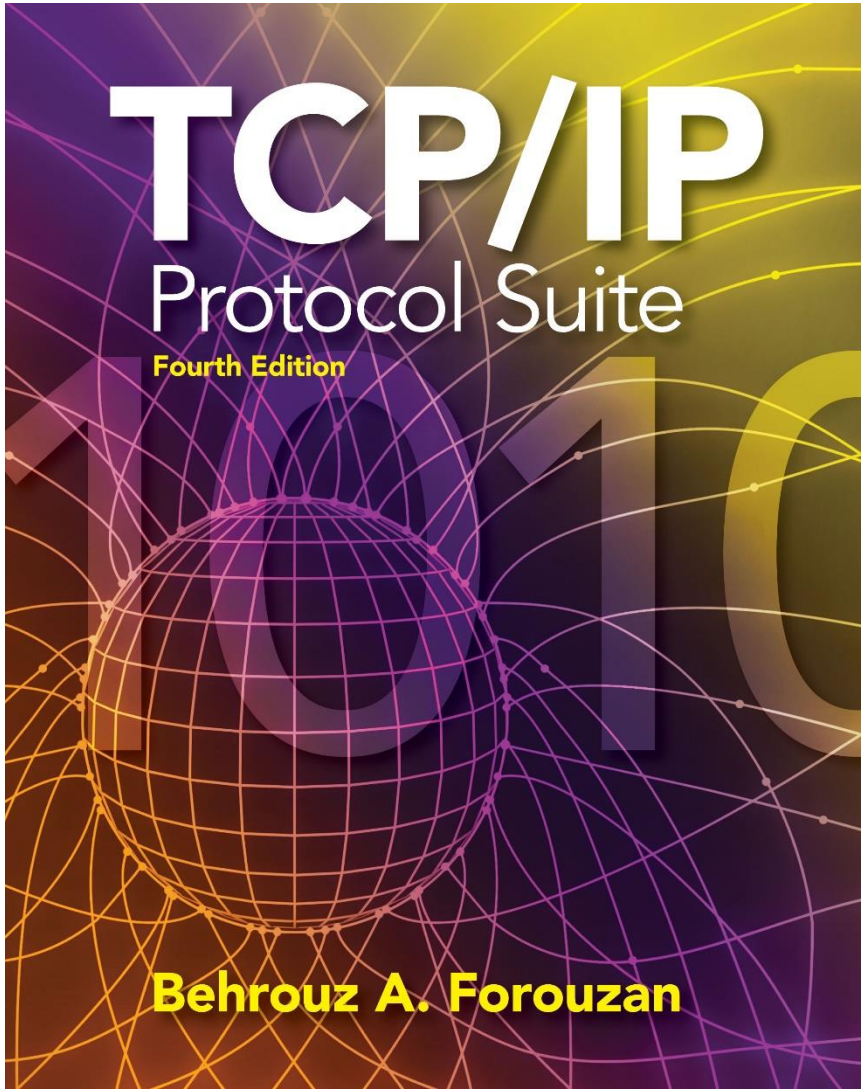
Chapter 7

Internet Protocol Version4 (IPv4)



OBJECTIVES:

- To explain the general idea behind the IP protocol and the position of IP in TCP/IP protocol suite.
- To show the general format of an IPv4 datagram.
- To discuss fragmentation and reassembly of datagrams.
- To discuss several options that can be in an IPv4 datagram and their applications.
- To show how a checksum is calculated for the header of an IPv4 datagram at the sending site and how the checksum is checked at the receiver site.
- To discuss IP over ATM and compare it with IP over LANs and/or point-to-point WANs.
- To show a simplified version of the IP package and give the pseudocode for some modules.

