Chapter 5

Transport Layer I

The transport layer is located between the network layer and the application layer. The transport layer is responsible for providing services to the application layer; it receives services from the network layer. It is responsible for process-to-process delivery of the entire message. A **process** is an application program running on a host. Whereas the network layer oversees source-to-destination delivery of individual packets, it does not recognize any relationship between those packets. It treats each one independently, as though each piece belonged to a separate message, whether or not it does. It also ensures that the whole message arrives intact and in order, overseeing both error control and flow control at the source-to-destination level.

A **connection is** a permanent connection (or session) between source and destination devices that is established prior to forwarding any traffic. Session establishment prepares the devices to communicate with one another. Through session establishment, the devices negotiate the amount of traffic that can be forwarded at a given time, and the communication data between the two can be closely managed. The session is terminated only after all communication is completed.

Transport layer protocols are of two kinds: connection-oriented protocols and connectionless protocols. In connection oriented protocol, we have to establish a connection before starting the communication. When connection is established, we send the message or the information and then we release the connection. Connection oriented service is more reliable than connectionless service. We can send the message in connection oriented service if there is an error at the receivers end. Example of connection oriented is TCP (Transmission Control Protocol) protocol. The connection less protocol is similar to the postal services, as it carries the full address where the message (letter) is to be carried. Each message is routed independently from source to destination. The order of message sent can be different from the order received. In connectionless protocol, the data is transferred in one direction from source to destination without checking that destination is still there or not or if it is prepared to accept the message. Authentication is not needed in this case. Example of Connectionless service is UDP (User Datagram Protocol) protocol. The differences between the connection-oriented and connectionless protocols are given below:

- In connection oriented service authentication is needed, while connectionless service does not need any authentication.
- Connection oriented protocol makes a connection and checks whether message is received or not and sends again if an error occurs, while connectionless service protocol does not guarantees a message delivery.
- Connection oriented service is more reliable than connectionless service.
- Connection oriented service interface is stream based and connectionless is message based.

Transport layer has two protocols: User datagram protocol (UDP) and Transmission control protocol (TCP). Each of these protocols are described next.

5.1 User Datagram Protocol (UDP)

The User Datagram Protocol (UDP) is called a connectionless, unreliable transport protocol. It does not add anything to the services of IP except to provide process-to-process communication instead of host-to-host communication. Also, it performs very limited error checking. UDP is a very simple protocol using a minimum of overhead. If a process wants to send a small message and does not care much about reliability, it can use UDP. Sending a small message by using UDP takes much less interaction between the sender and receiver than using TCP or SCTP. Data may be lost or received out of sequence without any UDP mechanisms to recover or reorder the data. If reliability is required when using UDP as the transport protocol, it must be handled by the application. Due to low control overhead, limited error control and no flow control, it is faster.

UDP is a connectionless protocol. Each UDP packet (also called user datagram) is independent from other packets sent by the same application program. This feature can be considered as an advantage or disadvantage depending on the application requirement. It is an advantage if, for example, a client application needs to send a short request to a server and to receive a short response. If the request and response can each fit in one single user datagram, a connectionless service may be preferable. The overhead to establish and close a connection may be significant in this case. In the connection-oriented service, to achieve the above goal, at least 9 packets are exchanged between the client and the server; in connectionless service only two packets are exchanged. The connectionless service provides less delay; the connection-oriented service creates more delay. If delay is an important issue for the application, the connectionless service is preferred. One of the example of application where UDP used is the domain name system (DNS). A client-server application such as DNS uses the services of UDP because a client needs to send a short request to a server and to receive a quick response from it. The request and response can each fit in one user datagram. Since only one message is exchanged in each direction, the connectionless feature is not an issue; the client or server does not worry that messages are delivered out of order.

