CHAPTER

User Datagram Protocol 10

What You Will Learn

In this chapter, you will learn about UDP, one of the major transport layer protocols in the TCP/IP stack. We'll talk about datagrams and the structure of the UDP header.

You will learn about ports and sockets and how they are used at the transport layer.

The User Datagram Protocol (UDP) is one the major transport layer protocols that rides on top of IPv4 or IPv6. Most explorations of the TCP/IP transport layer treat the other major protocol, the connection-oriented Transmission Control Protocol (TCP) first and present connectionless UDP later. But the complexities of TCP, and the reasons for these often sophisticated procedures, are better understood after appreciating the basic connectionless service provided by UDP. In addition, certain concepts that are shared by both UDP and TCP, such as ports, can be introduced in UDP and so reduce the number of new ideas that must be covered during TCP discussions to a more manageable level.

The UDP acronym shows the effects of early Internet efforts to distinguish connectionless packet delivery ("It's a datagram, not a packet!") from more conventional connection-oriented schemes in use at the time. The data unit of UDP is not a packet anyway, but a *datagram*, the content of a connectionless packet (many authors call IP packets datagrams as well, but we do not in this book). UDP datagrams have their own headers, naturally, and the UDP header is about as simple as a header can get. That's only to be expected, because UDP operation is also very simple, making UDP ideal for a first look at end-to-end functions on a network.

In recent years, UDP's popularity as a transport layer protocol for applications has been growing. The simple and fast operation of UDP makes it ideal for delay-sensitive traffic like voice samples (the digital representation of analog speech), multicast digital video, and other types of "real-time" traffic that cannot be resent if lost on the network. This use of UDP is not as originally intended, and there are other things that need to be done before UDP is ready for voice and video, but in the true spirit of Internet innovation, UDP was adapted for these new circumstances.