Chapter Outline

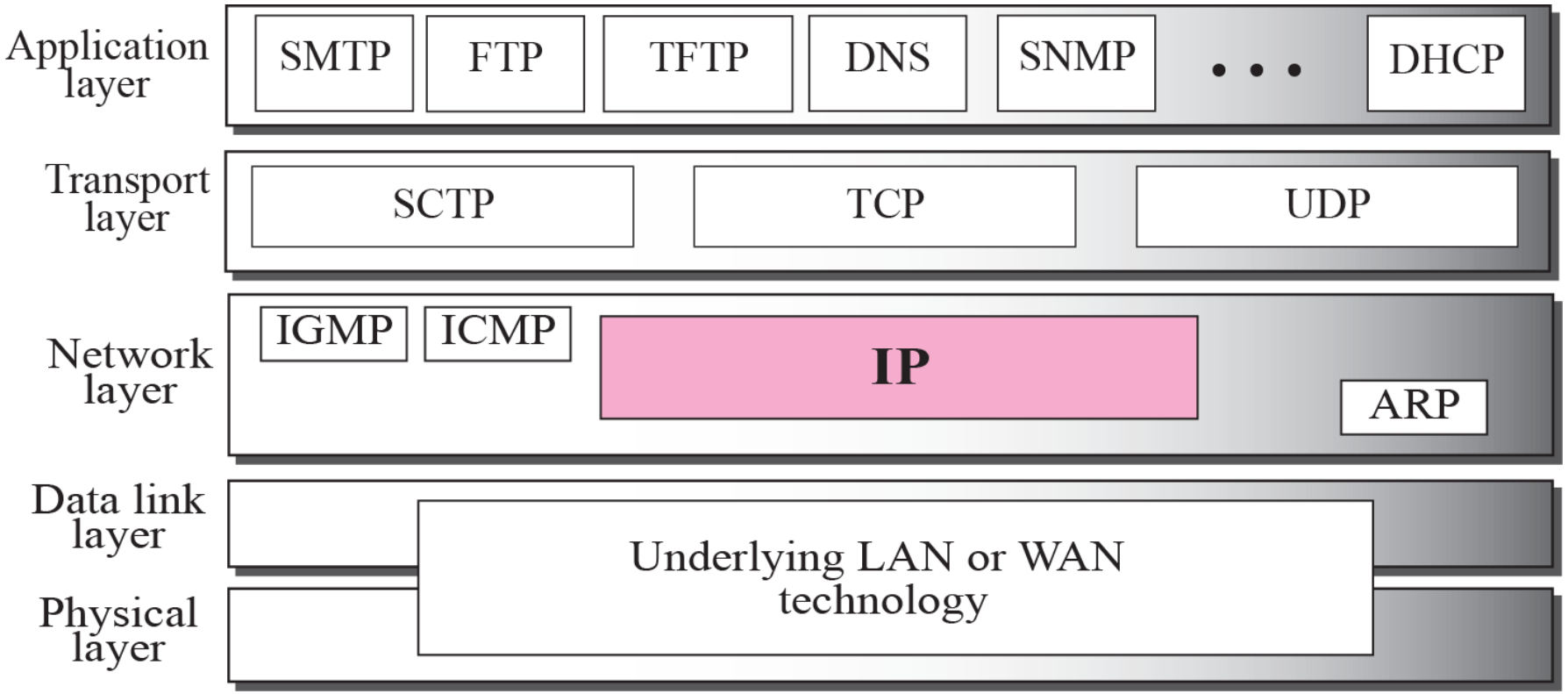
7-1 INTRODUCTION

The Internet Protocol (IP) is the transmission mechanism used by the TCP/IP protocols at the network layer.

