

UNIT – IV TRANSPORT LAYER

1. Transport Services

1.1. Services provided to the upper layer

1.2. Transport Service Primitives

2. Elements of Transport Protocols

2.1. Addressing

2.2. Connection Establishment

2.3. Connection Release

2.4. Error Control and Flow Control

2.5. Multiplexing

2.6. Crash Recovery

3. Connection management

3.1. Connection Establishment

3.2. Connection Release

4. UDP protocols

4.1. Header

4.2. UDP Services

5. TCP Protocols

5.1. TCP Segment Header

5.2. TCP Connection Establishment

5.3. TCP Data Transfer

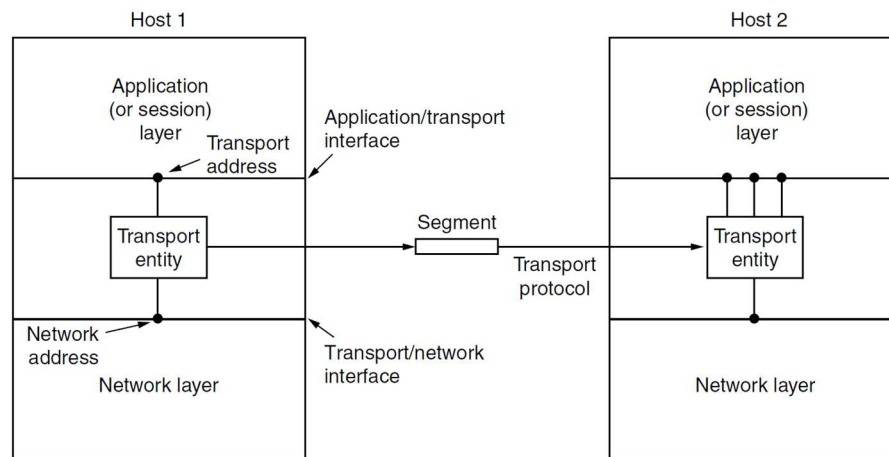
5.4. TCP Connection Release

5.5. UDP v/s TCP

1. TRANSPORT SERVICES:

1.1. Services provided to the upper layer:

- The main goal of the Transport layer is to provide efficient, reliable and cost-effective data transmission.
- The Transport layer should use the services provided by the network layer to achieve efficient, reliable and cost-effective data transmission.
- Transport Entity does this work.
- Figure below shows how the network layer, transport layer and the application layer are connected.



- There are 2 types of Transport Services
 - Connection-Oriented Transport Service: we have 3 phases *Connection Establishment, Data Transfer, Connection Release*
 - Connectionless Transport Service
- If, in a connectionless network, packets are lost or mangled, the transport entity can detect the problem and compensate for it by using retransmissions.
- If, in a connection-oriented network, a transport entity is informed halfway through a long transmission that its network connection has been abruptly terminated, with no indication of what has happened to the data currently in transit, it can set up a new network connection to the remote transport entity.

1.2. Transport Service Primitives

Transport Service Primitives are the basic operations (or functions) provided by the transport layer to enable communication between processes on different systems in a computer network.

They define how a transport service user (like an application) interacts with the transport service provider (the transport layer protocol) such as TCP or UDP

A transport service primitive is a function call that a process uses to request a specific service from the transport layer — such as establishing a connection, sending data, or terminating a connection

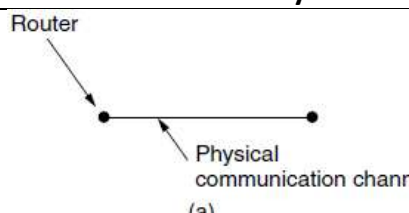
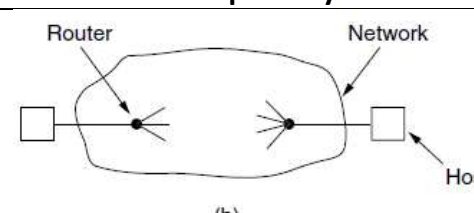
Main Transport Service Primitives

Primitive	Purpose / Function	Description
LISTEN	Wait for a connection	The server process waits (listens) for a client to request a connection.
CONNECT	Establish a connection	The client process requests a connection to a remote server.
SEND	Transmit data	Once connected, the process sends data across the established connection.
RECEIVE	Accept incoming data	The process waits to receive data from the remote process.
DISCONNECT / CLOSE	Terminate a connection	Ends the established connection between two processes.

2. ELEMENTS OF TRANSPORT PROTOCOLS

The transport service is implemented by a **transport protocol** used between the two transport entities. Transport protocols resemble the Data Link protocols except the dissimilarities between the environment in which the two protocols work.

