...3...
Transport Layer Protocols

Learning Outcomes...

- ❑ Explain the mechanism of process-to-process delivery.
- ❑ Compare multiplexing and demultiplexing
- ❑ Explain functioning of TCP/UDP protocols with example.
- ❑ Explain various congestion control methods at Transport layer.
- ❑ Describe the functioning of TLS.
- ❑ Describe the functioning of SCTP.

3.0 INTRODUCTION

- The transport layer is the fourth layer of the OSI reference model, which provides communication services between the computers connected in the network.
- The task of transport layer protocols is to deliver data in the right order and form enabling the application to use them.
- A transport layer protocol provides for logical communication between application processes running on different hosts. The original TCP/IP protocol suite specifies two protocols for the transport layer namely, UDP and TCP.
- A new transport layer protocol SCTP is recently developed for signaling message transport over IP networks and it is being actively studied to explore its applicability to new areas.
- TCP is a transport layer protocol that provides for a connection-oriented, reliable service to applications.
- A connection-oriented protocol establishes a connection, manages the data transfer and terminates the connection.
- Thus TCP, provides end-to-end connectivity using a virtual connection between the end points; therefore all data segments are sent over this virtual path. In addition, TCP uses flow, error and congestion control to manage the transfer of data between the two end points.
- UDP is connectionless and unreliable transport layer protocol. It doesn't require making a connection with the host to exchange data. Since, UDP is unreliable protocol, there is no mechanism for ensuring that data sent is received.
- Stream Control Transmission Protocol (SCTP) is a reliable, message-oriented transport layer protocol. SCTP provides some of the features of both UDP and TCP, (it is message-oriented like UDP and ensures reliable, in-sequence transport of messages with congestion control like TCP).
- In short, SCTP has mixed/combined features of TCP and UDP. SCTP is especially designed for internet applications.
- Fig. 3.1 shows position of UDP, TCP and SCTP in the TCP/IP protocol suite.

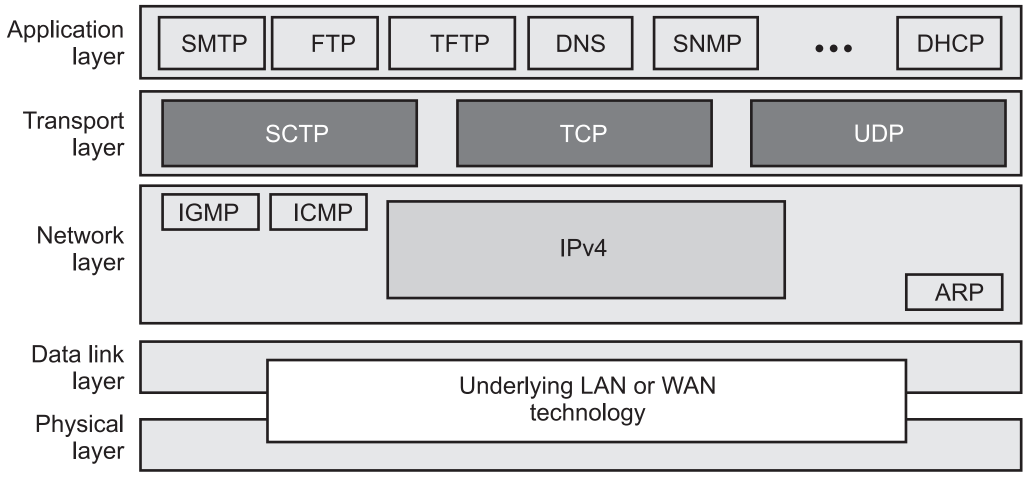


Fig. 3.1

- The transport layer is responsible for providing services to the application layer, it receives services from the network layer.
- Transport layer, (layer 4) in the TCP/IP model is responsible for process-to-process communication, flow control, congestion control and so on.
- The original TCP/IP protocol suite specifies two protocols for the transport layer namely, UDP and TCP.
- A new transport layer protocol SCTP is recently developed for signaling message transport over IP networks.

3.1 PROCESS TO PROCESS DELIVERY

- The transport layer is responsible for process-to-process delivery, ensuring data is transferred between applications (processes) running on different hosts.
- The transport layer's primary function is to deliver data from one application (process) to another, regardless of the underlying network protocols.
- The first duty of a transport layer is to provide process-to-process communication. A process is an application program running on the host and uses the services of the transport layer.
- Before we discuss how process-to-process communication can be accomplished, we need to understand the difference between host-to-host communication and process-to-process communication.
- The data link layer is responsible for delivery of frames between two neighboring nodes over a link. This is called node-to-node communication.
- The network layer is responsible for delivery of datagram between two hosts. This is called host-to-host communication.
- Real communication takes place between two processes (application programs) in a network. This is called process-to process communication.
- The transport layer is responsible for process-to-process communication, the delivery of a packet, part of a message, from one process to another.
- A network layer protocol can deliver the message only to the destination computer. However, this is an incomplete delivery.
- The message still needs to be handed to the correct process. This is where a transport layer protocol takes over.
- A transport layer protocol is responsible for delivery of the message to the appropriate process.
- Fig. 3.2 shows the domains of a network layer and a transport layer.

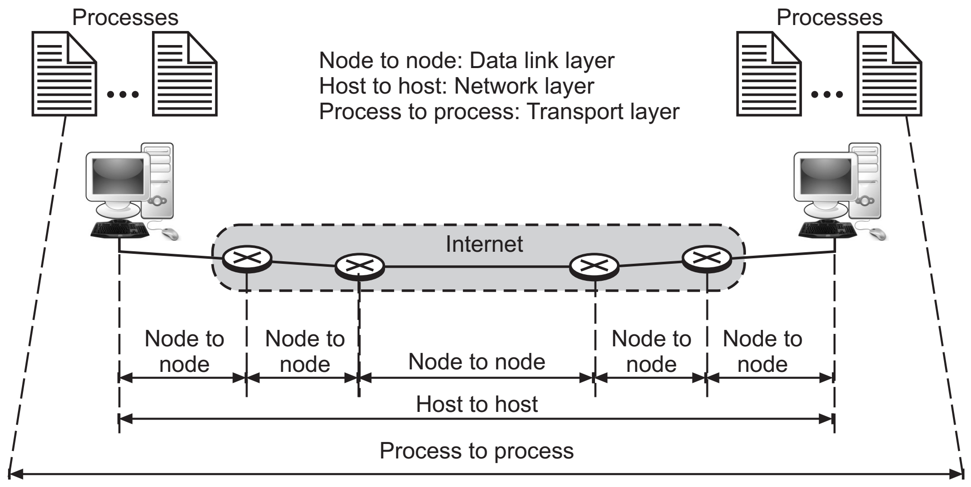


Fig. 3.2: Network Layer versus Transport Layer

3.1.1 Client/Server Paradigm

- Although there are several ways to achieve process-to-process communication, the most common one is through the client/server paradigm. A process on the local host, called a client, needs services from a process usually on the remote host, called a server.
- Both processes (client and server) have the same name. For example, to get the day and time from a remote machine, we need a Daytime client process running on the local host and a Daytime server process running on a remote machine.
- For communication, we must define the following:

  1. Local host
  2. Local process
  3. Remote host
  4. Remote process.
- Clients and servers connect across a Computer Network on separate hardware most of the time. However, they may share a system.
- A server host is a computer that executes one or more server applications that share resources with clients. A client typically does not share any of its resources but instead asks a server for material or services.
- As a result, clients start communication sessions with servers, waiting for incoming requests. Email or network printing and the World Wide Web (WWW) are computer applications that employ the client-server concept.

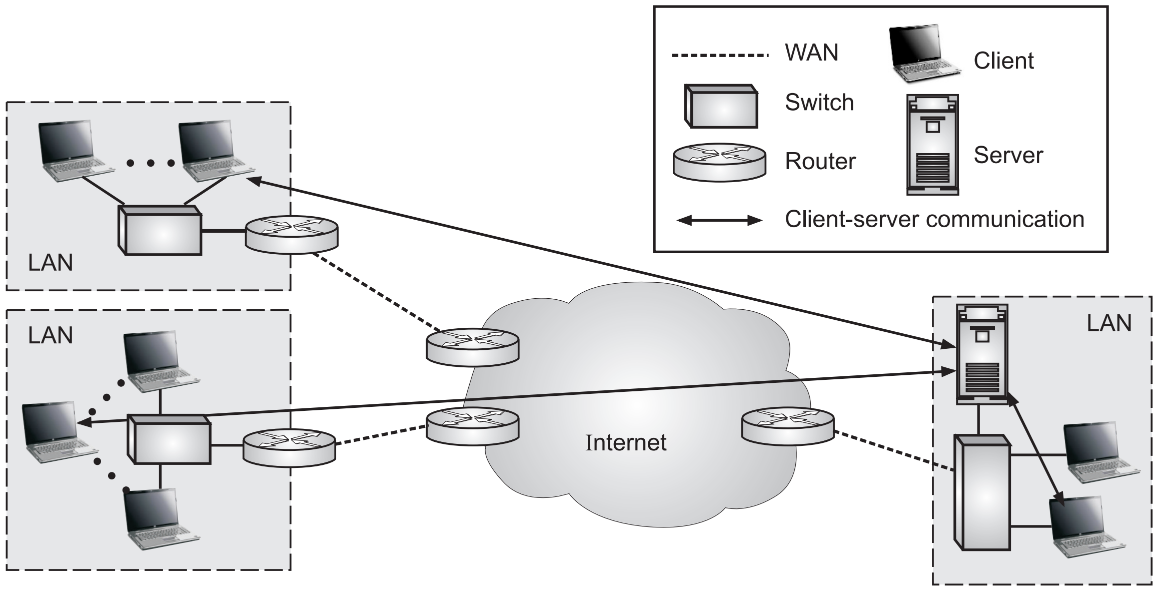


Fig. 3.3: Client/Server Paradigm

- In the client-server paradigm at the transport layer, a client process requests services from a server process, using protocols like TCP/IP to establish and maintain communication, ensuring reliable and ordered delivery of data between the two.
- Fig. 3.3 shows an example of a client-server communication in which three clients communicate with one server to receive the services provided by this server.

Roles of Client and Server:

- o **Client:** The client initiates the connection and sends requests to the server.
- o **Server:** The server listens for connections and processes requests from clients.

3.1.2 Multiplexing and Demultiplexing

- Multiplexing and demultiplexing are the two very important services that are performed by the transport layer.
- The transport layer at the source performs multiplexing while the transport layer at the destination performs demultiplexing as shown in Fig. 3.4.
- Whenever an entity accepts items from more than one source, it is referred to as multiplexing (many to one); whenever an entity delivers items to more than one source, it is referred to as demultiplexing (one to many).
- Fig. 3.4 shows communication between a client and two servers. Three client processes are running at the client site namely, P1, P2, and P3.
- The processes P1 and P3 need to send requests to the corresponding server process running in a server.

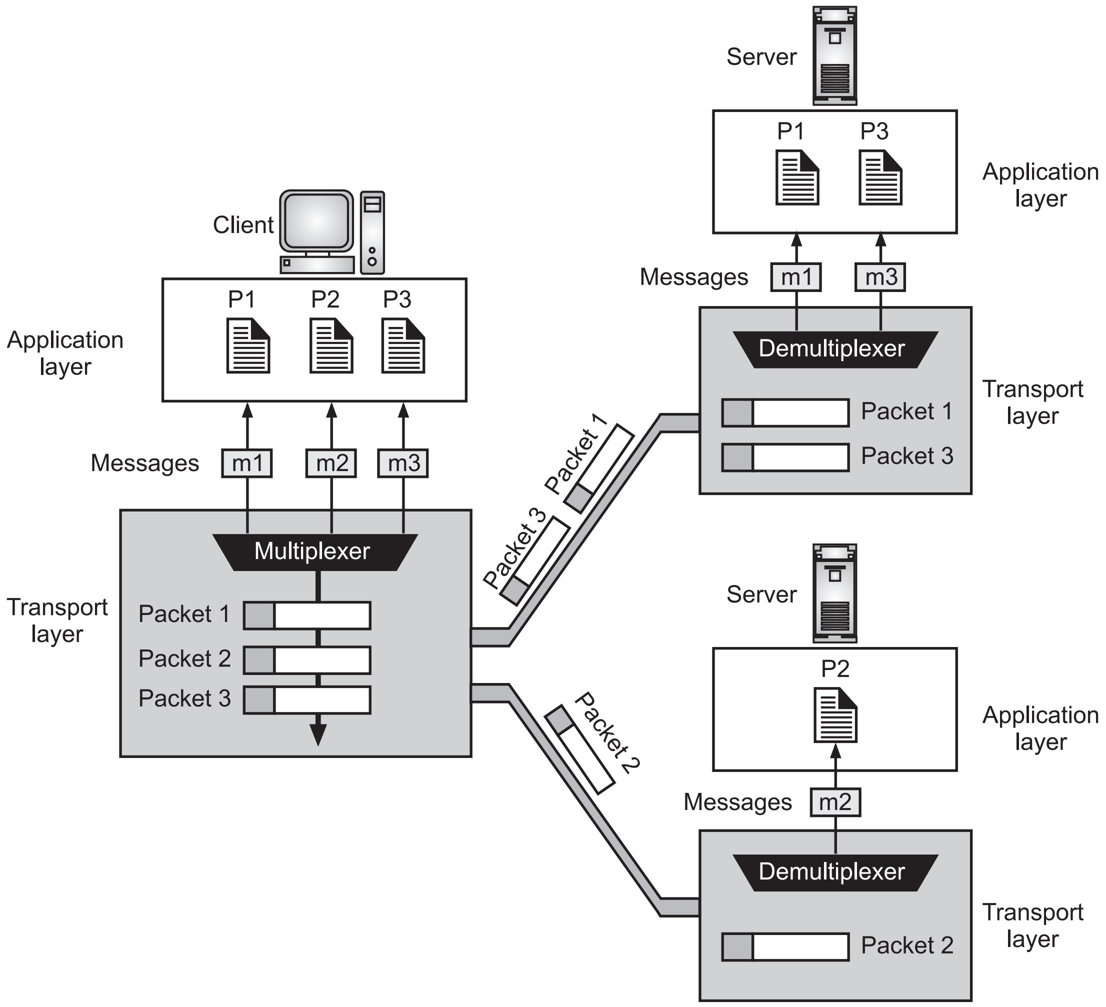


Fig. 3.4: Multiplexing and Demultiplexing

- The client process P2 needs to send a request to the corresponding server process running at another server.
- The transport layer at the client site accepts three messages from the three processes and creates three packets. It acts as a multiplexer.
- The packets 1 and 3 use the same logical channel to reach the transport layer of the first server.
- When they arrive at the server, the transport layer does the job of a multiplexer and distributes the messages to two different processes.
- The transport layer at the second server receives packet 2 and delivers it to the corresponding process.