Topics Discussed in the Section

- ✓ **Relationship of IP to the rest of the TCP/IP Suite**

Figure 7.1 *Position of IP in TCP/IP protocol suite*



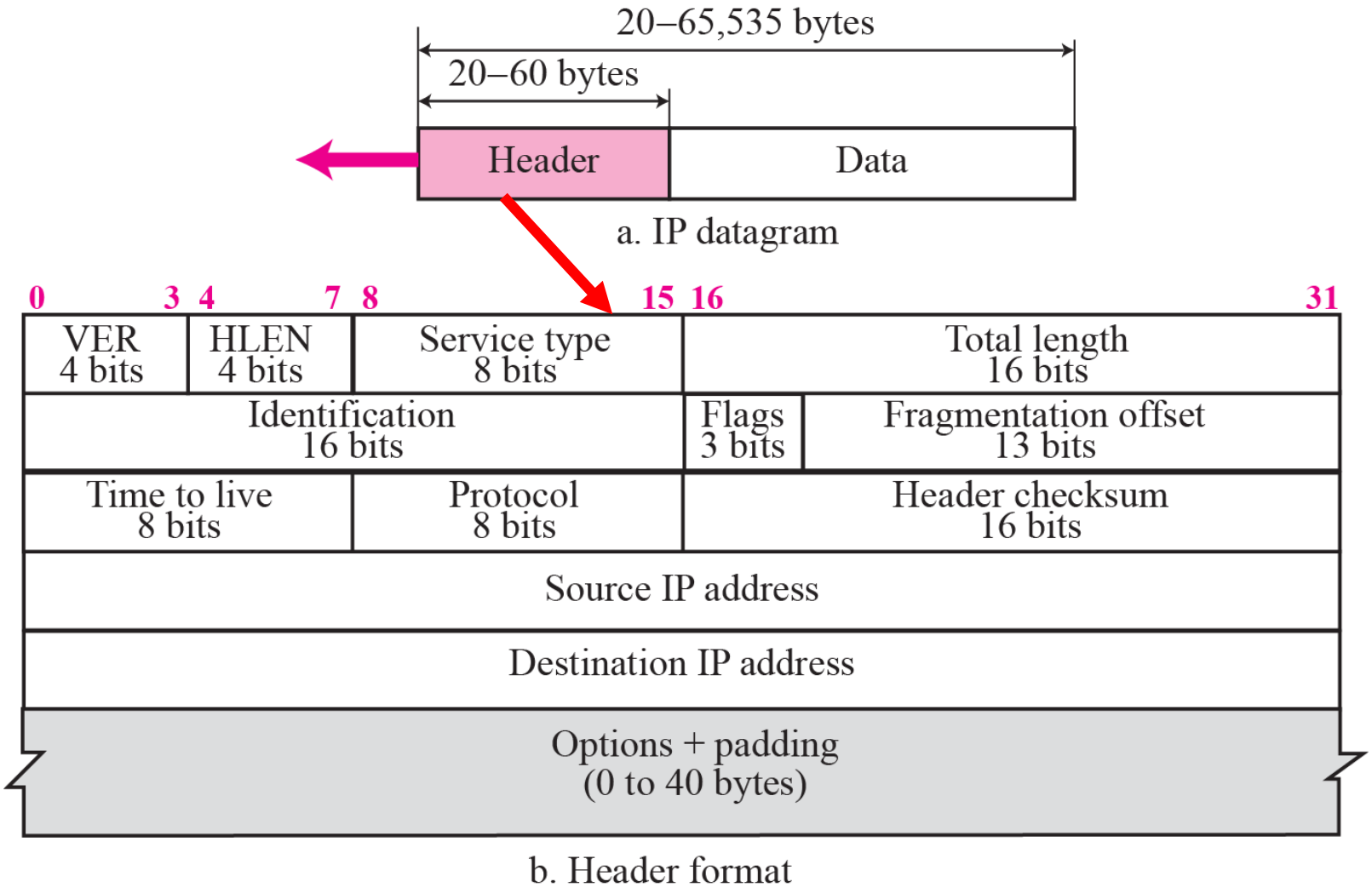
7-2 DATAGRAMS

Packets in the network (internet) layer are called *datagrams*. A datagram is a variable-length packet consisting of two parts: header and data. The header is 20 to 60 bytes in length and contains information essential to routing and delivery. It is customary in TCP/IP to show the header in 4-byte sections. A brief description of each field is in order.