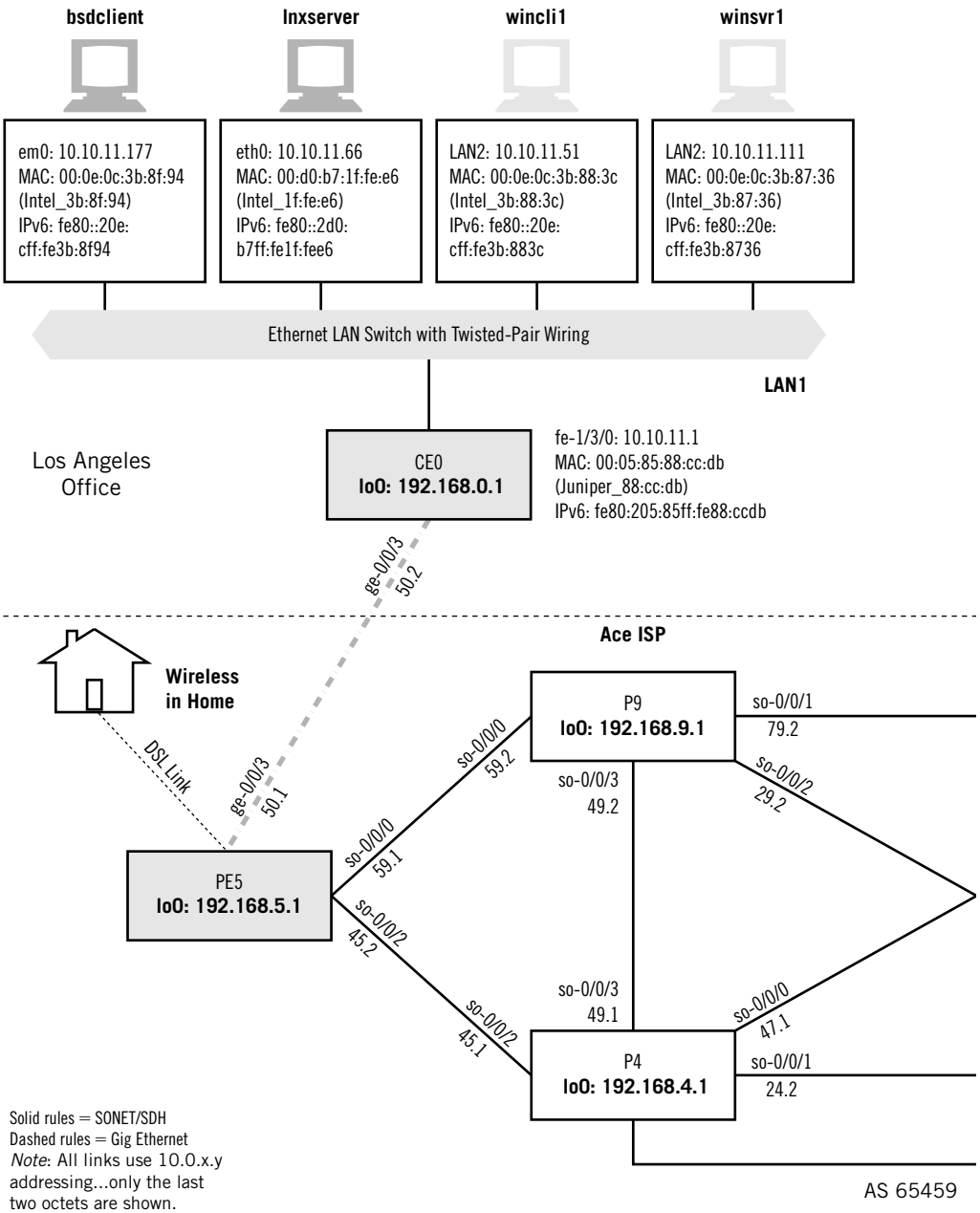
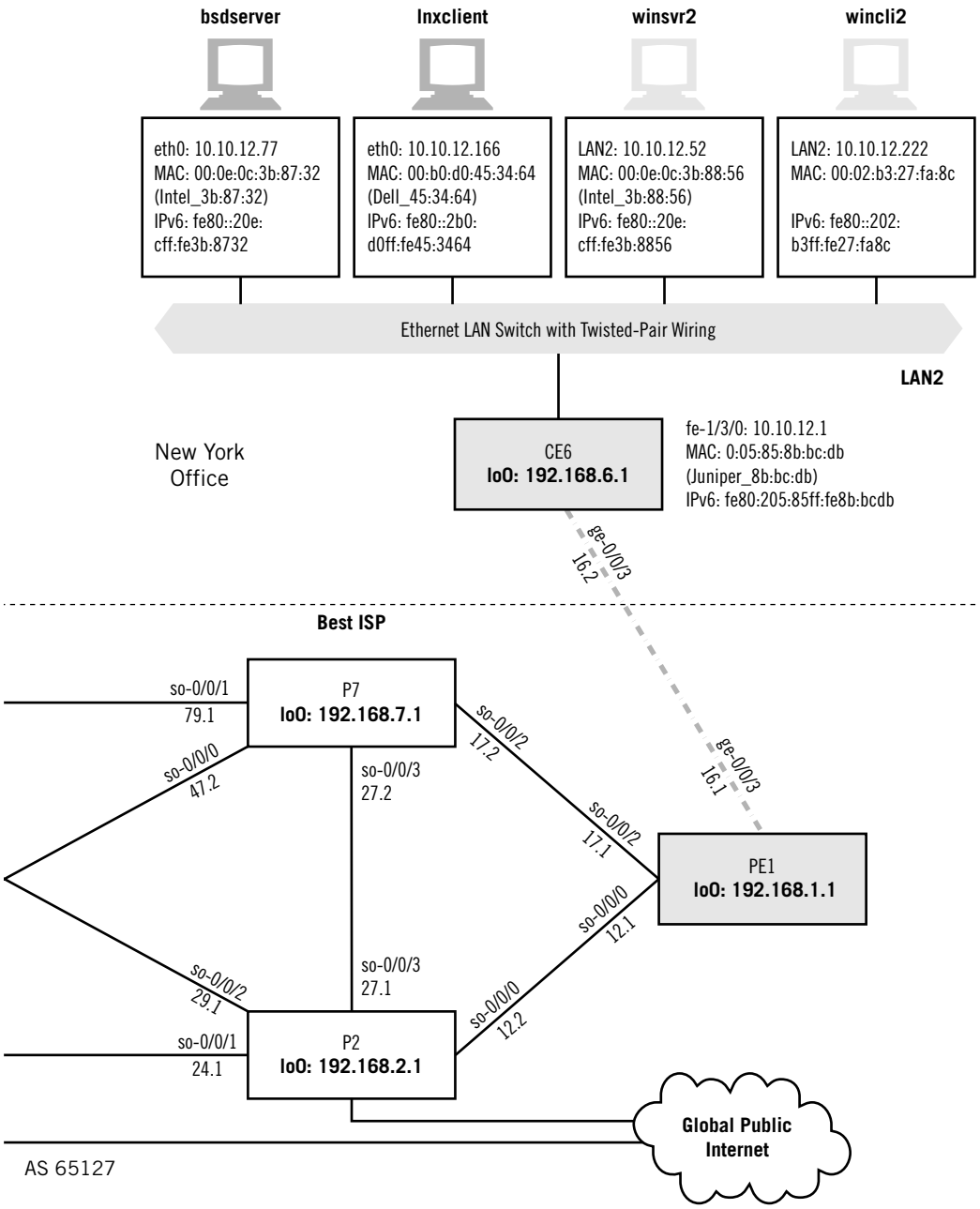


FIGURE 10.1

UDP ports and sockets on the Illustrated Network. Note that this chapter mainly uses the Unix-based hosts on the network to explore UDP.


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-0/0/2
17.2
so-0/0/0
47.2
so-0/0/3
27.2
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

UDP is used by many common network applications, including DNS, IPTV streaming media applications, voice over IP (VoIP), the Trivial File Transfer Protocol (TFTP), and online games. UDP is required for multicast applications.

UDP PORTS AND SOCKETS

Figure 10.1 shows the hosts on the Illustrated Network that we'll be using in this chapter to explore UDP ports and sockets. We'll primarily use the Unix-based hosts, both FreeBSD and Linux.

