Unit IV - Transport Layer

Introduction

The transport layer is the heart of the protocol hierarchy. The transport layer builds on the network layer to provide data transport from a process on a source machine to a process on a destination machine with a desired level of reliability that is independent of the physical networks currently in use. It provides the abstractions that applications need to use the network.

THE TRANSPORT SERVICE

Services Provided to the Upper Layers

The ultimate goal of the transport layer is to provide efficient, reliable, and cost-effective data transmission service to its users, normally processes in the application layer. To achieve this, the transport layer makes use of the services provided by the network layer. The software and/or hardware within the transport layer that does the work is called the transport entity. The transport entity can be located in the operating system kernel, in a library package bound into network applications, in a separate user process, or even on the network interface card.

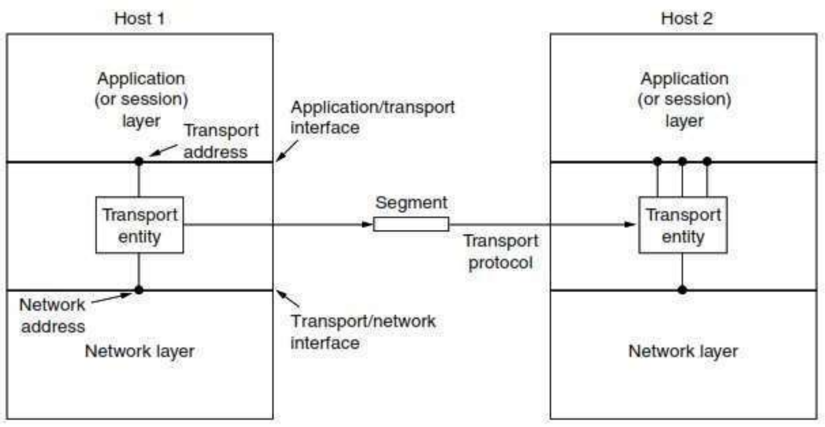


Figure 6-1. The network, transport, and application layers.

There are two types of network service

- Connection-oriented
- Connectionless

Connectionless versus Connection-Oriented Service

A transport-layer protocol can be either connectionless or connection-oriented.

Connectionless Service

In a connectionless service, the packets are sent from one party to another with no need for connection establishment or connection release. The packets are not numbered, so they may be delayed, lost, or arrive out of sequence. There is no acknowledgement either. One of the transport-layer protocols, User Datagram Protocol (UDP), is connectionless.

Connection-Oriented Service

In a connection-oriented service, a connection is first established between the sender and the receiver. Data are then transferred. At the end, the connection is released. Transmission Control Protocol (TCP) is a connection- oriented service.

Reliable versus Unreliable

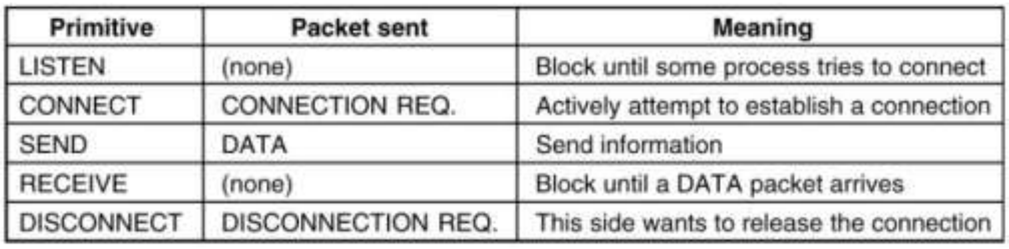
The transport-layer service can be reliable or unreliable. If the application-layer protocol needs reliability, a reliable transport-layer protocol is used to implement flow and error control. This means a slower and more complex service. However, if the application program does not need reliability because it uses its own flow and error control mechanism or if it needs fast service or the nature of the service does not demand flow and error control (e.g. real time applications), an unreliable protocol can be used. There are two different transport-layer protocols. UDP is connectionless and unreliable; TCP is connection-oriented and reliable.

