Lab 6 User Datagram Protocol (UDP)

1. Process-To-Process Delivery

The data link layer is responsible for delivery of frames between two neighboring nodes over a link. This is called node-to-node delivery. The network layer is responsible for delivery of datagrams between two hosts. This is called host-to-host delivery. Communication on the Internet is not defined as the exchange of data between two nodes or between two hosts. Real communication takes place between two processes (application programs). We need *process-to-process* delivery. However, at any moment, several processes may be running on the source host and several on the destination host. To complete the delivery, we need a mechanism to deliver data from one of these processes running on the source host to the corresponding process running on the destination host.

The transport layer is responsible for process-to-process delivery—the delivery of a packet, part of a message, from one process to another. Two processes communicate in a client/server relationship, as we will see later. Figure 6-1 shows these three types of deliveries and their domains.

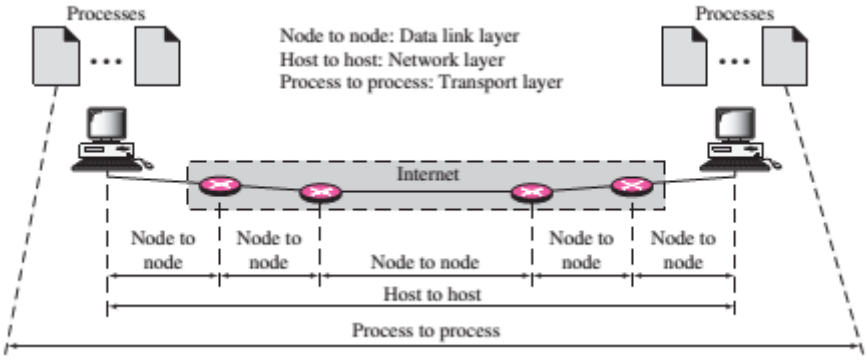

Figure 6-1 Types of data deliveries

1.1. Port Number

At the network layer, we need an IP address to choose one host among millions. A datagram in the network layer needs a destination IP address for delivery and a source IP address for the destination's reply.

At the transport layer, we need a transport layer address, called a port number, to choose among multiple processes running on the destination host. The destination port number is needed for delivery; the source port number is needed for the reply.

In the Internet model, the port numbers are 16-bit integers between 0 and 65,535. The client program defines itself with a port number, chosen randomly by the transport layer software running on the client host. This is the **ephemeral port number**.

The server process must also define itself with a port number. This port number, however, *cannot be chosen randomly.* If the computer at the server site runs a server process and assigns a random number as the port number, the process at the client site that wants to access that server and use its services will not know the port number. Of course, one solution would be to send a

special packet and request the port number of a specific service, but this requires more overhead. The Internet has decided to use universal port numbers for servers; these are called **well-known port numbers**. There are some exceptions to this rule; for example, there are clients that are assigned well-known port numbers. *Every client process knows the well-known port number of the corresponding server process*.

Well-known port numbers are shown as follows:

Table 6-1 Well-known port numbers of UDP

Port	Protocol	Description
7	Echo	Echoes a received datagram back to the sender
8	Discard	Discards any datagram that is received
11	Users	Active users
13	Daytime	Returns the date and the time
17	Quote	Returns a quote of the day
19	Chargen	Returns a string of characters
53	Nameserver	Domain Name Service
67	BOOTPs	Server port to download bootstrap information
68	BOOTPc	Client port to download bootstrap information
69	TFTP	Trivial File Transfer Protocol
111	RPC	Remote Procedure Call
123	NTP	Network Time Protocol
161	SNMP	Simple Network Management Protocol
162	SNMP	Simple Network Management Protocol (trap)

1.2. Socket Addresses

Process-to-process delivery needs two identifiers, IP address and the port number, at each end to make a connection. The combination of an IP address and a port number is called a socket address. The client socket address defines the client process uniquely just as the server socket address defines the server process uniquely.

A transport layer protocol needs a pair of socket addresses: the client socket address and the server socket address. These four pieces of information are part of the IP header and the transport layer protocol header. The IP header contains the IP addresses; the UDP or TCP header contains the port numbers.

2. Connectionless Versus Connection-Oriented Service

Flow the view of communication, in the OSI-model, an upper layer can provides two kinds of servers: Connection-Oriented Service and Connectionless Service.

2.1. Connection-Oriented Service

Connection means combing two equal entries for communication. Before communication, connection-oriented service establishes the connection. After transferring data, the connection will be released. Therefore, using connection-oriented service to transfer data contains three phases:

Establish the connection: In the service primitive and protocol data unit, source and destination addresses must be given. Meanwhile the quality of service and other options can be negotiated.

