CHAPTER

6 IPv4 and IPv6 Headers

What You Will Learn

In this chapter, you will learn about the IP layer. We'll start with the fields in the IPv4 and IPv6 packet headers. We'll discuss most of the fields in detail and show how many of them relate to each other.

You will learn about fragmentation, and how large content is broken up, spread across a sequence of many packets, and reassembled at the destination. We'll also talk about some of the perceived hazards of this fragmentation process.

Thus far, we've created a network of hosts and routers, linked them with a variety of architectures and link types (LANs and WANs), and discussed the frame formats and methods used to distribute packets among the nodes. We've considered the IPv4 and IPv6 address formats, and the ways that they map to lower, link layer addresses. Now it's time to concentrate on the IP layer itself.

Even casual users of the TCP/IP protocol suite are familiar with the basic IP packet, or, as it was initially called (and still often is) the *datagram.* An IP datagram or packet is the connectionless IP network-layer protocol data unit (PDU). When TCP/IP came along, packets were often associated with connection-oriented data networks such as X.25, the international packet data network standard. To emphasize the connectionless nature of IP, then a radical approach to network layer operation, the TCP/IP developers decided to invent a new term for the IP packet. Through analogy with the telegram (a terse message sent hop by hop through a network of point-to-point links), they came up with the term "datagram."

The IP layer of the whole TCP/IP protocol stack is the very heart of TCP/IP. The frames that are sent and delivered across the network from host to router and router to host contain IP packets. However, like almost all statements about nearly any network protocol, there are exceptions to the general "frames contain IP packets" rule. As shown in the last chapter, an important class of IP layer protocols known as the Address Resolution Protocols (ARPs) does not technically use IP packets, but ARP messages are very close in structure to IP packets. Also, the Internet Control Message Protocol (ICMP) uses IP packets and is included in the IP layer. We'll look at ICMP in the next chapter.

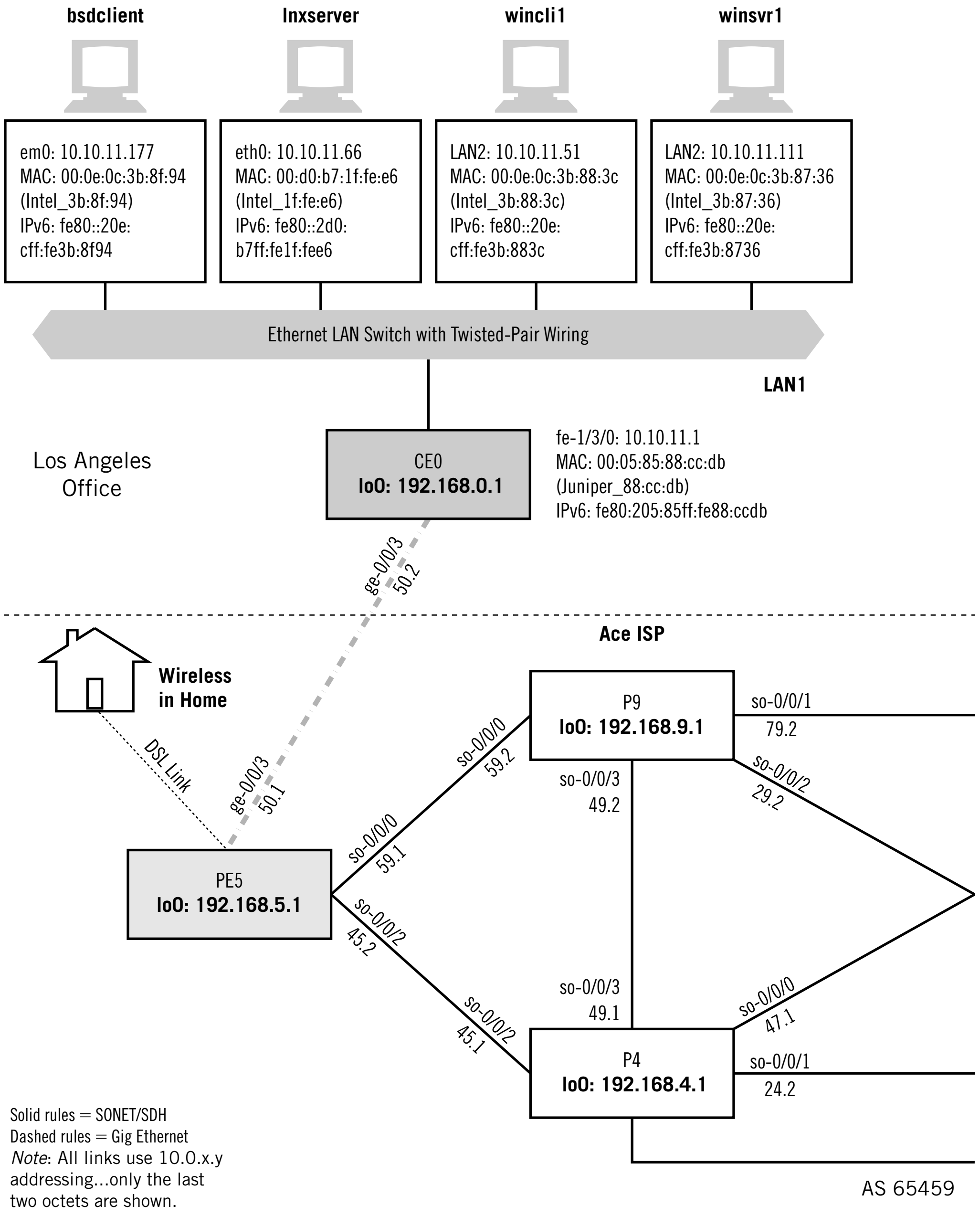
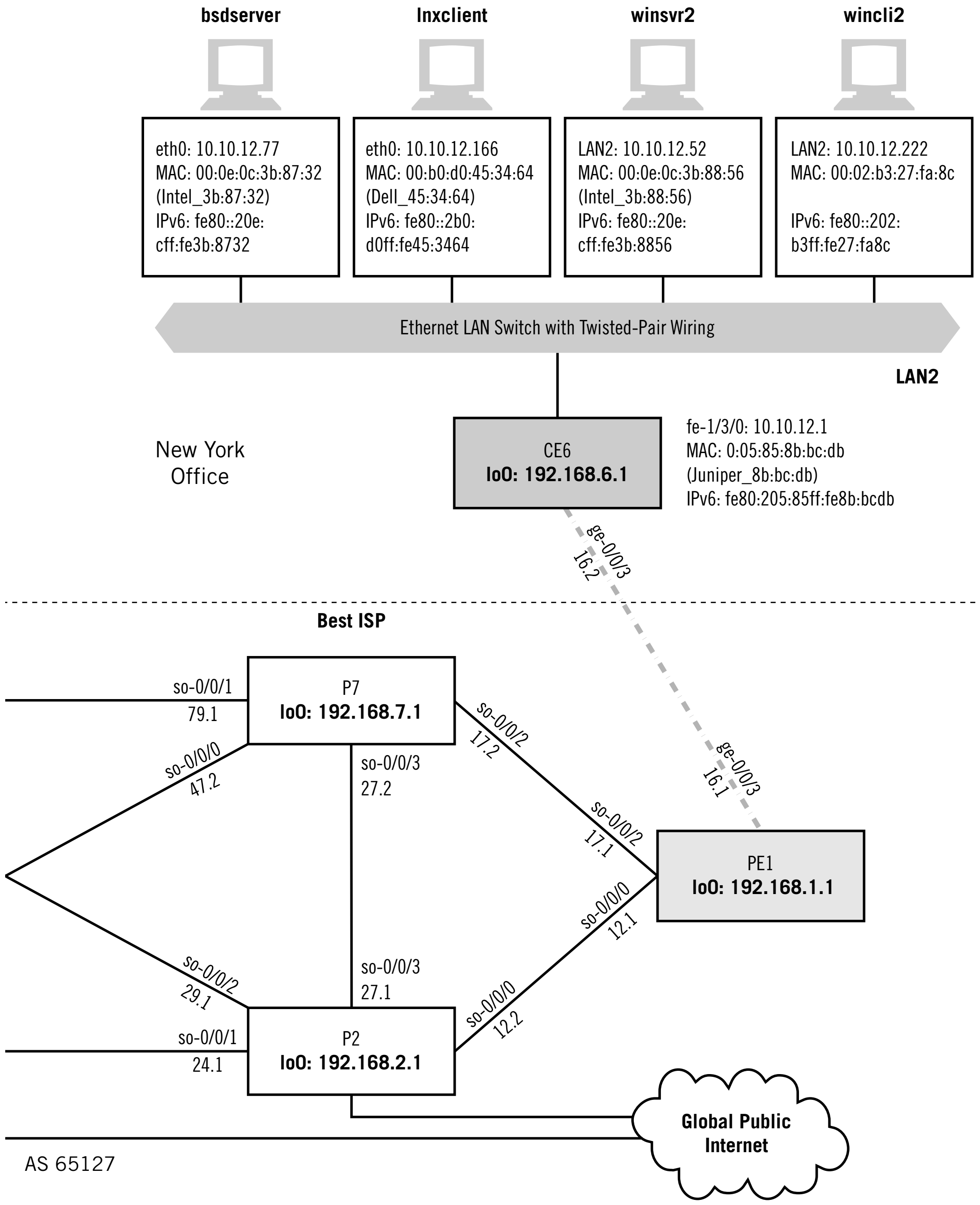


