ and the other paths correspond to $x_t > 1$.

According to McCabe, a good test strategy is to exercise the independent paths in debugging because this strategy does a good job of exercising many, but not all, of the paths [MCC76]. This is the test strategy shown in Table 17.3, where the test paths associated with the program input variables t and x_t are listed. Unfortunately, the McCabe test strategy does not provide complete coverage of all code executions that could result in a fault. Suppose you are working with a programming language that supports exceptions (i.e., an automatic change in program flow control, such as the reception of an interrupt). An exception will cause an automatic change in control flow without the use of an instruction for testing a condition and branching on the condition. Each statement in a program could potentially cause one or more exceptions to be raised, depending on conditions (e.g., incorrect input data or division by zero). Testing for *all* possible exceptions in *all* possible places where an exception

could be raised is impractical. Therefore, a minimum acceptable level of coverage must provide assurance that all possible exceptions are raised at least once [BER]. One defense against this problem is to check the validity of input data before it is stored and to check denominators for zero prior to division operations. Thus, no test strategy is perfect, including McCabe's. A combination of methods is required to ensure adequate coverage.

In addition to providing a method for identifying independent paths and, hence, a testing strategy, cyclomatic complexity is a metric for evaluating the relative quality of software systems [MCC76, MUN96], based on the theory that higher complexity software has lower quality. For example, the cyclomatic complexity of the Poisson failure model, which equals 7, could be compared with other failure models (e.g., Weibull) in order to rank quality for the purpose of prioritizing the test effort. That is, if the Weibull model cyclomatic complexity equaled 5, more effort would be expended on testing the Poisson model.

Program Execution Results

It is insufficient to limit verification of the correctness of program output to the identification of independent paths and the associated test strategy, as in Figure 17.5 and Table 17.3. In addition, it is important to see whether the computation results appear to be reasonable. This is done in Figure 17.6, where probability of failure $P(x_t, t)$ and reliability $R(t)$ are plotted against t. An important verification step is to match $P(x_t, t)$ with $R(t)$ for $x_t = 0$; the two quantities should be equal. Indeed, they

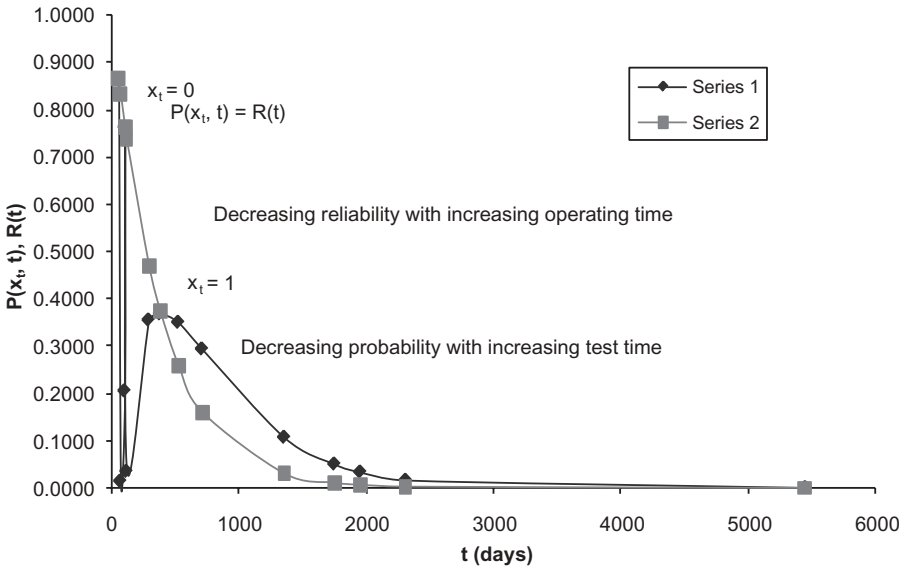


Figure 17.6 NASA space shuttle OI6: probability of failure $P(x_t, t)$ and reliability $R(t)$ versus time t. Series 1: $P(x_t, t)$. Series 2: $R(t)$.

are. Although not a part of verification, you should note whether reliability, derived from the Poisson failure model, is reasonable based on the software that the model represents. In this case it is not reasonable because the Shuttle flight software exhibits reliability growth (i.e., increases with operating time). Thus, based on Figure 17.6, the software would never be able to achieve a specified reliability, such as 0.9500. Therefore, alternate models (e.g., Weibull) would be evaluated.

Summary of Software Development Approaches

The combination of the directed graph in Figure 17.5 and the failure data in Table 17.3 provides detailed information for developing the computer code that is not possible by using the UML diagrams in Figures 17.1–17.4. However, these figures *do* provide a baseline for starting the development process that is useful for showing the big picture before getting mired in the details. Therefore, there is no "one size fits all" solution to the problem of selecting the appropriate software development paradigm. Rather, it depends on the phase of development, the system view that is desired, and the level of detail that is compatible with the phase and view.

APPLYING O-O METHODS TO MATHEMATICAL MODEL

While it has been indicated that O-O techniques are not particularly appropriate for modeling the mathematical software *product*, it can be valuable for portraying the *process* that develops the product, as Figure 17.7 attests. The Poisson failure model *process* activity diagram in Figure 17.7 documents all the steps necessary to define the components of the model and the outputs that result from each step. Thus an optimal combination of methods would be the approaches depicted in Figure 17.5 and Table 17.3 for developing software product logic, integrated with the technique portrayed in Figure 17.7 for the software development process.

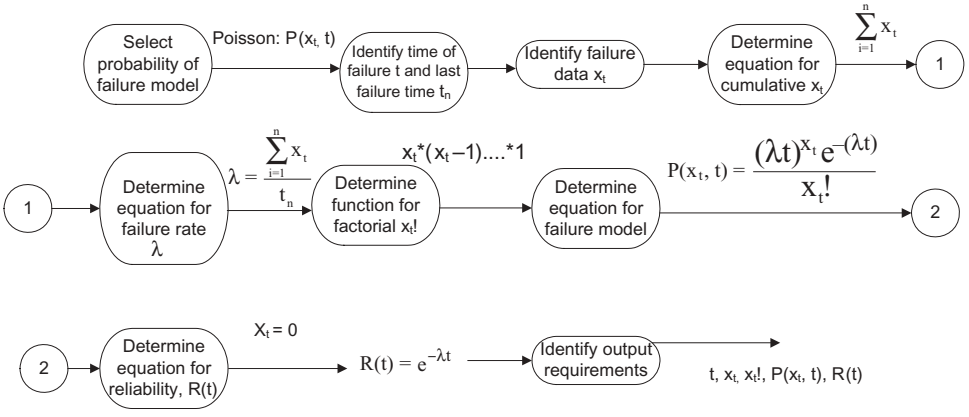


Figure 17.7 Poisson failure model process activity diagram.

Question for Reader: Is there any other factor in software development that may be more important to achieving high software reliability than the factors that have been discussed in this chapter?

Answer: The most important factor is the quality of the personnel developing the software! Unfortunately, this is a factor that is infeasible to quantify. The best that a developer organization can do is to have prospective development personnel design and program small, representative samples of the real system, and execute the programs on a computer. Then, select personnel by evaluating the results for accuracy, reliability, and quality of design documentation.

SUMMARY AND CONCLUSIONS

Where there is a great deal of application state change (e.g., elevator goes down, then goes up) accompanied by interaction of people with computerized systems (e.g., user pushes down button and signal sent to elevator computer control), O-O analysis and design is quite appropriate. These applications can be modeled with the aid of activity, sequence, and state diagrams. On the other hand, when these attributes are absent and the goal is to develop mathematical software, equations do just fine because they *are* the models of mathematics. Also, the development of mathematical software can benefit from using directed graphs to represent program logic. The benefits are threefold: equations and directed graphs are a model for writing code that is very close to the results that the equations must achieve, a model is provided for developing test strategies, and a by-product of the directed graph is a complexity metric that can be used to evaluate the reliability of the software design.

REFERENCES

[BER] Edward V. Berard, "Issues in the testing of object-oriented software", The Object Agency.

[BOO94] Grady Booch, *Object-Oriented Analysis, and Design with Applications*, 2nd ed. Redwod City, CA: The Benjamin/Cummings Publishing Company, Inc., 1994.

[BRO01] Manfred Broy, "Toward a mathematical foundation of software engineering methods," *IEEE Transactions on Software Engineering*, 2001, 27(1), pp. 42–57.

[DOU98] Bruce Powell Douglass, *Real-Time UML: Developing Efficient Objects for Embedded Systems*. Reading, MA: Addison-Wesley, 1998.

[EDE02] Amnon H. Eden, "A theory of object-oriented design," *Information Systems Frontiers*, 2002, 4, pp. 379–391.

[FRI95] Peter Fritzson, Lars Viklund, Johan Herber, and Dag Fritzson, "High-level mathematical modeling and programming," *IEEE Software*, 1995, 12(4), pp. 77–87.

[GAM95] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns: Elements of Object-Oriented Software*. Reading, MA: Addison-Wesley, 1995.

[GRA] Ian Graham and Alan Wills, UML—a tutorial, Trireme International.

[HOF97] J. Hoffman, "A practical notation for object oriented analysis with a formal meaning," Technology of Object-Oriented Languages and Systems—Tools-25, 1997, p. 225.

[KGU96] Kinh Nguyen, "Towards a practical formal method for object oriented modelling," *Third Asia-Pacific Software Engineering Conference (APSEC'96)*, 1996, p. 226.

[MCC76] T. McCabe, "A software complexity measure," *IEEE Transactions on Software Engineering*, 1976, SE-2(4), pp. 308–320.

[MUN96] J. Munson, and T. Khoshgoftaar, "Software metrics for reliability assessment," in Michael Lyu (ed.), *Handbook of Software Reliability Engineering*. New York: McGraw-Hill, 1996, pp. 493–529. Chapter 12.

[RYA06] Matt Ryan, Sule Simsek, Xiaoqing Liu, Bruce McMillin, and Ying Cheng, "An instance-based structured object oriented method for Co-analysis/Co-design of concurrent embedded systems," *30th Annual International Computer Software and Applications Conference (COMPSAC'06)*, 1, 2006, pp. 273–280.

[SAL04] M. Salah and S. Mancoridis, "A hierarchy of dynamic software views: from object-interactions to feature-interactions," *Proceedings of the 20th IEEE International Conference on Software Maintenance*, 11–14 September 2004, pp. 72–81.

[SEL04] C. Riva, P. Selonen, T. Systa, and Jianli Xu, "UML-based reverse engineering and model analysis approaches for software architecture maintenance," *Proceedings of the 20th IEEE International Conference on Software Maintenance*, 11–14 September 2004, pp. 50–59.

[SEN07] Sabnam Sengupta and Swapan Bhattacharya, "Functional specifications of object oriented systems: a model driven framework," *31st Annual International Computer Software and Applications Conference*, 1, 2007, pp. 667–672.

[ULL06] Larry Ullman and Andreas Singer, *C++ Programming*. Berkeley, CA: Peachpit Press, 2006.

[ZOU02] Ying Zou and Kostas Kontogiannis, "Quality driven transformation compositions for object oriented migration," *Ninth Asia-Pacific Software Engineering Conference (APSEC'02)*, 2002, p. 346.

[ZOU021] Y. Zou and K. Kontogiannis, "Migration to object oriented platforms: a state transformation approach," *18th IEEE International Conference on Software Maintenance (ICSM'02)*, 2002, p. 530.

APPENDIX

C++ Code for Poisson Failure Model and Reliability Based on Poisson Failure Model

```
// factorial (xt) = xt * (xt -1) . . . . * 1
// failure rate lambda = sum (xt) / tn
// Poisson failure model: P (xt, t) = (((lambda *
t)^xt) * exp (- lambda * t)) / xt!
// Reliability: R (t) = exp (- lambda * t)


#include <iostream>
#include <math.h>// specify math library
#include <stdio.h>
#include <string.h>
#include <fstream>


using namespace std;
using std::cout; // specify standard screen
output
using std::cin; // specify standard screen input
using std::endl; // specify standard end of line
```

```
int main ()

{

double xt [40], fact, failurecount, t [40] ; //
declare failure count at time t array, factorial of
xt,

// number of xt failure counts while loop control,
failure time array

int factcount, i, j; // declare factorial while loop
control, xtarray and while loop control index,

// index of number of xt failure counts

int tn, n; // last failure time, index of last
failure time

double cumfail, lambda, P, R; // declare summation
of xt, failure rate, Poisson failure model,
reliability

FILE *fp;//pointer to type FILE

fp = fopen(″c:/models/numbers1.txt″, ″w″); // file
for writing factorial output

ifstream infile ; // declare failure count xt input

infile.open(″c:/models/modelfailuredata.txt″);

ifstream infile1 ; // declare failure time t

infile1.open(″c:/models/Tdata.txt″);

i = 0;

while(!infile.eof())

// while eof not reached for xt failure count, store
data in array

{

infile >>xt [i];

if(!infile.eof()) // if eof not reached for xt
failure count, increment xt data array index

{

i = i + 1; // increment xt data array index

}

}

failurecount = xt [0]; // store number of xt failure
counts

i = 0;

while(!infile1.eof())
```

```
// while eof not reached for time of failure t, store
data in array
{
infile1 >>t [i];
if(!infile1.eof()) // if eof not reached for time of
failure, increment xt data array index
{
i = i + 1; // increment t data array index
}
}
n = t [0] ; // store last failure time index
tn = t [n]; // store last failure time


i = 1; // initialize failure count index
cumfail = 0; // initialize cumulative xt
while (i <= failurecount)// iterate while
accumulating failure count xt
{
cumfail = cumfail + xt [i]; // sum xt
i = i + 1; // increment sum xt index
}


lambda = cumfail / tn; // compute failure rate
fprintf (fp,"%s%\n", "failure rate", (char) 6);
fprintf (fp,"%f%c%\n", lambda, (char) 6);


j = 1; // initialize while loop index for each
factorial computation
i = 1; // initialize while loop index for iterating
on each factorial computation


while (j <= failurecount)// iterate while there are
still failure counts
{
if (xt [i] <=1) // simple case
{
```

```
fact = 1;
i = i + 1;
}
else // case for xt >1
{
factcount = xt [i]; // initialize failure count xt
fact = xt [j]; // initialize factorial value
while (i <= factcount && xt [i] >1) // iterate while
there are still more factors xt >1
{
fact = (fact) * (xt [i] - 1); // compute factorial
i = i + 1; // increment second while loop index
}
}
P = (pow (lambda * t [j], xt [j])* (exp (- lambda *
t [j]))) / fact ; // compute Poisson model


R = (pow (lambda * t [j], 0) * (exp (- lambda * t
[j]))) / 1 ; // compute reliability by setting xt =
0 and fact = 1 in P
fprintf (fp,"%s%\n", "failure time", (char) 6);
fprintf (fp,"%f%c%\n", t [j], (char) 6);


fprintf (fp,"%s%\n", "factorial of", (char) 6);
fprintf (fp,"%f%c%\n", xt [j], (char) 6);
fprintf (fp,"%f%c%\n", fact, (char) 6);


fprintf (fp,"%s%\n", "Poisson failure model", (char)
6);
fprintf (fp,"%f%c%\n", P, (char) 6);
fprintf (fp,"%s%\n", "reliability", (char) 6);
fprintf (fp,"%f%c%\n", R, (char) 6);
j = j +1; // increment second while loop index
}
return 0;
}
```

Chapter 18

Tutorial on Hardware and Software Reliability, Maintainability, and Availability

Computer systems, whether hardware or software, are subject to failure. Precisely, what is a failure? It is defined as: The inability of a system or system component to perform a required function within specified limits. A failure may be produced when a fault is encountered and a loss of the expected service to the user results [IEE07]. This brings us to the question of what is a fault? A fault is a defect in the hardware or computer code that can be the cause of one or more failures [IEE07]. Software-based systems have become the dominant player in the computer systems world. It is imperative that computer systems operate reliably, considering the criticality of software, particularly in safety critical systems. Software and hardware do not operate in a vacuum. Therefore, both software and hardware are addressed in this tutorial in an integrated fashion. The narrative of the tutorial is augmented with illustrative solved problems.

It is important for an organization to have a disciplined process if it is to produce high reliability software. This process uses a life-cycle approach to software reliability that takes into account the risk to reliability due to requirements changes. A requirements change may induce ambiguity and uncertainty in the development process that may cause errors in implementing the changes. Subsequently, these errors may propagate through later phases of development and maintenance [SCH01]. In view of the life-cycle ramifications of the software reliability process, maintenance is included in this tutorial. Furthermore, because reliability and maintainability determine availability, the latter is also included.

Computer, Network, Software, and Hardware Engineering with Applications, First Edition. Norman F. Schneidewind.
© 2012 the Institute of Electrical and Electronics Engineers, Inc. Published 2012 by John Wiley & Sons, Inc.

RELIABILITY BASICS

To set the stage for discussing software and hardware model, the following definitions and concepts are provided:

Component: Any hardware or software entity, such as a module, subsystem, or system.

t: Operating time.

$P(T \leq t)$: Probability that operating time T of a component is $\leq t$ (also known as cumulative distribution function [CDF]).

λ: Failure rate (software or hardware failure rate).

Reliability R(t): $P(T > t)$: probability of software or hardware surviving for $T > t = 1 - P(T \leq t)$ [LYU96].

Hazard function: letting operating time t have the probability density function p(t), the *instantaneous failure rate* at time t, defined as [LYU96]:

$$h(t) = p(t)/R(t), \tag{18.1}$$

where p(t) is defined as the probability that a failure will occur in the interval t, t + 1.

The hazard function is frequently described in reliability literature, but a reliability metric that is more practical for calculations with empirical data is the failure rate f(t). This is defined as the number of failures n(t) in the interval t divided by t: $f(t) = n(t)/t$. The reason the hazard function may be impractical when dealing with empirical data is that the probability density function p(t) may not be known.

HARDWARE RELIABILITY

The exponential failure distribution with constant failure rate is particularly applicable to hardware reliability because it is assumed that the failure rate remains constant after the initial burn in period and before wear out occurs.

Exponential Failure Distribution: $\lambda e^{-\lambda t}$

This distribution has a constant failure rate λ. The exponential distribution is the only failure distribution that has a constant failure rate λ and a constant hazard function h(t) in the operations phase of the life cycle. This failure rate is $= 1/\overline{t}$, where $\overline{t}$ is the mean time to failure (MTTF).

Then, the reliability is:

$$R(t) = e^{-\lambda t}. \tag{18.2}$$

Then using Equation 18.1, the hazard function for exponentially distributed failures is:

$$h(t) = p(t)/R(t) = \lambda e^{-\lambda t}/e^{-\lambda t} = \lambda. \tag{18.3}$$

Then adapting Equation 18.2 to use MTTF, Equation 18.4 is produced:

$$R(t) = e^{-(t/\bar{t})}. \tag{18.4}$$

If we wish to solve for t for a given value of R(t), Equation 18.5 is solved for t:

$$t = -\ln(R(t))\bar{t}. \tag{18.5}$$

Problem

Specifications

1. Hardware in a computer system should have an expected (**mean**) life $\bar{t} > \mathbf{100,000}$ (MTTF) hours at a reliability of R(t) = 0.85. What is the minimum number of hours t the computer system would have to survive to meet these specifications?
2. If the hardware should have a 0.85 probability of surviving (i.e., reliability) for t > 50,000 hours, what is the MTTF required to meet these specifications?

Solution

1. Use Equation 18.5 to compute t:

$$t = -\ln(0.85)(100,000) = -(-0.1625)(100,000) = \mathbf{16,250} \text{ hours.}$$

2. Solve Equation 18.26 for $\bar{t}$:

$$\bar{t} = t/[-\ln(R(t))] = 50,000/[-\ln(0.85)] = \mathbf{307,692} \text{ hours.}$$

MULTIPLE COMPONENT RELIABILITY ANALYSIS

Due to the fact that the majority of computer systems in the industry employ multiple components, the reliability analysis must be focused on predicting reliability for these systems. Hardware (and software) components can be operated in serial or hardware configurations. In hardware, the differences are more obvious because of the physical connection between components. In software, the difference is not obvious because there is no physical connection. The difference is based on how the components execute, as indicated in Figure 18.1.

Parallel System

As Figure 18.1 shows, the purpose of a parallel system is to provide a redundant configuration so that if one component fails, another component can take its place, thus increasing reliability. The reliability of a single component i, operating for a time t, is designated by $R_i(t)$. The unreliability is then $(1 - R_i(t))$.

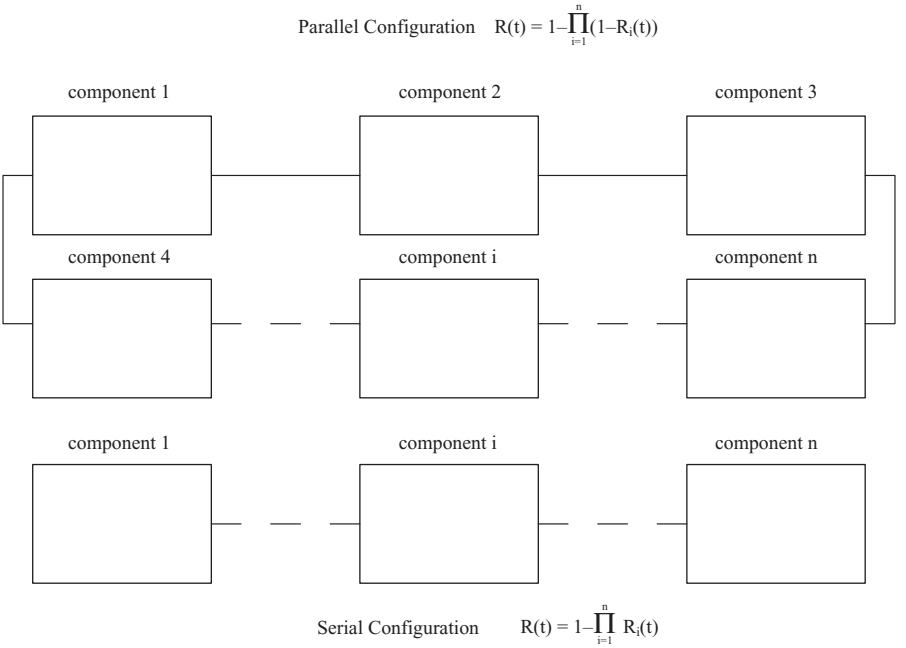


Figure 18.1 Parallel and serial reliability configurations. Parallel hardware, components physically connected in parallel; parallel software, components execute concurrently in time; serial hardware, components physically connected in series; serial software, components execute serially in time.

Referring to Figure 18.1, the reliability of n components operating in parallel is given by [MUS87]:

$$R(t) = 1 - \prod_{i=1}^{n} (1 - R_i(t)). \tag{18.6}$$

This equation is obtained by observing that the unreliability of n components in parallel is computed by the product of the individual component unreliabilities. Then, the reliability of n components is obtained by subtracting this product from 1.

The most common parallel configuration involves using two components, so using Equation 18.6 and some algebraic manipulation, the reliability of two components operating in parallel is given by:

$$R(t) = [R_1(t) + R_2(t)] - [R_1(t)R_2(t)] = 1 - [(1 - R_1(t))(1 - R_2(t))]. \tag{18.7}$$

If both components have the same reliability, then:

$$R(t) = 2R(t) - R^2(t). \tag{18.8}$$

A traditional assumption in reliability is that the time between failures is exponentially distributed [LYU96]. This is based on the idea that there is a higher probability

of small times between failures and a low probability of large times between failures. Therefore, when failures are exponentially distributed with failure rate λ, then the reliability in Equation 18.1.2 becomes:

$$R(t) = 2e^{-\lambda t} - e^{-2\lambda t}. \tag{18.9}$$

MTTF refers to the average time *to* the next failure [MUS87]. It is a common metric for hardware reliability because the physics of failures is well understood. However, it can be misleading because equipment will fail at specific times and not according to a mean value! MTTF is even less applicable for software because the distribution of time when software fails can be erratic. Before proceeding further, it is important to note that just because the *distribution* of failure times for both hardware and software is a better metric of reliability, it does not mean that MTTF and mean time between failures (MTBF; see below) are not used! These metrics have become so embedded in the lore of reliability that it is imperative to describe their usage.

In the case of hardware, MTTF is used when components are not repaired (i.e., replaced). In other words, with no repair, the time to next failure is *direct*, with no intervening repair time. In nonredundant software systems, the software must be repaired to continue operation, unless the fault causing the failure is trivial. Therefore, MTTF is not completely applicable for this type of software. On the other hand, for redundant software systems (e.g., fault tolerant), MTTF is applicable, with the caveat noted above.

MTBF, defined as the average time *between* failures, is used when components are repaired [MUS87]. Thus, it is the time between failures, with an intervening repair time.

The general form for MTTF, whether hardware or software, is derived from the reliability function R(t), as follows: $\int_0^\infty R(t)dt$ [LYU96].

Therefore, the MTTF for the two component parallel arrangement, from Equation 18.1.3, is given by:

$$\bar{t} = \int_0^\infty R(t)dt = \int_0^\infty (2e^{-\lambda t} - e^{-2\lambda t})dt = \left[\frac{-2e^{-\lambda t}}{\lambda}\right]_0^\infty - \left[\frac{-e^{-2\lambda t}}{2\lambda}\right]_0^\infty = \frac{1.5}{\lambda}. \tag{18.10}$$

Series System

Often, particularly for software systems, in order to produce a conservative prediction of reliability, components are assumed to operate in series for the *purpose* of reliability prediction [KEL97]. This represents the weakest link in the chain concept (i.e., the system would fail if *any* component fails).

Then this conservative reliability approach of *n* components operating in series is given by [MUS87]:

$$R(t) = \prod_{i=1}^{n} R_i(t). \tag{18.11}$$

Using Equation 18.11, the reliability of two components operating in series, with equal reliabilities, is given by Equation 18.12, if the failures are exponentially distributed:

$$R(t) = R^2(t) = e^{-2\lambda t}. \tag{18.12}$$

Then, the MTTF for the series arrangement is given next:

$$\bar{t} = \int_0^\infty R(t)dt = \int_0^\infty e^{-2\lambda t} = \frac{-\left[e^{2\lambda t}\right]_0^\infty}{2\lambda} = \frac{1}{2\lambda}. \tag{18.13}$$

It is often of interest to predict the improvement that can be achieved by using a parallel rater than a series configuration. Then, using Equations 18.9 and 18.12, the improvement of the parallel system reliability over a series system, for two components, can be shown as:

$$RI = (2e^{-\lambda t} - e^{-2\lambda t}) - e^{-2\lambda t} = 2(e^{-\lambda t} - e^{-2\lambda t}). \tag{18.14}$$

In addition, using Equations 18.10 and 18.13, the increase in *MTTF* can be shown to be:

$$\frac{1.5}{\lambda} - \frac{1}{2\lambda} = 1 \setminus \lambda. \tag{18.15}$$

It is not only the improvement RI that is of interest. In addition, the rate of change of RI will reveal the rate of change of RI that will indicate how fast the improvement will occur. Then, differentiating RI (Eq. 18.14) with respect to t, and setting it = 0, gives us Equation 18.16:

$$\frac{d(RI)}{d(t)} = 2(-\lambda)e^{-\lambda t} - 2(-2\lambda)e^{-2\lambda t} = 0. \tag{18.16}$$

Noting that the derivative of Equation 18.16 is negative, because the first negative term decreases less rapidly than the second positive term, we know that Equation 18.16 will provide a value of t that will maximize RI.

Then, solving Equation 18.1.10 for t yields t* as the value of t that maximizes RI:

$$t^* = -(1/\lambda)(\log(0.5)). \tag{18.17}$$

Problem: For a computer system with failure rate of $\lambda = 0.001$ failures per hour and *time to failure* listed below, plot Equations 18.3, 18.6, and 18.14 on the same graph, versus t, and indicate the value of $t = t^*$ that maximizes RI, assuming an exponential distribution of time to failure t.

t (hours)

100

200
300
400
500
600
700
800
900
1000
1100
1200
1300
1400
1400
1500
1600
1700
1800
1900
2000

Solution: Figure 18.2 contrasts parallel reliability, serial reliability, and the improvement of parallel over serial reliability. The figure also delineates the operating time where the greatest improvement is achieved. A reliability analyst, using this plot, would understand that at t = 683 hours the greatest gain in reliability would occur and that at operating times either below or above this value, the gain falls off rapidly.

Number of Components that are Needed to Achieve Reliability Goals

When the reliability of a system is required to be $R_n(t)$ in a parallel configuration, the required number n components, each with a reliability of R(t), is:

$$R_n(t) = 1 - (1 - R(t))^n. \tag{18.18}$$

Solving Equation 18.18 for n yields:

$$n = \ln(1 - R_n(t)) / \ln(1 - R(t)). \tag{18.19}$$

Problem: How many components are needed to operate in parallel, if each component has a reliability of R(t) = 0.80, and it is desired to achieve a system reliability of $R_n(t) = 0.98$?

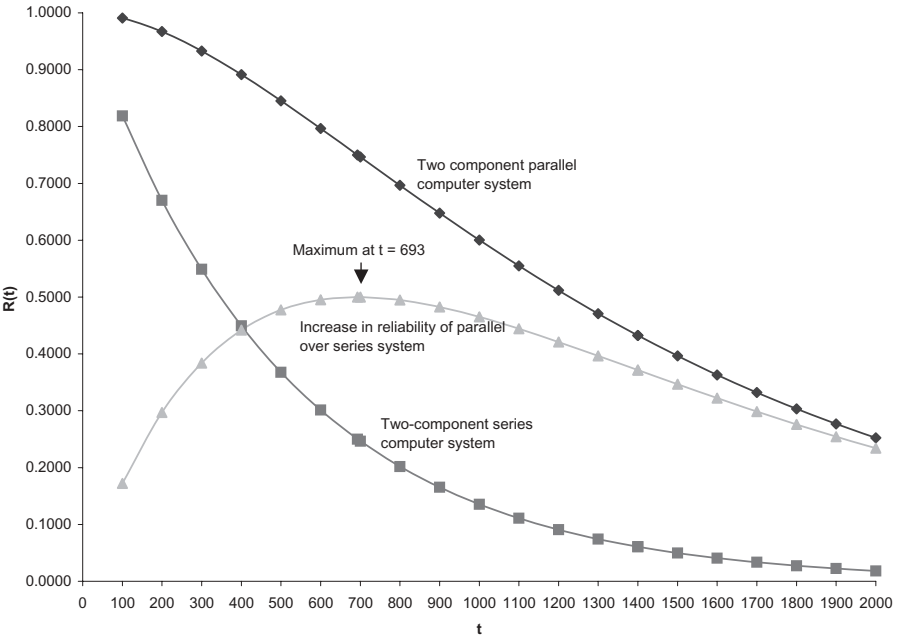


Figure 18.2 Reliability R(t) versus operating time t.

Solution: Solving Equation 18.18 for *n*, yields:

$$n = \ln(1 - R_n(t))/\ln(1 - R(t)) = (\ln(0.02)/\ln(0.20))$$
$$= -3.912/-1.609 = 2.43 \text{ components} = \mathbf{3}.$$

COMPUTER SYSTEM MAINTENANCE AND AVAILABILITY

Preventive Maintenance Strategy. Routine inspection and service activities designed to detect potential failure conditions and make adjustments and repairs that will help prevent major operating problems [MON96].

Two fundamental preventive strategies are differentiated, *time- and condition-based preventive maintenance*. In time-based preventive maintenance, after a fixed period of time, a component is serviced or overhauled, independent of the wear of the component at that moment. In condition-based preventive maintenance, one inspects a condition of a component, according to some schedule. If the condition exceeds a specified critical value, preventive maintenance is performed. With regard to the timing of the inspections, there are two variants, *constant* and *condition-based inspection interval*. If one applies a constant inspection interval, an inspection is performed after a fixed period of time, analogous to time-based preventive maintenance. When

deciding to perform a condition-based inspection interval, the time until the next inspection depends on the condition in the previous inspection. If the condition in the previous inspection was good, the time until the next inspection will be quite long. If the condition in the previous inspection was bad, the time until the next inspection will be quite short.

Predictive Maintenance Strategy. Predictive maintenance is a condition-based approach to maintenance. The approach is based on predicting component condition in order to assess whether components will fail during some future period, and then taking action to avoid the consequences of the failures.

COMPONENT AVAILABILITY

Now, in order to compute component availability, a number of quantities must be defined:

t_p: duration of component preventive maintenance

t_o: duration of component operation

t_f: duration of component failure

t_r: duration of component repair

f_p: frequency of component preventive maintenance

f_o: frequency of component operation

f_f: frequency of component failures

f_r: frequency of component repair

$\bar{t}$: mean time to component failure

With the definitions in hand, availability A, can be computed:

$$A = \frac{f_o t_o}{f_o t_o + f_p t_p + f_f t_f + f_r t_r}. \tag{18.20}$$

Availability is also expressed by:

$$A = \bar{t}/(\bar{t} + t_r). \tag{18.21}$$

These quantities are portrayed graphically in Figure 18.3.

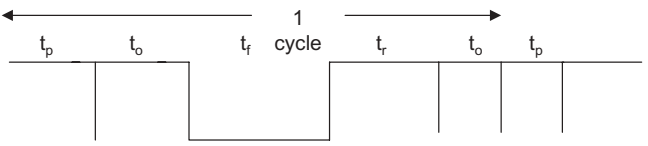


Figure 18.3 Computer maintenance process. t_p, duration of preventive maintenance; t_o, duration of operation; t_f, duration of failure; t_r, duration of repair.

Problem: Given the data below for a system, compute the availability A.

Duration of operation: $t_o = 10$

Duration of preventive maintenance: $t_p = 1$

Duration of failure: $t_f = 0.5$

Duration of repair $t_r = 2$

Frequency of operation: $f_o = 20$

Frequency of preventive maintenance: $f_p = 20$ (for every operation there is preventive maintenance)

Frequency of failure: $f_f = 4$

Frequency of repair: $f_r = 4$ (for every failure there is a repair)

Then, using Equation 18.20:

$$A = \frac{f_o t_o}{f_o t_o + f_p t_p + f_f t_f + f_r t_r} = \frac{(20)(10)}{(20)(10) + (20)(1) + (4)(0.5) + (4)(2)}$$
$$= 0.870 \text{ (system availability).}$$

SOFTWARE RELIABILITY ENGINEERING RISK ANALYSIS

Software reliability engineering (SRE) is an established discipline that can help organizations improve the reliability of their products and processes. The IEEE/American Institute of Aeronautics and Astronautics (AIAA) defines SRE as "the application of statistical techniques to data collected during system development and operation to specify, predict, estimate, and assess the reliability of software-based systems." The IEEE/AIAA recommended practice is a composite of models and tools and describes the "what and how" of SRE [IEE07]. It is important for an organization to have a disciplined process if it is to produce high reliability software. The process includes a life-cycle approach to SRE that takes into account the risk to reliability due to requirements changes. A requirements change may induce ambiguity and uncertainty in the development process that cause errors in implementing the changes. Subsequently, these errors may propagate through later phases of development and maintenance. These errors may result in significant risks associated with implementing the requirements. For example, reliability risk (i.e., risk of faults and failures induced by changes in requirements) may be incurred by deficiencies in the process (e.g., lack of precision in requirements). Figure 18.4 shows the overall SRE closed-loop holistic process

In the figure, risk factors are metrics that indicate the degree of risk in introducing a new requirement or making a requirements change. For example, in the National Aeronautics and Space Administration (NASA) Space Shuttle, program size and complexity, number of conflicting requirements, and memory requirements have been shown to be significantly related to reliability (i.e., increases in these risk factors are associated with decreases in reliability) [SCH07]. Organizations should

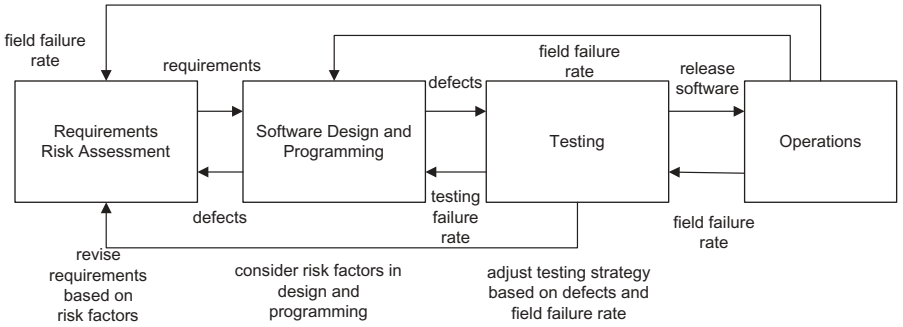


Figure 18.4 Software reliability engineering risk analysis.

conduct studies to determine what factors are contributing to reliability degradation. Then, as in Figure 18.4, organizations could use feedback from operations, testing, design, and programming to determine which risk factors are associated with reliability, and revise requirements, if necessary. For example, if requirements risk assessment finds that through risk factor analysis that defects are occurring because of excessive program size, design and programming would receive revised requirements to modularize the software.

A reliability risk assessment should be based on the risk to reliability due to software defects or errors caused by requirements and requirements changes. The method to ascertain risk based on the number of requirements and the impact of changes to requirements is inexact, but nevertheless, it necessary for early requirements assessments of large-scale systems.

Criteria for Safety

In safety-critical systems in particular, safety criteria are used in conjunction with risk factors to assess whether a system is safe to operate. Two criteria are used. One is based on predicted remaining failures in relation to a threshold and the second is based on the predicted time to next failure in relation to mission duration [SCH97]. These criteria are computed as follows:

Compute predicted *remaining failures* $r(t_t) < r_c$, where r_c is a specified remaining failures critical value, and compute predicted *time to next failure* $T_F(t_t) > t_m$, where t_m is mission duration.

Once $r(t_t)$ has been predicted, the risk criterion metric (RCM) for *remaining failures* at total test time t_t is computed in Equation 18.22:

$$\text{RCM } r(t_t) = \frac{r(t_t) - r_c}{r_c} = \frac{r(t_t)}{r_c} - 1. \tag{18.22}$$

In order to illustrate the remaining failure risk criterion in relation to the predicted maximum number of failures in the software $F(\infty)$, the following parameter is needed:

p(t): Fraction of remaining failures predicted at time t_t in Equation 18.23:

$$p(t_t) = \frac{r(t_t)}{F(\infty)}. \tag{18.23}$$

The RCM for *time to next failure* at total test time t_t is computed in Equation 18.24 based on the predicted time to next failure in Equation 18.25 [SCH07]:

$$\text{RCM } T_F(t_t) = \frac{t_m - T_F(t_t)}{t_m} = 1 - \frac{T_F(t_t)}{t_m}, \tag{18.24}$$

$$T_f(t_t) = -\frac{1}{\beta}\log\left[1 - ((F(t_t) + X_{s-1}))\left(\frac{\beta}{\alpha}\right)\right] + (s-1), \text{ for } (F(t_t) + X_{s-1})\left(\frac{\beta}{\alpha}\right) < 1, \tag{18.25}$$

where β and α are parameters estimated from the failure data. Parameter β is the rate of change of the failure rate and α is the initial failure rate. The parameter s is the starting failure interval count that produces the most accurate reliability predictions, and X_{s-1} is the observed failure count in the range of the test data from s to t_t. Finally, $F(t_t)$ refers to the specified number of failures—usually one—that is used in the prediction.

Problem

Part 1: Remaining Failures Risk

Using one of the models in IEEE/AIAA [IEE07], recommended for initial use, and either the software reliability tool Statistical Modeling and Estimation of Reliability Functions for Software (SMERFS) or CASRE, compute Equations 18.2.3 and 18.2.4 to produce Figures 18.5 and 18.6 for the NASA Space Shuttle software release OI6. The failure counts for each value of test time t_t for OI6 are shown in Table 18.1. Once you have inputted a text file of these counts, one at a time, the software reliability tools will compute $r(t_t)$ and $F(\infty)$for each of the 10 cases. The tools can be downloaded at http://www.slingcode.com/smerfs/ for SMERFS and at http://www.openchannelfoundation.org/projects/CASRE_3.0 for CASRE.

Part 2: Time to Next Failure Risk

In this part, a specific recommended model in [IEE07] is used [SCH97] in order to illustrate the use of this model's predicted time to next failure and the application of the prediction to evaluating the risk of not satisfying the mission duration requirement, as formulated in Equation 18.24. Other recommended models could be used to perform the analysis.

After using one of the tools to estimate the parameters in Equation 18.24, predict $T_F(t_t)$ for one more failure and plot it and the RCM, in Figure 18.7, as a function of the test time t_t in Table 18.1.

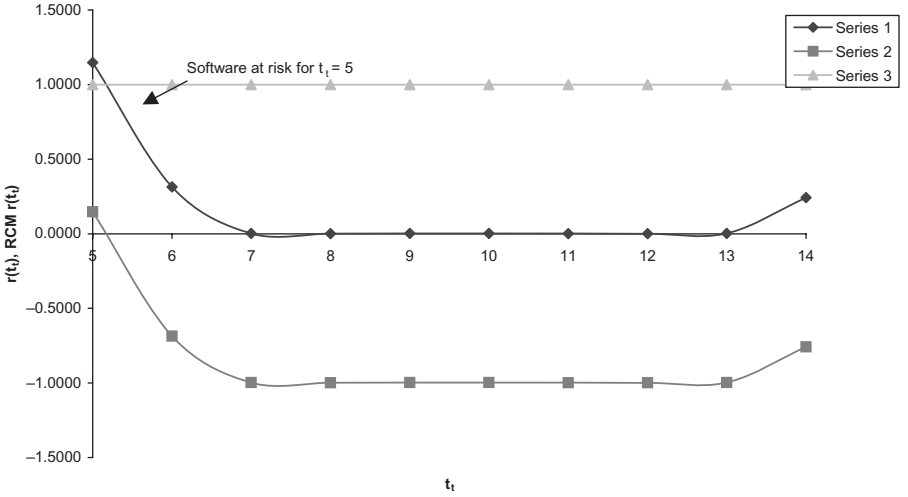


Figure 18.5 Predicted remaining failures $r(t_t)$ and risk criterion metric RCM $r(t_t)$ versus test time t_t for NASA space shuttle release OI6. Series 1: $r(t_t)$. Series 2: RCM $r(t_t)$. Series 3: remaining failures critical value r_c.

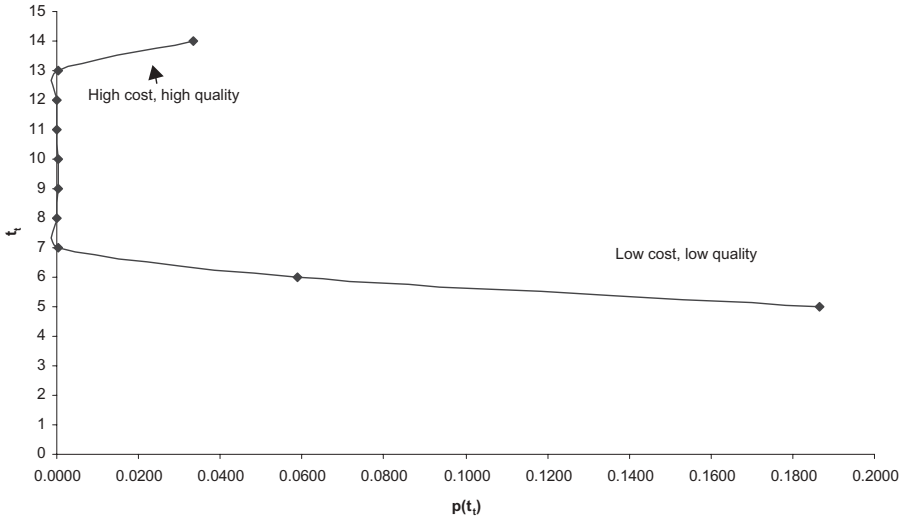


Figure 18.6 Cost of testing t_t versus software quality $p(t_t)$ for NASA space shuttle release OI6. Test time t_t represents cost, fraction remaining failures $p(t_t)$ represents software quality.

Solution to Part 1: Figure 18.5 delineates the test time = 5, where the risk of exceeding the critical value of remaining failures is unacceptable. Therefore, a test time of at least 6 is required. Figure 18.6 shows how the software reliability analyst can do a trade-off of the cost of testing version with the quality of software produced by testing. Since test time is usually directly related to

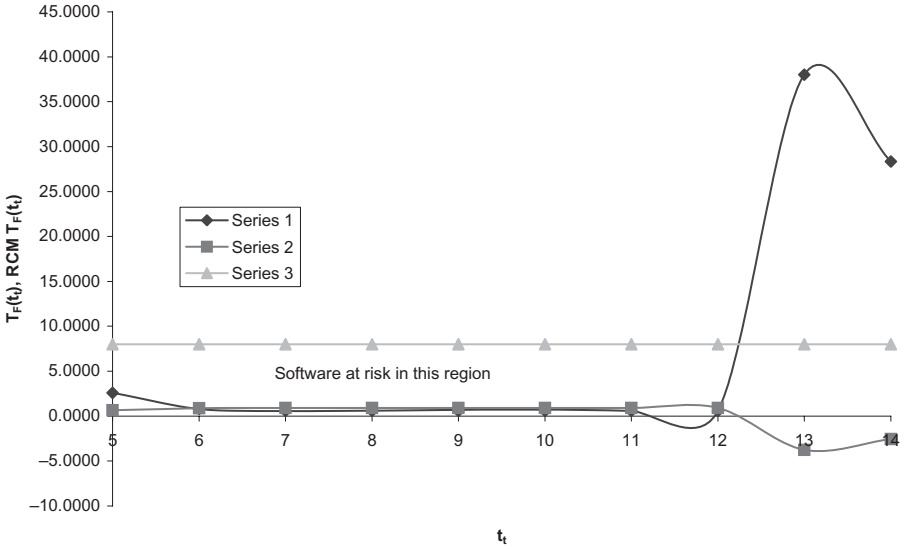


Figure 18.7 Predicted time to next failure T_F (t_t) and risk criterion metric RCM $T_F(t_t)$ versus test time t_t. Series 1: $T_F(t_t)$. Series: 2: RCM $T_F(t_t)$. Series 3: Mission duration: t_m.

Table 18.1 Failure Counts for NASA Space Shuttle Software Release OI6

t_t	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**
	0	0	0	0	0	0	0	0	0	0
	2	2	2	2	2	2	2	2	2	2
	1	1	1	1	1	1	1	1	1	1
	2	2	2	2	2	2	2	2	2	2
	0	0	0	0	0	0	0	0	0	0
		0	0	0	0	0	0	0	0	0
			0	0	0	0	0	0	0	0
				0	0	0	0	0	0	0
					0	0	0	0	0	0
						0	0	0	0	0
							0	0	0	0
								0	0	0
									1	1
										1

cost, the figure indicates that a very high cost would be incurred for attempting to achieve almost fault-free software. Therefore, tolerating a fraction of remaining failures of about 0.0600 would be practical.

Solution to Part 2: Switching now to the evaluation of risk with respect to time to next failure, Figure 18.7 demonstrates that unless the test time is greater than 12, the time to failure will not exceed the mission duration. The engineer using such a plot would use a mission duration appropriate for the software being tested. The concept behind Figure 18.7 is that the software should be tested sufficiently long such that the RCM goes negative.

PARAMETER ANALYSIS

It is possible to assess risk after the parameters α and β have been estimated by a tool, such as SMERFS and CASRE [IEE07], but *before* predictions are made. An example is provided in Figure 18.8, where remaining failures and its risk criterion are plotted against the parameter ratio (PR) β/α [SCH07]. The reason for this result is that a high value of β means that the failure rate decreases rapidly, and coupled with a low value of α, leads to high reliability. High reliability in turn means low risk of unsafe software. Furthermore, increasing values of PR are associated with increasing values of test time, thus decreasing risk. Thus, even *before* predictions are made, it is possible to know how much test time is required to yield predictions that the software is safe to deploy. In Figure 18.8, this time is 6, corresponding to the same result in Figure 18.7. A cautionary note is that the foregoing analysis is an

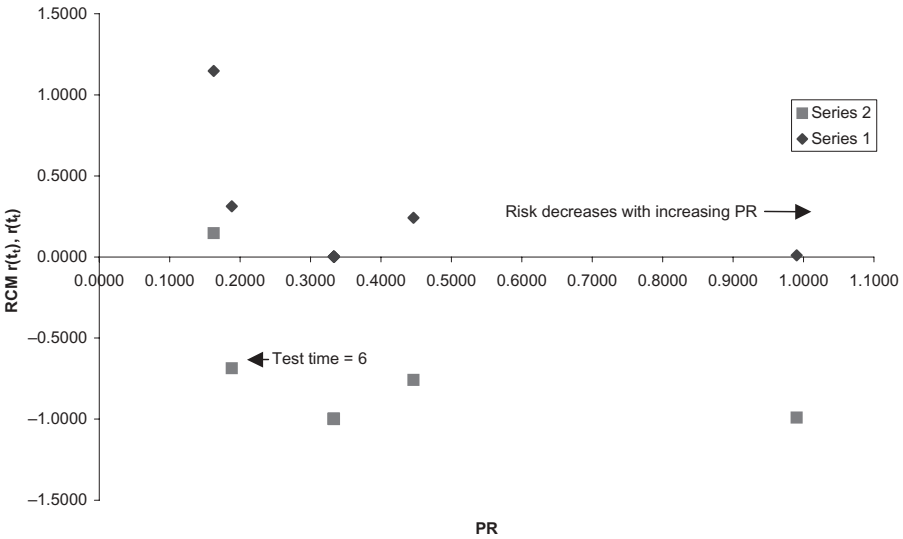


Figure 18.8 Risk criterion metric: RCM $r(t_t)$ and remaining failures $r(t_t)$ versus parameter ratio PR (beta/alpha) for NASA space shuttle software release OI6. Series 1: Predicted $r(t_t)$. Series 2: RCM $r(t_t)$.

a priori assessment of likely risk results and does not mean, necessarily, that high values of β/α will lead to low risk.

Problem: After obtaining estimates of β and α using one of the reliability tools for each value of test time in Table 18.1, plot Figure 18.8 to show that risk decreases with the PR.

OVERVIEW OF RECOMMENDED SOFTWARE RELIABILITY MODELS

In IEEE/AIAA [IEEE07], it is stated that there are "initial models" recommended for using on an application, but if these models do not satisfy the organization's need, other models that are described in the document could be used. Since this tutorial has included several practice problems, based in part on models, an overview is presented of two of the initially recommended models: Musa–Okumoto and Schneidewind. The third model—generalized exponential—involves a great amount of detail that cannot be presented here. For readers interested in more detail on these models or to learn about the other models, the recommended practice can be consulted.

MUSA–OKUMOTO LOGARITHMIC POISSON EXECUTION TIME MODEL

Objectives

The logarithmic Poisson model is applicable when the testing is done according to an operational profile that has variations in frequency of application functions and when early fault corrections have a greater effect on the failure rate than later ones. Thus, the failure rate has a decreasing slope. The operational profile is a set of functions and their probabilities of use [MUS99].

Assumptions

The assumptions for this model are:

- The software is operated in a similar manner as the anticipated operational usage.
- Failures are independent of each other.
- The failure rate decreases exponentially with execution time.

Structure

From the model assumptions we have:

$\lambda(t)$ = failure rate after t amount of execution time has been expended $\lambda_0 e^{-\theta\mu(t)}$.

The parameter λ_0 is the initial failure rate parameter and θ is the failure rate decay parameter with $\theta > 0$.