3.1.3 Connectionless vs. Connection-Oriented Service [S-22]

- A transport layer provides two types of services namely, connectionless and connection-oriented.

1. **Connectionless Service:**

- In a connectionless service, the packets are sent from one machine to another without connection establishment or connection release.
- The packets may arrive without order or they may be lost or delayed. Packets are not numbered. No acknowledgement is processed.
- In a connectionless service, the source process (application program) needs to divide its message into chunks of data of the size acceptable by the transport layer and deliver them to the transport layer one by one.
- The transport layer treats each chunk as a single unit without any relation between the chunks. When a chunk arrives from the application layer, the transport layer encapsulates it in a packet and sends it.
- To show the independence of packets, assume that a client process has three chunks of messages to send a server process.
- The chunks are handed over to the connectionless transport protocol in order.
- However, since there is no dependency between the packets at the transport layer, the packets may arrive out of order at the destination and will be delivered out of order to the server process.
- In Fig. 3.5, we have shown the movement of packets using a timeline, but we have assumed that the delivery of the process to the transport layer and vice versa are instantaneous.

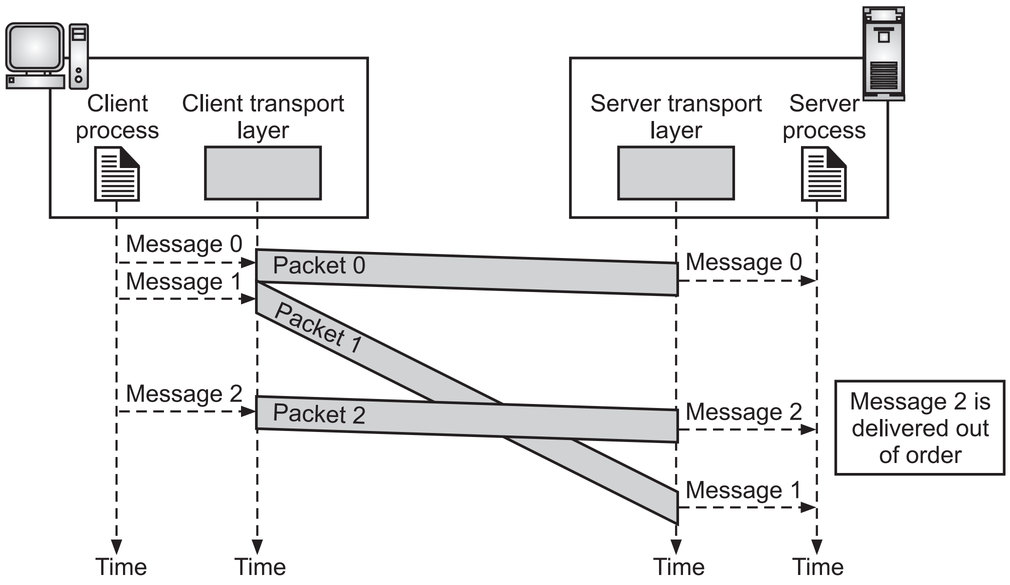


Fig. 3.5: Connectionless Service

- The Fig. 3.5, shows that at the client site, the three chunks of messages are delivered to the client transport layer in order (1, 2, and 3).
- Because of the extra delay in transportation of the second packet, the delivery of messages at the server is not in order (1, 3, 2).
- If these three chunks (1, 2, and 3) of data belong to the same message, the server process may have received a strange message.
- UDP is a transport layer's connectionless protocol.

2. Connection Oriented Service:

- In a connection-oriented service, connection is established first and then data are transferred in between sender and receiver. After the end of data transfer, connection is released.
- In a connection-oriented service, the client and the server first need to establish a connection between themselves.
- The data exchange can only happen after the connection establishment. After data exchange, the connection needs to be teared down as shown in Fig. 3.6.

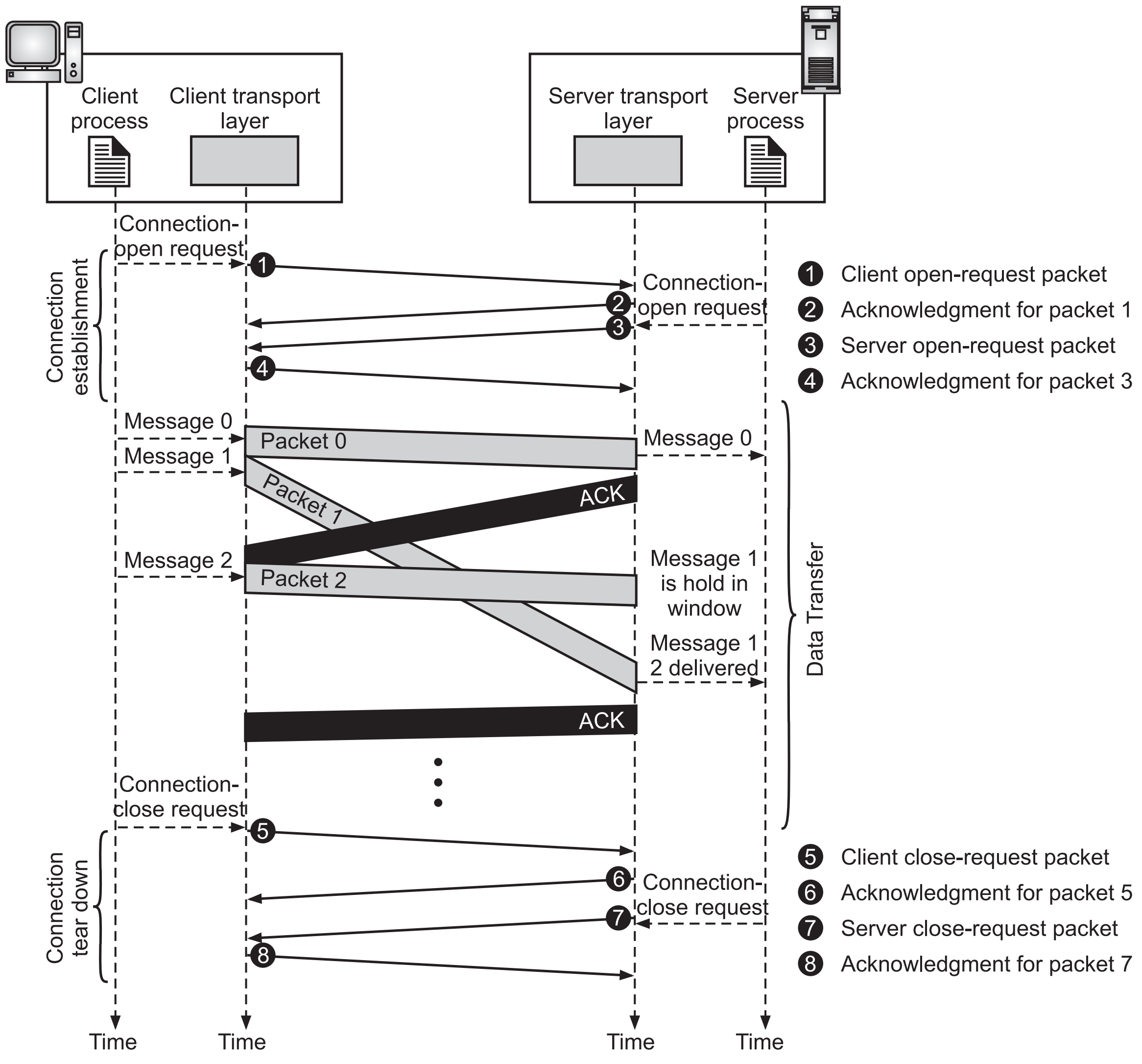


Fig. 3.6: Connection-oriented Service

- As we mentioned before, the connection-oriented service at the transport layer is different from the same service at the network layer.
- In the network layer, connection-oriented service means a coordination between the two end hosts and all the routers in between.

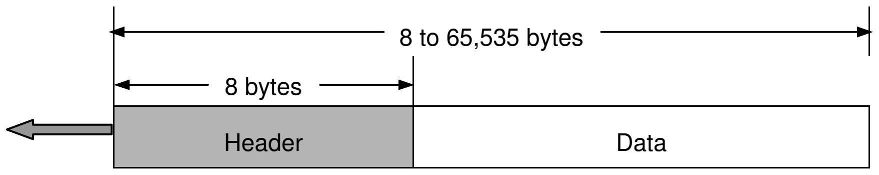
- At the transport layer, connection-oriented service involves only the two hosts; the service is end to end.
- This means that we should be able to make a connection-oriented protocol over either a connectionless or connection-oriented protocol.
- Fig. 3.6 shows the connection establishment, data transfer, and teardown phases in a connection-oriented service at the transport layer.
- Note that most protocols combine the third and fourth packets in the connection establishment phase into one packet.
- TCP and SCTP, these two transport layer protocols are connection oriented. **[S-22]**

3.2 USER DATAGRAM PROTOCOL (UDP) [S-22, W-22]

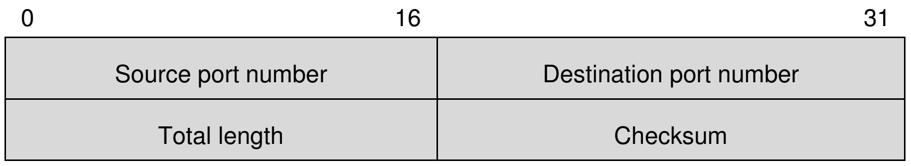
- The User Datagram Protocol (UDP) is the simplest Transport Layer communication protocol in the TCP/IP protocol suite and serves as the intermediary between the application programs and the network operations.
- The UDP was designed by David P. Reed in 1980 and formally defined in RFC 768 standard.
- UDP is a connectionless, unreliable Transport Layer protocol.
- UDP 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 powerless protocol. But it is a very simple protocol, using minimum overhead. If a process wants to send a small message and does not care much about reliability, it can use UDP.
- UDP is stateless protocol. It is a suitable protocol for streaming applications such as VoIP, multimedia streaming.

3.2.1 User Datagram

- UDP packets called user datagrams, have a fixed-size header of 8 bytes. Fig. 3.7 shows the format of a user datagram of UDP.



(a) UDP User Datagram



(b) Header Format of UDP

Fig. 3.7

- The fields in UDP datagram are as follows:

1. **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, 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, the port number, in most cases, is a well-known port number.

2. **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, the port number, in most cases, is a well-known port number.
- If the destination host is the client, the port number, in most cases, is an ephemeral port number.

3. **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. 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.

UDP Length = IP Length – IP header's Length

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

3.2.2 UDP Services [W-22, S-23, W-23, W-24]

- The general UDP services are Process to Process Communication, Connectionless Services, Flow Control, Error Control, Congestion Control, Encapsulation and Decapsulation, Queuing, Multiplexing and Demultiplexing.

1. **Process to Process Communication:**
- UDP provides a process to process communication using sockets, a combination of IP addresses and port numbers.
- Several well-known port numbers used by UDP are shown below:

Port No	Protocol	Description
7	Echo	Resends a received datagram back to the sender.
9	Discard	Discards a received datagram.
11	Users	Shows active users.
13	Daytime	Gives the date and time.
17	Quote	Gives a quote of the day.
19	Chargen	Gives a string of characters.
53	Nameserver	Shows Domain Name Service (DNS).
67	BOOTPs	Server port to download bootstrap information.
68	BOOTCPc	Client port to download bootstrap information.
69	TFTP	Trivial File Transfer Protocol (TFTP).
111	RPC	Remote Procedure Call.
123	NTP	Network Time Protocol.
161	SNMP	Simple Network Management Protocol.
162	SNMP	Simple Network Management Protocol.

2. **Connectionless Services:**
- UDP provides a connectionless service. This means that each user datagram sent by UDP is an independent datagram.

- In UDP there is no relationship between the different user datagrams even if they are coming from the same source process and going to the same destination program.
- The user datagrams are not numbered. There is no connection establishment and no connection termination as is the case for TCP.
- Each user datagram can travel on a different path. One of the ramifications of being connectionless is that the process that uses UDP cannot send a stream of data to UDP and expect UDP to chop them into different related user datagrams.
- Instead each request must be small enough to fit into one user datagram. Only those processes sending short messages, messages less than 65,507 bytes (65,535 minus 8 bytes for the UDP header and minus 20 bytes for the IP header), can use UDP.

3. Flow Control:

- UDP is a very simple protocol. There is no flow control, and hence no window mechanism. The receiver may overflow with incoming messages.
- The lack of flow control means that the process using UDP should provide for this service, if needed.

4. Error Control:

- There is no error control mechanism in UDP except for the checksum.
- Sender does not know if a message has been lost or duplicated. When the receiver detects an error through the checksum, the user datagram is silently discarded.
- The lack of error control means that the process using UDP should provide for this service if needed.

Checksum:

- UDP provides checksums for data integrity, and port numbers for addressing different functions at the source and destination of the datagram.
- The UDP checksum calculation is different from IP. UDP's checksum includes three sections namely, 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.
- If the checksum does not include the pseudo header, a user datagram may arrive safe and sound. If the IP header is corrupted, it may be delivered to the wrong host.
- The protocol field is added to confirm that the packet belongs to UDP. The value of protocol 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.

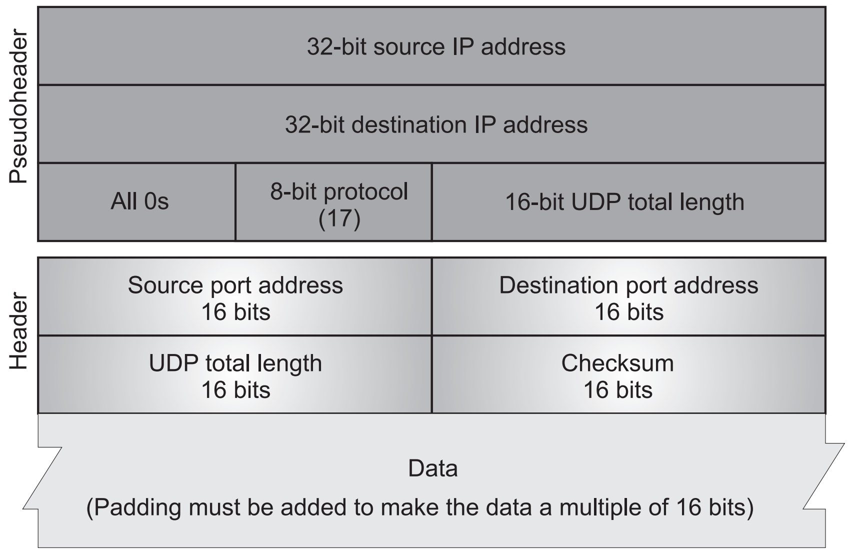

Pseudoheader
32-bit source IP address
32-bit destination IP address
All 0s
8-bit protocol (17)
16-bit UDP total length
Header
Source port address 16 bits
Destination port address 16 bits
UDP total length 16 bits
Checksum 16 bits
Data
(Padding must be added to make the data a multiple of 16 bits)

Fig. 3.8: Pseudo Header for Checksum Calculation

- The calculation of checksum and its inclusion in a user datagram are optional. If checksum is not calculated, the field is filled with 1s.