Let's look at a simple application of UDP between the `lnxclient` and `lnxserver` hosts. The standard Unix "echo" utility (not the same "echo" program as the application used in a previous chapter) sends a simple text string from a client to a server using UDP port 7. The server just bounces a UDP datagram back with the same content. But even with this simple interaction, all of the major points about UDP discussed in this chapter can be illustrated.

The capture is from `lnxserver` (`10.10.11.66`). The server is responding to the `lnxclient` (`10.10.12.166`) request to echo the string "TEST." The important sections of the request and response packets relevant to UDP are highlighted.

```
[root@lnxserver admin]# /usr/sbin/tethereal -V port 7
Capturing on eth0
Frame 1 (60 bytes on wire, 60 bytes captured)
   Arrival Time: May 6, 2008 16:31:30.947137000
   Time delta from previous packet: 0.000000000 seconds
   Time relative to first packet: 0.000000000 seconds
   Frame Number: 1
   Packet Length: 60 bytes
   Capture Length: 60 bytes
Ethernet II, Src: 00:05:85:88:cc:db, Dst: 00:d0:b7:1f:fe:e6
   Destination: 00:d0:b7:1f:fe:e6 (Intel_1f:fe:e6)
   Source: 00:05:85:88:cc:db (Juniper__88:cc:db)
   Type: IP (0x0800)
   Trailer: 00000000000000000000000000000000
Internet Protocol, Src Addr: 10.10.12.166 (10.10.12.166), Dst Addr:
10.10.11.66 (10.10.11.66)
   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: 32
   Identification: 0x0000
   Flags: 0x04
      .1.. = Don't fragment: Set
      ..0. = More fragments: Not set
   Fragment offset: 0
```

```
   Time to live: 62
   Protocol: UDP (0x11)
   Header checksum: 0x10d2 (correct)
   Source: 10.10.12.166 (10.10.12.166)
   Destination: 10.10.11.66 (10.10.11.66)
User Datagram Protocol, Src Port: 32787 (32787), Dst Port: echo (7)
   Source port: 32787 (32787)
   Destination port: echo (7)
   Length: 12
   Checksum: 0xac26 (correct)
Data (4 bytes)

0000  54 45 53 54                                       TEST

Frame 2 (46 bytes on wire, 46 bytes captured)
   Arrival Time: May 6, 2008 16:31:30.948312000
   Time delta from previous packet: 0.001175000 seconds
   Time relative to first packet: 0.001175000 seconds
   Frame Number: 2
   Packet Length: 46 bytes
   Capture Length: 46 bytes
Ethernet II, Src: 00:d0:b7:1f:fe:e6, Dst: 00:05:85:88:cc:db
   Destination: 00:05:85:88:cc:db (Juniper__88:cc:db)
   Source: 00:d0:b7:1f:fe:e6 (Intel_1f:fe:e6)
   Type: IP (0x0800)
Internet Protocol, Src Addr: 10.10.11.66 (10.10.11.66), Dst Addr:
10.10.12.166 (10.10.12.166)
   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: 32
   Identification: 0x0000
   Flags: 0x04
     .1.. = Don't fragment: Set
     ..0. = More fragments: Not set
   Fragment offset: 0
   Time to live: 64
   Protocol: UDP (0x11)
   Header checksum: 0x0ed2 (correct)
   Source: 10.10.11.66 (10.10.11.66)
   Destination: 10.10.12.166 (10.10.12.166)
User Datagram Protocol, Src Port: echo (7), Dst Port: 32787 (32787)
   Source port: echo (7)
   Destination port: 32787 (32787)
   Length: 12
   Checksum: 0xac26 (correct)
Data (4 bytes)

0000  54 45 53 54                                       TEST
```

The DF bit in the packet is set, and the UDP checksum field is used. Technically, the UDP checksum is optional, and the client decides whether to use it. The server responds with a checksum because the client used a checksum in the request. In fact, Windows XP and FreeBSD do the same.

The UDP checksum was made optional to cut processing on reliable networks like small LAN segments to a bare minimum. Today, client and server on the same LAN segment are not very common, and processing the checksum is not a burden for modern computing devices. Also, UDP checksum calculation can be offloaded to modern Ethernet chipsets, so it's less "expensive" than it used to be. Currently, use of the UDP checksum is common, and most traditional texts say it "should" be used with IPv4. Use of the UDP checksum is mandatory with IPv6.

Note that the program uses client UDP port `32787`. This is in the range of ports known as *registered ports*. We'll talk about those, and the *dynamic port* range of 49152 to 65535, later in this chapter. The dynamic port range that a Unix system uses is a kernel-tunable parameter and can be changed using tweaks to the `/etc/sysctl.conf` file, but information on exactly how to do it is scarce and beyond the scope of this book.