At the data link layer(fig-a), two routers communicate directly via a physical channel, whether wired or wireless, whereas at the transport layer(fig-b), this physical channel is replaced by the entire network.

Data –Link Layer	Transport Layer
 <p>Router</p> <p>Physical communication channel</p> <p>(a)</p>	 <p>Router</p> <p>Network</p> <p>Host</p> <p>(b)</p>
Two routers communicate directly via a physical channel, whether wired or wireless	The physical channel is replaced by the entire network
Over point-to-point links such as wires or optical fiber, it is usually not necessary for a router to specify which router it wants to talk to. i.e., each outgoing line leads to a particular router	Explicit addressing of destinations is required
The process of establishing a connection over the wire of is simple: the other end is always there	Initial connection establishment is complicated

The elements of Transport layer are:

- Addressing
- Connection Establishment
- Connection Release
- Error Control and Flow Control
- Multiplexing
- crash Recovery

2.1. Addressing:

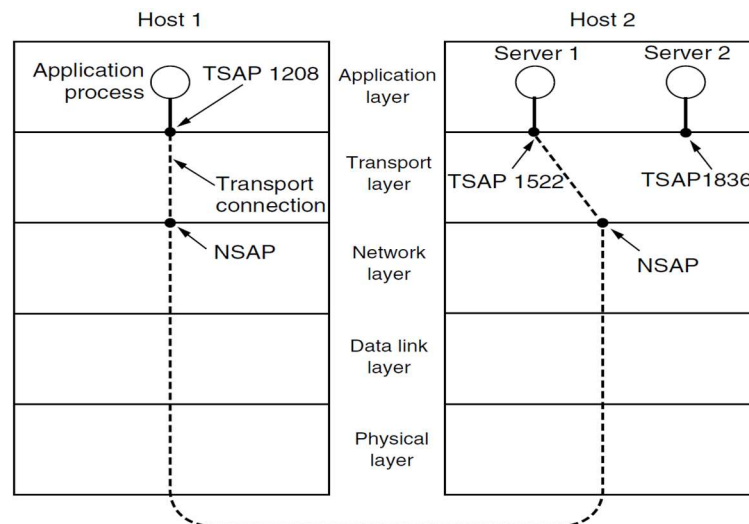
When an application (e.g., a user) process wishes to set up a connection to a remote application process, it must specify which one to connect to.

In the Internet, these endpoints are called **PORTS**. We will use the generic term **TSAP (Transport Service Access Point)** to mean a specific endpoint in the transport layer endpoints in the network layer called **NSAPs (Network Service Access Points)**. IP addresses are examples of NSAPs.

The figure below, illustrates the relationship between the NSAPs, the TSAPs, and a transport connection.

Application processes, both clients and servers, can attach themselves to a local TSAP to establish a connection to a remote TSAP

These connections run through NSAPs on each host, as shown. The purpose of having TSAPs is that in some networks, each computer has a single NSAP, so some way is needed to distinguish multiple transport endpoints that share that NSAP.

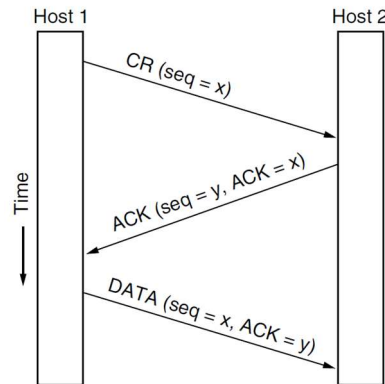


The above figure is explained as:

- A mail server process attaches itself to TSAP 1522 on host 2 to wait for an incoming call. How a process attaches itself to a TSAP is outside the networking model and depends entirely on the local operating system.
- An application process on host 1 wants to send an email message, so it attaches itself to TSAP 1208 and issues a CONNECT request. The request specifies TSAP 1208 on host 1 as the source and TSAP 1522 on host 2 as the destination. This action ultimately results in a transport connection being established between the application process and the server.
- The application process sends over the mail message.
- The mail server responds to say that it will deliver the message
- The transport connection is released.

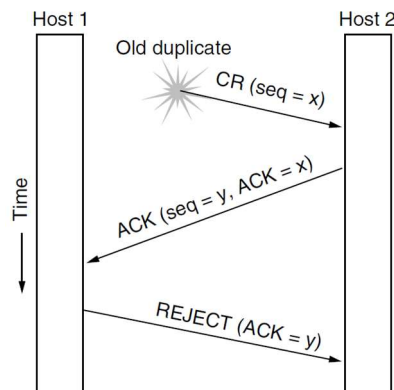
2.2. Connection Establishment

- Connection Establishment in Transport layer uses a 3-way handshaking mechanism.
- Fig below shows a 3-way handshaking under normal setup procedure when host 1 initiates.