Transport Service Primitives

- To allow users to access the transport service, the transport layer must provide some operations to application programs, that is, a transport service interface. Each transport service has its own interface.
- The transport service is similar to the network service, but there are also some important differences.
- The main difference is that the network service is intended to model the service offered by real networks. Real networks can lose packets, so the network service is generally unreliable.
- The (connection-oriented) transport service, in contrast, is reliable.

As an example, consider two processes connected by pipes in UNIX. They assume the connection between them is perfect. They do not want to know about acknowledgements, lost packets, congestion, or anything like that. What they want is a 100 percent reliable connection. Process A puts data into one end of the pipe, and process B takes it out of the other.

A second difference between the network service and transport service is whom the services are intended for. The network service is used only by the transport entities. Consequently, the transport service must be convenient and easy to use.

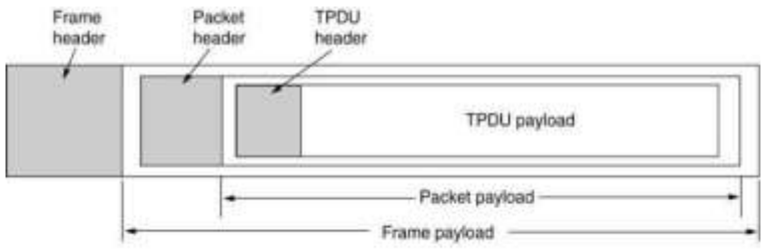


Primitive	Packet sent	Meaning
LISTEN	(none)	Block until some process tries to connect
CONNECT	CONNECTION REQ.	Actively attempt to establish a connection
SEND	DATA	Send information
RECEIVE	(none)	Block until a DATA packet arrives
DISCONNECT	DISCONNECTION REQ.	This side wants to release the connection

Figure. The primitives for a simple transport service.

Eg: Consider an application with a server and a number of remote clients.

1. The server executes a "LISTEN" primitive by calling a library procedure that makes a System call to block the server until a client turns up.
2. When a client wants to talk to the server, it executes a "CONNECT" primitive, with "CONNECTION REQUEST" TPDU sent to the server.
3. When it arrives, the TE unblocks the server and sends a "CONNECTION ACCEPTED" TPDU back to the client.
4. When it arrives, the client is unblocked and the connection is established. Data can now be exchanged using "SEND" and "RECEIVE" primitives.
5. When a connection is no longer needed, it must be released to free up table space within the 2 transport entries, which is done with "DISCONNECT" primitive by sending "DISCONNECTION REQUEST"



The nesting of TPDUs, packets, and frames.

- The term segment for messages sent from transport entity to transport entity.
- TCP, UDP and other Internet protocols use this term. Segments (exchanged by the transport layer) are contained in packets (exchanged by the network layer).
- These packets are contained in frames (exchanged by the data link layer).When a frame arrives, the data link layer processes the frame header and, if the destination address matches for local delivery, passes the contents of the frame payload field up to the network entity.
- The network entity similarly processes the packet header and then passes the contents of the packet payload up to the transport entity. This nesting is illustrated in Fig. 4.2.

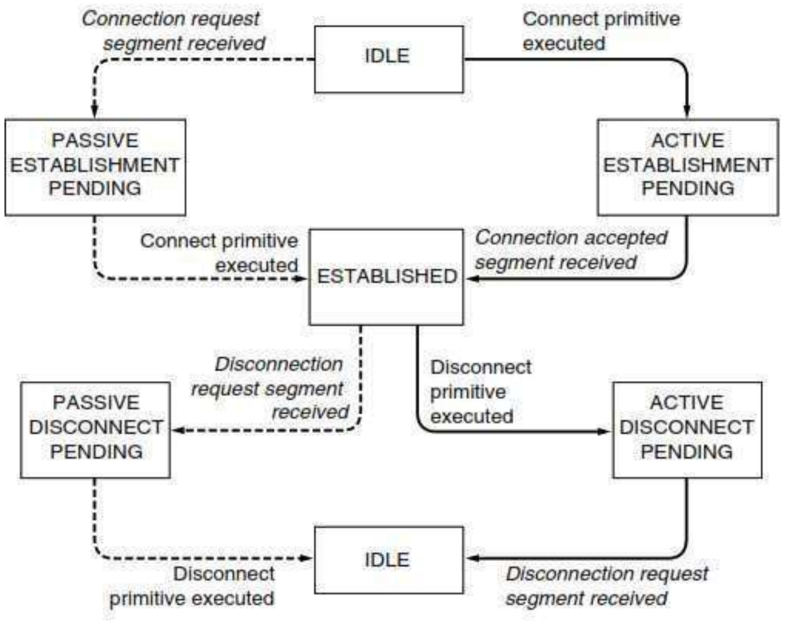
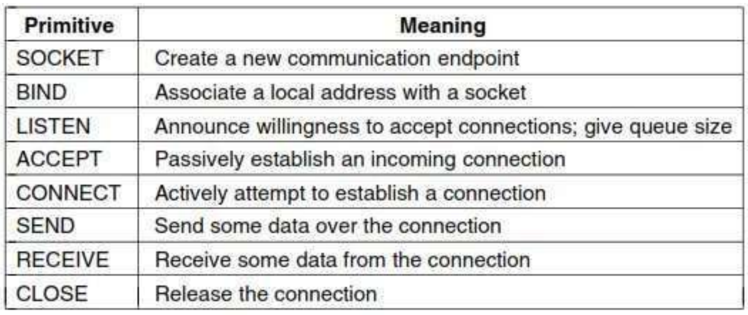