FIGURE 6.1

The LANs on the Illustrated Network use both IPv4 and IPv6 packets. We'll be looking at the headers generated by the hosts on the LANs.


bsdserver
lnxclient
winsvr2
wincli2
eth0: 10.10.12.77
MAC: 00:0e:0c:3b:87:32
(Intel_3b:87:32)
IPv6: fe80::20e:cff:fe3b:8732
eth0: 10.10.12.166
MAC: 00:b0:d0:45:34:64
(Dell_45:34:64)
IPv6: fe80::2b0:d0ff:fe45:3464
LAN2: 10.10.12.52
MAC: 00:0e:0c:3b:88:56
(Intel_3b:88:56)
IPv6: fe80::20e:cff:fe3b:8856
LAN2: 10.10.12.222
MAC: 00:02:b3:27:fa:8c
IPv6: fe80::202:b3ff:fe27:fa8c
Ethernet LAN Switch with Twisted-Pair Wiring
LAN2
New York Office
CE6
lo0: 192.168.6.1
fe-1/3/0: 10.10.12.1
MAC: 0:05:85:8b:bc:db
(Juniper_8b:bc:db)
IPv6: fe80:205:85ff:fe8b:bcdb
ge-0/0/3
16.2
Best ISP
so-0/0/1
79.1
P7
lo0: 192.168.7.1
so-0/0/0
47.2
so-0/0/3
27.2
so-0/0/2
17.2
ge-0/0/3
16.1
so-0/0/2
17.1
PE1
lo0: 192.168.1.1
so-0/0/0
12.1
so-0/0/2
29.1
so-0/0/3
27.1
so-0/0/0
12.2
so-0/0/1
24.1
P2
lo0: 192.168.2.1
Global Public Internet
AS 65127

Both IPv4 and IPv6 packet structures will be detailed in this chapter. However, for the sake of simplicity, whenever the term "IP" is used without qualification, "IPv4" is implied.

PACKET HEADERS AND ADDRESSES

Let's take a close look at the packets used on the Illustrated Network. We'll look at the IPv4 header and addresses first. We worked with the Windows clients and servers a lot in the last few chapters, and we'll work with them again in this chapter. But we'll also work with the Unix devices and tethereal captures in this chapter, especially for fragmentation and IPv6. And, as we'll soon see, one of the biggest differences between IPv4 and IPv6 is how fragmentation is handled.

Fragmentation

People talk loosely about the pros and cons of "IP packet fragmentation," but this terminology is not correct. It is not the IP packet itself that is fragmented, but the packet *content*. If the payload is too large to fit inside a single IP packet (as determined by the IP layer implementation), the content is spread across several packets, each with its own IP header.

In some cases, as we will see in this chapter, the content of an IP packet must be further broken up to traverse the next link on the network. However, it's not really the IP packet that is fragmented. The original packet is discarded, and a string of IP packets is created that preserves the packet content and overall header fields, but changes specifics. When we say that "the packet is the data unit that flows end-to-end through the network," it is not the packet that is unchanged, but the content.

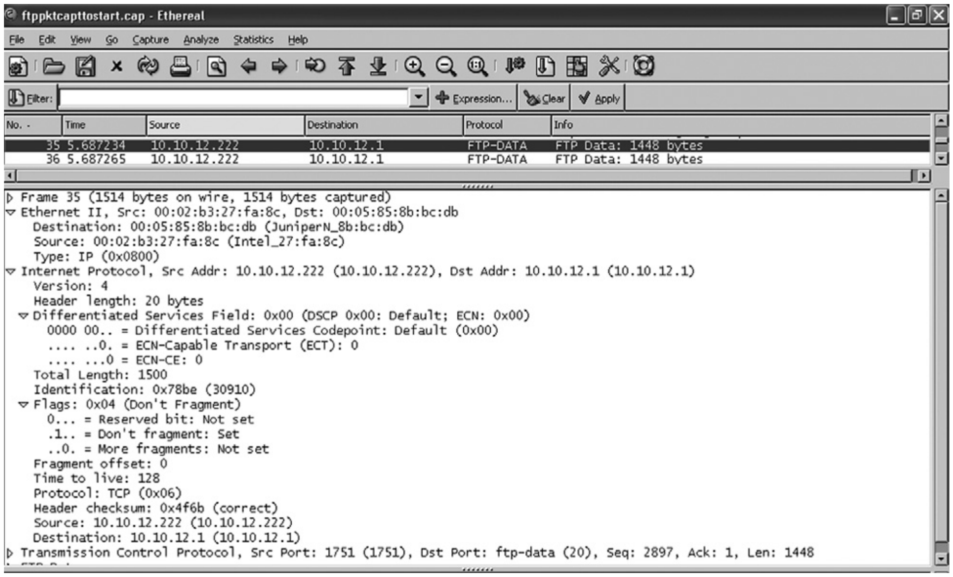
Naturally, if packet content is kept small enough, no fragmentation is necessary.

Figure 6.1 shows the parts of the Illustrated Network that we'll be using for our investigation of IP headers and fragmentation. The LAN clients and servers are highlighted, as are the local customer-edge routers.

Let's start with IPv4. We can just start a flow of IPv4 packets between a client and server and capture them. Then we can parse the packets until we find something of interest.

Let's take a good look at all the fields in an IPv4 packet header. We've already captured plenty of them. This example is from the FTP transfer from host (`wincli2`, with address `10.10.12.222`) to router (`CE6`, with address `10.10.12.1`) that we first saw in Chapter 2. Figure 6.2 shows a frame from the actual data transfer itself, frame 35, in fact.

