User Datagram Protocol (UDP)

OBJECTIVES:

- To introduce UDP and show its relationship to other protocols in the TCP/IP protocol suite.
- To explain the format of a UDP packet and discuss the use of each field in the header.
- To discuss the services provided by the UDP such as process-to-process delivery, multiplexing/demultiplexing, and queuing.
- To show how to calculate the optional checksum (and the sender needs to add a pseudoheader to the packet when calculating the checksum.)
- To discuss how some application programs can benefit from the simplicity of UDP.
- To briefly discuss the structure of the UDP package.

Chapter Outline

14.1 Introduction

14.2 User Datagram

14.3 UDP Services

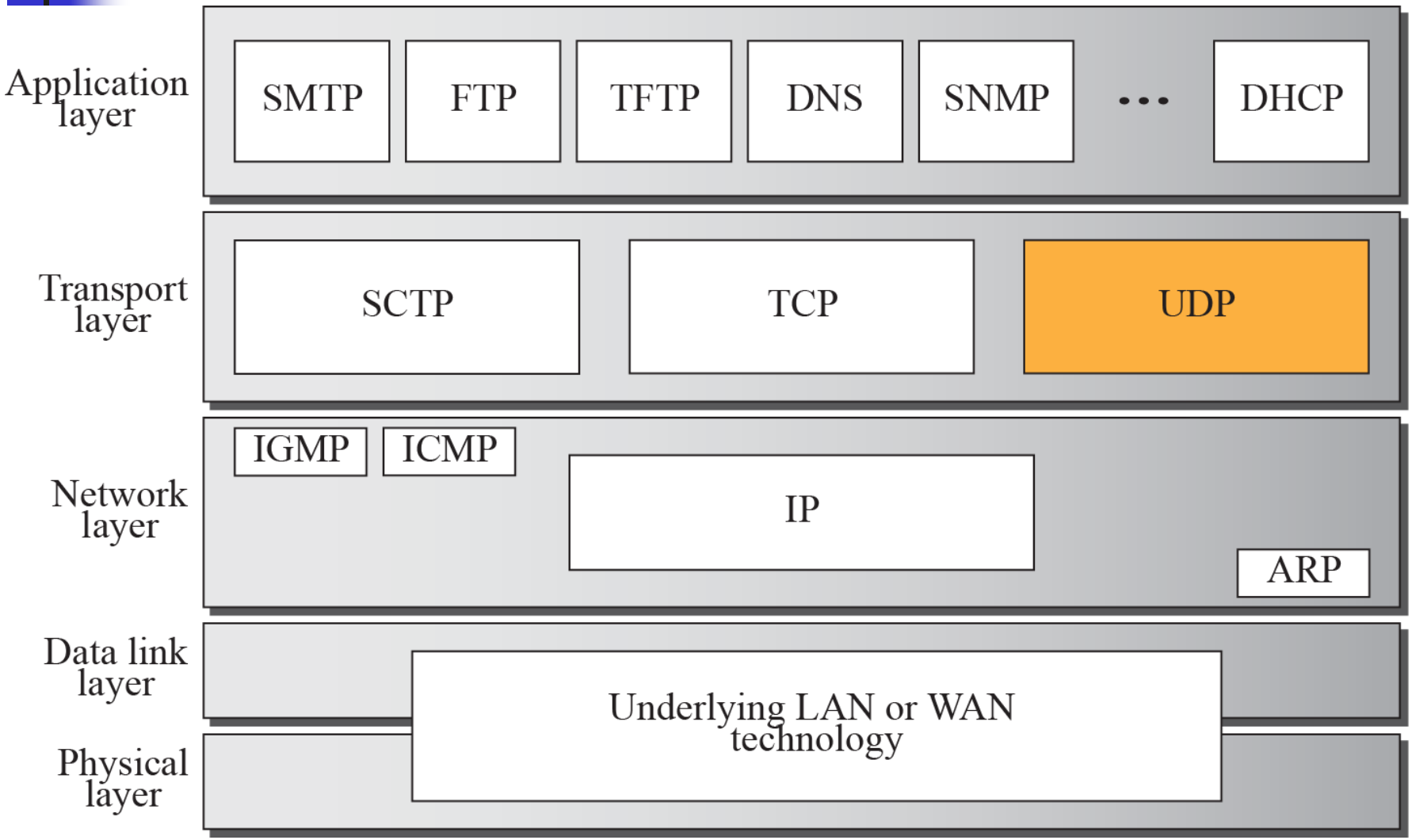
14.4 UDP Application

14.5 UDP Package

14-1 INTRODUCTION

Figure 14.1 shows the relationship of the User Datagram Protocol (UDP) to the other protocols and layers of the TCP/IP protocol suite: UDP is located between the application layer and the IP layer, and serves as the intermediary between the application programs and the network operations.

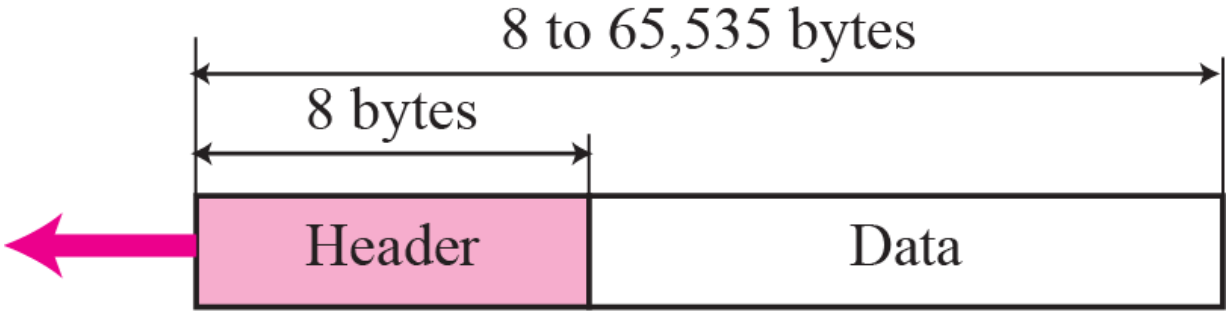
Figure 14.1 *Position of UDP in the TCP/IP protocol suite*