  Example: Fig. 3.9 shows the checksum calculation for a very small user datagram with only 7 bytes of data. Since the data is odd, padding is added for checksum calculation. The pseudo header as well as the padding will be dropped when the user datagram is delivered to IP.

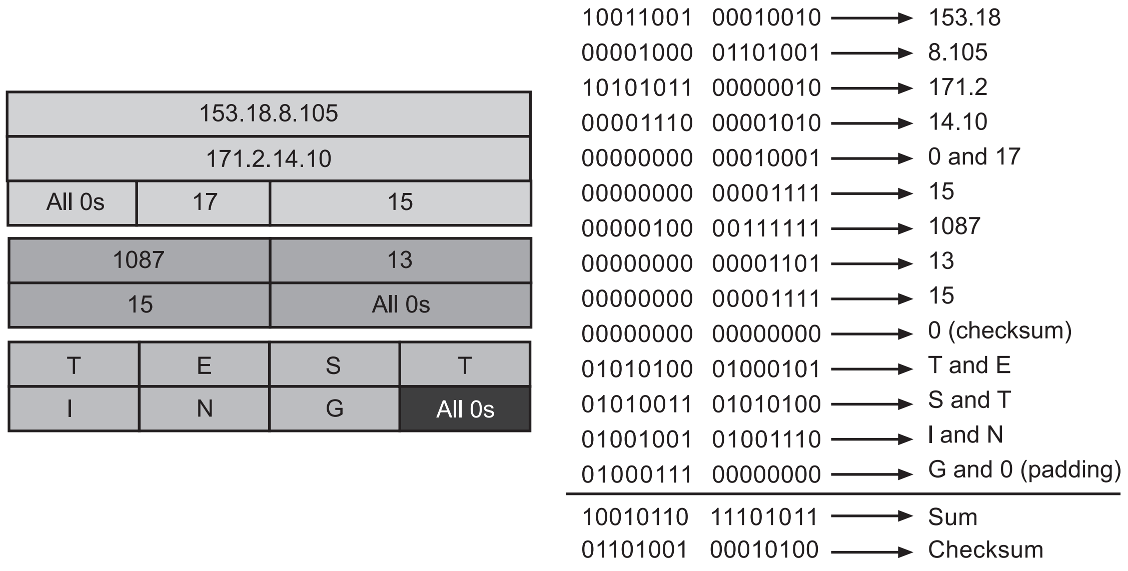


Fig. 3.9: Checksum Calculation of a Simple UDP User Datagram

5. **Multiplexing and Demultiplexing:**

- In a host running a TCP/IP protocol suite, there is only one UDP but possibly several processes that may want to use the services of UDP. To handle this situation, UDP multiplexers and demultiplexers (See Fig. 3.10).

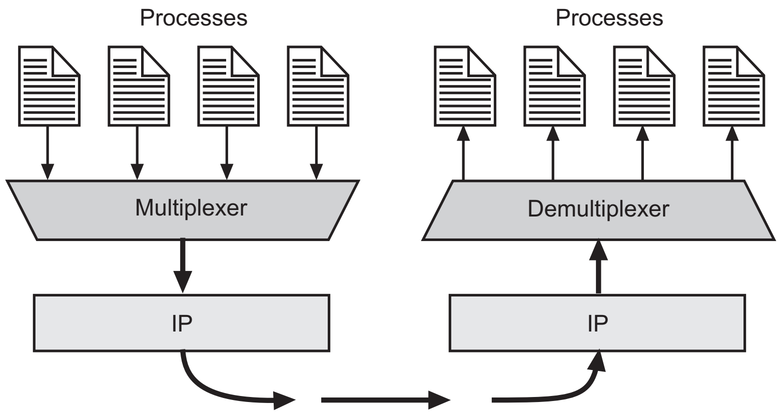


Fig. 3.10: Multiplexing and Demultiplexing

- UDP uses multiplexing to handle outgoing user datagrams from multiple processes on one host. UDP uses demultiplexing to handle incoming user datagrams that go to different processes on the same host.
  - **Multiplexing:** At the sender site, there may be several processes that need to send user datagrams. However, there is only one UDP. This is a many-to-one relationship and requires multiplexing. UDP accepts messages from different processes, differentiated by their assigned port numbers. After adding the header, UDP passes the user datagram to IP.
  - **Demultiplexing:** At the receiver site, there is only one UDP. However, we may have many processes that can receive user datagrams. This is a one-to-many relationship and requires

demultiplexing. UDP receives user datagrams from IP. After error checking and dropping of the header, UDP delivers each message to the appropriate process based on the port numbers.

6. **Encapsulation and Decapsulation:**

- To send a message from one process to another, the UDP protocol encapsulates and decapsulates messages.

7. **Queuing:**

- In UDP, queues are associated with ports. At the client site, when a process starts, it requests a port number from the operating system.
- Some implementations create both an incoming and an outgoing queue associated with each process. Other implementations create only an incoming queue associated with each process.

8. **Congestion Control:**

- UDP does not provide congestion control because of it is a connectionless protocol. UDP assumes that the packets sent are small and sporadic, and cannot create congestion in the network.
- This UDP assumption may or may not be true today when UDP is used for real-time transfer of audio and video.

3.2.3 Features of UDP

- Features of UDP includes:
  1. **Connectionless Service:** UDP provides connectionless service. In UDP each packet is independent from other packets sent by the same application.
  2. **Lack of Congestion Control:** UDP does not provide congestion control and it does not create additional traffic in an error prone network.
  3. **Lack of Error Control:** UDP does not provide error control. So UDP provides unreliable service.
  4. **Transaction-oriented:** UDP is transaction-oriented, suitable for simple query-response protocols such as the Domain Name System (DNS).
  5. **Datagram:** UDP provides datagrams, suitable for modeling other protocols such as IP tunneling or Remote Procedure Call (RPC) and the Network File System (NFS). An UDP datagram is used in Network File System (NFS), DNS, SNMP, TFTP etc.
  6. **Simple:** UDP is a simple, datagram-oriented, transport-layer protocol. UDP is simple, suitable for bootstrapping or other purposes without a full protocol stack, such as the DHCP and Trivial File Transfer Protocol (TFTP).
  7. **Stateless:** UDP is stateless, suitable for very large numbers of clients, such as in streaming media applications such as IPTV.
  8. **Lack of Retransmission Delays:** The lack of retransmission delays makes UDP suitable for real-time applications such as Voice over IP, online games, and many protocols using Real Time Streaming Protocol.
  9. **Support Multicast:** Because it supports multicast, it is suitable for broadcast information such as in many kinds of service discovery and shared information such as Precision Time Protocol (PTP) and Routing Information Protocol (RIP).
  10. **Faster in Data Transfer:** UDP is a lightweight protocol for faster and simpler data transmissions.
  11. **Queuing:** UDP is simple and suitable for query-based communications. The Queues are associated with the ports in UDP.
  12. **Low Overhead:** UDP is designed to provide application processes with the ability to transfer data with a minimal overhead.

13. **Port:** UDP uses that concept of port, which allows to distinguish the different applications running on a machine. Besides the datagram and its data, a UDP message contains a source port number and destination port number.
14. **No Acknowledgment/Not Reliable:** In UDP data are transmitted with no acknowledgment of whether it is received or not. UDP is thus not as reliable as TCP. UDP does not guarantee ordered delivery of data.

3.2.4 UDP Applications [S-23, S-24]

- Used for simple request response communication when size of data is less and hence there is lesser concern about flow and error control.
- It is suitable protocol for multicasting as UDP supports packet switching. UDP is used for some routing update protocols like RIP (Routing Information Protocol).
- Normally used for real time applications which cannot tolerate uneven delays between sections of a received message.
- Following implementations uses UDP as a transport layer protocol:
  - NTP (Network Time Protocol)
  - DNS (Domain Name Service)
  - BOOTP, DHCP.
  - NNP (Network News Protocol)
  - Quote of the day protocol
  - TFTP, RTSP, RIP, OSPF.
- Application layer can do some of the tasks through UDP-
  - Trace Route
  - Record Route
  - Time stamp
- UDP takes datagram from Network Layer, attach its header and send it to the user. So, it works fast. Actually, UDP is null protocol if we remove checksum field.

3.3 TRANSMISSION CONTROL PROTOCOL (TCP) [S-22]

- TCP is a reliable connection-oriented, reliable protocol i.e., a connection is established between the sender and receiver before the data can be transmitted.
- TCP divides the data it receives from the upper layer into segments and tags a sequence number to each segment which is used at the receiving end for reordering of data.
- TCP lies between the application layer and the network layer, and serves as the intermediary between the application programs and the network operations.
- TCP is the most commonly used transport layer protocol. TCP is a connection-oriented, reliable, in-order transport protocol.

3.3.1 TCP Services [S-22]

- Following services offered by TCP:

1. **Process to Process Communication:**

- TCP provides process to process communication, i.e. the transfer of data takes place between individual processes executing on end systems. This is done using port numbers or port addresses.
- Port numbers are 16 bit long that help identify which process is sending or receiving data on a host.

- Following table shows some well-known port numbers used by TCP:

Port	Protocol	Description
7	Echo	Echoes a received datagram back to the sender.
9	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.
20 and 21	FTP	File Transfer Protocol (Data and Control).
23	TELNET	Terminal Network.
25	SMTP	Simple Mail Transfer Protocol.
53	DNS	Domain Name Server.
67	BOOTP	Bootstrap Protocol.
79	Finger	Finger.
80	HTTP	Hypertext Transfer Protocol.

2. **Stream Oriented/Stream Delivery Service:**

- TCP is a stream-oriented protocol means that the data is sent and received as a stream of bytes, (unlike UDP or IP that divides the bits into datagrams or packets).
- However, the network layer, that provides service for the TCP, sends packets of information not streams of bytes.
- Hence, TCP groups a number of bytes together into a segment and adds a header to each of these segments and then delivers these segments to the network layer.
- At the network layer, each of these segments are encapsulated in an IP packet for transmission. The TCP header has information that is required for control purpose.
- It may not be possible for sending and receiving process to produce and obtain data at same speed, therefore, TCP needs buffers for storage at sending and receiving ends.

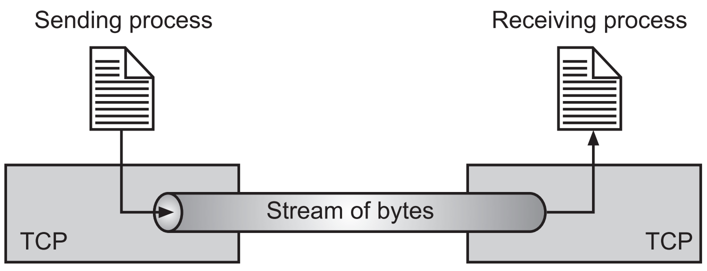


Fig. 3.11: Stream Delivery

3. **Full Duplex Communication Service:**

- This means that the communication can take place in both directions at the same time.

4. **Connection Oriented Service:**

- Unlike UDP, TCP provides connection-oriented service. It defines following three different phases:
  - Connection establishment.
  - Data transfer.
  - Connection termination.

  (Note: This is a logical connection, not physical).

5. **Reliability Service:**
- TCP is reliable as it uses checksum for error detection, attempts to recover lost or corrupted packets by re-transmission, acknowledgement policy and timers.
- It uses features like byte number and sequence number and acknowledgement number so as to ensure reliability. Also, it uses congestion control mechanisms.

6. **Multiplexing and Demultiplexing Service:**
- TCP does multiplexing and de-multiplexing at the sender and receiver ends respectively as a number of logical connections can be established between port numbers over a physical connection.

3.3.2 TCP Features [S-23, W-23]

- TCP has following several features:

1. **Numbering System:**
- TCP keep tracks of the segments being transmitted or received by assigning numbers.
  - **(i) Byte number assigned to data bytes to be transferred:** Byte number used for flow control and error control. All data bytes transmitted in each connection are numbered by the TCP. Numbering starts with a randomly generated number. Numbering is independent in each direction. TCP stores the received data bytes in a sending buffer and numbers them.
  - **(ii) Sequence number to segments:** TCP assigns a sequence number to each segment transmitted. Sequence number is the first byte number carried in that segment. Segment number is assigned when:
    - o Segment carries both data and control information (piggy backing).
    - o Segment without data, no sequence number.
  - **(iii) Acknowledgement Number to received segments:** It indicates number of the next byte that the receiver expects to receive. Acknowledge number is cumulative. Confirmation of received data bytes.

2. **Flow Control:** [S-23]
- TCP 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.

3. **Error Control:** [S-23]
- TCP implements an error control mechanism for reliable data transfer. Segments are checked for error detection.
- Although error control considers a segment as the unit of data for error detection (loss or corrupted segments), error control is byte-oriented.
- Error control includes detecting:
  - o Corrupted segments and Lost segments.
  - o Out-of-Order Segments.
  - o Duplicated Segments.

4. **Congestion Control:**
- TCP 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.

5. **Interoperability:**
- TCP has become the industry standard. It supports interoperability across networks.
- TCP framework used to develop complete range of computer communication standards.

3.3.3 TCP Segment [W-22, S-23, W-24]

- A packet in TCP is called a segment. TCP segment consists of data bytes to be sent and a header that is added to the data by TCP as shown in Fig. 3.12.