Connection request segment received
IDLE
Connect primitive executed
PASSIVE ESTABLISHMENT PENDING
ACTIVE ESTABLISHMENT PENDING
Connect primitive executed
ESTABLISHED
Connection accepted segment received
Disconnection request segment received
Disconnect primitive executed
PASSIVE DISCONNECT PENDING
ACTIVE DISCONNECT PENDING
Disconnect primitive executed
IDLE
Disconnection request segment received

Figure 4.3 - A state diagram for a simple connection management scheme. Transitions labelled in italics are caused by packet arrivals. The solid lines show the client's state sequence. The dashed lines show the server's state sequence.

In fig. 4.3 each transition is triggered by some event, either a primitive executed by the local transport user or an incoming packet. For simplicity, we assume here that each TPDU is separately acknowledged. We also assume that a symmetric disconnection model is used, with the client going first. Please note that this model is quite unsophisticated. We will look at more realistic models later on.

These primitives are socket primitives used in Berkley UNIX for TCP.

The socket primitives are mainly used for TCP. These sockets were first released as part of the Berkeley UNIX 4.2BSD software distribution in 1983. They quickly became popular. The primitives are now widely used for Internet programming on many operating systems, especially UNIX -based systems, and there is a socket-style API for Windows called "winsock."



Primitive	Meaning
SOCKET	Create a new communication endpoint
BIND	Associate a local address with a socket
LISTEN	Announce willingness to accept connections; give queue size
ACCEPT	Passively establish an incoming connection
CONNECT	Actively attempt to establish a connection
SEND	Send some data over the connection
RECEIVE	Receive some data from the connection
CLOSE	Release the connection

Figure 6-5. The socket primitives for TCP.

The first four primitives in the list are executed in that order by servers.

The SOCKET primitive creates a new endpoint and allocates table space for it within the transport entity. The parameter includes the addressing format to be used, the type of service desired and the protocol. Newly created sockets do not have network addresses.

- The BIND primitive is used to connect the newly created sockets to an address. Once a server has bound an address to a socket, remote clients can connect to it.
- The LISTEN call, which allocates space to queue incoming calls for the case that several clients try to connect at the same time.
- The server executes an ACCEPT primitive to block waiting for an incoming connection.

Some of the client side primitives are. Here, too, a socket must first be created

- The CONNECT primitive blocks the caller and actively starts process. When it completes, the client process is unblocked and the connection is established.
- Both sides can now use SEND and RECEIVE to transmit and receive data over the full-duplex connection.
- Connection release with sockets is symmetric. When both sides have executed a CLOSE primitive, the connection is released.

ELEMENTS OF TRANSPORT PROTOCOLS

The transport service is implemented by a transport protocol used between the two transport entities. The transport protocols resemble the data link protocols. Both have to deal with error control, sequencing, and flow control, among other issues. The difference transport protocol and data link protocol depends upon the environment in which they are operated.

These differences are due to major dissimilarities between the environments in which the two protocols operate, as shown in Fig.