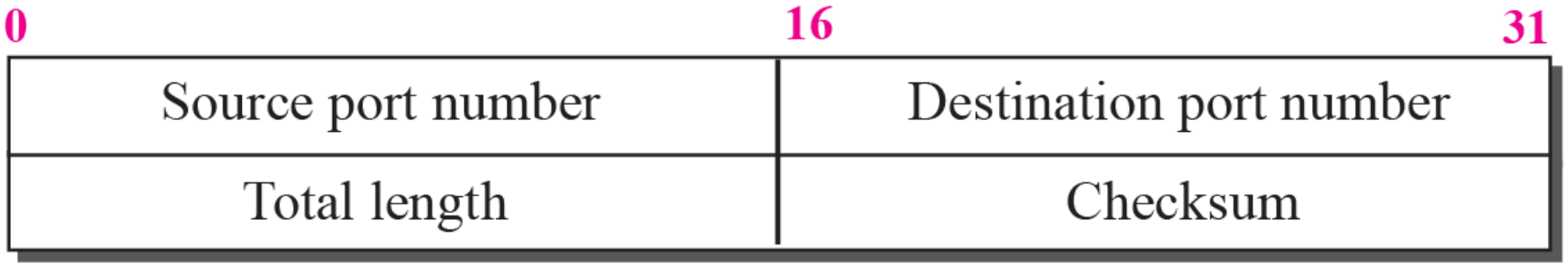
14-2 USER DATAGRAM

UDP packets, have a fixed-size header of 8 bytes. Figure shows the format of a user datagram.

Figure 14.2 *User datagram format*



a. UDP user datagram



b. Header format

Example 14.1

The following is a dump of a UDP header in hexadecimal format.

CB84000D001C001C

a. What is the source port number?
b. What is the destination port number?
c. What is the total length of the user datagram?
d. What is the length of the data?

Example 14.1 *Continued*

Solution

a. The source port number is the first four hexadecimal digits $(CB84)_{16}$ or 52100.

b. The destination port number is the second four hexadecimal digits $(000D)_{16}$ or 13.

c. The third four hexadecimal digits $(001C)_{16}$ define the length of the whole UDP packet as 28 bytes.

d. The length of the data is the length of the whole packet minus the length of the header, or 28 – 8 = 20 bytes.

UDP Services

In this section, we discuss services provided by UDP.

Topics Discussed in the Section

- ✓ **Process-to-Process Communication**
- ✓ **Connectionless Service**
- ✓ **No Flow Control**
- ✓ **No Error Control except checksum**
- ✓ **Encapsulation and Decapsulation**
- ✓ **Queuing**
- ✓ **Multiplexing and Demultiplexing**