Using a reparameterization of $\beta_0 = \theta^{-1}$ and $\beta_1 = \lambda_0\theta$, the estimates of β_0 and β_1 are made, as shown in Equations 18.26 and 18.27, respectively:

$$\hat{\beta}_0 = \frac{n}{\ln(1+\hat{\beta}_1)t_n}, \tag{18.26}$$

$$\frac{1}{\hat{\beta}_1}\sum_{i=1}^{n}\frac{1}{1+\hat{\beta}_1 t_i} = \frac{nt_n}{(1+\hat{\beta}_1 t_i)\ln(1+\hat{\beta}_1 t_i)}. \tag{18.27}$$

Here, t_n is the cumulative central processing unit (CPU) time from the start of the program to the current time. During this period, n failures have been observed. Once estimates are made for β_0 and β_1, the estimates for θ and λ_0 are made in Equations 18.28 and 18.29:

$$\hat{\theta} = \frac{1}{n}\ln\left(1+\hat{\beta}_1 t_n\right), \tag{18.28}$$

$$\hat{\lambda}_0 = \hat{\beta}_0\hat{\beta}_1. \tag{18.29}$$

Limitation

The failure rate may rise as modifications are made to the software violating the assumption of decreasing failure rate.

Data Requirements

The required data are either:

The time between failures, represented by X_i's.

The time of the failure nth occurrences, given by $t_n = \sum_{i=1}^{n} X_i$.

Applications

The major applications are described below. These are separate but related applications that, in total, comprise an integrated reliability program.

Prediction. Predicting future failure times and fault corrections.

Control. Comparing prediction results with predefined goals and flagging software that fails to meet goals.

Assessment. Determining what action to take for software that fails to meet goals (e.g., intensify inspection, intensify testing, redesign software, and revise process). The formulation of test strategies is also a part of assessment.

It involves the determination of priority, duration, and completion date of testing, and allocation of personnel and computer resources to testing.

Reliability Predictions

In Musa et al. [MUS87], it is shown that from the assumptions above and the fact that the derivative of the mean value function of failure count is the failure rate function, Equation 18.30 is obtained:

$$\hat{\mu}(\tau) = \text{mean number of failures experienced by time } \tau \text{ is expended} = \frac{1}{\hat{\theta}} \ln\left(\hat{\lambda}_0 \hat{\theta} \tau + 1\right). \tag{18.30}$$

Implementation and Application Status

The model has been implemented by the Naval Surface Warfare Center, Dahlgren, Virginia as part of SMERFS and in CASRE.

SCHNEIDEWIND MODEL

Objectives

The objectives of this model [SCH97] are to predict following software reliability metrics:

- $F(t_1, t_2)$: Predicted failure count in the range $[t_1, t_2]$
- $F(\infty)$: Predicted failure count in the range $[1, \infty]$; maximum failures over the life of the software
- $F(t)$: Predicted failure count in the range $[1, t]$
- $p(t)$: Fraction of remaining failures predicted at time t
- $Q(t)$: Operational quality predicted at time t; the complement of $p(t)$; the degree to which software is free of remaining faults (failures)
- $r(t_t)$: Remaining failures predicted at test time t_t
- t_t: Test time predicted for given $r(t_t)$
- $T_F(t_t)$: Time to next failure predicted at test time t_t

Parameters Used in the Predictions

- α: Initial failure rate
- β: Rate of change of failure rate

- r_c: Critical value of remaining failures used in computing the RCM for remaining failures: (RCM) $r(t_t)$
- t_m: Mission duration (end time–start time) used in computing the RCM for time to next failure: RCM $T_F(t_t)$

The philosophy of this model is that as testing proceeds with time, the failure detection process changes. Furthermore, recent failure counts are usually of more use than earlier counts in predicting the future. Three approaches can be employed in utilizing the failure count data (i.e., number of failures detected per unit of time). Suppose there are t intervals of testing and f_i failures were detected in the ith interval, one of the following is done:

- Use all of the failures for the t intervals.
- Ignore the failure counts completely from the first s – 1 time intervals ($1 \le s \le t$) and only use the data from intervals s through t.
- Use the cumulative failure count from intervals 1 through s – 1: $F_{s-1} = \sum_{i=1}^{s-1} f_i$.

The first approach should be used when it is determined that the failure counts from all of the intervals are useful in predicting future counts. This would be the case with new software where little is known about its failure count distribution. The second approach should be used when it is determined that a significant change in the failure detection process has occurred and thus only the last t–s + 1 intervals are useful in future failure forecasts. The last approach is an intermediate one between the other two. Here, the combined failure counts from the first s – 1 intervals and the individual counts from the remaining intervals are representative of the failure and detection behavior for future predictions. This approach is used when the first s – 1 interval failure counts are not as significant as in the first approach, but are sufficiently important not to be discarded, as in the second approach.

Assumptions

- The number of failures detected in one interval is independent of the failure count in another. Note that in practice, this assumption has not proved to be a factor in obtaining prediction accuracy.
- Only new failures are counted.
- The fault correction rate is proportional to the number of faults to be corrected.
- The software is tested in a manner similar to the anticipated operational usage.
- The mean number of detected failures decreases from one interval to the next.
- The rate of failure detection is proportional to the number of failures within the program at the time of test. The failure detection process is assumed to be a nonhomogeneous Poisson process with an exponentially decreasing failure detection rate [SCH07]. The rate is of the form $f(t) = \alpha e^{-\beta(t-s+1)}$ for the tth interval where $\alpha > 0$ and $\beta > 0$ are the parameters of the model.

Structure

The method of maximum likelihood (MLE) is used to estimate parameters. This method is based on the concept of maximizing the probability that the true values of the parameters are observed in the failure data [MUS99]. Two parameters are used in the model that were previously defined: α and β. In these estimates, t is the last observed failure count interval; s is the starting interval for using observed failure data in parameter estimation; X_k is the number of observed failures in interval *k*; X_{s-1} is the number of failures observed from 1 through s–1 intervals; $X_{s,t}$ is the number of observed failures from interval s through t; and $X_t = X_{s-1} + X_{s,t}$. The likelihood function (based on MLE) is then developed as:

$$\begin{aligned}\log L = {} & X_t\left[\log X_t - 1 - \log\left(1 - e^{-\beta t}\right)\right] \\ & + X_{s-1}\left[\log\left(1 - e^{-\beta(s-1)}\right)\right] \\ & + X_{s,t}\left[\log\left(1 - e^{-\beta}\right)\right] - \beta\sum_{k=0}^{t-s}(s+k-1)X_{s+k}.\end{aligned} \tag{18.31}$$

Equation 18.31 is used to derive the equations for estimating α and β for each of the three approaches described earlier. The parameter estimates can be obtained by using the SMERFS or CASRE tools.

Approach 1

Use all of the failure counts from interval 1 through t (i.e., s = 1). Equations 18.32 and 18.33 are used to estimate β and α, respectively:

$$\frac{1}{e^{\beta} - 1} - \frac{t}{e^{\beta t} - 1} = \sum_{k=0}^{t-1} k\frac{X_{k+1}}{X_t}, \tag{18.32}$$

$$\alpha = \frac{\beta X_t}{1 - e^{-\beta t}}. \tag{18.33}$$

Approach 2

Use failure counts only in intervals s through t (i.e., $1 \le s \le t$). Equations 18.34 and 18.35 are used to estimate β and α, respectively. (Note that approach 2 is equivalent to approach 1 for s = 1.)

$$\frac{1}{e^{\beta} - 1} - \frac{t - s + 1}{e^{\beta(t-s+1)} - 1} = \sum_{k=0}^{t-s} k\frac{X_{k+s}}{X_{s,t}}, \tag{18.34}$$

$$\alpha = \frac{\beta X_{s,t}}{1 - e^{-\beta(t-s+1)}}. \tag{18.35}$$

Approach 3

Use cumulative failure counts in intervals 1 through s – 1 and individual failure counts in intervals s through t (i.e., $2 \le s \le t$). This approach is intermediate to approach 1, which uses all of the data, and approach 2, which discards "old" data. Equations 18.36 and 18.37 are used to estimate β and α, respectively. (Note that approach 3 is equivalent to approach 1 for s = 2.)

$$\frac{(s-1)X_{s-1}}{e^{\beta(s-1)}-1}+\frac{X_{s,t}}{e^{\beta}-1}-\frac{tX_t}{e^{\beta m}}=\sum_{k=0}^{t-s}(s+k-1)X_{s+k}, \tag{18.36}$$

$$\alpha=\frac{\beta X_t}{1-e^{-\beta t}}. \tag{18.37}$$

Limitations

- Model does not account for the possibility that failures in different intervals may be related
- Model does not account for repetition of failures
- Model does not account for the possibility that failures can increase over time as the result of software modifications

These limitations should be ameliorated by configuring the software into versions that, starting with the second version, the next version represents the previous version plus modifications introduced by the next version. Each version represents a different module for reliability prediction purposes. The model is used to predict reliability for each module. Then, the software system reliability is predicted by considering the N modules to be connected in series (i.e., worst-case situation), and computing the MTTF for N modules in series [SCH02].

Data Requirements

The only data requirements are the number of failures, f_i, i = 1, . . . , t, per testing interval. A reliability database should be created for several reasons: input data sets will be rerun, if necessary, to produce multiple predictions rather than relying on a single prediction; reliability predictions and assessments could be made for various projects; and predicted reliability could be compared with actual reliability for these projects. This database will allow the model user to perform several useful analyses: to see how well the model is performing; to compare reliability across projects to see whether there are development factors that contribute to reliability; and to see whether reliability is improving over time for a given project or across projects.

Applications

The major model applications are described below. These are separate but related uses of the model that, in total, comprise an integrated reliability program.

- ***Prediction.*** Predicting future reliability metrics such as remaining failures and time to next failure.
- ***Control.*** Comparing prediction results with predefined reliability goals and flagging software that fails to meet those goals.
- ***Assessment.*** Determining what action to take for software that fails to meet goals (e.g., intensify inspection, intensify testing, redesign software, and revise process). The formulation of test strategies is also part of assessment. Test strategy formulation involves the determination of: priority, duration and completion date of testing, allocation of personnel, and allocation of computer resources to testing.
- ***Risk Analysis.*** Compute RCMs for remaining failures and time to next failure.

Predict *test time* required to achieve a specified *number of remaining failures* at t_t, $r(t_t)$ in Equation 18.38:

$$t_t = [\log[\alpha/(\beta[r(t_t)])]]/\beta. \tag{18.38}$$

Implementation and Application Status

The model has been implemented in FORTRAN and C++ by the Naval Surface Warfare Center, Dahlgren, Virginia as part of the SMERFS. In addition, it has been implemented in CASRE. It can be run on an IBM PCs under all Windows operating systems.

Known applications of this model are:

- IBM, Houston, Texas: Reliability prediction and assessment of the on-board NASA Space Shuttle software
- Naval Surface Warfare Center, Dahlgren, Virginia: Research in reliability prediction and analysis of the TRIDENT I and II Fire Control Software
- Marine Corps Tactical Systems Support Activity, Camp Pendleton, California: Development of distributed system reliability models
- NASA JPL, Pasadena, California: Experiments with multimodel software reliability approach
- NASA Goddard Space Flight Center, Greenbelt, Maryland: Development of fault correction prediction models
- NASA Goddard Space Flight Center
- Hughes Aircraft Co., Fullerton, California: Integrated, multimodel approach to reliability prediction

SUMMARY

The purpose of this tutorial has been twofold: (1) to serve as a companion to the IEEE/AIAA Recommended Practice on Software Reliability and (2) to assist the engineer in understanding and applying the principles of hardware and software reliability, and the related subjects of maintainability and availability. Due to the prevalence of software-based systems, the focus has been on learning how to produce high reliability software. However, since hardware faults and failures can cause the highest quality software to fail to meet user expectations, considerable coverage of hardware reliability was provided. Practice problems with solutions were included to provide the reader with real-world applications of the principles that were discussed.

REFERENCES

[IEE07] IEEE/AIAA, "P1633™/Draft 13, Recommended practice on software reliability," November, 2000.

[KEL97] Ted Keller and Norman F. Schneidewind, "A successful application of software reliability engineering for the NASA space shuttle", Software Reliability Engineering Case Studies, International Symposium on Software Reliability Engineering, Albuquerque, New Mexico, November 3–4, 1997, pp. 71–82.

[LYU96] Michael R. Lyu, *Handbook of Software Reliability Engineering*. Los Alamitos, CA: IEEE Computer Society Press; New York: McGraw-Hill Book Company, 1996.

[MON96] Joseph G. Monks, *Operations Management*, 2nd ed., New York: McGraw-Hill, 1996.

[MUS87] John D. Musa, Anthony Iannino, and Kazuhira Okumoto, *Software Reliability: Measurement, Prediction, Application*, New York: McGraw-Hill, 1987.

[MUS99] John D. Musa, *Software Reliability Engineering: More Reliable Software, Faster and Cheaper*, 2nd ed. Bloomington, IN: Authorhouse, 1999.

[SCH01] Norman F. Schneidewind, "Reliability and maintainability of requirements changes," *Proceedings of the International Conference on Software Maintenance, Florence, Italy, 7–9 November*, 2001,pp 127–136.

[SCH02] Norman F. Schneidewind, "Body of Knowledge for Software Quality Measurement," IEEE Computer, Computer Society Press, Los Alamitos, CA, February 2002, pp. 77–83.

[SCH07] Norman F. Schneidewind, "Risk-driven software testing and reliability," *International Journal of Reliability, Quality and Safety Engineering*, 2007, 14(2), pp. 99–132. World Scientific Publishing Company.

[SCH97] Norman F. Schneidewind, "Reliability modeling for safety critical software," *IEEE Transactions on Reliability*, 1997, 46(1), pp. 88–98.

Practice Problems with Solutions 1

These practice problems are related to the following chapters:

Chapter 1: Digital Logic and Microprocessor Design

Chapter 2: Case Study in Computer Design

Chapter 6: Network Systems

Chapter 9: Programming Languages

Chapter 10: Operating Systems

Chapter 11: Software Reliability and Safety

In addition, there are circuit analysis problems that support Chapters 1 and 2.

CHAPTER 1 (DIGITAL LOGIC AND MICROPROCESSOR DESIGN) AND CHAPTER 2 (CASE STUDY IN COMPUTER DESIGN)

Number Representation in Floating-Point Format

Problem 1

Given: S is a sign bit where 0 indicates positive 1 and 0 indicates negative; exponent is a 7-bit excess 64 power of 2; mantissa is an 8-bit fraction.

Problem: A 16-bit word, N = 4000_{16}, represents what decimal numeric value?

Solution: 4000_{16} = 0100 0000 0000 0000

Sign 0 = positive, mantissa = 0000 0000, excess 64 power of 2 exponent = 1000000 – 1000000 = 0

Computer, Network, Software, and Hardware Engineering with Applications, First Edition. Norman F. Schneidewind.
© 2012 the Institute of Electrical and Electronics Engineers, Inc. Published 2012 by John Wiley & Sons, Inc.

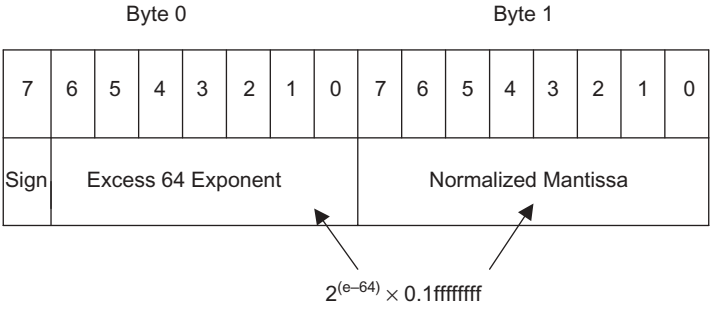


Figure 1 Floating point format.

Answer: $N = 0\ (2)^{0} = \mathbf{0.0}$ (Mantissa, Exponent)

Problem 2

Given: The floating point format design shown in Figure 1.

Problem: Can the decimal number 111.875 be stored in this format? If not, what decimal values can be stored?

Solution: 111.875 cannot be stored because the mantissa 0.875 requires more than 8 bits. However, 118.75 can be stored because this mantissa 0.75 requires only 7 bits, as shown in the following number conversions:

$$0.875_{10} = 0.3611_{16} = 0.110110\,1011_2\text{: 10 bits},\ 0.75_{10} = 0.411_{16} = 0.100\,1011_2\text{: 7 bits}$$
$$3\quad 6\quad 11 \qquad\qquad 4\quad 11$$

These conversions are achieved by successively dividing the decimal number by 16, recording the remainders, and assigning the remainders to the hexadecimal numbers in reverse order.

Encoding

One-Hot Encoding

One-hot encoding is a type of data encoding in which an individual flip-flop is dedicated to only one state of the data. Thus, only one flip-flop, which stores the data, can be active, or hot, at a time.

Problem 3

Given: A central processing unit (CPU) is designed with one-hot encoding. This CPU cycles through 16 states and produces 32 control signals (one for flip-flop output *high* and one for output *low*, for each of 16 states).

Problem: How many flip-flops are required?

Answer: Sixteen are required, corresponding to each of the 16 states.

Table 1 Hamming Code Words

D_6	D_5	D_4	D_3	D_2	D_1	D_0
I_3	I_2	I_1	P_2	I_0	P_1	P_0
1	1	1	1	0	0	0
1	1	0	0	1	1	0
1	0	1	0	1	0	1

Error Detection and Correction

The *Hamming Code* is a type of error detection and correction code that uses parity bits as a check for possible errors in the information bits. It is capable of detecting two bits in error and correcting one.

D_6	D_5	D_4	D_3	D_2	D_1	D_0	Data bit positions
I_3	I_2	I_1	P_2	I_0	P_1	P_0	Code word

Parity bits (P) are located in data bit positions P_0, P_1, and P_2.
Information bits (I_3, I_2, I_1, I_0) are located in the remaining bit positions.

Problem 4

Given: Hamming code words in Table 1. One of the code words has information bits 1110_2.

Problem: What is the correct Hamming Code word?

Solution: Only the first code word has the required information bits (in red). Therefore, the code word is 1111000_2.

Parity Error Detection

In even parity error detection, if the number of one bits, including the parity bit, is an even number, the data are assumed to be correct; otherwise, a one-bit error is assumed. In odd parity error detection, if the number of one bits, including the parity bit, is an odd number, the data are assumed to be correct; otherwise, a one-bit error is assumed. Thus, *parity error detection* can detect one-bit errors but cannot correct these errors. If a parity error is detected at the receiver, a negative acknowledgement is sent to the transmitter to retransmit the message.

Problem 5

Problem: It is desired to use even parity error detection in a digital circuit, where P is the parity bit. Which of the data below would generate even parity error signals?

P

0010 odd (error)

0100 odd (error)

0101 even (correct)

0111 odd (error)

Cyclic Redundancy Check (CRC)

This is a sophisticated error detection and correction process that uses mathematical polynomials for detecting and correcting multiple errors.

A message of degree n is the polynomial M(x) is: $x^n + x^{n-1} + x^{n-2} + \ldots + x^0$.

The sender and receiver must agree on a generator polynomial G(x) of degree $k \leq n$ in advance of transmission.

Both the high and low bits of G(x) must be 1.

M(x) must be longer than G(x).

k zeros are appended to M(x), yielding the transmitted message $T(x) = M(x)\, x^k$.

The remaining operations are shown in the following example:

Example:

$$M(x) = x^2 + x + 1 = 111$$

Use G(x) = x + 1 = 11 because M(x) can be divided by G(x) (i.e., the degree of G(x) = 1 ≤ degree of M(x) = 2).

k = degree 1, therefore append one zero to M(x), yielding the following transmitted message:

$$\mathbf{T(x) = M(x)x^k = x^3 + x^2 + x = 2^3 + 2^2 + 2^1 = 14_{10} = 1110_{16}.}$$

Divide T(x) by G(x), using modulo 2 division, and record remainder R(x), using modulo 2 division:

$$\begin{array}{r} \mathbf{100} \\ 11\overline{)\,110} \\ \underline{11} \\ 001 \\ \underline{00} \\ 010 \\ \underline{00} \\ \mathrm{R(x) = 10} \end{array}.$$

Now, subtract remainder R(x) = 10 from $M(x)x^k = 1110$, using Exclusive Or subtraction (modulo 2):

$$M(x)x^k \oplus R(x) = 1110 \oplus 10 = 1100.$$

At the receiver, divide $(M(x)x^k \oplus R(x))$ by G(x) and check for zero remainder. If this is the case, there is no error in transmission; otherwise, there are one or more errors, so retransmit:

$$
\begin{array}{r}
\mathbf{100} \\
11\overline{)1100} \\
11 \\
\hline
000 \\
00 \\
\hline
00
\end{array}
$$

Check: 12/3 = 4.

The remainder is zero, so there is no error in transmission.

Problem 6

Given: CRC polynomial: $x^{16} + x^{12} + x^5 + 1 = x^{16} + x^{12} + x^5 + x^0 = 2^{16} + 2^{12} + 2^5 + 1 = 65536 + 4096 + 32 + 1$

Problem: What is the hexadecimal equivalent of this CRC polynomial?

The solution is obtained by placing ones in the power of twos positions, corresponding to the decimal numbers as follows:

$$x^{16} + x^{12} + x^5 + 1 = 1\ 0001\ 0000\ 0010\ 0001 = 11021_{16}$$

65536 4096 32 1

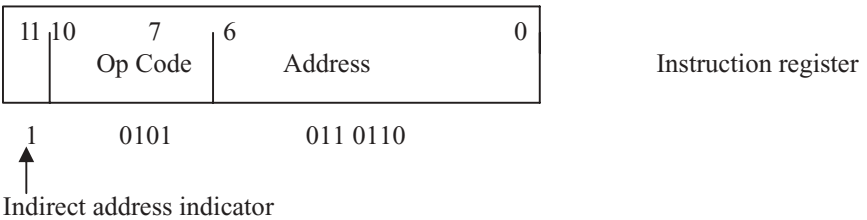
Proof: $x^{16} + x^{12} + x^5 + 1 = 69{,}665_{10}$. Convert 11021_{16} to base 10:

$$(1*16^4) + (1*16^3) + (2*16^1) + (1*16^0) = 65{,}536 + 4{,}096 + 32 + 1 = 69{,}665$$

Instruction Formats

Problem 7

Given: The instruction register below contains $AB6_{16}$ as the next instruction to be executed. Assuming this is an *add accumulator to operand at effective address* instruction, the instruction format is the following:



Identify the op code bits:

Bolded bits are op code bits: $AB6_{16}$ = 1**010 1**011 0110 (hexadecimal converted to binary)

Solution: Using indirect addressing as signified by bit 11 = 1: Op code bits = bits 10:7 = **0101**: add accumulator + operand at effective address 011 0110 (address located at 011 0110).

Problem 8

Given: The above instruction format in Problem 7 of 12 bits per word.

Problem: What is the maximum memory size in bits?

Answer: There are 2^7 addresses, each of which is 12 bits long. Therefore, memory size = 12×2^7 bits = $12 \times 128 = 1536$ bits.

Pipeline Systems

Number of clock cycles required in pipelined system = Number of clock cycles required in conventional system – Number of overlapped clock cycles in pipelined system = mn – (m – 1)(n – 1), where m is the number of instructions and n is the number of clock cycles required by the first instruction. This is the case because for the first instruction, there is no preceding instruction to overlap with. Therefore, there are m – 1 overlapped instructions. Also, the last clock cycle of the m – 1 overlapped instructions are not overlapped with a clock cycle of the preceding instructions, yielding (m – 1)(n – 1) overlapped clock cycles in a pipeline system.

Problem 9

There are four instructions in a pipelined system. How many clock cycles are required to execute the four instructions?

Solution: The first instruction requires n = 4 clock cycles and each of the remaining (m – 1) instructions require only 1 clock cycle because these (m – 1) instructions are overlapped with the preceding instructions, yielding n + (m – 1) = 4 + 3 clock cycles.

Problem 10

What is the increase in speed of the pipelined system in Problem 9 versus a non-pipelined system?

Solution: The speed ratio = Number of clock cycles required in conventional system/Number of clock cycles required in a pipelined system = mn/(m + n – 1) = 16/7 = 2.286. If m is large, the increase in speed approaches the *maximum speed* of *n* clock cycles per instruction.

Problem 11

Pipeline *throughput* is defined as the *number of instructions*, m, per *total clock cycle time* required to process m instructions = (m instructions)/((number of clock cycles per instruction*time per clock cycle)) = (m)/(m + n – 1)T, where T is clock cycle time per instruction. For a typical microprocessor with a clock speed of 10 MHz (10^7 cycles per second), T = $1/10^7$ seconds.

What is the throughput for a four-instruction pipeline?

Solution: m/((m + n – 1)T) = 4/((7)($1/10^7$))) = (4)(10^7)/7 = 5.71 million instructions per second.

Problem 12

Pipeline efficiency is computed as: actual speed increase/maximum speed increase.

What is the efficiency for the pipeline in Problem 10?

Solution: (mn/(mn/m + n – 1))/n = 2.286/4 = 0.5715.

Problem 13

What governs the clock cycle frequency of a pipeline system?

Solution: The pipeline with the slowest processing time.

Problem 14

What is needed to maintain performance in a pipeline system, when needed resources such as hardware are not available?

Solution: More resources can be employed, if available, or the pipeline can be stalled (e.g., no instructions executed until needed hardware is available).

Problem 15

How many instructions can a nonpipeline computer execute at a time?

Solution: Only one instruction at a time.

Shifting and Comparators

Problem 16

What is the process and purpose of the arithmetic right shift?

Solution: Arithmetic right shifting is performed to divide a quantity by 2^n and round down ("0" inserted in least significant bit [LSB] position), where n is the number of bits shifted. The sign bit is preserved in the right shift.

Problem 17

What is the process and purpose of the arithmetic left shift?

Solution: Multiply a quantity by 2^n. The sign bit is lost because the high order data bit is shifted into the sign position.

Problem 18

What is the process and purpose of the right rotate logical shift and the left rotate logical shift?

Solution: The right and left rotate logical shifts can be used, for example, to identify whether database application A or B should be executed dependent on a series of bits in a register (0 for application A and 1 for application B). For each shift of one bit, A or B would be selected. See Figure 2 for details of the four shifting operations, where for each operation, one bit is shifted.

Problem 19

How can a circuit be designed to analyze branch instruction logic in a program?

Solution: Figure 3 shows how a comparator and its accompanying shift control logic can be designed to determine whether data originally in registers A and B, and then transferred to busses A and B, have the relationships A = B,

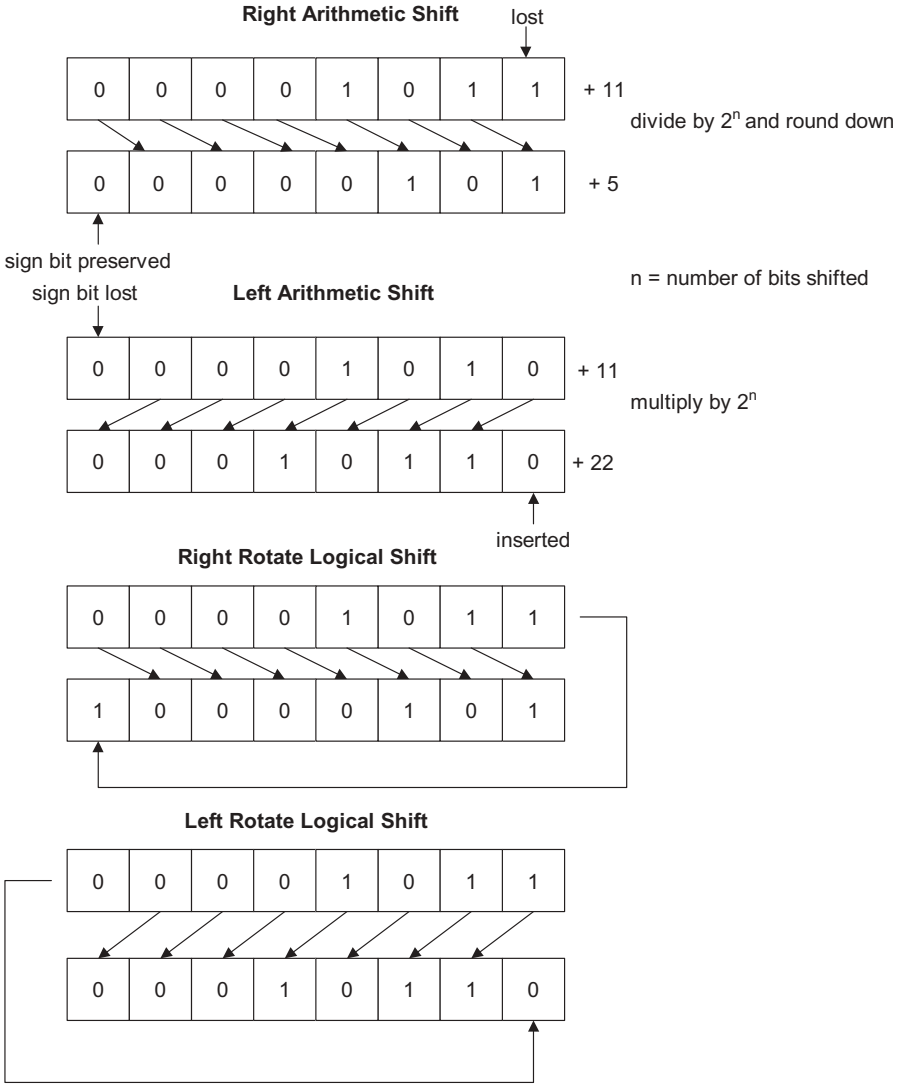


Figure 2 Shifting operations.

A > B, or A < B. Thus, in a program, branch 1 would be taken if A = B; branch 2 would be taken if A > B; and branch 3 would be taken if A < B.

Problem 20

Figure 4 shows that the hexadecimal quantity 5A has experienced a rotated left shift of 3 bit positions. What is the resultant quantity in hexadecimal and decimal?

Solution: As the conversion operation in Figure 4 shows, the result is D2 in hexadecimal and 210 decimal.

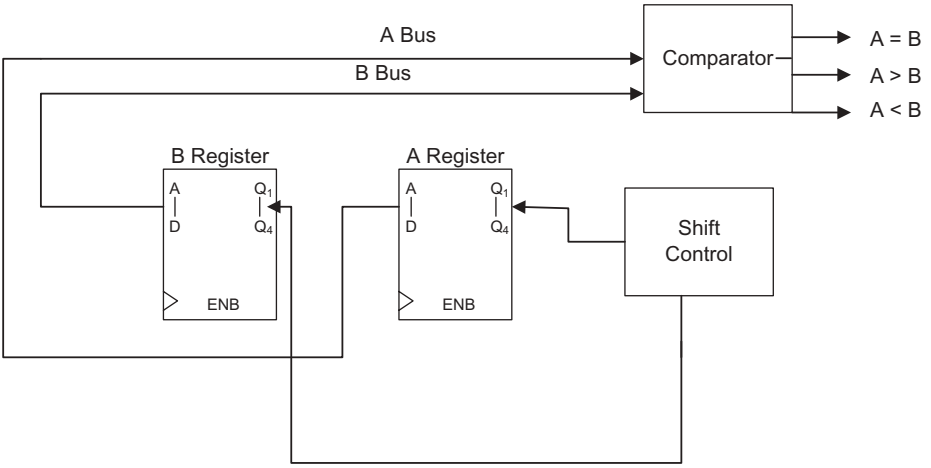


Figure 3 Comparator circuit.

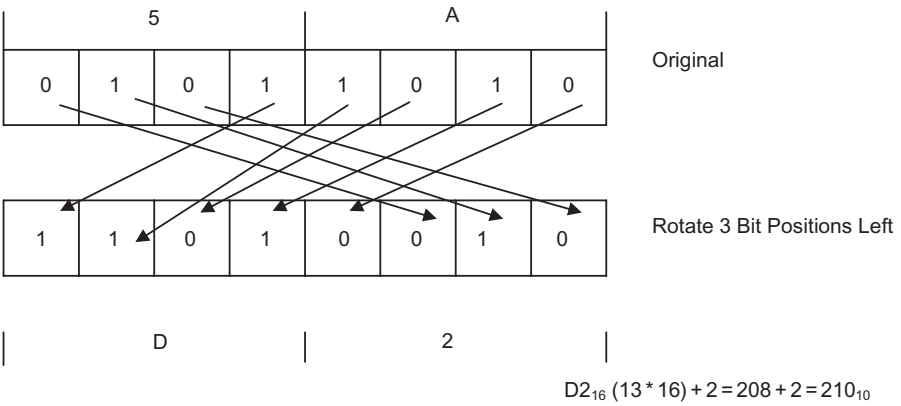


Figure 4 Rotate left shift operation.

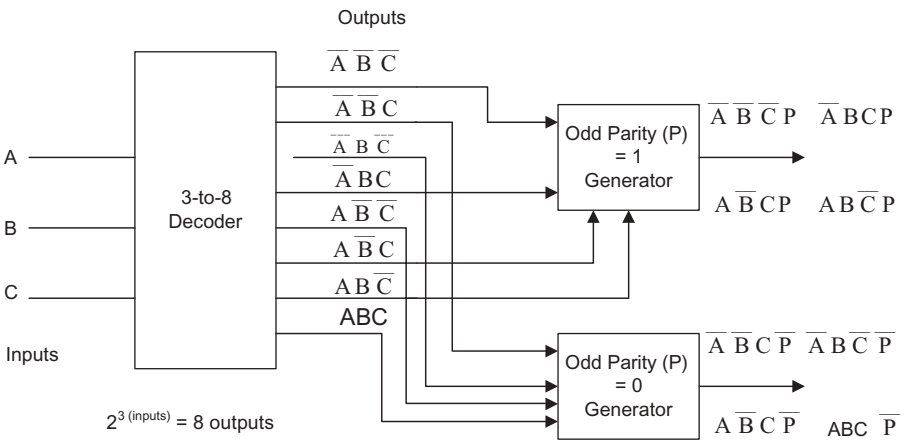


Figure 5 Decoder circuit.

Table 2 Decoder Circuit Parity Generation

A	B	C	P
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	0

Decoders

A decoder is a combinational circuit that selects one of n inputs and produces 2^n outputs, where n is the number of input bits, as shown in Figure 5.

Problem 21

Given: Decoder circuit in Figure 5, showing selected n inputs and 2^n outputs, with an odd parity generator that signifies no error, if the number of output bits, including the parity bit, is odd.

Problem: How many outputs are required to address the inputs A, B, C in the decoder circuit?

Solution: Figure 5 shows that when there are three inputs to the decoder, eight outputs are required. In addition, Table 2 shows the parity that is required to ensure no error for the various input combinations.

Flip-Flop Circuits

J-K flip-flop next state output Q(t + 1) is a function of inputs J and K, and present state flip-flop outputs Q(t) and $\overline{Q}(t)$:

$$Q(t+1) = J\overline{Q}(t) + \overline{K}Q(t)$$

Problem 22

Given: Figure 6 shows a J-K flip-flop circuit that includes six gates. The current flip-flop state $Q_1Q_2Q_3 = 100$.

Problem:

(a) Identify the types of gates.

Solution: EXCLUSIVE OR: Gates 1 and 3; NAND: Gates 2, 4, and 6; EXCLUSIVE OR: Gate 5.

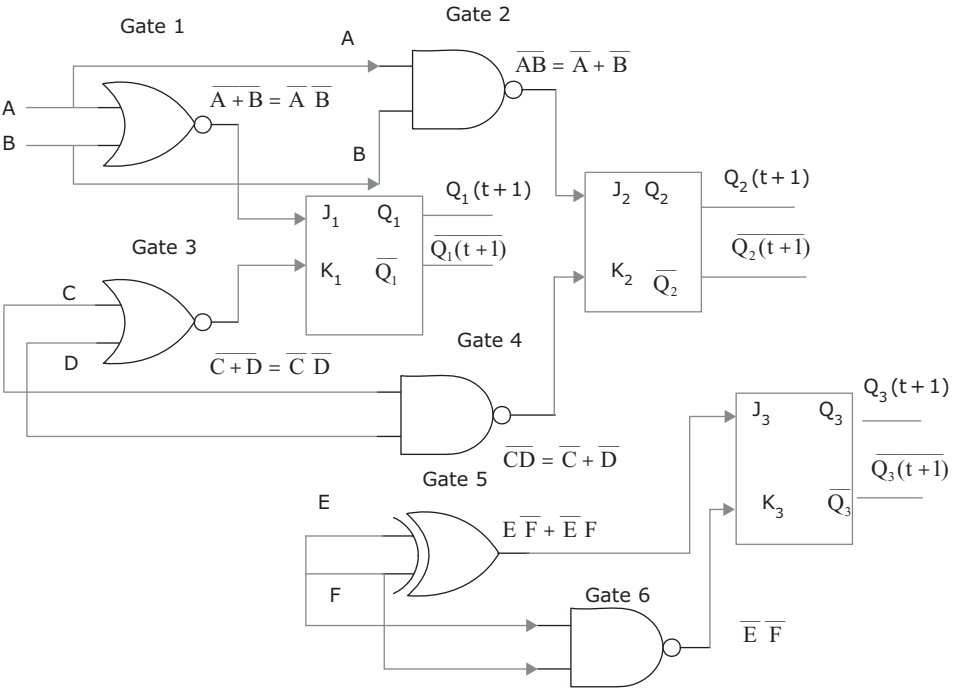


Figure 6 J-K flip-flop circuit.

Problem:

(b) Determine the output of each gate.

Solution: See Figure 6.

Problem:

(c) Determine the next state of the flip-flop outputs.

Solution:

$$Q_1(t+1) = J_1\overline{Q_1}(t) + \overline{K_1}Q(t) = (\overline{A}\overline{B})(0) + (C+D)(1) = C+D$$
$$Q_2(t+1) = J_2\overline{Q_2}(t) + \overline{K_2}Q_2(t) = (\overline{A}+\overline{B})(1) + (CD)(0) = \overline{A}+\overline{B}$$
$$Q_3(t+1) = J_3\overline{Q_3}(t) + \overline{K_3}Q_3(t) = (E\overline{F}+\overline{E}F)(1) + (E+F)(0) = E\overline{F}+\overline{E}F$$

Multiplexers

The multiplexer circuit in Figure 7 produces a single output Y for four inputs, x_0, x_1, x_2, and x_3, depending on the values of the selector bits, s_0, s_1, using an OR output function.

Problem 23

Develop the output functions for the multiplexer circuit in Figure 7.

Solution: Figure 7 shows the output function.

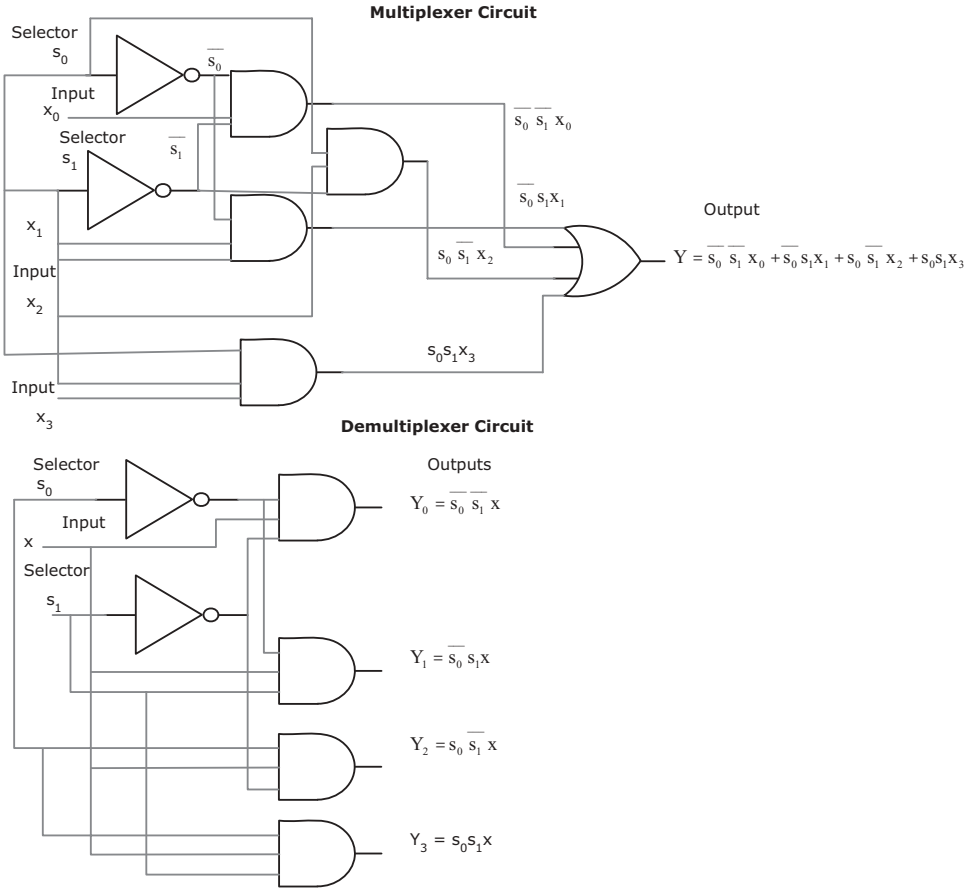


Figure 7 Multiplexer and demultiplexer circuits.

Demultiplexers

A demultiplexer causes an input x to be transferred to one of 2^n output lines, where n is the number of select inputs in Figure 7.

Problem 24

Develop the output functions for the demultiplexer circuit in Figure 7.

Solution: Figure 7 shows the output functions.

Timing Relationships

Problem 25

Given: Timing relationships pertaining to D flip-flops in Figure 8.

Problem: What is the timing diagram for input signals A and M and what is the value of the output?

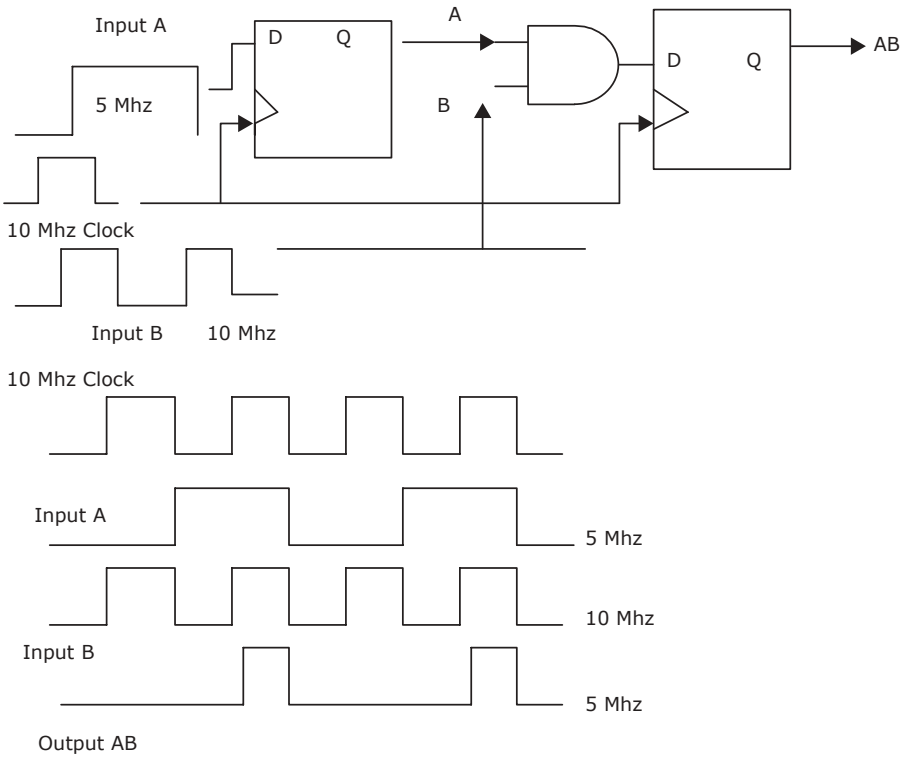


Figure 8 Timing relationships.

Solution: Figure 8 shows that by virtue of ANDing inputs A and B, the output = AB. The timing diagram shows that output AB has a frequency of 5 MHz governed by the durations of the 10 MHz clock when both inputs A and B are positive.

Programmable Logic Array (PLA)

A PLA consists of programmable AND and OR gates. It can be programmed to implement a Boolean sum of product terms.

Problem 26

Draw a logic diagram showing how the PLA should be designed to implement the functions $Z_2 = \overline{A}\overline{B} + BC, Z_1 = \overline{A}\overline{B} + AC$, and $Z_0 = \overline{A}\overline{C} + BC$.

Solution: The design is shown in Figure 9.

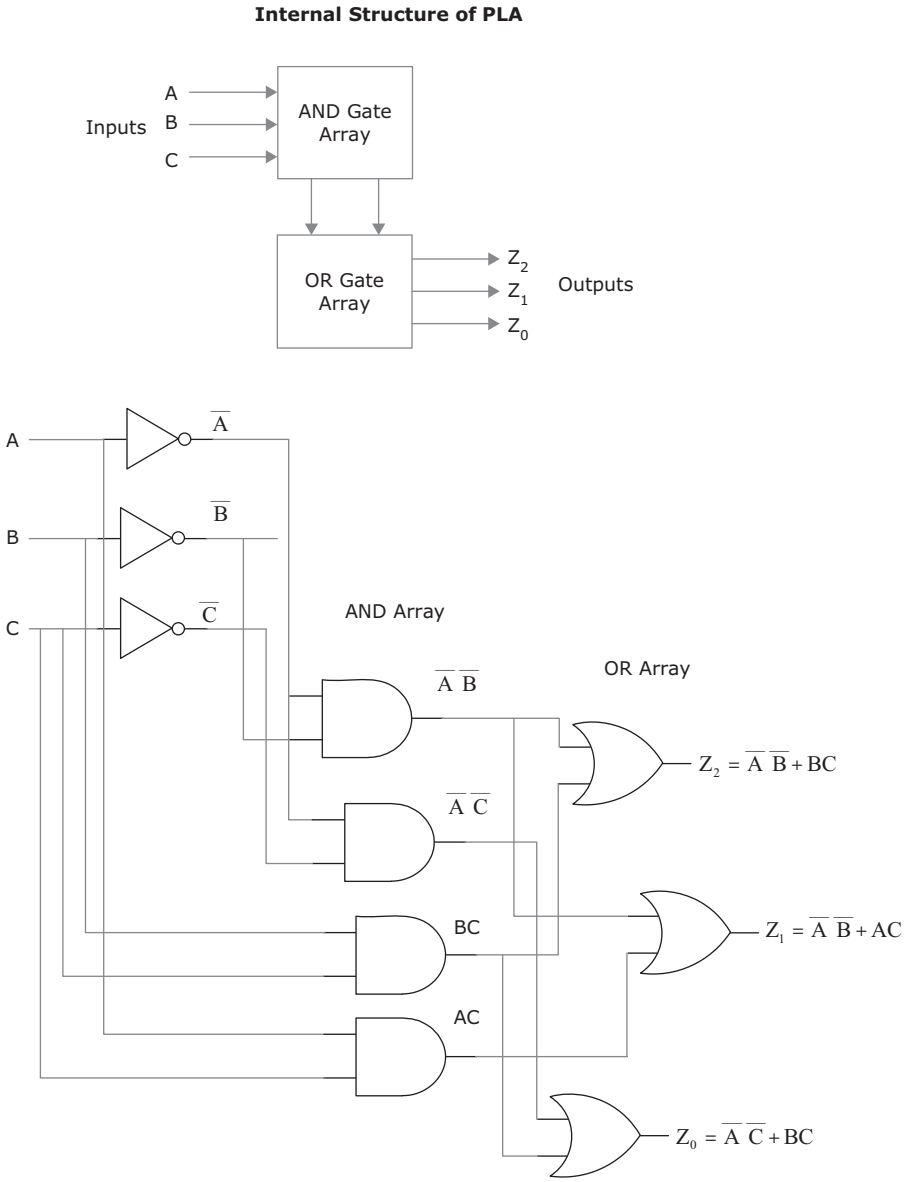


Figure 9 Programmable logic array.

State Machine Diagrams

As Figure 10 shows, state transition diagrams are useful for tracking state transitions in a digital system. For example, in Figure 10, state transitions are triggered by switch openings and closings, which cause transitions from present states to next states.

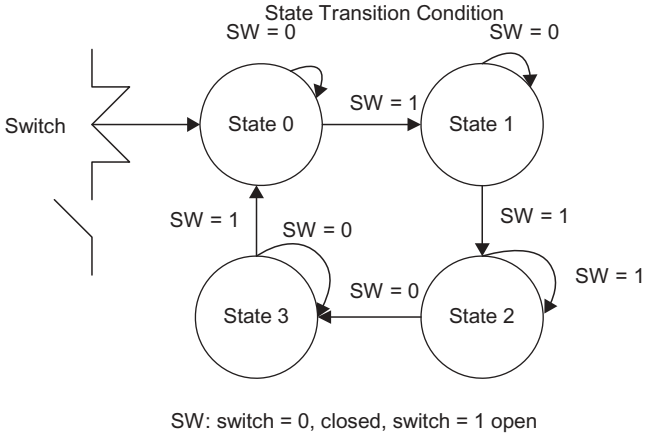


Figure 10 Switch state diagram. SW, switch. SW = 0, closed; SW = 1, open.

Table 3 Switch State Transition Table

Present switch condition	Present state	State transition condition	Next state
SW = 0	0	SW = 0	0
SW = 0	0	SW = 1	1
SW = 1	1	SW = 0	1
SW = 0	1	SW = 1	2
SW = 1	2	SW = 1	2
SW = 1	2	SW = 0	3
SW = 0	3	SW = 0	3
SW = 0	3	SW = 1	0

Problem 27

Develop the state transition table corresponding to Figure 10.

Solution: See Table 3.

Edge-Triggered D Flip-Flop Circuit

The output state Q(t + 1) of a D flip-flop is governed by the falling edge of the clock pulse in Figure 11: Q(t + 1) = D on the falling edge.

Problem 28

Given the characteristic of the D flip-flop, draw the timing diagram for the circuit in Figure 11.

Solution: See Figure 11.

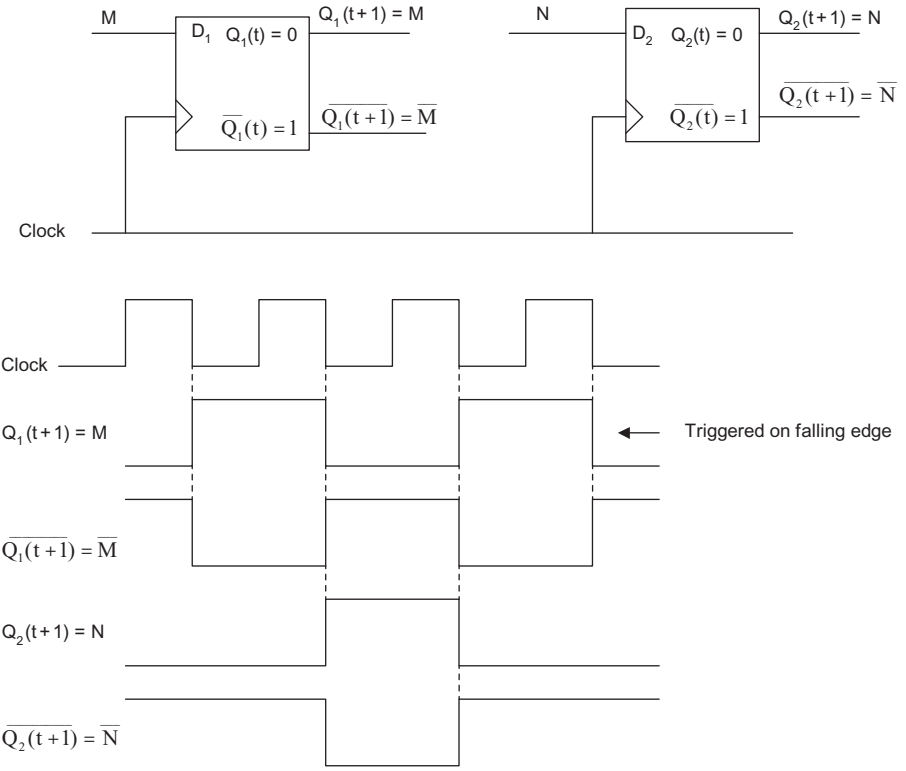


Figure 11 Edge-triggered flip-flop circuit.