At the data link layer, two routers communicate directly via a physical channel, whether wired or wireless, whereas at the transport layer, this physical channel is replaced by the entire network. This difference has many important implications for the protocols.

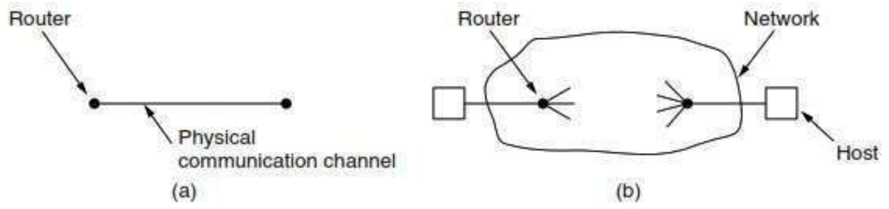


Figure 6-7. (a) Environment of the data link layer. (b) Environment of the transport layer.

In the data link layer, it is not necessary for a router to specify which router it wants to talk to. In the transport layer, explicit addressing of destinations is required.

In the transport layer, initial connection establishment is more complicated, as we will see. Difference between the data link layer and the transport layer is the potential existence of storage capacity in the subnet Buffering and flow control are needed in both layers, but the presence of a large and dynamically varying number of connections in the transport layer may require a different approach than we used in the data link layer.

The transport service is implemented by a transport protocol between the 2 transport entities.

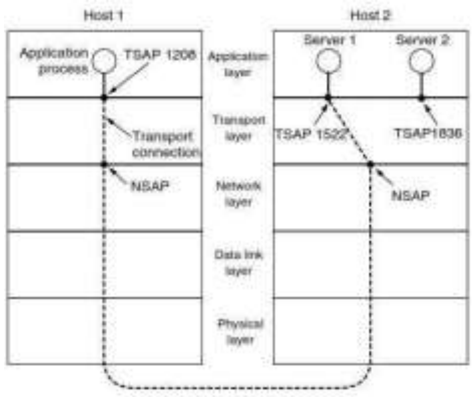


Figure 6-8. TSAPs, NSAPs, and transport connections.

Figure 6.8 illustrates the relationship between the NSAP, TSAP and transport connection. Application processes, both clients and servers, can attach themselves to a 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 end points that share that NSAP.

The duties of transport layer protocols are:

1. Process to Process Communication
2. Addressing
3. Connection Establishment.
4. Connection Release.
5. Error control and flow control
6. Multiplexing.
7. Congestion control

The 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 delivery. The network layer is responsible for delivery of datagrams between two hosts. This is called host-to-host delivery. Real communication takes place between two processes (application programs). We need process-to-process delivery. The transport layer is responsible for process-to-process delivery-the delivery of a packet, part of a message, from one process to another. Figure 4.1 shows these three types of deliveries and their domains

The transport layer is responsible for process-to-process delivery.

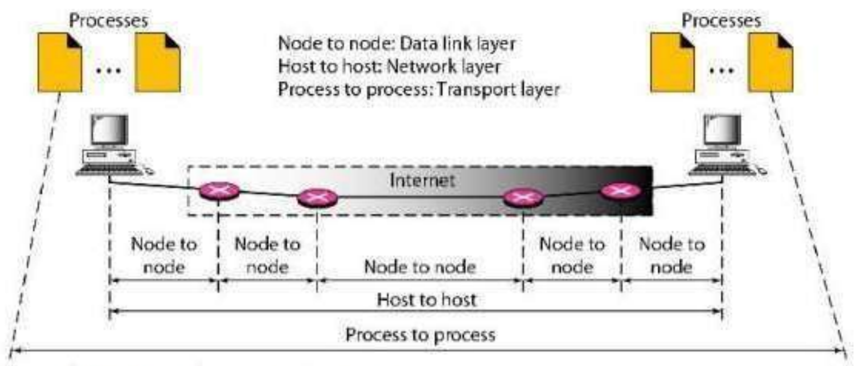


Figure 4.1 Types of data deliveries

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:

Addressing

Whenever we need to deliver something to one specific destination among many, we need an address. At the data link layer, we need a MAC address to choose one node among several nodes if the connection is not point-to- point. A frame in the data link layer needs a Destination MAC address for delivery and a source address for the next node's reply.