5.2 Transmission Control Protocol (TCP)

As with UDP, TCP provides process-to-process communication using port numbers. TCP, unlike UDP, is a stream-oriented protocol. In UDP, a process sends messages with predefined boundaries to UDP for delivery. UDP adds its own header to each of these messages and delivers it to IP for transmission. Each message from the process is called a user datagram, and becomes, eventually, one IP datagram. Neither IP nor UDP recognizes any relationship between the datagrams. At the transport layer, TCP groups a number of bytes together into a packet called a segment. TCP adds a header to each segment (for control purposes) and delivers the segment to the IP layer for transmission. The segments are encapsulated in an IP datagram and transmitted. To ensure reliability, TCP uses flow and error control extensively which makes the process slower than UDP.

Features of TCP

To provide the services mentioned in the previous section, TCP has several features that are briefly summarized next.

- ***Numbering System***
  Although the TCP software keeps track of the segments being transmitted or received, there is no field for a segment number value in the segment header. Instead, there are two fields called the sequence number and the acknowledgment number. These two fields refer to a byte number and not a segment number.

  1. *Byte Number*
     TCP numbers all data bytes (octets) that are transmitted in a connection. Numbering is independent in each direction. When TCP receives bytes of data from a process, TCP stores them in the sending buffer and numbers them. The numbering does not necessarily start from 0. Instead, TCP chooses an arbitrary number between 0 and 232− 1 for the number of the first byte. For example, if the number happens to be 1,057 and the total data to be sent is 6,000 bytes, the bytes are numbered from 1,057 to 7,056. The byte numbering is used for flow and error control.
  2. *Sequence Number*

     After the bytes have been numbered, TCP assigns a sequence number to each segment that is being sent. The sequence number for each segment is the number of the first byte of data carried in that segment. When a segment carries a combination of data and control information (piggybacking), it uses a sequence number. If a segment does not carry user data, it does not logically define a sequence number. The field is there, but the value is not valid. However, some segments, when carrying only control information, need a sequence number to allow an acknowledgment from the receiver. These segments are used for connection establishment, termination, or abortion. Each of these segments consume one sequence number as though it carries one byte, but there are no actual data. We will elaborate on this issue when we discuss connections.

  3. *Acknowledgment Number*

     Communication in TCP is full duplex; when a connection is established, both parties can send and receive data at the same time. Each party numbers the bytes, usually with a different starting byte number. The sequence number in each direction shows the number of the first byte carried by the segment. Each party also uses an acknowledgment number to confirm the bytes it has received. However, the acknowledgment number defines the number of the next byte that the party expects to receive. In addition, the acknowledgment number is cumulative, which means that the party takes the number of the last byte that it has received, safe and sound, adds 1 to it, and announces this sum as the acknowledgment number. The term cumulative here means that if a party uses 5,643 as an acknowledgment number, it has received all bytes from the beginning up to 5,642. Note that this does not

mean that the party has received 5,642 bytes because the first byte number does not have to start from 0.

- ***Flow Control***

  TCP, unlike UDP, provides flow control. The sending TCP controls how much data can be accepted from the sending process; the receiving TCP controls how much data can to be sent by the sending TCP. This is done to prevent the receiver from being overwhelmed with data. The numbering system allows TCP to use a byte-oriented flow control.

- ***Error Control***

  To provide reliable service, TCP implements an error control mechanism. Although error control considers a segment as the unit of data for error detection (loss or corrupted segments), error control is byte-oriented.

- ***Congestion Control***

  TCP, unlike UDP, takes into account congestion in the network. The amount of data sent by a sender is not only controlled by the receiver (flow control), but is also determined by the level of congestion, if any, in the network.

TCP Segment

Before discussing TCP in more detail, let us discuss the TCP packets themselves. A data unit in TCP is called a segment. The format of a segment is shown in Fig. 5.1. The segment consists of a header of 20 to 60 bytes, followed by data from the application program. The header is 20 bytes if there are no options and up to 60 bytes if it contains options.

- Source port address

  This is a 16-bit field that defines the port number of the application program in the host that is sending the segment. This serves the same purpose as the source port address in the UDP header.