Figure 14.5 *Encapsulation and decapsulation*

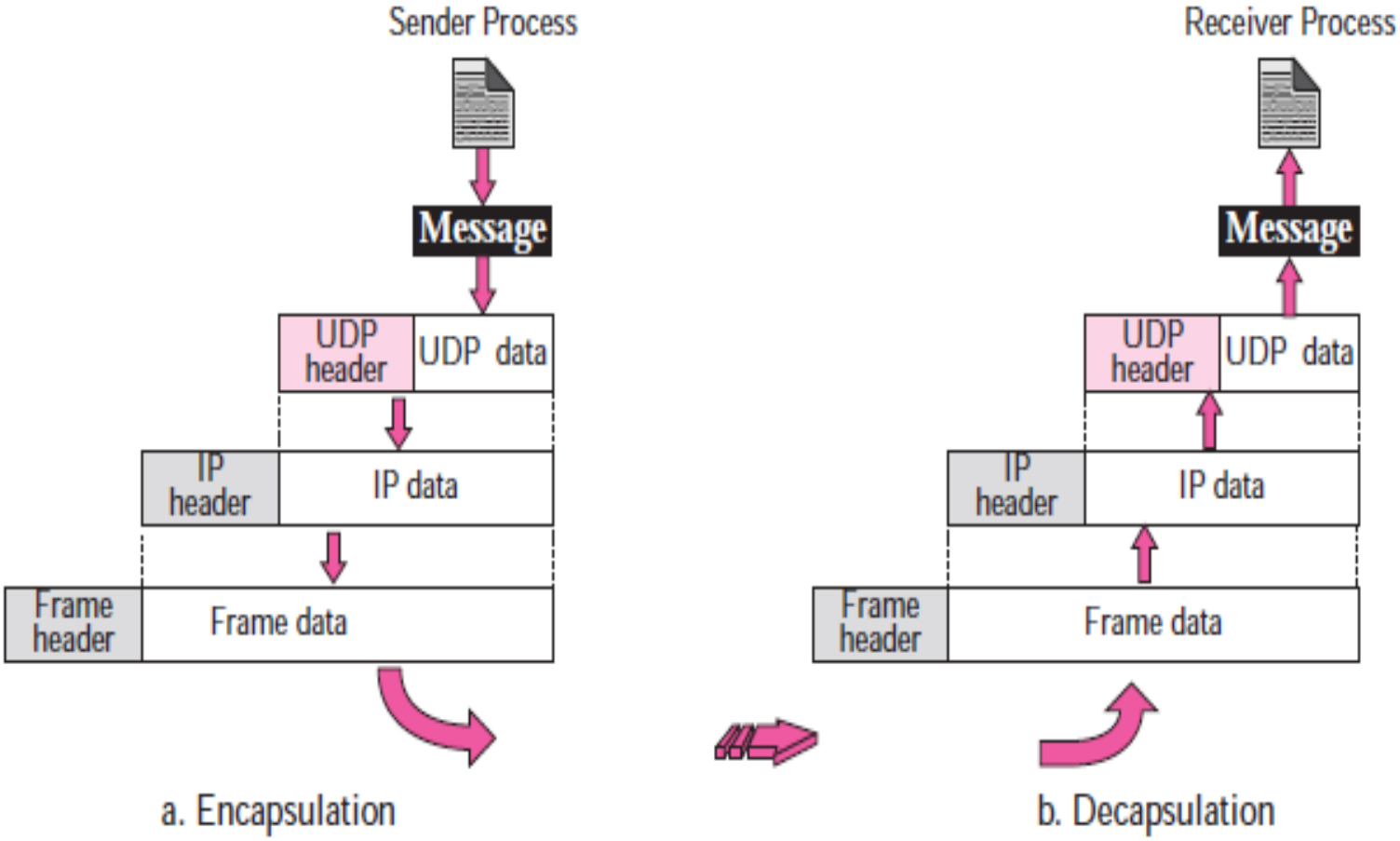
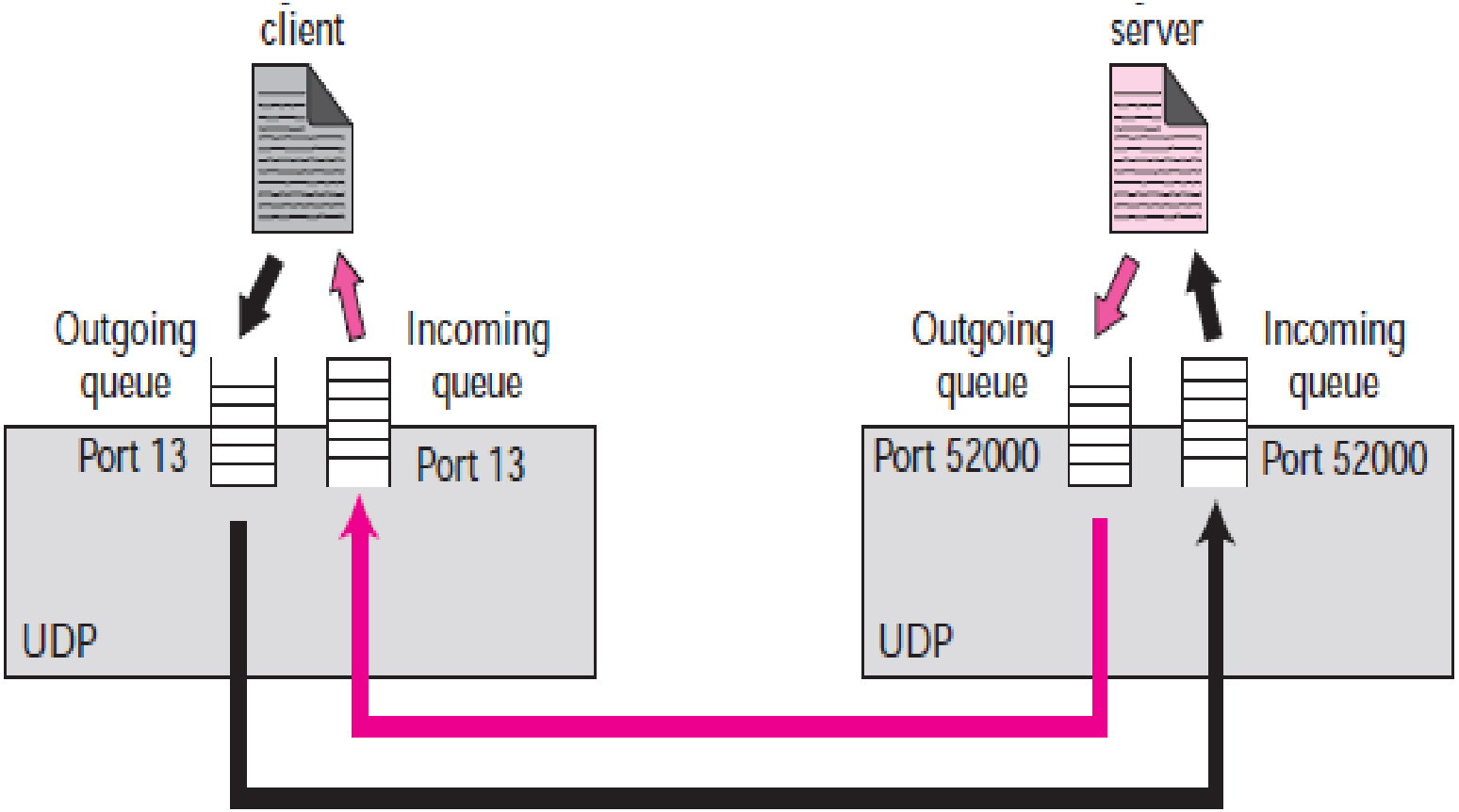
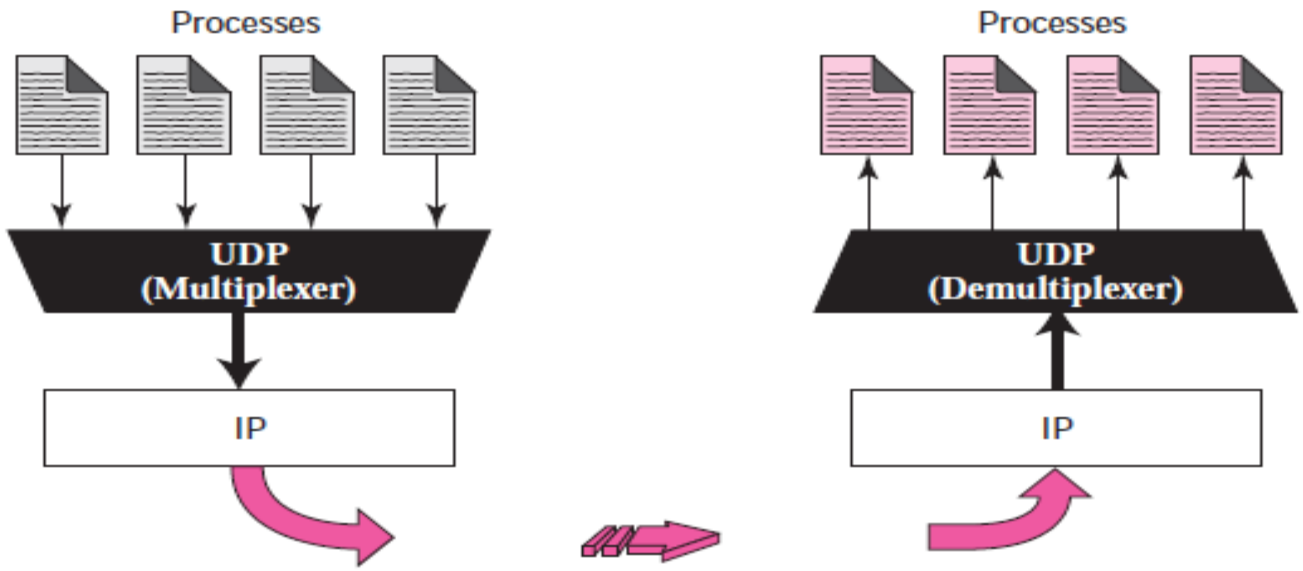