The operation is as follows

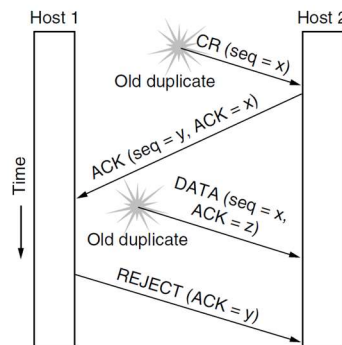
- ✓ Host 1 chooses a sequence number, x , and sends a CONNECTION REQUEST segment containing it to host 2
 - ✓ Host 2 replies with an ACK segment acknowledging x and announcing its own initial sequence number, y .
 - ✓ Finally, host 1 acknowledges host 2's choice of an initial sequence number in the first data segment that it sends.
- Fig below shows a 3-way handshaking under the presence of delayed duplicate control segments when host 1 initiates.



The operation is as follows

- ✓ The first segment is a delayed duplicate CONNECTION REQUEST from an old connection.
- ✓ This segment arrives at host 2 without host 1's knowledge.
- ✓ Host 2 reacts to this segment by sending host 1 an ACK segment, in effect asking for verification that host 1 was indeed trying to set up a new connection.
- ✓ When host 1 rejects host 2's attempt to establish a connection, host 2 realizes that it was tricked by a delayed duplicate and abandons the connection.
- ✓ In this way, a delayed duplicate does no damage.
- ✓

- Fig below shows a 3-way handshaking under delayed CONNECTION REQUEST and an ACK are floating around in the subnet



The operation is as follows

- ✓ Like the previous example, host 2 gets a delayed CONNECTION REQUEST and replies to it.
- ✓ At this point, it is crucial to realize that host 2 has proposed using y as the initial sequence number for host 2 to host 1 traffic, knowing full well that no segments containing sequence number y or acknowledgements to y are still in existence.
- ✓ When the second delayed segment arrives at host 2, the fact that z has been acknowledged rather than y tells host 2 that this, too, is an old duplicate

2.3. Connection Release

There are 2 types of terminating a connection.

a. Asymmetric Release

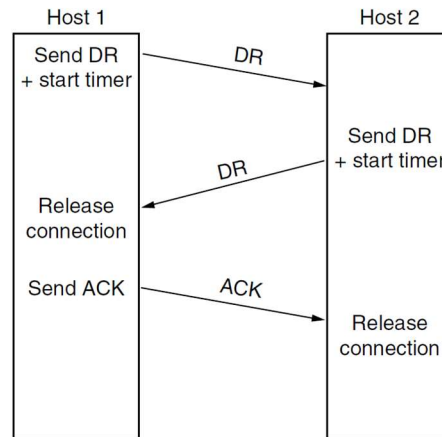
- In asymmetric release, the transmitter sends a single connection release message to the other side.
- Once the message is processed the entire connection is considered closed — for both directions of communication.
- The other side cannot send any more data now
- It's a faster release, but not reliable.

b. Symmetric Release

- In networking, symmetric release means that each endpoint of a transport connection independently closes its own half of the connection
- The connection is fully terminated only when both sides have confirmed that they are finished sending data.
- TCP (Transmission Control Protocol) is an example for Symmetric release.
- It is the most reliable way of connection release.

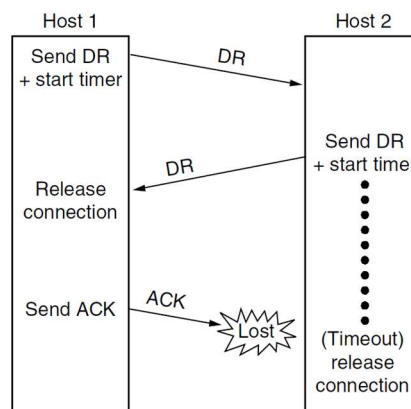
The following explain the 3-way handshaking for connection release.

Figure below shows a normal case of connection release



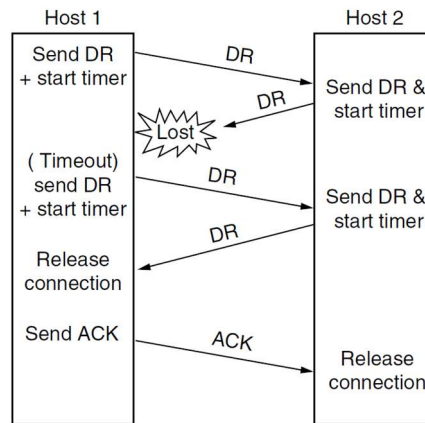
- ✓ one of the users sends a DR (DISCONNECTION REQUEST) segment to initiate the connection release.
- ✓ When it arrives, the recipient sends back a DR segment and starts a timer, just in case its DR is lost.
- ✓ When this DR arrives, the original sender sends back an ACK segment and releases the connection.
- ✓ Finally, when the ACK segment arrives, the receiver also releases the connection.