We can see the sockets in use on a Linux host by using the `netstat -lp` command to display listening sockets. (Although the options imply these are listening ports, it is the socket information that is displayed.)

```
root@lnxserver admin]# netstat -lp
Active Internet connections (only servers)
Proto Recv-Q Send-Q Local Address           Foreign Address    State
       PID/Program name
tcp        0      0 *:32768                 *:*                LISTEN
       1664/
tcp        0      0 localhost.localdo:32769 *:*                LISTEN
       1783/xinetd
tcp        0      0 localhost.localdoma:783 *:*                LISTEN
       1853/spamd -d -c -a
tcp        0      0 *:sunrpc                *:*                LISTEN
       1645/
tcp        0      0 *:x11                   *:*                LISTEN
       2103/X
tcp        0      0 *:ssh                   *:*                LISTEN
       1769/sshd
tcp        0      0 localhost.localdoma:ipp *:*                LISTEN
       6813/cupsd
tcp        0      0 localhost.localdom:smtp *:*                LISTEN
       1826/
udp        0      0 *:32768                 *:*
       1664/
udp        0      0 *:echo                  *:*
       1923/Echo
udp        0      0 *:sunrpc                *:*
       1645/
```

```
udp        0      0 *:631                   *:*
       6813/cupsd
udp        0      0 localhost.localdoma:ntp *:*
       1800/
udp        0      0 *:ntp                   *:*
       1800/
Active UNIX domain sockets (only servers)
Proto RefCnt Flags       Type       State         I-Node PID/Program name
Path
unix  2      [ ACC ]     STREAM     LISTENING     2663   1939/
/tmp/jd_sockV4
unix  2      [ ACC ]     STREAM     LISTENING     2839   2053/
/tmp/.gdm_socket
unix  2      [ ACC ]     STREAM     LISTENING     2714   2016/
/tmp/.font-unix/fs7100
unix  2      [ ACC ]     STREAM     LISTENING     2542   1872/
/tmp/.iroha_unix/IROHA
unix  2      [ ACC ]     STREAM     LISTENING     2849   2103/X
/tmp/.X11-unix/X0
unix  2      [ ACC ]     STREAM     LISTENING     2535   1862/gpm
/dev/gpmctl
```

The output is difficult to parse, but we can see our little `echo` utility (highlighted, and the second line of the UDP section) patiently waiting for clients on port 7 (the output identifies it as the standard "echo" port). UDP, being a stateless protocol, is not technically in a "listening" state, but that's what the server socket essentially does. The asterisks (*:*) mean that communications will be accepted from another IP address and port.

The command to reveal the same type of information on `bsdserver` is `sockstat`.

```
bsdserver# sockstat
USER     COMMAND    PID   FD PROTO  LOCAL ADDRESS         FOREIGN ADDRESS
root     sendmail    88   4  tcp4   *:25                  *:*
root     sendmail    88   6  tcp4   *:587                 *:*
root     sshd        83   4  tcp4   *:22                  *:*
root     inetd       79   4  tcp4   *:21                  *:*
root     inetd       79   5  tcp4   *:23                  *:*
root     syslogd     72   5  udp4   *:514                 *:*

USER     COMMAND    PID   FD PROTO  LOCAL ADDRESS         FOREIGN ADDRESS
root     sendmail    88   5  tcp46  *:25                  *:*
root     sshd        83   3  tcp46  *:22                  *:*
root     syslogd     72   4  udp6   *:514                 *:*
USER     COMMAND    PID   FD PROTO  ADDRESS
admin    sshd     48218   3  stream sshd[48216]:4
root     sshd     48216   4  stream sshd[48218]:3
smmsp    sendmail    91   3  dgram  syslogd[72]:3
root     sendmail    88   3  dgram  syslogd[72]:3
root     syslogd     72   3  dgram  /var/run/log
```

The little "echo" port is not listed because it is not running on this host. Note that the `syslogd` process in FreeBSD listens on *both* the UDP and TCP ports (in this case, port 514) for clients.

What about Windows XP? The command here is `netstat -a` (all), but be prepared to be surprised. Windows hosts listen to a larger number of sockets than Unix systems. It depends on exactly what the system is doing, but even on our "quiet" test network, `winsrv2` has 25 TCP and 19 UDP processes waiting to spring into action. They range from `Netbios` (an old IBM and Microsoft LAN protocol) to Microsoft-specific functions. Heavily loaded systems have even higher numbers.

What about looking at UDP with IPv6? It's not really necessary. We are now high enough in the TCP/IP protocol stack not to worry about differences between IPv4 and IPv6. (In practical terms, we still have to worry about DNS a bit, but we'll talk about that in Chapter 19.) With the exception of the checksum use and something called the *pseudo-header*, UDP is the same in both.

WHAT UDP IS FOR

UDP was defined in RFC 768 and refined in RFC 1122. All implementations must follow both RFCs to make interoperability reliable, and all do. UDP uses IP protocol ID 17. Any IPv4 or IPv6 packet received with 17 in the protocol ID field is given to the local UDP service.

UDP is defined as stateless (no session information is kept by hosts) and unreliable (no guarantees of any QoS parameters, not even delivery). This does *not* mean that UDP traffic is somehow lower priority on the network or through routers. It's not as if UDP traffic is routinely tossed by stressed-out routers. It just means that if the application using UDP needs to keep track of a session history ("How many datagrams did you get before that link failed?") or guaranteed delivery ("I'm not sending any more until I know if you got the datagrams I sent."), then the *application* itself must do it, because UDP can't and won't.