The Ethernet frame is of type `0x0800` to show it carries an IPv4 packet. All of the lines from "Internet Protocol" to the line before "Transmission Control Protocol" interpret


ftppktcapttostart.cap - Ethereal
File Edit View Go Capture Analyze Statistics Help
Filter:
Expression...
Clear
Apply

```
No. -  Time        Source         Destination    Protocol   Info
   35 5.687234     10.10.12.222   10.10.12.1     FTP-DATA   FTP Data: 1448 bytes
   36 5.687265     10.10.12.222   10.10.12.1     FTP-DATA   FTP Data: 1448 bytes
```

```
▷ Frame 35 (1514 bytes on wire, 1514 bytes captured)
▽ Ethernet II, Src: 00:02:b3:27:fa:8c, Dst: 00:05:85:8b:bc:db
    Destination: 00:05:85:8b:bc:db (JuniperN_8b:bc:db)
    Source: 00:02:b3:27:fa:8c (Intel_27:fa:8c)
    Type: IP (0x0800)
▽ Internet Protocol, Src Addr: 10.10.12.222 (10.10.12.222), Dst Addr: 10.10.12.1 (10.10.12.1)
    Version: 4
    Header length: 20 bytes
  ▽ Differentiated Services Field: 0x00 (DSCP 0x00: Default; ECN: 0x00)
      0000 00.. = Differentiated Services Codepoint: Default (0x00)
      .... ..0. = ECN-Capable Transport (ECT): 0
      .... ...0 = ECN-CE: 0
    Total Length: 1500
    Identification: 0x78be (30910)
  ▽ Flags: 0x04 (Don't Fragment)
      0... = Reserved bit: Not set
      .1.. = Don't fragment: Set
      ..0. = More fragments: Not set
    Fragment offset: 0
    Time to live: 128
    Protocol: TCP (0x06)
    Header checksum: 0x4f6b (correct)
    Source: 10.10.12.222 (10.10.12.222)
    Destination: 10.10.12.1 (10.10.12.1)
▷ Transmission Control Protocol, Src Port: 1751 (1751), Dst Port: ftp-data (20), Seq: 2897, Ack: 1, Len: 1448
```

FIGURE 6.2

Capture of IPv4 header fields. The frame is broken out to show the content and meaning of every field in the IPv4 header. Note that the DF (Don't Fragment) bit is set on the packet.

fields in the IPv4 header. The source and destination addresses are listed first. Although we'll see that they are not the first fields in the header, they are definitely the fields that most frequently are of interest.

Ethereal interprets a field in the IPv4 header called the Type of Service (TOS) field according to something called Differentiated Services (DiffServ). DiffServ is only one way to interpret these fields. The figure shows that there are three things indicated by the 8 bits in the TOS field:

Differentiate Services Code Point (DSCP)—The default is zero, which means this packet does not require special handling by any router or host other than IP's normal best-effort service.

Explicit-Congestion-Notification Capable Transport (ECT)—This bit is set by devices when the transport is able to provide an indication of network congestion to network-attached devices. The value of zero shows that Ethernet is not an ECT, so packets cannot tell devices when the LAN is congested.

ECN Congestion Explicit (ECT-CE)—On transport that can report congestion, this bit is set when some predefined criteria for network congestion is met. This is often a percentage of output buffer fullness. On Ethernet this bit is always zero.

We'll say a little more about DSCP and quality of service (QOS) in a later chapter. However, the incomplete support for and variations in QOS implementations rule out QOS or DSCP as a topic for an entire chapter.

There are also four flag bits shown in the figure. The two most important are the bits that indicate this packet content is not to be fragmented (the DF bit is set to 1) and that there are no more frames carrying pieces of this packet's payload (the More Fragments bit is set to 0).

In the following, we talk about fragmentation in IPv4 in more detail, and then explore all of the fields in the IPv4 header in more detail.

THE IPv4 PACKET HEADER

The general structure of the IPv4 packet is shown in Figure 6.3. The minimum header (using no options, the most common situation) has a length of 20 bytes (always shown in a 4-bytes-per-line format), and a maximum length (very rarely seen) of 60 bytes. Some of the fields are fairly self-explanatory, such as the fields for the 4-byte (32-bit) IPv4 source and destination address, but others have specialized purposes.

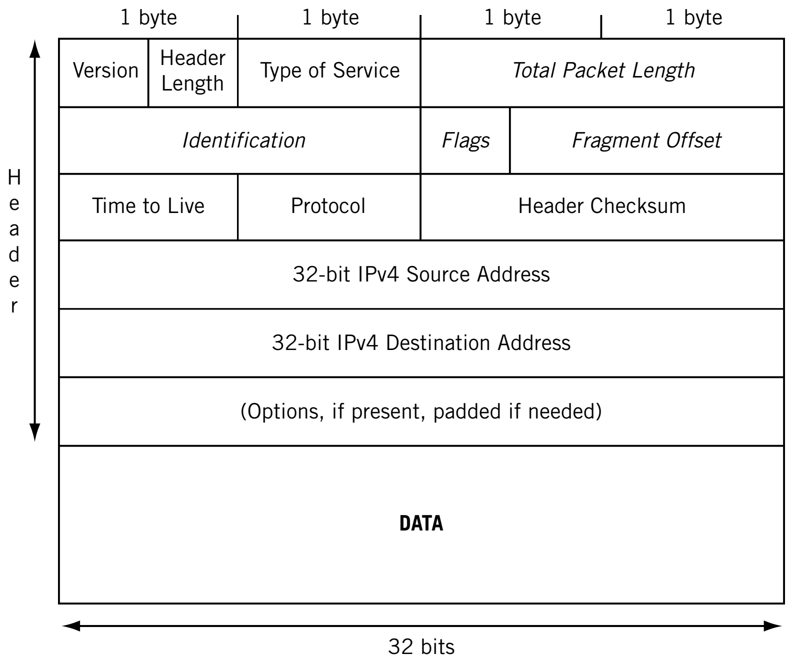


FIGURE 6.3

IPv4 Packet and Header

Version—Currently set to `0x04` for IPv4.

Header Length—Technically, this is the *Internet* header length (IHL). It is the length of the IP header in 4-byte (32-bit) units known as "words," and includes any option fields present and padding needed to align the header on a 32-bit boundary. In Figure 6.2, this is 20 bytes, which is most common.

Type of Service (TOS)—Contains parameters that affect how the packet is handled by routers and other equipment. Never widely used, it was redefined as Differentiated Services (DiffServ or DS) code points and is still hampered because of a lack of widespread implementation, especially from one routing domain to another. The meaning of these bits, which are all set to 0 in Figure 6.2, was detailed earlier in this chapter.

The next four fields, shown in italics in Figure 6.3, figure directly in the fragmentation process. Fragmentation, introduced in Chapter 4, occurs when a packet is forwarded onto a data link and the packet content will not fit inside a single frame. In these cases, the packet content must be fragmented and spread across several frames, then reassembled at the destination host. Fragmentation will be discussed in detail in the next section of this chapter.

Total Packet Length—This is the length of the whole packet in bytes. The maximum value for this two-byte field is 65,535 bytes. This length is approached by no common TCP/IP implementation or network MTU size. The packet in Figure 6.2 is 1500 bytes long, the most common length due to the prevalence of Ethernet LANs.

Identification—A 16-bit number set for each packet to help the destination host reassemble like-numbered fragments. Even intact, single packets could be fragmented by routers (sometimes repeatedly) on their way to a destination, so this field must be filled in. This field is set to `0x78be` (`30910`) in Figure 6.2.