Figure 14.6 *Queues in UDP*





Multiplexing

At the sender site, there may be several processes that need to send user datagrams. However, there is only one UDP. This is a many-to-one relationship and requires multiplexing. UDP accepts messages from different processes, differentiated by their assigned port numbers. After adding the header, UDP passes the user datagram to IP.

Demultiplexing

At the receiver site, there is only one UDP. However, we may have many processes that can receive user datagrams. This is a one-to-many relationship and requires demultiplexing. UDP receives user datagrams from IP. After error checking and dropping of the header, UDP delivers each message to the appropriate process based on the port numbers.

10-5 CHECKSUM

The error detection method we discuss here is called the checksum.

Example 10.18

Suppose our data is a list of five 4-bit numbers that we want to send to a destination. In addition to sending these numbers, we send the sum of the numbers. For example, if the set of numbers is (7, 11, 12, 0, 6), we send (7, 11, 12, 0, 6, 36), where 36 is the sum of the original numbers. The receiver adds the five numbers and compares the result with the sum. If the two are the same, the receiver assumes no error, accepts the five numbers, and discards the sum. Otherwise, there is an error somewhere and the data are not accepted.

Example 10.19

We can make the job of the receiver easier if we send the negative (complement) of the sum, called the checksum. In this case, we send (7, 11, 12, 0, 6, −36). The receiver can add all the numbers received (including the checksum). If the result is 0, it assumes no error; otherwise, there is an error.

Example 10.20

How can we represent the number 21 in one's complement arithmetic using only four bits?

Solution

The number 21 in binary is 10101 (it needs five bits). We can wrap the leftmost bit and add it to the four rightmost bits. We have (0101 + 1) = 0110 or 6.

Example 10.21

How can we represent the number −6 in one's complement arithmetic using only four bits?

Solution

In one's complement arithmetic, the negative or complement of a number is found by inverting all bits. Positive 6 is 0110; negative 6 is 1001. If we consider only unsigned numbers, this is 9.

Example 10.22

using one's complement arithmetic. Figure 10.24 shows the process at the sender and at the receiver. The sender initializes the checksum to 0 and adds all data items and the checksum (the checksum is considered as one data item and is shown in color). The result is 36. However, 36 cannot be expressed in 4 bits. The extra two bits are wrapped and added with the sum to create the wrapped sum value 6. In the figure, we have shown the details in binary. The sum is then complemented, resulting in the checksum value 9 (15 – 6 = 9). The sender now sends six data items to the receiver including the checksum 9.

Example 10.22 (continued)

The receiver follows the same procedure as the sender. It adds all data items (including the checksum); the result is 45. The sum is wrapped and becomes 15. The wrapped sum is complemented and becomes 0. Since the value of the checksum is 0, this means that the data is not corrupted. The receiver drops the checksum and keeps the other data items. If the checksum is not zero, the entire packet is dropped.