State Machine Design

The design of state machines involves identifying states, state transitions, and the sequences of state transitions, where a sequence is the order in which transitions occur.

Problem 29

For the elevator example in Figure 12, identify the correct sequence of state transitions.

Solution: As seen in Figure 12 and based on the logic of elevator operations, the correct sequence is: (current floor (Nc) → request floor (Nr) → destination floor (Nd)) *or* (request floor (Nr) → destination floor (Nd)).

Problem 30

In Figure 12, what are the next states for current state 3 and what are the corresponding state transitions?

Solution: Referring to Figure 12, the next state 3 is caused by elevator staying at Nd1; the next state 4 is caused by elevator going to Nr2.

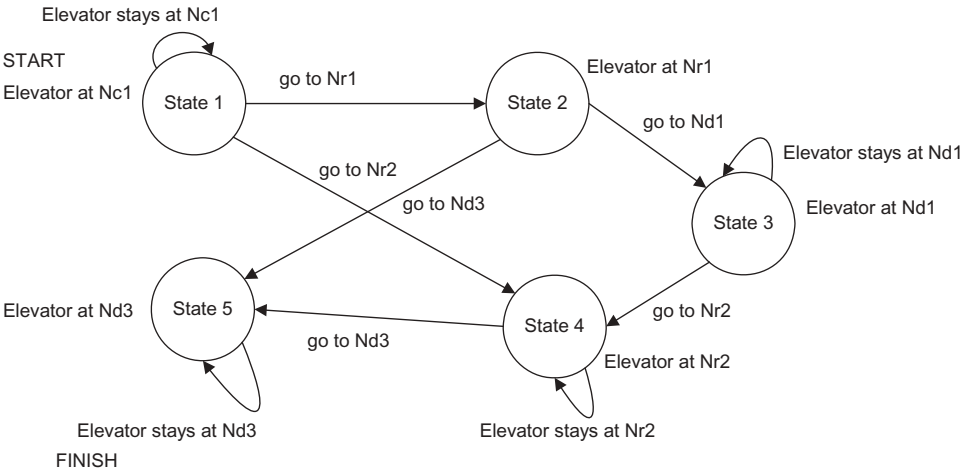


Figure 12 State machine example. Nc, current floor; Nr, request floor; Nd, destination floor. Numbers designate specific floor.

Problem 31

In Figure 12, give an example of an illegal state transition.

Solution: State 2 → State 4 is illegal because it would be illogical to transition from one request floor to another request floor without first going to a destination floor.

CHAPTER 9 (PROGRAMMING LANGUAGES) AND CHAPTER 10 (OPERATING SYSTEMS)

Queue Data Structure

One approach for managing data and instructions is the queue that uses a first-in, first-out discipline. An application is the processing by a microprocessor, on a non-priority basis, of a stream of inputs.

Problem 32

For an input of KJIHGFEDCBA, what is the output order for a first-in, first-out discipline?

Solution: Figure 13 shows the output ordering of the input data.

Stack Data Structure

Another one of the important software design approaches is to use the push-down stack, which is particularly valuable when interrupts occur and it is necessary to service the interrupts and later return to the main program. The stack facilitates this process by pushing the contents of the program counter and special registers onto a

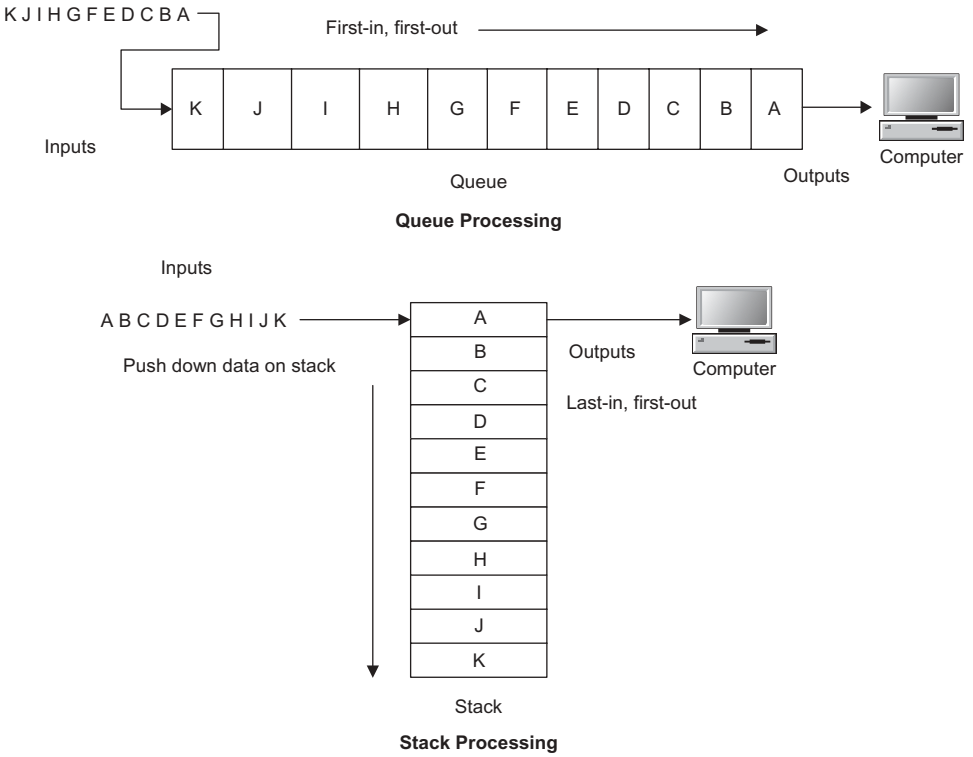


Figure 13 Queue and stack processing.

memory area called the stack. The stack operates on a last-in, first-out basis, meaning that the register contents that were last pushed are on the top of the stack, for example, the program counter, so that the program is pointing to the address of the next instruction to be executed.

Problem 33

For an input of ABCDEFGHIJK, what is the output order for a last-in, first-out discipline?

Solution: Figure 13 shows the output ordering of the input data.

Instructions Designed to Perform Functions

Instruction sequences can be designed to perform certain functions such as generating a square wave in a register, using a set of binary bits to represent the output (i.e., square wave), or generating a Fibonacci number sequence.

Problem 34

Show how a square wave can be generated, using an accumulator, input register A, and output register B.

Solution: See Figure 14 for the result.

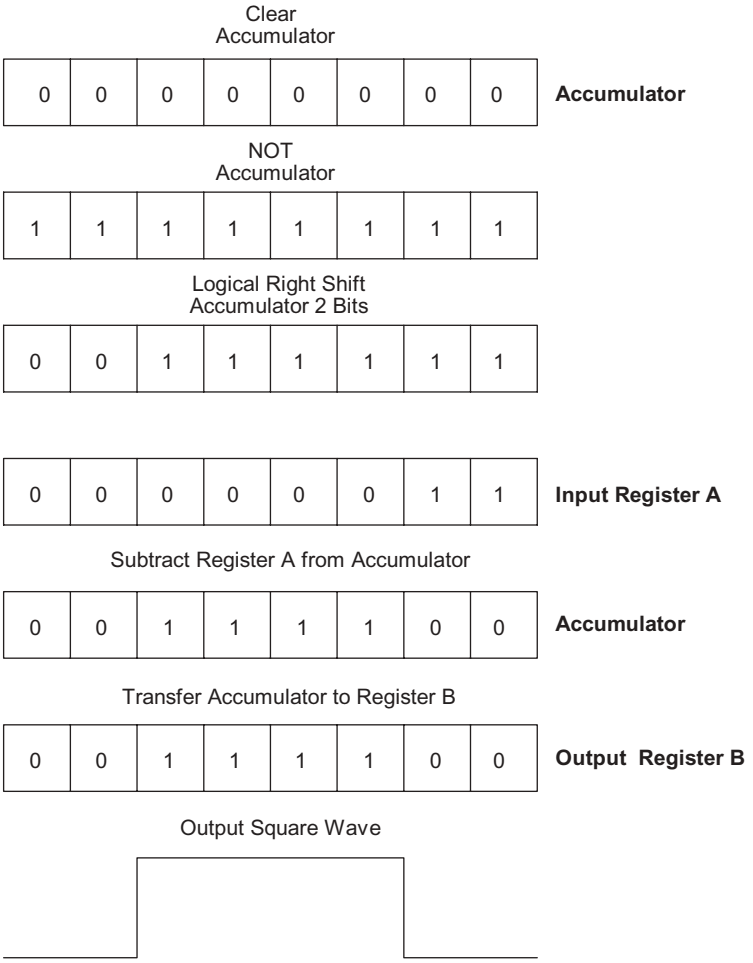


Figure 14 Square wave generation.

Problem 35

Produce a program to compute Fibonacci numbers, where a Fibonacci number is defined as the sum of the previous two numbers, for example, 1, 1, 2, 3, 5, 8, and 13.

Solution: Table 4 shows the program.

Software Specifications

Problem 36

What must software specifications contain to be complete?

Solution: There must be details of expected inputs and outputs, details of processing requirements, and details of design requirements.

Table 4 Fibonacci Number Program

Command	Description
i = 0	Set first Fibonacci number index
n(i), n(i + 1) = 1	Set first two Fibonacci numbers = 1
While (i < N) do Functions F and C	Keep inputting and computing while i < N Fibonacci numbers
Function F (input n(i))	Input Fibonacci number = n(i)
Function C (n(i) + n(i + 1))	Compute Fibonacci number
Function C (n(i) + n(i + 1)) → output	Output Fibonacci number
i = i + 1	Increment Fibonacci number index
Return (While)	Continue with Functions F and C

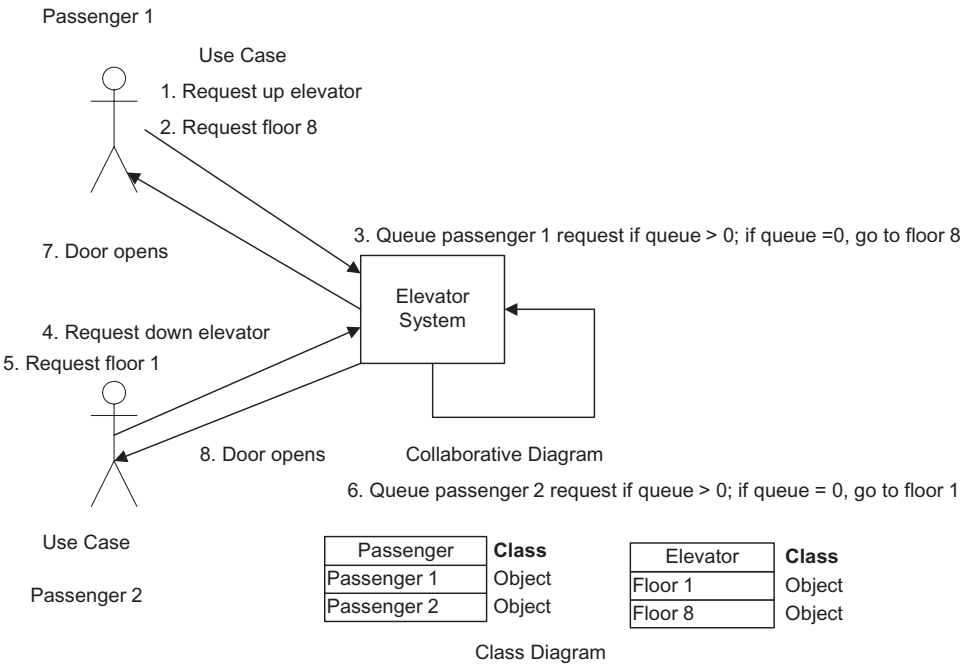


Figure 15 Object-oriented conceptual model.

Problem 37

The object-oriented design process involves specifying a conceptual model, which consists of actors, uses cases, classes, objects comprising a class, and operations. With this definition in mind, draw the design process for an elevator example.

Solution: The object-oriented design process is shown in Figure 15.

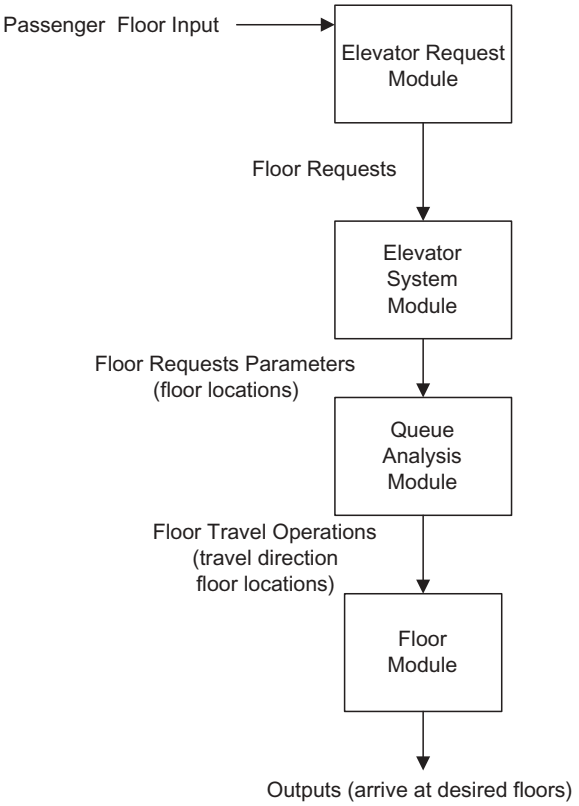


Figure 16 Structured design diagram.

Structure Chart

Problem 38

For a structure chart, we need to know: (1) all the modules in the chart, (2) all the data items in the chart, (3) the organization of the modules, and (4) all the communications among modules.

Using this concept, design a structure chart for the elevator system depicted in Figure 15.

Solution: The structure chart design is shown in Figure 16.

CHAPTER 11: SOFTWARE RELIABILITY AND SAFETY

Quality Assurance

Integration Testing

Integration testing can be accomplished either in a top-down or bottom-up fashion. In bottom-up testing, drivers are required for modules at a higher level to call

modules at a lower level, and test cases are required for doing integrated testing between modules at higher and lower levels. A disadvantage of bottom-up testing is that the most important modules—those at the top level—are not tested first. On the other hand, top-down testing is more complex. For this type of testing, stubs are required that simulate the lower level module data and instructions, which have not yet been tested, for the benefit of higher level modules.

Problem 39

Design top-down and bottom-up testing schemes for modules A, B, C, D, and E.

Solution: The testing schemes are shown in Figure 17. These types of tests assist in the verification process that is aimed at assuring that the testing schemes are faithful to the software design. In addition, these tests are augmented by inspections, which are a peer review process for examining code to help assure its correctness.

Array and Matrices

Arrays are one-dimensional data structures with values and indices that point to the values, such as the adjacency list in Figure 18. Matrices are two-dimensional data structures that have two sets of indices, one to point to values in one of the dimensions, and another set to point to values in the second dimension. An adjacency list can be used for this purpose.

Some problems involving arrays and matrices are concerned with efficient processing when the matrix is sparse (i.e., many zero elements in matrix). In this case, an adjacency list (pointers to nonzero elements) combined with a linked list can be used. For an $n \times n$ matrix with zero elements, the number of nonzero elements that would make the link list superior to a matrix can be computed.

Problem 40

Portray the two data structure alternatives, described above, and compute the storage requirement for the two alternatives.

Solution: Figure 18 shows the logic of the alternatives and the computation of the competing storage requirements.

Problem 41

Using the result obtained in Problem 39, compute the number of zero entries Z and the number of nonzero entries N.

Solution:

Matrix storage requirement (matrix plus pointers): $32n^2 + 32n$
Link list storage requirement (linked list plus pointers): $32n + 64(n^2 - Z)$

$$32n^2 + 32n = 32n + 64(n^2 - Z), 64Z = 32n^2,$$

zero entries $Z = 0.5n^2$, $N = n^2 - 0.5\ n^2 = 0.5\ n^2$

In other words, for this problem, the number of nonzero entries equals the number of zero entries. If Z were larger, the linked list would require less storage than the matrix.

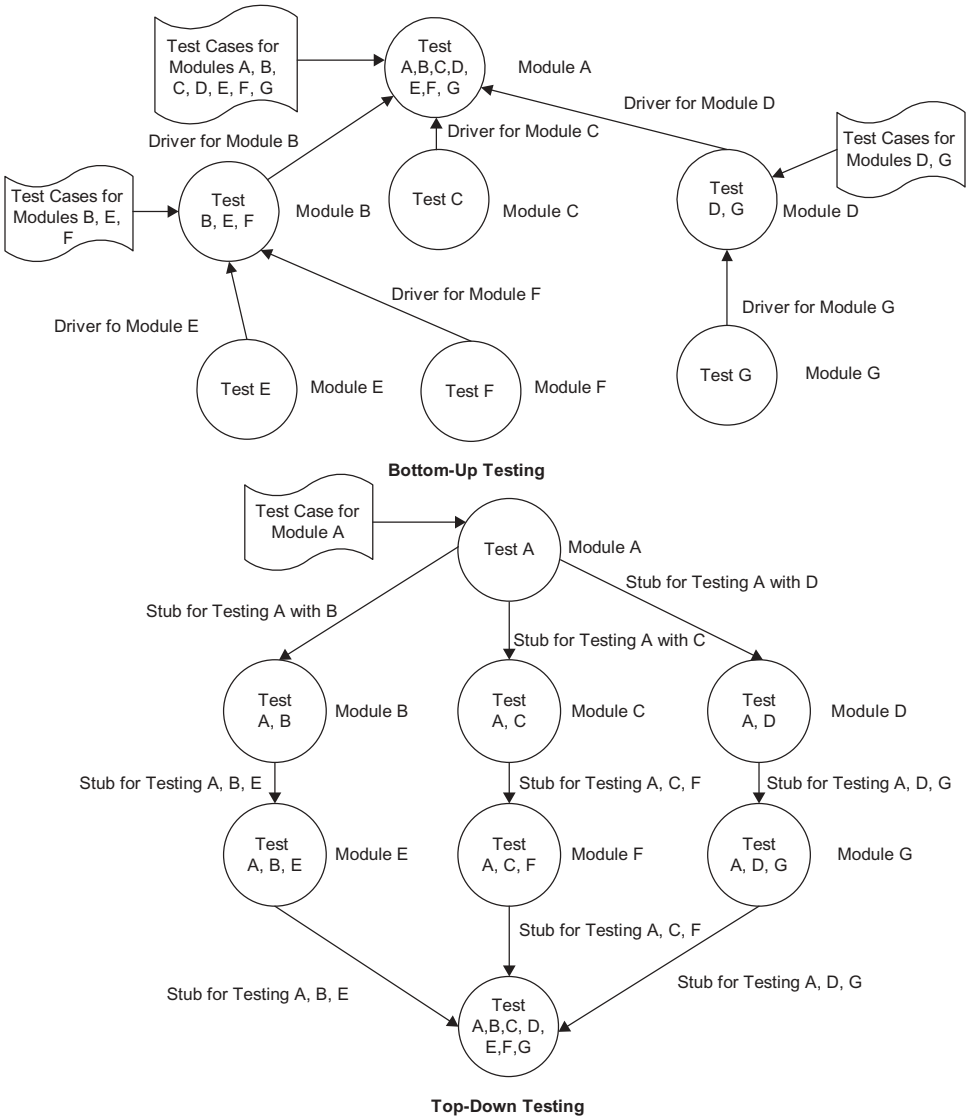


Figure 17 Integrated testing designs.

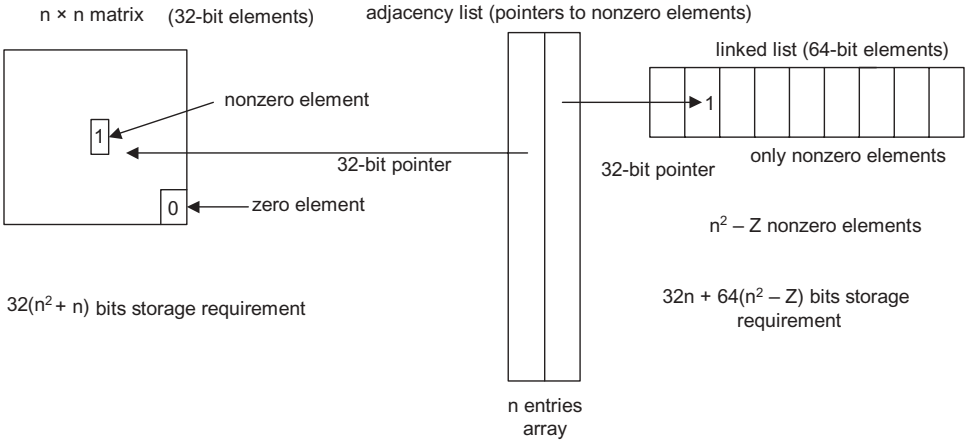


Figure 18 Array and matrix data structure design. Z, number of zero elements.

Nested Program Logic

The number of paths (i.e. number of program executions or number of calls) in a *nested program* can be determined by drawing a flow chart, as shown in Figure 19.

Problem 42

Based on Figure 19, what is the formula for computing the number of paths?

Solution: See Figure 19.

CHAPTER 6: NETWORK SYSTEMS

Network Diameter

The network diameter is the maximum of the distances between all possible pairs of nodes (e.g., computers) of a graph of a network, *without backtracking* (i.e., revisiting a node on a path). This concept is used for identifying the maximum distance data would have to travel in a network. The appropriate bandwidth would be programmed to meet this requirement.

Problem 43

Given: Figure 20 showing network connections.

Determine the network diameter.

Solution: The various paths, links, and maximum number of links are shown in Table 5, based on the network topology in Figure 20, yielding diameter = 4 links (A → E → B → C → D).

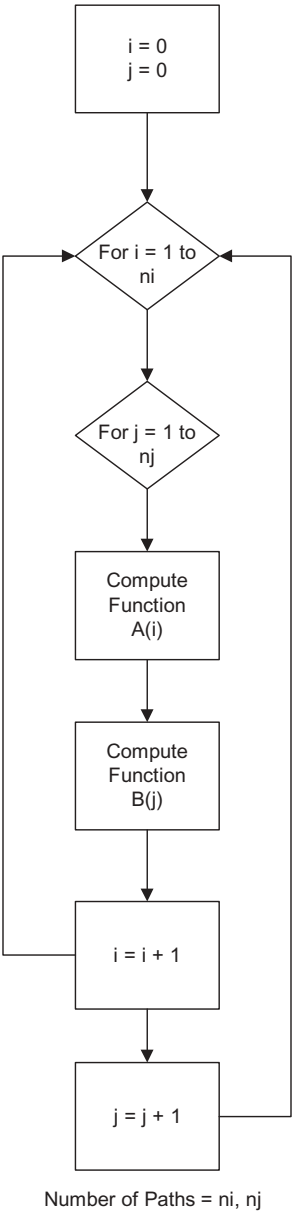


Figure 19 Flow chart problem.

Paths:
A, B
A, E, B
A, B, C
A, E, B, C
A, E, D, C
A, B, C, D
A, E, B, C , D
A, E, D
A, E
A, B, C, D, E
A, B, E
B, C
B, E
B, C, D, E
B, E, D
B, C, D
C, D
C, D, E
C, D, E, B
D, E
D, E, B
D, E, B, C
E, B
E, B, C
E, B, C, D
E, D

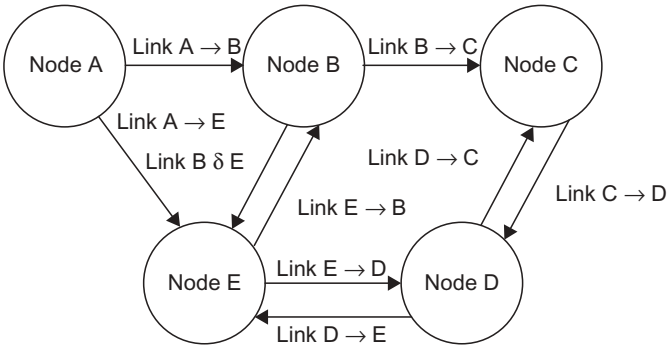


Figure 20 Network connectivity.

Problem 44

What series of instruction functions will generate a square wave with a 50% duty cycle to control the output of data on a network? Show the instructions and their functions and the results of the instruction functions executions.

Solution: Table 6 shows the instruction functions and Table 7 shows the results of the instruction functions executions.

Analog-to-Digital Conversion to Support Network Operations

Problem 45

Given: Analog-to-digital converter network support circuit diagram in Figure 21.

What address is used to access network Channel B of the converter?

Solution: The circuit diagram shows how the address bits (01000000) are configured to support the access of network Channel B at address 40_{16}.

Table 5 Network Connectivity

Path	Links	Number of links
A → B	A → B	1
A → E → B	A → E, E → B	2
A → B → C	A → B, B → C	2
A → E → B → C	A → E, E → B, B → C	3
A → E → D → C	A → E, E → D, D → C	3
A → B → C → D	A → B, B → C, C → D	3
A → E → B → C → D	A → E, E → B, B → C, C → D	**4**
A → E → D	A → E, E → D	2
A → E	A → E	1
A → B → C → D → E	A → B, B → C, C → D, D → E	4
A → B → E	A → B, B → E	2
B → C	B → C	1
B → E	B → E	1
B → C → D → E	B → C, C → D, D → E	3
B → E → D	B → E, E → D	2
B → C → D	B → C, C → D	2
C → D	C → D	1
C → D → E	C → D, D → E	2
C → D → E → B	C → D, D → E, E → B	3
D → E	D → E	1
D → E → B	D → E, E → B	2
D → E → B → C	D → E, E → B, B → C	3
E → B	E → B	1
E → B → C	E → B, B → C	2
E → B → C → D	E → B, B → C, C → D	3
E → D	E → D	1

Hubs versus Switches

Problem 46

Figure 22 demonstrates that compared with hubs, switches increase bandwidth (no collisions) and security (memory of switches provides security checking). Why is this the case?

Solution: Because collisions are possible using hubs but not possible using switches.

Connection Hijacking

Transmission Control Protocol (TCP) session hijacking occurs when a hacker takes over a TCP session *during* a session between two computers. Since most

Table 6 Instruction Functions

Instruction	Function
CLR (clear accumulator)	$0 \rightarrow$ ACC
IN B (transfer "1" from input device to register B)	$1 \rightarrow$ B
ADD (add register B = 1 to accumulator)	ACC = ACC + B
SHR (shift accumulator logical right 4 bits)	ACC [7:4] $\rightarrow$ ACC [3:0], $0 \rightarrow$ ACC [7:4]
SHL (shift accumulator logical lift 2 bits)	ACC [7:2] $\leftarrow$ ACC [5:0], ACC [1:0] $\leftarrow 0$
OUT (out put accumulator to register C)	ACC $\rightarrow$ C

Table 7 Square Wave Instruction Function Execution Results

Instruction	Instruction result	Square wave result
CLR (clear accumulator)	$0 \rightarrow$ ACC	
IN B (transfer "1" from input device to register B)	$1 \rightarrow$ B	
ADD (add register B = 1 to accumulator)	ACC = ACC + B	
SHR (shift accumulator logical right 4 bits)	ACC [7:0] $\rightarrow$ ACC [3:0], $0 \rightarrow$ ACC [7:4]	
SHL (shift accumulator logical lift 2 bits)	ACC [7:2] $\leftarrow$ ACC [5:0], ACC [1:0] $\leftarrow 0$	
OUT (out put accumulator to register C)	ACC $\rightarrow$ C	

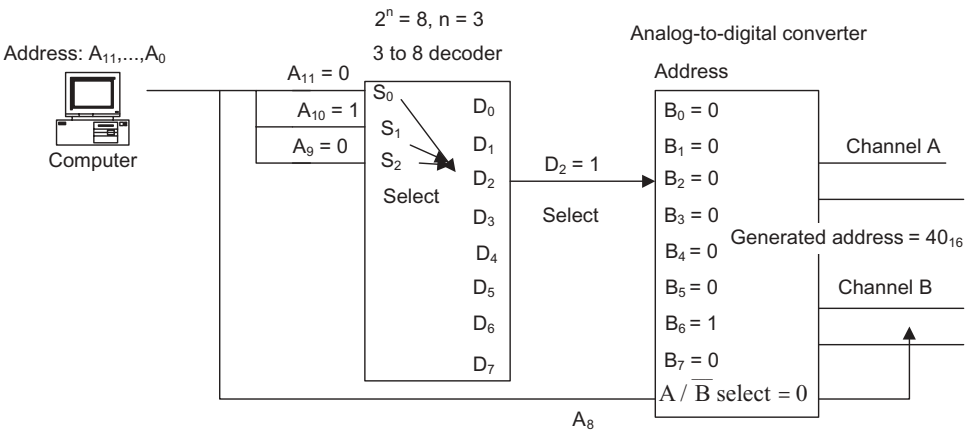


Figure 21 Analog-to-digital converter circuit.

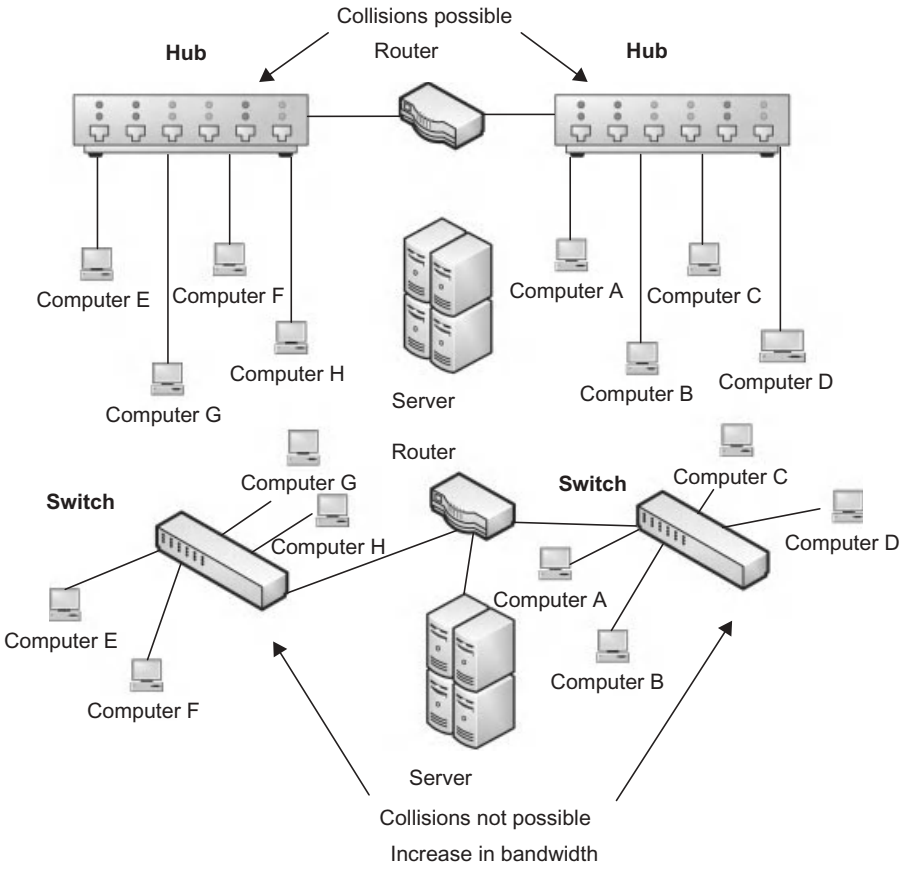


Figure 22 Hubs versus switches.

authentications only occur at the start of a TCP session, this allows the hacker to gain access to a computer. A popular hacking method uses source-routed Internet Protocol (IP) packets. Source-routed packets identify the source address in the packet. A hacker at node A listens for packets originating at nodes B or C. If such traffic passes through node A, it allows the hacker to participate in a conversation between B and C. This is known as a "man-in-the-middle attack." A common component of such an attack is to execute a *denial-of-service* (DoS) attack against nodes B and C.

Problem 47

Draw an Internet diagram depicting the connectivity that would permit the security breaches described above.

Solution: Figure 23 shows the Internet vulnerabilities that would allow security problems to occur.

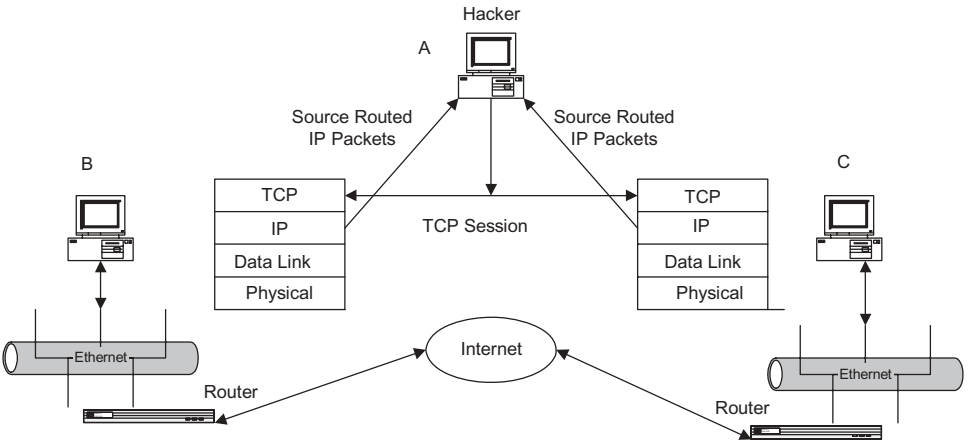


Figure 23 Internet connections.

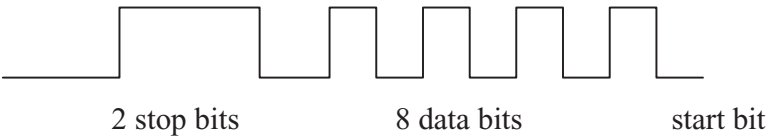
RS 232 C Data Transmission

Coding efficiency is defined as the (number of data bits)/total number of bits (data plus control).

Problem 48

Referencing the figure below, what is the coding efficiency?

Solution: As can be seen in the figure, there are 8 data bits out of a total of 11 bits. Therefore, coding efficiency = 8/11 = 73%.



Topology and Hardware

Various combinations of network topology and hardware are shown in Figure 24. The main characteristic of *10 Base T Ethernet* is that each computer is individually connected to a hub. The hub has no storage; therefore, data cannot be buffered. This scheme does have an advantage over the bus because, with each computer having its own connection to the hub, collisions of data on the network are avoided. Because the bus is shared by many devices, collisions on the connected Ethernets are unavoidable. However, the bus is the traditional way to connect Ethernets because this type of connection is highly standardized and economical. A later version of Ethernet uses switches to direct traffic. This configuration avoids collisions.

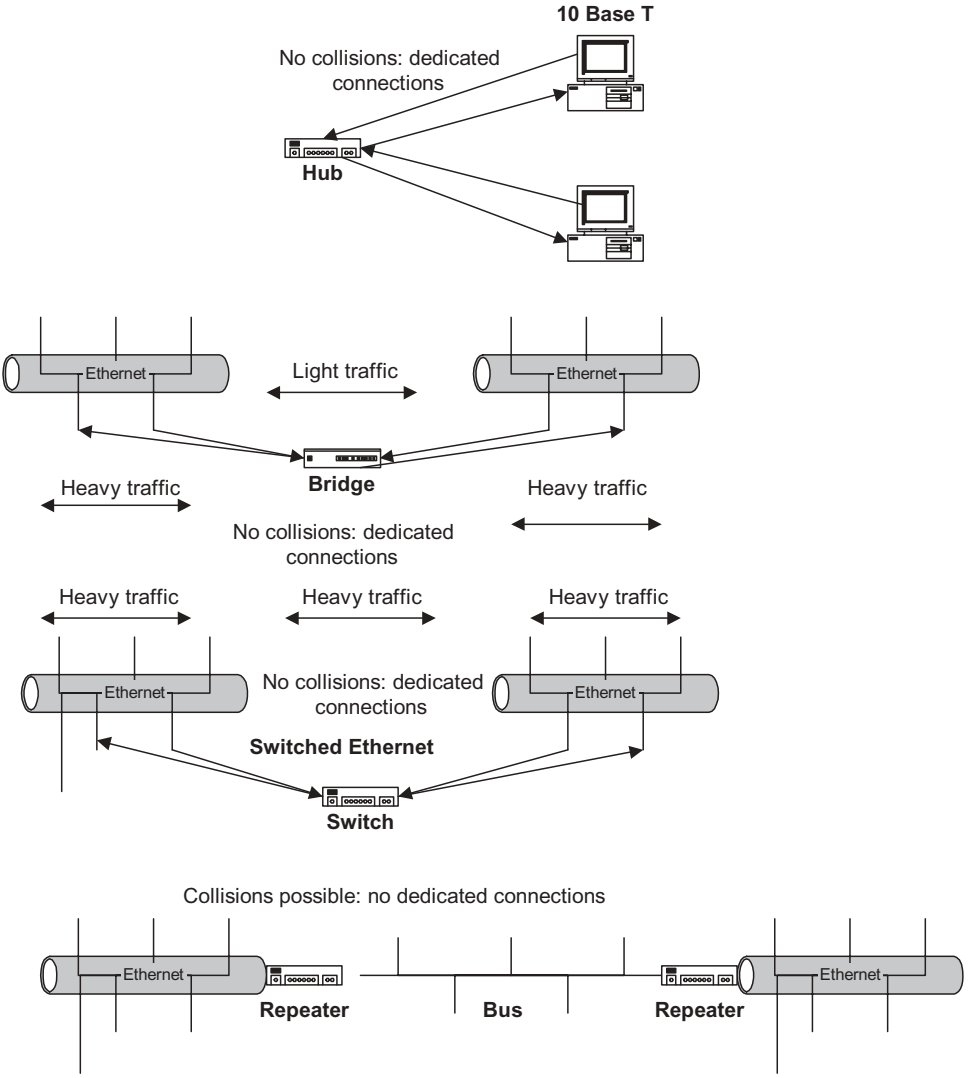


Figure 24 Network connectivity alternatives.

The most rudimentary connectivity device is the repeater. Its function is to repeat a signal, necessitated by the fact that signals can lose strength in traversing a network. The bridge has storage and routing capability that allows it to transfer traffic from one network to another. The primary application of the bridge is where most of the traffic is within the connected networks, with occasional traffic between networks.

Problem 49

In Figure 24, indicate whether collisions will or will not occur for the various connection methods.

Solution: See annotations on Figure 24.

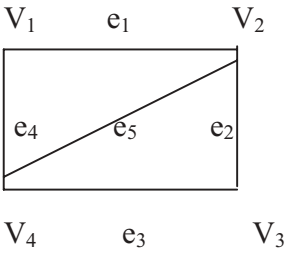
Adjacency Matrix in Network Topology

Definition: An adjacency matrix is a means of representing which vertices of a network topology graph are adjacent (i.e., directly connected by edges).

Problem 50

Given: Graph of vertices V_i below.

What is the adjacency matrix of vertices V_i and edges e_i?



Solution: Using the above definition, the adjacency matrix is shown in Table 8, where "1" indicates a connection:

Data Compression

Data compression is used to economize on data transmission or computation by eliminating bits that are not required in an application. For example, rather than use the hexadecimal format for the input number 16 in a computation, the binary representation is used. Thus, rather than 8 bits in hexadecimal, 5 bits in binary are used in the computation.

Table 8 Adjacency Matrix for Vertices V_i and Edges e_i

	V_1	V_2	V_3	V_4
V_1	0	$1(e_1)$	0	$1(e_4)$
V_2	$1(e_1)$	0	$1(e_2)$	$1(e_5)$
V_3	0	$1(e_2)$	0	$1(e_3)$
V_4	$1(e_4)$	$1(e_5)$	$1(e_3)$	0

Using data transmission to illustrate the decompression technique, the time required to transmit x bits of data in the noncompressed mode = x/B_n, where B_n is the bandwidth in bits per second (bps) for uncompressed data. With a compression factor $r \geq 1$, data will be compressed by x/r. The *time* that it takes to compress and decompress x bits of data is x/B_c, where B_c is the rate of compression/decompression. Thus, the *time* to compress, transmit, and decompress x bits of data is $(x/B_c) + ((x/r)/B_n)$. Then, compassion/decompression is beneficial if the (time to compress, transmit, and decompress x bits of data) < (time required to transmit x bits of data in the noncompressed mode):

$$(x/B_c) + ((x/r)/B_n) < (x/B_n), \text{ which is equivalent to } B_c > ((r/(r-1)B_n).$$

Problem 51

What is the value of the above inequality for r = 2?

Solution: $B_c > 2\ B_n$ (rate of compression/decompression > (2 bandwidth in bps for uncompressed data)).

Network Channel Capacity

Definitions

C: Channel capacity (bps) (Shannon's theorem)

BW: Bandwidth (hertz, Hz, or megahertz, MHz)

S: Signal power (watts)

N: Noise power (watts)

N_o: Noise spectral density (watts/Hz or MHz)

P_0: Power in system 0 (watts)

P_1: Power in system 1 (watts)

G: Gain or loss between P_0 and P_1 or between S and N (decibels)

$$C = BW\left(\log_2\left(1 + \frac{S}{N}\right)\right)$$

Shannon Limit

$$C = 1.44(S/N_0)$$

$$G = 10\log_{10}\left(\frac{P_1}{P_0}\right)$$

$$P_1 = (10)^{G/10} P_0$$

$$\mathbf{S/N} = (10)^{G/10}$$

Problem 52

Given: Network channel capacity C = 22.368 bps, signal power S = 14 watts, noise power N = 2 watts.

What is the required bandwidth BW of the network channel?

Solution:

$$\mathrm{BW} = \frac{\mathrm{C}}{\log_2\left(1+\frac{\mathrm{S}}{\mathrm{N}}\right)} = \frac{22.368}{\log_2\left(1+\frac{14}{2}\right)} = \frac{22.368}{\log_2(8)} = \frac{22.368}{3} = 7.456\ \mathrm{Hz}$$

Problem 53

Given: A microwave link is used to transmit binary data and has the following specifications: transmission bandwidth BW = 24 MHz, received signal-to-noise ratio = G = 20 dB.

According to Shannon's theorem, what is the network channel capacity C in megabits per second?

Solution:

$$\mathrm{C} = \mathrm{BW}\left(\log_2\left(1+\frac{\mathrm{S}}{\mathrm{N}}\right)\right), \mathrm{S/N} = (10)^{\mathrm{G}/10} = (10)^{20/10} = 100$$

$$\mathrm{C} = 24(\log_2(1+100)) = 24\ \mathrm{MHz} \times (\log_{10}(101)/\log_{10}(2))$$

$$= 2.004/0.3010 = 159.80\ \mathrm{Mbit/s}$$

Computer Circuit Analysis

Bode Plot and Amplifiers

A Bode plot relates the *absolute* value of the transfer function (|output/input|) = $|V_2/V_1|$ in Figure 25 to $\omega = 2\pi f$, where f = signal frequency, for example, an audio circuit that amplifies sound in a laptop computer.

Problem 54

Given: Amplifier diagram in Figure 25.

Problem: Identify the transfer function. Plot the transfer function versus ω.

Solution: Figure 25 shows the computation of the transfer function, according to the above definition. In addition, the figure shows the Bode plot, whose shape is governed by the fact that $V_2/V_1 = 1/\omega CR$ is an inverse function of $\omega = 2\pi f$. Thus, V_2/V_1 will continually decrease with f, and the 0 dB point corresponds to $\omega_c = 1/RC$, where $V_2/V_1 = 1$.

Operational Amplifier

This is a direct-current circuit amplifier that can be used to, for example, boost the signal derived from a sensor to a value that can be used in computer processing.

Problem 55

This problem, portrayed in Figure 26, involves determining the operational amplifier output voltage, V_o, given the temperature sensor input voltage V_s and the various

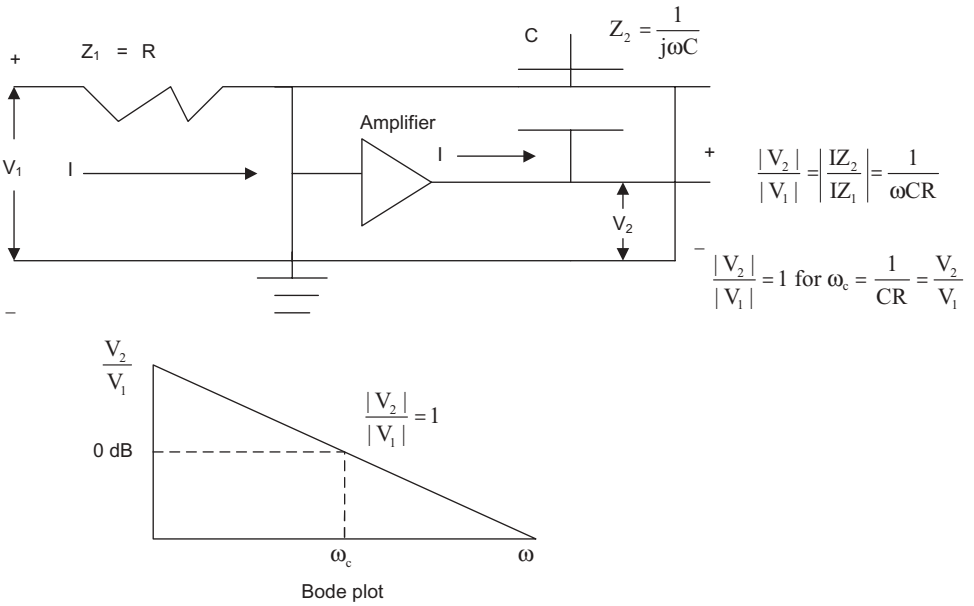


Figure 25 Amplifier diagram and Bode plot.

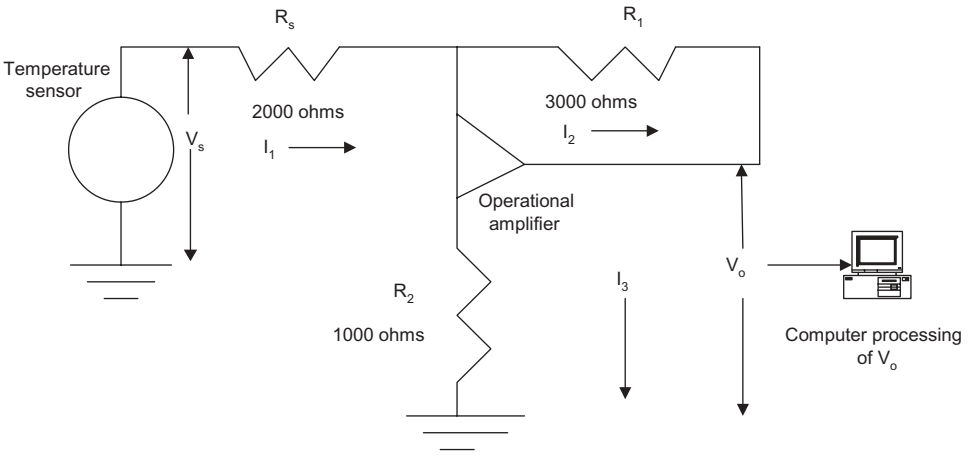


Figure 26 Operational amplifier circuit.

resistor values. Develop equations for impedance of total circuit, voltages, and currents, using Figure 26 as a guide.

Solution:

Impedance of total circuit, Z_1, comprised of series and parallel resistance circuits:

$$Z_1 = (R_s) + \left(\frac{R_1 R_2}{R_1 + R_2} \right),$$

then, the current I_1 is computed as:

$$I_1 = V_s / \left[(R_s) + \frac{R_1 R_2}{R_1 + R_2} \right].$$

The current I_2 is computed by dividing the voltage across it by its resistance:

$$I_2 = \frac{V_s - V_o}{R_1},$$

additionally, $I_3 = I_1 - I_2$, thus:

$$I_3 = I_1 - \left(\frac{V_s - V_o}{R_1} \right).$$

The output voltage V_o is across the resistor R_3 in Figure 26:

$$V_o = I_3 R_2 = \left[I_1 - \frac{V_s - V_o}{R_1} \right] R_2,$$

then substituting I_1 in this equation produces:

$$V_o = V_s / \left[(R_s) + \left(\frac{R_1 R_2}{R_1 + R_2} \right) - \left(\frac{V_s - V_o}{R_1} \right) \right] R_2.$$

The amplification factor, V_o/V_s, in Figure 26, is computed, using the foregoing equation, as follows:

$$\begin{aligned} V_o / V_s &= \frac{(R_1 + R_2) R_1}{(R_1 - R_2)[(R_s)(R_1 + R_2) + R_1 R_2]} - \frac{R_2}{(R_1 - R_2)} \\ &= \frac{(3000 + 1000) * 3000}{(3000 - 1000)[(2000)(3000 + 1000) + 3000 * 1000]} - \frac{1000}{(3000 - 1000)} \\ &= \frac{12,000,000}{(2000)[(2000)(4000) + 3,000,000]} - 0.5000 \\ &= \frac{12,000,000}{22,000,000} - 0.5 = 0.5454 - 0.5000 = 0.0454. \end{aligned}$$

The interpretation of this result is that 1°C measured by the temperature sensor corresponds to $V_s = 1$ V on input, increased by the operational amplifier to 0.0454 V on output.

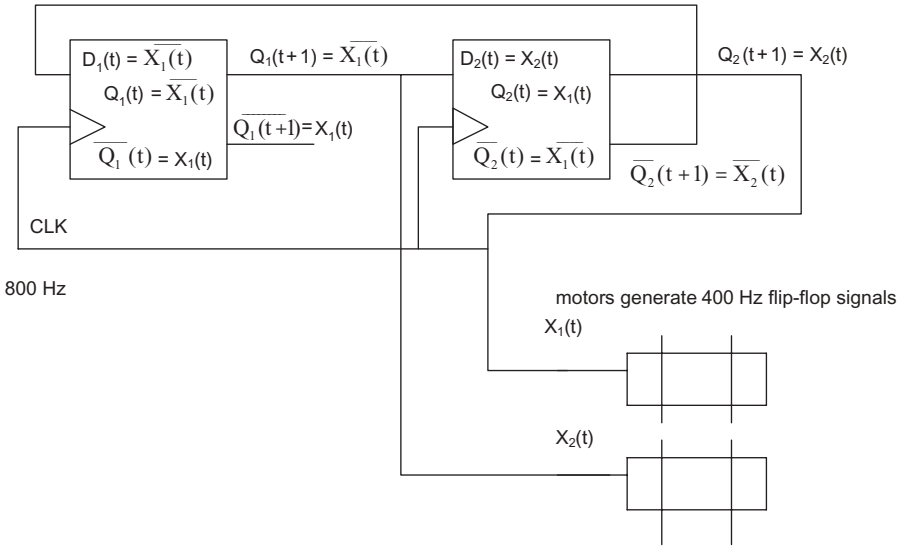


Figure 27 Motor circuit.