Flags—Only the first 3 bits of this field are defined. Bit 1 is reserved and must be set to 0. Bit 2 (DF) is set to 0 if fragmentation is allowed or 1 if fragmentation is *not* allowed. Bit 3 (MF) is set to 0 if the packet is the last fragment, or 1 if there are more fragments to come. Note that the MF field does not imply any sequencing of the arriving fragments, nor does it guarantee that the set is complete. Other fields are examined to determine sequencing and completeness. The packet in Figure 6.2 will generate an error when it encounters a device that wants to fragment the packet content.

Fragment Offset—When a packet is fragmented, the fragments must fall on an 8-byte boundary. That is, an 800-byte packet can be fragmented into two packets of 400 bytes each, but not as eight packets of 100 bytes each, since 100 is not evenly divisible by 8. This field contains the number of 8-byte units, or *blocks*, in the packet fragment. The offset is 0 in Figure 6.2.

The rest of the IP header fields do not deal with fragmentation.

Time to Live (TTL)—This 8-bit field value is supposed to be the number of *seconds*, up to 255 maximum, that a packet can take to reach the destination. Each router is supposed to decrement this field by a preconfigured amount which must be greater than 0. If a packet arriving at a router has this field set to 0, it is discarded and never routed. Unfortunately, there is no standard way to track time across a group of routers, so most TCP/IP networks interpret this field as a simple *hop count* between routers and simply decrement this field by 1. The TTL in Figure 6.2 is 128, a fairly typical value.

Protocol—This 8-bit field contains the number of the transport-layer protocol that is to receive and process the data content of the packet. The protocol number for TCP is 6 and UDP is 17, but almost 200 have been defined. The packet in Figure 6.2 carries TCP.

Header Checksum—An error-detection field for the IP header only, not the packet data fields. If the computed checksum does not match at the receiver, the header is damaged and not routed. Figure 6.2 not only shows the header checksum of `0x4f6b`, but Ethereal tells us that it is correct.

Source and Destination Addresses—The 32-bit IPv4 addresses of the source and destination hosts. The packet in Figure 6.2 is sent from 10.10.12.222 to 10.10.12.1.

Options—The IPv4 options are seldom used today for data transfer and will not be described further, nor do they appear in Figure 6.2.

Padding—When options *are* used, the padding field makes sure the header ends on a 32-bit boundary. That is, the header must be an integer number of 4-byte "words." The header in Figure 6.2 is not padded, and few are since options use is unusual.

FRAGMENTATION AND IPv4

Let's look at IPv4 fragmentation on the Illustrated Network. We can determine how the MTU size and fragmentation affect IPv4 data transfer rates.

It's not all that important (and not all that interesting) to show the fragmentation process with a capture. Moreover, it is difficult to convey a sense of what's going on with a series of snapshots, even when Ethereal parses the fragmentation fields. Appreciating the effects of a small MTU size on data transfers is more important.

Let's use the `bsdclient` on LAN1 and `bsdserver` on LAN2 to show what fragmentation does to data throughput. We'll use FTP to transfer a small file (about 30,000 bytes) called `test.stuff` from the server to the client. Why so small a file? Just to show that if fragmentation plays a role in small transfers, the effects will be magnified with larger files. First, we'll use the default MTU sizes.

```
bsdclient# ftp 10.10.12.77
Connected to 10.10.12.77.
220 bsdserver FTP server (Version 6.00LS) ready.
Name (10.10.12.77:admin): admin
331 Password required for admin.
Password:
230 User admin logged in.
Remote system type is UNIX.
Using binary mode to transfer files.
ftp> get test.stuff
local: test.stuff remote: test.stuff
150 Opening BINARY mode data connection for 'test.stuff' (29752 bytes).
100%
|***************************************************************************
***********************| 29752 00:00 ETA
226 Transfer complete.
29752 bytes received in 0.01 seconds (4.55 MB/s)
```

This is about 4.5 MBps (or about 36 Mbps) and transfer time of about 1/100th of a second. Not too bad. (Keep in mind that 1/100th of a second is about the smallest interval that can be reported without special hardware.) This is good throughput, but remember there are only two routers involved, connected by a SONET link at 155 Mbps and the LAN runs at 100 Mbps. There is also no other traffic on the network, so the transfer rate is totally dependent on the ability of the host to fill the pipe from server to client.

Now let's change to Maximum Transmission Unit size at the server connected to LAN2 (the server LAN) from the default of 1500 to 256 bytes. How much of a difference will this make?

```
ftp> get test.stuff
local: test.stuff remote: test.stuff
150 Opening BINARY mode data connection for 'test.stuff' (29752 bytes).
100%
|***************************************************************************
***********************| 29752 00:00 ETA
226 Transfer complete.
29752 bytes received in 1.30 seconds (22.29 KB/s)
ftp>
```

The transfer time is up to 1.3 seconds, about 130 times longer than before! And the transfer rate fell from about 36 Mbps to about 184 *KILOBITS* per second, three orders of magnitude less than before. This is the "performance penalty" of fragmentation. (It should be pointed out that these numbers are not precise, and there are many other reasons that file transfers speed up or slow down. However, the point is entirely valid.)

We can view a lot of packet statistics, including fragment statistics, using the `netstat` utility. With `netstat`, we can monitor an interface in real time, display the host routing table, observe running network processes, and so on. We'll do more with `netstat` later. For now, we'll just see how many fragments our 30,000-byte file transfer has generated.

To do this, we'll look at the IP statistics on the client before and after the file transfer has been run with the small MTU size. We'll set the counters to zero first.

```
bsdclient# netstat -sp ip
ip:
      0 total packets received
      0 bad header checksums
      0 with size smaller than minimum
      0 with data size < data length
      0 with ip length > max ip packet size
      0 with header length < data size
      0 with data length < header length
      0 with bad options
      0 with incorrect version number
      0 fragments received
      0 fragments dropped (dup or out of space)
      0 fragments dropped after timeout
      0 packets reassembled ok
      [many more lines deleted for clarity...]
```

Now we'll reset the counters, run the transfer again, and check the IP statistics.

```
bsdclient# netstat -sp ip
ip:
      57 total packets received
      0 bad header checksums
      0 with size smaller than minimum
      0 with data size < data length
      0 with ip length > max ip packet size
      0 with header length < data size
      0 with data length < header length
      0 with bad options
      0 with incorrect version number
      171 fragments received
      0 fragments dropped (dup or out of space)
      0 fragments dropped after timeout
      57 packets reassembled ok
      [many more lines deleted for clarity...]
```

The file was transferred as 171 fragments that were reassembled into 57 packets. Let's take a closer look at fragmentation of the MTU size in IPv4.

Fragmentation and MTU

If an IP packet is too large to fit into the frame for the outgoing link, the packet content must be fragmented to fit into multiple "transmission units." The Maximum Transmission Unit (MTU) size is a key concept in all TCP/IP networks, often complicated by the fact that different types of links (LAN or WAN) have very different MTU sizes. Many of these are shown in Table 6.1. The link protocols shown in italics have "tunable" (configurable) MTU sizes instead of defined defaults, but almost all interfaces allow you to lower the MTU size. The figures shown are the usual maximums. The 9000-byte packet size is not standard in Gigabit Ethernet, but common.

Hosts reassemble any arriving fragmented packets to avoid routers pasting together and then tearing apart packets repeatedly as they are forwarded from link to link. Fragments themselves can even be fragmented further as a packet makes its way from, for example, Gigabit Ethernet to frame relay to Ethernet.