Nevertheless, there is a whole class of applications that use UDP, some almost exclusively. These are applications that are invoked to exchange quick, request–response pairs of messages, such as DNS ("Quick! What IP address goes with `www.example.com`?"). These applications could suffer while waiting for all the overhead that TCP requires to set up a connection between hosts before sending a message.

Multicast allows one source to send a *single* packet stream to multiple destinations (TCP is strictly a one-source-to-one-destination protocol), so UDP must be used for multicast data transfer as well. Multicast is not only used with video or audio, but also in applications such as the Dynamic Host Configuration Protocol (DHCP).

In other words, UDP is a low-overhead transport for applications that do not need, or cannot have, the "point-to-point" connections or guaranteed delivery that TCP provides.

Packets carrying UDP traffic in IPv4 sometimes have the DF (Don't Fragment) bit set in the IPv4 header. However, no one should be surprised or upset to find a UDP datagram riding inside an IPv4 packet without the DF bit set.

THE UDP HEADER

Figure 10.2 shows the UDP header. There are only four fields, and the data inside the datagram (the message) are optional.

The header is only 8 bytes (64 bits) long. First are the 2-byte Source Port field and the 2-byte Destination Port field. These fields are the datagram counterparts of the source and destination IP addresses at the packet level. But unlike IP addresses, there is no structure to the port fields: All values between 0 and 65,353 are represented as pure numerics. This does not mean that all port numbers, source and destination, are the same, however. Port values can be divided into well-known, registered, and dynamic port numbers.

The Length field gives the length in bytes of the UDP datagram, and includes the header fields along with any data. The minimum length is 8 (the header alone), and the maximum value is 65,353. However, the achievable maximum UDP datagram lengths are determined by the size of the send and receive buffers on the host end systems, which are usually set to around 8000 bytes (although they can be changed).

As already mentioned, hosts are required to handle 576-byte IP packets at a minimum, but many protocols (the most common being DNS and DHCP) limit the maximum size of the UDP datagram that they use to 512 bytes or less.

The Checksum field is the most interesting field in the UDP header. This is because the checksum is *not* a simple value calculated on the UDP header fields and data, if present. The UDP checksum is computed on what is called the *pseudo-header*. The pseudo-header fields for IPv4 are shown in Figure 10.3.

The all-zero byte is used to provide alignment of the pseudo-header, and the data field must be padded to align it with a 16-bit boundary. The 12 bytes of the UDP pseudo-header are prepended to the UDP datagram, and the checksum is computed on the whole object. For this computation, the Checksum field itself is set to zero, and the 16-bit result placed in the field before transmission. If the checksum computes to zero, an all-1s value is sent, and all-1s is not a computable checksum. The pseudo-header fields are not sent with the datagram.

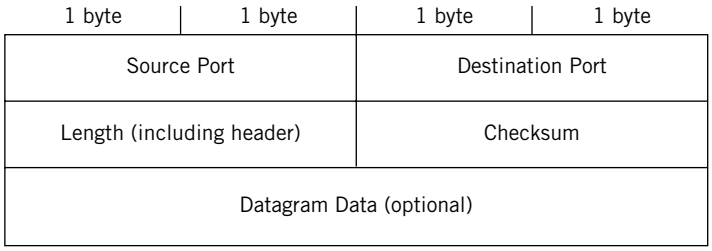


FIGURE 10.2

The four UDP header fields. Technically, use of the checksum is optional, but it is often used today.

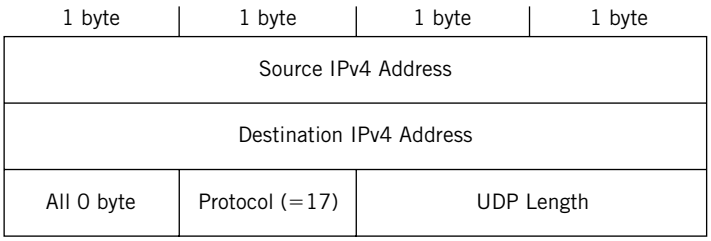


FIGURE 10.3

The UDP IPv4 pseudo-header. These fields are used for checksum computation and include fields in the IP header.

At the receiver, the value of the Checksum is copied and the field again set to zero. The checksum is again computed on the pseudo-header and compared to the received value. If they match, the datagram is processed by the receiving application indicated by the destination port number. If they do not match, the datagram is silently discarded (i.e., no error message is sent to the source).

Naturally, using 32-bit IPv4 addresses to compute transport layer checksums will not work in IPv6, although the procedure is the same. RFC 2460 establishes a different set of pseudo-header fields for IPv6. The IPv6 pseudo-header is shown in Figure 10.4.

The Next Header value is not always 17 for UDP, because other extension headers could be in use. Length is the length of the upper layer header and the data it carries.

IPv4 AND IPv6 NOTES

The presence of the IP source and destination address in an upper layer checksum computation strikes many as a violation of the concept of protocol layer independence. (The same concern applies to NAT, discussed in Chapter 27.) In fact, a lot of TCP/IP books mention that including packet level fields in the end-to-end checksum helps assure (when the checksum is correct at the receiver) that the message has not only made its way to right port, but to the correct system.