Figure below shows a case where the ACK is lost



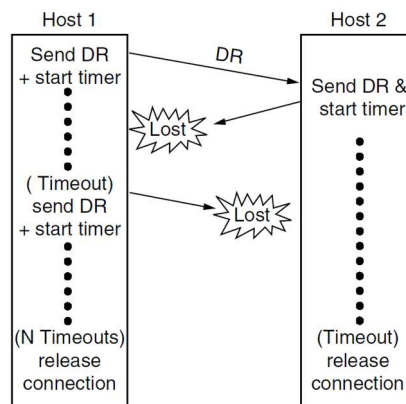
- ✓ one of the users sends a DR (DISCONNECTION REQUEST) segment to initiate the connection release.
- ✓ When it arrives, the recipient sends back a DR segment and starts a timer, just in case its DR is lost.
- ✓ When this DR arrives, the original sender sends back an ACK segment and releases the connection.
- ✓ If the final ACK segment is lost, as shown the situation is saved by the timer. When the timer expires, the connection is released.

Figure below shows a case where the second DR is lost



- ✓ Host 1 initiates the Disconnection request(DR) and starts the timer.
- ✓ Host 2 receives DR and initiates its own DR and starts its timer
- ✓ Considering this DR is lost, Host 1 timer elapses and will start all over again.

Figure below shows a case where the second DR is lost except that now we assume all the repeated attempts to retransmit the DR also fail due to lost segments. After N retries, the sender just gives up and releases the connection. Meanwhile, the receiver times out and also exits.



2.4. Error Control and Flow Control

Error control is ensuring that the data is delivered with the desired level of reliability, usually that all of the data is delivered without any errors. Flow control is keeping a fast transmitter from overrunning a slow receiver.

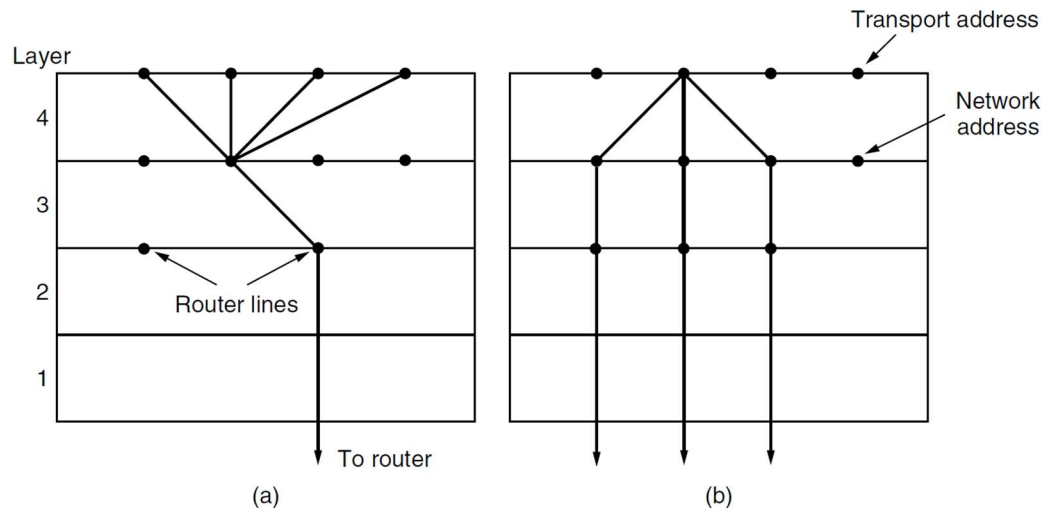
The solutions that are used at the transport layer are

- 1) A frame carries an error-detecting code (e.g., a CRC or checksum) that is used to check if the information was correctly received.

- 2) A frame carries a sequence number to identify itself and is retransmitted by the sender until it receives an acknowledgement of successful receipt from the receiver. This is called **ARQ (Automatic Repeat reQuest)**.
- 3) There is a maximum number of frames that the sender will allow to be outstanding at any time, pausing if the receiver is not acknowledging frames quickly enough. If this maximum is one packet the protocol is called **stop-and-wait**. Larger windows enable pipelining and improve performance on long, fast links.
- 4) The **sliding window** protocol combines these features and is also used to support bidirectional data transfer.