Fragmentation is something that all network administrators used to try to avoid. As a famous paper circulated in 1987 asserted bluntly, "Fragmentation [is] considered harmful." As recently as 2004, an Internet draft (*http://ietfreport.isoc.org/all-ids/draft-mathis-frag-harmful-00.txt*) took this one step further with the title, "Fragmentation Considered Very Harmful." The paper asserts that most of the harm occurs when a fragment of packet content, especially the first, is lost on the network. And a number of older network attacks involved sending long sequences of fragments to targets, never finishing the sequence, until the host or router ran out of buffer space and crashed. Also,

Table 6.1 Typical MTU Sizes*

Link Protocol	Typical MTU Limit	Maximum IP Packet
Ethernet	1518	1500
IEEE 802.3	1518	1492
Gigabit Ethernet	9018	9000
IEEE 802.4	8191	8166
IEEE 802.5 (Token Ring)	4508	4464
FDDI	4500	4352
SMDS/*ATM*	9196	9180
Frame relay	4096	4091
SDLC	2048	2046

** Frame overhead accounts for the differences between the theoretic limit and maximum IP packet size.*

because of the widespread use of *tunnels* (see Chapter 26), there are link layers that really need an MTU larger than 1500 to support encapsulation, and you can't fragment MTUs inside a tunnel.

There are several reasons for the quest to determine the smallest of the MTU sizes on the links between source and destination. This "minimum" MTU size can be used between a source and destination in order to avoid fragmentation. The main reasons today follow:

- Fragmentation is processor intensive. Early routers were hard pressed to both route and fragment. Even today, high link speeds force routers to concentrate on routing and minimize "housekeeping" tasks.
- Many hosts struggle to reassemble fragments. Fragmentation puts the reassembly burden on the receiving host, which can be a cell phone, watch, or something else. This requires processing power and delays the processing of the packet.
- Fragmentation fields are favorite targets for hacking. TCP/IP implementation behaviors are not spelled out in detail for many situations where the fragmentation fields are set to inconsistent or contradictory values. Many a host and router have been hung by exploiting this variable behavior.
- Fragments can be lost, out-of-sequence, or errored. The more pieces there are, the more things that can go wrong. The worse occurs when the first fragment is lost on the network.
- Early IP implementations avoided fragmentation by setting the default IP packet size very low, to only 576 bytes. All link protocols then in common use could handle this small packet size, and many IP implementations to this day still use this default packet size. Naturally, the smaller the MTU size, the greater the number of packets sent for a given message, and the greater the chances something can go wrong.

Fragmentation behavior changes in IPv6. In IPv6, routers do not perform fragmentation.

Fragmentation and Reassembly

The point has already been made that fragmentation is a processor-intensive operation. Naturally, if all hosts sending packets were aware of the *minimum* MTU size on a path from source to destination before sending an IP packet, the problem would be solved. There are ways to determine the path MTU size.

Path MTU Determination

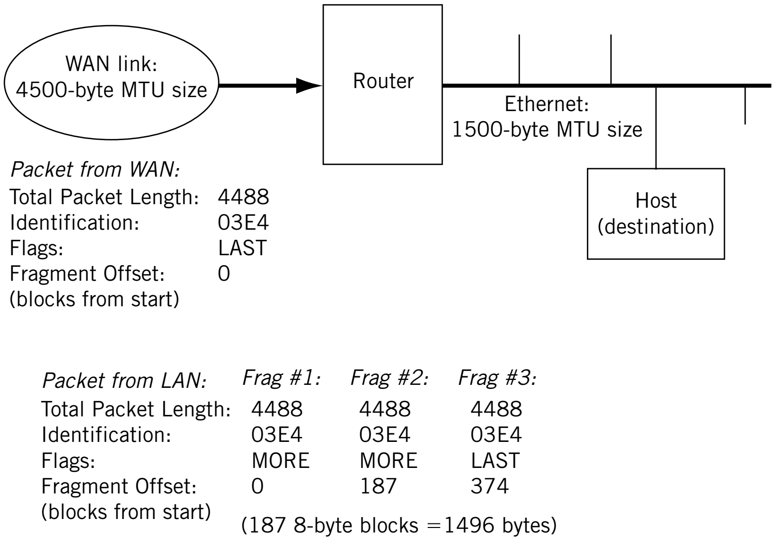
The commonly used method to determine this path MTU is slow, but it works. The method involves "testing" the path to the destination before sending "live" packets to a destination system where the path MTU is not known. The source system sends out an echo packet. (The echo service just bounces back the content of the packet to the sender.) The echo packet is usually the MTU size of the source system's own TCP/IP network, which could be 1500 bytes for Ethernet, 4500 for Token Ring, and so on. This packet has the DF bit set in the Flags field in the IPv4 header. If the echo packet comes back successfully, then the MTU size is fine and can be used for "live" data.

However, if the current path through the routers includes a smaller MTU size on a link or network that the packet must traverse as the packet makes its way to the destination, the router attached to this smaller MTU size network *must* discard the packet, since the DF bit is set. The router sends an ICMP error message back to the source indicating the error condition, which is that the packet was discarded because the DF bit was set. The source can then adjust the packet size downward and try again. This process can be repeated several times, trying to find the optimal path MTU.

This path MTU determination method works, but it is awkward and slow. The live data basically wait until the path MTU size is determined for a destination. And because each packet is independently routed, if there are multiple paths through the router network (and there usually are, this being the whole point of using routers), the MTU size may change with every possible path that an IP packet can take from the source to the destination. However, this method is better than nothing.

A FRAGMENTATION EXAMPLE

Figure 6.4 shows a router on a TCP/IP network. The arriving IP packet is coming from a WAN link with a configured MTU size of 4500 bytes. The destination system is attached to the router by means of an Ethernet LAN, which has an MTU size of 1500 bytes.



Packet from WAN:	
Total Packet Length:	4488
Identification:	03E4
Flags:	LAST
Fragment Offset: (blocks from start)	0

Packet from LAN:	*Frag #1:*	*Frag #2:*	*Frag #3:*
Total Packet Length:	4488	4488	4488
Identification:	03E4	03E4	03E4
Flags:	MORE	MORE	LAST
Fragment Offset: (blocks from start)	0	187	374

(187 8-byte blocks = 1496 bytes)

FIGURE 6.4

An IPv4 fragmentation example, showing the various header field values for each of the three fragments loaded into the frames.

Obviously, the 4500-byte packet must be fragmented across three Ethernet frames to reach the destination host.

Figure 6.4 shows the portions of the IP packet data and the values of the fragmentation fields for each fragment. The figure also shows how the destination system interprets the fragmentation fields to reassemble the entire packet at the destination.

We've already looked at the problems with fragmentations from the router and network perspective. From the perspective of the receiving host, there are two main reasons that fragmentation should be avoided. One is the need to wait for undelivered fragments, and the other is the lack of knowledge on the part of a destination of the reassembled datagram size. Let's look at the destination host reassembly process to explore the "performance penalty" that fragmentation involves.

A fragmented packet is always reassembled at the destination host and never by routers. (Why put together packets that might require fragmentation all over again?) However, because all packets are independently routed, the pieces of a packet can arrive out of sequence. When the first fragment arrives, local buffer memory is allocated for the reassembly process. The Fragment Offset of the arriving packet indicates exactly where in the sequence the newly arrived fragment should be placed.

At a busy destination, such as a Web server, many different packets from several sources can arrive in fragments. All of these pieces can be subjected to the reassembly process at the same time. The destination host IP layer software will associate packets having matching Identification, Source, Destination, and Protocol fields as belonging to the same packet.

However, the Total Length field in a packet fragment's header only indicates the length of that particular fragment, not the entire packet before fragmentation. It is only when the destination system receives the *last* fragment that the total length of the original packet can be determined.

If a packet is partially reassembled and the final piece to complete the set has not arrived, IP includes a tunable reassembly time-out parameter. If the reassembly timer expires, the remaining packet fragments are discarded. If the final piece of the packet arrives after the time-out, this packet fragment must be discarded as well.