Figure 10.24 *Example 10.22*

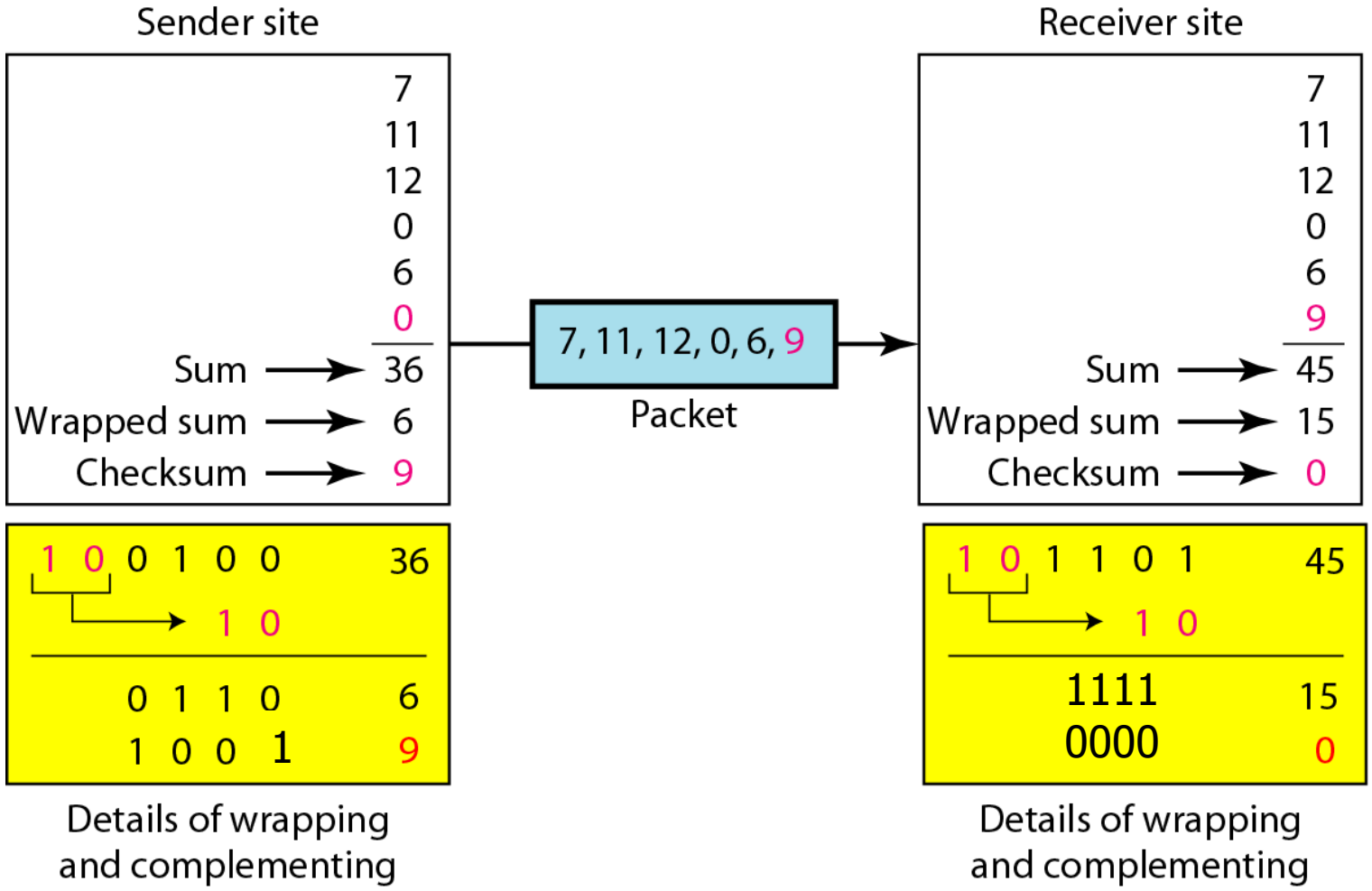
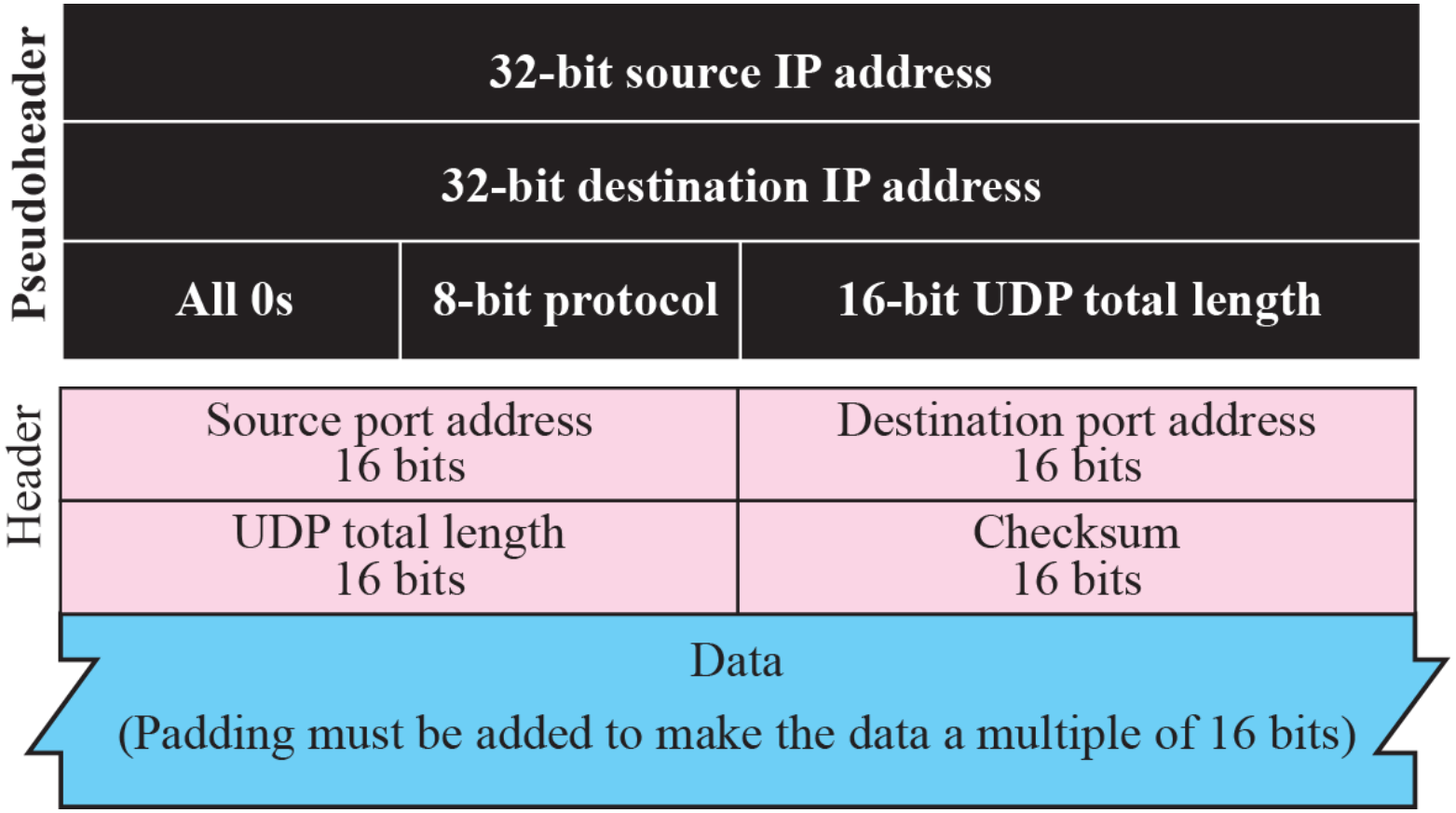