Transfer the data: In this phase, there is no need to include the complete source and

destination addresses in each packet. Instead, a connection identification is used to identify a packet. Due to the shorter connection identification, control messages are much less than the size of packets, leading to lower system cost and thus improving the efficiency of communication channel. In addition, packets are sent and received in fixed order.

Release the connection: Some service primitives are used to release the connection.

Considering the three phases of connection-oriented service, connection acts like a pipe. Senders send packets from one end and receivers get them in the same order on the other end. This connection is also called **Virtual Circuit**, which preserves packets from being lost, repeated and out of order.

If two users usually need to communicate, permanent virtual circuits can be established, which avoid the establishing and releasing phases, like the private line in the telephone network.

2.2. Connectionless Service

In the case of connectionless service, there is no need to establish a connection between two entries. Hence, connectionless service is agiler and convenient. However, it is unable to preserve packets from being lost, repeated and out of order. Besides, including complete source and destination addresses in each packets leads to a higher cost.

There are 3 types of connectionless services:

Datagram: With features of finishing after sending packets and no responses happening, datagram service is simple, less-cost but unreliable. It is suitable for the communication which has a large redundancy and requires high real-time.

Confirmed delivery: It is also called reliable datagram. This kind of service generates a corresponding confirming message to each packet. However, this confirming message is not from the end user but from the layer which provides services. Thus it makes sure that packets are sent to the receiver successfully but whether packets are received correctly is not confirmed.

Request-reply: When getting a packet, receivers reply senders with reply messages. However, packets from both sides are probably lost. If the receiver finds there is something wrong with the packet, it replies with a packet representing such error.

3. Introduction of UDP

The **User Datagram Protocol** (UDP) is one of the core members of the Internet protocol suite. UDP uses a simple connectionless transmission model with a minimum of protocol mechanism. It has no handshaking dialogues, and thus exposes any unreliability of the underlying network protocol to the user's program. There is no guarantee of delivery, ordering, or duplicate protection. UDP provides checksums for data integrity, and port numbers for addressing different functions at the source and destination of the datagram.

With UDP, computer applications can send messages, in this case referred to as datagrams, to other hosts on an Internet Protocol (IP) network without prior communications to set up special transmission channels or data paths. UDP is suitable for purposes where error checking and correction is either not necessary or is performed in the application, avoiding the overhead of such processing at the network interface level. Time-sensitive applications often use UDP because dropping packets is preferable to waiting for delayed packets, which may not be an option in a real-time system [1]. If error correction facilities are needed at the network interface level, an application may use the Transmission Control Protocol (TCP) or Stream Control Transmission Protocol (SCTP) which are designed for this purpose.

A number of UDP's attributes make it especially suited for certain applications.

- It is transaction-oriented, suitable for simple query-response protocols such as the Domain Name System or the Network Time Protocol.
- It provides datagrams, suitable for modeling other protocols such as in IP tunneling or Remote Procedure Call and the Network File System.
- It is simple, suitable for bootstrapping or other purposes without a full protocol stack, such as the DHCP and Trivial File Transfer Protocol.
- It is stateless, suitable for very large numbers of clients, such as in streaming media applications for example IPTV
- The lack of retransmission delays makes it suitable for real-time applications such as Voice over IP, online games, and many protocols built on top of the Real Time Streaming Protocol.
- Works well in unidirectional communication, suitable for broadcast information such as in many kinds of service discovery and shared information such as broadcast time or Routing Information Protocol

4. Packet format of UDP

UDP packets, called user datagrams, have a fixed-size header of 8 bytes. Figure 6-2 shows the format of a user datagram.

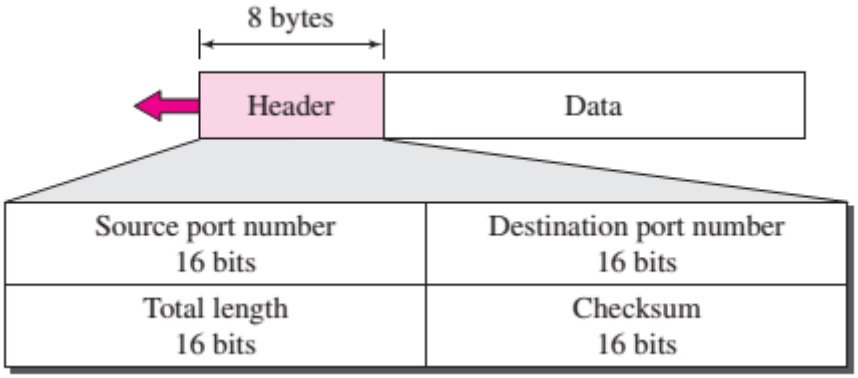

Figure 6-2 Packet format of UDP

Source port number: This is the port number used by the process running on the source host. It is 16 bits long, which means that the port number can range from 0 to 65,535. If the source host is the client (a client sending a request), the port number, in most cases, is an ephemeral port number requested by the process and chosen by the UDP software running on the source host. If the source host is the server (a server sending a response), the port number, in most cases, is a well-known port number.

