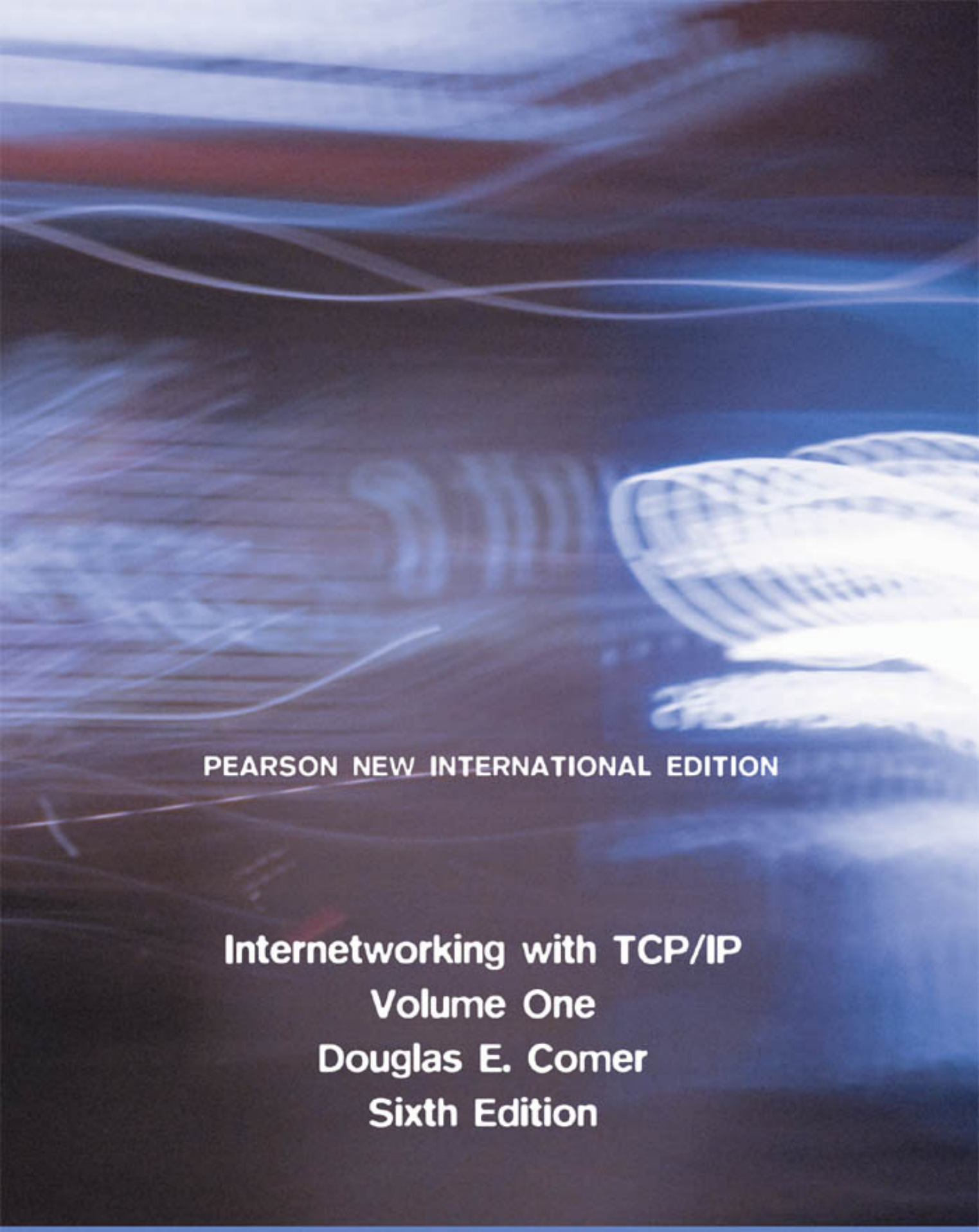

PEARSON NEW INTERNATIONAL EDITION

Internetworking with TCP/IP
Volume One
Douglas E. Comer
Sixth Edition

ALWAYS LEARNING™

PEARSON®

Pearson New International Edition

Internetworking with TCP/IP
Volume One
Douglas E. Comer
Sixth Edition

PEARSON®

ICMP includes *destination unreachable* messages that report when a datagram cannot be forwarded to its destination, *packet too big* messages that specify a datagram cannot fit in the MTU of a network, *redirect* messages that request a host to change the first-hop in its forwarding table, *time exceeded* messages that report when a hop limit expires or reassembly times out, and *parameter problem* messages for other header problems. In addition, ICMP *echo request/reply* messages can be used to test whether a destination is reachable. A set of older ICMPv4 messages that were intended to supply information to a host that booted are no longer used.