The presence of a pseudo-header also shows how late in the development process that TCP and UDP were separated from IP. Not only that, but the transport layer and network layer (or, to give them more intuitive names, the end-to-end layer and routing layer) have *always* been tightly coupled in any working network.

The use of the UDP checksum is not required for IPv4, but highly recommended. It is required in IPv6, of course. In IPv4, servers that receive client datagrams with the checksum field set are supposed to reply using the checksum, but this is not always enforced. If the IPv4 checksum field is not used, it is set to all 0 bits (recall that all 0 checksums are sent as all-1s).

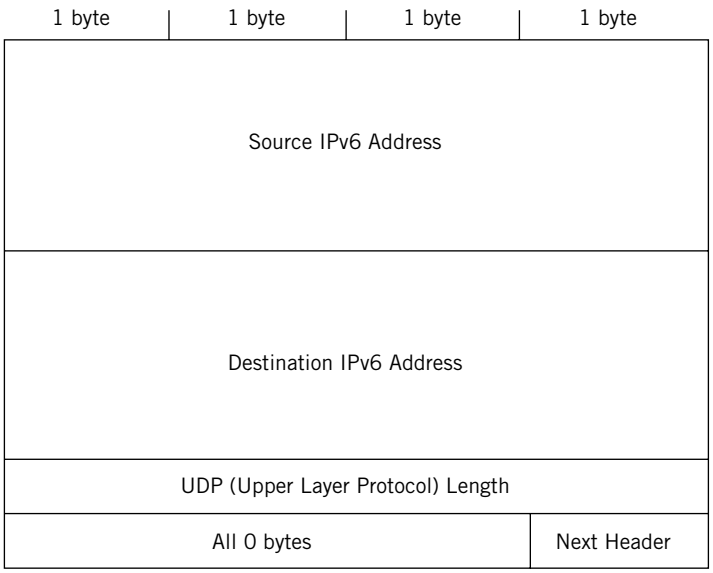


FIGURE 10.4

The UDP IPv6 pseudo-header. Use of the UDP checksum is not optional in IPv6.

PORT NUMBERS

Each application running above UDP (and TCP) and IP is indexed by its port number, allowing for the multiplexing of the IP layer. Just as frames with different types of packets inside (on Ethernet, IPv4 is `0x0800` and IPv6 is `0x86DD`) are multiplexed onto a single LAN interface, the individual IPv4 or IPv6 packets are multiplexed and distributed by the protocol number (UDP is IP protocol number 17, and TCP is 6).

The port numbers in turn multiplex and distribute datagrams from applications, allowing them to share a single UDP or TCP process, which is usually integrated closely with the operating system. This function of frame Ethertype, packet protocol, and datagram port is shown in Figure 10.5. The figure shows how IPv4 data for DNS makes its way from frame through IPv4 through UDP to the DNS application listening on UDP port 53.

Well-Known Ports

Port numbers can run from 0 to 65353. Port numbers from 0 to 1023 are reserved for common TCP/IP applications and are called *well-known ports*. The use of well-known ports allows client applications to easily locate the corresponding server application processes on other hosts. For example, a client process wanting to contact a DNS

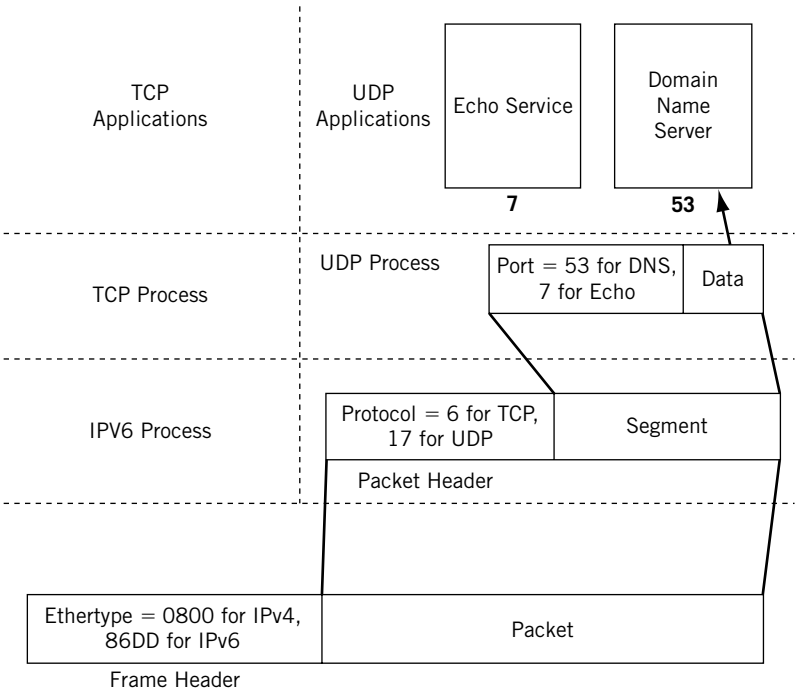


FIGURE 10.5

UDP port multiplexing and distribution, showing how a single IP layer (IPv6 in this case) can be used by multiple transport protocols and applications.

process running on a server must send the datagram to some destination port. The well-known port number for DNS is 53, and that's where the server process should be listening for client requests. These ports are sometimes called "privileged" ports, although a number of applications that formerly ran in "privileged" mode, such as HTTP servers, do not run this way anymore except when binding to the port. It should be noted that it is getting harder and harder to register new applications in the space below 1023 (these often use registered ports in the range 1024 to 49151).