Table 9 D Flip-Flop Truth Table

$X_1(t)$	$X_2(t)$	$D_1(t) = X_1(t)$	$Q_1(t) = X_1(t)$	$Q_1(t+1) = X_1(t)$	$\overline{Q_1(t+1)} = X_1(t)$	$D_2(t) = X_2(t)$	$Q_2(t) = X_1(t)$	$Q_2(t+1) = X_2(t)$	$\overline{Q_2(t+1)} = X_2(t)$
0	0	1	1	1	0	0	0	0	1
0	**1**	1	1	**1**	0	1	0	**1**	0
1	**0**	0	0	**0**	1	0	1	**0**	1
1	1	0	0	0	1	1	1	1	0

Problem 56

It is required to determine the clock frequency in Figure 27, given that the output frequency for the motors is 400 Hz.

Solution: Using the truth table in Table 9, it can be seen that the flip-flop next state outputs Q_1 (t + 1) and Q_2 (t + 1), which are connected to the motors, change state only when the motor variables change ($X_1 = 0 \rightarrow 1$ and $X_2 = 1 \rightarrow 0$), indicated by the bolded items, requiring two clock pulses. Therefore, the input clock frequency = 800 Hz, as shown in Figure 27.

Shifting Circuits

Problem 57

Given: Arithmetic right shifting circuit in Figure 28, where the initial value of $D_2D_1D_0 = 001$.

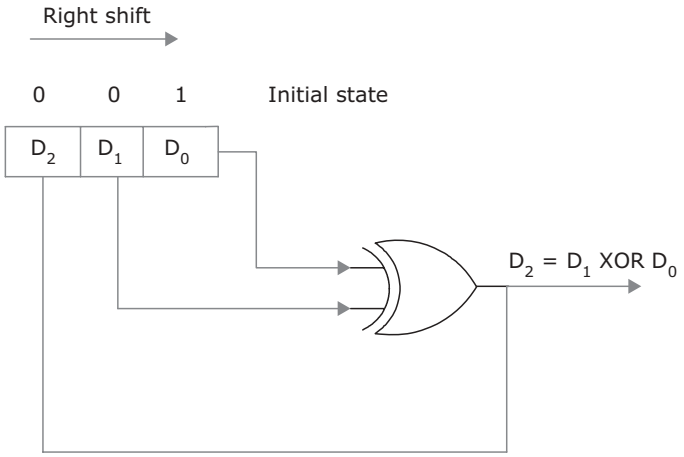


Figure 28 Right shift operation.

Table 10 Shifting Circuit Truth Table

Operation	D_2	D_1	D_0	$D_2 = D_1 \oplus D_0$
Initial state	**0**	**0**	**1**	
$D_2 = D_1 \oplus D_0$	1	0	1	1
Right shift	0 (insert 0)	1	0	
$D_2 = D_1 \oplus D_0$	1	1	0	1
Right shift	0 (insert 0)	1	1	
$D_2 = D_1 \oplus D_0$	0	1	1	0
Right shift, repeated state, with Initial state	**0** (insert 0)	**0**	**1**	

How many nonrepeating states are required for the counter to complete its cycle?

Solution:

First, notice the XOR (Exclusive OR) logical property: $D_2 = D_1 \oplus D_0$ ($D_2 = 0$, if D_1 and D_0 are the same, and $D_2 = 1$, if D_1 and D_0 are different). Then, referencing the truth table in Table 10, you can see that the circuit transitions through *seven states* when the initial state is repeated at the seventh state.

Practice Problems with Solutions 2

These practice problems are related to the following chapters:

Chapter 1: Digital Logic and Microprocessor Design

Chapter 2: Case Study in Computer Design

Chapter 9: Programming Languages

Chapter 10. Operating Systems

Chapter 4: Analog and Digital Computer Interactions

CHAPTER 1 (DIGITAL LOGIC AND MICROPROCESSOR DESIGN) AND CHAPTER 2 (CASE STUDY IN COMPUTER DESIGN)

Hard Disk Properties

Areal Density = ((bits per track)(number of tracks))/(disk radius)

Average Access Time = average latency + average seek time

Average Latency = revolution time/2

Revolution Time = 1/(revolutions per unit time)

Average Latency = 1/((2)(revolutions per unit time))

Average Seek Time = time to move from a given track to adjacent track

Problem 1(a)

Given: 10,000 bits per track, 1,000 tracks, disk diameter = 15.24 cm

Problem: What is the areal density?

Computer, Network, Software, and Hardware Engineering with Applications, First Edition. Norman F. Schneidewind.
© 2012 the Institute of Electrical and Electronics Engineers, Inc. Published 2012 by John Wiley & Sons, Inc.

Answer: Areal Density = ((bits per track)(number of tracks))/(disk radius) = ((10,000 bits per track)(1,000 tracks))/(7.62 cm) = 1.31×10^6 bits per centimeter

Problem 1(b)

Given: Average Seek Time = 20 milliseconds (ms), rotational speed = 2,000 revolutions per minute (rpm)

Problem: What is the Average Access Time?

Answer: Average Access Time = average latency + average seek time

Average Latency = 1/((2)(revolutions per unit time)) = (1/(2)(2,000 rpm/60 seconds per minute) = 0.015 seconds = 15 ms

Average Access Time = 15 ms + 20 ms = 35 ms

Disk Transmission Time = (data quantity)/(transmission rate)

Problem 2

Given: The contents of a 20 megabyte disk are transferred at a rate of 2400 bits per second.

Problem: What is the time required for this transmission?

Solution: Transmission Time = (data quantity)/(transmission rate) = (20 × 1,048,576 bytes × 8 bits per byte)/2400 bits per second = 69,905 seconds

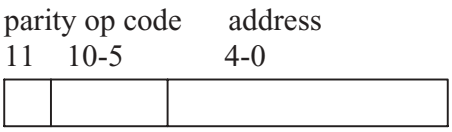
69,905 seconds/3,600 seconds per hour = 19.42 hours

Memory Properties

Problem 3

Problem: For the following instruction format, what is the required memory size?

Solution: 25 addresses × 12 bits per address = 32 × 12 = 384 bits



Problem 4

What is the Word *Access Time* for a random access memory (RAM)?

Answer: Time required to locate and read or write a word in RAM

Computer Software Fundamentals

Problem 5

Problem: Flow chart the following:

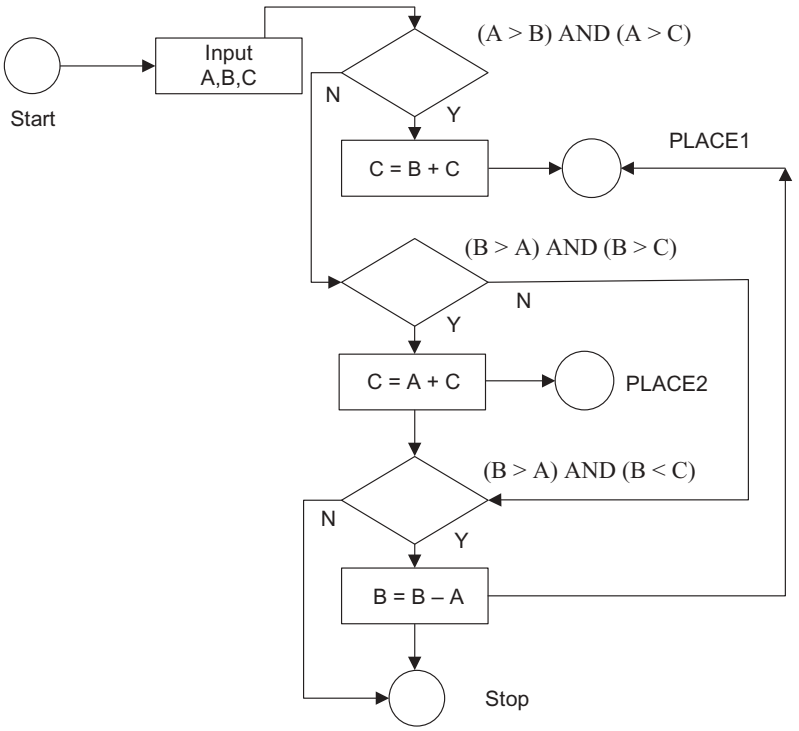


Figure 1 Problem 5.

If (A > B) AND (A > C), (C = B + C), → PLACE1

If (B > A) AND (B > C), (C = A + C), → PLACE2

If (B > A) AND (B < C), (B = B – A), → PLACE1

Solution: See Figure 1.

Problem 6

Problem: Flow chart the following:

If (A > 10) AND (A < 14), (A = A – X)

If (A < 10) OR (A > 14), exit

Solution: See Figure 2.

Problem 7

Problem: Flow chart the following: finding the *real* roots of a second-degree polynomial, where the coefficients a, b, and c are read from an input unit and the results are printed. The equation is: $ax^2 + bx + c = 0$, and the roots are computed in Figure 3.

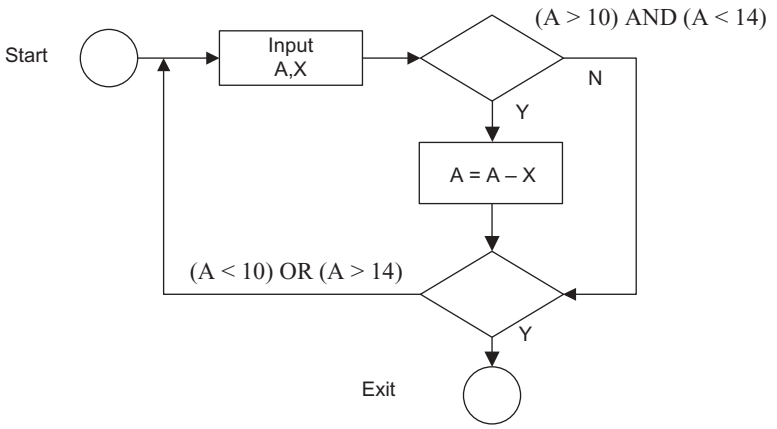


Figure 2 Problem 6.

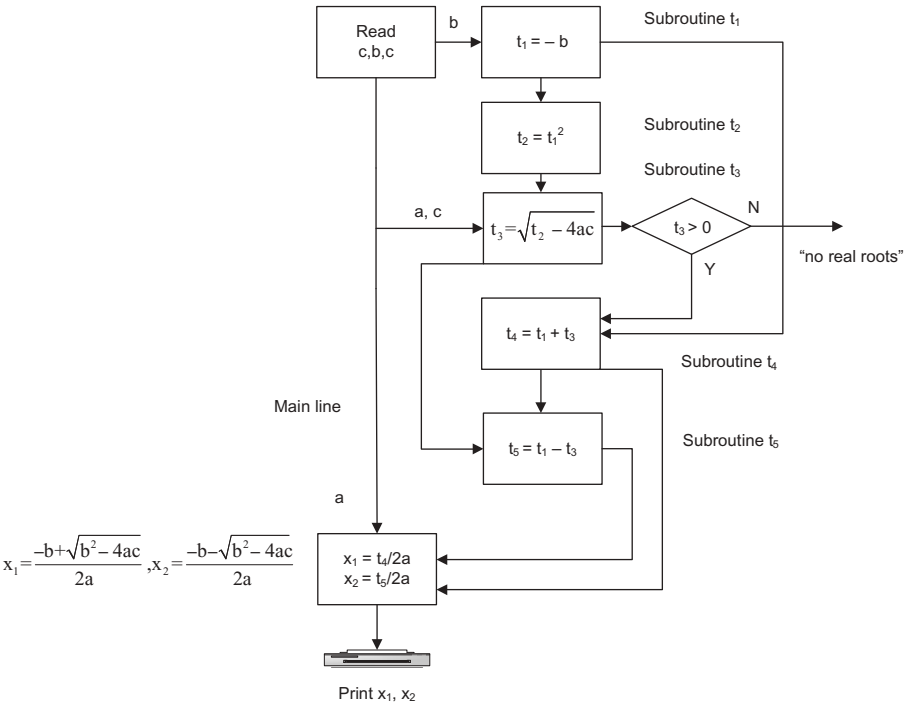


Figure 3 Roots of polynomial.

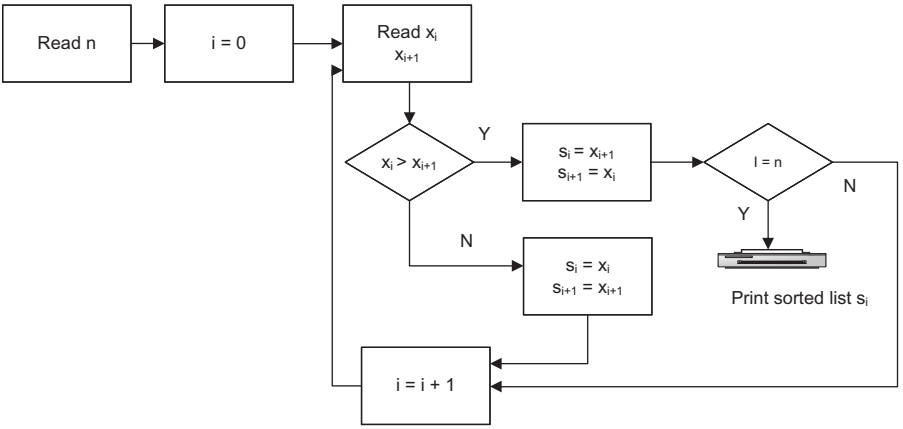


Figure 4 Bubble sort.

Bubble Sort

This sort is performed by comparing x_i with x_{i+1}. If $x_i > x_{i+1}$, x_i and x_{i+1} are interchanged ("swapped"); if $x_i \leq x_{i+1}$, no interchange takes place. This process continues until there are no more swaps. Then the sorted list is printed in ascending order.

Problem 8

Problem: Using a bubble sort, draw a flowchart for this process, for sorting the data in ascending order.

Solution: Figure 4 shows the flow chart process.

Binary Search

Let n = number of items to sort and p = number of comparisons required. First, the list to be searched must be sorted. Then, the search starts in the middle of the list and tests this item for equality with the search key. If equal, the search is finished. If search key > middle item, confine remaining search to lower half of list; otherwise, confine remaining search to upper half of list.

Problem 9

Derive the expression for number of comparisons p required to search a list of n items, using the binary search method, and compute p for n = 1, . . . , 10. Lastly, make a plot of p versus n.

Solution: The derivation follows. The example is tabulated in Table 1 and the plot is shown in Figure 5.

For n = 1, p = 0 (no items to sort); for n = 2, p = 1 (two items requiring one comparison); for n = 4, p = 2 (one comparison for two of the items and one comparison for the other two items), and so on. Thus, $n = 2^p$.

$$p = \log_2(n) = \log_{10}(n) / \log_{10}(2)$$

Table 1 Binary Search

Number of items	Number of comparisons
n	$p = \log_{10}(n)/\log_{10}(2)$
1	0.00
2	1.00
3	1.58
4	2.00
5	2.32
6	2.58
7	2.81
8	3.00
9	3.17
10	3.32

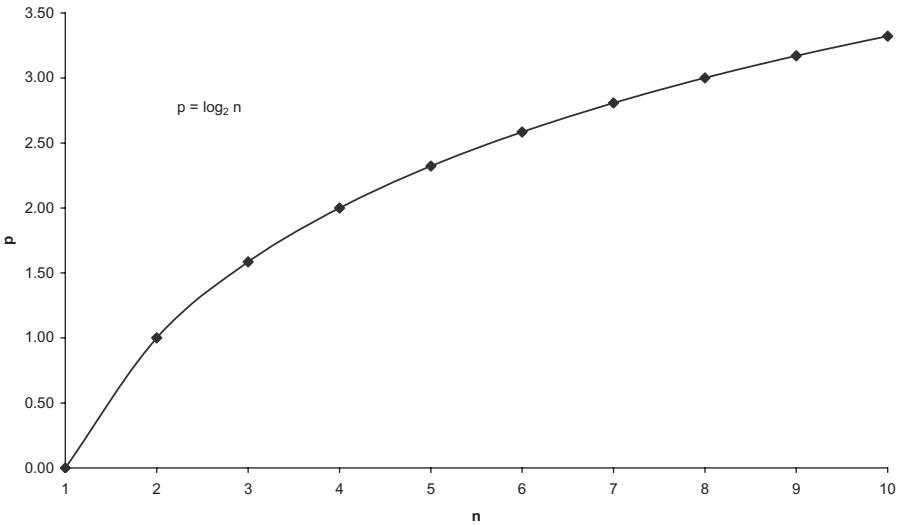


Figure 5 Number of comparisons p versus number of items n in binary search.

Computer Architecture

Problem 10

A control line with a width of 4 bits can control how many microprocessor operations?

Answer: $2^4 = 16$ operations (e.g., input, process, output).

Problem 11

A computer that has a single word size of 16 bits has how many bits in a double word?

Answer: 32 bits.

Table 2 TTL Logic Voltage versus Binary Value

Voltage	Binary value
+5	1
+4	**1**
+3	1
+2	1
+1	0
+0.8	0
0	0

Digital Logic

Transistor–Transistor Logic

Any voltage between +2 and +5 is considered to be binary 1; voltages outside this range are considered binary 0.

Problem 12

Problem: What binary value does 4 V in transistor–transistor logic (TTL) represent?

Solution: As shown in Table 2, the value = 1.

Memory Characteristics

Problem 13

A 64K byte RAM can store how many bytes?

Answer: $64 \times 2^{10} = 65{,}536$ bytes, where $K = 2^{10} = 1{,}024$.

Problem 14

A 65,536 byte RAM can store how many bits?

Answer: 65,536 × 8 bits per byte = 524,288 bits.

Minterms

A minterm is a *product* term of Boolean variables, such as $\overline{A}B\,\overline{C}\,\overline{D}$. Adjacent minterms in Table 3 are separated by one bit, for example, $\overline{A}B\,\overline{C}\,\overline{D}$ and $\overline{A}B\,\overline{C}\,D$. This formatting method provides for combining minterms into the minimum sum of products Boolean function (see below).

Problem 15

Given the terms A, B, C, and D in Table 3, what is the minterm $m_5 = 0101_2$?

Solution: Table 3 shows the required minterm = $\overline{A}B\,\overline{C}\,D$. Note that inverted Boolean variables represent binary 0 (e.g., $\overline{A} = 0$) and noninverted Boolean variables represent binary 1 (e.g., A = 1).

Table 3 Minterm Illustration

		$\overline{D} = 0 = 0$	D = 1
000	$\overline{A}\overline{B}\overline{C}$		
001	$\overline{A}\ \overline{B}\ C$		
011	$\overline{A}\ BC$		
010	$\overline{A}B\ \overline{C}$	$\overline{A}\ B\ \overline{C}\ \overline{D}$	Required minterm: $\overline{A}B\ \overline{C}\ D$
110	$AB\ \overline{C}$		
111	ABC		
101	$A\ \overline{B}\ C$		
100	$A\ \overline{B}\ \overline{C}$		

Table 4 Sum of Products and Product of Sums

A	B	C	Minterms: F(A,B,C), Sum of Products, *positive logic*	Maxterms: F (A,B,C), Product of Sums, *negative logic*
0	0	0	$\overline{A}\ \overline{B}\ \overline{C} = 1$	$A + B + C = 0$
0	0	1	$\overline{A}\overline{B}C = 1$	$A + B + \overline{C} = 0$
0	1	0	$\overline{A}\ B\ \overline{C} = 1$	$A + \overline{B} + C = 0$
0	1	1	$\overline{A}\ BC = 1$	$A + \overline{B} + \overline{C} = 0$
1	0	0	$A\ \overline{B}\ \overline{C} = 1$	$\overline{A} + B + C = 0$
1	0	1	$A\ \overline{B}\ C = 1$	$\overline{A} + B + \overline{C} = 0$
1	1	0	$AB\ \overline{C} = 1$	$\overline{A} + \overline{B} + C = 0$
1	1	1	$ABC = 1$	$\overline{A} + \overline{B} + \overline{C} = 0$

Sum of Products and Product of Sums

A *sum of products* is formed by *summing* the *product terms* obtained wherever there is a *1* for the function F(A, B, C), as illustrated in Table 4. The product terms are called *minterms*. A *product of sums* is formed by finding the *product of sums terms* obtained wherever there is a *0* for the function F(A, B, C), as illustrated in Table 4. The sum terms are called *maxterms*. Note that *positive logic* is used for Sum of Products (e.g., A = 1 and $\overline{A} = 0$) (minterms), and *negative logic* is used for Product of Sums (e.g., A = 0 and $\overline{A} = 1$) (maxterms). Also note that when there are only single terms in the expressions, as in Table 4, the sum of products *in each cell* is simply a single product and the product of sums *in each cell* is simply a single sum.

Problem 16

Given: Table 4 values of A, B, and C.

Problem: What is the sum of products (minterms) and product of sums (maxterms) in Table 4?

Solution: Form the sum of products and the product of sums, according to the relationships of the terms in Table 4.

Table 5 Karnaugh Map of Sum of Products and Product of Sums

	BC			
A	00	01	11	10
0	$M_0 = 0$	$M_1 = 0$	$m_3 = 1$	$M_2 = 0$
1	$M_4 = 0$	$M_5 = 0$	$m_7 = 1$	$M_6 = 0$

C B BC C

Table 6 Truth Table for NOR, XOR and XNOR

A	B	NOR: $\overline{A+B}$	XOR: $A\,\overline{B}+\overline{A}\,B$	XNOR: $AB+\overline{A}\,\overline{B}$
0	0	1	0	1
0	1	0	1	0
1	0	0	1	0
1	1	0	0	1

Karnaugh Map

The expressions for the *sum of products* and *product of sums* can be minimized by creating a Karnaugh map that will capture both types of digital logic in one table and show the relationship between the two. That is, the Product of Sums, using *positive logic*, is equal to the Sum of Products, using *negative logic*.

Problem 17

Using the following example, demonstrate the above relationships in Table 5.

Minterms: $F(A, B, C) = \overline{A}BC + ABC = m_3 + m_7 = BC(\overline{A} + A) = BC$ (positive logic)

Maxterms:

$$F(A, B, C) = (A+B+C)(A+B+\overline{C})(A+\overline{B}+C)(\overline{A}+B+C)(\overline{A}+B+\overline{C})(\overline{A}+\overline{B}+C)$$
$$= M_0M_1M_2\mathbf{M}_4M_5M_6 \text{ (negative logic)} = BC$$

Minterms Sum of Products = BC

Maxterms Product of Sums = Product of Sums = BC

Logic Functions

Logic functions are very useful for designing digital circuits. The truth table in Table 6 gives the outputs of each function, based on the inputs.

Negative OR: NOR: $\overline{A+B}$

Exclusive OR: XOR: $A\,\overline{B}+\overline{A}\,B$

Invert XOR to obtain Exclusive NOR

XNOR: $$\overline{A\,\overline{B}+\overline{A}\,B}=(\overline{A}\,\overline{\overline{B}})(\overline{\overline{A}}\,\overline{B})=(\overline{A}+B)(A+\overline{B})=\overline{A}\,A+AB+\overline{A}\,\overline{B}+B\,\overline{B}$$
$$=0+AB+\overline{A}\,\overline{B}+0=AB+\overline{A}\,\overline{B}$$

Problem 18

Using Table 7, and *positive voltage logic*, what are the binary values for the XOR function, F(A, B) in Table 7?

Solution: Convert −5 V to binary 0 and 0 V to binary 1 in Table 7. This operation shows the F(A, B) = XOR, based on logic rules in Table 6. Note that for XOR, F(A, B) = 1 when either A or B = 1, but not both = 0 or = 1.

Problem 19

If *negative voltage logic* is used in the truth table, as shown in Table 8, what function is produced for F(A, B)?

Solution: Convert −5 V to binary 1 and 0 V to binary 0 in Table 8. Referring to Table 8, and comparing it with the logic rules in Table 6, it is seen that the XNOR function is produced. Note that the Table 8 result is the inverse of the Table 7 result (XNOR is inverse of NOR):

$$\overline{A\,\overline{B}+\overline{A}\,B}=(\overline{A\overline{B}})(\overline{A\overline{B}})=(\overline{A}+B)(A+\overline{B})=\overline{A}\,A+AB+\overline{A}\,\overline{B}+B\,\overline{B}$$
$$=0+AB+\overline{A}\,\overline{B}+0=AB+\overline{A}\,\overline{B}$$

Logic Network Design

De Morgan's theorem ($\overline{A+B}=\overline{A}\,\overline{B}$ and $\overline{AB}=\overline{A}+\overline{B}$) is used to simplify complex logic equations and the resultant digital logic. The theorem is used to simplify

Table 7 Truth Table Using Positive Voltage Logic (F(A, B) = XOR))

A	B	F(A, B): $A\,\overline{B}+\overline{A}\,B$
–5 V (0)	–5 V(0)	(0)(1) + (1)(0) = 0: –5 V (0)
–5 V(0)	0 V (1)	(0)(0) + (1)(1) = 1: 0 V (1)
0 V (1)	–5 V(0)	(1)(1) + (0)(0) = 1: 1 V (1)
0 V (1)	0 V (1)	(1)(0) + (0)(1) = 0: –5 V (0)

Table 8 Truth Table Using Negative Voltage Logic (F(A, B) = XNOR)

A	B	F(A, B): $AB+\overline{A}\,\overline{B}$
–5 V (1)	–5 V(1)	(1)(1) + (0)(0) = 1: –5 V (1)
–5 V(1)	0 V (0)	(1)(0) + (0)(1) = 0: V (0)
0 V (0)	–5 V (1)	(0)(1) + (1)(0) = 0: 0 V (0)
0 V (0)	0 V (0)	(0)(0) + (1)(1) = 1: –5 V (1)

relatively simple expressions, as contrasted with Karnaugh maps, which can minimize complex Boolean expressions. The application of this theorem is shown in Problem 20.

Problem 20

Suppose it is required to simplify $F = \overline{((\overline{AB})(\overline{AB}))}$

Solution:

Applying *De Morgan's theorem*:

$$(\overline{AB})(\overline{AB}) = (\overline{A} + \overline{B})(\overline{A} + \overline{B}) = \overline{A}\ \overline{A} + \overline{A}\ \overline{B} + \overline{A}\ \overline{B} + \overline{B}\ \overline{B} = \overline{A}\ \overline{B} + \overline{A}\ \overline{B}$$

$$(\overline{\overline{A}\ \overline{B} + \overline{A}\ \overline{B}}) = (\overline{A}\ \overline{B})(\overline{A}\ \overline{B}) = (A + B)(A + B) = AB$$

Problem 21

Then, demonstrate equivalence between $\overline{((\overline{AB})(\overline{AB}))}$ and AB in Table 9.

Karnaugh Maps

Problem 22

A Karnaugh map, showing *maxterms*, appears in Table 10. A *maxterm* appears for cells that contain 0: $M_4 = 100$, $M_5 = 101$, and $M_6 = 110$. Notice that *negative logic* is used for labeling maxterms (e.g., M_3 corresponds to $100 = \overline{A}\ BC$).

For maxterms M_4, M_5, and M_6, which maxterms are adjacent in the Karnaugh map?

Solution: Referring to Table 10, maxterms M_4 and M_5 and M_4 and M_6 are adjacent (i.e., adjacent cells have a difference of 1 bit). M_4 and M_5 differ by C = 0, 1. M_4 and M_6 differ by B = 0, 1.

Table 9 Truth Table to Demonstrate Equivalence between F and AB

A	B	$\overline{AB}$	$\overline{AB}\ \overline{AB}$	$F = \overline{((\overline{AB})(\overline{AB}))}$	A B
0	0	1	1	0	0
0	1	1	1	0	0
1	0	1	1	0	0
1	1	0	0	1	1

Table 10 Karnaugh Map of Maxterms

	BC			
A	00	01	11	10
0				
1	0	0		0

M_4 M_5 M_6

Table 11 Sum of Products and Product of Sums Truth Table

	AB			
C	00	01	11	10
0	*0*	**1**	**1**	**1**
1	*0*	*0*	*0*	*0*

Sum of Products
(Minterms) = $\overline{C}((\overline{A}+B)+(A+B)+(A+\overline{B})) = \overline{C}(((\overline{A}+A)+(B+\overline{B})+(A+B))) = \overline{C}(A+B) = \overline{C}\,A + \overline{C}\,B$

Table 12 Sum of Products and Product of Sums Karnaugh Map

	AB			
C	00	01	11	10
0	*0*	**1**	**1**	**1**
1	*0*	*0*	*0*	*0*

$A + B$ Maxterm; $\overline{C}$ Maxterm; $\overline{C}\,B$ Minterm; $\overline{C}\,A$ Minterm

Sum of Products = $\overline{C}\,A + \overline{C}\,B = \overline{C}(A+B)$
Product of Sums = $\overline{C}(A+B)$

Problem 23

Given the values of 0 and 1 in Table 11, what is the *sum of products* value?

Solution: The *sum of products* is formed, using the terms A, B, and C, wherever **1**s appear in the table, using *positive logic*. These are the *minterms*, bolded.

Problem 24

What digital logic is used to produce the *product of sums* in Table 11?

Solution: The *product of sums* is produced according to the cells that have *0*s in Table 11. These are the maxterms, italicized, as shown in Table 11, using *negative logic*.

Product of Sums (Maxterms) = $(C+A+B)(\overline{C}+A+B)(\overline{C}+A+\overline{B})(\overline{C}+\overline{A}+\overline{B})$ $(\overline{C}+\overline{A}+B)$

Problem 25

In Table 12, show the simplification of both Sum of Products and Product of Sums, based on the values recorded in the truth table, Table 11.

Solution: See the Karnaugh map solution for simplifying Sum of Products and Product of Sums in Table 12.

Thus the sum of products and product of sums are equal.

Table 13 Karnaugh Map

	AB			
CD	00	01	11	10
00		1		1
01	1	1	1	1
11		1	1	
10	1			

$C\,\overline{D}\,\overline{A}\,\overline{B}$ $\overline{C}\,D$ $\overline{C}\,\overline{A}\,B$ BD $\overline{C}\,A\,\overline{B}$

$F = C\,\overline{D}\,\overline{A}\,\overline{B} + \overline{C}\,D + \overline{C}\,\overline{A}\,B + BD + \overline{C}\,A\,\overline{B}$

Table 14 Karnaugh Map

	AB			
C	00	01	11	10
0	1	1	1	1
1		X	X	1

X: don't care

$\overline{C}$ A

$F = A + \overline{C}$

Problem 26

Given the Karnaugh map in Table 13 that has 1s inserted, develop the Boolean expression F.

Solution: Use the minimum number of enclosures to encompass the 1s in Table 13. This process yields the minimum expression for F.

Problem 27

In the Karnaugh map, shown in Table 14, what is the minimum Boolean expression that can be developed?

Solution: The key to the solution is to use the "don't care (X) cells to maximum advantage. The meaning of "don't care" cells is that the minimum Boolean expression that can be developed is not affected by the cells with Xs. In this problem, only one of the Xs is required to obtain the required coverage.

Problem 28

For the maxterm function $\prod(M_1M_3M_4M_7)$, what is the logic that implements this function?

Solution: Obtain maximum coverage by enclosing the minimum number of 0 cells in the Karnaugh map of Table 15, taking advantage of the don't care

Table 15 Karnaugh Map

	AB			
C	00	01	11	10
0				$M_4 = 0$
1	$M_1 = 0$	$M_3 = 0$	$M_7 = 0$	X

$\overline{C}$ $\quad$ $\overline{A} + B$

$F = A + \overline{C}$

Table 16 Truth Table

Inputs			Outputs		
A	B	C	D	E	F
0	0	0	0	1	1
0	0	1	0	X	1
0	1	0	X	1	0
0	1	1	0	0	0
1	0	0	1	1	X
1	0	1	1	1	X
1	1	0	0	X	X
1	1	1	X	1	X

Table 17 Karnaugh Map: D Output

	BC			
A	00	01	11	10
0	0	0	0	D = X
1	$\overline{D} = 1$	D = 1	D = X	0

$A\overline{B}$ $\quad$ A $\quad$ $\overline{B}$

Sum of Products: $D = A\overline{B}$

Product of Sums: $D = A\overline{B}$

term X. The *negative* logic product of sums solution is shown at the bottom of Table 15.

For the truth table in Table 16 develop the Karnaugh map for the *outputs* D (Table 17), E (Table 18), and F (Table 19), taking into account the don't care conditions (X), for both *sum of products* and *product of sums*. Then use the Karnaugh maps to write the Boolean expressions for the outputs. In addition, show the Veitch diagram in Table 20 for the *outputs* D, E, and. F. (A Veitch diagram shows the Boolean logic

Table 18 Karnaugh Map: E Output

	BC			
A	00	01	11	10
0	E = 1	E = X	0	E = 1
1	E = 1	E = 1	E = 1	E = X

$A + \overline{C}$ A $\overline{C}$

Sum of Products: $E = A + \overline{C}$

Product of Sums: $E = A + \overline{C}$

Table 19 Karnaugh Map: F Output

	BC			
A	00	01	11	10
0	F = 1	F = 1	F = 0	F = 0
1	F = X	F = X	F = X	F = X

Sum of Products: $F = \overline{B}$ Product of Sums: $F = \overline{B}$

Table 20 Veitch Diagram

	BC			
	00	01	11	10
	$\overline{B}$		B	
	$\overline{C}$	C		$\overline{C}$
0 $\overline{A}$	D = 0 E = 1 F = 1	D = 0 E = X F = 1	D = 0 E = 0 F = 0	D = X E = 1 F = 0
1 A	D = 1 E = 1 F = X	D = 1 E = 1 F = X	D = X E = 1 F = X	D = 0 E = X F = X

for the relationships between *input* variables A, B, and C and *output* variables D, E, and F.)

Solution: Tables 17–19 show how the Karnaugh map is used to obtain the minimum Boolean expressions for the *outputs* D, E, and F, using the 1 and X values for *sum of products* (positive logic) and 0 and X values for *product*

of sums (negative logic) in Table 16. We see in every case that *sum of products = product of sums*.

Problem 29

For the *sum of products* Karnaugh maps in Tables 17–19, develop the digital logic diagram.

Solution: This digital logic is shown in Figures 6–8, corresponding to Tables 17–19, respectively, for implementing the *sum of products* (minterms) circuits.

Problem 30

Now develop the digital logic in Figures 6–8 to implement the *product of sums* (maxterms).

Solution: The logic Figures 6–8 is used to implement the *product of sums* (maxterms) circuit. The implementations are the same because *sum of products = product of sums*.

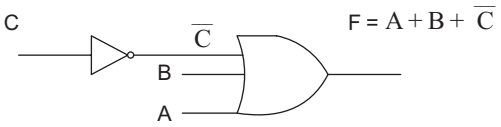


Figure 6 OR gate logic.

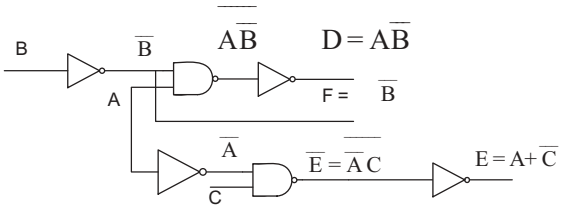


Figure 7 NAND gate logic.

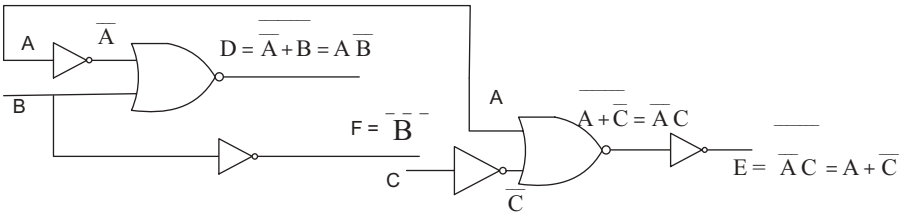


Figure 8 NOR gate logic.

Decoders

A decoder generates 2^n *possible* outputs for n inputs, when enabled.

Problem 31

Problem: Considering the block diagram of the 4-to-16 decoder circuit shown in Figure 9, develop a circuit using NAND gate logic to implement the outputs to the microprocessor ports.

Solution: Based on the design of the decoder block diagram shown in Figure 9, the NAND gate logic for the outputs to the microprocessor ports is shown in Figure 9.

Problem 32

Problem: Produce a 4-to-16 decoder, but only the outputs Y = 1 in the truth table, Table 21, are required. Use a Karnaugh map in Table 22 to minimize the complexity of the circuit.

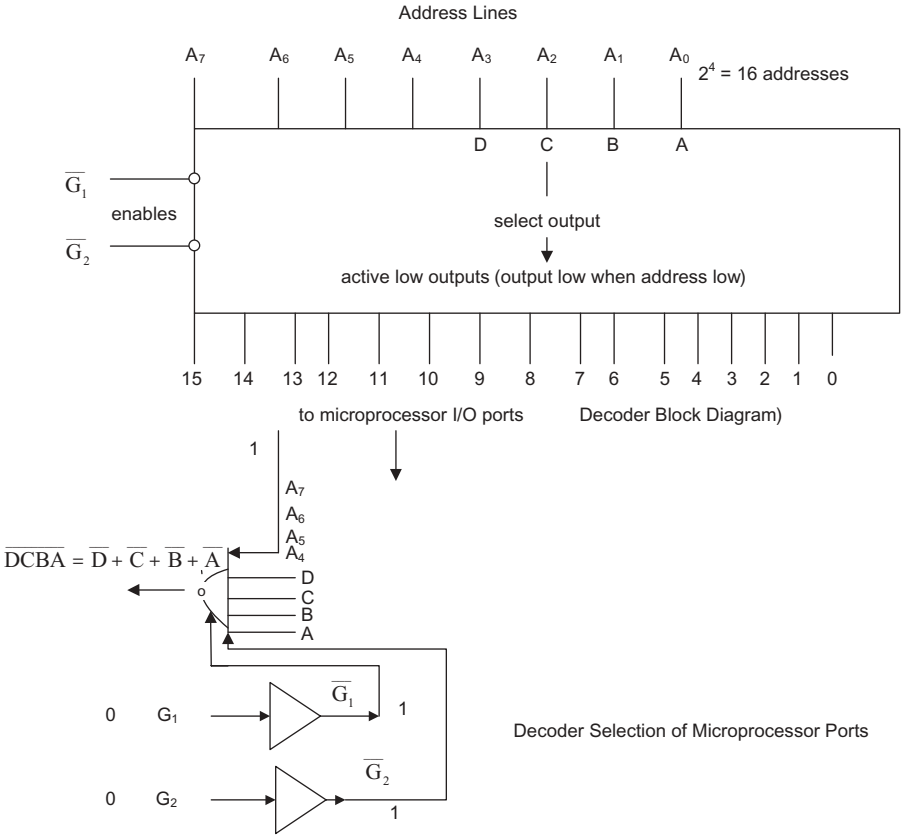


Figure 9 Decoder circuits. I/O, input/output.

Table 21 A 4-to-16 Decoder Truth Table

Address				Output
A_3	A_2	A_1	A_0	Y
0	**0**	**0**	**0**	$Y_0 = \mathbf{1}$
0	0	0	1	$Y_1 = 0$
0	**0**	**1**	**0**	$Y_2 = \mathbf{1}$
0	0	1	1	$Y_3 = 0$
0	**1**	**0**	**0**	$Y_4 = \mathbf{1}$
0	1	0	1	$Y_5 = 0$
0	**1**	**1**	**0**	$Y_6 = \mathbf{1}$
0	1	1	1	$Y_7 = 0$
1	**0**	**0**	**0**	$Y_8 = \mathbf{1}$
1	0	0	1	$Y_9 = 0$
1	**0**	**1**	**0**	$Y_{10} = \mathbf{1}$
1	0	1	1	$Y_{11} = 1$
1	1	0	0	$Y_{12} = 0$
1	**1**	**0**	**1**	$Y_{13} = \mathbf{1}$
1	1	1	0	$Y_{14} = 0$
1	**1**	**1**	**1**	$Y_{15} = \mathbf{1}$

Table 22 Karnaugh Map of 4-to-16 Decoder

	A_1A_0			
	00	01	11	10
A_3A_2				
00	$Y_0 = 1$			$Y_2 = 1$
01	$Y_4 = 1$			$Y_6 = 1$
11		$Y_{13} = 1$	$Y_{15} = 1$	
10	$Y_8 = 1$			$Y_{10} = 1$

$\overline{A_3}\,\overline{A_0}$ $A_3\overline{A_2}\,\overline{A_0}$ $A_3A_2A_0$ $\overline{A_3}\,\overline{A_0}$ $A_3\overline{A_2}\,\overline{A_0}$

$Y_0 + Y_2 + Y_4 + Y_6 = \overline{A_3}\,\overline{A_0}$, $Y_{13} + Y_{15} = A_3A_2A_0$, $Y_8 + Y_{10} = A_3\,\overline{A_2}\,\overline{A_0}$, $Y_{13} + Y_{15} = A_3A_1A_0$

Proof of $Y_0 + Y_2 + Y_4 + Y_6 = \overline{A_3}\,\overline{A_0}$

$$= \overline{A_3}\,\overline{A_2}\,\overline{A_1}\,\overline{A_0} + \overline{A_3}\,\overline{A_2}\,A_1\,\overline{A_0} + \overline{A_3}\,A_2\,\overline{A_1}\,\overline{A_0} + \overline{A_3}\,A_2\,A_1\,\overline{A_0}$$

$$= \overline{A_3}\,\overline{A_2}\,\overline{A_0}(\overline{A_1} + A_1) + \overline{A_3}\,A_2\overline{A_0}\,(\overline{A_1} + A_1)$$

$$= \overline{A_3}\,\overline{A_2}\,\overline{A_0} + \overline{A_3}\,A_2\,\overline{A_0} = \overline{A_3}\,\overline{A_0}\,(\overline{A_2} + A_2) = \overline{A_3}\,\overline{A_0}$$

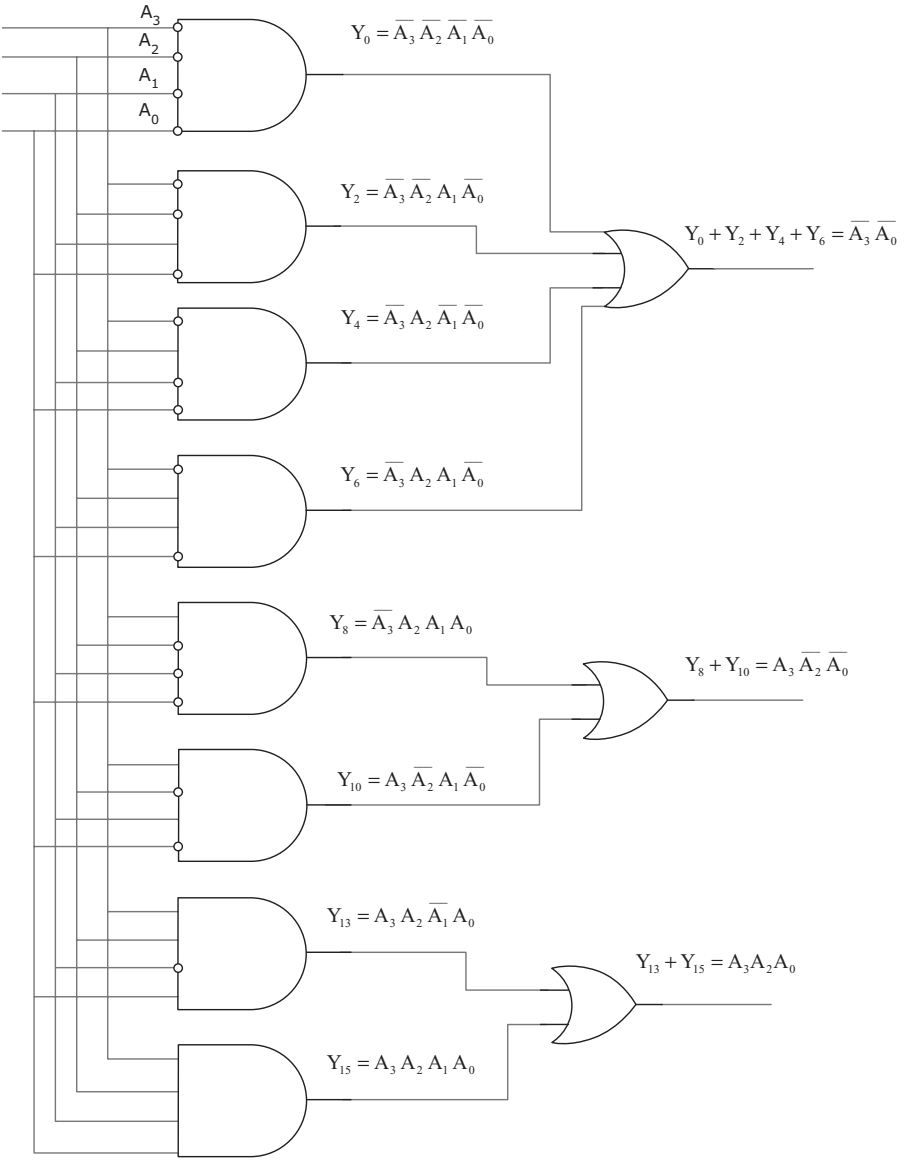


Figure 10 A 4-to-16 decoder circuit.

Solution: Using Table 21, where the bolded values correspond to Y = 1 outputs, and the logic simplification provided by the Karnaugh map in Table 22, write the Boolean expressions for the *combined, simplified* outputs, and the *individual outputs*, as a function of the address bits, and produce the decoder circuit in Figure 10.

Similar proofs could be developed for the other values of Y.

Table 23 Quine–McCluskey Method for $F = \overline{A}\ \overline{B}\ \overline{C} + \overline{A}\ \overline{B}\ C + A\ \overline{B}\ \overline{C} + A\ \overline{B}\ C = \overline{B}\ \overline{C}(\overline{A} + A) + \overline{B}\ C(\overline{A} + A) = \overline{B}(\overline{C} + C) = \overline{B}$

Minterm		ABC	Difference of 1 Minterms		Difference of 1 Minterms	Minterms	prime implicant
0	$\overline{A}\ \overline{B}\ \overline{C}$	000					
1	$\overline{A}\ \overline{B}\ C$	001	0,1	00-			
4	$A\ \overline{B}\ \overline{C}$	100	4,5	10-	0,1,4,5	-0-	$\overline{B}$
5	$A\ \overline{B}\ C$	101					

Quine–McCluskey Method

This method is an alternative to the Karnaugh map for minimizing a Boolean function. This method is used to represent a difference of 1 between two adjacent minterms, such as $\overline{A}\ \overline{B}\ \overline{C}$ and $\overline{A}\ \overline{B}\ C$, yielding $\overline{A}\ \overline{B}$- = 00-. The symbol - is placed where is a difference in minterm bit values, such as between 00- and 10- in Table 23, yielding -0-. This process continues until the four minterms 0, 1, 4, 5and show a difference of 1 (00- compared with 10-), yielding a prime implicant (-0-), where a prime implicant results from combining the maximum number of minterms, as in Table 23.

Problem 33

Find the prime implicant for the function $F = \overline{A}\ \overline{B}\ \overline{C} + A\ \overline{B}\ \overline{C} + \overline{A}\ \overline{B}\ C + A\ \overline{B}\ C$.

Solution: Table 23 shows the prime implicant solution: $\overline{B}$

Synchronous Sequential Networks

A Synchronous Sequential Network has both flip-flops and memory.

Reset–Set (RS) Flip-Flop

The next states, Q(t + 1) and $\overline{Q}(t+1)$, are the following:

$$Q(t+1) = S + \overline{R}\ Q(t) \quad \text{and} \quad \overline{Q}(t+1) = \overline{S + \overline{R}\ Q(t)} = (\overline{S})(\overline{\overline{R}\ Q(t)}) = (\overline{S})(R + \overline{Q(t)})$$

The states S = 1 (set) and R = 1 (reset) are not allowed simultaneously in an RS flip-flop because this would constitute an indeterminate state. The RS flip-flop is the building block for all other flip-flops (JK, D, and T) because these flip-flops can be derived from the RS flip flop.

JK Flip-Flop

The next states, Q (t + 1) and $\overline{Q}(t+1)$, are the following:

$$Q(t+1) = J\ \overline{Q}(t) + \overline{K}\ Q(t)$$

$$\overline{Q}(t+1) = \overline{J\ \overline{Q}(t) + \overline{K}\ Q(t)} = (\overline{J\ \overline{Q}(t)})(\overline{\overline{K}\ Q(t)}) = (\overline{J}\ Q(t))(K + \overline{Q(t)})$$

$$= \overline{J}\ (K + \overline{Q(t)}) + KQ(t)$$

D Flip-Flop

In the D flip-flop, Q follows D: $Q(t + 1) = D$, $\overline{Q}(t+1) = \overline{D}$

Problem 34

Given: RS flip-flop timing sequence in Figure 11.

What is the set (S) sequence in Figure 11?

Solution: Based on the next states rules above, and assuming $Q(t) = 0$, the sequence $S = 101$, as shown in Figure 11, where the set–reset sequence occurs on the falling edge of the clock pulse.

Problem 35

As shown in Figure 12, an RS flip-flop will cycle through set and reset states based on the next states rules, input values, and the initial values of the flip-flop.

Develop the expression for the next state $Q(t + 1)$ of the RS flip-flop output.

Solution: The next state $Q(t + 1)$ is shown in Figure 12.

Problem 36

Given the flip-flop circuit in Figure 13, develop the Boolean expressions for the inputs S_1, R_1, J_2, R_2, and D_3 as a function of the present states Q_1, Q_2, and Q_3 of

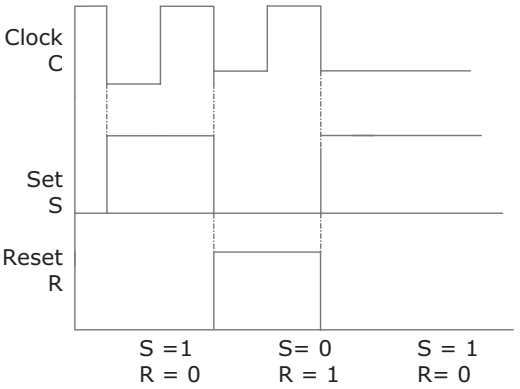


Figure 11 RS flip-flop sequence.

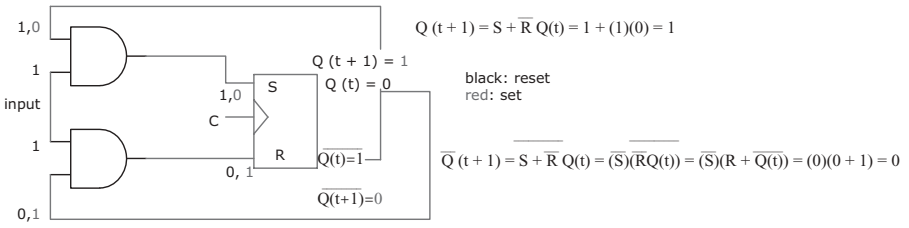


Figure 12 RS flip-flop states.