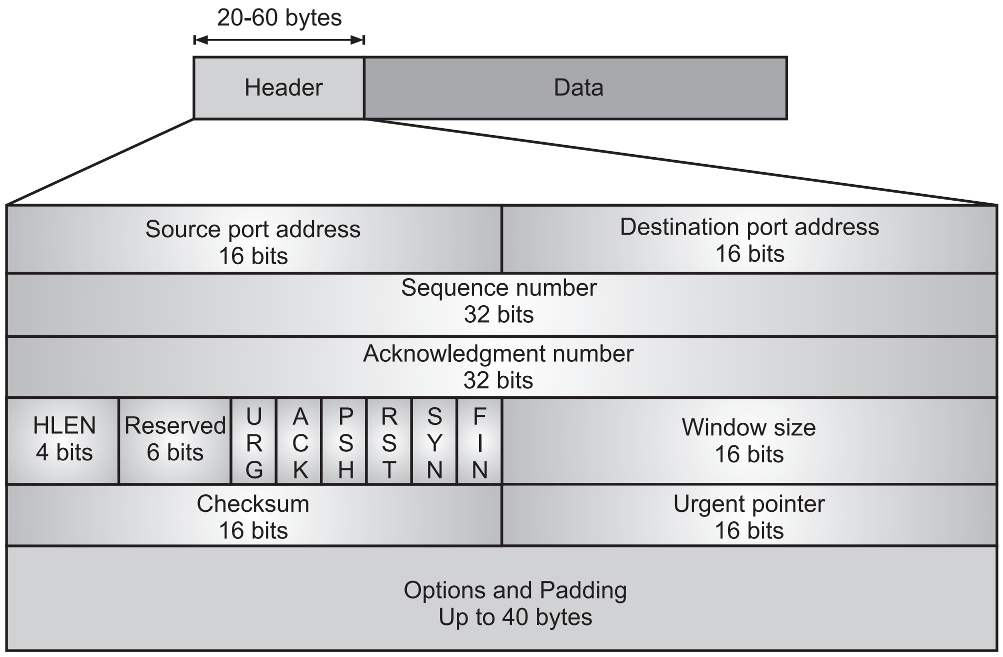


Fig. 3.12: TCP Segment Format

- The header of a TCP segment can range from 20-60 bytes. 40 bytes are for options. If there are no options, header is of 20 bytes else it can be of upmost 60 bytes.
- Header Fields in TCP Segment Structure are explained below:

1. **Source Port Address:** 16 bit field that holds the port address of the application that is sending the data segment.
2. **Destination Port Address:** 16 bit field that holds the port address of the application in the host that is receiving the data segment.
3. **Sequence Number:** 32 bit field that holds the sequence number, i.e. the byte number of the first byte that is sent in that particular segment. It is used to reassemble the message at the receiving end if the segments are received out of order.
4. **Acknowledgement Number:** 32 bit field that holds the acknowledgement number, i.e. the byte number that the receiver expects to receive next. It is an acknowledgment for the previous bytes being received successfully.
5. **Header Length (HLEN):** This is a 4 bit field that indicates the length of the TCP header by number of 4-byte words in the header, i.e., if the header is of 20 bytes (minimum length of TCP header), then this field will hold 5 (because 5 × 4 = 20) and the maximum length 60 bytes, then it will hold the value 15 (because 15 × 4 = 60). Hence, the value of this field is always between 5 and 15.
6. **Reserved:** This is a 6-bit field reserved for future use.
7. **Control Flags:** These are 6, 1-bit control bits that control connection establishment, connection termination, connection abortion, flow control, mode of transfer etc. [S-23]

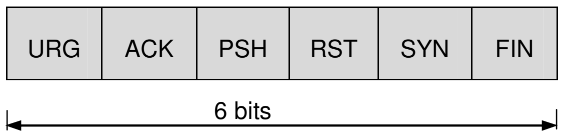


Fig. 3.13: Control Field

The function of control fields in TCP are:

- **URG:** Urgent pointer is valid.
- **ACK:** Acknowledgement number is valid (used in case of cumulative acknowledgement).
- **PSH:** Request for push.
- **RST:** Reset the connection.
- **SYN:** Synchronize sequence numbers.
- **FIN:** Terminate the connection.

8. **Window Size:** This field tells the window size of the sending TCP in bytes.
9. **Checksum:** This field holds the checksum for error control. It is mandatory in TCP as opposed to UDP.
10. **Urgent Pointer:** This field (valid only if the URG control flag is set) is used to point to data that is urgently required that needs to reach the receiving process at the earliest. The value of this field is added to the sequence number to get the byte number of the last urgent byte.
11. **Options:** There can be up to 40 bytes of optional information in the TCP header.

3.3.4 TCP Connection (Establishment and Termination) and Three-way Handshaking [W-23]

- TCP is a connection-oriented protocol and every connection-oriented protocol needs to establish connection in order to reserve resources at both the communicating ends.
- TCP transmits data in full-duplex mode. When two TCPs in two machines are connected, they are able to send segments to each other simultaneously.

3.3.4.1 Connection Establishment [W-24]

- To establish a connection, TCP uses a three-way handshaking. Fig. 3.14 shows an example of connection establishment using three-way handshaking.
- In this example we take, an application program, known as the client, wants to make a connection with another application program, known as 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 or initiate 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. 3.14.
- To establish a connection, the three-way (or 3-step) handshake occurs:
  1. **SYN Segment:** In this segment only the SYN flag is set. SYN segment is for synchronization of sequence numbers. The client in the 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). 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.
  2. **SYN+ACK Segment:** It sets two flag bits set namely, SYN and ACK. A SYN + ACK segment cannot carry data, but does consume one sequence number. SYN-ACK 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. Because it contains an acknowledgment, it also needs to define the receive window size.

3. **ACK Segment:** ACK segment acknowledges the receipt of the second segment with the ACK flag and acknowledgment number field. An ACK segment, if carrying no data, consumes no sequence number.

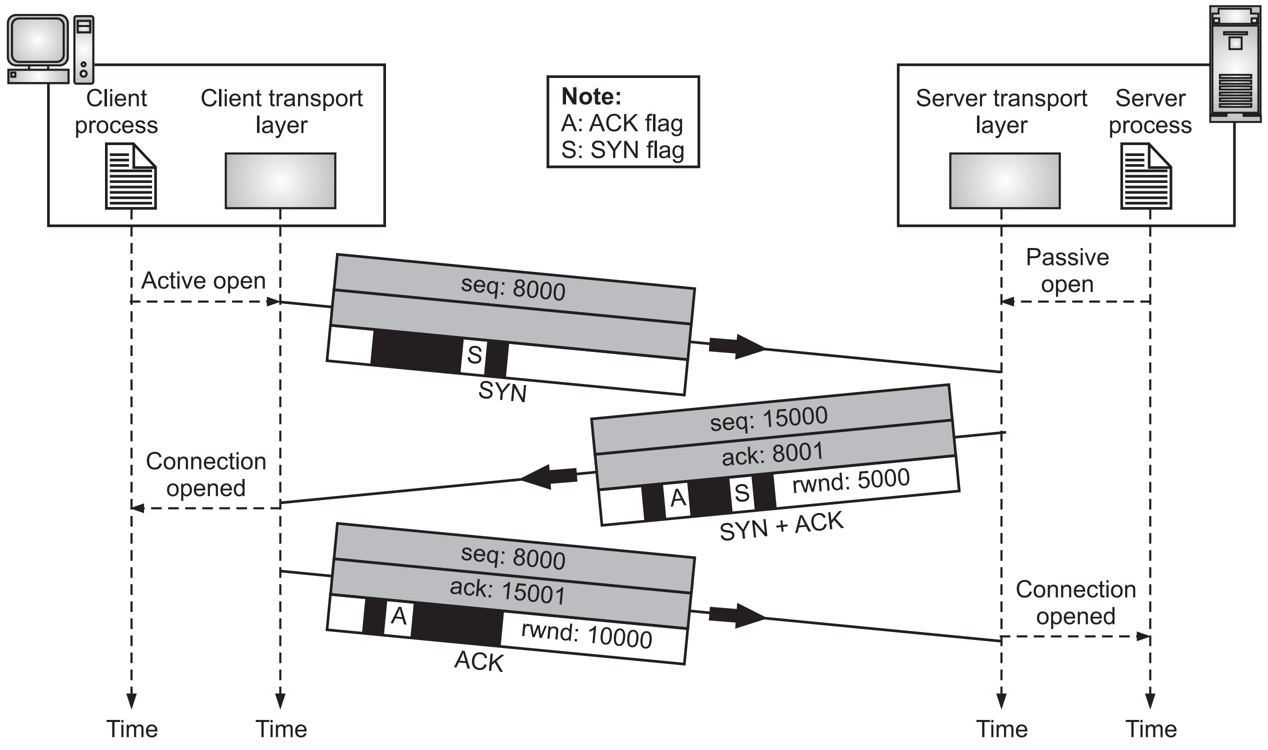


Fig. 3.14: Connection Establishment of TCP using Three-way Handshaking

3.3.4.2 Data Transfer in TCP

- After connection of TCP is established, bidirectional data transfer can take place. The client and server can send data and acknowledgments in both directions.
- Fig. 3.15 shows an example of TCP in which, after a connection is established, the client sends 2,000 bytes of data in two segments. The server then sends 2,000 bytes in one segment.
- The client sends one more segment. The first three segments carry both data and acknowledgment, but the last segment carries only an acknowledgment because there is no more data to be sent.
- The data segments sent by the client have the PSH (push) flag set so that the server TCP tries to deliver data to the server process as soon as they are received.
- The segment from the server, on the other hand, does not set the push flag. Most TCP implementations have the option to set or not set this flag.

3.3.4.3 Connection Termination

- Any of the two parties involved in exchanging/transferring data i.e., client or server can close the connection, although it is usually initiated by the client.
- Most implementations today allow two options for connection termination for TCP namely, three-way handshaking and four-way handshaking with a half-close option.

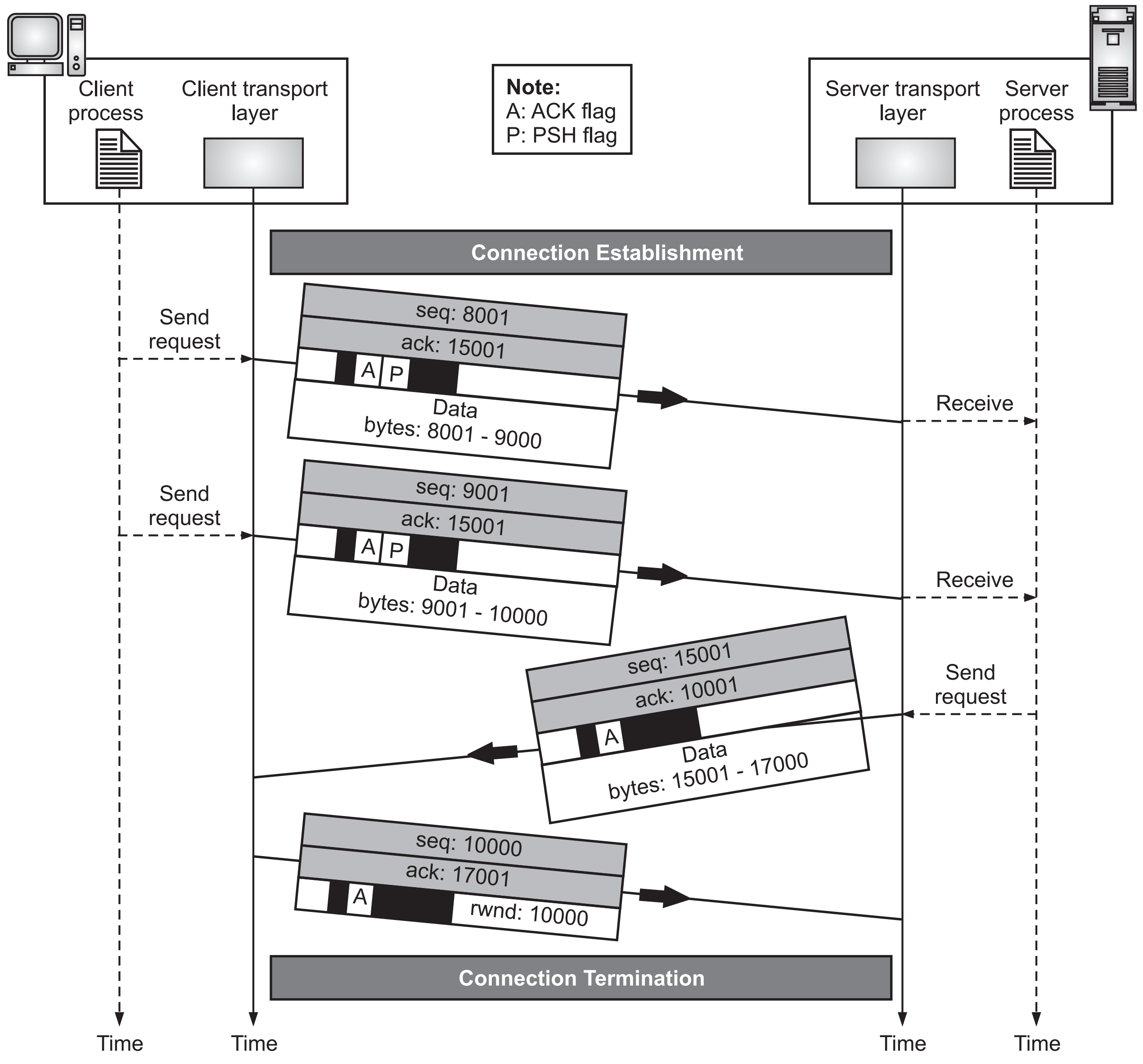


Fig. 3.15: Data Transfer in TCP

3.3.4.4 Three-Way Handshaking **[W-23, S-24]**

- Fig. 3.16 shows three-way handshaking for connection termination for TCP.
  1. **FIN Segment:** In a common situation, the client TCP, after receiving a close command from the client process, sends the first segment, a FIN segment in which the FIN flag is set. The FIN segment consumes one sequence number if it does not carry data.
  2. **FIN+ACK Segment:** The server TCP, after receiving the FIN segment, informs its process of the situation and sends the second segment, a FIN+ACK segment, to confirm the receipt of the FIN segment from the client and at the same time to announce the closing of the connection in the other direction. This segment can also contain the last chunk of data from the server. If it does not carry data, it consumes only one sequence number. The FIN + ACK segment consumes one sequence number if it does not carry data.
  3. **ACK Segment:** The client TCP sends the last segment, an ACK segment, to confirm the receipt of the FIN segment from the TCP server. This segment contains the acknowledgment number, which is one plus the sequence number received in the FIN segment from the server. This segment cannot carry data and consumes no sequence numbers.