Figure 14.3 *Pseudoheader for checksum calculation*



checksum includes three sections: a pseudoheader, the UDP header, and the data coming from the application layer.

The **pseudoheader** is the part of the header of the IP packet in which the user datagram is to be encapsulated with some fields filled with 0s (see Figure 14.3).

Example 14.2

Figure 14.4 shows the checksum calculation for a very small user datagram with only 7 bytes of data. Because the number of bytes of data is odd, padding is added for checksum calculation. The pseudoheader as well as the padding will be dropped when the user datagram is delivered to IP (see Appendix F).

Figure 14.3 *Pseudoheader for checksum calculation*

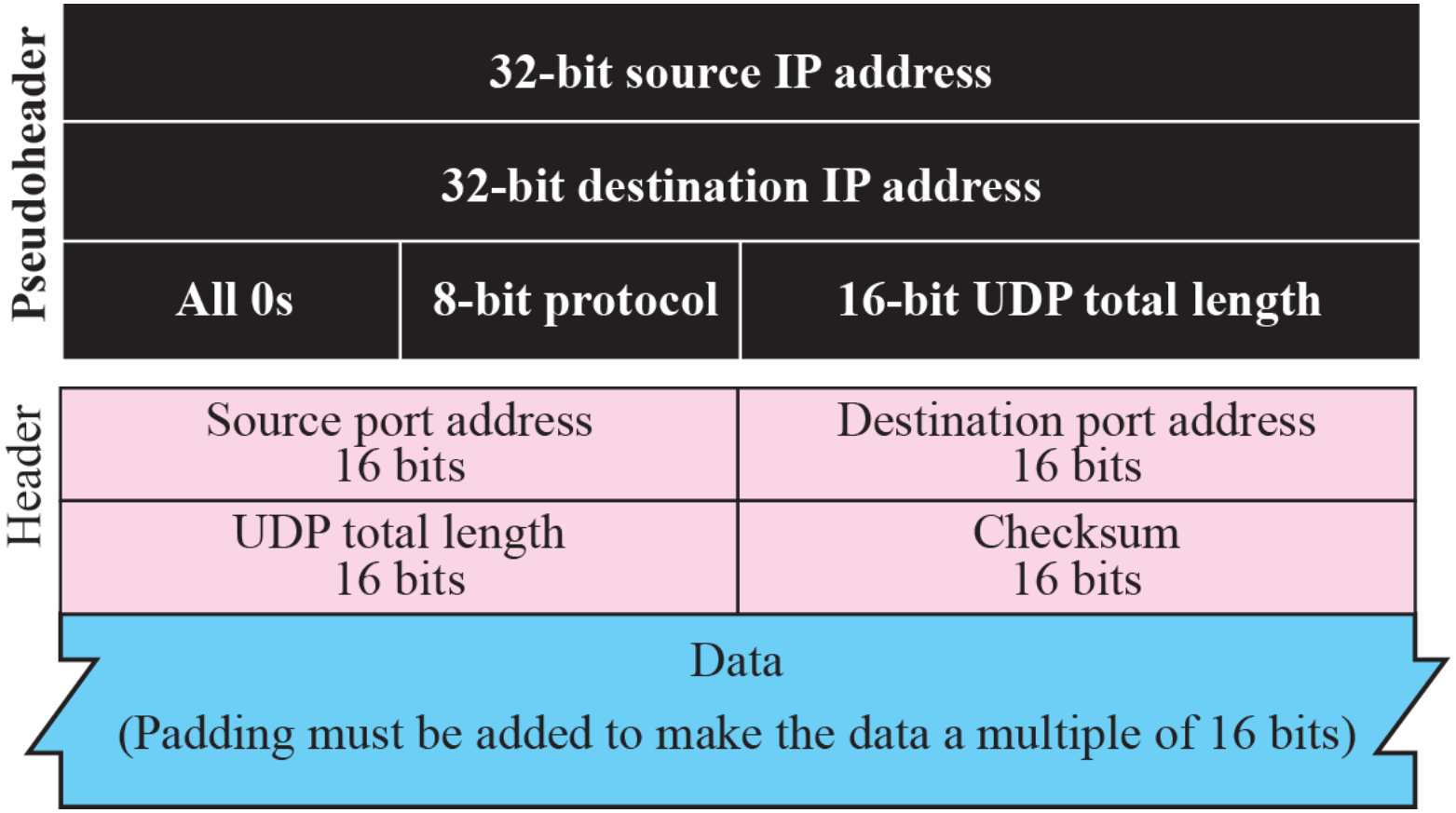
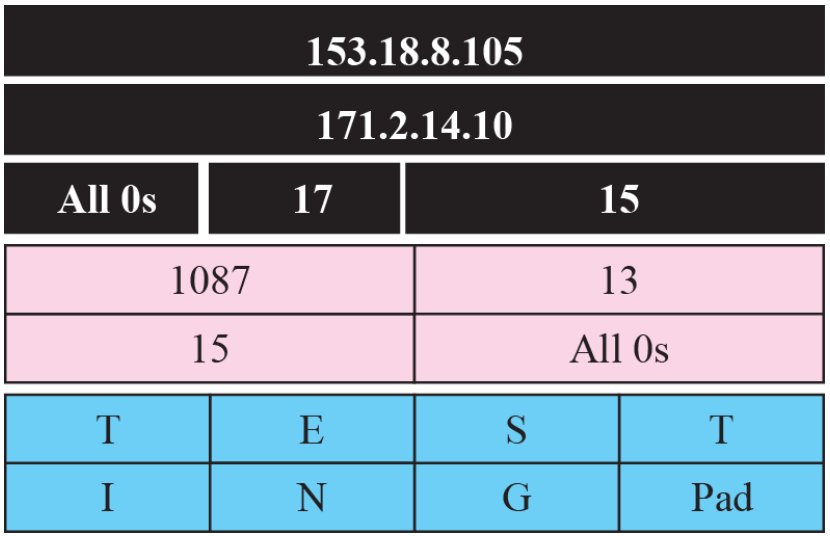
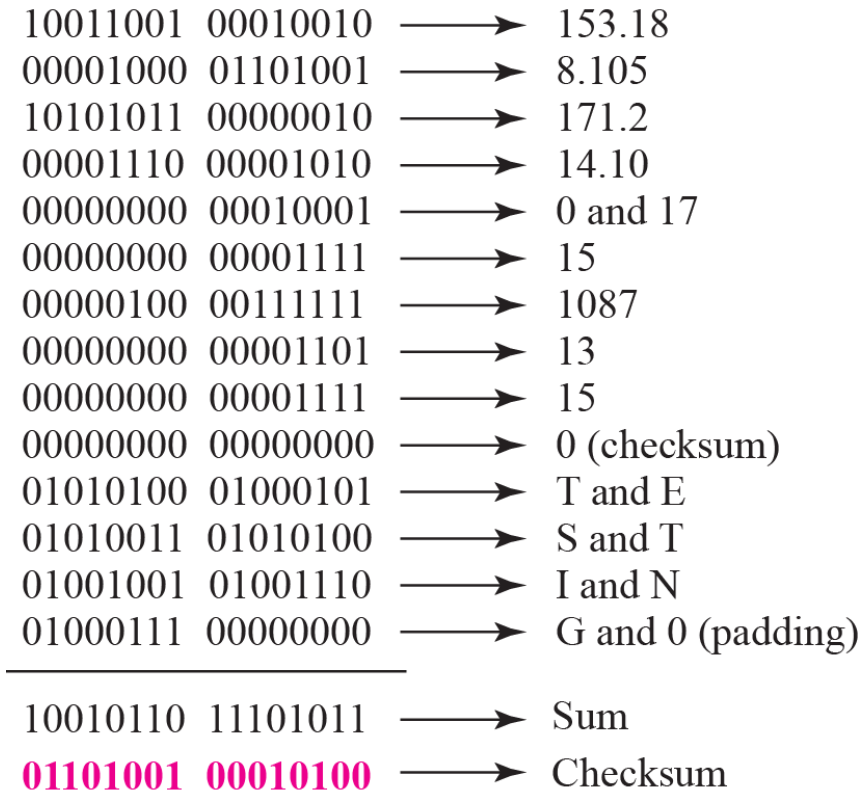