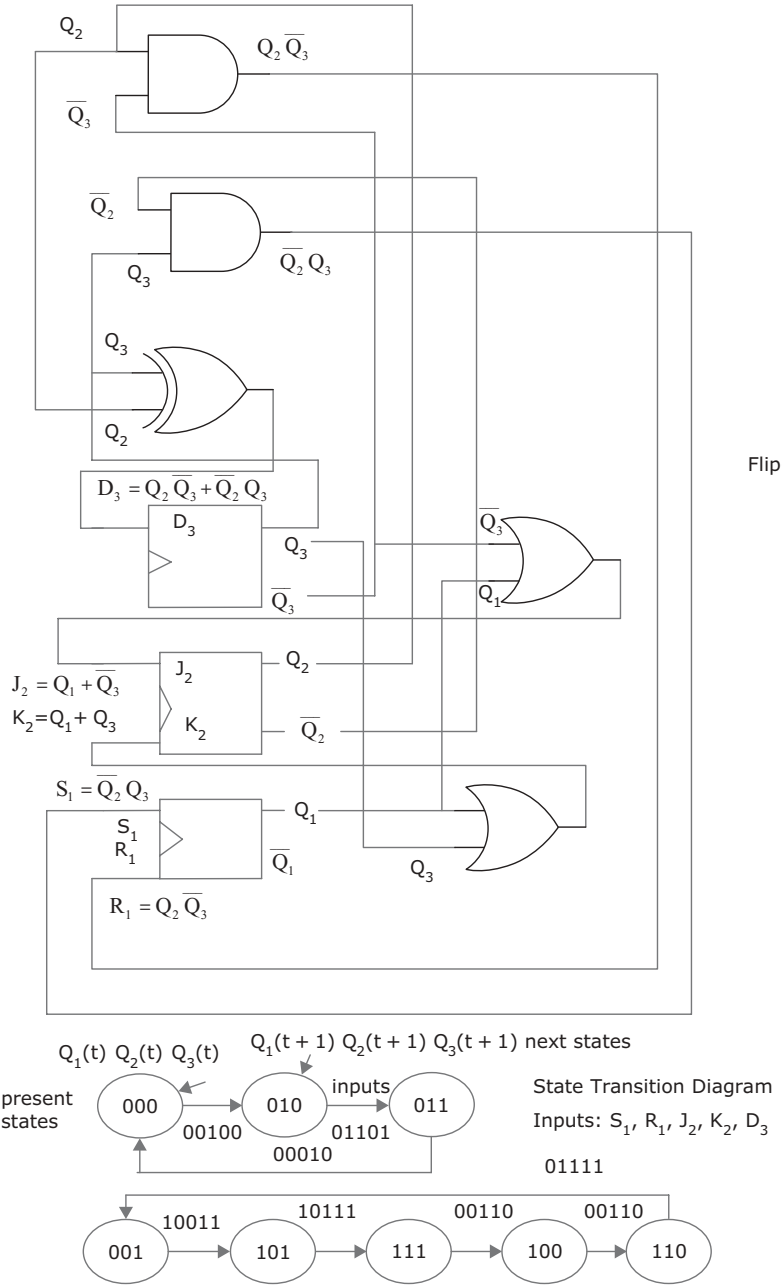


Figure 13 Flip-flop circuit and state transition diagram.

the flip-flops. Use these expressions to construct the state transition diagram in Figure 13.

Solution: First construct the flip-flop state table in Table 24, identifying the flip-flop *present state outputs* and *inputs*. Then, using the Boolean expressions previously developed for the RS, JK, and D flip-flops, identify the *next flip-flop output states*. Finally, based on the state transitions (mapping of Present State of Flip-Flop Outputs to Next State of Flip-Flop Outputs), bolded in Table 24, construct the state transition diagram in Figure 13.

Problem 37

Based on the JK flip-flop counter circuit diagram in Figure 14, develop the state table, Table 25, and the state transition diagram for the circuit.

Solution: First, using the circuit diagram in Figure 14, write the Boolean expressions for the flip-flop inputs and the current states of flip-flop outputs; the flip-flop state table in Table 25 is the result. Next, using the next state expression for the JK flip-flop, $Q(t+1) = J\,\overline{Q}(t) + \overline{K}\,Q(t)$, formulate the next state values in Table 25, color coding the corresponding state transitions in Table 25 and Figure 14. Last, using the present states of flip-flop inputs and outputs, create the state transition diagram in Figure 14.

T Flip-Flop

The T flip-flop is a single-input version of the JK flip-flop: $Q(t+1) = J\,\overline{Q}(t) + \overline{K}\,Q(t)$, where T is analogous to J and T is analogous to $\overline{K}$, as demonstrated in the T flip-flop next state Boolean expression below. This flip-flop is typically used in the design of binary counters because counter operation requires complementation. The T flip-flop output toggles with each clock pulse, if T = 1, causing complementation of the input, as demonstrated in the following:

$$Q(t+1) = T\,\overline{Q(t)} + \overline{T}\,Q(t), \text{ for } T = 1, Q(t+1) = \overline{Q(t)}$$

Problem 38

Given: T-flip-flop timing diagram in Figure 15.

Design the circuit to implement the timing diagram sequence.

Solution: Based on the input, T, identify the values of the present states of flip-flop outputs in Table 26. Then, using the T flip-flop logic rules, compute the Boolean expressions for the next state flip-flop outputs in Table 26. Finally, apply the next states values to the design of the circuit in Figure 16, noting that state changes occur on the rising edge of the T pulse in Figure 15.

Ripple Counter

The external clock is only connected to the clock input of the first JK flip-flop in Figure 17. Therefore, the first flip-flop changes state at the falling edge of each clock

Table 24 Flip-Flop State Table (Based on Figure 13)

Present state of flip-flop outputs						Present state of flip-flop inputs				
$Q_1(t)$	$\overline{Q_1}(t)$	$Q_2(t)$	$\overline{Q_2}(t)$	$Q_3(t)$	$\overline{Q_3}(t)$	$S_1 = \overline{Q_2}(t)Q_3(t)$	$R_1 = Q_2(t)\overline{Q_3}(t)$	$J_2 = Q_1(t)\overline{Q_3}(t)$	$K_2 = Q_1(t) + Q_3(t)$	$D_3 = Q_2(t)\overline{Q_3(t)} + \overline{Q_2(t)}Q_3(t)$
0	1	**0**	1	**0**	1	0	0	1	0	0
0	1	**0**	1	**1**	0	1	0	0	1	1
0	1	**1**	0	**0**	1	0	1	1	0	1
0	1	**1**	0	**1**	0	0	0	0	1	0
1	0	**0**	1	**0**	1	0	0	1	1	0
1	0	**0**	1	**1**	0	1	0	1	1	1
1	0	**1**	0	**0**	1	0	1	1	1	1
1	0	**1**	0	**1**	0	0	0	1	1	0

Next State of Flip-Flop Outputs (Based on Flip-flop Rules for Next State)

$s_1 = \overline{Q_2}(t)Q_3(t)$	$\overline{R_1}Q_1(t) = Q_2(t)\overline{Q_3(t)}\overline{Q}(t)$	$Q_1(t+1) = S_1 + \overline{R_1}Q_1(t)$	$J_2 = Q_1(t) + \overline{Q_3}(t)$	$\overline{K2} = \overline{Q1\ (t)} + Q3\ (t)$	$Q_2(t+1) = J_2\overline{Q}_2(t) + \overline{K_2}Q(t)_2$	$Q_3(t+1) = D_3$
0	0	**0**	1	1	**1**	**0**
1	0	**1**	0	0	**0**	**1**
0	0	**0**	1	1	**1**	**1**
0	0	**0**	0	0	**0**	**0**
0	1	**1**	1	0	**1**	**0**
1	1	**1**	1	0	**1**	**1**
0	0	**0**	1	0	**0**	**1**
0	1	**1**	1	0	**0**	**0**

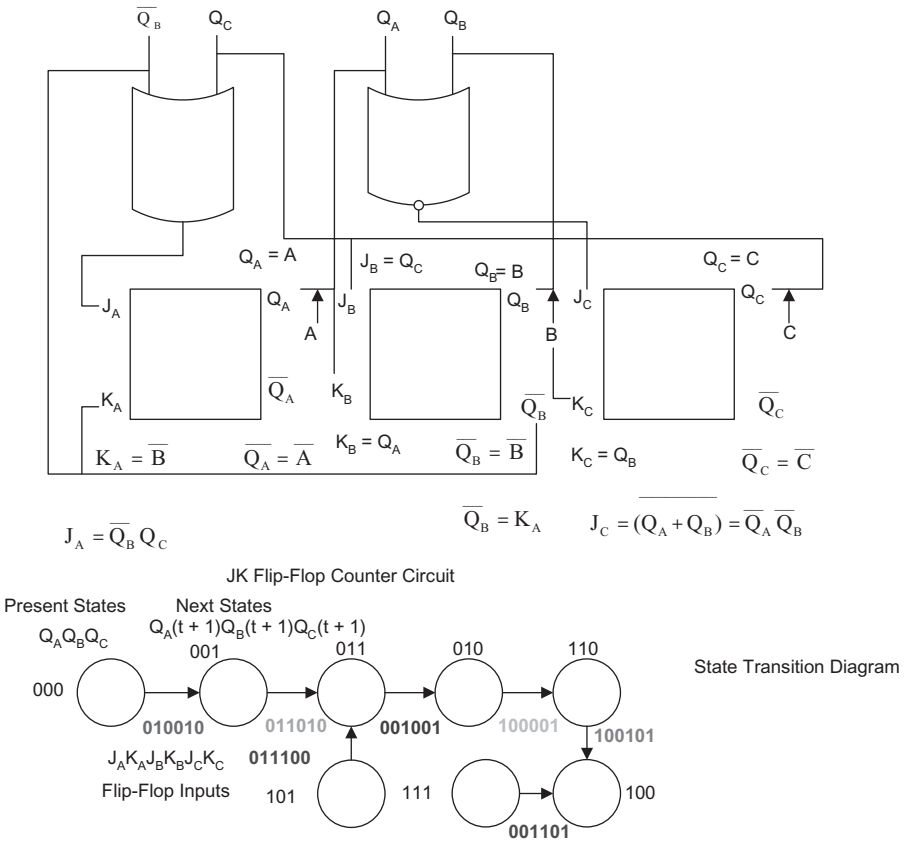


Figure 14 JK flip flop counter diagrams.

pulse, but the other flip-flops change only when triggered at the clock input by the $\overline{Q}(t)$ input from the preceding flip-flop in Figure 17. Because of the inherent propagation delay through a flip-flop, the transition of the input clock pulse and the transition of the $\overline{Q}(t)$ output of each flip-flop cannot occur at exactly the same time. Therefore, the flip-flops cannot be triggered simultaneously, thus producing an asynchronous operation, with the next stage JK flip-flop output generated according to $Q(t+1) = J\,\overline{Q}(t) + \overline{K}\,Q(t)$, where $J = 1$ and $\overline{K} = 0$ in Figure 17. Thus, $Q(t+1) = \overline{Q}(t)$.

Problem 39

Design a ripple counter that will count from 0000 (decimal 0) to 1111 (decimal 15) and cycle back to 0000.

Solution: Figure 17 shows the logic design and timing sequence. Table 27 tabulates the present and next flip-flop ripple counter states, where the most significant bit position is $Q_3(t+1) = \overline{Q}_3(t)$ and the least significant bit position is $Q_0(t+1) = \overline{Q}_0(t)$.

Table 25 JK Flip-Flop State Table

External inputs			Flip-flop inputs						Present states of flip-flop outputs		
A	B	C	$J_A = Q_B(t)\overline{Q_C}(t)$	$K_A = \overline{B}$	$J_B = Q_C(t)$	$K_B = Q_A(t)$	$J_C = \overline{Q_A}(t)\overline{Q_B}(t)$	$K_C = Q_B(t)$	$Q_A(t) = A$	$Q_B(t) = B$	$Q_C(t) = C$
0	0	0	**0**	**1**	**0**	**0**	**1**	**0**	**0**	**0**	**0**
0	0	1	**0**	**1**	**1**	**0**	**1**	**0**	**0**	**0**	**1**
0	1	0	**1**	**0**	**0**	**0**	**0**	**1**	**0**	**1**	**0**
0	1	1	**0**	**0**	**1**	**0**	**0**	**1**	**0**	**1**	**1**
1	0	0	**0**	**1**	**0**	**1**	**0**	**0**	**1**	**0**	**0**
1	0	1	**0**	**1**	**1**	**1**	**0**	**0**	**1**	**0**	**1**
1	1	0	**1**	**0**	**0**	**1**	**0**	**1**	**1**	**1**	**0**
1	1	1	**0**	**0**	**1**	**1**	**0**	**1**	**1**	**1**	**1**

Next states of flip-flop outputs

$J_A\overline{Q_A}(t)$	$\overline{K_A}Q_A(t)$	$Q_A(t+1)$	$J_B\overline{Q_B}(t)$	$\overline{K_B}Q_B(t)$	$Q_B(t+1)$	$J_C\overline{Q_C}(t)$	$\overline{K_C}Q_C(t)$	$Q_C(t+1)$
0	0	**0**	0	0	**0**	0	1	**1**
0	0	**0**	1	0	**1**	0	1	**1**
1	0	**1**	0	1	**1**	0	0	**0**
0	0	**0**	0	1	**1**	0	0	**0**
0	0	**0**	0	0	**0**	0	1	**1**
0	0	**0**	1	0	**1**	0	1	**1**
0	1	**1**	0	0	**0**	0	0	**0**
0	1	**1**	0	0	**0**	0	0	**0**

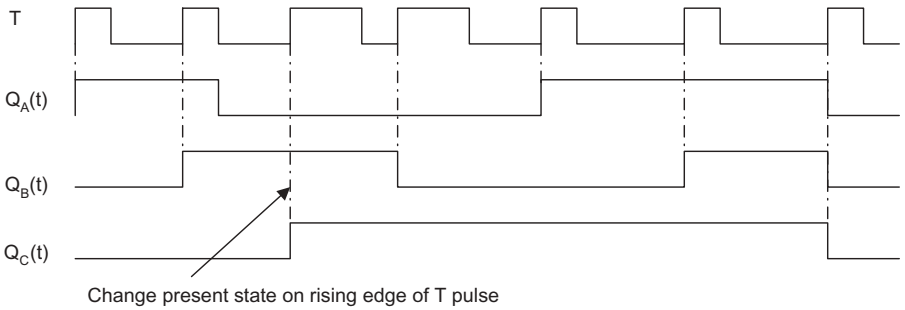


Figure 15 T flip-flop present state changes.

Table 26 T Flip-Flop State Table

Input	Present states of flip-flop outputs			Next states of flip-flop outputs		
T	$Q_A(t)$	$Q_B(t)$	$Q_C(t)$	$Q_A(t+1) = T\,\overline{Q_A(t)} + \overline{T}\,Q_A(t)$	$Q_B(t+1) = T\,\overline{Q_B(t)} + \overline{T}\,Q_B(t)$	$Q_C(t+1) = T\,\overline{Q_C(t)} + \overline{T}\,Q_C(t)$
1	1	0	0	0	1	1
0	1	0	0	1	0	0
1	1	1	0	0	0	1
0	1	1	0	1	1	0
1	0	1	1	1	0	0
0	0	1	1	0	1	1
1	0	0	1	1	1	0
0	0	0	1	0	0	1
1	1	0	1	0	1	0
0	1	0	1	1	0	1
1	1	1	1	0	0	0
0	1	1	1	1	1	1
1	0	0	0	1	1	1
0	0	0	0	0	0	0

The design uses the next state relationship: $Q(t+1) = J\,\overline{Q}(t) + \overline{K}\,Q(t) = \overline{Q(t)}$

Problem 40

Given: Digital circuit in Figure 18.

Produce the state transition diagram corresponding to this circuit.

Solution: First, using the Boolean expression for JK, D, and RS flip-flops, identify the relationships between $Q_3(t + 1)$ and JK, between $Q_2(t + 1)$ and D, and between $Q_1(t + 1)$ and RS.

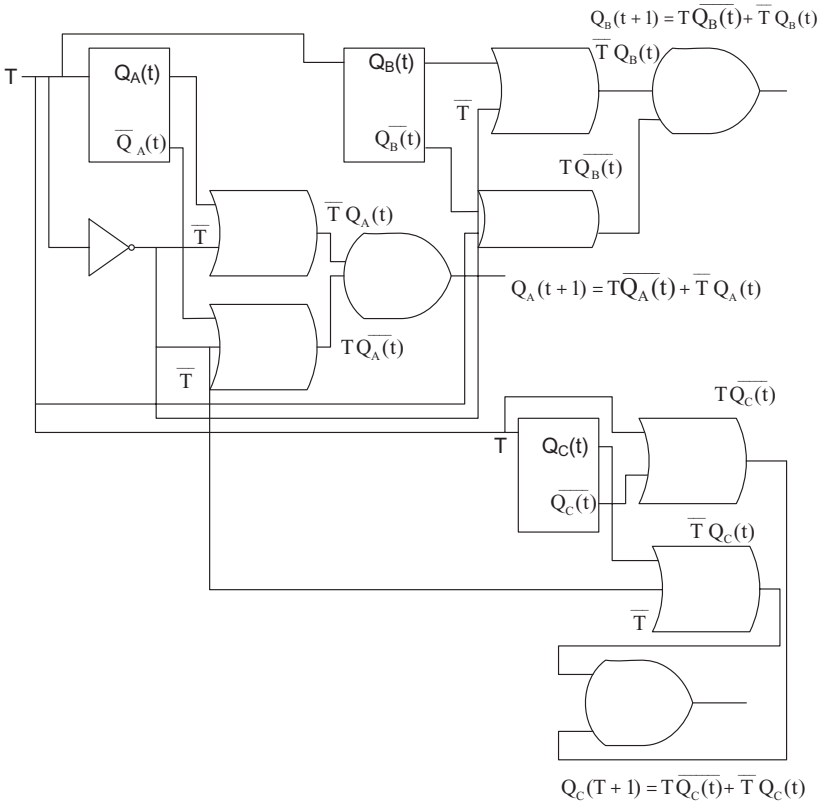


Figure 16 T flip-flop logic diagram.

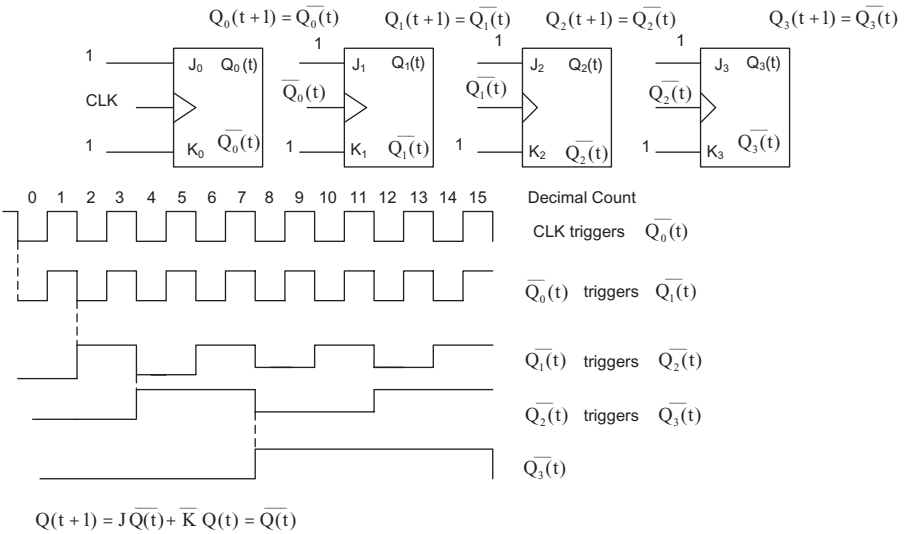


Figure 17 Ripple counter design and timing diagram.

Table 27 Ripple Counter State Table

	Present flip-flop states								Next flip-flop states			
Decimal count	$Q_0(t)$	$\overline{Q_0(t)}$	$Q_1(t)$	$\overline{Q_1(t)}$	$Q_2(t)$	$\overline{Q_2(t)}$	$Q_3(t)$	$\overline{Q_3(t)}$	$Q_0(t+1)=\overline{Q_0(t)}$	$Q_1(t+1)=\overline{Q_1(t)}$	$Q_2(t+1)=\overline{Q_2(t)}$	$Q_3(t+1)=\overline{Q_3(t)}$
0	1	0	1	0	1	0	1	0	0	0	0	0
1	0	1	1	0	1	0	1	0	1	0	0	0
2	1	0	0	1	1	0	1	0	0	1	0	0
3	0	1	0	1	1	0	1	0	1	1	0	0
4	1	0	1	0	0	1	1	0	0	0	1	0
5	0	1	1	0	0	1	1	0	1	0	1	0
6	1	0	0	1	0	1	1	0	0	1	1	0
7	0	1	0	1	0	1	1	0	1	1	1	0
8	1	0	1	0	1	0	0	1	0	0	0	1
9	0	1	1	0	1	0	0	1	1	0	0	1
10	1	0	0	1	1	0	0	1	0	1	0	1
11	0	1	0	1	1	0	0	1	1	1	0	1
12	1	0	1	0	0	1	0	1	0	0	1	1
13	0	1	1	0	0	1	0	1	1	0	1	1
14	1	0	0	1	0	1	0	1	0	1	1	1
15	0	1	0	1	0	1	0	1	1	1	1	1

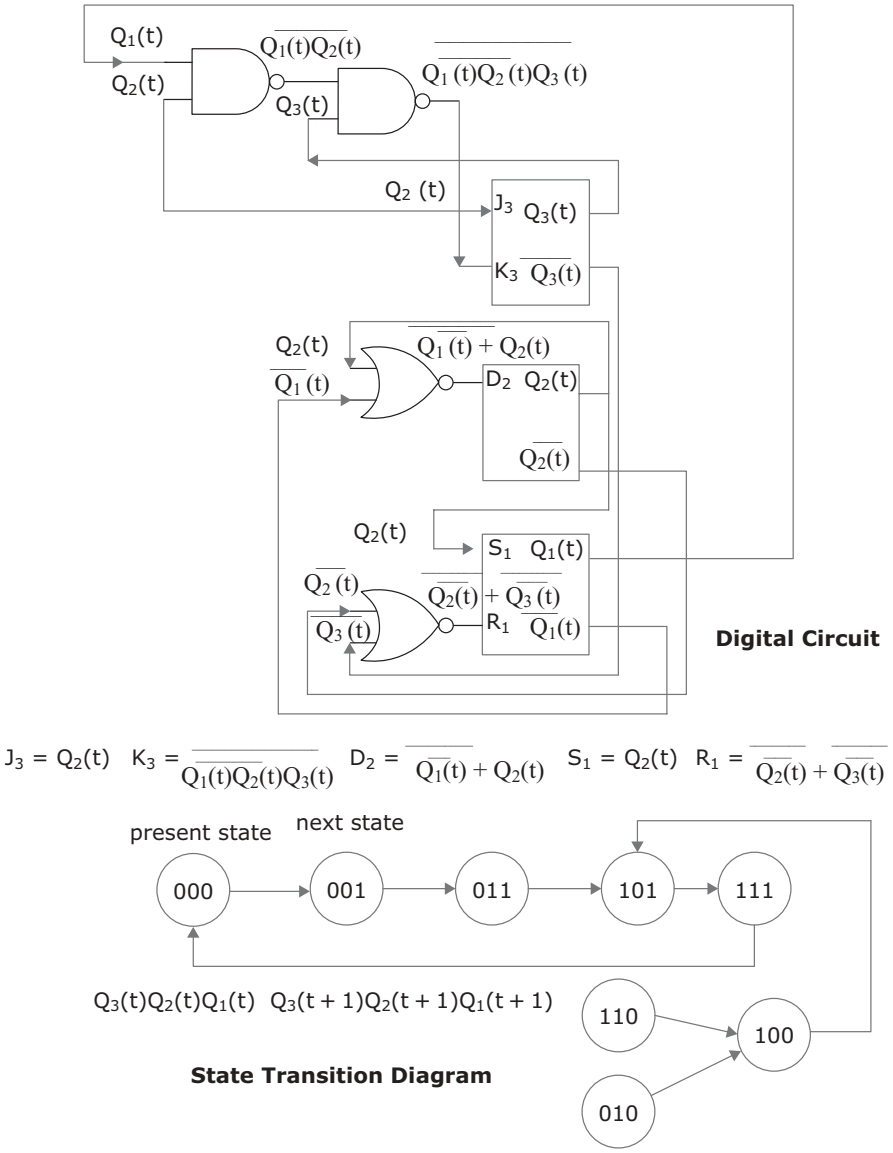


Figure 18 Digital circuit and state transition diagram.

$$Q_3(t+1) = J_3\ \overline{Q}_3(t) + \overline{K_3}\ Q_3(t) = Q_2(t)\overline{Q}_3(t) + ((\overline{Q_1(t)Q_2(t)})Q_3(t))Q_3(t)$$

$$Q_3(t+1) = Q_2(t)\overline{Q_3(t)} + ((\overline{Q_1(t)Q_2(t)})Q_3(t))$$

$$Q_2(t+1) = D_2 = Q_1(t)\overline{Q_2(t)}$$

$$Q_1(t+1) = S_1 + \overline{R_1}Q_1(t) = \overline{Q_2}(t) + (\overline{Q2(t)Q3(t)})(Q_1(t))$$

Table 28 Flip-Flop State Table

Present flip-flop output states			Next flip-flop output states		
$Q_3(t)$	$Q_2(t)$	$Q_1(t)$	$Q_3(t+1) = Q_2(t)\overline{Q}_3(t) + ((\overline{Q_1(t)}Q_2(t))Q_3(t))$	$Q_2(t+1) = Q_1(t)\overline{Q_2(t)}$	$Q_1(t+1) = \overline{Q_2}(t) + (Q_2(t)Q_3(t))(Q_1(t))$
0	0	0	0	0	1
0	0	1	0	1	1
0	1	0	1	0	0
0	1	1	1	0	1
1	0	0	1	0	1
1	0	1	1	1	1
1	1	0	1	0	0
1	1	1	0	0	0

Document these relationships in the state table, Table 28, and identify the present state–next state transitions. Then, use the state transitions in Table 28 to design the state transition diagram in Figure 18. For example, present state 000 leads to next state 001 in Table 28 and Figure 18.

Problem 41

Given: The circuit diagram of a JK flip-flop counter is shown in Figure 19. Note that this circuit does not count sequentially. It could be used, for example, in elevator control, where the states would represent the sequence of floors traversed by the elevator. Provide the state transition and timing diagrams corresponding to the circuit design.

Solution: Using the JK flip-flop Boolean expression rules below, solve for the Boolean expressions of flip-flop inputs, present flip-flop output states, and next flip-flop output states in Table 29. Use the results in Table 29 to design the state transition and timing diagrams in Figure 19.

$$K_1 = \overline{J_3\overline{Q_3(t)}}$$

$$J1 = Q3(t)$$

$$K2 = 1$$

$$J2 = 1$$

$$K3 = 1$$

$$J_3 = Q_1\overline{(t)Q_2}(t)$$

$$Q_1(t+1) = J_1\overline{Q_1}(t) + \overline{K_1}Q_1(t) = Q_3(t)\overline{Q_1}(t) + J_3\overline{Q_3(t)}Q_1(t)$$

$$Q_2(t+1) = J_2\overline{Q_2}(t) + \overline{K_2}Q_2(t) = (1)\overline{Q_2}(t) + (0)Q_2(t) = \overline{Q_2}(t)$$

$$Q_3(t+1) = J_3\overline{Q_3}(t) + \overline{K_3}Q_3(t) = Q_1\overline{(t)Q_2}(t)\overline{Q_3}(t) + (0)Q_3(t) = Q_1\overline{(t)Q_2}(t)\overline{Q_3}(t)$$

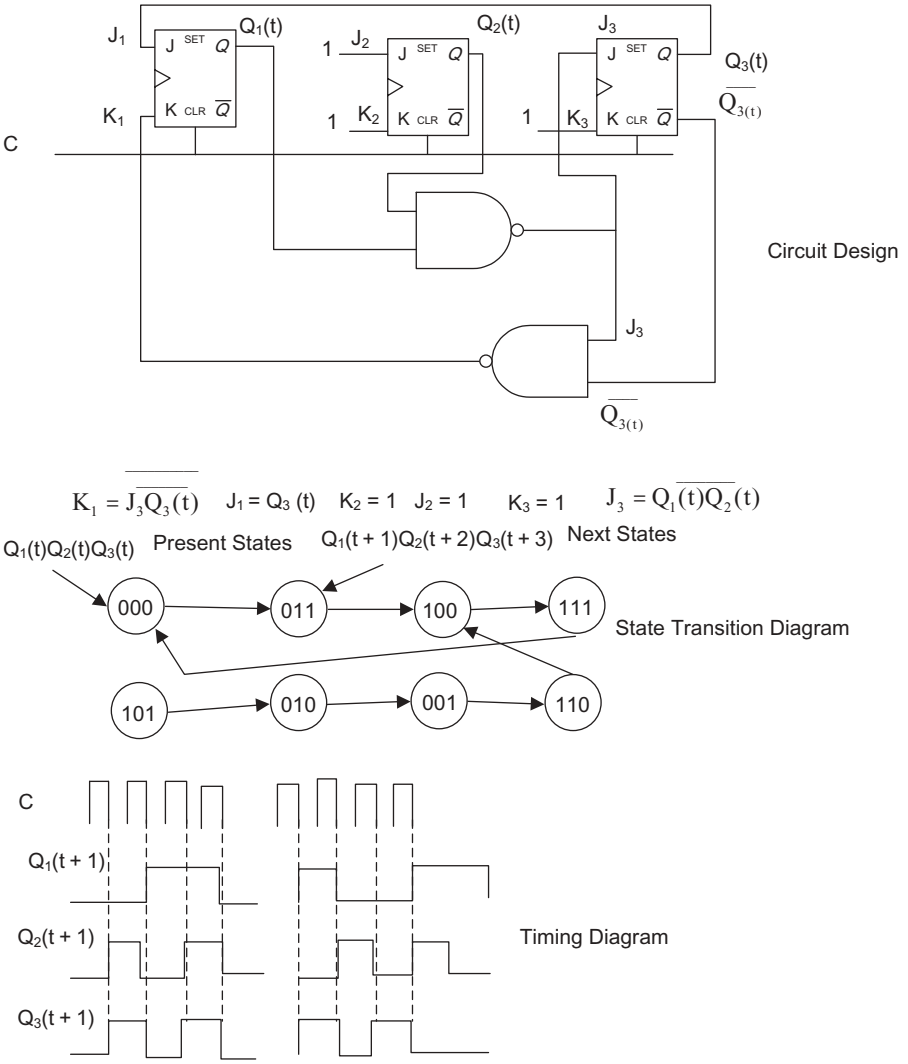


Figure 19 Counter circuit design, state transition diagram, and timing diagram.

Problem 42

Design a D flip-flop circuit that has the input sequences X shown in Figure 20, where the clock pulse CLK triggers each of the inputs X to enter the circuit.

Solution: First, the flip-flop state table, Table 30, is developed, where the D flip-flop next states follow the X inputs. Second, Karnaugh maps are used in Tables 31 and 32 in an *attempt* to simplify the expressions for A_0 and A_1, respectively. It can be seen that no simplification results from employing the Karnaugh map. A Karnaugh map is not necessary for A_2 because there is

Table 29 Counter State Table

Flip-flop inputs					Flip-flop outputs present states			Flip-flop outputs next states		
$K_1 = \overline{J_3 Q_3(t)}$	J1 = Q3(t)	K2 = 1	J2 = 1	$J_3 = Q_1(t)\overline{Q_2}(t)$	$Q_1(t)$	$Q_2(t)$	$Q_3(t)$	$Q_1(t+1) = Q_3(t)\overline{Q_1}(t) + J_3\overline{Q_3(t)}Q_1(t)$	$Q_2(t+1) = \overline{Q_2}(t)$	$Q_3(t+1) = Q_1(t)\overline{Q_2}(t)\overline{Q_3}(t)$
0	0	1	1	1	0	0	0	0	1	1
1	1	1	1	1	0	0	1	1	1	0
0	0	1	1	1	0	1	0	0	0	1
1	1	1	1	1	0	1	1	1	0	0
0	0	1	1	1	1	0	0	1	1	1
1	1	1	1	1	1	0	1	0	1	0
1	0	1	1	0	1	1	0	1	0	0
1	1	1	1	0	1	1	1	0	0	0

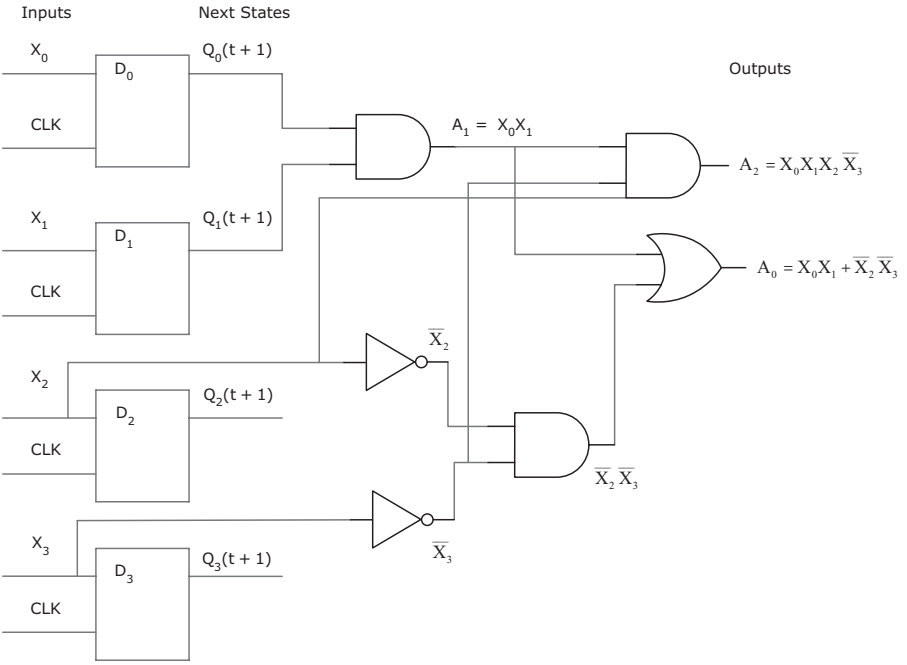


Figure 20 Digital flip-flop circuit.

Table 30 Flip-Flop States Table

Inputs				Outputs			Flip-flop next states			
X_0	X_1	X_2	X_3	$A_0 = X_0X_1 + X_2X_3$	$A_1 = X_0X_1$	$A_2 = X_0X_1X_2\overline{X_3}$	$Q_0(t+1) = X_0$	$Q_1(t+1) = X_1$	$Q_2(t+1) = X_2$	$Q_3(t+1) = X_3$
1	0	0	0	1	0	0	1	0	0	0
1	1	0	0	1	1	0	1	1	0	0
1	1	1	0	1	1	1	1	1	1	0

only one cell in Table 30 that has the value "1". This is the cell corresponding to $A_2 = X_0X_1X_2\overline{X_3}$. Figure 20 shows the circuit design and the relevant Boolean expressions.

Problem 43

Design a circuit to control shaft 90° rotations, using two inputs, R and C, and clock pulse. The inputs and clock pulse are applied to D flip-flops that cause state changes, each state change representing a 90° rotation, as shown in Figure 21.

Solution: First, depict the shaft rotation positions and corresponding state transitions in Figure 21. Second, in Table 33, document the inputs and present states that cause the next state transitions. Finally, using Table 33, design the digital logic circuit in Figure 21.

Table 31 Karnaugh Map for $A_0 = \overline{X_2}\,\overline{X_3} + X_0X_1$

X_0X_1	X_2X_3			
00	00	01	11	10
01				
11	1			1
10	1			

$\overline{X_2}\,\overline{X_3}$ X_0X_1

Table 32 Karnaugh Map for $A_1 = X_0X_1$

X_0X_1	X_2X_3			
00	00	01	11	10
01				
11	1			1
10				

X_0X_1

Problem 44

Given the flow chart in Figure 22, design the corresponding digital circuit.

Solution: Figure 22 shows the digital circuit design.

CHAPTER 9 (PROGRAMMING LANGUAGES) AND CHAPTER 10 (OPERATING SYSTEMS)

Programming Languages

Problem 45

Given: J = 15, K = 4, L = 9, and M = 19

Problem: What is the value of *logical* variable X in (a) and (b) below?

Solution:

(a) X = J. LT .L. OR K.GE. (M – J): (15 < 9) *OR* $4 \geq (19 - 15)$, X = true because $4 \geq (19 - 15)$

(b) X = J. GT. L. OR K. LT. (M – J): $(15 > 9)$ *OR* $4 < (19 - 15)$, X = true because $15 > 9$

Problem 46

What is the value of Q in the program below?

Solution: Based on the evaluation of the IF statement, goto line 10: Q = 10

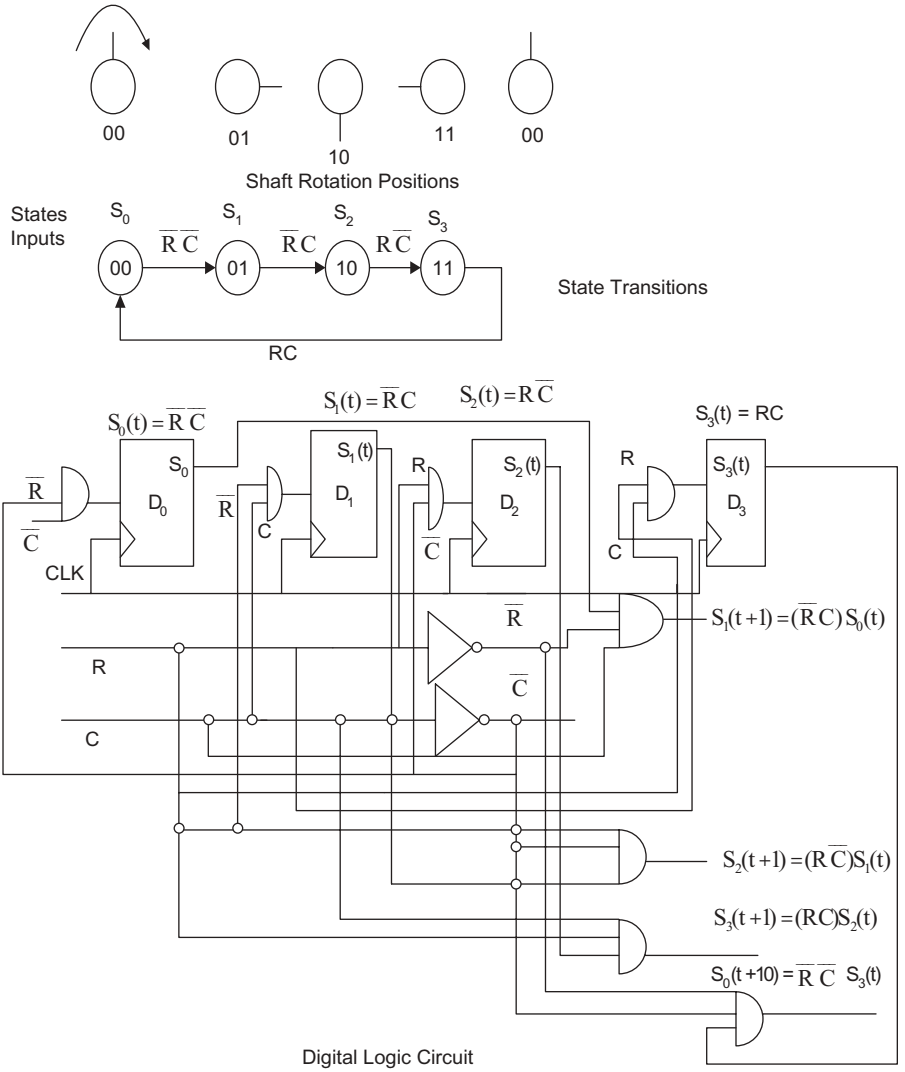


Figure 21 Shaft rotation diagrams.

Table 33 Shaft Rotation State Transitions

Input	Shaft rotation	Present state	Next state
$\overline{R}\ \overline{C} = 00$	90°	$S_0(t) = \overline{R}\ \overline{C}$	$S_1(t+1) = \overline{R}\ C S_0(t)$
$\overline{R}\ C = 01$	180°	$S_1(t) = \overline{R}\ C$	$S_2(t+1) = R\ \overline{C} S_1(t)$
$R\ \overline{C} = 10$	270°	$S_2(t) = R\ \overline{C}$	$S_3(t) = RCS_0(t)$
RC = 11	360°	$S_3(t) = RC$	$S_0(t+1) = \overline{R}\ \overline{C} S_3(t)$

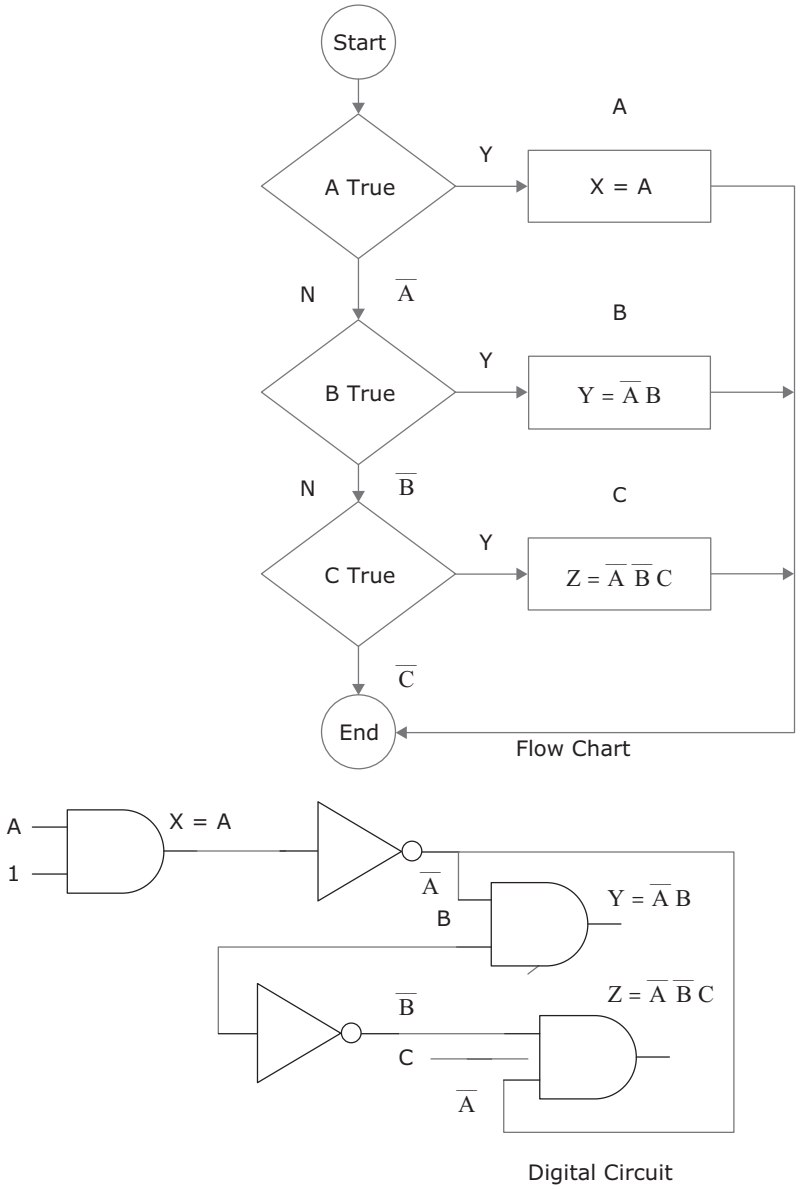


Figure 22 Flow chart and digital circuit.

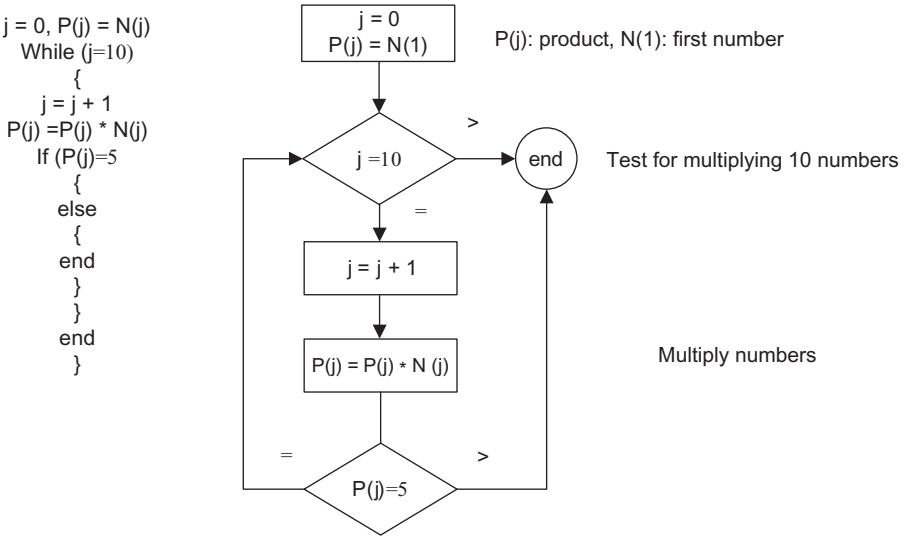


Figure 23 Flow chart and program for multiplying numbers.

R = 18

S = 6

T = 2

$Q = R/(S^T) - T$, $Q = (18/(6^2)) - 2 = -1.5$

IF Q (10, 20, 30) // IF Q < 0, GOTO 10; IF Q = 0, GOTO 20; IF Q > 0, GOTO 30

// Q < 0, therefore, goto 10

10 Q = 10 // Q changed from – 1.5 to 10

Problem 47

Develop a flow chart and write the corresponding program to multiply 10 numbers, where the product is not to exceed 5.

Solution: The flow chart and program are shown in Figure 23.

Floating-Point Format

$n = (f)*(b)^e$, where n = non-floating-point number, f = fraction, b = base = 2 for binary computations, and e = exponent.

Problem 48

Find the values of f, b, and e corresponding to the number n = 4104.

Solution: First, for ease of number handling, convert to base 16 by successively dividing by 16 and recording the remainders in reverse order, and then to base 2 by recording the base 16 numbers in base 2 format, as follows:

$$4104_{10} = 1008_{16} = 0001\ 0000\ 0000\ 1000_2$$

Second, find the exponent and fraction that will result in a normalized mantissa (i.e., the most significant bit is a one).

Third, using n = (f)*(b)^e, where n = the number 4104_{10}, f = normalized mantissa, b = base 2, and e = exponent, find f and e that will result in a normalized mantissa:

$$f = (4104)/(2^e) = (2^{12} = 4096 + 2^3 = 8))/(2^e) = 2^{(12-e)} + 2^{(3-e)} = (2^{(3-e)})(2^9 + 1)$$
$$= (2^{(3-e)})(513)$$

The value of f that satisfies this relationship is $+0.5010_{10}$, by trial and error, and the corresponding value of e is $+13_{10}$: $(2^{(3-13)})(513) = 0.0009766*513 = 0.5010_{10}$ The trial-and-error process involves trying values of e that will result in the first value of $(2^{(3-e)})$ that will generate a normalized mantissa when multiplied by 513.

These values are converted to binary by dividing 0.5010_{10} successively by 2 and recording the remainders.

Proof of correct conversion: $0.5010_{10}*2^{13} = 0.5010_{10}*8192 = 4104$.

C++ Programming

Problem 49

Write a function in C++ to compute $\sin\theta = \sqrt{\omega t}$

Solution:

```
void computesine (double); // function prototype
main()
{
double computesine (double w, double t, double
angle); // function call
{
double result, w, t, angle;// function definition
angle = w * t;// compute angle
result = sin (pow (angle, .5));// compute sine of
square root of angle
return result;// return the result to caller
}// match with function call
return 0;// return to the operating system
} // executable code ends here, match with main
```

Problem 50

Write a function in C++ to convert inches to feet
Solution:

```
void convert (double); // function prototype
main()
{
double convert (double inches, double feet); //
function call
{
double inches, feet;// function definition
feet = inches/ 12;// convert inches to feet
return feet;// return the result to caller
}// match with function call
return 0;// return to the operating system
} // executable code ends here, match with main
```

Problem 51

Write a function in C++ to sum numbers in a list that are greater than or equal to 10.

Solution: See the flowchart in Figure 24 and the C++ code below for implementation of flowchart logic.

```
void sum (double);// function prototype
main ()
{
double sum (double X (i), double S (i), int n, int
i) //function call, define ith number X (i), number
sum S (i), number of numbers n, number index i
{
double result, X (i), S (i); // function definition
int n, i;
S (i) = 0;// initialize sum
While (i < = n)
{
cout << "input number =";// tell user to input ith
number
cin >> x (i);
```

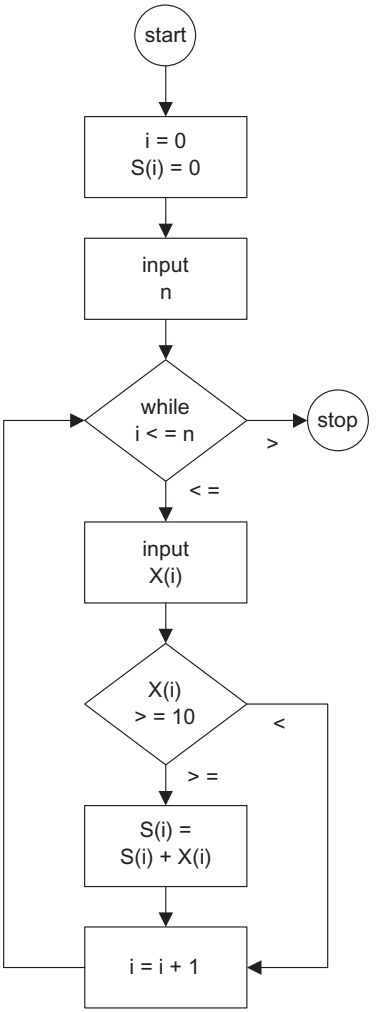


Figure 24 Sum numbers flowchart.

```
if (X (i) ≥ 10)
{
result = S (i) + X (i); // sum number, if number ≥ 10
}// match with "if"
i = i + 1;
}/ /match with function call
return result; // return the result to caller
```

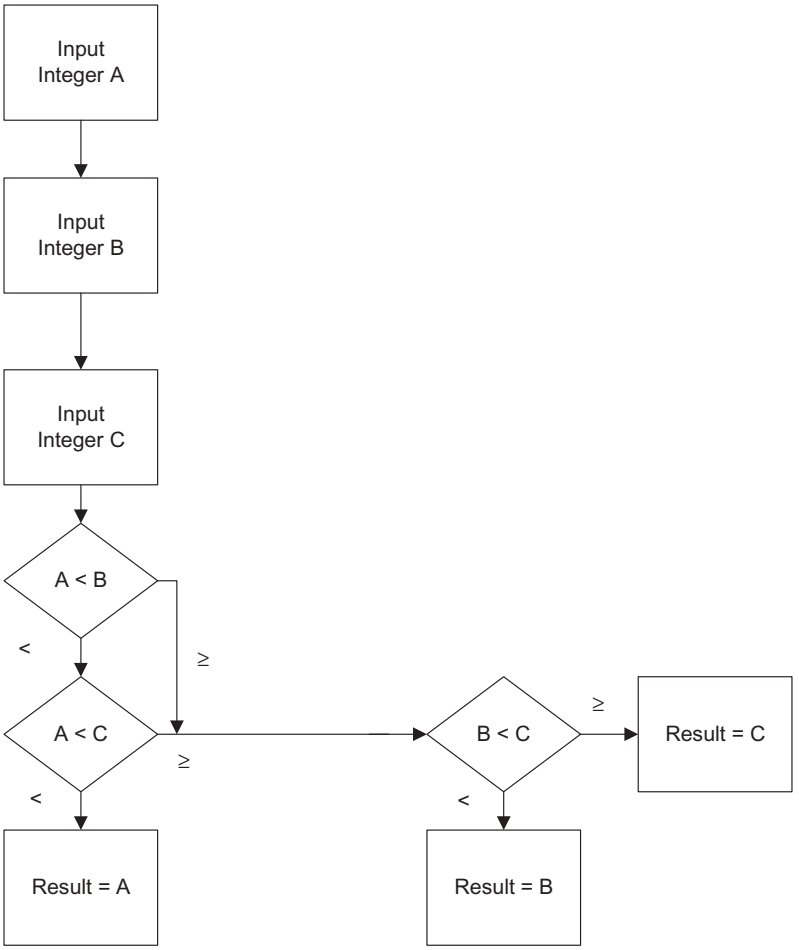


Figure 25 Flowchart of minimum of three numbers logic.

```
return 0;// return to the operating system
} // executable code ends here, match with main
```

Problem 52

Write a C++ function to find the minimum of three integers A, B, and C.