- Destination port address

  This is a 16-bit field that defines the port number of the application program in the host that is receiving the segment. This serves the same purpose as the destination port address in the UDP header.

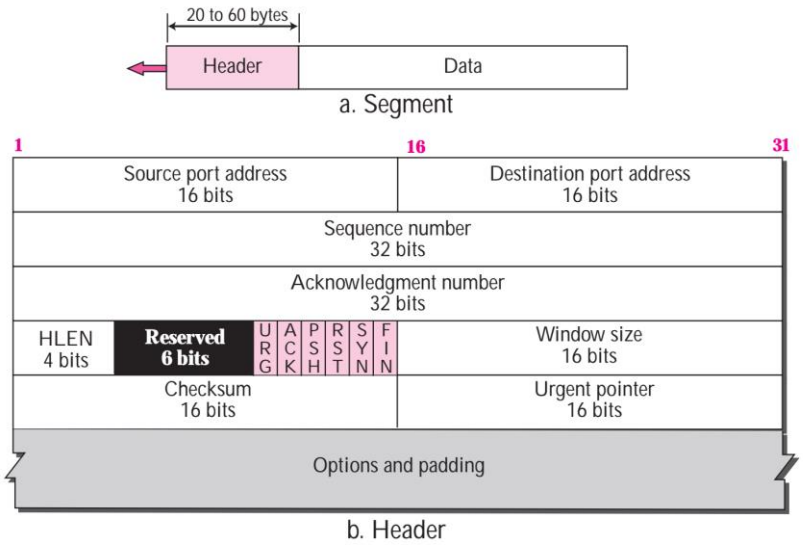

Figure 5.1 TCP Segment

- Sequence number

  This 32-bit field defines the number assigned to the first byte of data contained in this segment. As we said before, TCP is a stream transport protocol. To ensure connectivity, each byte to be transmitted is numbered. The sequence number tells the destination which byte in this sequence is the first byte in the segment. During connection establishment (discussed later) each party uses a random number generator to create an initial sequence number (ISN), which is usually different in each direction.

- Acknowledgment number

  This 32-bit field defines the byte number that the receiver of the segment is expecting to receive from the other party. If the receiver of the segment has successfully received byte number x from the other party, it returns x + 1 as the acknowledgment number. Acknowledgment and data can be piggybacked together.

- Header length

  This 4-bit field indicates the number of 4-byte words in the TCP header. The length of the header can be between 20 and 60 bytes. Therefore, the value of this field is always between 5 (5 × 4 = 20) and 15 (15× 4= 60).

- Reserved

  This is a 6-bit field reserved for future use.

- Control

  This field defines 6 different control bits or flags. One or more of these bits can be set at a time. These bits enable flow control, connection establishment and termination, connection abortion, and the mode of data transfer in TCP.

  URG: Urgent Bit: When set to 1, indicates that the priority data transfer feature has been invoked for this segment, and that the Urgent Pointer field is valid.

ACK: Acknowledgement Bit: When set to 1, indicates that this segment is carrying an acknowledgement, and the value of the Acknowledgement Number field is valid and carrying the next sequence expected from the destination of this segment.

PSH: Push Bit: The sender of this segment is using the TCP push feature, requesting that the data in this segment be immediately pushed to the application on the receiving device.

RST: Reset Bit: The sender has encountered a problem and wants to reset the connection

SYN: Synchronize Bit: This segment is a request to the synchronize sequence number and establish a connection; the Sequence Number field contains the initial sequence number (ISN) of the sender of the segment.

FIN: Finsh Bit: The sender of the segment is requesting that the connection be closed.

- Window size

  This field defines the window size of the sending TCP in bytes. Note that the length of this field is 16 bits, which means that the maximum size of the window is 65,535 bytes. This value is normally referred to as the receiving window (rwnd) and is determined by the receiver. The sender must obey the dictation of the receiver in this case.