An ICMP message travels in the data area of an IP datagram and has three fixed-length fields at the beginning of the message: an ICMP message *type* field, a *code* field, and an ICMP *checksum* field. The message type determines the format of the rest of the message as well as its meaning.

EXERCISES

9.1 Devise an experiment to record how many of each ICMP message type arrive at your host during a day.

9.2 Examine the *ping* application on your computer. Try using *ping* with an IPv4 network broadcast address or an IPv6 *All Nodes* address. How many computers answer? Read the protocol documents to determine whether answering a broadcast request is required, recommended, not recommended, or prohibited.

9.3 Explain how a *traceroute* application can use ICMP.

9.4 Should a router give ICMP messages priority over normal traffic? Why or why not?

9.5 Consider an Ethernet that has one conventional host, *H*, and 12 routers connected to it. Find a single (slightly illegal) frame carrying an IP packet that when sent by host *H* causes *H* to receive exactly 24 packets.

9.6 There is no ICMP message that allows a machine to inform the source that transmission errors are causing datagrams to arrive with an incorrect checksum. Explain why.

9.7 In the previous question, under what circumstances might such a message be useful?

9.8 Should ICMP error messages contain a timestamp that specifies when they are sent? Why or why not?

9.9 If routers at your site participate in ICMP router discovery, find out how many addresses each router advertises on each interface.

9.10 Try to reach a server on a nonexistent host on your local network. Also try to communicate with a nonexistent host on a remote network. In which case(s) do you receive an ICMP error message, and which message(s) do you receive? Why?

Chapter Contents

10

User Datagram Protocol (UDP)

10.1 Introduction

Previous chapters describe an abstract internet capable of transferring IP datagrams among host computers, where each datagram is forwarded through the internet based on the destination's IP address. At the internet layer, a destination address identifies a host computer; no further distinction is made regarding which user or which application on the computer will receive the datagram. This chapter extends the TCP/IP protocol suite by adding a mechanism that distinguishes among destinations within a given host, allowing multiple application programs executing on a given computer to send and receive datagrams independently.

10.2 Using A Protocol Port As An Ultimate Destination

The operating systems in most computers permit multiple applications to execute simultaneously. Using operating system jargon, we refer to each executing application as a *process*. It may seem natural to say that an application is the ultimate destination for a message. However, specifying a particular process on a particular machine as the ultimate destination for a datagram is somewhat misleading. First, because a process is created whenever an application is launched and destroyed when the application exits, a sender seldom has enough knowledge about which process on a remote machine is running a given application. Second, we would like a scheme that allows TCP/IP to be used on an arbitrary operating system, and the mechanisms used to identify a process

vary among operating systems. Third, rebooting a computer can change the process associated with each application, but senders should not be required to know about such changes. Fourth, we seek a mechanism that can identify a service the computer offers without knowing how the service is implemented (e.g., to allow a sender to contact a web server without knowing which process on the destination machine implements the server function).

Instead of thinking of a running application as the ultimate destination, we will imagine that each machine contains a set of abstract destination points called *protocol ports*. Each protocol port is identified by a positive integer. The local operating system provides an interface mechanism that processes use to specify a port or access it.

Most operating systems provide synchronous access to ports. From an application's point of view, synchronous access means the computation stops when the application accesses the port. For example, if an application attempts to extract data from a port before any data arrives, the operating system temporarily stops (blocks) the application until data arrives. Once the data arrives, the operating system passes the data to the application and restarts execution. In general, ports are *buffered* — if data arrives before an application is ready to accept the data, the protocol software will hold the data so it will not be lost. To achieve buffering, the protocol software located inside the operating system places packets that arrive for a particular protocol port in a (finite) queue until the application extracts them.

To communicate with a remote port, a sender needs to know both the IP address of the destination machine and a protocol port number within that machine. Each message carries two protocol port numbers: a *destination port* number specifies a port on the destination computer to which the message has been sent, and a *source port* number specifies a port on the sending machine from which the message has been sent. Because a message contains the port number the sending application has used, the application on the destination machine has enough information to generate a reply and forward the reply back to the sender.

10.3 The User Datagram Protocol

In the TCP/IP protocol suite, the *User Datagram Protocol* (*UDP*) provides the primary mechanism that application programs use to send datagrams to other application programs. UDP messages contain protocol port numbers that are used to distinguish among multiple applications executing on a single computer. That is, in addition to the data sent, each UDP message contains both a destination port number and a source port number, making it possible for the UDP software at the destination to deliver an incoming message to the correct recipient and for the recipient to send a reply.