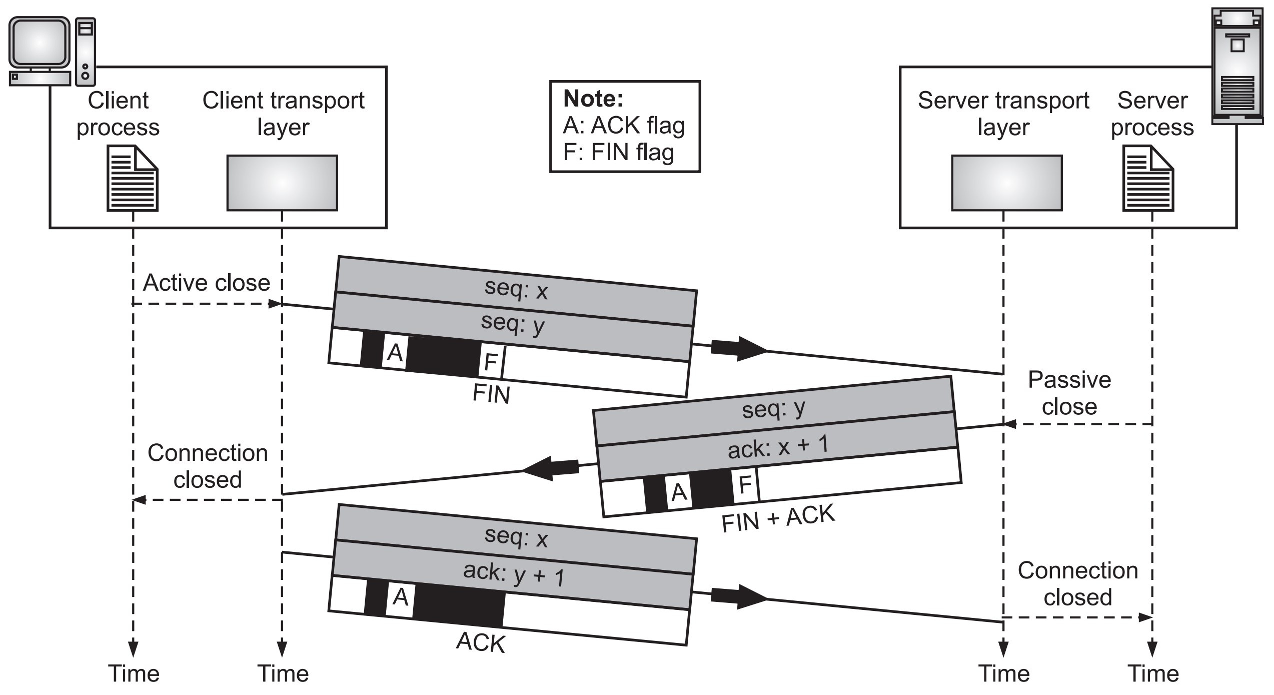


Fig. 3.16: three-way Handshaking for Connection Termination for TCP

Four-Way Handshaking:

- Four-way handshaking in TCP uses half-close option. In TCP, one end can stop sending data while still receiving data. This is called a half-close. Either the server or the client can issue a half-close request. It can occur when the server needs all the data before processing can begin.
- **For example:** An example of half-close option is sorting. When the client sends data to the server to be sorted, the server needs to receive all the data before sorting can start. This means the client, after sending all data, can close the connection in the client-to-server direction. However, the server-to-client direction must remain open to return the sorted data. The server, after receiving the data, still needs time for sorting; its outbound direction must remain open.
- Fig. 3.17 shows an example of a half-close option in four-way handshaking in TCP. The data transfer from the client to the server stops. The client half-closes the connection by sending a FIN segment.
- The server accepts the half-close by sending the ACK segment. The server, however, can still send data. When the server has sent all of the processed data, it sends a FIN segment, which is acknowledged by an ACK from the client.
- After half closing the connection, data can travel from the server to the client and acknowledgments can travel from the client to the server. The client cannot send any more data to the server. The second segment (ACK) consumes no sequence number.

Connection Reset in TCP:

- TCP at one end may deny a connection request, may abort an existing connection, or may terminate an idle connection. All of these are done with the reset (RST) flag.

3.3.5 Flow Control and Error Control [S-23]

- TCP is a connection-oriented protocol that provides extensive error control and flow control. In flow control TCP will verify that a sender is not overwhelming a receiver by sending packets faster than it can consume.
- TCP provides reliability using error control. Error control includes mechanisms for detecting corrupted segments, lost segments, out-of-order segments, and duplicated segments. Error control also includes a mechanism for correcting errors after they are detected.

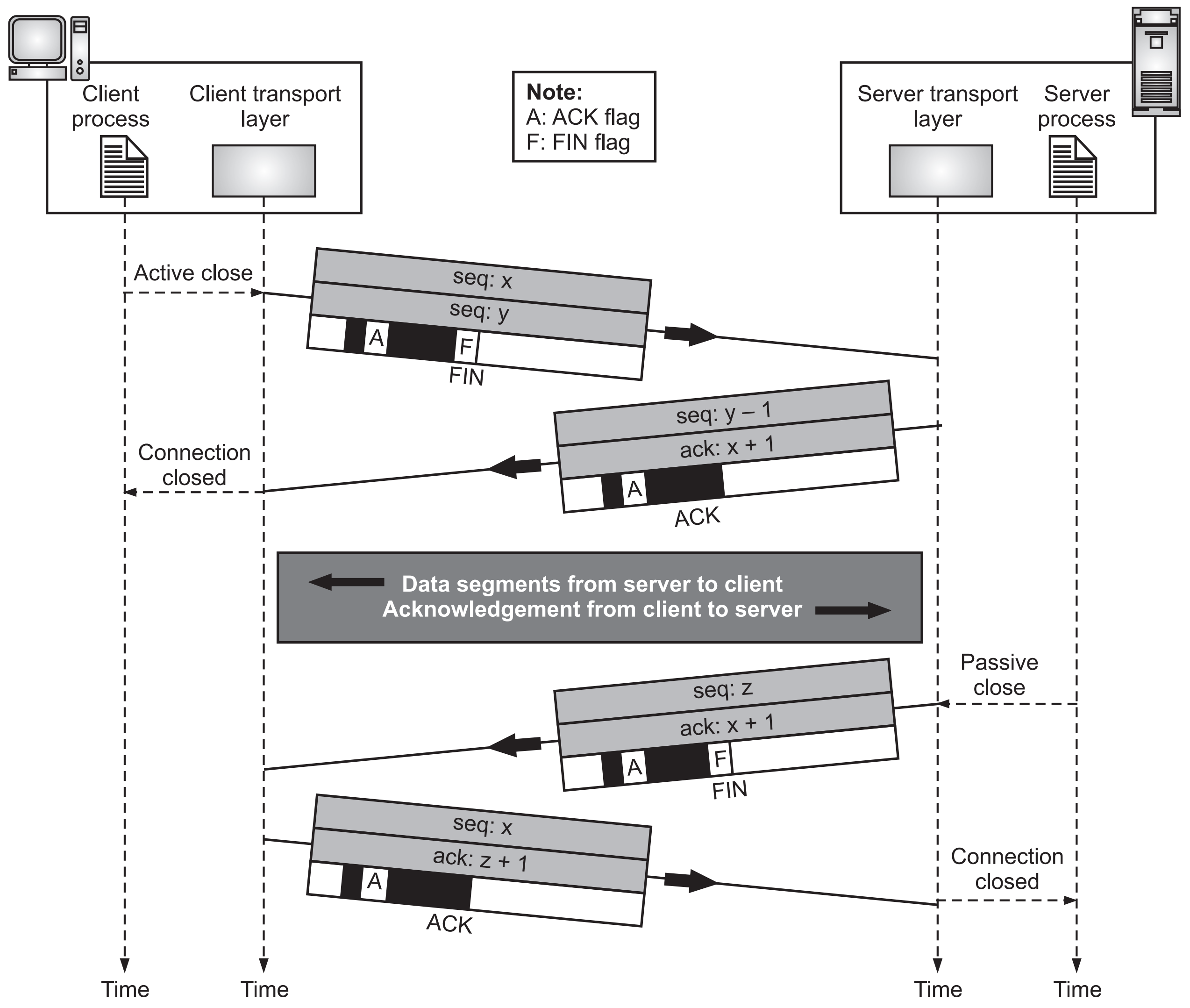


Fig. 3.17: Half Close in TCP

3.3.5.1 Flow Control in TCP **[S-23, W-24]**

- Flow control basically means that TCP will ensure that a sender is not overwhelming a receiver by sending packets faster than it can consume.
- The idea is that a node receiving data will send some kind of feedback to the node sending the data to let it know about its current condition.
- When we need to send data over a network, the sender application writes data to a socket, the transport layer (in our case, TCP) will wrap this data in a segment and hand it to the network layer (e.g. IP), that will somehow route this packet to the receiving node.
- On the other side of this communication, the network layer will deliver this piece of data to TCP, that will make it available to the receiver application as an exact copy of the data sent, meaning it will not deliver packets out of order, and will wait for a retransmission in case it notices a gap in the byte stream.
- TCP stores the data it needs to send in the send buffer, and the data it receives in the receive buffer. When the application is ready, it will then read data from the receive buffer.
- Flow control is all about making sure we don't send more packets when the receive buffer is already full, as the receiver wouldn't be able to handle them and would need to drop these packets.

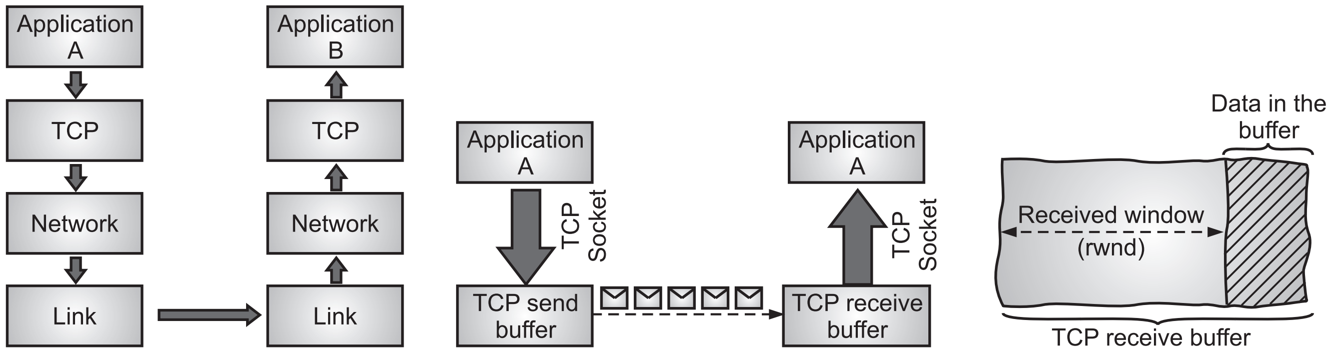


Fig. 3.18

- To control the amount of data that TCP can send, the receiver will advertise its Receive Window (rwnd), that is, the spare room in the receive buffer.
- Every time TCP receives a packet, it needs to send an ack message to the sender, acknowledging it received that packet correctly, and with this ackmessage it sends the value of the current receive window, so the sender knows if it can keep sending data.
- Flow control regulates the amount of data a source can send before receiving an acknowledgment from the destination.
- In communication at the transport layer, we are dealing with four entities namely, sender process, sender transport layer, receiver transport layer, and receiver process.
- The sending process at the application layer is only a producer. It produces message chunks and pushes them to the transport layer.
- The sending transport layer has a double role it is both a consumer and the producer. It consumes the messages pushed by the producer. It encapsulates the messages in packets and pushes them to the receiving transport layer.
- The receiving transport layer has also a double role: it is the consumer for the packets received from the sender. It is also a producer; it needs to decapsulate the messages and deliver them to the application layer.
- The last delivery, however, is normally a pulling delivery; the transport layer waits until the application-layer process asks for messages.

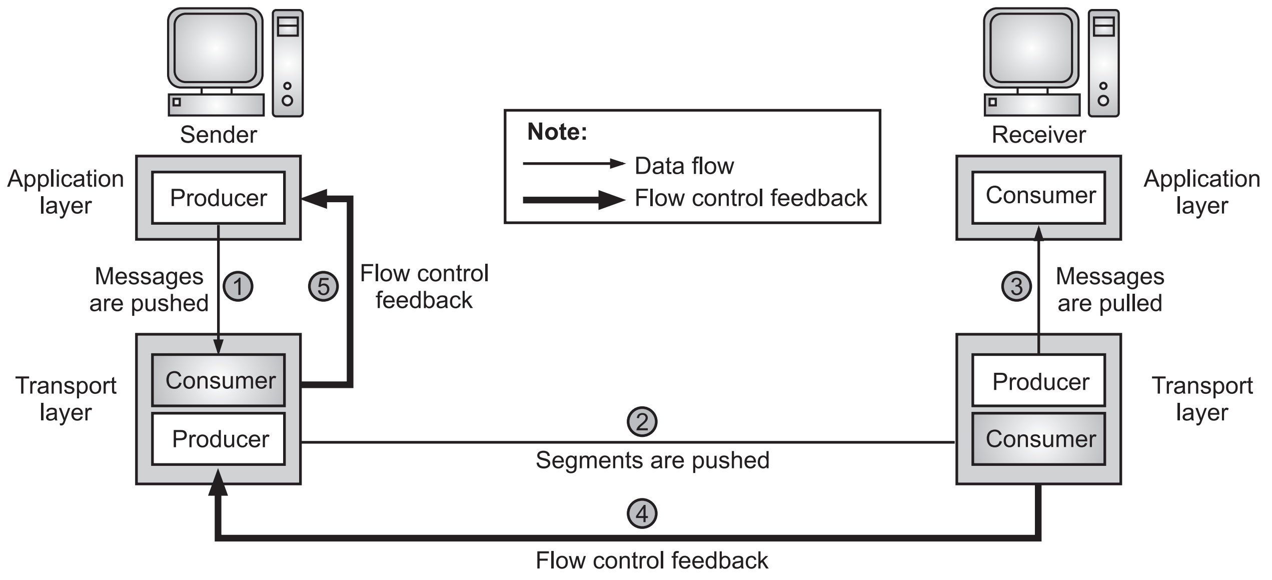


Fig. 3.19: Flow Control in TCP

- Fig. 3.20 shows an example of flow control in unidirectional data transfer (from client to server). Eight segments are exchanged between the client and server are explained below:
  1. **Segment 1:** From the client to the server (a SYN segment) to request connection: The client announces its initial seqNo = 100. When this segment arrives at the server, it allocates a buffer size of 800 (an assumption) and sets its window to cover the whole buffer (rwnd = 800). Note that the number of the next byte to arrive is 101.
  2. **Segment 2:** From the server to the client: This is an ACK + SYN segment. The segment uses ackNo = 101 to show that it expects to receive bytes starting from 101. It also announces that the client can set a buffer size of 800 bytes.
  3. **Segment 3:** The ACK segment from the client to the server.
  4. **Segment 4:** After the client has set its window with the size (800) dictated by the server, the process pushes 200 bytes of data. The TCP client numbers these bytes 101 to 300. It then creates a segment and sends it to the server. The segment shows the starting byte number as 101 and the segment carries 200 bytes. The window of the client is then adjusted to show 200 bytes of data are sent but waiting for acknowledgment. When this segment is received at the server, the bytes are stored, and the receive window closes to show that the next byte expected is byte 301; the stored bytes occupy 200 bytes of buffer.