2.5. Multiplexing

- Multiplexing, or sharing several conversations over connections, virtual circuits, and physical links plays a role in several layers of the network architecture.
- In the transport layer, the need for multiplexing can arise in a number of ways. For example, if only one network address is available on a host, all transport connections on that machine have to use it. When a segment comes in, some way is needed to tell which process to give it to. This situation, called **multiplexing**, and is shown below.



- In this figure, four distinct transport connections all use the same network connection (e.g., IP address) to the remote host. Multiplexing can also be useful in the transport layer for another reason. Suppose, for example, that a host has multiple network paths that it can use.
- If a user needs more bandwidth or more reliability than one of the network paths can provide, a way out is to have a connection that distributes the traffic among multiple network paths on a round-robin basis, as indicated in Fig. (b). This is called **inverse multiplexing**.

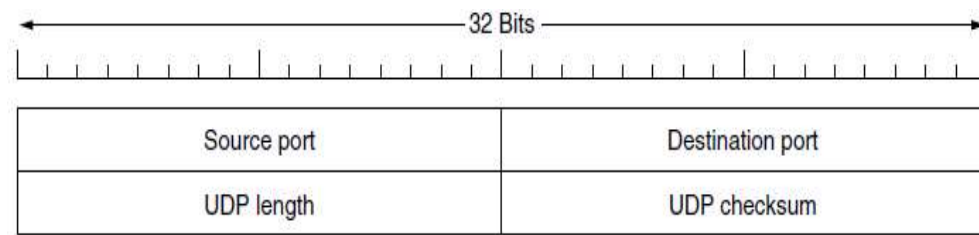
3. CONNECTION MANAGEMENT:

Refer 2.2 and 2.3

4. UDP PROTOCOLS

4.1. UDP Header

- The Internet protocol suite supports a connectionless transport protocol called UDP (User Datagram Protocol).
- UDP transmits segments consisting of an 8-byte header followed by the payload. The header is shown below. The two **ports** serve to identify the endpoints within the source and destination machines. When a UDP packet arrives, its payload is handed to the process attached to the destination port



- The source port is primarily needed when a reply must be sent back to the source. By copying the *Source port* field from the incoming segment into the *Destination port* field of the outgoing segment, the process sending the reply can specify which process on the sending machine is to get it.
- The *UDP length* field includes the 8-byte header and the data. The minimum length is 8 bytes, to cover the header. The maximum length is 65,515 bytes, which is lower than the largest number that will fit in 16 bits because of the size limit on IP packets.
- An optional *Checksum* is also provided for extra reliability. It checksums the header, the data, and a conceptual IP pseudo header. When performing this computation, the *Checksum* field is set to zero and the data field is padded out with an additional zero byte if its length is an odd number.
- The checksum algorithm is simply to add up all the 16-bit words in one's complement and to take the one's complement of the sum. As a consequence, when the receiver performs the calculation on the entire segment, including the *Checksum* field, the result should be 0.

4.2. UDP Services:

UDP (User Datagram Protocol) is a connectionless, lightweight, and fast transport layer protocol. It provides minimal services compared to TCP, mainly focusing on speed and simplicity rather than reliability