Topics Discussed in the Section

- ✓ **Format of the datagram packet**
- ✓ **Some examples**

Figure 7.2 *IP datagram*

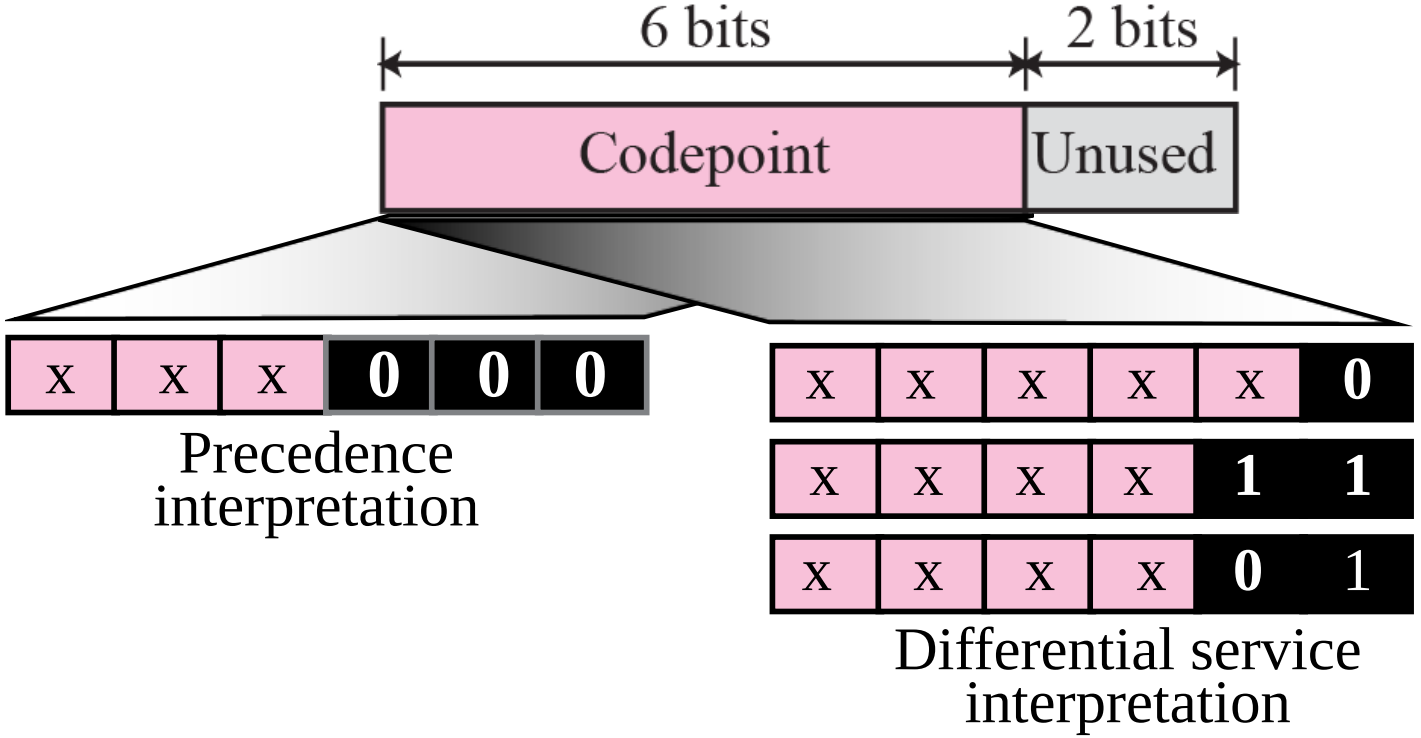


IP datagram fields

- **Version (VER).** This 4-bit field defines the version of the IP protocol.
- **Header length (HLEN).** This 4-bit field defines the total length of the datagram header in 4-byte words.
- **Service type.** In the original design of IP header, this field was referred to as type of service (TOS), which defined how the datagram should be handled. Part of the field was used to define the precedence**(**الأولوية**)** of the datagram; the rest defined the type of service (low delay, high throughput, and so on).
- **Total length**. This is a 16-bit field that defines the total length (header plus data) of the IP datagram in bytes.
- **Identification.** This field is used in fragmentation (discussed in the next section).
- **Flags.** This field is used in fragmentation (discussed in the next section).
- **Fragmentation offset**. This field is used in fragmentation (discussed in the next section).
- **Time to live.** A datagram has a limited lifetime in its travel through an internet.
- **Protocol.** This 8-bit field defines the higher-level protocol that uses the services of the IP layer.
- **Checksum.** The checksum concept and its calculation are discussed later in this chapter.

Figure 7.3 *Service type*

At now. IETF has changed the interpretation of this 8-bit field. This field now defines a set of differentiated services. The new interpretation is shown in Figure 7.3.



When the 3 right-most bits are 0s, the 3 left-most bits are interpreted the same as the precedence bits in the service type interpretation. Define the priority of the datagram). If a router is congested and needs to discard some datagrams, those datagrams with lowest precedence are discarded first.

- When the 3 right-most bits are not all 0s, the 6 bits define 56 (64 - 8) services based on the priority assignment by the Internet or local authorities(سلطة). Define the service type

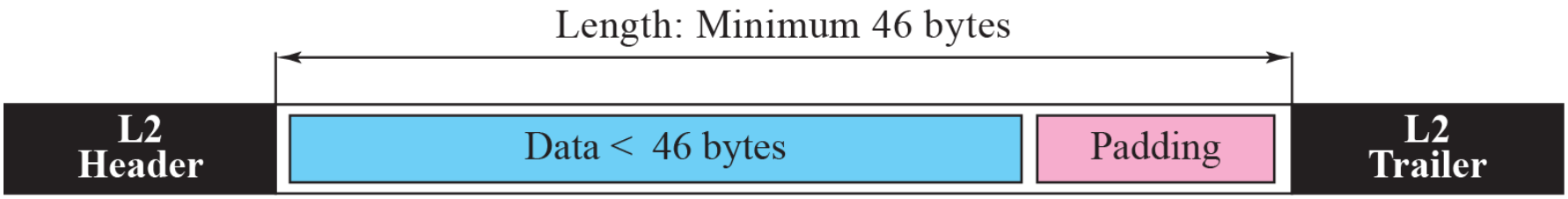
Table 7.1 *Values for codepoints*

Category	*Codepoint*	*Assigning Authority*
1	XXXXX0	Internet
2	XXXX11	Local
3	XXXX01	Temporary or experimental

Note

The total length field defines the total length of the datagram including the header.