Figure 14.4 *Checksum calculation for a simple UDP user datagram*





Note

UDP is an example of the connectionless simple protocol with the exception of an optional checksum added to packets for error detection.

UDP APPLICATION

Example 14.4

A client-server application such as DNS uses the services of UDP because a client needs to send a short request to a server and to receive a quick response from it. The request and response can each fit in one user datagram. Since only one message is exchanged in each direction, the connectionless feature is not an issue; the client or server does not worry that messages are delivered out of order.

Example 14.5

A client-server application such as SMTP ,which is used in electronic mail, cannot use the services of UDP because a user can send a long e-mail message, which may include multimedia (images, audio, or video). If the application uses UDP and the message does not fit in one single user datagram, the message must be split by the application into different user datagrams. Here the connectionless service may create problems. The user datagrams may arrive and be delivered to the receiver application out of order. The receiver application may not be able to reorder the pieces. This means the connectionless service has a disadvantage for an application program that sends long messages.

Example 14.6

Assume we are downloading a very large text file from the Internet. We definitely need to use a transport layer that provides reliable service. We don't want part of the file to be missing or corrupted when we open the file. The delay created between the delivery of the parts are not an overriding concern for us; we wait until the whole file is composed before looking at it. In this case, UDP is not a suitable transport layer.

Example 14.7

Assume we are watching a real-time stream video on our computer. Such a program is considered a long file; it is divided into many small parts and broadcast in real time. The parts of the message are sent one after another. If the transport layer is supposed to resend a corrupted or lost frame, the synchronizing of the whole transmission may be lost. The viewer suddenly sees a blank screen and needs to wait until the second transmission arrives. This is not tolerable. However, if each small part of the screen is sent using one single user datagram, the receiving UDP can easily ignore the corrupted or lost packet and deliver the rest to the application program. That part of the screen is blank for a very short period of the time, which most viewers do not even notice. However, video cannot be viewed out of order, so streaming audio, video, and voice applications that run over UDP must reorder or drop frames that are out of sequence.