Destination port number: This is the port number used by the process running on the destination host. It is also 16 bits long. If the destination host is the server (a client sending a request), the port number, in most cases, is a well-known port number. If the destination host is the client (a server sending a response), the port number, in most cases, is an ephemeral port number. In this case, the server copies the ephemeral port number it has received in the request packet.

Length: This is a 16-bit field that defines the total length of the user datagram, header plus data. The 16 bits can define a total length of 0 to 65,535 bytes. However, the total length needs to be much less because a UDP user datagram is stored in an IP datagram with a total length of 65,535 bytes. The length field in a UDP user datagram is actually not necessary. A user datagram

is encapsulated in an IP datagram. There is a field in the IP datagram that defines the total length. There is another field in the IP datagram that defines the length of the header. So if we subtract the value of the second field from the first, we can deduce the length of a UDP datagram that is encapsulated in an IP datagram. However, the designers of the UDP protocol felt that it was more efficient for the destination UDP to calculate the length of the data from the information provided in the UDP user datagram rather than ask the IP software to supply this information. We should remember that when the IP software delivers the UDP user datagram to the UDP layer, it has already dropped the IP header.

Checksum: This field is used to detect errors over the entire user datagram (header plus data).

5. UDP Encapsulation

When there exists datagrams to send through UDP, the process adds a pair of socket addresses and the length of data to the UDP. Then UDP attaches the header to the data and deliver them to the IP layer with this pair of socket addresses. IP layer also attaches its header to the packet and sets the field of **Protocol** as 17, indicating that this data comes from UDP. After that, this IP datagram is delivered to the Data Link Layer and encapsulated with header (maybe there is tail). At last, the physical layer encodes each bit by electric or optical signals and sends them to the remote host.

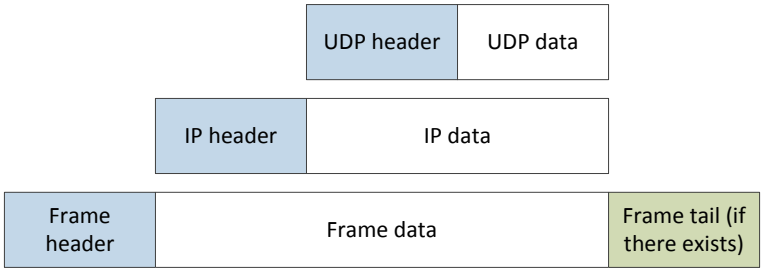

Figure 6-3 UDP Encapsulation

6. UDP Checksum

The UDP checksum calculation is different from the one for IP and ICMP. Here the checksum includes three sections: a pseudo header, the UDP header, and the data coming from the application layer.

The pseudo header is the part of the header of the IP packet in which the user datagram is to be encapsulated with some fields filled with 0s (see Figure 6-4).

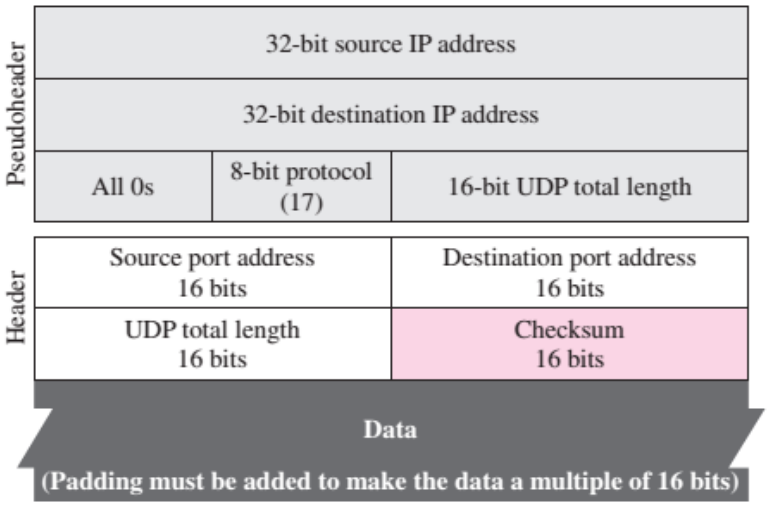

Figure 6-4 Add pseudo header to UDP datagram

If the checksum does not include the pseudo header, a user datagram may arrive safe and sound. However, if the IP header is corrupted, it may be delivered to the wrong host.