Figure 7.4 *Encapsulation of a small datagram in an Ethernet frame*



❑ **Protocol.** This 8-bit field defines the higher-level protocol that uses the services of the IP layer. An IP datagram can encapsulate data from several higher level protocols such as TCP, UDP, ICMP, and IGMP. This field specifies the final destination protocol to which the IP datagram should be delivered.

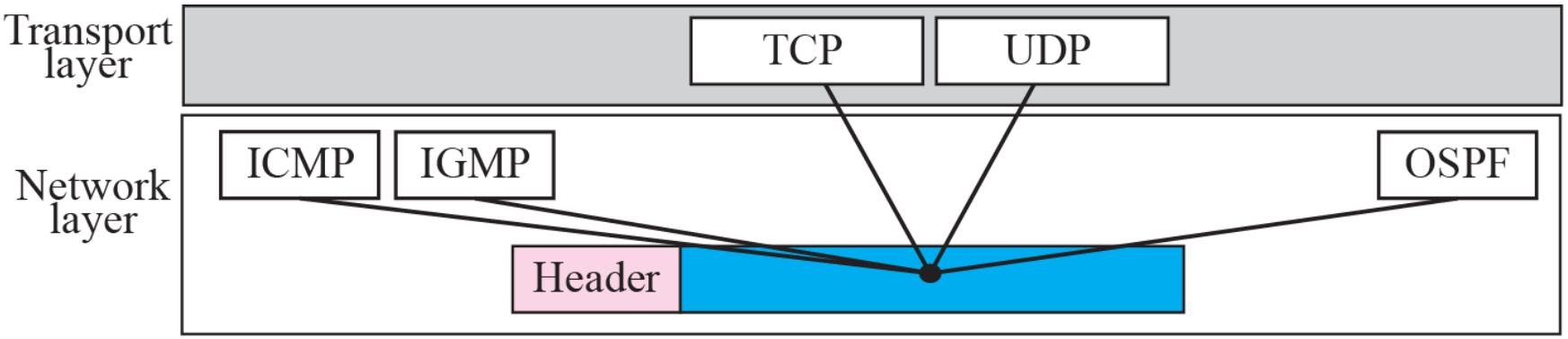


Table 7.2 *Protocols*

Value	*Protocol*	*Value*	*Protocol*
1	ICMP	17	UDP
2	IGMP	89	OSPF
6	TCP		

Example 7.1

An IP packet has arrived with the first 8 bits as shown:

01000010

The receiver discards the packet. Why?

Solution

There is an error in this packet. The 4 left-most bits (0100) show the version, which is correct. The next 4 bits (0010) show the wrong header length ($2 \times 4 = 8$). The minimum number of bytes in the header must be 20. The packet has been corrupted in transmission.

Example 7.2

In an IP packet, the value of HLEN is 1000 in binary. How many bytes of options are being carried by this packet?

Solution

The HLEN value is 8, which means the total number of bytes in the header is 8 × 4 or 32 bytes. The first 20 bytes are the base header, the next 12 bytes are the options.

Example 7.3

In an IP packet, the value of HLEN is 5_{16} and the value of the total length field is 0028_{16}. How many bytes of data are being carried by this packet?

Solution

The HLEN value is 5, which means the total number of bytes in the header is 5 × 4 or 20 bytes (no options). The total length is 40 bytes, which means the packet is carrying 20 bytes of data (40 − 20).

Example 7.4

An IP packet has arrived with the first few hexadecimal digits as shown below:

45000028000100000102 . . .

How many hops can this packet travel before being dropped? The data belong to what upper layer protocol?