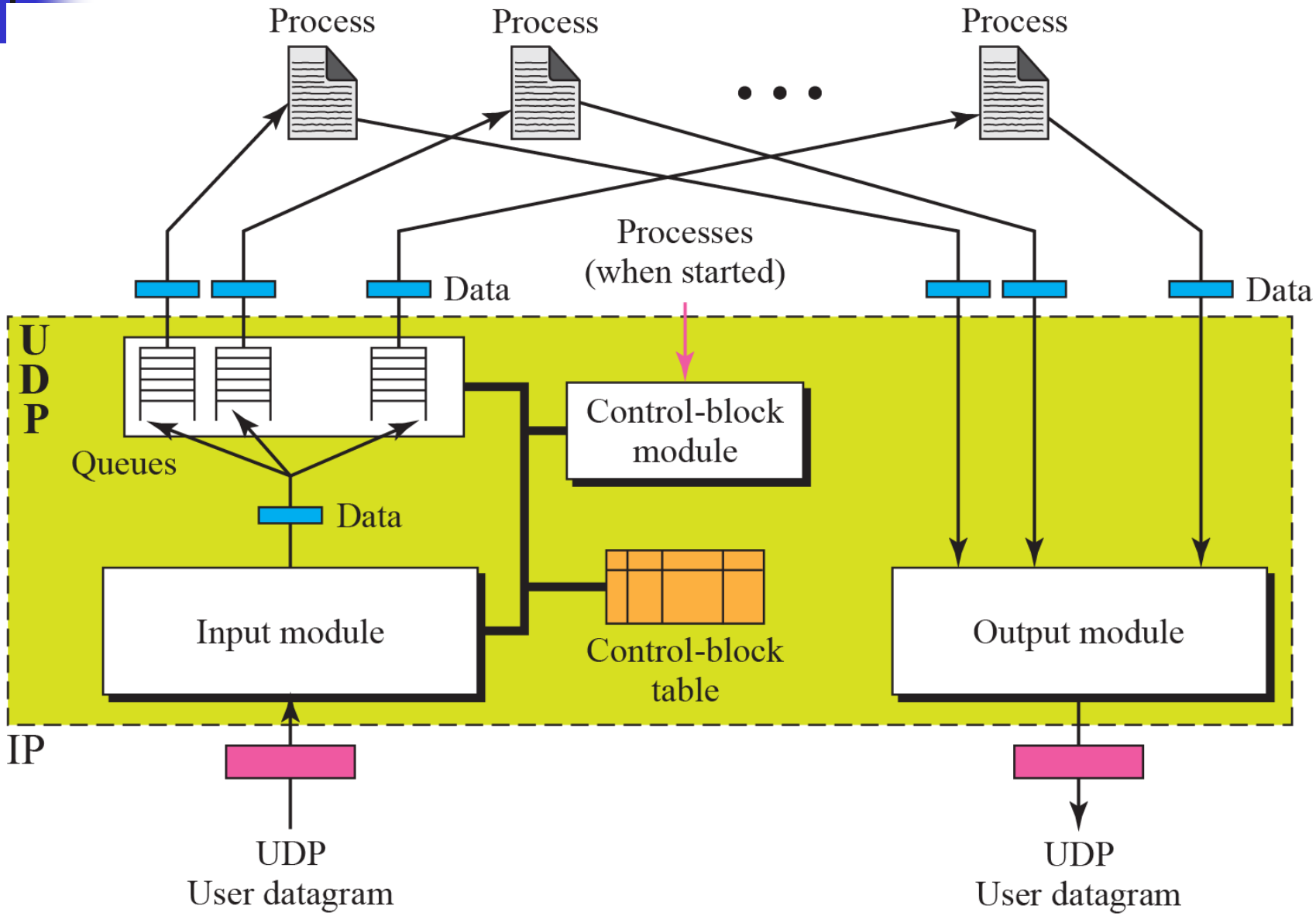
14-5 UDP PACKAGE

To show how UDP handles the sending and receiving of packets, we present a simple version of the UDP package.

We can say that the UDP package involves five components: a control-block table, input queues, a control-block module, an input module, and an output module.

- ✓ **Control-Block Table**
- ✓ **Input Queues**
- ✓ **Control-Block Module**
- ✓ **Input Module**
- ✓ **Output Module**

Figure 14.8 *UDP design*



- **Control-Block Table**
- In our package, UDP has a control-block table to keep track of the open ports.
- Each entry in this table has a minimum of four fields: the state, which can be FREE or IN-USE, the process ID, the port number, and the corresponding queue number.

- **Input Queues**
- Our UDP package uses a set of input queues, one for each process.

Control-Block Module

- The control-block module is responsible for the management of the control-block table. When a process starts, it asks for a port number from the operating system. The operating system assigns well-known port numbers to servers and ephemeral port numbers to clients.
- The process passes the process ID and the port number to the control-block module to create an entry in the table for the process.

Table 14.2 *Control Block Module*

```
1  UDP_Control_Block_Module (process ID, port number)
2  {
3       Search the table for a FREE entry.
4       if (not found)
5            Delete one entry using a predefined strategy.
6       Create a new entry with the state IN-USE
7       Enter the process ID and the port number.
8       Return.
9  } // End module
```

Input module

- The input module receives a user datagram from the IP. It searches the control-block table to find an entry having the same port number as this user datagram.
- If the entry is found, the module uses the information given to enqueue the data.
- If the entry is not found, it generates an ICMP message.

Table 14.3 *Input Module*

```
1  UDP_INPUT_Module (user_datagram)
2  {
3       Look for the entry in the control_block table
4       if (found)
5       {
6            Check to see if a queue is allocated
7            If (queue is not allocated)
8                 allocate a queue
9            else
10                enqueue the data
11      } //end if
12      else
13      {
14           Ask ICMP to send an "unreachable port" message
15           Discard the user datagram
16      } //end else
17
18      Return.
19 } // end module
```

Output module

- The output module is responsible for creating and sending user datagrams.

Table 14.4 *Output Module*

```
1  UDP_OUTPUT_MODULE (Data)
2  {
3       Create a user datagram
4       Send the user datagram
5       Return.
6  }
```