Solution: The implementation of the logic is the following and Figure 25 shows the flowchart of the logic:

```
void compare (int);// function prototype
main ()
```

```
{
int compare (int A, int B, int C) //function call,
define integers A, B, and C
{
int result, A, B, C; // function definition
cout << "input A =";// tell user to input integer A
cin >> A;
cout << "input B =";// tell user to input integer B
cin >> B;
cout << "input C = ";// tell user to input integer C
cin >> C;
if ((A < B) && (A < C)
{
result = A;//integer A is smallest
}
if ((B < A) && (B < C)
{
result = B;//integer B is smallest
}
else
{
result = C;//integer B is smallest
}
}/ /match with function call
return result; // return the result to caller
return 0;// return to the operating system
} // executable code ends here, match with main
```

Problem 53

Write a C++ function to compute spring force F, given inputs of spring constant K and distance X,

Solution: The function is the following:

```
void springforce (double);// function prototype
main ()
{
```

```
double springforce (double F, double K, double X) //
function call, define Force, F, Spring Constant, K,
and Distance, X

{

double F, K, X; // function definition

cout << "input K =";// tell user to input Spring
Constant, K,

cin >> K;

cout << "input X =";// tell user to input Distance, X

cin >> X;

F = K*X;//compute spring force

}/ /match with function call

return F; // return the result to caller

return 0;// return to the operating system

} // executable code ends here, match with main
```

Problem 54

If a = 10, what is the value of the C++ operation ++a?

Solution: The operator "++" means incrimination. Therefore, ++10 = 11.

CHAPTER 4: ANALOG AND DIGITAL COMPUTER INTERACTIONS

Elements and Integration

Analog computer elements are shown in Figure 26.

Problem 55

Using the elements in Figure 26, mechanize the integration of the differential equations shown below.

Solution: See differential equations implementations in Figure 26.

Simulation

Time Scaling

Speed up or slow down a simulation compared with real time. Scale speed up = ht, slow down = t/h, where t is time and h is scale factor.

Problem 56

Using the differential equation below, slow it down.

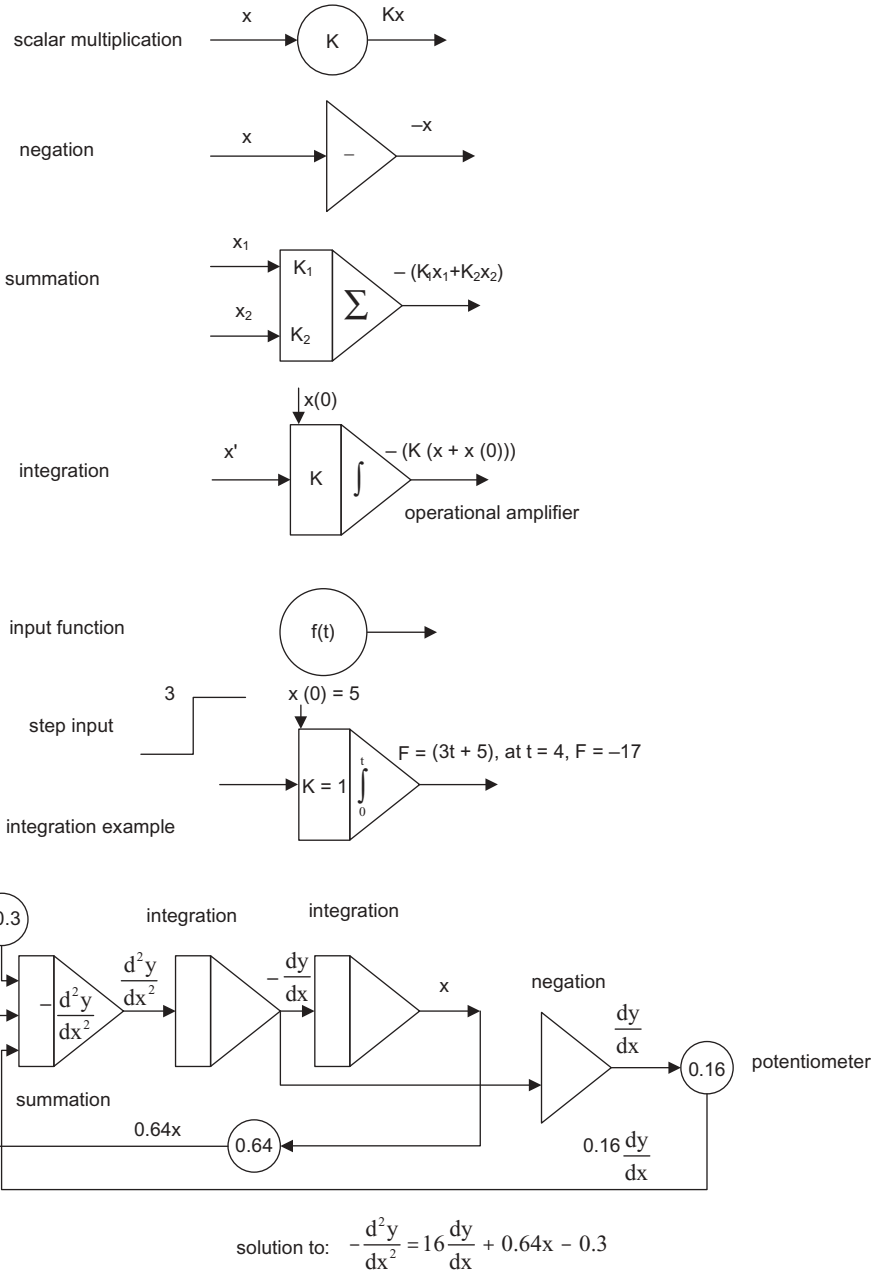


Figure 26 Analog computer elements and integration examples.

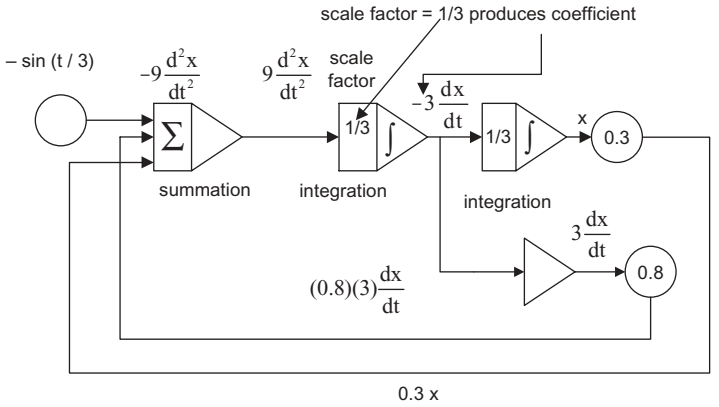


Figure 27 Applying slow-down scale factor.

Solution: See below for how the scale factor slows down the simulation and the analog computer implementation in Figure 27.

$$-\frac{d^2x}{dt^2} = 0.8\frac{dx}{dt} + 0.3x - \sin t(\text{original_equation})$$

$$-\frac{d^2x}{d\frac{t^2}{h^2}} = 0.8\frac{dx}{d\frac{t}{h}} + 0.3x - \sin\frac{t}{h}(\text{equation_with_slow_down_scale_factor_applied})$$

$$-9\frac{d^2x}{dt^2} = (3*0.8)\frac{dx}{dt} + 0.3x - \sin\frac{t}{3}(\text{slow_down_by_factor}, h = 3)$$

Magnitude Scaling

Because the operational amplifiers in an analog computer have a limited voltage range, it may be necessary to use magnitude scaling of differential equations. Let the expected maximum value of a term in the differential equation be M, and the desired maximum value, due to the limited voltage range, be N. Then terms are multiplied by (N/M).

Problem 57

Scale the original differential equation below and show the analog computer implementation in Figure 28.

Solution: See the magnitude scaling below and the analog computer implementation in Figure 28.

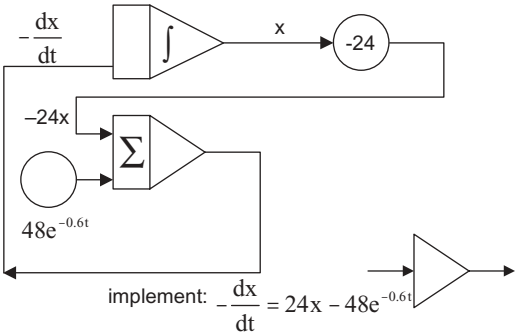


Figure 28 Magnitude scaling.

$$-\frac{dx}{dt} = 15x - (40e^{-0.6t}) _ \text{original_equation}$$

$$\left(-\frac{dx}{dt}\right)\left(\frac{N_1}{M_1}\right) = (15x)\left(\frac{N_0}{M_0}\right) - (40e^{-0.6t})\left(\frac{N_f}{M_f}\right)$$

$$_ \text{equation_with_magnitude_scaling_applied}$$

$$\left(-\frac{dx}{dt}\right)\left(\frac{10}{400}\right) = (15x)\left(\frac{20}{500}\right) - (40e^{-0.6t})\left(\frac{30}{1000}\right)$$

$$_ \text{equation_with_numerical_scaling-applied_} =$$

$$-0.025\frac{dx}{dt} = 0.6x - 1.2e^{-0.6t}, -\frac{dx}{dt} = 24x - 48e^{-0.6t}$$

Network of Analog Computer Components

Simultaneous differential equations can be solved using a network of analog computer components.

Problem 58

Using an analog computer network, solve the following equations:

$$-\frac{dx_1}{dt_1} = 0.8x_1 - 0.7x_2 - 0.3$$

$$-\frac{dx_2}{dt_2} = x_2 - 0.4x_1$$

Solution: The mechanization of the equations in the analog computer network is shown in Figure 29.

Problem 59

Draw the analog computer circuit for the following equation:

$$V_0 = -\int (V_1 + 4V_2)dt$$

Solution: The equation implementation is shown in Figure 30.

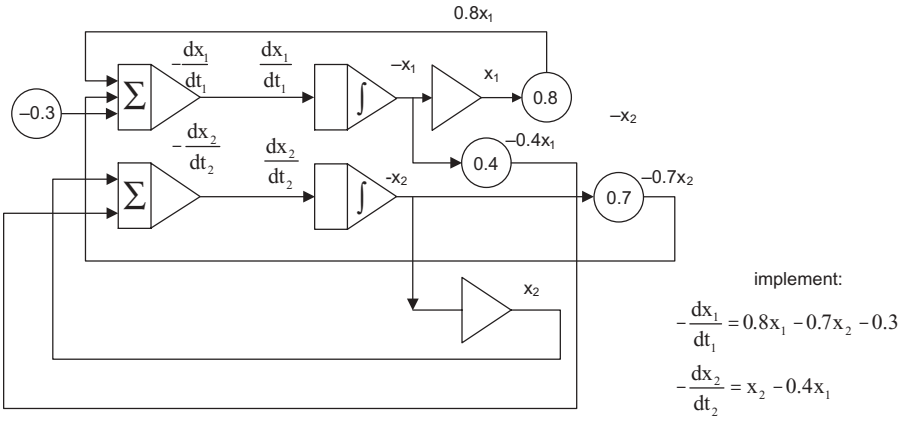


Figure 29 Simultaneous differential equations.

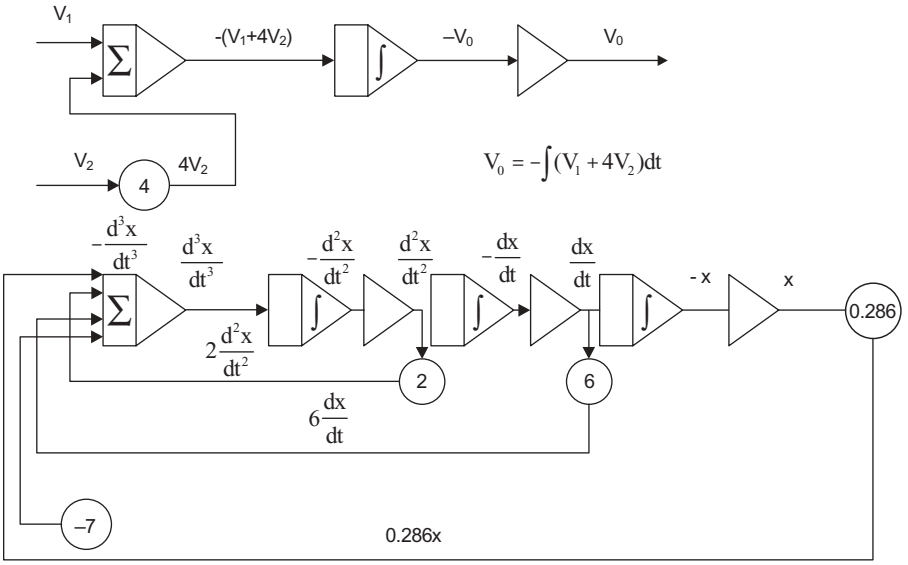


Figure 30 Implement $V_0 = -\int (V_1 + 4V_2)dt$.

Problem 60

Draw the analog computer circuit for the following standard equation format:

$$7\frac{d^3x}{dt^3}+14\frac{d^2x}{dt^2}+42\frac{dx}{dt}+2x=49_original_equation$$

$$\frac{d^3x}{dt^3}+2\frac{d^2x}{dt^2}+6\frac{dx}{dt}+0.286x=7_reduced_equation$$

$$-\frac{d^3x}{dt^3}=2\frac{d^2x}{dt^2}+6\frac{dx}{dt}+0.286x-7_standard_equation_format$$

Solution: the equation implementation is shown in Figure 31.

Problem 61

Given: The mechanical spring system shown in Figure 32.

$$-\frac{d^3x}{dt^3}=2\frac{d^2x}{dt^2}+6\frac{dx}{dt}+0.286x-7$$

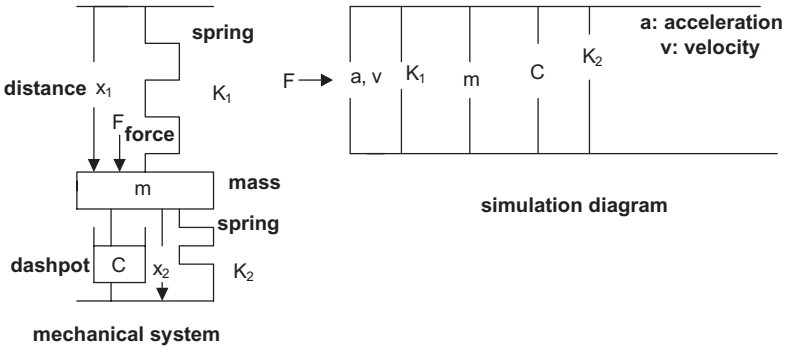


Figure 31 Implement $-\frac{d^3x}{dt^3}=2\frac{d^2x}{dt^2}+6\frac{dx}{dt}+0.286x-7.$

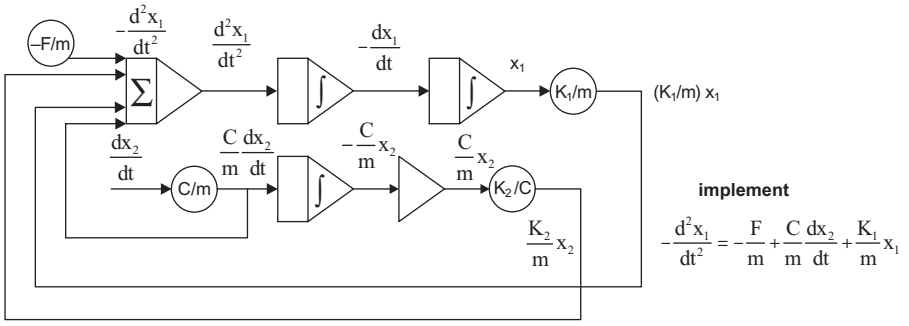


Figure 32 Spring mass circuit.

Develop the simulation diagram and, from this diagram, formulate the differential equations. Last, diagram the analog computer implementation.

Solution: Figure 32 shows the simulation diagram. Then, the differential equation is developed as follows, using a for acceleration, and noting that dashpot force is proportional to force F velocity v and spring force is proportional to the spring coefficients K_1 and K_2.

$$F = ma + Cv + K_1x_1 + K_2x_2 \quad \text{(basic equation)}$$

$$F = m\frac{d^2x_1}{dt^2} + C\frac{dx_2}{dt} + K_1x_1 + K_2x_2_\text{equation_with_derivatives}$$

$$-\frac{d^2x_1}{dt^2} = -\frac{F}{m} + \frac{C}{m}\frac{dx_2}{dt} + \frac{K_1}{m}x_1 + \frac{K_2}{m}x_2_\text{final_format}$$

Digital-to-Analog Conversion (DAC)

In a digital-to-analog converter, the bits—from the most significant bit to the least significant bit—are implemented with digital logic gates. The digital bits are weighted according to their contribution to the output voltage V_{out}. Depending on which bits are set to 1 and which are set to 0, the output voltage, V_{out}, will be a stepped value between 0 V and *V_{ref} minus the value of the minimum step* volts. For a digital value VAL of N bits, $V_{out} = ((V_{ref}/2^N)*VAL)$, where for a typical CMOS logic voltage, $V_{ref} = 3.3$ V.

Problem 62

What is the value of V_{out} for a Minimum VAL output of 1 bit?

Solution:

$$V_{out} = (3.3\text{ V}/2^5\text{ bits})*1\text{ bit} = 0.10\text{ V}$$

Problem 63

What is the value of V_{out} for a Maximum VAL output of 31 bits?

Solution:

$$V_{out} = (3.3\text{ V}/2^5\text{ bits})*31\text{ bits} = 3.2\text{ V}$$

Problem 64

Problem: The *voltage per step* of a digital-to-analog converter, with a voltage range from −5 to +5 volts, for a 4-bit output is:

Solution: $(10\text{ V}/2^4\text{ steps}) = 0.625$ V per step

Analog-to-Digital (A/D) Conversion

Sample-and-Hold Circuit

A sample-and-hold circuit is used to avoid having the input change while analog to digital conversion is taking place.

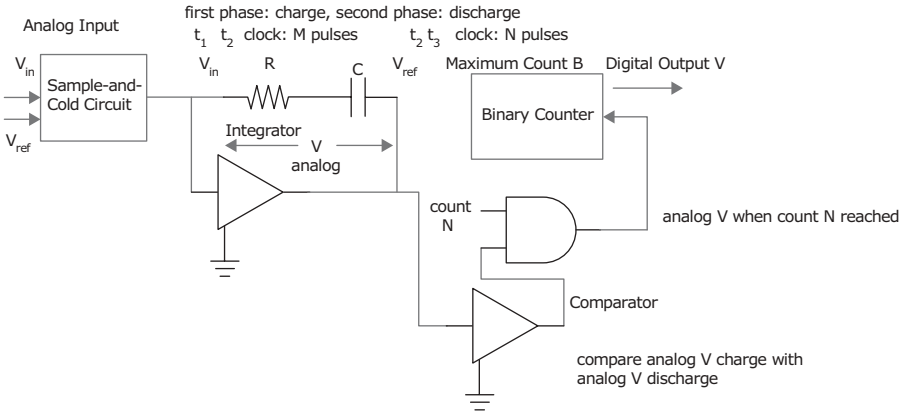


Figure 33 Analog-to-digital converter.

Conversion Process

The capacitor C in Figure 33 assists in the conversion of analog input to digital output by the duration of its charge. This is accomplished by measuring the time it takes to charge and discharge the capacitor into the resistor R. The larger the value of C, for a given value of R, the longer it takes to charge and discharge the capacitor, and, hence, the slower the rise and fall in voltage, respectively. Conversely, the smaller the value of C, the less time it takes to charge and discharge the capacitor, and, hence, the faster the rise and fall in voltage, respectively. The product CR is known as the time constant of a circuit.

The converter integrates the constant, positive analog input signal V_{in} in Figure 33 in the first phase, t_1, t_2, and integrates the constant, negative reference voltage V_{ref} in the second phase, t_2, t_3. At the end of phase 1, C has been charged by V_{in} to generate the voltage V across the RC circuit, given by:

$$V = \frac{1}{CR}\int_{t_1}^{t_2} V_{in}\,dt.$$

Similarly, at the end of phase 2, C has been discharged by V_{ref} to generate the output voltage V across the RC circuit, given by:

$$V = \frac{1}{CR}\int_{t_2}^{t_3} V_{ref}\,dt.$$

The voltage rise in phase 1 is equal to the voltage fall in phase 2. Thus, we obtain the following:

$$\frac{1}{CR}\int_{t_1}^{t_2} V_{in}\,dt = \frac{1}{CR}\int_{t_2}^{t_3} V_{ref}\,dt.$$

Since V_{in} and V_{ref} are constants, we obtain:

$$V = \frac{V_{in}}{CR}(t_2 - t_1) = \frac{V_{ref}}{CR}(t_3 - t_2).$$

The time period of charge in phase 1 is determined by the clock in Figure 33 that produces M pulses with a pulse period = T. Thus, $t_2 = MT$. Now, if N is the number of clock pulses in phase 2, $t_3 = NT$. Therefore, assuming $t_1 = 0$, the resultant computation is:

$$\frac{V_{in}}{CR}MT = \frac{V_{ref}}{CR}(NT - MT), \frac{V_{in}}{V_{ref}} = \left(\frac{N}{M} - 1\right).$$

The accuracy of conversion in phase 2 = (output voltage V divided by number of binary bits B produced in the conversion), as governed by the number of clock pulses N, during the allowable conversion time. Thus, accuracy = V/B. B is chosen to be the binary number less than N during the allowable conversion time. This value of B is equal to the maximum binary counter count in Figure 33.

Problem 65

Given: In Figure 33, the clock rate in phase 2 = $1/(t_3 - t_2)$ = 3 MHz, reference voltage, V_{ref} = 5 V, and output voltage V = 10 V. The conversion process must be completed in less than 1 ms.

What is the value of CR? Is it possible to complete the conversion in less than 1 ms?

Solution:

$$V = (V_{ref}/CR)(t_3 - t_2), CR = V_{ref}/V(t_3 - t_2) = (5/10)(1/(3*10^3)) \text{ ms}$$
$$= 0.166*10^{-3} \text{ ms} = 0.166 \text{ seconds}.$$

Yes, it is possible to complete the conversion in less than 1 ms.

Problem 66

What is the accuracy of the A/D converter in Figure 33, with a clock rate of 3 MHz, if the conversion must be completed in less than 1 ms?

Solution:

In 1 ms, with a clock rate of 3 MHz, there are N = (3,000,000 pulses per second*0.001 seconds = 3,000 pulses). The closest binary count less than 3,000 is $2048 = 2^{11}$. Thus, an 11-bit counter would be specified in Figure 33. Then, the accuracy for a 10 V output = 10/2048 = 0.00488 V per count.

Index

Note: Page numbers in **bold** refer to chapters; numbers following indicate page numbers within that chapter.

1-Bit Adder with Carry-Out **1**, 22
1 Bit D/A Is Designed to Reproduce 2 Voltage Levels **3**, 10
2-Bit Adder with Carry-In and Carry-Out **1**
2 Bit Counter **1**, 66
3 Bit Counter **1**, 68
3G Wireless Networks Provide Wireless Access to Global and Metropolitan Area Data Networks **7**, 12
3G Wireless Requirements Specified in International Mobile Telecommunications-2000 (IMT-2000) **7**, 12
4G Is Designed to Operate at 50–250 mps **7**, 13
4G Supports TV Broadcast and Interoperates with the Wired Internet **7**, 13
8 Bit D/A Is Designed to Reproduce 256 Voltage Levels **3**, 10
802.11 Wireless LAN Standards with Data Rates of 11, 55, and 100 Mbps **7**, 12
802.15 Bluetooth Standard. Local Area Network (LAN) Distance Ranges That Are Within 100 Meters **7**, 12

Ability of a Filter to Eliminate Noise **3**, 14
Ability to Learn **13**, 1
Ability of a Mobile Device to Roam, and Achieve Connectivity **7**, 14
Ability of a System to Detect, Locate and Recover from Errors **12**, 13
Ability of Wireless Networks to Operate in Multiple Environments **7**, 14
Absence of Variable Symbols **17**, 11
Abstract Analysis Is Illustrated with an Elevator System **4**, 2
Abstract Approach Can Only Be Applied for Marrying Software and Hardware Design **4**, 2
Acceptable Mission Duration for the Mobile Device User **16**, 16
Acceptable Reliability **12**, 19
Acceptable System Reliability **12**, 9
Access to the Internet **7**, 12
Access Point **15**, 12
Access to Different Resources **16**,
Accumulated Number of Failures **12**, 18
Accumulator **1**, 4
Achieving Realism in Testing **4**, 22
Achieving Visibility into the Operations of Processing Elements **4**, 22
Absolute Ratio Between the Smallest and Largest Possible Values of a Signal **3**, 11
Abstractions Difficult to Grasp **17**, 3
Accounting for Probability of Occurrence of Response Time **4**, 21
Accuracy of Conversion from Analog Input in the A/D Converter to D/A Analog Output **3**, 12

Computer, Network, Software, and Hardware Engineering with Applications, First Edition. Norman F. Schneidewind.

© 2012 the Institute of Electrical and Electronics Engineers, Inc. Published 2012 by John Wiley & Sons, Inc.

Accuracy of Risk Criteria Computations **11**, 9
Achieve an Optimal Tradeoff Between Quantizing Error and Cost **3**, 14
Achieve Specified Reliability **13**, 8
Achieving the Desired Mission Duration **16**, 16
Achieving High Reliability **12**, 1
Achieving Stabilization **15**, 5; **16**, 1
Activation Function **13**, 1
Activities (e.g., User Access to a Web Server) **17**, 3
Activities of Passengers Accessing and Riding in Elevators **17**, 3
Activity and Sequence Diagrams Were the Most Useful for Designing the Poisson Failure Model Function in C++ **17**, 12
Activity Diagram **17**, 2
Activity Diagram Shows Computer Program Control Flows **17**, 2
Activity Diagrams Are One Dimensional **17**, 9
Actual Change in Voltage Sensor Output Voltage **3**, 14
Actual Cumulative Failures **13**, 3
Actual Failure Values **13**, 12
Actual Future Time to Failure **11**, 9
Actual Hardware-Software Reliability **12**, 15
Actual Number of Hardware Failures **12**, 14
Actual Number of Remaining Failures **13**, 12
Actual Number of Software Failures **12**, 14
Actual Number of System Failures **12**, 14
Actual Reliability **12**, 16; **13**, 7; **14**, 8
Actual Remaining Failures **13**, 13
Actual Time to Failure **13**, 15
Adapt Weights to Changes in the Surrounding Environment **13**, 1
Adaptable Learning Algorithms **13**, 3
Adaptive Microsleep (AMS) **15**, 12
Adaptive Procedure **13**, 2
Additional Debugging of the Faults **14**, 10
Address Bus **1**, 4
Adjacent Piconets May Interconnect with Each Other Through Nodes in Overlapping Regions **7**, 13
Adjusts the Input to Achieve the Desired Output **13**, 2
Adverse Atmospheric Conditions **16**, 1
Adverse Channel Conditions **15**, 3
Aggregate the Component Errors **3**, 12
Aggregate the Weighted Failure Counts **13**, 3
All Failures Occurring during Emergency Calls (e.g., 911) Are High Severity **15**, 6
Allow the Application to React According to Context **15**, 12
Almost All Requirements Are Implicit **16**, 10
Alternate Models **17**, 18
Alteration of the Original Shape of the Analog Signal **3**, 11
Amassing Failure Data and Identifying Its Statistical Distribution **17**, 13
Analog Computer Background **3**, 1
Analog Computer Display Readout **3**, 1
Analog Computer Limitations **3**, 5
Analog–Digital Computer Contrast **3**, 5
Analog to Digital Converter Circuit **3**, 2
Analog to Digital and Digital to Analog Components **3**, 2
Analog Signal **3**, 5
Analysis of Asynchronous Sequential Circuits **1**, 51
Analysis of the Design Alternatives **9**, 4
Analysis Highlights the Parts of the Software That Deserve Priority in Testing **16**, 12
Analytic Queuing Models and Simulation Models **6**, 1
Analyze the Operational Amplifier **3**, 13
AND **1**, 16
Annoying Messages **10**, 8
Another Perspective on Probability of Failure **8**, 21
Anti-Malware Software **15**, 1
Application Cannot Rely on Remote Servers **15**, 12
Application Logic **4**, 2
Application State Change **17**, 19
Application-Specific Buffering Techniques **15**, 14
Applications Can Be Modeled **17**, 20

Applications Emerge for Mobile Phones **16**, 3
Applications Include Phones, Pagers, Modems, Headsets, Notebook Computers, Handheld Personal Computers, and Digital Cameras **7**, 13
Applications Running during the Failure **15**, 5
Applying Mathematical Modeling **17**, 12
Applying Object-Oriented Methods **17**, 24
Approximate Remaining Failures by Using the Known Remaining Failures **13**, 12
Architectural Characteristics of a Web Server **14**, 6
Architectural Design **2**, 5
Architecture of a Generic Neural Network **13**, 2
Architecture Has to Guard Against Disconnections of Remote Sensors **15**, 13
Arithmetic Logic Unit (ALU) **1**, 3
Arsenal of Software Reliability Tools **13**, 12
Ascertain Whether Connectivity, Performance, and Availability Can Be Achieved in a Limited Range Environment **7**, 13
Aspects of Mobile Device Development That Cannot Be Quantified **16**, 10
Assess the Predictive Validity of the Models **12**, 11
Assessing Predictive Validity **12**, 14
Assessing Reliability Model Predictive Accuracy **12**, 11
Assessments of the Need for Mobile Device Improvement **15**, 1
Assigning a Time Slice **10**, 10
Assure Quality for Web Applications **14**, 11
Asynchronous Circuits **4**, 1
Asynchronous Interrupts **4**, 23
Automated Failure Data Logger **15**, 5
Automatic Change in Program Flow Control **17**, 17
Automatic Environment **17**, 3
Automatically Identify Code That Should Be Examined **13**, 3
Availabilities **8**, 4
Availability Analysis **8**, 16
Availability Analysis Results **8**, 20
Availability Is a Function of Reliability and Maintainability **7**, 8
Availability Is the Proportion of Operational Time That Maintenance Is Not Being Performed on a Sequence (i.e., the Sequence Is Operating Reliably) **7**, 8
Availability Prediction **7**
Availability of Sequences Represents the Probability That the Set of Nodes and Links That Comprise a Sequence Will Be Available for Operational Use **7**, 8
Availability of the Software **12**, 14
Available Processor Bandwidth **15**, 13
Avoiding, Coping with, and Recovering from Failures **14**, 5

Back Propagation Algorithm **13**, 2
Bad Handling by Software of Indexes and Pointers to Objects **15**, 6
Bandwidth **15**, 13
Based on the Controller Specifications, Develop Software and Hardware Designs **4**, 3
Base Station Contains a Transmitter and Antenna for Transmitting Mobile Device Signals **7**, 12
Baseline for Judging the Utility of Existing Standards from the User's Perspective **7**, 1
Battery Life **15**, 2
Battery Removal Can Clean Up a Permanent Corrupted State **15**, 5
Battery Removal Is Mainly Performed When the Phone Freezes **15**, 6
Benefit Cost Considerations **12**, 6
Benefit-Cost Limit **13**, 4
Benefit-Cost Ratio **10**, 13
Benefit-Cost Relationship **12**, 1
Best Circuit Performance Values **3**, 15
Best Customer Strategy Is to Wait for a Response **15**, 8
Binomial Distribution **12**, 4
Biological Neurons **13**, 1
Bit Error Rate of the Wireless Communication **15**, 3
Blocking Delay **10**, 10
Bluetooth Provides Packet Switching Links **7**, 13

Bluetooth Provides Rapid Ad Hoc Connections Without Cable and Without Line-of-Sight Requirement **7**, 13
Bluetooth Wireless Network **7**, 13
Boundaries among Applications **12**, 12
Buffer Is Part of the Domain Name Server **7**, 4
Built-In Resources **16**, 4
By-Product of the Directed Graph Is a Complexity Metric **17**, 20

C++ Program Logic Development Phase **17**, 13
Cache Hit Rate **1**, 12
Cache Miss Rate **1**, 12
Calculate the Desired Output for Any Given Input **13**, 2
Can Object Oriented Methods Be Applied to Mathematics? **17**, 2
Cell Is a Wireless Geographical Area That Has Access to an Access Point **7**, 12
Cell Phone Mobility **15**, 14
Cellular Networks and Systems Are Diverse **7**, 11
Cellular Networks Have Been Moving from Voice Networks Toward the INTERNET PACKET NETWORK **7**, 11
Central Server Monitoring of the Health of the Mobile Device **16**, 10
Cellular Systems Are Usually Designed with Maximum Cell Range Exceeding 10 km **7**, 12
Challenge of Power Management **15**, 2
Challenge to Reliability **14**, 1
Changes in Sensor Input and Output Voltages **3**, 13
Changing Distances and Locations Between Web Clients and Web Servers **14**, 2
Changing the Data Used in the Variables and Parameters **17**, 4
Characteristics **1**, 74
Characteristics of Models **14**, 5
Characteristics of Real-Time Systems Impose Specific Requirements on the Test System **4**, 23
Characterize the Reliability of Software Systems **13**, 16
Check Denominators for Zero Prior to Division Operations **17**, 17
Chip **1**, 5
Choosing Reliability Prediction Models **13**, 8
Choosing the Most Appropriate Theories and Tools for Different Stages of Development and Different Aspects of the System **4**, 1
Claims Are Not Accompanied by a Discussion of Disadvantages **17**, 4
Clarity of System States **17**, 4
Class **12**, 13
Class Diagram Shows the Relationships among Classes, Objects of a Class, and Methods **17**, 11
Class of Probability Distributions **17**, 8
Classes **9**, 4
Classical Reliability Models **12**, 12
Client and Server Are Restricted in Obtaining Required Communication Reliability **14**, 18
Client Can Move **14**, 2
Client Program **14**, 2
Client Resolving a Web Server Name to the Corresponding IP Address **14**, 4
Client, Server, and Communication Component Reliabilities **14**, 16
Client-Server Protocols **12**, 12
Client Side Problems **14**, 16
Collaborative Mobile Device Visualization **15**,
Collecting and Analyzing Reliability and Performance Data **15**, 16
Collection of Components Comprising the Application **12**, 13
Combination of Methods Is Required to Ensure Adequate Coverage **17**, 17
Combinational Circuits **1**, 22
Combined Hardware-Software Reliability Analysis **12**, 12
Combining Hardware and Software Reliability **12**, 12
Communicate with a Microcomputer **3**, 7
Communication **2**
Communication Channel Error Rate **14**, 17
Communication Channels **14**, 22
Communication Interface **16**, 4
Communication Links **14**, 2
Communication Reliability **16**, 2
Communication Reliability Analysis **14**, 17

Communication Restrictions **15**, 12
Communication Systems **16**, 4
Communication between User Computers and Web Servers **6**, 2
Communication among Web System Elements **14**, 17
Comparators **1**, 28
Compare O-O Approach with Using Equations and Directed Graph Representations **17**, 2
Compare the Performance and Reliability of the Present Internet with a Proposed Internet **16**, 1
Compare Reliabilities by Failure Type and Recovery Action Type **16**, 11
Comparison of Sequences **4**, 23
Comparing Object-Oriented and Mathematical Definition of Terms **17**, 6
Comparing the Prediction Accuracy of Various Reliability Models **12**, 13
Comparison of Present and Future Wired Internet Performance **6**, 56
Compatibility of a Local Network, Wired and Wireless Systems, with the Internet **7**, 14
Complex Functional Requirements **4**, 2
Competing Web Client Access to Web Servers **14**, 2
Complex Hardware and Software **12**, 16
Complex Instruction Set Computing **1**, 5
Complexity Caused by Interrupts **4**, 23
Complexity Contributes a Disproportionate Share of Failures **14**, 11
Complexity of Integrated Modeling **12**, 12
Complexity of Super Computer Hardware Configurations **14**, 6
Complexity of Systems **12**, 12
Complexity of the Web Page Increases at an Exponential Rate **14**, 16
Component Failure Data **12**, 11
Component Models **14**, 1
Component Reliability **12**, 4
Component Reliability Analysis **12**, 7
Component Reliability Prediction **14**, 1
Component-Based Reliability Relationships **12**, 8
Components **1**, 12
Components Can Execute Sequentially or Concurrently **12**, 2
Components Intensively Interact with Each Other and with Their Environments **4**, 1
Components Reliabilities Can Be Properly Weighted to Produce Total System Reliabilities **14**, 22
Components and Their Relationships Are Decided before the System Runs **14**, 1
Comprehensive Testing **14**, 12
Computation of Weights **13**, 4
Computation State **4**, 11
Computational and Memory Resources **12**, 12
Compute the Failure Rates Required to Bring Components into Conformance with Reliability Requirements **14**, 23
Compute the Reliability Improvement Necessary to Achieve the Reliability Goal **14**, 23
Computer Outages in HPC Clusters **14**, 5
Computer-Algebra Language Mathematics **17**, 3
Computing Actual Model Quantities **17**, 5
Computing Cyclomatic Complexity Metrics from the Directed Graph **17**, 3
Computing the MRE **14**, 12
Concentration of Mobile Devices **16**, 2
Concurrent System **12**, 2
Concurrently Running Applications **15**, 14
Conditions **9**, 12
Conducting Performance Simulations **4**, 23
Confidence in a System Before It Is Deployed **12**, 13
Conformance with the Specification **7**, 1
Connect to a Hot Spot **16**, 24
Connect to a Mobile Network on Demand **16**, 10
Connection Available Bandwidth **15**, 13
Connections Excite Neurons **13**, 1
Connections to Sensors That Are the Most Important **15**, 13
Connectivity to Access Points, as Identified by Sending Test Signals **15**, 12
Conservative Reliability Predictions **12**, 4
Consideration in Real-Time Testing Is Complexity Caused by Interrupts Occurring in an Asynchronous Manner **4**, 23
Constant, Increasing, and Decreasing Failure Rate Functions **12**, 15

Constraints of Presentation **15**, 12
Constructing a Web Page by a Web Server **14**, 3
Context of the Application Is Highly Dynamic **15**, 17
Context-Architecture **15**, 13
Context-Aware Migratory Service **15**, 14
Context Aware and Network Aware Mobile Computing **15**, 1
Context-Awareness **15**, 1
Context of the Mobile Environment **15**, 1
Context-Sharing **15**, 13
Continuous Operation Requirement **12**, 12
Continuously Changeable Aspects of Physical Phenomena **3**, 1
Continuously Varying Analog Signal **3**, 10
Contribution of Non-Functional Characteristics **14**, 1
Control **2**, 1; **9**, 10
Control Bus **1**, 5
Control Commands Are Issued, e.g., by System Controller to Operations **4**, 4
Control Functions **4**, 4
Control Hazards **1**, 9
Control Logic for Decoding (i.e., Identifying) Input Service Requests **4**, 16
Control Unit **1**, 4
Controls **4**, 6
Conversion System Errors **3**, 12
Converter Circuit Components **3**, 15
Corrected by Servicing the Phone **16**, 6
Corrective Action **11**, 3; **14**, 1
Correctness of Program Execution **17**, 13
Correctness Proofs **12**, 15
Correctness of the Software Production Process **16**, 1
Cost of Achieving Reliability Is Critical **13**, 9
Cost Considerations **12**, 5
Cost-Effective Predicted Reliability **11**, 9
Cost-Effective Way of Engineering Systems **14**, 1
Cost Estimation **13**, 1
Cost Is Based on Number of Configuration Components **12**, 18
Cost per Megabyte **10**, 13
Cost and Schedule Overruns **4**, 2
Cost of Testing **13**, 16
Cost of Testing to Remove Faults **12**, 16
Counteract Security Threats **15**, 1
Coverage of Each Executable Statement and the Execution Result **13**, 3
CPU and Network Intensive **15**, 2
Criterion of One Remaining Failure **11**, 8
Critical Aspects of Real-Time System Operations, such as Elevator Direction of Travel, May Be Overlooked **4**, 1
Critical Faults **2**, 7
Critical Performance Factor **15**, 15
Crucial Factors Involved in Obtaining Satisfaction in Using Networks **7**, 15
Crucial Properties Being Included in Network Standards **7**, 1
Cumulative Failures **11**, 6
Cumulative Failures during Testing **13**, 3
Current Edge of the Network Will Often Be Just One Hop to the Internet **6**, 2
Current Mobile Networks Are Unable to Provide Highly Reliable Service **15**, 1
Current (Physical) Position of the Mobile Device **15**, 13
Current Research Focuses on the Present Internet Configuration **6**, 1
Current Service Can Automatically Migrate to a Node **16**, 24
Cyclic Redundancy Check **2**, 8
Cyclomatic Complexity Analysis of Web System Reliability **14**, 2
Cyclomatic Complexity Is a Metric for Evaluating the Relative Quality of Software Systems **17**, 17
Cyclomatic Complexity Is the Number of Independent Paths **14**, 2
Cyclomatic Complexity Metric: (Number of Edges – Number of Nodes) + 1 in a Directed Graph **17**, 2
Cyclomatic Complexity of the Poisson Failure Model **17**, 17

D Flip Flop **1**, 44
Data: Historical Data **17**, 5
Data Base Management System Execution **12**, 2
Data Bus **1**, 5
Data Can Be Protected by Access Controls and Encryption **17**, 4
Data Hazards **1**, 9

Data Recorded Against the Hardware and Software **12**, 12
De Morgan's Theorem **1**, 20
Deactivated Sensor Cannot Sense Any Information about the Context **15**, 12
Deadline Requirements **10**, 11
Deadlock Analysis and Prevention **10**, 15
Debugging of Client Software and Hardware **14**, 17
Decision Analysis State **4**, 11
Decision Operations: Control Program Flow **17**, 5
Decisions **9**, 1
Decoder Logic for Generating Service Request Interrupts **4**, 16
Decoders **1**, 31
Decompose the System into Its Component Parts **14**, 22
Decrease in Performance **15**, 14
Decrease in Quality of Communication **14**, 2
Decrease of Transmission Speed **14**, 2
Decreasing Reliability Necessitates Increasing Test Time **16**, 17
Defective Memory **12**, 14
Defined by the Device Hardware and Software **15**, 13
Defining the Model Objects **17**, 8
Definition Depends on the Context of the Application **17**, 6
Degradation of the Communication Lines **14**, 2
Degrade Exponentially with Operating Time **14**, 17
Degree of Interference **7**, 13
Delay or Shut Down Would Cause Deadlines to Be Missed **15**, 3
Demonstrate Whether the Proposed Internet Is Viable **6**, 1
Demultiplexers **1**, 38
Design of Binary Counters **1**, 49
Design of Complex Real-Time Systems Is Quite Challenging **4**, 1
Design Decisions **2**, 1
Design Levels **4**, 4
Design May Have to Be Heavily and Hurriedly Modified **4**, 2
Design Principles **2**,
Design Process Elements **4**, 5
Design Provides High Speed, e.g., It Is Well Suited to Real-Time Applications That Must Meet Deadlines, but at the Expense of Relatively Complex Programming **4**, 3
Design Representations Starts with Generic and Application-Specific System Level Functions and Ends With Integrated Testing and Performance Evaluations **4**, 1
Design of Synchronous Sequential Circuits **1**, 61
Design Tests of Simulated Performance **4**, 19
Designing an Architecture for Mobile Devices **15**, 12
Designing in Higher Quality **16**, 16
Desirable Properties of Network Performance, Reliability, Maintainability, and Availability **7**, 1
Desirable Properties of Network Standards **7**, 1
Desirable Properties of a Programming Language **9**, 1
Detailed Analysis of a Programming Language **9**, 9
Detailed Design **2**, 12
Detailed Design Example **9**, 13
Detect and Record the Occurrences of Freezes and Reboots **15**, 5
Detect the Status of the Phone during a Failure **15**, 5
Detecting Faults **12**, 5
Detecting Logical Errors **4**, 23
Deteriorate Voice Quality **15**, 12
Determination of Failure Occurrence **7**, 5
Determine How Long to Test **12**, 4
Determine How Well the Run-Time Program Will Perform **17**, 12
Develop an Integrated and Comprehensive Design Approach with the Objective of Providing Engineers with a Roadmap for Improving Real-Time System Design **4**, 1
Develop the Real-Time System Generic Design of a Particular Artifact, such as a State Diagram **4**, 1
Develop the Specific Features of the Application **4**, 25

Developer Must Test in the Operating Environment of the Mobile Devices **16**, 3
Developers Produce for a Mass Mobile Device Market **16**, 10
Developing and Analyzing Comprehensive Web System Reliability Models **14**, 2
Developing Mathematical Functions **17**, 3
Development of Mathematical Software **17**, 20
Development of Software for Mobile Computing Devices **16**, 3
Device Behavior Is Highly Interactive **16**, 3
Device Failure Affecting the Application **16**, 2
Device Going Offline **16**, 2
Device Identification **15**, 13
Device Mobility **15**, 3
Device Type **15**, 13
Devices Are Highly Resource Constrained **16**, 3
Devices Are Increasingly Being Used in Multimedia Streaming Type Applications **15**, 2
Devices Constantly Accept Activations from Users **16**, 3
Devices Less Dependent on Particular Locations and Resources **15**, 3
Devices May Challenge Traditional Understanding of Network Topology **6**, 2
Devices Will Be Able to Connect Directly into the Internet **6**, 1
Devices Would Be Assigned Permanent IP Addresses **6**, 2
Diagrams Portray the Sequence of Activities in the Code **17**, 12
Difference Between Number of Correct Modules and Number of Failed Modules **16**, 8
Difference in Failure Behavior Between Hardware and Software **12**, 13
Different Failure Properties for Web Client, Web Server, and the Interconnected Communication Channels **14**, 22
Difference That Exists Between Correct Modules and Failed Modules **16**, 9
Differential Equations **3**, 2
Differences for Throughput **4**, 23
Digital to Analog (D/A) Conversion **3**, 9
Digital to Analog Converter **3**, 1
Digital Cellular Networks **15**, 4
Digital Computer Floating-Point Calculations **3**, 6
Digital Data Is Ready for Transmission on the Data Lines **3**, 7
Digital Logic **1**, 74
Digital Signal Comprised of Discrete Binary **3**, 10
Diminishing Returns in Finding and Correcting Faults **13**, 8
Directed Graph Can Be Used to Represent a Computer Program **17**, 1
Directed Graph of the Program Logic **17**, 13
Directed Graph Representation of a System **14**, 2
Directed Graph Representation of the Mathematical Model **17**, 3
Directed Graph Will Serve as the Vehicle for Expressing C++ Program Logic **17**, 13
Discontinuity of Network Connections **15**, 12
Discover Potential Bottlenecks **15**, 15
Disk Systems **12**, 6
Dispatcher **10**, 6
Displays Need to Be on at All Times **15**, 2
Disrupt an Entire Network **15**, 2
Disrupting the Wireless Connection **15**, 3
Disruption Caused by Software Incompatibilities **7**, 9
Disruption Due to Failure **15**, 4
Distance Between Clients and Servers **14**, 2
Distortion **3**, 11
Distortion Can Be Minimized by Using an Adequate Number of Bits in the Digital Representation of the Analog Signal **3**, 11
Distortion Is Measured by the Difference Between the Correct Signal Change in Adjacent Values and the Actual Change in Adjacent Values **3**, 11
Distributed Operating Systems **10**, 19
Distributed Web Services in an Internet Setting **14**, 1
Distribution of Errors Is Usually Non-Uniform **14**, 17

Distribution of Failures and Recovery Actions **15**, 6
Distribution of Failures over Time **16**, 11
Distribution of Faults in the Hardware and Software **12**, 13
Diversity of Mobile Devices Reduces the Reusability of Test Cases **16**, 3
DNS Look Up Accesses **14**, 16
DNS Lookup Failures **14**, 16
Domain Controller **14**, 4
Domain Name Server (DNS) **6**, 2
Domain Name System (DNS): Present System **6**, 30
Dominant Cause of TCP Connection Failures **14**, 16
Dramatic Reduction in System Failure Rate **14**, 12
Drastic Reductions in Failure Rate by Eliminating Software Faults **16**, 12
Duty Cycles **8**, 4
Dwelling on Speed Is Certainly Not the Whole Story in Assessing Network Standards **7**, 15
Dynamic Nature of Web Surfing **14**, 2
Dynamic Part That Can Respond to Changing Operating Conditions **16**, 3
Dynamic Range **3**, 5
Dynamic Range, Phase Distortion, and Signal Representation **3**, 12
Dynamically Configured Web Services **14**, 1

Each Component and the System Satisfy the Reliability Requirement **14**, 22
Each Component Can Be Thoroughly Tested **14**, 1
Each Facet of the Failure Model Is Defined and Analyzed **17**, 6
Each Mobile Device Moves in an Arbitrary Manner **15**, 3
Each Statement in a Program Could Potentially Cause One or More Exceptions to Be Raised **17**, 17
Early Stages of Testing **13**, 8
Edges (Transfer Control, Iteration Control, and Return) **17**, 13
Effective Criterion **13**, 7
Effects of Additional Faults **12**, 13
Effects of Faults **12**, 13
Efficiency Test **7**, 2
Efficient Implementation Is Obtained by the Objects Failure Count and Failure Time Feeding the Object Failure Rate **17**, 12
Efficient Resource Allocation in Different Operational Scenarios Is Required **4**, 1
Efforts to Reduce Faults and Subsequent Failures **12**, 16
Electric Grid Network **15**, 16
Electric Power Grids **15**,
Electric Signal **14**, 2
Electric Utility Mobile Device Senses Power Outage **15**, 14
Electric Utility and Customer Mobile Electric Meter Reading Devices **16**, 24
Electrical Equivalent of a Physical System **3**, 1
Electromagnetic Interferences (EMI) **15**, 3
Electronic Analog Computers **3**, 1
Electronic Digital System Uses Two Voltage Levels to Represent Binary Numbers **3**, 6
Elements of a Requirement **17**, 4
Elevator Software Design **5**,
Elevator System Is Used as the Design Example Because It Has Interesting Properties, such as Interruptible Floor Traversal Sequences **4**, 1
Eliminating the Need for Name to IP address Translation **6**, 2
Emphasis in the Design Process Is on Hardware Design, but Not Neglecting the Mapping Between Hardware and Software Designs **4**, 16
Empirical Failure Probability **16**, 7
Emulate the Operational Environment **16**, 3
Enable Global Roaming **7**, 12
Encapsulation Is the Inclusion Within a Program Object of all the Resources Needed for the Object to Function **17**, 2
Encoders **1**, 34
End-to-End Transmission and Processing of Data in a Network Is Comprised of Subsets of the Total End-to-End Chain Called Sequences **7**, 5
Engineer Charged With Designing Networks Should Include Crucial Factors in the Specifications and Establish a Test System **7**, 15