The protocol field is added to ensure that the packet belongs to UDP, and not to other transport-layer protocols. We will see later that if a process can use either UDP or TCP, the destination port number can be the same. The value of the protocol field for UDP is 17. If this value is changed during transmission, the checksum calculation at the receiver will detect it and UDP drops the packet. It is not delivered to the wrong protocol. Note the similarities between the pseudo header fields and the last 12 bytes of the IP header.

6.1. Checksum calculation of the sender

The sender calculates UDP checksum as follows:

1) Add pseudo header to the UDP datagram.
2) Fill the field of **Checksum** with all “0”s.
3) Fragment the datagram by length of 16 bits.
4) If the number of fragmentations is not an even number, then add a new fragmentation and fill it with all “0”s. This new fragmentation is used to calculate the checksum and will be dropped after the calculation.
5) Add all 16-bit fragmentations by using Radix-minus-one complement arithmetic addition.
6) Flip the result and insert it to the field of checksum.
7) Drop the pseudo header.
8) Deliver the UDP datagram to the IP layer.

The order of lines in the pseudo header and the additional “0”s have no effect on the calculation.

The following figure gives an example of UDP checksum. Here we suppose the length of UDP is 15 bytes and thus we need to add a new byte of all “0”s.

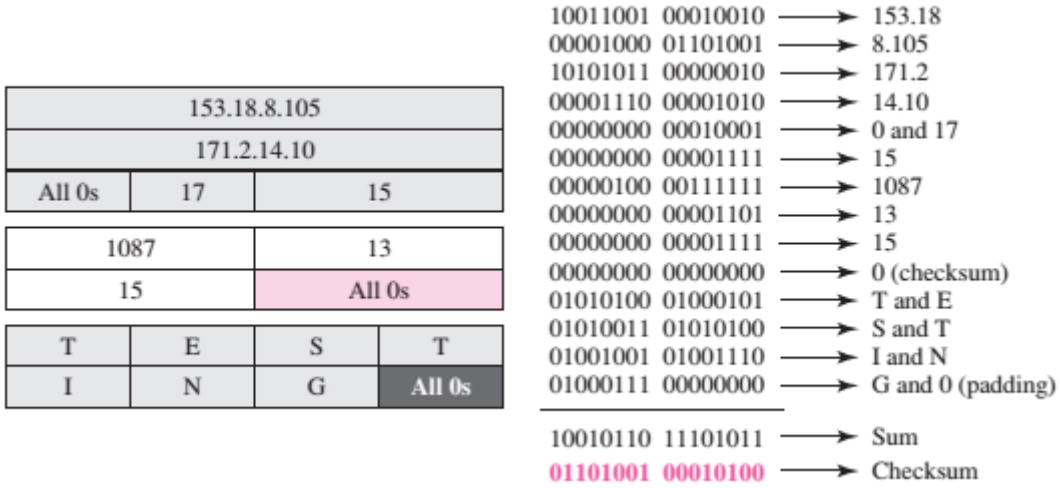

Figure 6-5 the process of UDP checksum

6.2. Checksum calculation of the receiver

The receiver calculates and verifies the UDP checksum as follows:

1) Add pseudo header to the UDP datagram.
2) Fill the datagram if necessary.
3) Fragment the datagram by length of 16 bits.
4) Add all 16-bit fragmentations by using Radix-minus-one complement arithmetic addition.
5) Flip the result.
6) If the result are all "0"s, then drop the header and the padding, and accept this datagram. Otherwise, drop this datagram.

During the calculation, the checksum is optional. If the checksum is not included, this field must be filled with "0"s.

7. Use of UDP

The following lists some uses of the UDP protocol:

- UDP is suitable for a process that requires simple request-response communication with little concern for flow and error control. It is not usually used for a process such as FTP that needs to send bulk data.
- UDP is suitable for a process with internal flow and error control mechanisms. For example, the Trivial File Transfer Protocol (TFTP) process includes flow and error control. It can easily use UDP.
- UDP is a suitable transport protocol for multicasting. Multicasting capability is embedded in the UDP software but not in the TCP software.
- UDP is used for management processes such as SNMP.
- UDP is used for some route updating protocols such as Routing Information Protocol (RIP)