Solution

To find the time-to-live field, we skip 8 bytes (16 hexadecimal digits). The time-to-live field is the ninth byte, which is 01. This means the packet can travel only one hop. The protocol field is the next byte (02), which means that the upper layer protocol is IGMP (see Table 7.2)

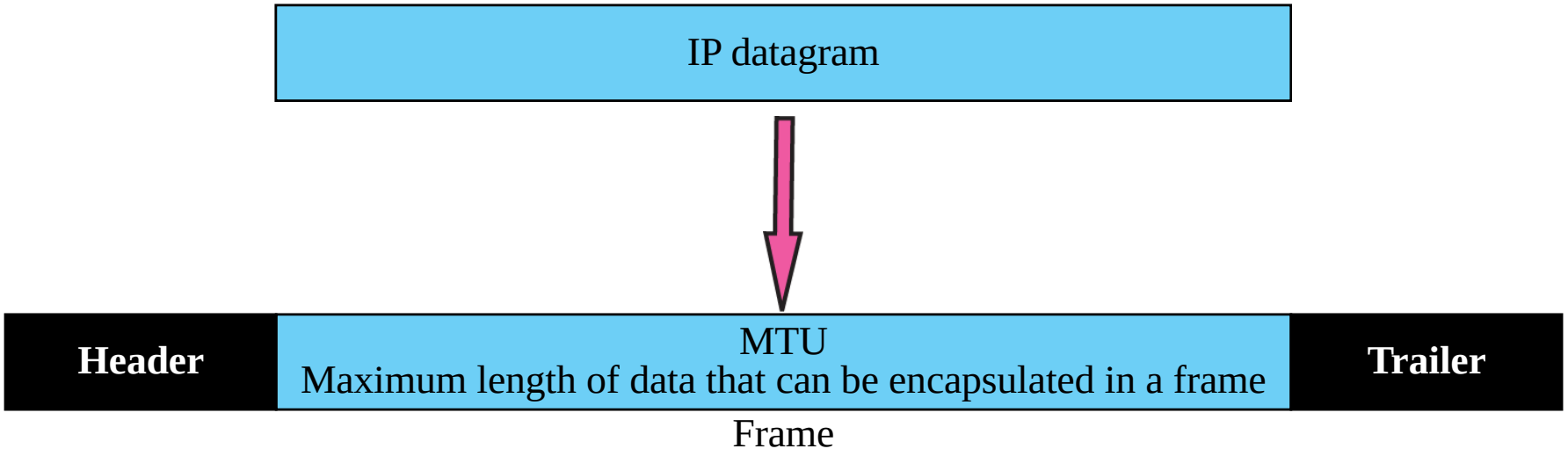
7-3 FRAGMENTATION

A datagram can travel through different networks. Each router decapsulates the IP datagram from the frame it receives, processes it, and then encapsulates it in another frame. The format and size of the received frame depend on the protocol used by the physical network through which the frame has just traveled. The format and size of the sent frame depend on the protocol used by the physical network through which the frame is going to travel.

Topics Discussed in the Section

- ✓ **Maximum Transfer Unit (MTU)**
- ✓ **Fields Related to Fragmentation**

Figure 7.6 *MTU*



When a datagram is encapsulated in a frame, the total size of the datagram must be less than this maximum size, which is defined by the restrictions imposed by the hardware and software used in the network

The value of the MTU differs from one physical network protocol to another. For example, the value for the Ethernet LAN is 1500 bytes, for FDDI LAN is 4352 bytes, and for PPP is 296 bytes.

Note

Only data in a datagram is fragmented.

The host or router that fragments a datagram must change the values of three fields: flags, fragmentation offset, and total length. The rest of the fields must be copied. Of course, the value of the checksum must be recalculated regardless of fragmentation.