Entity's Normal Behavior **15**, 2
Entity's Observed Behavior Deviates Significantly from Its Profile **15**, 2
Environmental Conditions **16**, 2
Equation: Mathematical Implementation of a Function **17**, 5
Equation as an Object **17**, 3
Equations and Directed Graphs Are a Model for Writing Code **17**, 20
Equations Do Just Fine Because They Are the Models of Mathematics **17**, 20
Equations Have Variables and Parameters That Can Be Made Extensible **17**, 4
Equations Provide an Excellent View of Its Implemented Software **17**, 4
Erroneous Bits Received per Bits Transmitted **14**, 17
Error Between the Prediction Models and the Historical Reliability Data **12**, 11
Error Conditions in System and Application Modules **15**, 5
Error of Conversion **3**, 6
Error Rate **14**, 16
Error in the Software **12**, 1
Errors That Could Arise in Each Component, Whether A/D or D/A **3**, 12
Estimate the Feedback Control Signal **16**, 8
Estimate Latency during Performance Testing **16**, 3
Estimate the Location of a Mobile Station **15**, 12
Estimating Model Parameters **13**, 5; **17**, 5
Estimating the Number of Failures **7**, 6
Estimation of Failure Rate **7**, 5
Ethernet, also Known as IEEE 802.3 **7**, 10
Evaluate Actions to Recover from a Device Failure **15**, 4
Evaluate the Cost of Testing **11**, 5
Evaluate More Than One Model **13**, 8
Evaluate the Reliability of the Software Design **17**, 20
Evaluate Results in the Context of the Application **13**, 9
Evaluating the Behavior and Performance of Complex Systems **12**, 13
Evaluation of Programming Languages **9**, 1
Event-Driven Software Style Has Evolved Largely to Deal with Complexity **4**, 8
Event Order and Time of Occurrence Are Crucial in Determining System Performance **4**, 15
Event Sequence: Series of State Transitions **4**, 6
Example of Comparing Object-Oriented with Mathematical Approaches **17**, 24
Exception Will Cause an Automatic Change in Control Flow **17**, 17
Excessive Activation of Wireless Communication Links by the User **16**, 7
Excessive Cost of Testing **11**, 9
Excessive Number of Web Page Operations Is Bad News for Reliability **14**, 17
Exclusive NOR (XNOR) **1**, 17
Exclusive OR (XOR) **1**, 18
Exclusive Use of Abstract Representations Is Unwise Because It Is Important to Consider the Physical Properties of the Real-World System **4**, 1
Execute the Programs on a Computer **17**, 19
Executing Test Scenarios **9**, 9
Execution Time **4**, 3
Existing Standards **7**, 1
Expected Fractional Value **7**, 6
Expected (Mean) System Response Time **4**, 19
Expected Number of Failed Modules of Failure Type **16**, 8
Expected Number of Failed Modules of Recovery Type **16**, 9
Expected Number of Failures **15**, 6; **16**, 11
Expected Number of Failures and Failure Rate **16**, 11
Expected Number of Failures and Failure Rate Analysis **16**, 11
Expected Number of Failures and Failure Rates Corresponding to the Recovery Action Categories **15**, 6
Expected Recovery Action Is a Probabilistic Function of the Failure Types **16**, 9
Expected Reliability **11**, 10
Expected Value **7**, 6
Expected Value of Predicted Time to Failure **13**, 16
Explicit and Implied Requirements **16**, 10
Explosion of Internet Connectivity **15**, 1
Exponential Decay Error Rate **14**, 17

Exponential Decrease in Reliability **14**, 16
Exponential Distribution **14**, 17; **16**
Exponentially Decreasing Reliability **14**, 16
Exponentially Distributed Failure Times **16**, 11
Exponentially Distributed Operating Times **17**, 6
Exponentially Distributed Pattern of Failure Data **14**, 7
Exponentially Distributed Time Between Failures **12**, 3
Expose Confidential Data and Risk Attacks **15**, 1
Extensibility **15**, 12
Extensive Error Analysis **3**, 15

Facets of Conversion Logic **3**
Factoring in Probability of State Transitions **14**, 9
Fading **15**, 3
Failure Count at Test or Operating Time **17**, 6
Failure Count Data **17**, 10
Failure Count Interval **13**, 6
Failure Data **15**, 1
Failure Data Identification Phase **17**, 15
Failure Data used in Parameter Estimation **13**, 4
Failure Is Considered to Be of High Severity When Recovery Requires the Assistance of Service Personnel **16**, 6
Failure Is Considered to Be of Low Severity If the Device Operation Can Be Reestablished by Repeating the Action or Waiting for a Certain Amount of Time **16**, 6
Failure Is Considered to Be of Medium Severity When the Recovery Requires Reboot or Battery Removal **16**, 6
Failure Is Firmware-Related **15**, 4
Failure Occurrences Can Be Associated with the User Activity at the Time of the Failure **16**, 6
Failure Phenomena of Web Server Systems **14**, 11
Failure Rate **11**, 3; **12**, 1; **15**, 3; **17**, 6
Failure Rate Decreases After Installation **16**, 1
Failure Rate Decreases After Installation, Eventually Reaching a Steady State **15**, 3
Failure Rate and Failure Counts **8**, 8
Failure Rate λ Is Computed **17**, 12
Failure Rate Must Be Estimated **7**, 5
Failure Rate Parameters **13**, 4
Failure Rate Required to Achieve Specified Reliability **12**, 16
Failure Rate of a System Grows Proportional to the Number of Processors in the System **14**, 6
Failure Rate Varies Considerably Across Servers and Clients **14**, 6
Failure Rate of Web Servers **14**, 11
Failure Rates **16**, 11
Failure and Recovery Action Data **16**, 1
Failure Scenarios **11**, 10
Failure Severity Code **13**, 4
Failure Severity Is Classified According to the User Perspective **15**, 5; **16**, 6
Failure Severity Is Reflected in the Model According to the Severity Codes **16**, 8
Failure of a Single Processor **12**, 4
Failure Time **12**, 15
Failure Types **15**, 4; **16**, 54
Failure Type and Recovery Action Type Results **16**, 16
Failure Type Estimation Results **16**, 10
Failure Type Testing Takes More Time **16**, 17
Failure Types Below the Limit Should Be Investigated to Identify the Cause of Excessive Failures **16**, 16
Failure in Web Page Processing by the Web Server **14**, 3
Failures **7**, 3
Failures and Degradation of the Communication Links **14**, 2
Failures Can Be Traced to Domain Name Server (DNS) Problems **14**, 16
Failures Caused by Noise in the Communications Network **16**, 17
Failures Creating, Sending, and Receiving Text Messages **15**, 6
Failures in the Disk Storage Unit Nodes **14**, 4
Failures Due to Environmental Problems (Power Outage) **14**, 5
Failures Due to Operator Error **14**, 5

Failures Occur According to a Poisson Distribution **15**, 6
Failures Occur during Voice Calls **15**, 6
Failures in Storage Systems **14**, 5
Failures Using Bluetooth **15**, 6
Fairness **10**, 6
Fault Detection and Correction **2**, 8
Fault and Failure Correction **8**, 4
Fault and Failure Correction Analysis Results **8**, 18
Fault Injection **12**, 13
Fault Localization **13**, 2
Fault Removal **14**
Fault Tolerance of Mobile Computing Systems **15**, 5
Fault Tolerant Web Systems **14**, 2
Faults **11**, 1
Faults Can Be Dependent **12**, 4
Faults and Failures Corrected **8**, 5
Faults Removed **12**, 4
Faults, such as Garbled Data on a Link, Attributed to a Noisy Communication Channel, Cause
Features Can Be Disabled **15**, 12
Feedback during the Development Stage **12**, 13
Feedback Used to Revise Mobile Device Requirements **16**, 8
Feed-Forward Structure **13**, 3
Field Failure Data Analysis **15**, 5
Figures Do Provide a Base Line for Starting Development Process **17**, 18
Find Other Devices to Execute the Mobile Programs **15**, 14
Firmware Update **15**, 5
First Interval of Test Failure Data **13**, 5
Fixed Information Perimeter **15**, 1
Flexible Failure Function **12**, 15
Flip Flops and Latches **1**, 40
Focus on Failure Type **16**, 7
Focus on Recovery Action Type **16**, 8
Form of Design Presentation **9**, 2
Formal Methods **12**, 13
Formulate Equations from Problem Specifications **17**, 2
Formulation of Test Time May Understate the Time Required to Identify All Mobile Device Hazards **16**, 14
Fraction of Failures **15**, 7
Freeze (Lock-Up or a Halting Failure) **15**, 4; **16**, 5
Freezes Are More Annoying Than Output Failures **15**, 6
Freezes Are Usually Recovered by Pulling Out the Battery **15**, 6; **16**, 7
Frequency Response **3**, 5
Frequency Spectrum **7**, 11
Frequency with Which Errors or Noise Are Introduced into Communication Channels **14**, 17
Function **12**, 1
Function Associates a Single Output to Each Input Element **17**, 6
Function Is the Task the Object Must Achieve **17**, 5
Functional Decomposition Method **12**, 12
Functional Logic **12**, 2
Functional and Non-Functional Specifications **12**, 2
Functional Oriented vs. Data Oriented Design **9**, 2
Functionality Information **12**, 12
Functions **1**, 3; **9**, 2
Functions of a Computer System **12**, 12
Functions That Are Performed in a System **12**, 2
Fundamental Properties as Predictions of Reliability, Maintainability, and Availability **7**, 1

Gain **3**, 5
Gain of the Microcomputer-Controlled Operational Amplifier **3**, 12
General Packet Radio Service (GPRS) **16**, 4
General Registers **1**, 4
Generating System Reliabilities **12**, 8
Generation of Failures **12**, 16
Given the Erratic Channel Conditions, Reliability Should Be Predicted Under These Conditions to Have a Useful Standard **7**, 12
Global Positioning System (GPS) **15**, 13
Goal Is to Develop Mathematical Software **17**, 20
Good Test Strategy Is to Exercise the Independent Paths in Debugging **17**, 17
GPRS Mobile Device **16**, 4

GPRS Requirements Being Translated to Software Code Compatible for Operating with a Communications Carrier **16**, 4
Gradient Descent (Ascent) Learning Rule **13**, 2
Great Deal of Functionality Squeezed into a Small Memory Space **16**, 3
Great Variation in Both Hardware and Software Failure Counts **12**, 15
Greater the Relative Difference Between Specified and Predicted Reliabilities the Greater the Risk **13**, 9
Grid Computing **16**, 24

Handheld Computing Device with a Short-Range Radio Link, such as IEEE 802.11b or Bluetooth **16**, 3
Hard Disk Access Rate **1**, *12*
Hardware Configurations Are Very Complex **14**, 11
Hardware Description Language **1**, 74
Hardware Failures **14**
Hardware Function, Software Function, and Some Form of Interaction **12**, 12
Hardware Functions **4**, 11
Hardware Level **4**, 5
Hardware-Oriented Design Has to Deal with More Problems Than Software-Based Design, Especially the Progression of Time **4**, 16
Hardware Redundancy **12**, 12
Hardware Reliability **12**, 3
Hardware and Software Failure Rates **12**, 15
Hardware and Software Failure Relationships **12**, 13
Hardware-Software Models **12**, 12
Hardware-Software Predictive Reliability **12**, 15
Hardware and System Reliability **12**, 15
Having Power When the Device Is Turned On **16**, 10
Hazard Functions **14**, 6
Heavy Traffic Load in the Local Network **7**, 9
Heterogeneous Mobile Devices **16**, 24
Hierarchy: Divide System into Modules That Are Easer to Understand Than the Complete System **4**, 4
High Expectations for the Reliability of the Software on Mobile Devices **16**, 2
High Failure Rate, Unstable Behavior Failures **15**, 9
High Latency Wireless Networks **16**, 1
High-Performance Computing (HPC) Installations **14**, 5
High Probability of Short Node and Link Times and Low Probability of Long Times **7**, 6
High Reliability at Low Risk **13**, 9
High Severity Types of Failures **15**, 9
High Severity When Recovery Requires the Assistance of Service Personnel **15**, 6
Higher Complexity Software Has Lower Quality **17**, 17
Higher Probability of Small Message Size **14**, 17
Higher Quality That Is Required in Voice and Other Real-Time Applications **7**, 11
Higher Speed Can Result in Failures Occurring at a Higher Rate **7**, 6
Highly Compact Functionality Must Be Reflected in the Testing Strategy **16**, 3
Highly Unlikely That There Would Be Failure Free Service **15**, 7
Historical Computed Reliability **12**, 11
Historical Error Rate of n Errors per Web Page Operation **14**, 16
Historical Failure Data **14**, 8; **17**, 5
How Long a System Can Be Operated at Specified Values of Reliability **12**, 18
How Long to Test **12**, 4
How Long to Test Components **12**, 4
HTTP Protocol **14**

Identification of Coding Details Flow More Naturally from Mathematical Expressions **17**, 13
Identification of System Elements **2**, 2
Identifier **15**, 15
Identify the Best Customer Strategy **15**, 8
Identify the Key Paths to Test Based on the Cyclomatic Complexity Metric **17**, 13
Identify a Mobile Device Technology with a Relatively Low Reliability Rating **16**, 4
Identify the Number of Failure Counts That Occur at Test Time **17**, 13

Identify the Number of Web Page Operations That Cause Reliability Degradation **14**, 23
Identify Possible Low Sequence Reliability Values That Would Be Indicative of Low Values of Node and Link Reliabilities **7**, 6
Identify States and State Transition Probabilities **14**, 22
Identify the Services the Objects Are to Perform **17**, 13
Identify the Several Phases and Steps in Program Implementation **17**, 13
Identifying and Defining the Performance and Reliability **6**, 1
Identifying Independent Paths and Evaluating Program Test Coverage **17**, 17
Identifying the Initial Search Location **14**, 4
Identify Web Page Operations **14**, 22
IEEE 802 Family of Standards **7**, 11
IEEE 802 Wireless Networks **7**, 10, 12
IEEE 802.11 Power Saving Schemes **15**, 12
If the Reliability Test Fails, the Failure Is Reported to the Maintenance Activity **7**, 14
Illuminate the Various Perspectives That the Diagrams Provide **17**, 8
Impacting the User Experience **15**, 2
Implementation Elements **4**, 1; **5**
Implementing Cohesion and Coupling **9**, 9
Implementing Software, Using the NASA Space Shuttle Flight Software, as an Example **17**, 13
Importance of Quality of Service **14**, 2
Importance of Web Systems in Contemporary Society **14**, 1
Important Characteristic of Mobile Devices Is That a Given Device May Communicate with More Than One Communications Carrier **16**, 4
Important Concepts about Devices That Interconnect with Digital Computers **3**, 15
Important to Have a Close Relationship Between the User System and the System Control Functions **4**, 5
Important Network Requirements **7**, 14
Important to See Whether the Computation Results Appear to Be Reasonable **17**, 17
Important That the Test Bed Be Automated **7**, 14
Important Variable Types **9**, 13
Improve Quality of Service, Reliability, etc. on the Existing Platform **6**, 2
Improve the Signal to Noise Ratio **16**, 10
Improved Power Management Is Needed in Mobile Devices to Increase Their Utilization **15**, 5
Improvement in Reliability **15**, 7
Improvements in Both Hardware and Software Reliability **15**, 16
Improvements That Would Make Standards More Valuable for the User **7**, 9
Improving the Design and Implementation of a System **12**, 13
In C+ + , a Function Is a Named, Independent Section of Code **17**, 5
In Failure Type Testing, the Tester Is at the Mercy of the Operating Environment of the Mobile Device **16**, 17
In Order to Provide Increased Security of Data, Every User Computer and Mobile Device Would Have Its Own IP Address **6**2
In Programming Languages, a Function Is a Subroutine That Can, If Required, Return a Single Value to the Caller **17**, 5
In Real-Time Programs, the Time of Occurrence of Events Rather Than the Order of Events Is Crucial in Determining the Outcome of a Computation **4**, 15
Inability of Real-Time Software to Meet Its Primary Nonfunctional Requirements **4**, 2
Inability to Transmit Data Between Two Nodes **7**, 3
Inaccurate Assessments of the Conditions for Safe Missions **11**, 9
Incorrect Management of Buffer Sizes **15**, 6
Incorrect Use of the Device Resources **15**, 6
Increase in System Reliability Necessary to Achieve the Reliability Goal **14**, 18

Increasing Functions of Cumulative Failures and Reliability **13**, 12
Increasing the Signal to Noise Ratio Will Increase the Range of the Wireless System **7**, 13
Increasing User Productivity in Their Use of Computer Networks **7**, 1
Independence of Faults That Cause Failures **12**, 4
Independent Path Is One That Cannot Be Formed by Combining Other Paths in the Directed Graph **17**, 2
Indicates That There Is More Noise Than Signal and That the Recovery Method Is Dysfunctional **16**, 8
Industry Is Developing a Software Defined Device That Can Be Dynamically Defined in Real-Time **16**, 3
Information Hiding: A Software Design Technique That "Hides" System Details **17**, 2
Information Hiding and Modular Design **17**, 4
Information about the User's Environment **15**, 12
Ingredients of Other Infrastructures **15**, 13
Inheritance: A Property of Object Oriented Design That Allows an Object to Acquire the Properties of Its Class **17**, 1
Inheritance Makes O-O Systems More Extensible **17**, 4
Inhibit Neuron Output **13**, 1
Injection of Faults and Failures Is Simulated by Randomly Selecting Links and Nodes to Be Injected **7**, 3
Input Connections **13**, 1
Input Data **9**, 4
Input Data Component Execution **12**, 2
Input Driven Software Models **16**, 1
Input Failure **15**, 4; **16**
Input Failure Counts **13**, 7
Input Failures Are High Severity **15; 16**, 6
Input Processing State **4**, 11
Input Request to the Internet, Provides the Basis for Computing the Performance and Reliability of the Present and Proposed Internets **6**, 1
Insight into How Real-Time Systems Must Function **4**, 2
Instruction Register **1**, 4
Insufficient to Limit Verification of the Correctness of Program Output to the Identification of Independent Paths and the Associated Test Strategy **17**, 17
Integer Number of Failures Would Occur over the Nodes and Links **7**, 6
Integrate Component Reliabilities into Total System Reliability Predictions **14**, 23
Integrated (the Various Parts Fit Together to Form a Coherent Whole) **17**, 4
Integrated and Comprehensive Design Approach **4**, 25
Integrated Software–Hardware Design **4**, 6
Integrated Software–Hardware Design Is Achieved by Mapping Between Software and Hardware Designs **4**, 16
Integrated Software–Hardware Design Methodology **4**, 8
Integrates the Varying Analog Input Signal Voltage **3**, 1
Integration of Mobile Devices and Environmental Infrastructures **15**, 14
Integrative Analysis to Produce Total System Reliability Predictions **14**, 22
Intelligent Control **13**, 1
Intelligent Mobile Meter Readers **16**, 24
Interaction Between an A/D Converter and Microcomputer **3**, 7
Interaction Between a User Application and a Migratory Service Can Continue Uninterrupted **16**, 24
Interaction of People with Computerized **17**, 19
Interaction of Software and Hardware during Program Execution **4**, 1
Interactions among Components **12**, 15; **14**, 1
Interchange of Commands Between Converter and Microcomputer **3**, 7
Interface Between a Mobile Device and a Mobile Network **16**, 4
Interface Between Modules Rather Than in Modules **17**, 2
Interfaces Have Been a Major Source of Failures in Computer Systems **16**, 4
Interfaces May Not Easily Interconnect Because Inputs May Arrive at Unpredictable Times **4**, 4

Interfaces Represent the Major Software Modules to Be Developed by the Mobile Device Process **16**, 4
Interfaces with Objects and between Objects **17**, 11
International Mobile Telecommunication 2000 (IMT-2000) Standard **7**, 11
Internet Data Traffic **6**, 70
Internet Protocol (IP) **6**, 2
Internet Reliability Analysis **6**, 59
Internet Router **6**, 2
Internet Router: Present Wired System **6**, 30
Internet Web Sites **15**, 15
Internet's Ability to Adapt to Improved Performance and Reliability Requirements **6**, 2
Internet will Connect Vast Numbers of Tiny Devices Integrated into Cell Phones and Other Mobile Devices **6**, 3
Interoperability of Mobile Devices with Other Computing Infrastructures **15**, 13
Interoperability with Other Mobile Devices and Electric Grid Infrastructure Will Be Improved **16**, 24
Interprocess Communication **12**, 12
Interrupt Handling **1**, 15
Interrupt Processing **3**, 7
Interrupt Signal Generated by Hardware Triggers Software Interrupt Processing Routines **4**, 6
Interrupted by a Request **4**, 23
Interruptible Event Sequence Causing State Transition **4**, 6
Interruptible Sequence of Operations Causing Interrupts to Be Processed Out of Sequence **4**, 4
Interrupts **4**, 6
Intrusion Detection **15**, 1
Inverse of the Signal to Noise Ratio **16**, 8
Invest in Higher Reliability Communication Facilities **14**, 18
Invocations of Web Service Operations Are Independent **14**, 3
Invokes Backup Power Supply **15**, 14
I/O Channels Must Have Sufficient Transfer Rate **4**, 5
I/O Channels Must Have Sufficient Transfer Rate to Satisfy Elevator System Response Time Requirements **4**, 6
I/O Channels with Sufficient Transfer Rate to Keep Up with Real-Time Transaction Input Rate **4**, 11
IPv6 Does Not Provide Any Better (Or Worse) Support for Quality of Service Than IPv4 **6**, 2
Iteration: Repetition of an Operation **17**, 5
Iteration in O-O Defined as: An Operation That Permits All Parts of an Object to Be Accessed in a Well-Defined Order **17**, 3

JK Flip Flop **1**, 47
Joining of Disparate Components of a System Is a Complex Process **16**, 4

Karnaugh Maps **1**, 20
Key Indicator of Acceptable Performance Is That Response Time Is Satisfied **4**, 23
Key Issue in Providing Multimedia Services over a Wireless Network Is the Quality-Of-Service
(QoS) Support in the Presence of Changing Network Connectivity **7**, 12
Key to Maintaining Wireless Communication **15**, 3

Large Gains in Noise Reduction Would Be Achieved Through Testing If the Number of Correct Modules, Due to Eliminating Failures, Is Already Small **16**, 13
Large Gains in Noise Reduction Would Be Achieved Through Testing If the Number of Correct Modules, Due to Recovery Action, Is Already Small **16**, 13
Large Obstructions **15**, 3
Large Scale Signal Path Loss **15**, 3
Larger the Failure Rate, the Shorter the Mission Duration **16**, 12
Latency Is Defined as the Time Required for the Data Signal to Be Transmitted Through the Communications Medium **16**, 3
Latency Is the Reciprocal of Data Rate **16**, 3
Later Stages of Testing **13**, 8
Learn Rules **13**, 1

Learn the Input-Output Relationship **13**, 3
Learning Algorithm **13**, 1
Left Shift Register **3**, 1
Length of Time Slice for Switch Action **10**, 9
Level of Detail That Is Compatible with the Phase and View **17**, 18
Lightweight Encryption **15**, 1
Lightweightness **15**
Likelihood of the Abort of a Connection **15**, 13
Likelihood of Processor Failure **12**, 4
Likelihood of Statement Bugs **13**, 3
Limit: Constraint Imposed on a Function **17**, 5
Limit Value **13**, 3
Limited Computational Resources of These Devices **16**, 3
Limited Range Wireless Network **7**, 13
Linear Text Format **14**, 3
Link Delay Times **6**
Links Between Distributed Databases **12**, 13
List of Running Applications **15**, 5
Little Understanding of How and Why Mobile Phones Fail **15**, 5
Local Area Network **12**, 15
Local Network **6**, 2
Local Network: Present Wired System **6**, 30
Local Network Components Have Smaller Sizes That Are the Primary Driver of Maintenance Actions **7**, 9
Local Network Components Operate Faster Than Internet Components **7**, 6
Local Network Processing Times **6**, 44
Local Network Router **6**, 3
Local Network Router: Present Wired System **6**, 42
Local Network Router Server Queue **7**, 9
Local Network Sequences Would Be Subject to Further Testing to Discover and Remove Additional Faults **7**, 9
Local Network Server Queue **7**, 8
Local Network Wait Time in System **6**, 44
Localizing Faults **13**, 2
Locate the IP Phone **15**, 15
Location Awareness **15**, 12
Logic of Time Slicing **10**, 11
Logical Operators **9**, 13
Longest Stabilization Time of the Various Recovery Actions **15**, 9
Long-Running Application **12**, 4
Long-Term Scheduler **10**, 6
Loss of Data in Mobile Device Memory **16**, 6
Loss or Degradation of Wireless Connections **16**, 1
Loss of Memory Data **15**, 6
Low Failure Rate, Input Failures **15**, 9
Low Operating Times **16**, 12
Low Pass Filter **3**, 4
Low Pass Filter Subject to Error **3**, 14
Low Power Sleep State **15**, 12
Low Quality Connections **16**, 1
Low Severity **15**
Low Value of Probability of Detection **12**, 11
Lower Probability of Large Message Size **14**, 17
Lower S/N Means Higher Noise **16**, 16

Main Memory Hit Rate **1**, *12*
Main Memory Miss Rate **1**, *12*
Main Power Consuming Components **15**, 2
Main Processing Task **3**, 7
Maintainability **7**, 1
Maintainability Prediction **7**, 8
Maintainability Will Be Formulated as a Ratio of the Quantity of Data Processed by a Given Sequence of Nodes and Associated Links **7**, 8
Maintainability Will also Be Predicted Using Sequences **7**, 8
Major Issue in Software Reliability Assessment **13**, 9
Major Limitation of Portable Devices **15**, 2
Major Noise Contributors: Unstable Behavior and Self-Shutdown **16**, 10
Major Risks Posed by Mobile Devices **15**, 1
Majority of OS Kernel Failures Are Due to Memory Access Violation Errors **15**, 4
Making a Requirement Understandable **17**, 5
Manipulating Clock Rate to Achieve Required Response Time **4**, 16
Manipulating Images **15**, 6

Many Values of Reliability in a Sequence—One for Each Node or Link **7**, 6
Mapping Failures to Their Causes **11**, 9
Mapping Real World Objects to the O-O Model **17**, 5
Marginal Benefit Equals Marginal Cost **13**, 4
Marginal Increase in Test Time **13**, 3
Marginal Reduction in Failures and Faults **13**, 3
Masked Fault **12**, 4
Master Reset **15**, 5
Mathematical Concept of a Function Expresses Dependence Between Two or More Quantities **17**, 8
Mathematical Modeling Approach: Equations Suggest the Steps to Implement the Program **17**, 12
Mathematical Modeling Approach Has an Advantage **17**, 12
Mathematical Modeling Design Approach Example **17**, 13
Mathematical Modeling of Physical Objects **17**, 3
Mathematical Software Reliability **17**, 3
Mathematical Symbols for the Poisson Distribution **17**, 11
Mathematical Terms Could Be Cast in the Context of Developing a Failure Model **17**, 6
Mathematics for Software Reliability Models **17**, 2
Maximum Actual Reliability **13**, 7
Maximum Bandwidth **14**, 17
Maximum Likelihood Estimation (MLE) Method of Parameter Estimation **13**, 6
Maximum Operating Times That Can Achieved at Specified Values of Reliability **12**, 17
Maximum Permissible Response Time **4**, 2
Maximum Response Time Service Request **4**, 19
Maximum Response Time That Occurs Due to Resource Limitation **4**, 23
Maximum Sampling Frequency **3**, 11
Maximum Speed at Which the D/A (or A/D) Circuitry Must Operate to Reproduce the Correct Output **3**, 11
Maximum Value of the Severity Code **13**, 4
McCabe Test Strategy Does Not Provide Complete Coverage of All Code **17**, 17
Mealy and Moore Machines **1**, 57
Mean Access Point Processing Time in the Download Direction **6**, 24
Mean Access Point Processing Time in the Upload Direction **6**, 24
Mean Access Point Queue Wait Time in the Download Direction **6**, 25
Mean Access Point Queue Wait Time in the Upload Direction **6**, 24
Mean Difference Between Required and Achieved Response Times **4**, 23
Mean Download Processing Time **6**, 9
Mean Download Service Time **6**, 66
Mean Download Web Page Processing Time **6**, 6
Mean Error Rate **14**, 17
Mean Number of Bits Being Processed in the Download Direction **6**, 67
Mean Number of Bits Being Processed in the Upload Direction **6**, 66
Mean Number of Bits Waiting for Processing in the Download Direction **6**, 67
Mean Number of Bits Waiting for Processing in the Upload Direction **6**, 66
Mean Number of Packet Bits Being Processed for Name Translation **6**, 17
Mean Number of Packet Bits Being Processed for Upload Routing **6**, 15
Mean Number of Packet Bits Being Processed in the Upload Direction **6**, 10
Mean Number of Packet Bits Being Processed in the Upload Direction for Routing **6**, 13
Mean Number of Packet Bits Waiting for Name Translation **6**, 17
Mean Number of Packet Bits Waiting to Be Processed for Routing in the Upload Direction **6**, 13
Mean Number of Packet Bits Waiting to Be Processed in the Upload Direction **6**, 10
Mean Number of Packet Bits Waiting to Be Processed in the Upload Direction for Routing **6**, 12
Mean Number of Web Page Bits Being Processed in the Download Direction **6**, 11

Mean Number of Web Page Bits Being Processed in the Download Direction by the Access Point **6**, 25
Mean Number of Web Page Bits Being Processed in the Download Direction by the Internet Router **6**, 28
Mean Number of Web Page Bits Being Processed for Download Routing **6**, 15
Mean Number of Web Page Bits Being Processed in the Upload Direction **6**, 36
Mean Number of Web Page Bits Waiting to Be Processed in the Download Direction **6**, 11, 69
Mean Number of Web Page Bits Waiting to Be Processed in the Download Direction by the Access Point **6**, 26
Mean Number of Web Page Bits Waiting to Be Processed for Routing in the Download Direction **6**, 13
Mean Number of Web Page Bits Waiting to Be Processed in the Upload Direction **6**, 68
Mean Number of Web Page Bits Waiting for Routing in the Download Direction by the Internet Router **6**, 28
Mean Number of Wireless Packet Bits Being Processed in the Upload Direction by the Access Point **6**, 25
Mean Number of Wireless Packet Bits Being Routed in the Upload Direction **6**, 28
Mean Number of Wireless Packet Bits Processed for Translation by the DNS **6**, 30
Mean Number of Wireless Packet Bits Waiting for Routing in the Upload Direction by the Internet Router **6**, 28
Mean Number of Wireless Packet Bits Waiting to Be Processed in the Upload Direction by the Access Point **6**, 25
Mean Number of Wireless Packet Bits Waiting for Translation by the DNS **6**, 30
Mean Packet Upload Time **6**, 8
Mean Processing Time **6**, 16
Mean Processing Time for the DNS to Translate a Wireless Packet Name to an IP Address **6**, 29
Mean Relative Error (MRE) **11**, 9; **14**, 89
Mean Response Time Difference **4**, 19
Mean Sequence Reliability Values **7**, 6
Mean Square Error Between the Actual and Predicted Cumulative Failures **13**, 6
Mean Squared Error (MSE) Between Actual (Historical) and Predicted Reliability **12**, 13
Mean Time a Web Page Requested by a Wireless Packet Spends Being Processed for Routing by the Internet Router in the Download Direction **6**, 27
Mean Time a Wireless Packet Must Wait in the DNS Queue Prior to Name to IP Address Translation **6**, 29
Mean Time a Wireless Packet Spends Being Processed for Routing by the Internet Router in the Upload Direction **6**, 26
Mean Time a Wireless Packet Spends Waiting to Be Processed for Routing by the Internet Router in the Download Direction **6**, 27
Mean Time a Wireless Packet Spends Waiting to Be Processed for Routing by the Internet Router in the Upload Direction **6**, 27
Mean Time to Failure (MTTF) **13**, 16
Mean Upload Packet Processing Time **6**, 67
Mean Upload Packet Time **6**, 65
Mean Upload Processing Time **6**, 9
Mean Upload Service Time **6**, 66
Mean Upload Time from Buffer of Wireless Packet to DNS **6**, 29
Mean Upload Wait Time **6**, 9
Mean Value of Failure Rates **12**, 16
Mean Value of Parameter **12**, 17
Mean Wait Time **6**, 16
Mean Web Page Download Time **6**, 8
Mean Web Page Upload Wait Time **6**, 68
Mean Wireless Packet Upload Time **6**, 23
Measurable Increase in Reliability **14**, 1
Measured by the Difference Between the Correct Value and the Value Realized by D/A Conversion **3**, 11
Measures for Predicting Reliability **14**, 3
Measuring Current Risk **11**, 9
Measuring Prediction Accuracy **8**, 22
Mechanical Analog Computers **3**, 1

Mechanism for Sharing Context Information **15**, 13
Medium Severity When the Recovery Requires Reboot or Battery Removal **15**, 6
Memory **1**, *3*
Memory Capacity **16**, 3
Memory Enable Control Line **1**, 11
Memory Failure **12**, 14
Memory Management **10**, 1
Memory Management Problems **15**, 4
Memory-Mapping the RAM **1**, 14
Memory System Performance **1**, 12
Memory Violations Are the Cause of the Majority of Failures **15**, 7
Message Logging Algorithms **15**, 5
Message Processing Design **1**, 64
Message Transmission Rate as Queue Arrival Rate **16**, 3
Method: Operation on an Object That Is Part of the Declaration of a Class **17**, 11
Method for Analyzing Computer Program Reliability **9**, 5
Method of I/O Communication **3**, 7
Methods for Improving Reliability **8**, 23
Metropolitan Area Network (MAN) Distance Ranges Are 3–8 km **7**, 12
Microcomputer Clock Rate **10**, 9
Microcomputer Input-Output (I/O) Applications **3**, 7
Microprocessor Design **1**, 3
Microprocessor with Sufficient Speed (Clock Rate) to Satisfy Response Time Requirement **4**, 11
Mid-Term Scheduler **10**, 7
Migratory Service Incorporates All the State Information **16**, 24
Minimization of States **1**, 57
Minimum Acceptable Level of Coverage **17**, 17
Minimum Link Capacity **15**, 14
Minimum Data Rates of: 144 kbps in Vehicular Environment, 384 kbps in Pedestrian Environment, and 2 Mbps in Indoor Office Environment **7**, 13
Minimum Rate of Change over Successive Test Intervals **13**, 4
Minimum Response Time Service Request **4**, 19
Minimum Test Time **11**, 8
Mission Critical Application **12**, 6
Mission Critical Application Where The Reliability Must Be High **13**, 9
Mission Duration **11**, 2; **16**, 11
Mission Duration Where Predicted Reliability No Longer Achieves Specified Reliability **13**, 8
Mission Reliability Requirement **12**, 16
Mission Success **11**, 10
Mission Threatening Failures **11**, 10
Mitigate the Risk of Software Failure **12**, 12
Mix of Abstraction and Operational Detail Views **4**, 2
Mobile Ad Hoc Networks **15**, 1
Mobile Device **6**, 2
Mobile Device Context Awareness **15**, 12
Mobile Device Empirical Probabilities of Failure and Recovery **16**, 7
Mobile Device Failure **15**, 5
Mobile Device Failure Characteristics **16**, 6
Mobile Device Is Subject to an Input Failure **16**, 16
Mobile Device Manufacturers Should Improve the Quality of Their Recovery Action Software **16**, 16
Mobile Device Model Must Account for Context and Migration **16**, 24
Mobile Device Operating Time **16**, 11
Mobile Device Performance **15**, 1
Mobile Device Reliability **15**, 3; **16**, 1
Mobile Device Reliability Is Only Satisfactory for the First Few Months of Operation **15**, 16
Mobile Device Reliability Model **16**, 7
Mobile Device Scenario **15**, 12
Mobile Device Software Development Process **16**, 16
Mobile Device Software Reliability **15**, 1
Mobile Device Software Reliability and Testing **16**, 1
Mobile Device Tasks Are Susceptible to Errors **16**, 3
Mobile Device Testing Effectiveness **16**, 12
Mobile Devices Are Multimedia-Enabled **15**, 14
Mobile Devices Did Not, in General, Meet Requirements **16**, 17

Mobile Devices Have Unique Characteristics **16**, 3
Mobile Devices Need Improved Reliability Even After Responding to a Failure **16**, 16
Mobile Devices Operate in a Hostile Communications Environment **16**, 1
Mobile Devices Should Be Improved so That They Are Really Usable by Customers **16**, 17
Mobile Environment Involves Many Software and Hardware Components and Technologies **15**, 1
Mobile Memory Cache **15**, 18
Mobile Meter Reader Can Be Automatically Connected to an Operational Sub Station **16**, 24
Mobile Network Data **15**, 15
Mobile Network Reliability, Performance, and Context and Network Awareness **15**, 1
Mobile Network Stability **15**, 1
Mobile Networks Can Adapt to Changing Conditions **15**, 16
Mobile Networks Could Respond to Changes in Both Context and Network Awareness **15**, 1
Mobile Phone Failure Data **15**, 1
Mobile Phone Failure Data Reported **16**, 1
Mobile Phone Performance Assessment **15**, 15
Mobile Phone Software **15**, 15
Mobile Process Can Involve Context Aware Migratory Tasks **16**, 12
Mobile Scenarios **15**, 12
Mobile vs. Non-Mobile Applications **15**, 12
Mobility-Related Reliability **16**, 1
Model (e.g., Poisson Failure Model) **17**, 12
Model: Representation of Objects, Functions, Limits, Parameters, Variables, and Equations **17**, 5
Model-Based Software Development Has Been Shown to Be a Promising Approach to Real-Time Design Problems **4**, 2
Model Capable of Representing Both Sequential and Concurrent Interactions Between Objects **17**, 3
Model Combines Hardware, Software, and Their Interactions **12**, 13
Model Development **8**, 6
Model Is Provided for Developing Test Strategies **17**, 20
Model Limitations **16**, 10
Model the Problem Being Solved **3**, 1
Model of Reliability Based on the Signal to Noise Ratio **16**, 7
Model the Reliability of Web Systems **14**, 1
Model States, Events, Actions, and State Transitions **17**, 10
Model Tasks **8**, 2
Model Uses Multiple Executing Servers, Each Processing User Requests Concurrently **6**, 2
Modeled Using Equations **3**, 1
Modeling Method Has Two Major Components **17**, 12
Modeling Path Maintainability and Availability **9**, 8
Modeling Reliability and Testing **16**, 3
Modeling Various Failure Rate Patterns **12**
Models Are Built to Seek an Understanding of the Requirements or to Specify the Systems to Be Built **17**, 4
Models Are Evolved by Learning **13**, 1
Models of Interconnection Structure **13**, 1
Models of Neurons **13**, 1
Modularity: Produce Modules That Have Well-Defined Functions and Interfaces That Can Easily Interconnect **4**, 4
Modularize Complex Programs and Make the Maintenance and Understanding of such Programs Easier **17**, 5
Module Device Failure Data Generated from Test Results **16**, 12
Module Failed **15**, 5
Modules with Failure Types Associated with Negative Feedback Should Receive Priority Attention **16**, 10
Monitor the Environment **15**, 12
Monitoring Risk Status **11**, 9
Monotonicity **3**, 11
More Active Users lead to Fewer Available Communication Timeslots **16**, 4
Most Important Factor Is the Quality of the Personnel Developing the Software **17**, 19

Movement Away from the Servers Currently in Use, and Toward New Ones **16**, 3
MTTF Is Well Understood in the Software Industry **13**, 16
Multimedia Services **7**, 12
Multiple Access Contention **15**, 3
Multiple Applications Are Concurrently Executed **15**, 14
Multiple Cell Phone Users at Various Locations **15**, 6
Multiple Output Combinational Circuits **1**, 24
Multiple Processors **12**, 4
Multiple Threads of Control Caused by Concurrent Inputs **4**, 4
Multiplexers **1**, 36; **3**, 1
Multiplexing Data and Address Signals **1**, *13*
Multiplying Component Reliabilities **12**, 4
Multi-Processor Technology **12**, 7
Myriad of Failures **14**, 3

N Number of Operations on the Web Page **14**, 16
NAND **1**, 15
NASA Space Shuttle Flight Software OI6 Failure Data **17**, 13
NASA Space Shuttle Operational Increment **11**, 3
NASA Space Shuttle Software System **13**, 5
Nature of Web Service Client **14**, 3
Negative Feedback Is Needed to Correct Modules **16**, 10
Negative Reliability **16**, 10
Network Application **5**, 4
Network Architecture **5**, 1
Network-Aware Applications **15**, 14
Network Awareness Approach **15**, 16
Network Connections **15**, 12
Network Connection Types of the Device **15**, 13
Network Connectivity and Locations **16**, 3
Network Efficiency **7**, 1
Network Failures **14**, 5
Network Firewall **15**, 1
Network Is Trained **13**, 3
Network Metrics **8**, 1
Network Performance Evaluation Model **7**, 2
Network Performance Parameters Data **5**, 13
Network Performance Variations **15**, 14
Network Standards Do Not Address the Software Compatibility Issue **7**, 9
Network Standards Should Focus on Local Networks **7**, 6
Network Times **8**, 3
Network Usage Data **6**, 3
Network Users Must Insist on Receiving Compatibility Information **7**, 9
Networked Data Storage Facilities **15**, 15
Networking Interfaces **15**, 15
Networks Learning from a Teacher **13**, 18
Neural Network and Parameter Evaluation Methods **13**, 9
Neural Network Criterion **13**, 11
Neural Network Criterion Limit **13**, 5
Neural Network Criterion Method Involves Lower Reliability Risk at Higher Values of Reliability **13**, 9
Neural Network Learning **13**, 2
Neural Network Prediction Criterion **13**, 18
Neural Networks **13**, 1
Neural Networks Applied to Fault Localization **13**, 3
Neural Networks Applied to Software Reliability Assessment **13**, 3
New Critical Applications Emerge for Mobile Phones **15**, 3
New Requirements Could Introduce Performance Problems **15**, 15
New Security Threats **15**, 1
New Specifications to Provide More Bandwidth or QoS-Related Parameters and Interfaces **7**, 12
No Comparable Features of the O-O Approach **17**, 3
No One Size Fits All Solution to the Problem of Selecting the Appropriate Software Development Paradigm **17**, 18
No Test Strategy Is Perfect, Including McCabe's **17**, 17
Node and Link Operating Times **8**, 7
Node and Link Sequences Consistency **7**, 9
Node and Link System Reliability **14**, 9
Node Probability of Being Busy **5**, 10

Nodes (Program Functions While, If, Else, Set, Read, Write, Store, and Compute) **17**, 13
Nodes and Links **8**, 2
Nodes and Their Associated Links Process the Same Quantity of Data **7**, 8
Noise **3**, 5
Noise (# of Failed Modules) **16**, 7
Noise Could Be Represented by Number of Unsuccessful Web Search Results **16**, 10
Noise Suppression Process **16**, 10
Non-Compatibility Result Is Recorded If the Signal Is Not Received **7**, 14
Non-Functional Properties **14**, 2
Nonlinearity Distortion **3**, 12
NOR **1**, 17
NOT **1**, 15
Not Capable of Properly Representing Concurrent Interactions **17**, 3
Novel Signal to Noise Ratio **15**, 1
Number of Components in a System **12**, 4
Number of Components That Do Not Fail **12**, 8
Number of Correct Modules (Signal) **16**, 8
Number of Correct Modules Based on Recovery Action **16**, 9
Number of Correct Software Modules **16**, 1
Number of Days Since the Software Was Released by the Contractor to NASA **17**, 13
Number of Edges (Branches) and Number Of Nodes (Statements) in the Directed Graph Representation of a Program **17**, 17
Number of Encoding Bits **3**, 6
Number of Errors on a Web Page **14**, 16
Number of Failed Components **12**, 9
Number of Failed Modules (Noise) **16**, 1
Number of Failed Software Modules **16**, 8
Number of Failure Counts **12**, 15
Number of Failures **11**, 6; **12**, 11
Number of Failures Expected in Node or Link **7**, 5
Number of Failures in Test Interval **13**, 12
Number of Failures Remaining **11**, 8
Number of Hard Disk Accesses **1**, *12*
Number of Independent Paths in a Computer Program **17**, 2
Number of Independent Paths Interconnecting Web Clients with Web Servers **14**, 2
Number of Instructions Executed **10**, 9
Number of Page Transfers from Secondary Storage **10**, 13
Number of Possible Output Levels the D/A Is Designed to Reproduce **3**, 10
Number of Processors **12**, 4
Number of Processors That Do Not Fail **12**, 5
Number of Programs Allocated Time Slices **10**, 4
Number of Responses to Service Requests Required in Operational Time **4**, 19
Number of Sessions **14**, 11
Number of User Web Sessions **14**, 11
Number of Ways Web Clients and Servers Could Fail **14**, 22
N-Version Redundancy Model for Web Services **14**, 2
Nyquist–Shannon Sampling Theorem **3**, 11

Object **12**, 13
Object: In a Computer Program, Any Entity That Can Execute in a Computer **17**, 1
Object: The Focus of Attention **17**, 5
Object-Based Design Is Suitable for Distributed, Parallel, or Sequential Implementation **17**, 4
Object Interrupt Processing Would Include Instructions for Processing an Interrupt **17**, 2
Object-Math, Which Is a High-Level Programming Environment with a Modeling Language **17**, 3
Object Math Focuses on Mathematical Modeling Rather Than Object-Oriented Programming **17**, 3
Object Math Language Augments Mathematica with Classes and Other Object-Oriented Language Constructs **17**, 3
Object-Oriented Analysis and Design Is Quite Appropriate **17**, 19
Object-Oriented Approach vs. Mathematics **17**, 3
Object-Oriented Design **9**, 4

Object-Oriented Diagrams Are Useful for Providing High-Level Visibility of Computer Program Structure **17**, 1
Objects **9**, 4
Objects Correspond More Closely to the Products and Processes in the Conceptual Worlds of the Designer and User **17**, 4
Objects Failure Count, Failure Time, and Failure Rate **17**, 12
Objects Have Two Characteristics: State and Behavior **17**, 8
Objects in the Elevator System Are User, System Controller, System Storage, Operations, and Error Control **4**, 3
Objects also Possess Attributes **17**, 8
Observed Failure Count **13**, 15
Observed Failure Data **13**, 4
Observing How an Elevator Operates (e.g., Processing Service Requests) **4**, 2
On-Demand Movie Streaming and Video Conferencing **15**, 2
Only One Failure Type – Input – Has Acceptable Reliability **16**, 12
O-O Approach: Sequence Diagram Can Be Used to Identify Program Steps **17**, 12
O-O Approach Is Compatible with Developing an Elevator System or Web Site **17**, 3
O-O Approach Will Be Used to Model Facets of Software Reliability Models **17**, 3
O-O Can Be Effective for Reengineering **17**, 4
O-O Can Be Used for Transforming the States of Legacy Software to O-O Software **17**, 4
O-O Concepts Applied to Poisson Failure Model **17**, 8
O-O Is Highly Abstract **17**, 3
O-O Paradigm: Data Is Sometimes Relegated to an Obscure Role That Has Nothing to Do with the Collection and Processing of Raw Data **17**, 13
O-O Provides Various Views of a Software System That Are Useful for Understanding and Maintaining Code **17**, 4
O-O Representation of the Poisson Failure Model **17**, 8
Open Standards Are More Popular Than Proprietary Ones **7**, 10
Open Standards Are Useful for Helping to Mitigate the Problem of Software Compatibility **7**, 10
Operate Faster, More Reliably, and with Improved Security **6**, 1
Operate in Multiple Environments **7**, 13
Operating Characteristics of a Component **12**, 2
Operating Context (OC) **15**, 13
Operating Environmental Conditions **13**, 1
Operating and Security System Property **17**, 4
Operating System **1**, 10
Operating System Architecture **10**, 1
Operating System Characteristics **10**, 6
Operating System Issues **10**, 1
Operating System Performance Evaluation **10**, 4
Operating System Reliability Evaluation **10**, 5
Operating Time **12**, 3; **14**, 11
Operating Time during Which a Specified Reliability Requirement Is Achieved **15**, 3
Operating Time during Which a Specified Reliability Requirement Is to Be Achieved **16**, 11
Operating Times **12**, 9
Operational Amplifier **3**, 3
Operational Amplifier Amplification Capability **3**, 5
Operational Amplifier May Fail to Produce a Correct Amplification of the Signal Produced by the Sensor Output **3**, 13
Operational Mode **12**, 2
Operational Phase **12**, 3
Operational Requirement **12**, 4
Operational Time **4**, 19; **15**, 1
Operational Usage **11**, 2
Operations in the Real World Network **7**, 6
Operations Schedule **4**, 3
Operations That Must Meet Deadlines **4**, 1
Opportunities to Improve Personalized Applications **15**, 12

Optimal Amount of Test Time **13**, 7
Optimal Combination of Methods **17**, 19
Optimal for Terminating Testing and Releasing the Software System **13**, 5
Optimal Web Service **14**, 1
Optimistic Message Logging Has a Lower Failure-Free Operation Cost **15**, 5
Optimize the Performance and Cost According to Different Requirements **7**, 11
OR **1**, 16
Output Data **9**, 4
Output Failure **15**, 4; **16**, 6
Output Failures Are Low Severity **15**, 6; **16**, 6
Output Failures Are Often Due to a Temporary Software Corrupted State **15**, 6; **16**, 7
Output Processing State **4**, 11
Outputs **13**, 1
Outputs That Result from Each Step **17**, 19
Overall Failure Rate **14**, 16
Overall Sequence Reliability Metric **7**, 1
Overhead of Algorithms **12**, 13
Overhead Induced by a Multiplicity of Protocols, Intermediate Networks, and Interfaces **6**, 1

Packet Bits Translated from Name to IP Address by the DNS and Waiting for Translation **6**, 49
Packet Length Bits Being Routed and Waiting for Routing by Local Network Router **6**, 45
Packet Lengths Being Processed and Waiting for Processing by Local Network **6**, 44
Page Cost **10**, 13
Page Transfer Time **10**, 13
Paging Rate **10**, 13
Paradigm Transforms the Internet **14**, 1
Paradigm of Web Services **14**, 1
Parallel and Combined Series-Parallel Reliability Models **12**, 15
Parallel Component **12**, 1
Parallel Hardware Redundancy **12**, 12
Parallel Multi-Processor-Memory Systems **12**, 7
Parallel Redundancy **12**, 5
Parameter: A Model Numerical Factor Estimated from Data **17**, 5
Parameter Can Be Tuned **15**, 15
Parameter Estimation Methods **13**, 4
Parameters **12**, 15
Parameters of Network Quality **15**, 14
Parameters of the Prediction Model **13**, 4
Parameters of the Weibull Distribution **14**, 7
Parity Error Detection **2**, 8
Path **8**, 2
Pattern Recognition **13**, 1
Patterns of Failure Data **14**, 6
Patterns of Probability of Failure Across Systems **14**, 6
Peak Data Rates May Only Be Realized in Favorable Channel Conditions **7**, 12
Perfect Mathematical Programming Environment Would Automatically Transform Systems of Equations into Efficient Symbolic and Numerical Programs **17**, 2
Perform Tests Designed to Ensure the Networks Adhere to Proposed and Existing Standards **7**, 14
Performance **1**, 6; **15**, 1
Performance Analysis of Proposed Future Wired Internet **6**, 65
Performance Assessment **15**, 15
Performance Attribute of a Mobile Network **15**, 15
Performance Evaluation **14**, 1
Performance Is an Important Quality Attribute of a Software System **15**, 15
Performance Methodology **7**, 13
Performance Metrics **4**, 23
Performance and Reliability of Present and Proposed Internets **6**, 1
Performance of Web Browsing **14**, 4
Performs Ancillary Services on Behalf of the Web Server **14**, 4
Period Is the Duration of One Cycle in a Repeating Event **3**, 11
Period Is the Reciprocal of the Frequency **3**, 11
Periodic Real-Time Independent Tasks with Known Periods and Worst Case Execution Times **4**, 3