- Checksum

  This 16-bit field contains the checksum.

Three-way handshaking

The connection establishment in TCP is called three-way handshaking. In our example, an application program, called the client, wants to make a connection with another application program, called the server, using TCP as the transport layer protocol.

The process starts with the server. The server program tells its TCP that it is ready to accept a connection. This request is called a passive open. Although the server TCP is ready to accept a connection from any machine in the world, it cannot make the connection itself.

The client program issues a request for an active open. A client that wishes to connect to an open server tells its TCP to connect to a particular server. TCP can now start the three-way handshaking process as shown in Fig. 5.2.

To show the process we use time lines. Each segment has values for all its header fields and perhaps for some of its option fields too. However, we show only the few fields necessary to understand each phase. We show the sequence number, the acknowledgment number, the control flags (only those that are set), and window size if relevant.

The three steps in this phase are as follows.

- The client sends the first segment, a SYN segment, in which only the SYN flag is set. This segment is for synchronization of sequence numbers. The client in our example chooses a random number as the first sequence number and sends this number to the server. This sequence number is called the initial sequence number (ISN). Note that this segment does not contain an acknowledgment number. It does not define the window size either; a window size definition makes sense only when a segment includes an acknowledgment. Note that the SYN segment is a control segment and carries no data. However, it consumes one sequence number. When the data transfer starts, the ISN is incremented by 1. We can

say that the SYN segment carries no real data, but we can think of it as containing one imaginary byte.

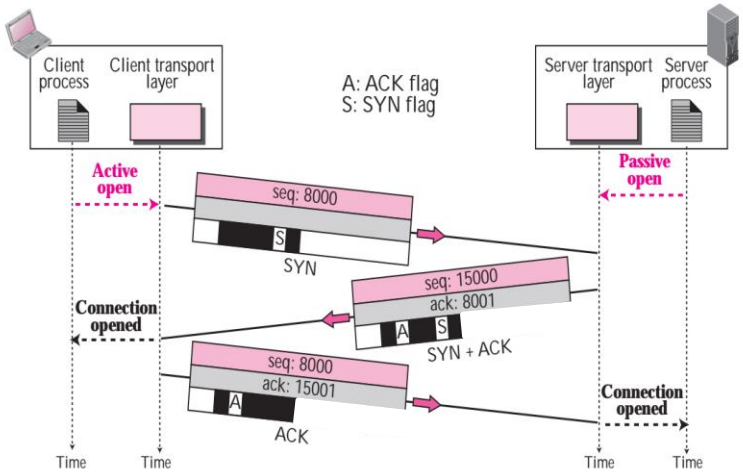

Figure 5.2 Illustration of TCP three-way hand-shaking

- The server sends the second segment, a SYN + ACK segment with two flag bits set: SYN and ACK. This segment has a dual purpose. First, it is a SYN segment for communication in the other direction. The server uses this segment to initialize a sequence number for numbering the bytes sent from the server to the client. The server also acknowledges the receipt of the SYN segment from the client by setting the ACK flag and displaying the next sequence number it expects to receive from the client.
- The client sends the third segment. This is just an ACK segment. It acknowledges the receipt of the second segment with the ACK flag and acknowledgment number field. Note that the sequence number in this segment is the same as the one in the SYN segment; the ACK segment does not consume any sequence numbers. The client must also define the server window size. Some implementations allow this third segment in the connection phase to carry the first chunk of data from the client. In this case, the third segment must have a new sequence number showing the byte number of the first byte in the data. In general, the third segment usually does not carry data and consumes no sequence numbers.

Interesting scenarios for reliable data transfer

TCP provides reliable data transfer. To get a good a clear idea of how the reliability is ensured, let's now walk through a few simple scenarios.