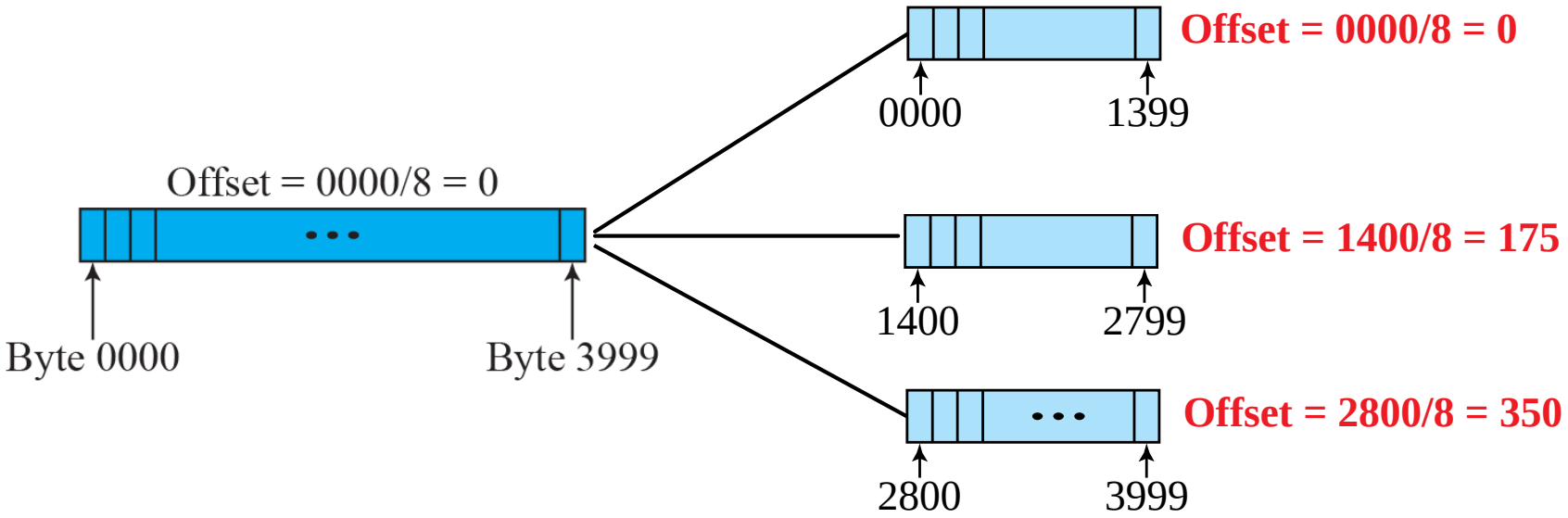
Identification field

The value in the identification field is copied into all fragments. In other words, all fragments have the same identification number, which is also the same as the original datagram. The identification number helps the destination in reassembling the datagram. It knows that all fragments having the same identification value should be assembled into one datagram.