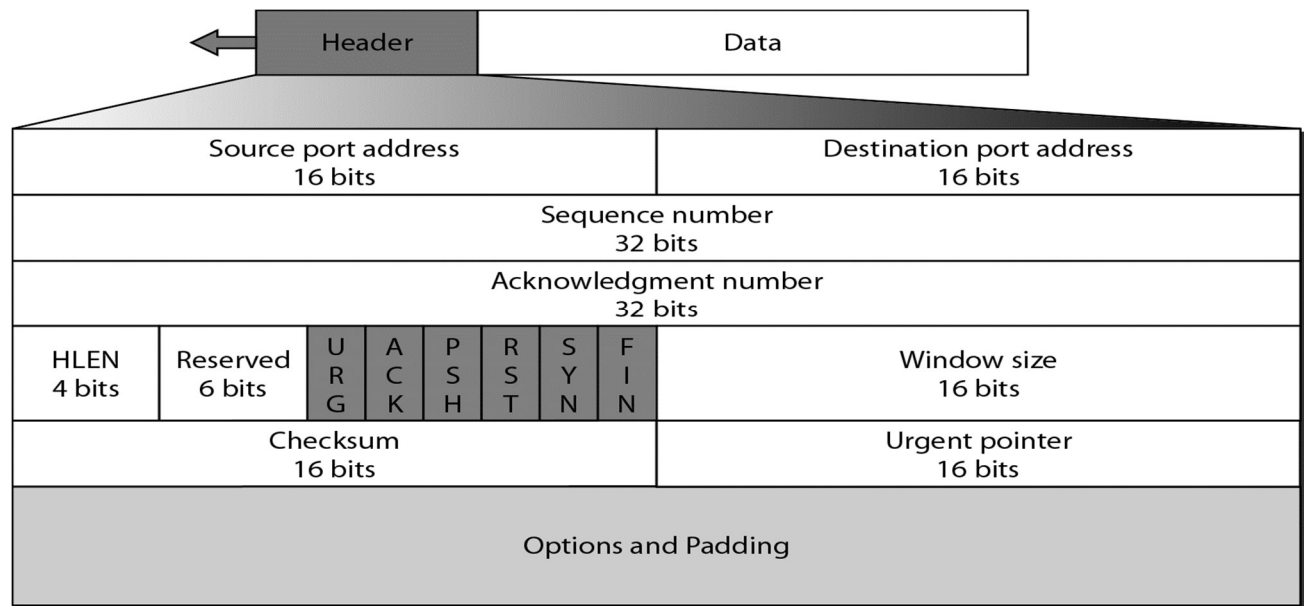
Main Services Provided by UDP

Service	Description
Process-to-Process Communication	UDP allows data to be sent from one specific process (identified by a port number) on a host to another specific process on another host. → Each process is identified by a (IP address, Port number) pair.
Connectionless Service	UDP does not establish a connection before data transfer. → Each message (datagram) is sent independently, without setup or teardown phases.
Unreliable Service	UDP provides no guarantee of delivery, ordering, or duplicate protection. → If a packet is lost, corrupted, or arrives out of order, UDP does not correct it.
Low Overhead / Fast Transmission	UDP adds only an 8-byte header, making it much faster and efficient for real-time or short transactions.
Error Detection (Checksum)	UDP includes a 16-bit checksum to detect errors in the header and data. → If an error is detected, the datagram is discarded silently.
Multiplexing and Demultiplexing	UDP uses port numbers to allow multiple applications to send and receive data over the same IP network. → Multiplexing: Combining data from multiple sources. → Demultiplexing: Delivering received data to the correct application.
No Flow or Congestion Control	UDP does not control data rate or congestion — it simply sends packets as fast as the application allows. → This is useful for time-sensitive applications (e.g., live video).

5. TCP PROTOCOLS:

5.1. TCP Segment Header

- A key feature of TCP, and one that dominates the protocol design, is that every byte on a TCP connection has its own 32-bit sequence number
- The sending and receiving TCP entities exchange data in the form of segments. A **TCP segment** consists of a fixed 20-byte header (plus an optional part) followed by zero or more data bytes.
- The TCP software decides how big segments should be. It can accumulate data from several writes into one segment or can split data from one write over multiple segments. Two limits restrict the segment size. First, each segment, including the TCP header, must fit in the 65,515- byte IP payload. Second, each link has an **MTU (Maximum Transfer Unit)**. Each segment must fit in the MTU at the sender and receiver so that it can be sent and received in a single, unfragmented packet