- ***Lost Acknowledgement***

  Fig. 5.3 depicts the first scenario, in which Host A sends one segment to Host B. Suppose that this segment has sequence number 92 and contains 8 bytes of data. After sending this segment, Host A waits for a segment from B with acknowledgment number 100. Although the segment from A is received at B, the acknowledgment from B to A gets lost. In this case, the timeout event occurs, and Host A retransmits the same segment. Of course, when Host B receives the

retransmission, it observes from the sequence number that the segment contains data that has already been received. Thus, TCP in Host B will discard the bytes in the retransmitted segment.

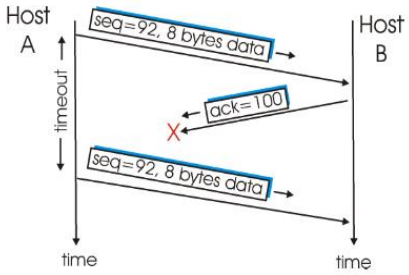

Figure 5.3 Retransmission due to a lost acknowledgment

- ***Acknowledgement arrives before timeout***

  In a second scenario, shown in Fig. 5.4, Host A sends two segments back to back. The first segment has sequence number 92 and 8 bytes of data, and the second segment has sequence number 100 and 20 bytes of data. Suppose that both segments arrive intact at B, and B sends two separate acknowledgments for each of these segments. The first of these acknowledgments has acknowledgment number 100; the second has acknowledgment number 120. Suppose now that neither of the acknowledgments arrives at Host A before the timeout. When the timeout event occurs, Host A resends the first segment with sequence number 92 and restarts the timer. As long as the ACK for the second segment arrives before the new timeout, the second segment will not be retransmitted.

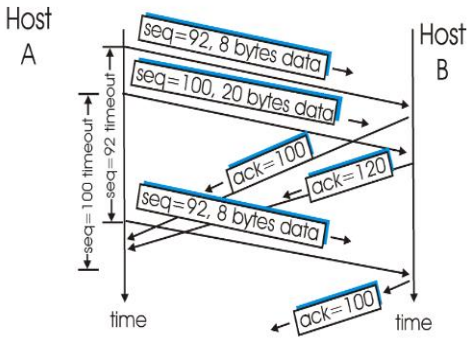

Figure 5.4 Segment is not retransmitted because its acknowledgment arrives before the timeout.

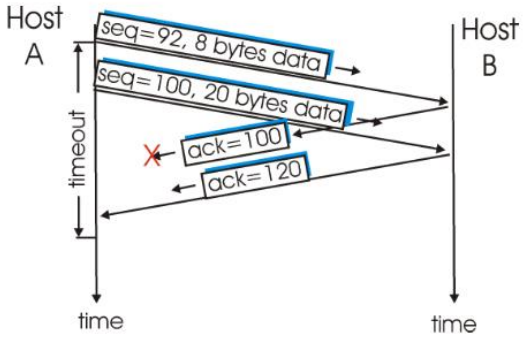

Figure 5.5 A cumulative acknowledgment avoids retransmission of first segment

- ***Cumulative Acknowledgement***

  In a third and final scenario, suppose Host A sends the two segments, exactly as in the second example. The acknowledgment of the first segment is lost in the network, but just before the timeout event, Host A receives an acknowledgment with acknowledgment number 120. Host A therefore knows that Host B has received everything up through byte 119; so Host A does not resend either of the two segments. This scenario is illustrated in Fig. 5.5.

5.3 Applications

Applications of UDP

UDP is used in services which need to meet any of the following requirements

- Applications that can tolerate some data loss, but require little or no delay
- Applications with simple request and reply transactions
- Unidirectional communications where reliability is not required or can be handled by the application
- Examples
  1. All audio and video transmission (VoIP, IPTV)
  2. DHCP
  3. DNS (may also use TCP)
  4. SNMP
  5. TFTP (has own control mechanism, that TCP is not required)

Applications of TCP

TCP is used in those applications that require reliable data transfer and can tolerate delay

Example

1. HTTP
2. FTP
3. Telnet
4. SMTP