This description of the reassembly process shows the twin problems of memory allocation woes from packet size uncertainties and delays due to the reassembly time-out.

Arriving IP packets have no way to inform the destination system that "I am the first of 10 fragments." If so, it would be easy for the destination system to allocate memory for reassembly that was the best-fit for remaining contiguous buffer space. But all packet fragments can indicate is "I am the first of many," "I am the second of many," and so on, until one finally says, "I am the last of many." This uncertainty of reassembled size makes many TCP/IP implementations allocate as large a block of memory as available for reassembly. Obviously, a fragmented packet may have been quite large to begin with, because it was fragmented in the first place. But the net result is that local buffers become quite fragmented. And if smaller blocks of memory are allocated, the resulting non-contiguous pieces must be moved to an adequate sized memory buffer before the transport layer can process the reassembled datagram.

The reassembly time-out value must have a value low enough to make the recovery process delay of the transport layer reasonable. The transport layer contains session (connection) information that will detect a missing packet in a sequence of segments (the contents of the packets), and TCP always requests missing information to be resent. Too long a value for the reassembly timer makes this retransmission process very inefficient. Too short a value leads to needlessly discarded packets. In most TCP/IP implementations, the reassembly timer is set by the software vendor and cannot be changed. This is yet another reason to avoid fragmentation.

Reassembly "deadlock" used to be a problem as well. When memory was a scarce commodity in hosts, all available local buffer memory could end up holding partially assembled fragments. An arriving fragment could not be accepted even if it completed a set and the system eventually hung. However, in these days of cheap and plentiful memory, this rarely happens.

Limitations of IPv4

The limitations of IPv4 are often cast solely in terms of address space. As important as that is, it is only part of the story. Address space is not the only IPv4 limitation. Some others follow:

- The fragmentation fields are present in every IPv4 packet.
- Fragmentation is always done with a performance penalty and is best avoided. Yet the fields involved—all 6 bytes worth and more than 25% of the basic 20-byte IPv4 header—must be present in each and every packet.
- IPv4 Options were seldom used and limited in scope.
- The IPv4 Type of Service field was never used as intended.
- The IPv4 Time To Live field was also never used as intended.
- The 8-bit IPv4 Type field limited IPv4 packet content to 256 possibilities.

All of these factors contributed to the structure of the IPv6 packet header.

The IPv6 Header Structure

Let's go back to our Windows devices and capture some IPv6 packets. Then we can examine those headers and compare them to IPv4 headers.

```
bsdserver# ping6 fc00:fe67:d4:b:205:85ff:fe8b:bcdb
PING6(56=40+8+8 bytes) fc00:fe67:d4:b:20e:cff:fe3b:8732 -->
fc00:fe67:d4:b:205:85ff:fe8b:bcdb
16 bytes from fc00:fe67:d4:b:205:85ff:fe8b:bcdb, icmp_seq=0 hlim=64
 time=16.027 ms
16 bytes from fc00:fe67:d4:b:205:85ff:fe8b:bcdb, icmp_seq=1 hlim=64
 time=0.538 ms
16 bytes from fc00:fe67:d4:b:205:85ff:fe8b:bcdb, icmp_seq=2 hlim=64
 time=0.655 ms
```

```
16 bytes from fc00:fe67:d4:b:205:85ff:fe8b:bcdb, icmp_seq=3 hlim=64
 time=0.622 ms
^C
--- fc00:fe67:d4:b:205:85ff:fe8b:bcdb ping6 statistics ---
4 packets transmitted, 4 packets received, 0% packet loss
round-trip min/avg/max/std-dev = 0.538/4.461/16.027/6.678 ms
```

Here is the first packet we captured:

```
bsdserver# tethereal -V
Capturing on em0
Frame 1 (70 bytes on wire, 70 bytes captured)
   Arrival Time: May 23, 2008 18:39:58.914560000
   Time delta from previous packet: 0.000000000 seconds
   Time since reference or first frame: 0.000000000 seconds
   Frame Number: 1
   Packet Length: 70 bytes
   Capture Length: 70 bytes
Ethernet II, Src: 00:0e:0c:3b:87:32, Dst: 00:05:85:8b:bc:db
   Destination: 00:05:85:8b:bc:db (JuniperN_8b:bc:db)
   Source: 00:0e:0c:3b:87:32 (Intel_3b:87:32)
   Type: IPv6 (0x86dd)
Internet Protocol Version 6
   Version: 6
   Traffic class: 0x00
   Flowlabel: 0x00000
   Payload length: 16
   Next header: ICMPv6 (0x3a)
   Hop limit: 64
   Source address: fc00:fe67:d4:b:20e:cff:fe3b:8732 (fc00:fe67:d4:b:20e:
cff:fe3b:8732)
   Destination address: fc00:fe67:d4:b:205:85ff:fe8b:bcdb (fc00:fe67:d4:
b:205:85ff:fe8b:bcdb)
Internet Control Message Protocol v6
   Type: 128 (Echo request)
   Code: 0
   Checksum: 0x7366 (correct)
   ID: 0x0565
   Sequence: 0x0000
   Data (8 bytes)

0000 6e b9 73 44 43 f4 0d 00             n.sDC...
```

In contrast to the IPv4 header, there are only eight lines (and eight fields) in the IPv6 header. Since the packet is simple enough, let's look at the header fields in detail as we examine the meaning and values in this IPv6 packet.

The IPv6 header is shown in Figure 6.5. Besides the new expanded, 16-byte IP source and destination addresses, there are only six other fields in the entire IPv6 header. This simpler header structure makes for faster packet processing in most cases.

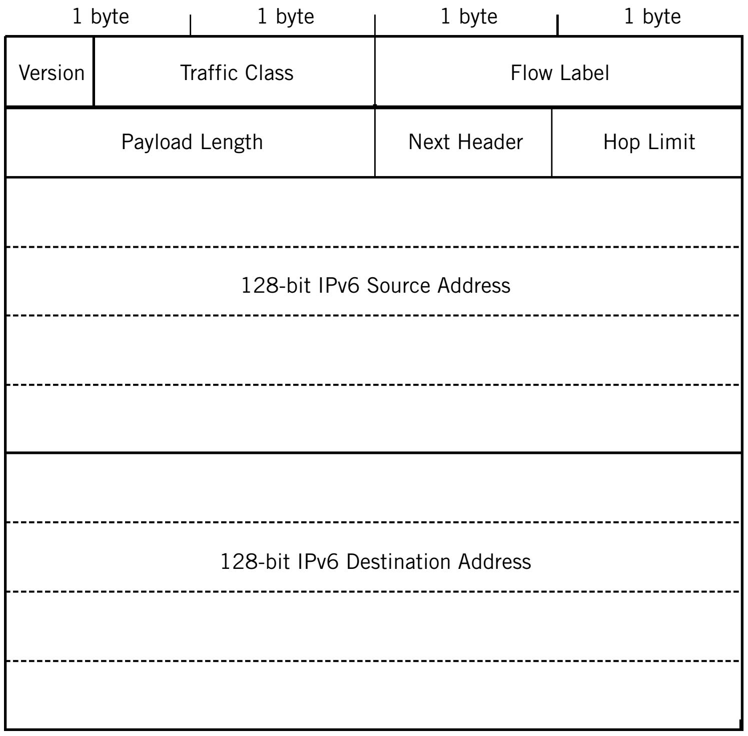


FIGURE 6.5

The IPv6 header fields. Note the reduction in field number of how the address fields occupy most of the header.

IPv6 packets have their own frame Ethertype value, `0x86dd`, making it easy for receivers that must handle both IPv4 and IPv6 on the same interface to distinguish the frame content.