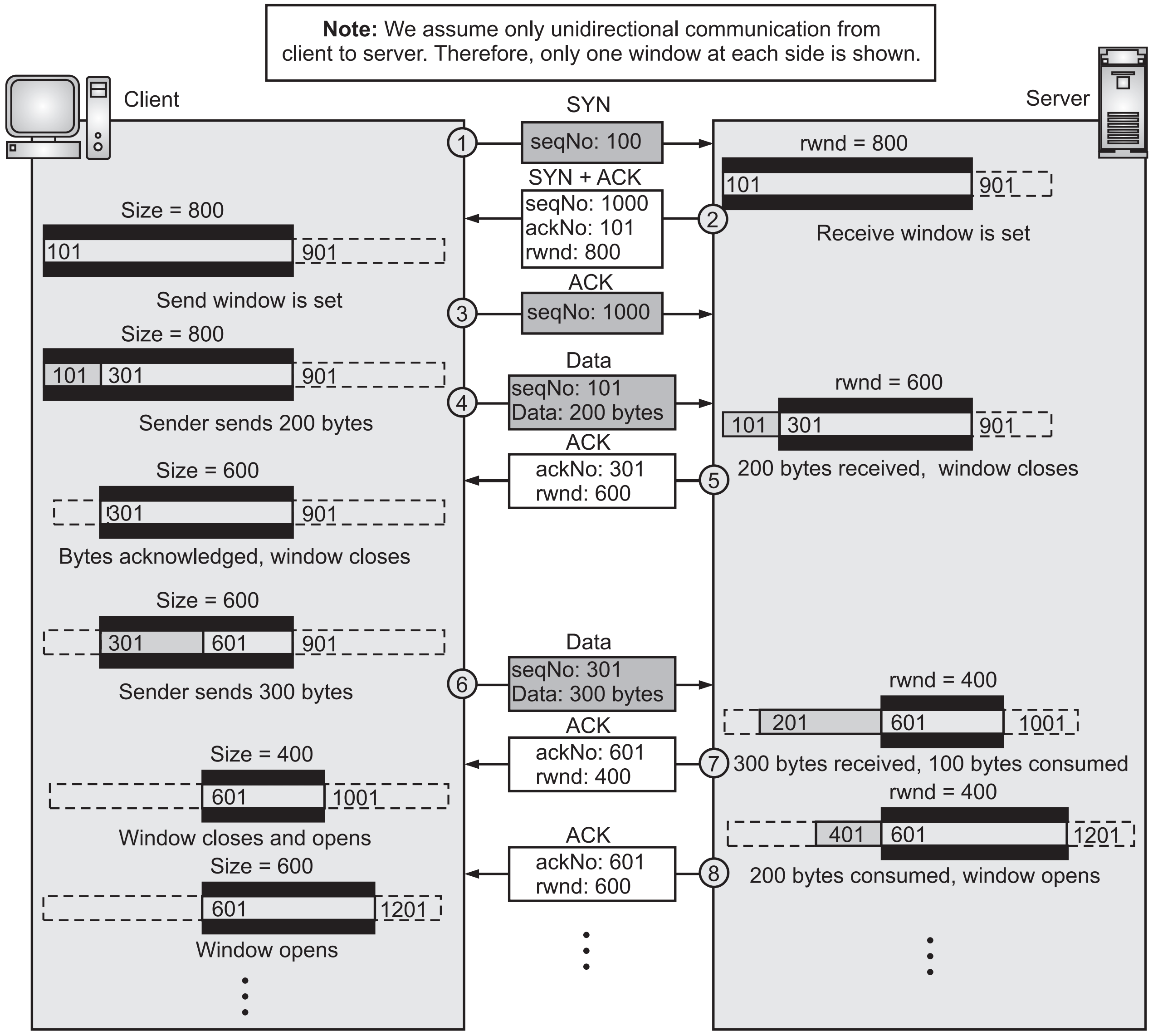


Fig. 3.20

5. **Segment 5:** The feedback from the server to the client: The server acknowledges bytes up to and including 300 (expecting to receive byte 301). The segment also carries the size of the receive window after decrease (600). The client, after receiving this segment, purges the acknowledged bytes from its window and closes its window to show that the next byte to send is byte 301. The window size, however, decreases to 600 bytes. Although the allocated buffer can store 800 bytes, the window cannot open (moving its right wall to the right) because the receiver does not let it.
6. **Segment 6:** Sent by the client after its process pushes 300 more bytes. The segment defines seqNo as 301 and contains 300 bytes. When this segment arrives at the server, the server stores them, but it has to reduce its window size. After its process has pulled 100 bytes of data, the window closes from the left for the amount of 300 bytes, but opens from the right for the amount of 100 bytes. The result is that the size is only reduced 200 bytes. The receiver window size is now 400 bytes.
7. **Segment 7:** In segment 7, the server acknowledges the receipt of data, and announces that its window size is 400. When this segment arrives at the client, the client has no choice but to reduce its window again and set the window size to the value of rwnd = 400 advertised by the server. The send window closes from the left by 300 bytes, and opens from the right by 100 bytes.
8. **Segment 8:** It is also from the server after its process has pulled another 200 bytes. Its window size increases. The new rwnd value is now 600. The segment informs the client that the server still expects byte 601, but the server window size has expanded to 600. We need to mention that the sending of this segment depends on the policy imposed by the implementation. Some implementations may not allow advertisement of the rwnd at this time; the server then needs to receive some data before doing so. After this segment arrives at the client, the client opens its window by 200 bytes without closing it. The result is that its window size increases to 600 bytes.

3.3.5.2 Error Control [S-23]

- TCP is a reliable transport layer protocol means that an application program that delivers a stream of data to TCP relies on TCP to deliver the entire stream to the application program on the other end in order, without error, and without any part lost or duplicated.
- TCP provides reliability using error control. Error control includes mechanisms for detecting and resending corrupted segments, resending lost segments, storing out-of-order segments until missing segments arrive, and detecting and discarding duplicated segments.
- Error control in TCP is achieved through the use of three simple tools: are checksum, acknowledgment, and time-out.
- TCP protocol has methods for finding out corrupted segments, missing segments, out-of-order segments and duplicated segments.
- Reliability can be achieved to add error control service to the transport layer. Error control at the transport layer is responsible to:
  1. Detect and discard corrupted packets.
  2. Keep track of lost and discarded packets and resend them.
- Fig. 3.21 shows the error control between the sending and receiving transport layer.

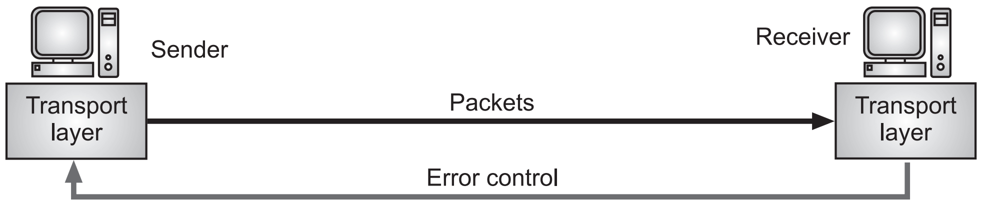


Fig. 3.21: Error Control at Transport Layer

- Error control in TCP is mainly done through use of three simple techniques:
  1. **Checksum:** Every segment contains a checksum field which is used to find corrupted segment. If the segment is corrupted, then that segment is discarded by the destination TCP and is considered as lost.
  2. **Acknowledgement**: TCP has another mechanism called acknowledgement to affirm that the data segments have been delivered. Control segments that contain no data but has sequence number will be acknowledged as well but ACK segments are not acknowledged.
  3. **Retransmission**: When a segment is missing, delayed to deliver to receiver, corrupted when it is checked by receiver then that segment is retransmitted again. Segments are retransmitted only during two events (when the sender receives three duplicate acknowledgements (ACK) or when a retransmission timer expires).
     - **(i) Retransmission after RTO:** TCP always preserve one Retransmission Time-Out (RTO) timer for all sent but not acknowledged segments. When the timer runs out of time, the earliest segment is retransmitted. Here, no timer is set for acknowledgement. In TCP, RTO value is dynamic in nature and it is updated using Round Trip Time (RTT) of segments. RTT is the time duration needed for a segment to reach receiver and an acknowledgement to be received to the sender.
     - **(ii) Retransmission after Three duplicate ACK segments:** RTO method works well when the value of RTO is small. If it is large, more time is needed to get confirmation about whether a segment has delivered or not. Sometimes, one segment is lost and the receiver receives so many out-of-order segments that they cannot be saved. In order to solve this situation, three duplicate acknowledgement method is used and missing segment is retransmitted immediately instead of retransmitting already delivered segment. This is a fast retransmission because it makes it possible to quickly retransmit lost segments instead of waiting for timer to end.
  4. **Out-of-Order Segments:** TCP implementations today do not discard out-of-order segments. They store them temporarily and flag them as out-of-order segments until the missing segments arrive. Note, however, that out-of-order segments are never delivered to the process. TCP guarantees that data are delivered to the process in order.

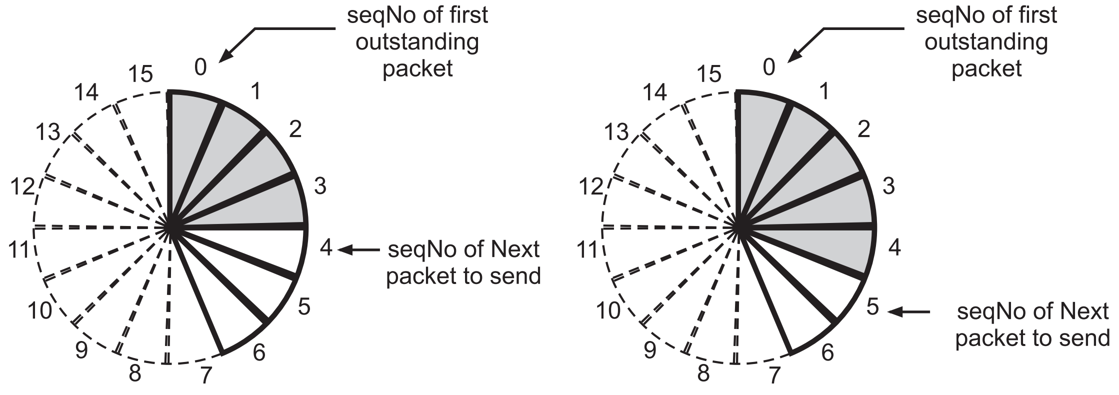
Combination of Flow and Error Control:

- Flow control requires the use of two buffers, one at the sender site and the other at the receiver site. The error control requires the use of sequence and acknowledgment numbers by both sides.
- These two requirements can be combined if we use two numbered buffers, one at the sender, one at the receiver.
- At the sender, when a packet is prepared to be sent, we use the number of the next free location, x, in the buffer as the sequence number of the packet.
- When the packet is sent, a copy is stored at memory location x, awaiting the acknowledgment from the other end. When an acknowledgment related to a sent packet arrives, the packet is purged and the memory location becomes free.
- At the receiver, when a packet with sequence number y arrives, it is stored at the memory location y until the application layer is ready to receive it. An acknowledgment can be sent to announce the arrival of packet y.

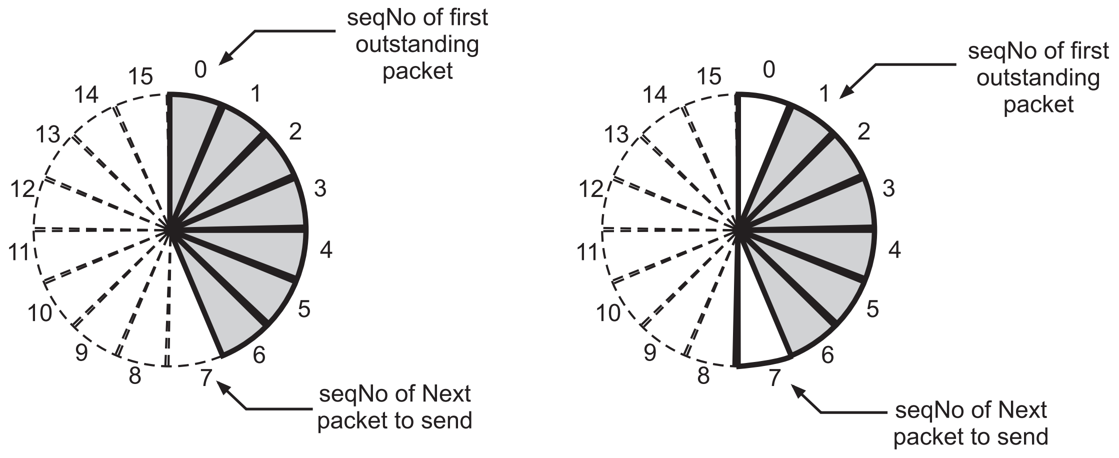
Sliding Window:

- Transmission Control Protocol (TCP) uses a sliding window for flow control. In the transport layer, the sliding window protocol, often used by TCP, is a flow control mechanism that regulates the amount of data a sender can transmit before receiving acknowledgment from the receiver, ensuring efficient and reliable data transfer.