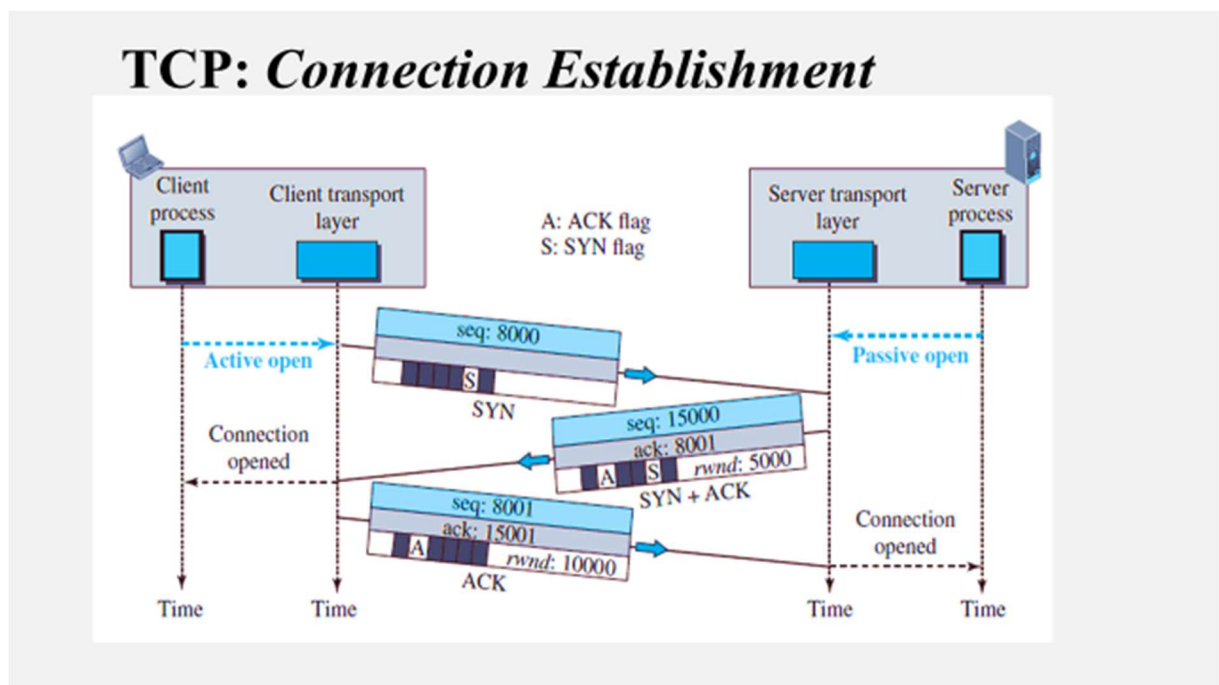
FIELD	SIZE (BITS)	DESCRIPTION
Source Port	16	Port number of the sending application.
Destination Port	16	Port number of the receiving application.
Sequence Number	32	Indicates the position of the first byte of data in this segment in the overall data stream.
Acknowledgment Number	32	Indicates the next byte expected from the sender. Used for acknowledgment.
Header Length (HLEN)	4	Specifies the length of the TCP header in 32-bit words (minimum value = 5).
Reserved	6	Reserved for future use; must be zero.
Control Flags (6 bits)	6	Control various TCP operations:
→ URG	Urgent pointer field is valid.	
→ ACK	Acknowledgment field is valid.	
→ PSH	Push data to the receiving application immediately.	
→ RST	Reset the connection.	
→ SYN	Synchronize sequence numbers (used in connection setup).	
→ FIN	Sender has finished sending data (used to terminate connection).	
Window Size	16	Specifies how many bytes the sender is willing to receive (for flow control).
Checksum	16	Error-checking for header + data. Ensures integrity.
Urgent Pointer	16	Points to the last byte of urgent data (used when URG flag = 1).
Options and Padding	Variable (0–40 bytes)	Used for additional features like Maximum Segment Size (MSS), Window Scaling, etc.
Data	Variable	Actual application data being transferred.

5.2. TCP Connection Establishment

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. This implies that each party must initialize communication and get approval from the other party before any data are transferred. The connection establishment in TCP is called three-way handshaking.

For 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.

Consider the TCP Connection Establishment as shown below



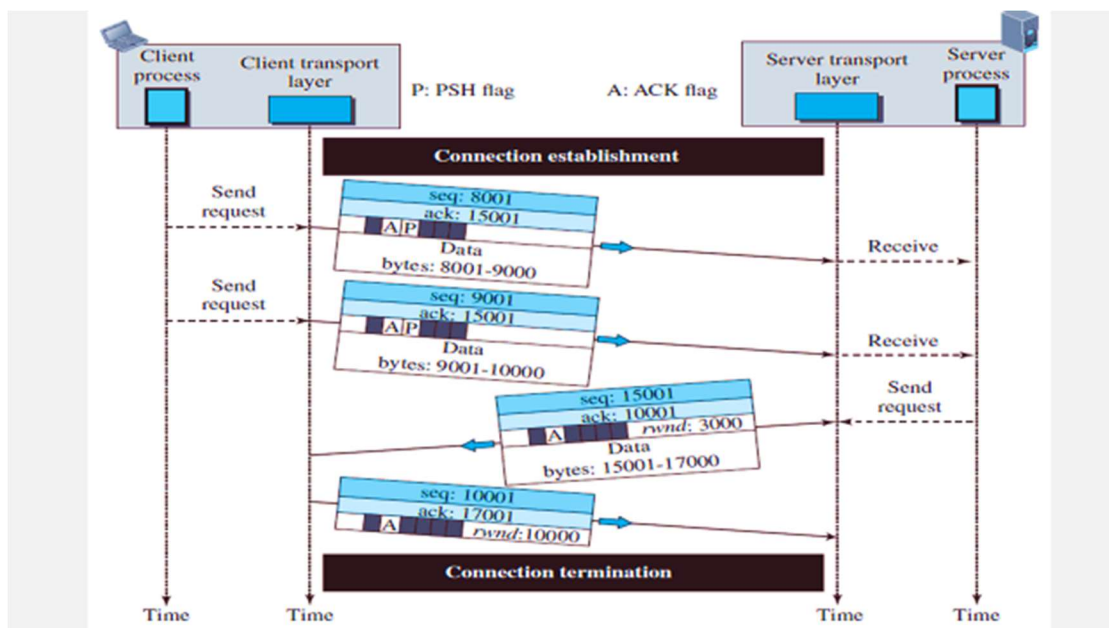
- The process starts with the server.
- The server program tells its TCP that it is ready to accept a connection. This request is called a *passive open*.
- The server TCP is ready to accept a connection from any machine in the world.
- But it cannot make the connection itself.
- The client program issues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