Version—A 4-bit field for the IP version number (`0x06`).

Traffic Class—A 12-bit field that identifies the major class of the packet content (e.g., voice or video packets). Our capture shows this field as the default at 0, meaning that it is ordinary bulk data (as FTP should carry) and requires no special handling at devices.

Flow Label—A 16-bit field used to label packets belonging to the same flow (those with the same values in several TCP/IP header parameters). The flow label here is 0, but this is common.

Payload Length—A 16-bit field giving the length of the packet in bytes, excluding the IPv6 header. The payload of this packet, an ICMP message, is 16 bytes long.

Next Header—An 8-bit field giving the type of header immediately following the IPv6 header (this served the same function as the Protocol field in IPv4). This packet carries an ICMPv3 message, so the value is `0x3a`.

Hop Limit—An 8-bit field set by the source host and decremented by 1 at each router. Packets are discarded if the hop limit is decremented to zero (this replaces the IPv4 Time To Live field). The hop limit here is `64`, half of the FTP value in our IPv4 example. Generally, implementers choose the default to use, but values such as 64 or 128 are common.

IPv4 AND IPv6 HEADERS COMPARED

Figure 6.6 shows the fields in the IPv4 packet header compared to the fields in the IPv6 header.

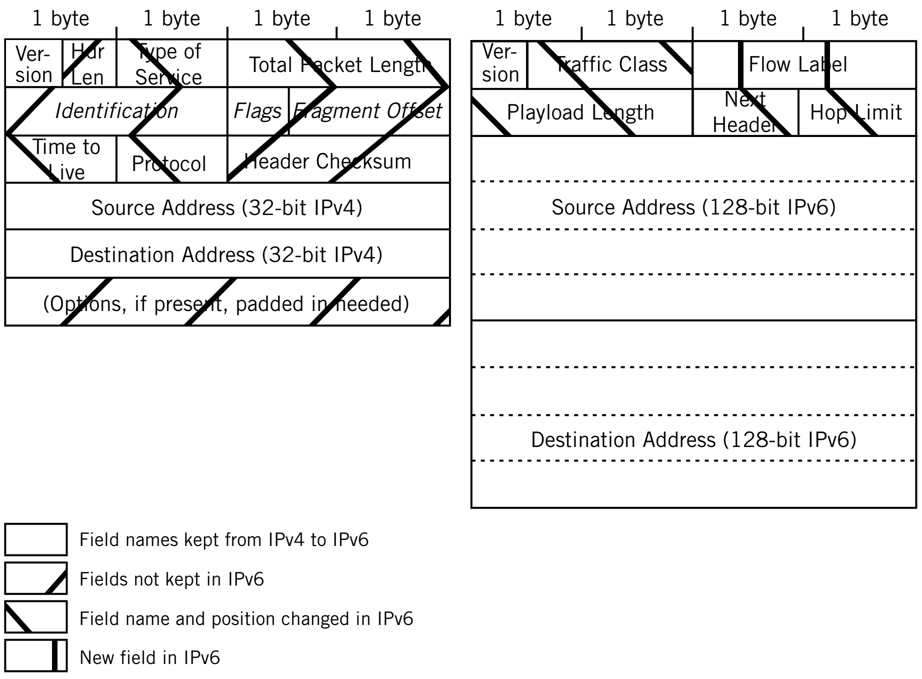


FIGURE 6.6

IPv4 and IPv6 headers compared, showing how the old fields and new fields relate to each other.

IPv6 Header Changes

In summary, the following are some of the most important changes to the IP header in IPv6.

- Longer addresses (32 bits to 128 bits). No fragmentation fields.
- No header checksum field. No header length field (there is a fixed length header).
- Payload length given in bytes, not "blocks" (32-bit units). Time to Live (TTL) field becomes Hop Limit.
- Protocol field becomes Next Header (determines content format). 64-bit alignment of the packet, not 32-bit alignment. A Flow Label field has been added.
- No Type of Service bits (which were seldom respected anyway). Many of the IPv4 fields vanish completely, especially the fields used for packet fragmentation. IPv6 addresses fragmentation performance penalties and problems by forbidding it altogether in routers. Source hosts can still fragment, however, if the source host wants to send packets larger than the Path MTU size to a destination. In IPv6, as in IPv4, fragmentation issues can be avoided altogether by making all packets 1280 bytes long—the minimum established by RFC 2460—but this results in many "extra" packets.
- The IPv4 header Checksum field is absent because destination host error checking is the preferred method of error detection in today's more reliable networks, and almost all transmission frames provide better error detection than the IP layer. There is no header length field because all IPv6 headers are the same length. The Payload Length field excludes the IPv6 header fields and is measured in bytes, rather than the awkward 4-byte units of IPv4.
- The TTL field, never interpreted as time anyway, is gone as well. In its place is the Hop Limit field, a simple indication of the number of routers that a packet can pass through before it should reach the destination host. The Protocol field of IPv4 has become the Next Header field in IPv6. The term "next header" is more accurate because the information inside the IPv6 packet is not necessarily a higher layer protocol (e.g., TCP segment) in IPv6. There are many other possibilities.
- The entire packet must be an integer number of 64-bit (8-byte) units. The 32-bit unit for IPv4 was established when many high-performance computers were 32-bit machines, meaning memory access and internal bus operations moved 32-bit units (called a "word") around. Today high-performance computers often support 64-bit words. It only made sense to align the new IPv6 header for ease and speed of processing on the newer architecture computers.
- Finally, in place of the ToS field in IPv4, the IPv6 header defines a Flow Label field. Flows are used by routers to pick out IPv6 packets containing delay-sensitive data such as voice, video, and multimedia. The Type of Service field was usually ignored by routers in IPv4, and other uses were not standardized.

- The IPv6 specification includes a concept known as Extension Headers. Extension Headers essentially take the place of the Options in the IPv4 packet header. IPv6 Extension Headers are only present when necessary and are designed to be extensible (new functions may be defined in the future), but the term "extensible Extension Headers" is awkward.
- The current Extension Headers include a Hop-by-Hop Option Header, examined by every router handling the IPv6 packet and an Authentication Header for enhanced security on TCP/IP networks (these are used in IPv4 as part of IPSec). There is also a Fragmentation header for the use of the source host when there is no way to prevent the source from sending packets larger than the path MTU size (IPv6 routers cannot fragment, but hosts can). Also, there used to be a Routing Header specifying the IP addresses of the routers on the path from source to destination (similar to "source routing" in token ring LANs), but this is deprecated by RFC 5095. There are several others, but these show the kinds of capabilities included in the IPv6 Extension Headers.