- Since the sequence numbers (each data segment is assigned a sequence number, allowing the receiver to identify the order of segments) used modulo 2m a circle can represent the sequence number from 0 to 2m-1 (See Fig. 3.22).
- The buffer is represented as a set of slices, called the sliding window, that occupy part of the circle at any time.
- At the sender site, when a packet is sent, the corresponding slice is marked. When all the slices are marked, it means that the buffer is full and no further messages can be accepted from the application layer.
- When an acknowledgment arrives, the corresponding slice is unmarked. If some consecutive slices from the beginning of the window are unmarked, the window slides over the range of the corresponding sequence number to allow more free slices at the end of the window.
- Fig. 3.22 shows the sliding window at the sender. The sequence number are modulo 16 (m = 4) and the size of the window is 7.


seqNo of first outstanding packet
seqNo of Next packet to send
seqNo of first outstanding packet
seqNo of Next packet to send
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15

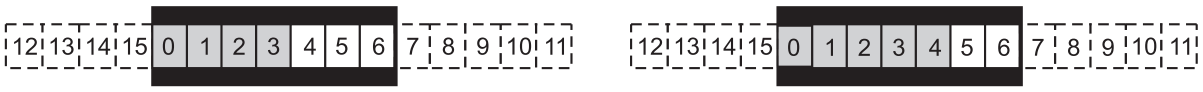
(a) Four Packets have been send **(b) Five Packets have been send**


seqNo of first outstanding packet
seqNo of Next packet to send
seqNo of first outstanding packet
seqNo of Next packet to send
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15

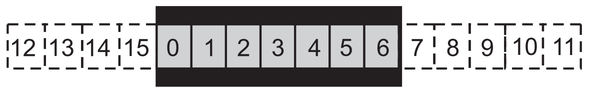
(a) Seven Packets have been send Window is Full **(b) Packet 0 has been Acknowledged, Window Slide**

Fig. 3.22: Sliding Window in Circular Format

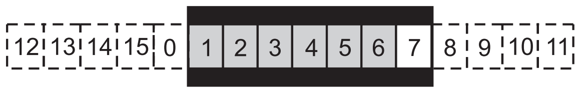
- The "window" represents the maximum amount of data (in bytes or packets) the sender is allowed to transmit without receiving an acknowledgment.
- Most protocols show the sliding window using linear representation as shown in Fig. 3.23.



(a) Four Packets have been Sent **(b) Five Packets have Been Sent**



(c) Seven Packets have been Sent
Window is Full



(d) Packet 0 have been Acknowledged
and Windows Slide

Fig. 3.23: Sliding window in linear format

3.3.6 Congestion Control (Open Loop and Closed Loop)

- In the transport layer, congestion control mechanisms, particularly in protocols like TCP, aim to prevent network overload by adjusting the sender's transmission rate based on network conditions, ensuring efficient and reliable data delivery.

Why Congestion Control is Needed?

- A network is a shared resource, and if multiple senders transmit data without coordination, it can lead to congestion, where routers and network links become overwhelmed.
- Congestion results in packet loss, increased delays, and reduced network performance.
- The transport layer is responsible for ensuring reliable data delivery, and congestion control is crucial for achieving this reliability.
- TCP uses an "end-to-end" approach, meaning that congestion control is primarily handled by the sender, based on feedback from the network (e.g., packet loss or delay).
- Congestion in a network may occur if the load on the network—the number of packets sent to the network - is greater than the capacity of the network—the number of packets a network can handle.
- Congestion control refers to the mechanisms and techniques to control the congestion and keep the load below the capacity.
- Congestion in a network or internetwork occurs because routers and switches have queues—buffers that hold the packets before and after processing.
- The packet is put in the appropriate output queue and waits its turn to be sent. These queues are finite, so it is possible for more packets to arrive at a router than the router can buffer.
- Congestion control refers to techniques and mechanisms that can either prevent congestion, before it happens, or remove congestion, after it has happened.

3.3.6.1 Open-Loop and Closed-Loop Congestion Control

- Congestion control can be implemented using open-loop (proactive) or closed-loop (reactive) mechanisms.
  1. **Open-Loop Congestion Control:** In open-loop congestion control, policies are applied to prevent congestion before it happens. In these mechanisms, congestion control is handled by either the source or the destination.
  2. **Closed-Loop Congestion Control:** Closed-loop congestion control mechanisms try to alleviate congestion after it happens.

Benefits of Transport-Layer Congestion Control:

1. **Reliable Data Delivery:** By preventing congestion, congestion control contributes to reliable and timely data transfer.
2. **Improved Network Utilization:** By dynamically adjusting transmission rates, congestion control helps ensure that network resources are used efficiently.

3.4 TLS (TRANSPORT LAYER SECURITY)

- Transport Layer Security, or TLS, is a widely adopted security protocol designed to facilitate privacy and data security for communications over the Internet.
- A primary use case of TLS is encrypting the communication between web applications and servers, such as web browsers loading a website. TLS can also be used to encrypt other communications such as email, messaging, and Voice over IP (VoIP).
- TLS was derived from a security protocol called Secure Socket Layer (SSL). TLS ensures that no third party may eavesdrop or tampers with any message.
- TLS (Transport Layer Security) is a cryptographic protocol that provides secure communication over a network, encrypting data exchanged between a client and a server, ensuring privacy and integrity, and is often implemented on top of TCP.
- TLS is a security protocol that encrypts data transmitted over a network, providing confidentiality and ensuring that data cannot be intercepted or tampered with during transit.

3.4.1 Working of TLS

- TLS is a cryptographic protocol that provides end-to-end security of data sent between applications over the Internet.
- It is mostly familiar to users through its use in secure web browsing, and in particular the padlock icon that appears in web browsers when a secure session is established.
- Transport Layer Security (TLS) encrypts data sent over the Internet to ensure that eavesdroppers and hackers are unable to see what we transmit which is particularly useful for private and sensitive information such as passwords, credit card numbers, and personal correspondence.
- TLS uses encryption for the client and server to generate a secure connection between the applications.
- It begins when users access a secured website by specifying the TLS encryption method like the Advanced Encryption Standard (AES).
- It works with two security layers – the TLS record protocol and the TLS handshake protocol. These protocols use symmetric and asymmetric cryptography methods to secure data transfer and communications between the clients and web servers.
- The TLS handshake protocol, for example, uses asymmetric cryptography to generate public and private keys that encrypt and decrypt data.
- Then, the overall process is as follows:
  1. The client sends a list of all TLS versions along with suggestions for a cipher suite and generates a random number that will be used later.
  2. The server confirms which options it will use to initiate the connection.
  3. The server sends a TLS certificate to the client for the authentication process.
  4. After validating the certificate, the client creates and sends a pre-master key encrypted by the server's public key and decrypted by the server's private key.
  5. The client and server generate session keys using the previously generated random numbers and the pre-master key.
  6. Both the client and server have a finished message that has been encrypted with a session key.
  7. The TLS handshake process is finished, and both the client and server have created secure symmetric encryption.

Transport Layer Security (TLS) Handshake Protocol:

- The working condition of the TLS Handshake protocol is shown below:
  - A client sends a synchronous message "client hello" requesting a connection and presents a list of supported cipher suites and a random string of bytes.
  - The server responds with a "server hello" message containing a server certificate.
  - A client sends a synchronous message "client hello" requesting a connection and presents a list of supported cipher suites and a random string of bytes.
  - The server responds with a "server hello" message containing a server certificate.
  - The server is sending its SSL certificate to the client for the purpose of authentication. The client then authenticates the server by verifying the server's SSL certificate and also sends a certificate for authentication if requested by the server.
  - The client sends the client key exchange, change Cipher specification finished message to the server.
  - The server decrypts the message sent by client secret with the private key.
  - Both client and server generate session keys from the client random, the server random, and the secret message.
  - The client sends a "finished" message that has been encrypted with a session key.
  - The server responds with a finished message which was encrypted with a session key.
  - The client and server have successfully achieved secure symmetric encryption, meaning the handshake is complete and communication can continue with the established session keys.
  - Finally transfer the application data.

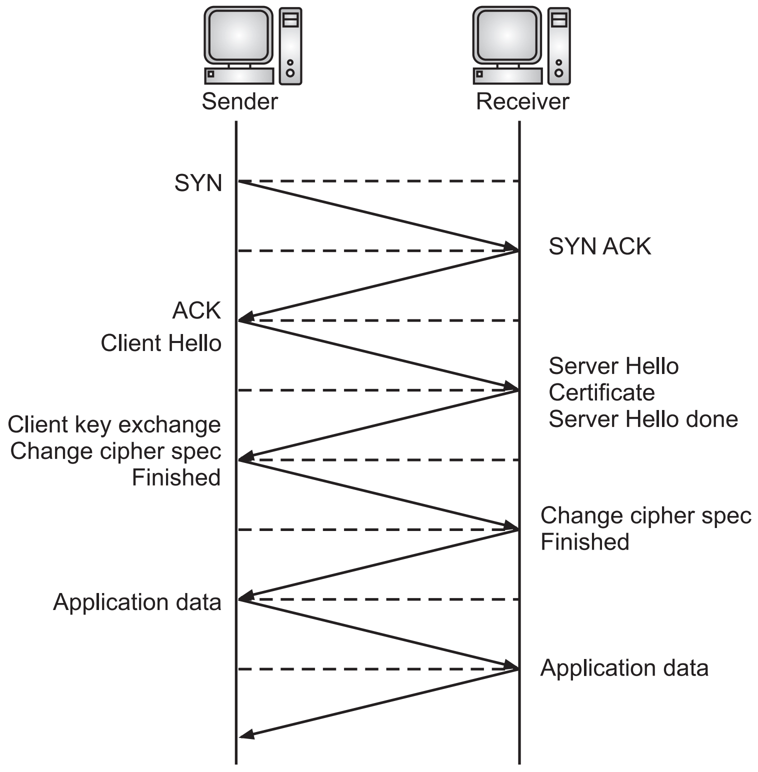


Fig. 3.24: TLS Handshake

Benefits of TLS:

1. **Encryption:** TLS/SSL can help to secure transmitted data using encryption.
2. **Interoperability:** TLS/SSL works with most web browsers, including Microsoft Internet Explorer and on most operating systems and web servers.

3. **Algorithm Flexibility:** TLS/SSL provides operations for authentication mechanism, encryption algorithms and hashing algorithm that are used during the secure session.
4. **Ease of Deployment:** Many applications TLS/SSL temporarily on a windows server 2003 operating systems.
5. **Ease of Use:** Because we implement TLS/SSL beneath the application layer, most of its operations are completely invisible to client.
6. **Data Privacy:** TLS encrypts data, preventing eavesdropping and ensuring that sensitive information remains confidential.
7. **Data Integrity:** TLS ensures that data is not tampered with during transit, providing assurance that the data received is the same as the data sent.
8. **Authentication:** TLS allows for verification of the identities of both the client and server, preventing man-in-the-middle attacks.
9. **Trust:** TLS helps build trust between users and websites, as users can be assured that their data is being protected.

3.4.2 Applications of TLS

- Applications of TLS includes:
  1. **Secure Web Browsing (HTTPS):** TLS is the foundation of secure web browsing, where websites use HTTPS (Hypertext Transfer Protocol Secure) to encrypt communication.
  2. **Email:** TLS is used to encrypt email communication, protecting the privacy of emails.
  3. **VoIP (Voice over IP):** TLS can secure voice and video communication over the internet.
  4. **File Transfers:** TLS can secure file transfers, ensuring that data is not intercepted during transmission.
  5. **Virtual Private Networks (VPNs):** TLS can be used to secure VPN connections, protecting data from eavesdropping.
  6. **Other Applications:** TLS can be used to secure a wide variety of applications that require secure communication over a network.

3.5 STREAM CONTROL TRANSMISSION PROTOCOL (SCTP) [S-22, W-22, S-23]

- SCTP (Stream Control Transmission Protocol) is a transport-layer protocol, similar to TCP, that provides reliable, in-sequence data transmission.
- But with features like multi-homing and message-oriented data transfer, making it suitable for applications requiring robust and flexible communication.
- SCTP is a relatively new protocol in the transport layer of TCP/IP protocol suite is designed as a general-purpose transport layer protocol that can handle multimedia and stream traffic, which are increasing every day on the Internet.
- Stream Control Transmission Protocol (SCTP) is a reliable, message-oriented transport layer protocol. SCTP combines the features of TCP and UDP.
- SCTP maintains the message boundaries and detects the lost data, duplicate data as well as out-of-order data.
- SCTP provides the Congestion control as well as Flow control. SCTP is especially designed for Internet applications such as IUA (ISDN over IP), M2UA and M3UA (telephony signaling), H.248 (media gateway control), H.323 (IP telephony), and SIP (IP telephony), need a more sophisticated service than TCP can provide.
- SCTP lies between the application layer and the network layer (See Fig. 3.25) and serves as the intermediary between the application programs and the network operations.

3.5.1 Services

- Some important services provided by SCTP are as stated below:

1. **Process to Process Communication:**

- SCTP provides process to process communication and also uses all important ports of TCP space and also some extra port numbers.
- Following table list some extra port numbers used by SCTP:

Sr. No.	Protocol	Port Number	Description
1.	IUA	9990	ISDN over IP.
2.	M2UA	2904	SS7 telephony signaling.
3.	M3UA	2905	SS7 telephony signaling.
4.	H.248	2945	Media gateway control.
5.	H.323	1718, 1719, 1720, 11720	IP telephony.
6.	SIP	5060	IP telephony.

2. **Multi-Stream Facility:**

- SCTP provides multi-stream service to each connection called as association. If one stream gets blocked, then the other stream can deliver the data.

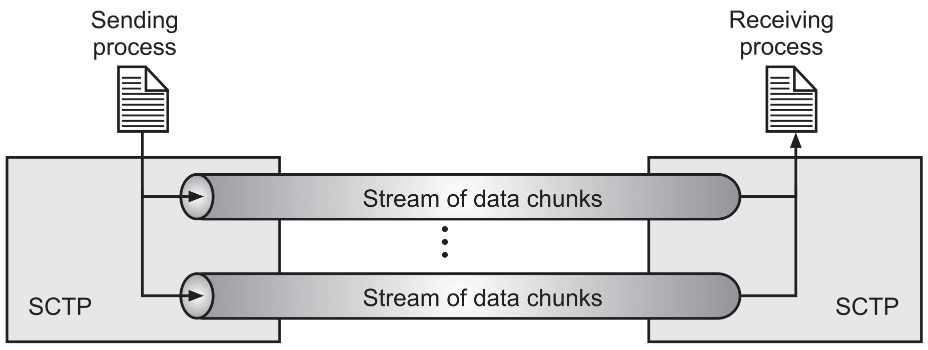


Fig. 3.25: Multiple Stream Concept

3. **Full-Duplex Communication:**

- SCTP provides full-duplex service, (the data can flow in both directions at the same time).