Ports used on servers are *persistent* in the sense that they last for a long time, or at least as long as the application is running. Ports used on clients are *ephemeral* ("lasting a short time," although the term technically means "lasting a day") in the sense that they "come and go" as the user runs client applications.

Technically, UDP port numbers are independent from TCP port numbers. In practice, most of the applications indexed by port numbers are the same in UDP or TCP (although a few applications can use either protocol), excepting a handful that are maintained for historical reasons. This does not imply that applications can use TCP or UDP as they choose. It just means that it's easier to maintain one list rather than two. But no matter what port numbers are used, UDP port 1000 is a different

application than TCP port 1000, even though both applications might perform the same function.

Some of the more common well-known port numbers are shown in Table 10.1. In the table, the UDP and TCP port numbers are identical.

Port numbers above 1023 can be either *registered* or *dynamic* (also called *private* or *non-reserved*). Registered ports are in the range 1024 to 49151. Dynamic ports are in the range 49152 to 65535. As mentioned, most new port assignments are in the range from 1024 to 49151.

Registered port numbers are non–well-known ports that are used by vendors for their own server applications. After all, not every possible application capability will be reflected in a well-known port, and software vendors should be free to innovate. Of course, if another vendor chooses the same port number for a server process, and they are run on the same system, there would be no way to distinguish between these two seemingly identical applications.

- *Well-known ports*—Ports in the range 0 to 1023 are assigned and controlled.
- *Registered ports*—Ports in the range 1024 to 49151 are not assigned or controlled, but can be registered to prevent duplication.
- *Dynamic ports*—Ports in the range 49152 to 65535 are not assigned, controlled, or registered. They are used for temporary or private ports. They are also known as private or non-reserved ports. Clients should choose ephemeral port numbers from this range, but many systems do not.

Table 10.1 Some Well-Known Ports Used by UDP and TCP Services and Functions

Port Number	Service	Meaning
7	Echo	Used to echo data back to the sender
9	Discard	Used to discard data at receiver
13	Daytime	Reports time information in user-friendly format
17	Quote	Returns a "quote of the day" (rarely used today)
19	Chargen	Character generator
53	DNS	Domain Name Service
67	DHCP server	Server port used to send configuration information
68	DHCP client	Client port used to receive configuration information
69	TFTP	Trivial file transfer
161	SNMP	Used to receive network management queries
162	SNMP traps	Used to receive network problem reports
1011–1023	Reserved	Reserved for future use

Vendors can register their application's ports with ICANN. Other software vendors are supposed to respect these registered values and register their own server application port numbers from the pool of unused values. Some registered UDP and TCP protocol numbers are shown in Table 10.2.

The private, or dynamic, port numbers are used by clients and *not* servers. Datagrams sent from a client to a server are typically only sent to well-known or registered ports (although there are exceptions). Server applications are usually long lived, while client processes come and go as users run them. Client applications therefore are free to choose almost any port number not used for some other purpose (hence the term "dynamic"), and many use different source port numbers every time they are run. The server has no trouble replying to the proper client because the server can just reverse the source and destination port numbers to send a reply to the correct client (assuming the IP address of the client is correct).

All TCP/IP implementations must know the range of well-known, registered, and private ports when choosing a port number to use. Unix systems hold this information is the `/etc/services` file. Windows users can find this `C:\%SystemRoot%\system32\drivers\etc\SERVICES` file, where `%SystemRoot%` will be automatically referred to a folder such as `WinNT` or `WINDOWS`. Most ports are the same for UDP or TCP, but some are unique to one or the other. For example, FTP control uses TCP port 21.

Table 10.2 Selected Registered UDP and TCP Ports with Service and Brief Description of Meaning

Port Number	Service	Brief Description of Use
1024	Reserved	Reserved for future use
1025	Blackjack	Network version of blackjack
1026	CAP	Calendar access protocol
1027	Exosee	ExoSee
1029	Solidmux	Solid Mux Server
1102	Adobe 1	Adobe Server 1
1103	Adobe 2	Adobe Server 2
44553	Rbr-debug	REALBasic Remote Debug
46999	Mediabox	MediaBox Server
47557	Dbbrowse	Databeam Corporation
48620–49150	Unassigned	These ports have not been registered
49151	Reserved	Reserved for future use

Here is the beginning of the file from `winsvr2`:

```
# Copyright (c) 1993-1999 Microsoft Corp.
#
# This file contains port numbers for well-known services defined by IANA
#
# Format:
#
# <service name>  <port number>/<protocol>  [aliases...]   [#<comment>]
#

echo          7/tcp
echo          7/udp
discard       9/tcp        sink null
discard       9/udp        sink null
systat       11/tcp        users           #Active users
systat       11/tcp        users           #Active users
daytime      13/tcp
daytime      13/udp
qotd         17/tcp        quote           #Quote of the day
qotd         17/udp        quote           #Quote of the day
chargen      19/tcp        ttytst source   #Character generator
chargen      19/udp        ttytst source   #Character generator
ftp-data     20/tcp                        #FTP, data
ftp          21/tcp                        #FTP. control
telnet       23/tcp
[many more lines not shown...]
```

For the latest global list of well-known, registered, and private port numbers, see *www.iana.org/assignments/port-numbers*. The port numbers are the same for IPv4 and IPv6.