IPv6 AND FRAGMENTATION

What would happen if we put IPv6 into a situation where it has to fragment packet content to make it fit into a frame? Let's use the Illustrated Network to find out. Two useful ping parameters are the size of the packet to bounce off a remote device and the count of packets sent. We'll capture the packets sent when `bsdserver` sends a 2000-byte packet (too large for an Ethernet frame) to the router.

```
bsdserver# ping6 -s 2000 -c 1 fc00:fe67:d4:b:205:85ff:fe8b:bcdb
PING6(2048=40+8+2000 bytes) fc00:fe67:d4:b:20e:cff:fe3b:8732 -->
fc00:fe67:d4:b:205:85ff:fe8b:bcdb
2008 bytes from fc00:fe67:d4:b:205:85ff:fe8b:bcdb, icmp_seq=0 hlim=64
 time=2.035 ms

--- fc00:fe67:d4:b:205:85ff:fe8b:bcdb ping6 statistics ---
1 packets transmitted, 1 packets received, 0% packet loss
round-trip min/avg/max/std-dev = 2.035/2.035/2.035/0.000 ms
bsdserver#
```

This makes 2008 bytes with the IPv6 header. Here's what we have (the data fields, which contain test strings, have been omitted):

```
bsdserver# tethereal -V
Capturing on em0
Frame 1 (1510 bytes on wire, 1510 bytes captured)
   Arrival Time: May 25, 2008 08:39:21.231993000
   Time delta from previous packet: 0.000000000 seconds
   Time since reference or first frame: 0.000000000 seconds
   Frame Number: 1
   Packet Length: 1510 bytes
   Capture Length: 1510 bytes
Ethernet II, Src: 00:0e:0c:3b:87:32, Dst: 00:05:85:8b:bc:db
   Destination: 00:05:85:8b:bc:db (JuniperN_8b:bc:db)
   Source: 00:0e:0c:3b:87:32 (Intel_3b:87:32)
   Type: IPv6 (0x86dd)
Internet Protocol Version 6
   Version: 6
   Traffic class: 0x00
   Flowlabel: 0x00000
   Payload length: 1456
   Next header: IPv6 fragment (0x2c)
   Hop limit: 64
   Source address: fc00:fe67:d4:b:20e:cff:fe3b:8732 (fc00:fe67:d4:b:20e:
    cff:fe3b:8732)
   Destination address: fc00:fe67:d4:b:205:85ff:fe8b:bcdb (fc00:fe67:d4:
    b:205:85ff:fe8b:bcdb)
Fragmentation Header
   Next header: ICMPv6 (0x3a)
   Offset: 0
   More fragments: Yes
   Identification: 0x000000e5
Internet Control Message Protocol v6
   Type: 128 (Echo request)
   Code: 0
   Checksum: 0x74df
   ID: 0x0e60
   Sequence: 0x0000
   Data (1440 bytes) (OMITTED)

Frame 2 (622 bytes on wire, 622 bytes captured)
   Arrival Time: May 25, 2008 08:39:21.232007000
   Time delta from previous packet: 0.000014000 seconds
   Time since reference or first frame: 0.000014000 seconds
   Frame Number: 2
   Packet Length: 622 bytes
   Capture Length: 622 bytes
Ethernet II, Src: 00:0e:0c:3b:87:32, Dst: 00:05:85:8b:bc:db
   Destination: 00:05:85:8b:bc:db (JuniperN_8b:bc:db)
   Source: 00:0e:0c:3b:87:32 (Intel_3b:87:32)
   Type: IPv6 (0x86dd)
Internet Protocol Version 6
   Version: 6
   Traffic class: 0x00
   Flowlabel: 0x00000
   Payload length: 568
   Next header: IPv6 fragment (0x2c)
   Hop limit: 64
   Source address: fc00:fe67:d4:b:20e:cff:fe3b:8732 (fc00:fe67:d4:
     b:20e:cff:fe3b:8732)
   Destination address: fc00:fe67:d4:b:205:85ff:fe8b:bcdb (fc00:fe67:
     d4:b:205:85ff:fe8b:bcdb)
Fragmentation Header
   Next header: ICMPv6 (0x3a)
   Offset: 1448
   More fragments: No
   Identification: 0x000000e5
Data (560 bytes) (OMITTED)

(Frames 3 and 4, the echoed frames sent back in response, are mirror
images of Frames 1 and 2 and have been omitted for brevity.)

bsdserver#
```

Because the host cannot pack 2000 bytes into an Ethernet frame, the IPv6 host does the fragmenting *before* it sends the bits onto the LAN. There are no fragmentation fields in the IPv6 header, however, so IPv6 includes a second header (next header) that carries the information needed for the destination to reassemble the fragments (in this case, two of them). The important fields are highlighted in bold in the preceding code.

The first frame (the capture says "packet") is 1510 bytes long, including 1456 bytes of payload (given in the Payload Length field). The Next Header value of `0x2c` indicates that the next header is an IPv6 fragment header. The Fragmentation Header fields are listed in full:

- *Next Header* (`0x3a`)—The header following the Fragmentation Header is an ICMPv6 message header.
- *Offset* (`0`)—This is the first fragment of a series.
- *More Fragments* (Yes)—There are more fragments to come.
- *Identification* (`0x000000e5`)—Only reassemble fragments that share this identifier value.

The data field in the ICMPv6 message is 1440 bytes long. The rest of the 1510 bytes are from the various headers pasted onto these bytes.

Frame 2 holds the rest of the 2000 bytes in the ping. This frame is 622 bytes long and carries 568 bytes of payload. The Next Header is still an IPv6 fragment (`0x2c`). The Fragmentation Header fields follow:

- *Next header* (`0x3a`)—The header following the Fragmentation Header is an ICMPv6 message header.
- *Offset* (`1448`)—These bytes start 1448 bytes after the content of the first frame. (The "extra" 8 bytes are for the ICMPv6 header.)
- *More Fragments* (No)—The contents of this packet complete the series.
- *Identification* (`0x000000e5`)—This fragment goes with the previous one with this identifier value.

The data field in the ICMPv6 message is 560 bytes long. Along with the 1440 bytes in the first fragment, these add up to the 2000 bytes sent.

QUESTIONS FOR READERS

Figure 6.7 shows some of the concepts discussed in this chapter and can be used to help you answer the following questions.

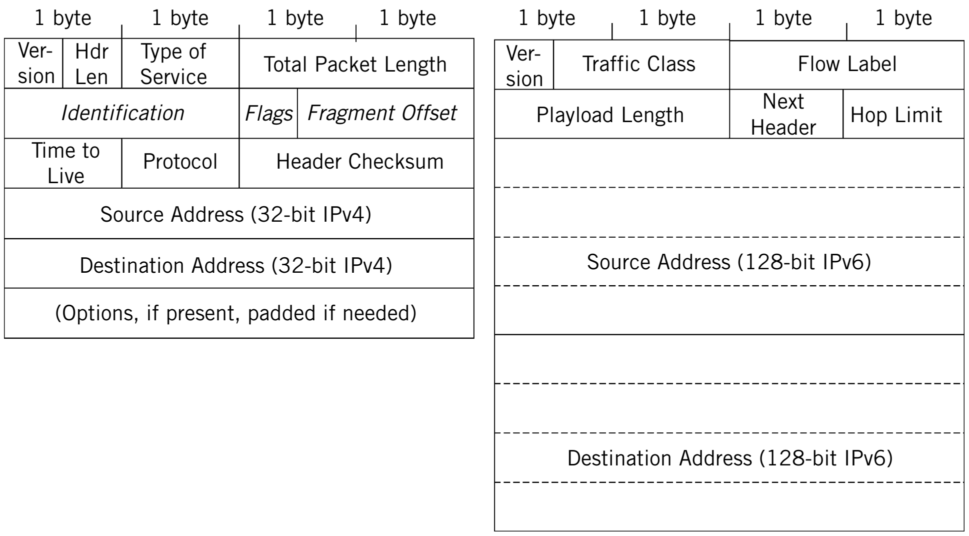


FIGURE 6.7

The IPv4 and IPv6 packet header fields. IPv6 can employ most IPv4 options as “next header” fields following the basic header.

1. Why are diagnostics like ping messages routinely given high hop-count values such as 64 or 128?
2. Without any IPv4 options in use, what value should be seen in the Header Length field most of the time?
3. How does an IP receiver detect missing fragments?
4. Is there any way for an IP receiver to determine how many fragments are supposed to arrive?
5. Since almost all the IPv4 header fields are options in IPv6, is it correct to say that the IPv6 header is “simplified”?