4. **Connection-Oriented Service:**

- The SCTP is a connection-oriented protocol, just like TCP with the only difference that, it is called association in SCTP.
- If User1 wants to send and receive message from User2, the steps are:

  Step 1: The two SCTPs establish the connection with each other.

  Step 2: Once the connection is established, the data gets exchanged in both the directions.

  Step 3: Finally, the association is terminated.

5. **Reliability:**

- SCTP uses an acknowledgement mechanism to check the arrival of data.

6. **Multihoming:**

- A TCP connection involves one source address and one destination IP address. This means that even if the sender or receiver is a multihomed host (connected to more than one physical address with multiple IP addresses), only one of these IP addresses per end can be utilized during the connection.

- An SCTP association, on the other hand, supports multihoming service. The sending and receiving host can define multiple IP addresses in each end for an association.
- In this fault-tolerant approach, when one path fails, another interface can be used for data delivery without interruption.
- This fault-tolerant feature is very helpful when we are sending and receiving a real-time payload such as Internet telephony.
- In Fig. 3.26, the client is connected to two local networks with two IP addresses. The server is also connected to two networks with two IP addresses. The client and the server can make an association using four different pairs of IP addresses.
- However, note that in the current implementations of SCTP, only one pair of IP addresses can be chosen for normal communication; the alternative is used if the main choice fails.
- In other words, at present, SCTP does not allow load sharing between different paths.

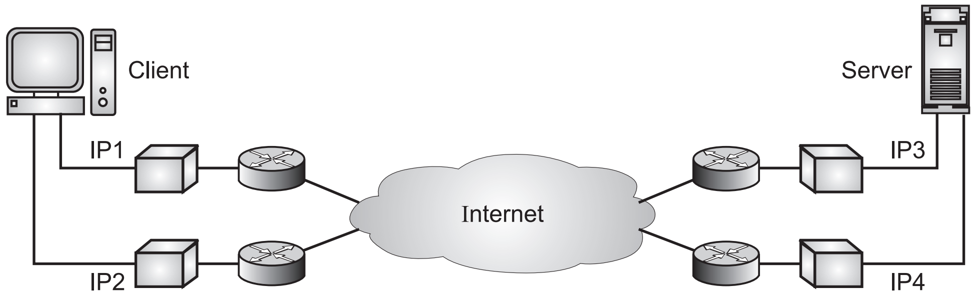


Fig. 3.26: Concept of Multihoming in SCTP

3.5.2 Features of SCTP [W-24]

- Features of SCTP are listed below:
  1. **Transmission Sequence Number (TSN):** The unit of data in SCTP is a data chunk. Data transfer in SCTP is controlled by numbering the data chunks. In SCTP, TSN is used to assign the numbers to different data chunks.
  2. **Stream Identifier (SI):** The SI is a 16 bit number and starts with 0. In SI, there are several streams in each association and it is needed to identify them. Each data chunk needs to carry the SI in the header, so that it is properly placed in its stream on arrival.
  3. **Packets:** In SCTP, the data is carried out in the form of data chunks and control information is carried as control chunks. Data chunks and control chunks are packed together in the packet.
  4. **Multihoming:** Multihoming allows both ends, (sender and receiver) to define multiple IP addresses for communication. But, only one of these can be defined as primary address and the remaining can be used as alternative addresses.
  5. **Flow Control:** Like TCP, SCTP implements flow control to avoid overwhelming the receiver.
  6. **Error Control:** Like TCP, SCTP implements error control to provide reliability. TSN numbers and acknowledgment numbers are used for error control.
  7. **Congestion Control:** Like TCP, SCTP implements congestion control to determine how many data chunks can be injected into the network.

3.5.3 Packet Format of SCTP [S-23, S-24]

- An SCTP packet has a mandatory general header and a set of blocks called chunks. There are two types of chunks: control chunks and data chunks.

- A control chunk controls and maintains the association. A data chunk carries user data. In a packet, the control chunks come before the data chunks.
- The Fig. 3.27 shows the general format of an SCTP packet.

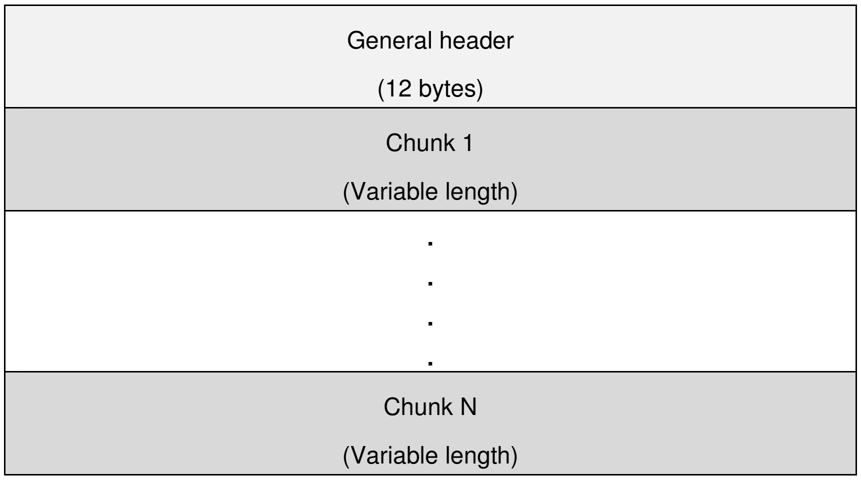


Fig. 3.27: Packet Format of SCTP

General Header of SCTP Packet:

- The general header (packet header) of SCTP defines the endpoints of each association to which the packet belongs, guarantees that the packet belongs to a particular association, and preserves the integrity of the contents of the packet including the header itself.
- The format of the general header is shown in the Fig. 3.28.

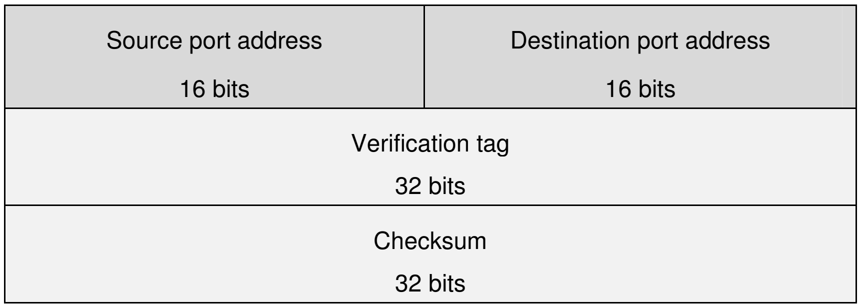


Fig. 3.28: General Header of SCTP

- There are four fields in the general header of SCTP are explained below:
  1. **Source Port Address:** This is a 16-bit field that defines the port number of the process sending the packet.
  2. **Destination Port Address:** This is a 16-bit field that defines the port number of the process receiving the packet.
  3. **Verification Tag:** This is a number that matches a packet to an association. This prevents a packet from a previous association from being mistaken as a packet in this association. It serves as an identifier for the association; it is repeated in every packet during the association. There is a separate verification used for each direction in the association.
  4. **Checksum:** This 32-bit field contains a CRC-32 checksum. Note that the size of the checksum is increased from 16 (in UDP, TCP and IP) to 32 bits to allow the use of the CRC-32 checksum.

Chunks:

- Control information or user data are carried in chunks. Chunks have a common layout as shown in Fig. 3.29.

- The first three fields are common to all chunks; the information field depends on the type of chunk. The important point to remember is that SCTP requires the information section to be a multiple of 4 bytes; if not, padding bytes (eight 0s) are added at the end of the section.

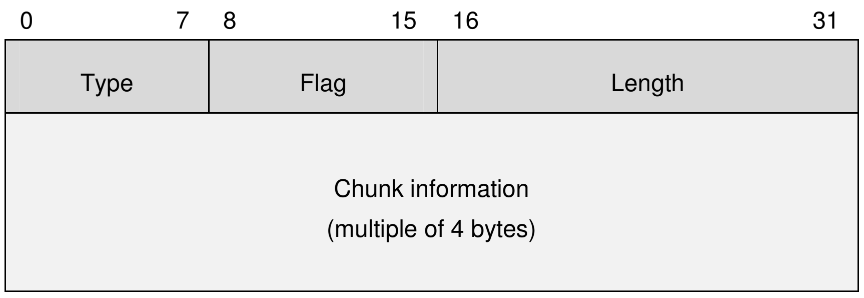


Fig. 3.29: Layout of Chunk

- The description of the common fields in SCTP chunks are as follows:

1. **Chunk Type:** Identifies the contents of the chunk value field. This is 1-byte long. This 8-bit field can define up to 256 types of chunks. Only a few have been defined so far; the rest are reserved for future use. Following table shows the list of chunks and its description:

Type	Chunk	Description
0	DATA	User data.
1	INIT	Sets up an association.
2	INIT ACK	Acknowledges INIT chunk.
3	SACK	Selective acknowledgment.
4	HEARTBEAT	Probes the peer for liveliness.
5	HEARTBEAT ACK	Acknowledges HEARTBEAT chunk.
6	ABORT	Abort an association.
7	SHUTDOWN	Terminates an association.
8	SHUTDOWN ACK	Acknowledges SHUTDOWN chunk.
9	ERROR	Reports errors without shutting down.
10	COOKIE ECHO	Third packet in association establishment.
11	COOKIE ACK	Acknowledges COOKIE ECHO chunk.
14	SHUTDOWN COMPLETE	Third packet in association termination.
192	FORWARD TSN	For adjusting cumulating TSN.

2. **Chunk Flags:** Consists of 8 flag-bits whose definition varies with the chunk type. The default value is zero. This indicates that no application identifier is specified by the upper layer for the data.
3. **Chunk Length:** Specifies the total length of the chunk in bytes. This field is 2 - bytes long. If the chunk does not form a multiple of 4 bytes (that is, the length is not a multiple of 4) it is implicitly padded with zeros which are not included in the chunk length.
4. **Chunk Value:** A general purpose data field.

Practice Questions

1. Enlist protocols of transport layer.
2. Explain the mechanism of process-to-process delivery.
3. Compare multiplexing and demultiplexing.
4. Explain types of multiplexing.
5. Compare connection-less and connection-oriented service.
6. What is UDP? Which services provided by UDP?
7. Describe UDP datagram with diagram.
8. What are the applications of UDP?
9. Describe TCP with its services.
10. Describe flow and error control in TCP.
11. Enlist features of UDP.
12. What SCTP? Enlist its features.
13. How to establish and terminate connection in TCP?
14. Differentiate between TCP and IP.
15. What are the services provided by SCTP? Explain two of them in detail.
16. Explain features of SCTP.

MSBTE Questions with Answers

Summer 2023

1. State the use of six flags in TCP header. **[2 M]**

Ans. Refer to Section 3.3.3, Point (7).

2. List UDP services and UDP applications, (any 4 each). **[4 M]**

Ans. Refer to Sections 3.2.2 and 3.2.4.

3. Explain TCP with respect to flow control and error control. **[6 M]**

Ans. Refer to Section 3.3.2.

4. Describe the fields of SCTP packet format. Explain SCTP association establishment process. **[6 M]**

Ans. Refer to Section 3.5.3.

Winter 2023

1. List any two features of TCP. **[2 M]**

Ans. Refer to Section 3.3.2.

2. Enlist any two services offered by UDP. **[2 M]**

Ans. Refer to Section 3.2.2.

3. The dump of UDP header in hexadecimal format is as follows: BC82D00D002B001D obtain the: (i) Source port number (ii) Destination port number (iii) Total length (iv) Packet direction. **[4 M]**

Ans. Refer to Section 3.2.

4. Compare TCP with UDP on any four points. **[4 M]**

Ans. Following table compare TCP and UDP:

Sr. No.	Characteristics	TCP	UDP
1.	Connection	TCP is connection oriented Protocol.	UDP is connection less protocol.

Contd...

2.	Reliability	It provides reliable delivery of messages.	It provides unreliable delivery of messages.
3.	Error Handling	TCP makes checks for errors and reporting.	UDP does error checking but no reporting.
4.	Flow controlling	TCP has flow control.	UDP has no flow control.
5.	Data transmission order	TCP gives guarantee that the order of the data at the receiving end is the same as the sending end.	No guarantee of the data transmission order.
6.	Header Size	20 bytes.	8 bytes.
7.	Acknowledgment	TCP acknowledges the data reception.	UDP has no acknowledgment Section.
8.	Use	Used where reliability is important.	Used where time sensitivity is more important.
9.	Data Interface to application	Stream-based: No particular structure for data.	Message based data: Data sent in discrete packages by application.
10.	Overhead	Low.	Very low.
11.	Speed	High.	Very high.
12.	Application	FTP, Telnet, SMTP, DNS, HTTP.	DNS, BOOTP, DHCP, TFTP, RIP.

5. Explain how TCP connections are established using 3-way handshake. **[6 M]**

Ans. Refer to Section 3.3.4.4.

Summer 2024

1. Enlist applications of UDP (any two). **[2 M]**

Ans. Refer to Section 3.2.4.

2. Differentiate between TCP and UDP. **[4 M]**

Ans. Refer to Q. 4 of Winter 2023.

3. Describe the packet format of SCTP. **[4 M]**

Ans. Refer to Section 3.5.3.

4. Explain TCP connection establishment using three-way handshake mechanism. **[4 M]**

Ans. Refer to Section 3.3.4.4.

5. Explain association establishment process in SCTP. **[6 M]**

Ans. Refer to Section 3.5.

Winter 2024

1. Enlist any two services provided by UDP. **[2 M]**

Ans. Refer to Section 3.2.2.

2. Draw a neat labelled sketch of TCP Header format. **[2 M]**

Ans. Refer to Section 3.3.3.

3. Compare TCP and UDP w.r.t (i) Connection setup (ii) Retransmission of data (iii) Reliability and acknowledgement (iv) Flow control and error control. **[4 M]**

Ans. Refer to Q. 4 of Winter 2023.

4. Explain connection establishment process of TCP. **[4 M]**

Ans. Refer to Section 3.3.4.1.

5. Describe the flow control in TCP. **[4 M]**

Ans. Refer to Section 3.3.5.1.

6. Explain following services provided by TCP with neat labelled sketch of each service: (i) Stream delivery service (ii) Sending and receiving buffer (iii) Bytes and segments. **[6 M]**

Ans. Refer to Q. 4 of Winter 2023.

7. How flow control, congestion control and error control is handled by SCTP? **[6 M]**

Ans. Refer to Section 3.5.2.

❖❖❖