The Socket

The combination of IPv4 or IPv6 address and port numbers forms an abstract concept called a *socket*. We've mentioned the socket concept briefly before, and will do so again and again in later chapters. The socket concept is important for many reasons, and a later chapter will explore some of them more completely. For now, all that is important to mention is that, for each client–server interaction, there is a socket on each host at the endpoints of the network. The sockets at each end uniquely identify that particular client–server interaction, although the same sockets can be used for subsequent interactions.

Sockets are usually written in IPv4 and IPv6 by adding a colon (:) to the IP address, although sometimes a dot (.) is used instead. In IPv6, it is also necessary to add brackets to avoid confusion with the `::` notation, such as in `[FC00:490:f100:1000::1]:80`. A UDP socket on `lnxclient`, for example, would be `10.10.12.166:17`, while one on `bsdserver` would be `10.10.12.77:17`.

Action	Condition	Outcome
UDP request sent to server	Server available	Sender gets UDP reply from server
UDP request sent to server	Port is closed on server	Sender gets ICMP "Port unreachable" message
UDP request sent to server	Server host does not exist	Sender gets ICMP "Host unreachable" message
UDP request sent to server	Port is blocked by firewall/router	Sender gets ICMP "Port unreachable—Administrative prohibited"message
UDP request sent to server	Port is blocked by silent firewall/router	(timeout)
UDP request sent to server	Reply is lost on way back	(timeout)

FIGURE 10.6

UDP protocol actions, showing the request–reply outcomes.

UDP OPERATION

The delivery of UDP datagrams is by no means certain. The lack of an expected response on the part of a server to a UDP client request is handled by a simple timeout. Responses are not always expected, as might be the case with streaming audio and video. The client might resend the datagram, but in many cases this might not be the best strategy.

In some cases, lack of response is not a reliable indication that anything is wrong with the network or remote host. Routers routinely filter out unwanted packets, and many do so silently, while others send the appropriate ICMP "administratively prohibited" message.

In general, there are five major possible results when an application sends a UDP request, shown in Figure 10.6. Note that *any* of the replies can be lost on the way back to the sender, generating a timeout.

UDP OVERFLOWS

We've looked at UDP as a sort of quick-and-dirty request–response interaction between hosts over a network. Delivery is not guaranteed, but neither is an important network property called *flow control*. A lot of nonsense has been written about flow control, which is a very simple idea. It just means that no sender should ever be able to

overwhelm a receiver with traffic. In other words, receivers must have a way to tell senders to slow down. UDP, of course, has no such mechanism.

The confusion over flow control often comes from treating flow control as a synonym for a related concept called *congestion control*. While flow control is strictly a local property of individual senders and receivers, congestion control is a global property of the network. No sender overwhelms a receiver: There's just too much traffic in the router network for things to work properly.

Congestion control often uses flow control to accomplish its goals (source quench was a not-too-sophisticated mechanism). There's not much else a router can use other than flow control to tell senders to shut up for a while. But that's no excuse for treating the two as one and the same.

What has this to do with UDP? Well, it is possible for UDP receivers' buffers, which are usually fixed, to overflow with unexpected UDP datagrams and be forced to discard traffic. Most UDP implementations include a way to display "UDP socket overflows" or discarded UDP datagrams.

But what if an application needs guaranteed delivery, sequencing, and flow control to work properly, and we don't want to add these to the application? Files cannot use quick request–response messages to transfer themselves over a network. That's the job of TCP, which is the topic of the next chapter.

QUESTIONS FOR READERS

Figure 10.7 shows some of the concepts discussed in this chapter and can be used to help you answer the following questions.

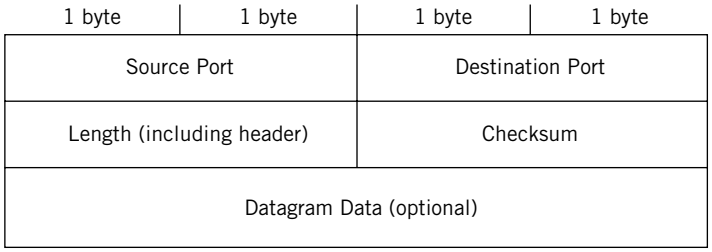


FIGURE 10.7

The UDP header (a) and pseudo-header (b) fields for IPv4.

1. Which UDP header field does UDP use for demultiplexing?
2. What is UDP's only attempt at error control?
3. A socket is comprised of which two TCP/IP components?
4. What is the registered port range? Is this assigned or controlled?
5. What is the dynamic or private port range? Are these assigned or controlled?