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. This sequence number is called the initial sequence number (ISN). 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.
- The server sends the second segment, a SYN + ACK segment with two flag bits set as: 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. Because the segment contains an acknowledgment, it also needs to define the receive window size, *rwnd* (to be used by 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 ACK segment does not consume any sequence numbers if it does not carry data,

5.3. TCP: Data Transfer

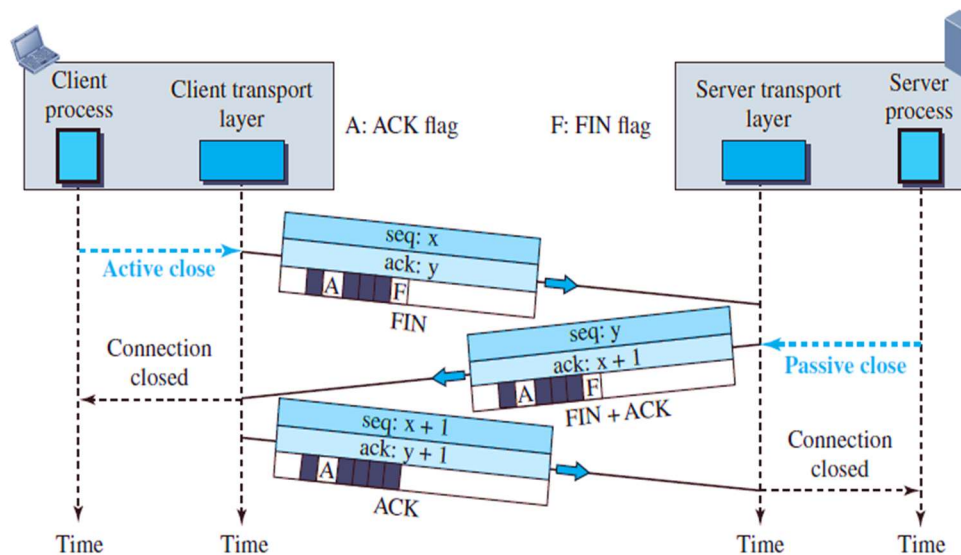
- After connection is established, bidirectional data transfer can take place. The client and server can send data and acknowledgments in both directions.
- Data traveling in the same direction as an acknowledgment are carried on the same segment. The acknowledgment is *piggybacked* with the data.
- Figure shows an example.



- In this example, 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.
- Note the values of the sequence and acknowledgment numbers.
- The data segments sent by the client have the PSH (push) flag set so that the server TCP knows 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

5.4. TCP: Connection Termination

- Either of the two parties involved in exchanging data (client or server) can close the connection, although it is usually initiated by the client.
- Most implementations today allow *three-way handshaking* for connection termination, as shown in Figure.



1. In this 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. It is only a control segment, it consumes only one sequence number because it needs to be acknowledged.
2. 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. It consumes only one sequence number because it needs to be acknowledged.
3. 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.

5.5. TCP v/s UDP

TCP	UDP
It stands for Transmission Control Protocol.	It stands for User Datagram Protocol.
It is a connection-oriented protocol, which means that the connection needs to be established before the data is transmitted over the network.	It is a connectionless protocol, which means that it sends the data without checking whether the system is ready to receive or not.
TCP is a reliable protocol as it provides assurance for the delivery of data packets	UDP is an unreliable protocol as it does not take the guarantee for the delivery of packets.
TCP is slower than UDP as it performs error checking, flow control, and provides assurance for the delivery of	UDP is faster than TCP as it does not guarantee the delivery of data packets
The size of TCP header is 20 to 60 bytes	The size of the UDP header is 8 bytes
It follows the flow control mechanism in which too many packets cannot be sent to the receiver at the same time.	This protocol follows no such mechanism.
TCP performs error checking by using a checksum. When the data is corrected, then the data is retransmitted to the receiver	It does not perform any error checking, and also does not resend the lost data packets.
This protocol is mainly used where a secure and reliable communication process is required, like military services, web browsing, and e-mail	This protocol is used where fast communication is required and does not care about the reliability like VoIP, game streaming, video and music streaming, etc