Figure 7.7 *Flags field*

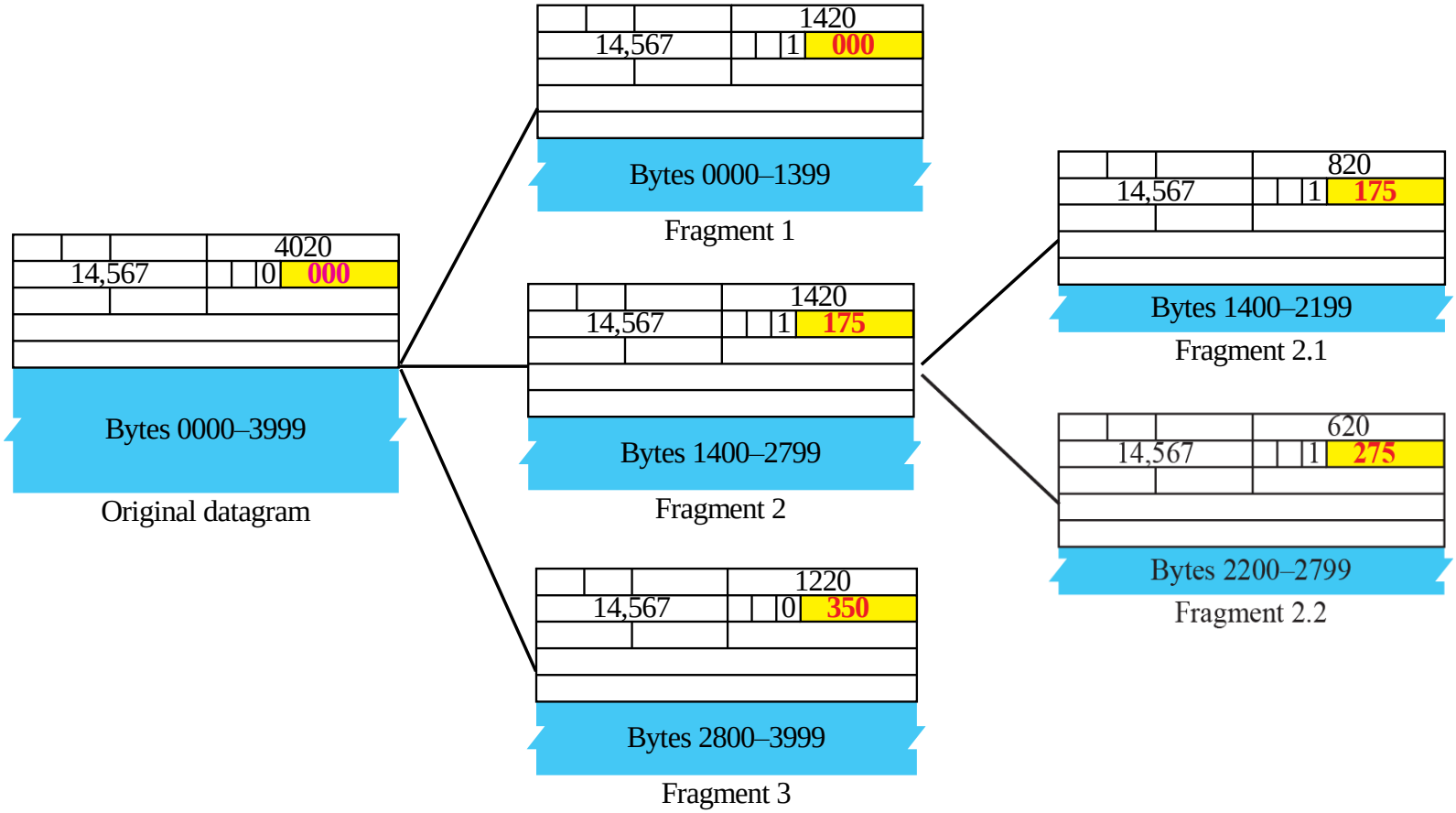
D: Do not fragment
M: More fragments

Figure 7.8 *Fragmentation example*



Offset= First byte /8

Figure 7.9 *Detailed fragmentation example*



Example 7.5

A packet has arrived with an M bit value of 0. Is this the first fragment, the last fragment, or a middle fragment? Do we know if the packet was fragmented?

Solution

If the M bit is 0, it means that there are no more fragments; the fragment is the last one. However, we cannot say if the original packet was fragmented or not. A nonfragmented packet is considered the last fragment.

Example 7.6

A packet has arrived with an M bit value of 1. Is this the first fragment, the last fragment, or a middle fragment? Do we know if the packet was fragmented?

Solution

If the M bit is 1, it means that there is at least one more fragment. This fragment can be the first one or a middle one, but not the last one. We don't know if it is the first one or a middle one; we need more information (the value of the fragmentation offset). See also the next example.

Example 7.7

A packet has arrived with an M bit value of 1 and a fragmentation offset value of zero. Is this the first fragment, the last fragment, or a middle fragment?

Solution

Because the M bit is 1, it is either the first fragment or a middle one. Because the offset value is 0, it is the first fragment.

Example 7.8

A packet has arrived in which the offset value is 100. What is the number of the first byte? Do we know the number of the last byte?

Solution

To find the number of the first byte, we multiply the offset value by 8. This means that the first byte number is 800 (of that fragment). We cannot determine the number of the last byte unless we know the length of the data.

Example 7.9

A packet has arrived in which the offset value is 100, the value of HLEN is 5 and the value of the total length field is 100. What is the number of the first byte and the last byte?

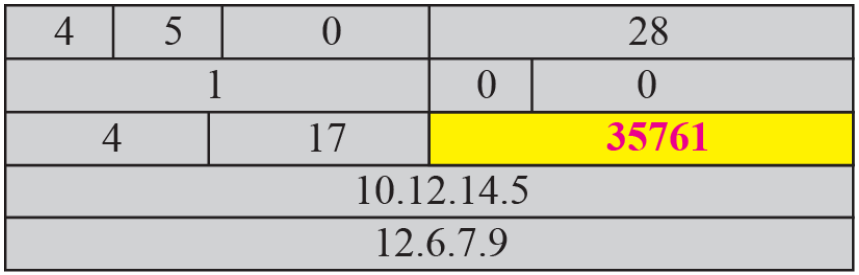
Solution

The first byte number is 100 × 8 = 800. The total length is 100 bytes and the header length is 20 bytes (5 × 4), which means that there are 80 bytes in this datagram. If the first byte number is 800, the last byte number must be 879.

Example 7.18

Figure 7.25 shows the checking of checksum calculation at the receiver site (or intermediate router) assuming that no errors occurred in the header. The header is divided into 16-bit sections. All the sections are added and the sum is complemented. Since the result is 16 0s, the packet is accepted.

Figure 7.25 *Example of checksum calculation at the receiver*



4, 5, and 0	→	01000101 00000000
28	→	00000000 00011100
1	→	00000000 00000001
0 and 0	→	00000000 00000000
4 and 17	→	00000100 00010001
Checksum	→	**10001011 10110001**
10.12	→	00001010 00001100
14.5	→	00001110 00000101
12.6	→	00001100 00000110
7.9	→	00000111 00001001
Sum	→	**1111 1111 1111 1111**
Checksum	→	**0000 0000 0000 0000**