UDP uses the underlying Internet Protocol to transport a message from one machine to another. Surprisingly, UDP provides applications with the same best-effort, connectionless datagram delivery semantics as IP. That is, UDP does not guarantee that messages arrive, does not guarantee messages arrive in the same order they are sent, and does not provide any mechanisms to control the rate at which information flows

between a pair of communicating hosts. Thus, UDP messages can be lost, duplicated, or arrive out of order. Furthermore, packets can arrive faster than the recipient can process them. We can summarize:

> *The User Datagram Protocol (UDP) provides an unreliable, best-effort, connectionless delivery service using IP to transport messages between machines. UDP uses IP to carry messages, but adds the ability to distinguish among multiple destinations within a given host computer.*

An important consequence arises from UDP semantics: an application that uses UDP must take full responsibility for handling the problems of reliability, including message loss, duplication, delay, out-of-order delivery, and loss of connectivity. Unfortunately, application programmers sometimes choose UDP without understanding the liability. Moreover, because network software is usually tested across Local Area Networks that have high reliability, high capacity, low delay, and no packet loss, testing may not expose potential failures. Thus, applications that rely on UDP that work well in a local environment can fail in dramatic ways when used across the global Internet.

10.4 UDP Message Format

We use the term *user datagram* to describe a UDP message; the emphasis on *user* is meant to distinguish UDP datagrams from IP datagrams. Conceptually, a user datagram consists of two parts: a header that contains meta-information, such as source and destination protocol port numbers, and a payload area that contains the data being sent. Figure 10.1 illustrates the organization.

Figure 10.1 The conceptual organization of a UDP message.

The header on a user datagram is extremely small: it consists of four fields that specify the protocol port from which the message was sent, the protocol port to which the message is destined, the message length, and a UDP checksum. Each field is sixteen bits long, which means the entire header occupies a total of only eight octets. Figure 10.2 illustrates the header format.

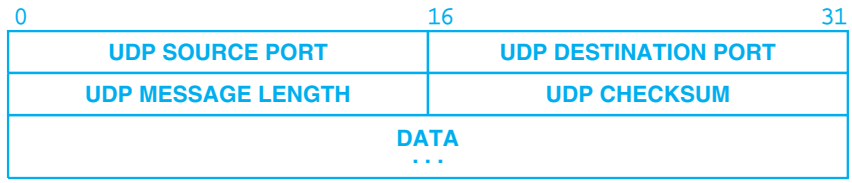

Figure 10.2 The format of fields in a UDP datagram.

The *UDP SOURCE PORT* field contains a 16-bit protocol port number used by the sending application, and the *UDP DESTINATION PORT* field contains the 16-bit UDP protocol port number of the receiving application. In essence, protocol software uses the port numbers to demultiplex datagrams among the applications waiting to receive them. Interestingly, the *UDP SOURCE PORT* is optional. We think of it as identifying the port to which a reply should be sent. In a one-way transfer where the receiver does not send a reply, the source port is not needed and can be set to zero.

The *UDP MESSAGE LENGTH* field contains a count of octets in the UDP datagram, including the UDP header and the user data. Thus, the minimum value is eight, the length of the header alone. The *UDP MESSAGE LENGTH* field consists of sixteen bits, which means the maximum value that can be represented is 65,535. As a practical matter, however, we will see that a UDP message must fit into the payload area of an IP datagram. Therefore, the maximum size permitted depends on the size of the IP header(s), which are considerably larger in an IPv6 datagram than in an IPv4 datagram.

10.5 Interpretation Of the UDP Checksum

IPv4 and IPv6 differ in their interpretation of the *UDP CHECKSUM* field. For IPv6, the UDP checksum is required. For IPv4, the UDP checksum is optional and need not be used at all; a value of zero in the *CHECKSUM* field means that no checksum has been computed (i.e., a receiver should not verify the checksum). The IPv4 designers chose to make the checksum optional to allow implementations to operate with little computational overhead when using UDP across a highly reliable local area network. Recall, however, that IP does not compute a checksum on the data portion of an IP datagram. Thus, the UDP checksum provides the only way to guarantee that data has arrived intact and should be used†.

Beginners often wonder what happens to UDP messages for which the computed checksum is zero. A computed value of zero is possible because UDP uses the same checksum algorithm as IP: it divides the data into 16-bit quantities and computes the one's complement of their one's complement sum. Surprisingly, zero is not a problem because one's complement arithmetic has two representations for zero: all bits set to zero or all bits set to one. When the computed checksum is zero, UDP uses the representation with all bits set to one.

†The author once experienced a problem in which a file copied across an Ethernet was corrupted because the Ethernet NIC had failed and the application (NFS) used UDP without checksums.