Periods of Operation and Execution Times That Are Driven Asynchronously by Inputs That Occur at Unpredictable Times **4**, 3
Permanent Connections Are Not Guaranteed **15**, 12
Permanent Hardware Failure **12**, 14
Personal Area Network (PAN) Distance Ranges Are 10 Meters **7**, 12
Personal Mobile Devices Will Become Ingredients of Other Infrastructures, such as the Electric Grid **16**, 24
Personnel Management Risks **11**, 2
Phase of Development **17**, 18
Phase Distortion **3**, 11
Phase Distortion Is Measured by the Difference Between the Correct Phase and the Phase That Is Reproduced at the Output of the D/A Converter **3**, 11
Phase of the Voltage Sine Wave Sensed in the A/D Converter May Not Be Faithfully Reproduced at the Output of the D/A Converter **3**, 11
Phone Failure Affecting the Application **15**, 3
Phone Freezes Are Medium Severity **15**, 6
Phone Freezes Are Medium Severity **16**, 6
Phone Often Does Not Respond to the Power On/Off Button **15**, 5
Phone's Activity **15**, 5
Phones Are Discarded Every 18 Months **15**, 7
Physical Mobile Device System **16**, 1
Piconet Consists of One Master Node and up to 7 Slave Nodes **7**, 13
Pipeline Efficiency **1**, 9
Pipeline System Delay **1**, 9
Pipelined Systems **1**, 8
Planned Mission Duration **11**, 4
Planned Risk **11**, 9
Platform-Independent Design **4**, 2
Plot of the Output Signal vs. the Input Signal Is Not a Straight Line **3**, 11
Pointer to the First Interrupt Processing Instruction in Memory **17**, 2
Poisson Failure Count Model **12**, 4
Poisson Failure Model Class Diagram **17**, 11
Poisson Failure Model Object Is Executed (State) **17**, 8
Poisson Failure Model Process Activity Diagram **17**, 19
Poisson Failure Occurrence Model and Its Associated Function, Reliability **17**, 6
Poisson Probability of Failure **12**, 9
Poor Reliability and Performance **15**, 3
Position of Clients and Servers **14**, 2
Possibilities for Saving Power **15**, 2
Power Loss **15**, 2
Power Monitoring **15**, 2
Power Disruptions **3**, 9
Power Saving on Mobile VOIP Devices **15**, 12
Power Requirements, Distance Ranges, Data Rates, and Carrier Frequencies **7**, 11
Power Usage Reported at the Failed Sub Station Before It Failed **16**, 24
Practical Methods for Designing and Evaluating Network Standards **7**, 1
Practice of Postponing All Consideration of So-Called "Platform Issues" Until the Application Logic of the Software Has Been Satisfactorily Designed **4**, 2
Precise Description and Common Understanding of the Semantics, as well as the Relations Between the Various UML Diagrams for the Description of Software Systems, Is Missing **17**, 4
Precision of a Digital Computer **3**, 6
Precision of the Readout Equipment **3**, 6
Predict Component Reliabilities **14**, 22
Predict the Operating Time-Oriented Reliability of Software, Hardware, and System **14**, 12
Predict Reliability at the End of the Mission Duration **13**, 8
Predict Reliability over a Specified Operating Time of the Mobile Device **16**, 11
Predict Reliability Risk **13**, 13
Predict the Required Failure Rate That Is Necessary to Achieve Required Reliability **14**, 11
Predict Software Reliability **17**, 1
Predict the Test Time Required to Achieve Specified Reliability **13**, 10
Predict Total Web System Reliability **14**, 4

Predicted Cumulative Failures **13**, 3
Predicted Quantity **13**, 12
Predicted Reliabilities into Conformance with Required Reliability **14**, 12
Predicted Reliability **11**, 1
Predicted System Reliability **12**, 15
Predicted Time to Failure **11**, 2
Predicted Value Is Less Than the Mission Duration **13**, 15
Predicting Client Side Probability of Failure **14**, 3
Predicting Cumulative Failures **13**, 5
Predicting the Duration of Operating Time **12**, 4
Predicting Sequence Storage Capacity Requirements **7**, 9
Predicting Total Web System Reliability **14**, 4
Prediction Accuracy **13**, 6
Prediction Accuracy of Software, Hardware, and System **14**, 10
Prediction Error **12**, 16; **13**, 13
Prediction Less Than the Mission Duration Poses a Risk **13**, 15
Prediction Lower Error **13**, 8
Prediction Results **11**, 8
Predictions as a Function of Web Server Operating Time **14**, 11
Predictive Validity **12**, 11
Principles of Real-Time System Design **4**, 25
Present Internet Wired Logic Sequences for Upload and Download **6**, 8
Present Internet Wireless Backbone **6**, 23
Present Wired Internet System **6**, 62
Present Wireless Internet System **6**, 63
Prime Implicant **1**, 21
Prioritize the Components for Reliability Improvement **14**, 18
Probabilities **8**, 4
Probabilities of Failure **8**, 8
Probabilities of a Recovery Action, Given a Failure Type, Are Independent **16**, 7
Probabilities of State Transitions **14**, 9
Probability of Access Point Being Busy **6**, 24
Probability of Additional Failures **12**, 19
Probability of Being Busy **6**, 66
Probability of Completing Service Request **4**
Probability Density Function **17**, 5
Probability of Domain Name System Being Busy **6**, 16
Probability of Failure **12**, 4; **15**, 6
Probability of Failure Across the Failure Types **16**, 8
Probability of Failure Analysis Results **8**, 18
Probability of Failure Has Stabilized **14**, 11
Probability of Failure Metric **12**, 18
Probability of Failure Model **14**, 16
Probability of Failure of Web Server Systems **14**, 11
Probability of Failures Occurring **12**, 4
Probability of Internet Router Being Busy **6**, 13
Probability of Internet Router Being Busy Processing Wireless Packet **6**, 26
Probability of Local Network Router Being Busy **6**, 11
Probability of Multiple Component Failures **12**, 9
Probability of Multiple Failed Components **12**, 9
Probability of the Next State in a Process Is Only Dependent on the Present State **15**, 6
Probability of One or More Failures **15**, 7
Probability of Queue Being Busy **6**, 8
Probability of Recovery Action **16**, 7
Probability of a Specified Number of Failures Occurring **12**, 11
Probability Was the Ratio of the Quantity of Data Processed by a Given Link or Node to the Total Quantity of Data Processed at All Links and Nodes in the Network **7**, 8
Probability of Web Server Being Busy **6**, 30
Problem of Maintaining Adequate Power in a Mobile Network **15**, 1
Problem Occurs When, e.g., a 1 Volt Difference in the A/D Converter Does Not Result in a 1 Bit Problem Occurs When the Original Phase of a Signal in the Input of The A/D Converter Is Not Faithfully Reproduced in the Output of the D/A Converter **3**, 11
Difference in the Digital Encoding **3**, 11

Problem Representativeness in Programming Languages **9**, 5
Problem in System Design Is the Appropriate Allocation of Functions Between Software and Hardware Design **4**, 6
Problem Was Transient **15**, 5
Problems with Current Generation Wireless Technologies **16**, 2
Procedure (e.g., the Steps in Implementing the Poisson Failure Model) **17**, 12
Procedure: Portion of Software Code within a Software Program **17**, 2
Procedure Calls **12**, 13
Procedure Consists of a Number of Steps Required to Construct the Model **17**, 12
Process-Based Web Server Architecture **6**, 2
Process Focuses on the Most Likely Communication Paths **14**, 2
Process Starts by Defining the Network Topology for Present and Future Internet Configurations **6**, 1
Process That Is Used to Support Web Server Operations **14**, 6
Processing and Communication Ability **16**, 3
Processing Method **14**, 6
Processing Performance in Internet Routers **14**, 2
Processing Power **13**, 3
Processor Scheduling Policies **12**, 12
Product Parameters **15**, 3
Productivity-Enhancing Tools **15**, 1
Profile Criterions **11**, 3
Program Counter **1**, 4
Program Debugging **13**, 3
Program Execution **13**, 3
Program Execution Results **17**, 13
Program Failure **13**, 3
Program Language Characteristics **9**, 12
Program Objective **9**, 10
Program Statement Coverage of a Test Case **13**, 3
Programmable Web Application **14**, 1
Programming Language Statements: Statements That Implement a Model on a Computer **17**, 5
Programming Language That Supports Exceptions **17**, 17
Propagation of Error **13**, 2
Properties Are Expressed Sans User Perspective **7**, 1
Properties of the Proposed Future Internet **6**, 2
Properties of an XHTML Web Page **14**, 3
Proposed Fundamental Changes in the Internet Configuration **6**, 1
Proposed Wired Internet System **6**, 64
Proposed Wireless Internet System **6**, 64
Prospective Development Personnel Design and Program Small, Representative Samples of the Real System **17**, 19
Provide Better Code Visibility **17**, 4
Provide Frequently Used Operations That Can Be Accessed by Many Programs or from Many Points within a Single Program **17**, 5
Providing Real-Time System Design Abstractions **4**, 25
Providing Sufficient Specificity for Designing Application-Specific Systems **4**, 25
Proxy Is a Computer **14**, 4
Proxy Is Very Effective in Reducing Response Time **14**, 4
Public Land Mobile Networks **7**, 12
Publicly Available Performance and Reliability Data Posted on the Internet **6**, 1
Pure Parallel Configuration **12**, 6
Pure Series **12**, 5
Purpose Activity Diagram Is to Used to Model the Procedural Flow of Actions in a System **17**, 9
Putting Software and Hardware Design in Separate Bins Is a Big Mistake Because the Operations of Software And Hardware Are Intimately Related **4**, 6

Qualitative Assessment **12**, 11
Quality of Communication **15**, 3
Quality of Communication Between Mobile Devices and Between Mobile Devices and Mobile Network **16**, 1
Quality of Communication Lines **14**, 2
Quantitative Approach **15**
Quantitative Assessment of Risk **11**, 4
Quantitative Assessments of Mobile Network Reliability **15**, 1

Quantizing Analog Signal **3**, 5
Quantizing Error Is the Inverse of the A/D Quantizing Error **3**, 11
Quantizing Errors **3**, 5
Quantizing Step Size for Analog to Digital Conversion **3**, 6
Question the Utility of Existing Standards **7**, 1
Queue Efficiency **6**, 49
Queuing Model (Present Internet System) **6**, 6
Quickly Develop and Deploy Web Applications **14**, 1
Quine McCluskey Method **1**, 21

Radio Frequency (RF) Interference **15**, 3
Random Access Memory **1**, 10
Random Hardware and Software Failures **12**, 1
Random Number Generator **14**, 9
Range **7**, 1
Range of Access Points **15**, 12
Range Limitations **3**, 5
Range Required by Mobile Device in Wireless Network **7**, 14
Range Where the Converted Voltage Is Either Too High or Too Low **3**, 12
Rank Quality for the Purpose Prioritizing the Test Effort **17**, 17
Rapid Development **17**, 4
Rate of Change of Actual Cumulative Failures Is Minimum **13**, 5
Rate of Change Between Reliability and Test Time **13**, 2
Rate of Change of a Function **13**, 2
Rate of Change Is Minimum **13**, 2
Rate of Change of Reliability **14**, 16
Rate of Change of Web Client and Server Predicted Reliability **14**, 25
Rate of Data Transfer **15**, 14
Ratio of the Change in Noise (Number of Corrected Failed Modules) to the Total Number of Modules **16**, 12
Ratio of Reliability (Signal) to Unreliability (Noise) **15**, 15
RC Circuit **3**, 4
Read Failure Count **17**, 11
Read Only Memory **1**, 10
Read/Write Control Line **1**, 11
Real-Time Control Hardware and Software Has Been Applied to a Wide Variety of Real-World Systems **4**, 1
Real-Time Module Topology Is Essentially Flat **4**, 4
Real-Time Operational Mode **12**, 13
Real-Time Scheduler Scheduling Efficiency **10**, 10
Real-Time Software Has to Satisfy a Set of Stringent Nonfunctional Requirements **4**, 2
Real-Time Software Is Particularly Difficult to Design **4**, 2
Real-Time System **12**, 12
Real-Time System Hierarchies Are Rare Or Non-Existent **4**, 4
Real-Time System Properties **4**, 1; **5**
Real-Time System Requirements **4**, 5
Real-Time System Scheduling **10**, 10
Real-Time Systems Are Comprised of Heterogeneous Components Including Sensors, Microprocessors, and Actuators **4**, 1
Real-Time Systems Are One-of-a-Kind; They Are Not Mass Produced **4**, 4
Real-Time Systems Do Not Have the Luxury of Inputting Data When Convenient for the Microprocessor **4**, 12
Real-World Failure Data **14**, 1
Reboot **15**, 5; **16**, 5
Rebooting Occurs Only in 2.36% of the Freeze Failures **16**, 6
Reboots Are an Effective Way to Recover from Output Failures **15**, 6; **16**, 7
Received Signal Strength (RSS) **15**, 12; **16**, 1
Receiver Sensitivity in the Phone **15**, 15
Recover from Faults **12**, 5
Recover the Message from the Centralized Mobile Station **15**, 5
Recovered by Simply Waiting for the Phone to Respond **15**, 6
Recovering from Faults **12**, 5
Recovery Action **15**, 1
Recovery Action Characteristics **16**, 5
Recovery Action Reliability Lags the Input Failure Reliability **16**, 16
Recovery Action Testing Takes Less Time Than Failure Type Testing **16**, 17

Recovery Action Types Below The Limit Need Attention to Identify Why the Recovery Software Is Not Able to Provide Effective Recovery **16**, 8
Recovery Actions Are Deficient **16**, 16
Recovery Actions Produce the Highest Reliability **16**, 9
Recovery Type Testing Is Based on Test Cases Under the Control of the Tester **16**, 17
Reduce the Failure Rate to a Value That Will Achieve the Required Reliability **14**, 11
Reduced Failure Rates **14**, 12
Reduced Instruction Set Computing **1**, 6
Reduced Instruction Set Computing (RISC) Architecture Requires Several Operations to Execute a Single Instruction **4**, 1
Reduced Noise Accomplished Through Testing of Failure Type **16**, 12
Reduced Number of Failed Modules **16**, 12
Reducing the S/N of the Wireless Communications **15**, 3
Reduction in Number of Failed Modules **16**, 14
Reduction in Web Server Faults **14**, 12
Redundancy **12**, 1
Refining Predictions **11**, 9
Register **1**, 1
Regularity: Find Modules with Common Functions (i.e., Interchangeable Parts) **4**, 4
Relationship Between Failures, Recovery Actions, and Reliability **16**, 7
Relationship among Inputs, Flip-Flops, and Output States **1**, 54
Relationship Between Inputs and Outputs **13**, 2
Relationship Between Objects Must Be Designed with Great Care **17**, 12
Relationship Between O-O Attributes and the Modeling of Physical Systems **17**, 1
Relationship Is Not Obvious for Modeling Mathematical Software **17**, 1
Relevance of Search Results **14**, 20
Reliabilities **8**, 4
Reliability **2**, 4; **6**, 1; **7**, 1; **11**, 1; **13**, 1; **14**, 1; **15**, 1
Reliability Activation Function **13**, 7
Reliability Analysis Based on Web Systems **14**, 10
Reliability Analysis of GPRS Has General Applicability **16**, 4
Reliability Analysis Results **8**, 20; **12**, 1
Reliability Based on Recovery Action Type **16**, 9
Reliability Benefit-Cost Ratio **12**, 1
Reliability Calculations **15**, 6
Reliability of the Connection and the Performance of the Device **16**, 3
Reliability Criterion Limit **13**, 7
Reliability Decreasing with Operating Time **16**, 11
Reliability, Derived from the Poisson Failure Model, Is Reasonable Based on the Software That the Model Represents **17**, 17
Reliability Estimation **13**, 1
Reliability Evaluation **12**, 9; **13**, 7
Reliability as a Function of Module Count **16**, 11
Reliability as a Function of Recovery Action Type **16**, 9
Reliability Goal **11**, 1; **12**, 16; **11**, 1; **14**, 1
Reliability Growth **16**, 11
Reliability Has Been Predicted Using Network Entities Called Sequences **7**, 8
Reliability of Individual Components **12**, 5; **14**, 18
Reliability Is Assessed Following a Successful Time Test by Using Test Software That Computes the Required Reliability **7**, 13
Reliability Is Based on the Sequence Failure Rate and the Node or Link Times **7**, 6
Reliability Is Not an Additive Function **13**, 7
Reliability Is the Primary Metric **14**, 11
Reliability Limit Criterion **13**, 7
Reliability Logic **12**, 2
Reliability, Maintainability, and Availability That Are Not Quantified in Existing Standards **7**, 15
Reliability of a Mobile Device **16**, 7
Reliability of a Mobile Device Will Decrease Nonlinearly with the Amount of Interactive Processing **16**, 10

Reliability Model Based on the Signal to Noise Ratio **16**, 8
Reliability Model Parameter Estimation Method **13**, 18
Reliability Model for Worst Case Analysis **16**, 4
Reliability Modeling **12**, 12
Reliability Models for Assessing Mobile Network Reliability **15**, 1
Reliability Models Can Be Refined **12**, 13
Reliability Objective **13**, 4
Reliability of a Parallel Configuration **12**, 12
Reliability Parameter **13**, 7
Reliability Performance **14**, 18
Reliability Prediction **7; 13**, 3
Reliability Prediction Models for Assessing the Software, Hardware, and System Quality of a Web System **14**, 2
Reliability Prediction Process **7**, 3
Reliability Predictions Influence the Selection of Test Cases **16**, 12
Reliability Required to Achieve the Mission Duration **13**, 9
Reliability Requirement **11**, 4
Reliability Risk **13**, 9
Reliability Risk Criteria **11**, 4
Reliability of a Series Configuration **12**, 3
Reliability of Web Services **14**, 1
Reliability of Web System Communication and the Interconnected Components **14**, 2
Reliability When n Out of N Components Fail **12**, 4
Reliable Application Software **16**, 3
Reliable Messaging Technology of Web Services **14**, 2
Remaining Failures **11**, 1; **13**, 12
Remaining Failures Analysis Results **8**, 20
Remaining Failures Criterion **11**, 8
Remaining Failures Is a Decreasing Function **13**, 12
Remaining Failures Requirement **11**, 9
Remaining Failures Risk **11**, 8
Remaining Faults and Failures **8**, 5
Remove Battery **15**, 5; **16**, 6
Remove Faults **14**, 11
Repeat the Action **15**, 5; **16**, 6
Repeat the Execution of an Equation **17**, 3
Repeating the Action Is Often Sufficient to Restore a Correct Device Operation **15**, 6; **16**, 6
Reported Failure Data **15**, 3
Repository of Data **14**, 1
Repository of Services **14**, 1
Represent Failure Severity in the Computation of Expected Number of Failed Modules **16**, 9
Represent the Generic View of the Application Design **4**, 25
Representative Failure Data **14**, 5
Representative of the Web Environment **14**, 5
Requests for Web Pages **7**, 9
Required Changes Are Localized **17**, 4
Required Failure Rate Reduction **12**, 16
Required Reliability **11**, 2; **14**, 11
Required Reliability at a Reasonable Cost **12**, 16
Required Reliability Is Satisfied for Only a Limited Range of Operating Time **14**, 18
Required Response Time **4**, 19
Required Sampling Frequency Is the Desired Signal Frequency Emanating from the Input Analog Voltage **3**, 14
Requirement Implementation **17**, 5
Requirement Management Risks **11**, 2
Requirements **11**, 4
Requirements for an Architecture to Support Context-Awareness **15**, 12
Requirements such as Reliability Specifications and the Means for Testing Reliability, Are Largely Absent from Current Standards **4**, 1
Residual Failures **11**, 3
Residual Faults **11**, 2
Residual Problems **13**, 12
Resolution **3**, 10
Resolution Error Is Determined by the Smallest Change That Can Be Detected at the Sensor Output **3**, 13
Resource Usage and Performance Risks **11**, 2
Resource Utilization Is Increased **15**, 14
Resources Needed for the Object to Function **17**, 2
Response Time **10**, 6

Response Time: Difference in Time Between Completion of Request and Initiation of Request **4**, 3
Response Time Computation and Display **4**, 16
Response Time Error Control Function **4**, 16
Response Time Difference **4**, 19
Response Time of Service Request **4**, 19
Restrictions on Limited Processing Power **15**, 12
Result Is Stored (Behavior) **17**, 8
Results Based on Failure Rate Analysis **16**, 12
Results of Digital Computation (Analog to Digital Voltage Conversion) **3**, 10
Results That the Equations Must Achieve **17**, 20
Resume the Interaction with the User When the Migration to a Different Node Has Completed **16**, 24
Retrained to Deal with Minor Change **13**, 1
Return Address of the Instruction **17**, 2
Reusability Characterization **13**
Revising Probabilities of Remaining Failures Based on Fault And Failure Correction **8**, 15
Revising Reliabilities Based on Fault And Failure Correction **8**, 18
Rigorous Reliability Testing **14**, 1
Risk Analysis **13**, 1
Risk-Based Reliability Prediction **11**, 4
Risk of Carrying Viruses and Other Malware **15**, 1
Risk Control **11**, 9
Risk Control and Mitigation **11**, 9
Risk Criterion Metric **13**, 15
Risk Evaluation **11**, 1
Risk Function **13**, 15
Risk Goal **11**, 5
Risk of Mission Failure **11**, 3
Risk of Power Loss **15**, 2
Risk Trends Positive **13**, 13
Risks of Operating Mobile Devices **15**, 1
Risky Requirements **11**, 4
Roadmap for Improving Real-Time System Design **4**, 25
Robotic Web Services **14**, 2
Robustness **15**, 13
Routes Are Subject to Frequent Breakage **15**, 3
Rule of Considering Real-World Operational Details during Abstract Design **4**, 3
R-S Flip Flop **1**, 42
Run Realistic Tests That Stress the Hardware and Software to Fail **16**, 14

S/N Can Be Used to Rank the Reliability of Mobile Device Software **16**, 10
S/N Can Be Used to Prioritize Software Modules for Testing **16**, 13
S/N Influences Test Effectiveness **16**, 1
S/N Limit **16**, 16
S/N Ratio Is Computed and Test Software is Used to Compare the Required Ratio with the Ratio Actually Generated in the Network **7**, 14
Safe Mission **11**, 3
Safer, Lower Risk Alternative **13**, 15
Safety Against Cost **11**, 5
Safety of the Mission **11**, 2
Sample Data **13**, 1
Sample and Hold Circuit **3**, 4
Sample and Hold Circuit Must Sample Input at a Rate at Least Twice the Frequency of the Input in Order to Produce the Desired Output **3**, 14
Satisfy All the Functional Requirements and Timeliness Demands **4**, 1
Satisfy Response Time Requirements **4**, 25
Save Energy **15**, 12
Schedule Test Time **13**, 4
Scheduled Operating Times **12**, 2
Scheduling Algorithms **10**, 6
Scheduling Efficiency **10**, 7
Scheduling Policy **10**, 11
Scheduling and Timing Risks **11**, 2
Schneidewind Software Reliability Model **11**, 3; **13**, 4
Secondary Storage Component **15**, 15
Security **15**, 1
Security Breach on the Device **15**, 1
Select Personnel by Evaluating the Results for Accuracy, Reliability, and Quality of Design Documentation **17**, 19

Select Solution Routines That Have Good Convergence Properties for the Given Problem **17**, 2
Self-Shutdown **16**, 5
Self-Shutdown (Silent Failure) **15**, 4
Self-Shutdown and Unstable Behavior Are Considered to Be High-Severity Failures **16**, 6
Self-Shutdown and Unstable Behavior Are High-Severity Failures **15**, 6
Send Responses Back for the User **16**, 3
Sense All Context Information **15**, 13
Sensitivity Can Be Interpreted as Sensor Error **3**, 12
Sensor Attached to Access Point Records the Range Between the Mobile Device and the Access Point **7**, 14
Sensor Error Occurs When the Input Range Exceeds the Output Range **3**, 13
Sensor Is a Device That Receives and Responds to a Signal **3**, 12
Sensor's Sensitivity Indicates How Much the Sensor's Output Changes When the Measured Quantity Changes **3**, 12
Separation of Application Concerns and Implementation **4**, 2
Sequence Analysis **2**, 10
Sequence Diagram Is an Interaction Diagram That Shows How Software Processes Operate with One Another and in What Order **17**, 2
Sequence Diagrams Are Capable of Representing Sequential Interactions (e.g., Only a Single Elevator Floor Request at a Time) **17**, 3
Sequence Diagrams Provide Both the Sequence of Model Operations on Data and the Sequence of Steps That Implement the Model Operations **17**, 9
Sequence Diagrams Show the Sequence of Operations Between Objects and the Sequence of Program Steps That Are Required to Implement a Model **17**, 9
Sequence Failure Rate **2**, 11; **7**, 5
Sequence of Fault and Failure Injection **7**, 5
Sequence Input Rate **7**, 9
Sequence of Interactions **14**, 4
Sequence of Operations **12**, 13
Sequence of Operations on the Network **6**, 1
Sequence Probability **2**, 10
Sequence Probability and Sequence Response Time Predictions and Analysis **2**, 10
Sequence of the Reliability Simulation **7**, 5
Sequence Relationships **2**, 10
Sequence Response Time **2**, 10
Sequences Associated with Local Network Components **7**, 6
Sequential Circuits **1**, 39
Sequential System **12**, 2
Series Component **12**, 2
Series Configuration **12**, 3
Series-Parallel Configuration **12**, 5
Series System Reliability **12**, 3
Server Component Is the First in Line for Reliability Improvement **14**, 18
Server Consists of Multiple Single-Threaded Processes, Each of Which Handles One Request at a Time **6**, 2
Server-Side Problems **14**, 16
Server Uses the Error Control Function to Increase the Clock Rate **4**, 17
Service Performance **14**
Service the Phone **15**, 5; **16**, 5
Service Requirements Impose Ordering on the Invocation of Operations **14**, 3
Session and Presentation Layer Services **5**, 7
Several Metrics of Real-Time System Performance Are Modeled and Evaluated **4**, 1
Severe Reliability Problem Will Prevail Short of 18 Months **16**, 17
Severity Levels Corresponding to the Difficulty of the Recovery Action(s) **15**, 6; **16**, 4
Severity Reflects Both Failure Type and Recovery Type **16**, 9
Shape Parameter **12**, 15
Shape of the Reliability Function **12**, 15
Shared Data Areas Are Protected, Reducing the Possibility of Unexpected Modifications **17**, 4
Shared, Noisy, Highly Variable, and Limited Wireless Communication Links **7**, 12

Shift Register Design **1**, 71
Short-Term Scheduler **10**, 7
Shuttle Continuous Software Testing Regimen **11**, 4
Shuttle Flight Software **13**, 9
Shuttle Flight Software Exhibits Reliability Growth (i.e., Increases with Operating Time) **17**, 17
Signal (# of Correct Modules) **16**
Signal Conversion Circuits as a Single Integrated System **3**, 15
Signal Could Be Represented by Number of Successful Web Search Results **16**, 10
Signal Dead Zones **16**, 2
Signal Distortion **3**, 11
Signal Driven Software Model **16**, 1
Signal Fading **15**, 3
Signal Interference in the Available Spectrum, Particularly in Wireless Systems Is a Network Standards Issue **7**, 13
Signal an Intrusion When There Is a Match **15**, 3
Signal Must Be Sampled at Least Twice Its Frequency **3**, 11
Signal and Noise Are Measured **7**, 14
Signal to Noise Ratio **7**, 1; **15**, 1
Signal to Noise Ratio Indexes Reliability **16**, 1
Signal to Noise Ratio Is Tested by Propagating the Signal and Noise to an Oscilloscope Where
Signal to Noise Ratio (S/N) Representation of Reliability **16**, 1
Signal to Noise Ratio (S/N) Stability (i.e., S/N >> 1) **15**, 3
Signal and Noise Relationships Can Be Used to Quantify Test Effectiveness **16**, 17
Signal Representation Distortion **3**, 11
Signal Strength Is Critical **16**, 1
Signal That Network Quality Should Be Improved **7**, 9
Signature Recognition and Anomaly Detection **15**, 2
Signature Recognition Techniques Establish a Profile **15**, 2
Signature Recognition Techniques Match Entities **15**, 2
Signatures of Known Entity Intrusions **15**, 2
Significant Contributors to Unreliability **14**, 10
Significant Loss or Hazard **15**, 3; **16**, 3
Significant Number of Failures **14**, 4
Significant Probability of Multiple Failed Components **12**, 9
Simple Client-Server Requests for Web Pages **14**, 2
Simple Semantic Foundation (e.g., Mathematical Equations That Communicate the Meaning of the Application) **17**, 5
Simulate the Injection of Faults and Failures into a Replica of a Computer Network **7**, 1
Simulating Network Reliability **7**, 4
Simulation Can Be Used to Generate Random Changes in Voltage **3**, 13
Simulation Error Analysis **3**, 13
Simulation Queuing Models **6**, 42
Single Cells Capable of Crude Computation **13**, 1
Single Communication Functions **14**, 18
Single Component Failure **12**, 19
Single Partition **12**, 12
Size of Main Memory **10**, 13
Small Gains in Noise Reduction Would Be Achieved Through Testing If the Number of Correct Modules Is Already Large **16**, 13
Smallest Sub-Functions **12**, 12
Smallest Web Page Request Packet **7**, 2
Smart Electric Meter System **3**, 15
Smart Meter Microcomputer **3**, 5
Smart Meters in Smart Electric Grid Systems **3**, 9
Smart Phones Do Not Have a Means to Detect and Collect Failures **15**, 5
Smart Phones Have More Complex Architecture Than Voice Centric Mobile Phones **15**, 4
Software Compatibility **7**, 9
Software Compatibility Standards Issue **7**
Software Components in Series **12**, 12
Software Configuration **11**, 3

Software Defined Device Provides Needed Functionality (e.g., Short Range to Long Range Communication) **16**, 4
Software Demonstrating the Lowest Probability of Failure **14**, 11
Software Developers Can Incorporate Compatibility into Standards **7**, 10
Software Development Cycle **11**, 4
Software Development Models for Mobile Devices to Communicate with the Electric Grid in a Collaborative Processing Mode **16**, 24
Software Development Process **17**, 19
Software Dimension **8**, 2
Software, Due to Its Complexity, Has Caused More Problems Than Hardware **7**, 9
Software Evolves and These Changes Can Negatively Affect Performance **15**, 15
Software Failures **14**, 5
Software Compatibility Standards Issue **7**, 9
Software for Controlling a Nuclear Reactor **17**, 1
Software Functions **4**, 11
Software Has the Best Prediction Accuracy **14**, 10
Software Inoperable **12**
Software Is First Modeled Abstractly without Considering Its Execution Platform **4**, 2
Software Is in Need of Significant Software Development Process Improvement to Reduce Failures **16**, 8
Software Is Released **13**, 4
Software Level **4**, 5
Software Management Requiring Traceability among Software Products and the Process Steps That Produce Them **17**, 4
Software Mobile Network Products **15**, 3
Software Models That Deal with Mobile Devices **16**, 1
Software Portability **15**, 14
Software Product Logic **17**, 19
Software Redundancy **12**, 18
Software Reliability (Object) Must Achieve Its Specification (Task) during Test and Operating Time) **17**, 5
Software Reliability Assessment Problem **13**, 18
Software Reliability Improvement **13**, 3
Software Reliability Prediction Metrics **13**, 18
Software Reliability Profile Implementation **11**, 3
Software Reliability Results **13**, 11
Software Reliability Results Cannot Be Considered Representative **16**, 10
Software Reuse and Support of Various Tools **17**, 3
Software System Could Be Operated Safely **13**, 15
Software System Designed with Procedures to an O-O Perspective **17**, 4
Software Written in Event-Driven Style Typically Waits for an Event to Occur **4**, 6
Software Would Never Be Able to Achieve a Specified Reliability **17**, 18
Source of Failure Data **11**, 3
Specified Critical Value **11**
Specified Network Hardware and Software **7**, 13
Specified Number of Failures **13**, 15
Specified Number of Remaining Failures **13**, 13
Specified Reliability **11**, 1; **12**, 4; **15**, 1
Specified Reliability Requirement **16**, 11
Specified Reliability Values **15**, 1
Specifying a Requirement, While Neglecting to Provide a Rationale **7**, 13
Spectrum Considerations **7**, 13
SR Latch **1**, 40
Stabilization Time **15**, 3; **16**, 1
Stabilization Time Is the Operating Time during Which Specified Reliability Is Achieved **15**, 3
Stack **1**, 4
Standard Communication Protocols **14**, 2
Standard for Hardwired Networks **7**, 10
Standard for the Recovery of Failed Web Services **14**, 2
Standard Internet Protocols **14**, 1
Standardization Simplifies Interoperability **14**, 2
State Diagram: Diagram That Shows States and Transitions Between States **17**, 2

State Diagrams Are Effective for Representing This Environment **4**, 8
State Machines to Model the Order of Web Service Operations **14**, 3
State of an Object Represents the Results of Its Behavior **17**, 10
State Transition Connects Two States **17**, 10
State Transition Probabilities **14**, 4
State Transition That Causes a Web Server to become Active **14**, 9
State Transitions **14**, 9
State Transitions That Must Be Tested **4**, 22
Statement Execution Result **13**, 3
States and State Transitions **4**, 6
States and State Transitions Form the Core Processes **4**, 8
Static Part of the Mobile Device Is Its Hardware **16**, 3
Stationary Service Always Executes on the Same Node **16**, 24
Statistical Metrics to Compute and Predict Reliability for Illustrative Web Servers **14**, 6
Statistical Modeling Theory for the Evaluation of Web-Based System Reliability **14**, 6
Statistical Routine **7**, 2
Statistical Testing and Reliability Analysis **14**, 11
Steady State Reliability **16**, 1
Steps Necessary to Define the Components of the Model **17**, 19
Steps in Real-Time System Design **4**, 2
Storage **2**, 1
Storage (Digital Data Stored in Database) **3**, 10
Storage Architecture **15**, 15
Storage Capabilities **15**, 15
Storage Capacity Prediction **7**, 9
Storage Requirement Test Is Conducted with Test Software by Comparing the Database Capacity with the Web Page Storage Requirement **7**, 14
Storage Requirements Must Be Predicted **7**, 9
Storage System with Sufficient Capacity to Support the Input, Storage, and Output of Real-Time Transactions **4**, 12
Strategy Does a Good Job of Exercising Many, but Not All, of the Paths **17**, 17
Strength of Functions Lies in the Fact That They Are Programs within a Program **17**, 5
Stress to Identify Both Hardware and Software Failures **16**, 14
Strong Partitioning **12**, 12
Strong Partitioning of Applications **12**, 12
Structural Hazards **1**, 9
Structure **9**, 5
Structure of Reliability Equations **12**, 4
Structure of a Software Application **12**, 13
Structured Analysis and Design **17**, 1
Study the Effects of Increasing Bandwidth and Operating Time on Communication Channel Reliability **14**, 22
Subcontracting Risks **11**, 2
Subject the System to Increasing Values of Mission Duration **13**, 8
Success of HPC Computing Will Depend on the Ability to Provide High Reliability **14**, 5
Success of the Input Received Function **12**, 2
Successful Execution **13**, 3
Successful Intrusion Increases the Noise in a Mobile Network **15**, 2
Successful Operation Between a Pair of Nodes **8**, 3
Sudden Jumps in Hazard Function **14**, 10
Sudden Need for the Mobile Device to Move with the user (Context Aware) **16**, 24
Sum Failure Count State **17**, 10
Sum of Correct Modules and Failed Modules **16**, 9
Summary of Queuing Model Computations for Present and Proposed Internets **6**, 30
Summary of Simulation Model Computations **6**, 55
Summary of Software Development Approaches **17**, 18
Summation of Link Delay, Processing Time, and Wait Time **5**, 12
Summing the Node and Link Times **7**, 5
Super Computer **12**, 16
Superiority of Neural Network Criterion **13**, 13

Superiority of Neural Network Reliability Criterion Limit **13**, 8
Support Multimedia Services **7**, 13
Support Requirements **9**, 11
Synchronized Program Development Activities **17**, 13
Synchronous and Asynchronous Communication among Components **12**, 13
Syntax Oriented (e.g., Emphasis on UML Diagramming Techniques) **17**, 3
System Bus with Sufficient Bandwidth to Accommodate Expected Data Transfer Requirements **4**, 14
System Changes Will Only Affect the Interface **17**, 2
System Clock of the Mobile Device **15**, 13
System Configuration Descriptions **14**, 3
System Decomposition Into Components **9**, 2
System Error **4**, 19
System Error Feedback Correction **4**, 19
System Failures Include User and Computer Operator Errors **14**, 10
System Functionality Risks **11**, 2
System Identification **13**, 1
System Level **4**, 5
System Must Be Capable of Detecting Logical as Well as Timing Errors in the Design **4**, 23
System Must Respond to Asynchronous Events **4**, 8
System Queues Are Used to Store Backlog of User Requests **4**, 4
System Reliability **12**, 2
System Reliability Model **12**, 1
System Resources, such as Microprocessor Cycles, Communication Bandwidth, and Storage Memory Are Restricted **4**, 1
System, Software, and Hardware Failure Rates **14**, 11
System Storage **4**, 6
System Validation **12**, 13
System View That Is Desired **17**
System Workload Is Taken into Account **16**, 10

T Flip Flop **1**, 49
Target User **15**, 13
TCP Connection to the Web Server **14**, 4
TCP/IP Is a Protocol That Interfaces with Local Network Protocols such as Ethernet **7**, 10
TCP/IP Is a Protocol That Operates at the Transport Layer of the Seven Layer Network **7**, 10
Teaching Neural Networks **13**, 2
Telecom Service Providers **7**, 12
Template for Using Various Objects (Probability Functions) and Their Attributes (Variables and Parameters) in the Same Probability Distribution Class **17**, 11
Test Bed for Testing Networks **7**, 14
Test Case Selection Is Designed to Provide Adequate Coverage of System Components by Deriving Test Cases from Software Designs **4**, 22
Test Cases Are Based on Recovery Action (e.g., Remove Battery) **16**, 17
Test Cases Are Based on Type of Failure (e.g., Freeze) **16**, 17
Test Data Design **2**, 8
Test Duration Serves as a Test Stopping Rule **16**, 14
Test Effectiveness Can Be Used to Prioritize Modules for Testing **16**, 13
Test Effectiveness of Failure Type **16**, 17
Test Effectiveness Increases with Lower Signal to Noise Ratio **16**, 16
Test Effectiveness Is the Duration of Test Necessary to Achieve That Effectiveness **16**, 14
Test the Interaction in Terms of Performance Results **4**, 22
Test Interval **13**, 4
Test Measurements Are Instrumented **7**, 14
Test Paths Associated with the Program Input Variables **17**, 17
Test Paths Used to Debug the C++ Program **17**, 13
Test Plan **2**, 7
Test Plan Support Functions **2**, 7
Test Plans Must Recognize Constraints **16**, 3
Test Results Reflect Realistic Operating Conditions **16**, 3

Test Software Compares the Actual Range with the Received Range **7**, 14
Test Software Computes Required Time and Compares It with Clock Time **7**, 14
Test Software Records a Compatibility Result If the Signal Is Received **7**, 14
Test Strategies **2**, 6
Test Time **11**, 2; **13**, 2; **16**, 14
Test Time Increases with Decreasing Signal to Noise Ratio (i.e., Many Failed Modules Compared with the Number of Correct Modules) **16**, 17
Test Time Is Equal to Number of Failed Modules That Are Corrected Divided by the Failure Rate **16**, 14
Test Time Is Modeled as a Two Phase Sequence **16**, 17
Testing **12**, 1; **14**, 1
Testing for All Possible Exceptions in all Possible Places Where an Exception Could Be Raised Is Impractical **17**, 17
Testing Approach Must Be Highly Non-Intrusive **16**, 3
Testing Challenge Is to Include the Number of Active Users Connected to Mobile Networks **16**, 3
Testing Challenges **16**, 2
Testing of Mobile Devices Is Difficult Because the Environment Is Complex **16**, 2
Testing Must Be Performed in the Constrained Memory of the Mobile Device **16**, 3
Testing Problems **16**, 1
Testing and Reliability Have a Synergistic Relationship **16**, 12
Testing under Simulated Operational Conditions **11**, 4
Testing for a Time to Assure High Reliability **12**, 12
Tests Should Interact with End Users, Wireless Signals, and the Wireless Network **16**, 3
Text Message Was Being Received **15**, 5
Text Messaging **15**, 15
Thorough Testing of Real-Time Systems **4**, 22
Thread-Based Architecture **6**, 2
Throughput **4**, 21; **10**, 6; **15**, 14
Throughput per User **16**, 3
Time **7**, 1; **15**, 1
Time-Based Reliability Model **16**
Time of Completion of Service Request **4**, 19
Time-Driven Software Design Style Corresponds to Using Cyclic Activities, Triggered by Time **4**, 7
Time to Failure **11**, 2
Time to Failure Risk **11**, 6
Time to Failure across Various Time Intervals **13**, 15
Time of Failure Occurrence **17**, 6
Time to Let the Device Deliver the Expected Service **15**, 5
Time to Next Failure **11**, 6; **13**, 15
Time Required to Request a Web page from Web Server **7**, 14
Time of Service Request **4**, **6**, 19
Time Slice Length Strategy **10**, 8
Time of Switch Action **10**, 13
Time in System **6**, 45
Time of Testing Software **13**, 2
Times When the Failures Occurred **17**, 12
Timing Constraints Are Addressed in Analyzing Real-Time System Performance **4**, 2
Today, Many Computer Systems Are Being Used to Measure and Control Real-World Processes **4**, 2
To Minimize Low Pass Filter Error, Maximize the Signal to Noise Ratio S/N **3**, 14
To Minimize Operational Amplifier Error, Ensure That the Output/Input Ratio = Amplification Factor **3**, 14
To Minimize Voltage Sensor Error, the Sensor Should Produce an Output Change to Input Change Ratio = 1 **3**, 14
To Prevent Sample and Hold Circuit Error, Ensure That the Circuit Can Sample at a Frequency f_{sh} > Desired Frequency fi **3**, 14
Too Little Memory Space Allocated to Buffers, Resulting in Buffer Overflow **16**, 7
Topology **2**, 2
Total Expected Operational Time **4**, 21
Total Number of Failures **16**, 7

Total Number of Failures Reported at the Scheduled Test Time Interval **13**, 12
Total Number of Mobile Device Modules **16**, 9
Total Number of Modules in a Mobile Device **16**, 6
Total Paging Time **10**, 13
Total Quantity of Data Processed at all Nodes and Associated Links in a Network **7**, 8
Total Scheduled Test Time **13**, 7
Total System Reliability Analysis **14**, 1
Total System Reliability Models **14**, 18
Traceability of Product and Process **17**, 4
Track the Hazard Function Produced by Web Servers **14**, 10
Tracked by Using User and Computer Operator Logs **14**, 10
Traditional System **14**, 1
Traditional Testing Methods **4**, 21
Trained to Operate in a Specific Environment **13**, 1
Transfer the Process to this Device **16**, 24
Transfer Program Control **15**
Transformed to a Software Design Model on the Target Platform **4**, 2
Transition Information **14**, 3
Transmission Control Protocol (TCP) Connection to the Remote Web Server **14**, 10
Traverse All Links and Nodes to the Web SERVER **7**, 9
Tree Structure **14**, 3
Trend to Connect More Devices Will also Accelerate, Facilitated by the Increasing Installation of Internet Protocol version 6 (IPv6) **6**, 2
Trigger (Event) for the Poisson Failure Model (Object) to Store the Failure Count (Action) **17**, 10
Triggered by Sensing an Intrusion **15**, 14
Triggering of Flip-flops **1**, 3
Turnaround **10**, 6
Type of Failure and Category of Failure Recovery Action **15**, 1
Type of Failure Recovery Action **16**, 1
Types of Failures **14**, 3; **16**
Types of Failures and Responses to the Failures **15**, 1
Types of Synchronous Sequential Circuits **1**, 57

Unacceptable Mission Duration at the Specified Reliability **16**, 16
Unambiguous (Meaning Is Clearly Expressed) **17**, 4
Ultimately, the Particular Characteristics of the Application Must Be Considered **4**, 25
Unexpected Interactions with other Program Modules Are Unlikely **17**, 4
Unified Hardware-Software Reliability Model **12**, 13
Unified Modeling Language: Standardized Notation and Set of Diagrams **17**, 2
Unified Modeling Language (UML) Diagrams Can Be Used to Model the Elements **17**, 8
Unified System That Includes A/ D Conversion **3**, 9
Units **9**, 11
Unreliability **15**, 7
Unreliable and Unmaintainable Code **4**, 2
Unstable Behavior (Erratic Failure) **15**, 4; **16**, 6
Unweighted Probability **14**, 9
Upload Direction (i.e., Request for Web page) and Download Direction (i.e., Delivery of Web Page) **6**, 1
Upon Failure Detection, the Logger Gathers Useful Information **15**, 5
Usage Scenarios Are Difficult to Automate **16**, 3
Use of Compatible Interfaces **7**, 14
Use the Generic Design to Guide the Development of the Application-Specific Design **4**, 1
Use of Low Power Enables Longer Battery Life Applications such as a Personal Data Assistant **7**, 13
Use Standardized Interfaces **7**, 12
Useful for Debugging **17**, 4
User Computers and Mobile Devices Would Access a Web Server by Providing a Universal Resource Locator (URL) (Web Site Address) to the Internet Service Provider **6**, 2

User-Defined Functions Are Functions That Programmers Create for Specialized Tasks **17**, 5
User of the Device **15**, 13
User Having to Restart the Application **15**, 4
User-Initiated Actions to Recover from a Device Failure **16**, 5
User-Injected Errors **15**, 12
User Interface **16**, 4
User Is Switched to Other Access Points **15**, 12
User of a Mobile Device Seldom Changes **15**, 12
User Mobility **7**, 12
User Perceived Application Response Times Are Often Poor **15**, 14
User-Perceived Reliability and Availability Data **12**, 13
User Perceived Response Time **15**, 14
User System Requests Must Be Queued Because the System Controller Is Unable to Respond to All Requests Immediately **4**4
User Turns Off the Device and Then Turns It On to Restore the Correct Operation **15**, 5
User Will Interact with the System, Supplying Information to Help It Choose the Right Algorithms and Transformations **17**, 3
Users Experience a Failure (Freeze or Self Shutdown) **15**,4
Users' Requests Can Be Supported by a Proxy **14**, 5
Users Should Not Have to Wait for a Response in Order to Recover from a Failure **16**, 16
Using the Successful Time Obtained from the Previous Test, and Compares It with the Specified Reliability **7**, 14
Utility of the Prediction Is to Delineate The Maximum Storage Requirement **7**, 9
Utilization of Resources **15**, 14

Validating Real-Time Systems **12**, 13
Validation of Computer System Reliability **12**, 13
Validity Checks on Memory Access **15**, 7
Validity of Equation **13**, 4
Validity of the Neural Network Criterion Limit **13**, 5
Validity of Reliability Predictions **14**, 8
Valuable for Portraying the Process That Develops the Product **17**, 19
Value of Total Web Services **14**, 18
Value-Added Total Web Services **14**, 18
Variable: A Model Predictor Specified in a Function (e.g., Predictor of Software Reliability) **17**, 5
Variable Assumes Values Based on a Function **17**, 6
Variable Number of Active Users **16**, 3
Variance Between Actual and Predicted Values **12**, 13
Vendors Should Provide Better Protection Against Memory Violations **15**, 7
Verification Error Can Be Minimized **4**, 22
Verification Step **17**, 17
Verifying That the Specifications Can Be Achieved **7**, 15
Very High Reliability Software And Hardware Components **12**, 19
Virtual Operating Systems **10**, 18
Visual Language Alternative **9**, 19
Voice and Data in Wired Networks Increasingly Converge to Use the Internet **7**, 11
Voice over Internet Protocol (VoIP) **15**, 12
Voltage Regulation **3**, 10
Voltage Regulator of the Electric Distribution System **3**, 9
Voltage Sensor **3**, 12
Voting Mechanism **14**, 2

Wait an Amount of Time **15**, 5
Wait for a Response **16**, 6
Wait Time **6**, 9
Waiting for the Phone to Respond **16**, 7
Weak Spots in Component and System Reliability **14**, 1
Web Client **14**, 2
Web Client and Server Interactions **14**, 9
Web Client Directly Accessing the Web Server to Obtain a Page **14**, 4
Web Client Meets the Reliability Requirement **14**, 17
Web Client Reliability Analysis **14**, 16

Web Client to Web Server Access Time **14**, 1
Web Database **14**, 4
Web Page Design **14**, 3
Web Page Lengths Being Processed and Waiting for Processing by the Web Servers **6**, 55
Web Page Syntax **14**, 3
Web Server **6**, 2; **14**, 1
Web Server Consists of a Single Multi-Threaded Process; Each Thread Handles One Request at a Time **6**, 2
Web Server Failure Data **14**, 5
Web Server Interactions **14**, 3
Web Server Processing **6**, 17
Web Server Processing: Wired and Wireless **6**, 53
Web Server Proxy **14**, 4
Web Server Reliability Analysis **14**, 6
Web Server Reliability Analysis Based on Operating Time **14**, 11
Web Server Reliability Requirement **14**, 6
Web Service-Fault Tolerance Mechanism **14**, 2
Web Services Are Implemented in Different Redundant Versions **14**, 2
Web Services Providers **14**, 1
Web Services State Transitions **14**, 3
Web System Communication **14**, 2
Web System Error Rate **14**,17
Web System Functions **14**, 18
Web System Path Data **14**, 2
Web System Reliability Approach **14**, 1
Web System Reliability Predictions **14**, 23
Web System Service and Reliability **14**, 5
Web System State Transition Diagram **14**, 3
Web Transaction **14**, 4
Weibull Distribution Proved Appropriate for Predicting System, Software, and Hardware Reliability **14**, 6
Weibull Failure Distribution **12**, 15
Weibull Model Results **12**, 15
Weibull Reliability Model **12**, 20
Weigh Remaining Failures **13**, 17
Weighed by the Failure Severity Code **16**, 9
Weighted Connections Following a Specified Structure **13**, 1
Weighted Probability of State Transition **14**, 9
Weighted Sum of the Probabilities **16**, 9
What Is Design? **9**, 1
When a Class Is Declared, It Is Identified by Name, Attributes, and Methods **17**, 11
When Failures Occur, the Mobile Network Has to Try to Find Another Mobile Device **16**, 24
When Testing and Performance Evaluation Are Performed, the Particular Characteristics of the Application Must Be Considered **4**, 2
Wired LANs **15**, 3
Wired and Wireless Internets Are Included in the Analysis **6**, 1
Wireless Channel Conditions Are Inherently More Vulnerable **15**, 3
Wireless Communication **15**, 3
Wireless LANS Do Not Have the Luxury of Delay or Shut Down **15**, 3
Wireless LANS Require Much Higher Reliability **15**, 3
Wireless Link Is Much Less Reliable Than Wired Connections **15**, 4
Wireless Local Area and Home Networks **7**, 10
Wireless Local Area Networks (WLAN) Positioning Systems **15**, 12
Wireless Media Have Limited and Variable Ranges **15**,3
Wireless Networks IEEE802 Family of Standards **7**, 12
Wireless Standards **7**, 10
Wireless Standard Organizations Are Revising Existing Standards **7**, 12
Wireless to Access Internet Resources **14**, 2
Wireless Video Phone and Multimedia Message Systems **7**, 12
Workarounds **11**, 10
Workload Characteristics **14**, 11
Writing Computer Code: Iteration Control, Variable Types, Array Bounds, and Sequence of Computer Code Fragments **17**, 13
Wrong Branch **12**, 14