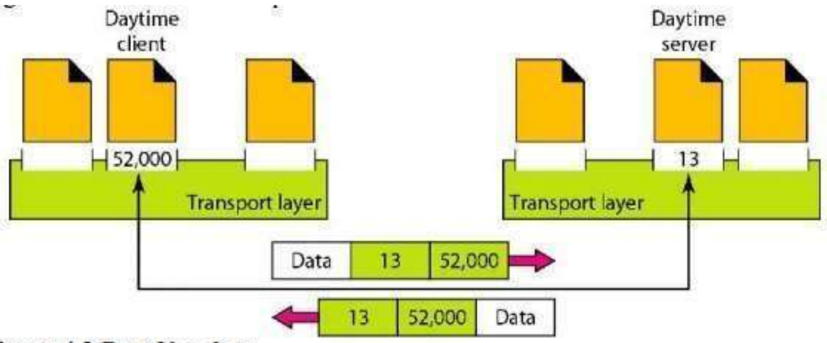


Figure 4.2 Port Numbers

Figure 4.2 shows this concept.

The IP addresses and port numbers play different roles in selecting the final destination of data. The destination IP address defines the host among the different hosts in the world. After the host has been selected, the port number defines one of the processes on this particular host (see Figure 4.3).

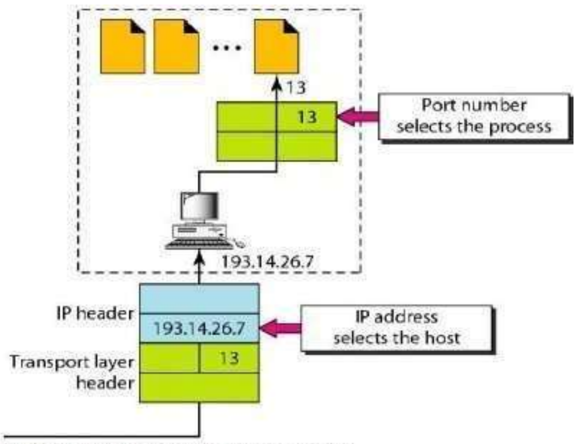


Figure 4.3 IP addresses versus port numbers

IANA Ranges

The IANA (Internet Assigned Number Authority) has divided the port numbers into three ranges: well known, registered, and dynamic (or private), as shown in Figure 4.4.

Well-known ports. The ports ranging from 0 to 1023 are assigned and controlled by lANA. These are the well- known ports.

Registered ports. The ports ranging from 1024 to 49,151 are not assigned or controlled by lANA. They can only be registered with lANA to prevent duplication.

Dynamic ports. The ports ranging from 49,152 to 65,535 are neither controlled nor registered. They can be used by any process. These are the ephemeral ports.

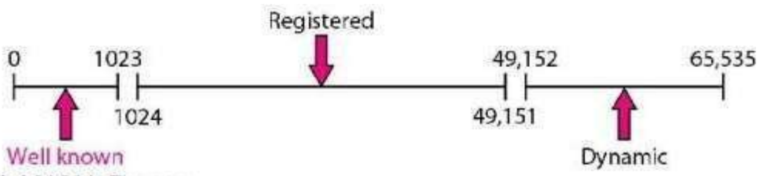


Figure 4.4 IANA Ranges

Socket Addresses

Process-to-process delivery needs two identifiers, IP address and the port number, at each end to make a connection. The combination of an IP address and a port number is called a socket address. The client socket address defines the client process uniquely just as the server socket address defines the server process uniquely (see Figure 4.5).

UDP or TCP header contains the port numbers.

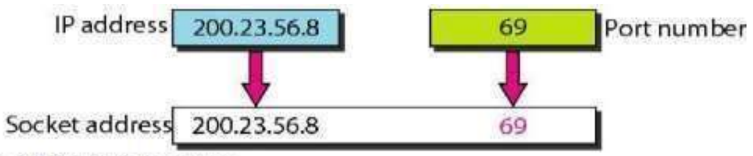


Figure 4.5 Socket Address

Connection establishment:

With packet lifetimes bounded, it is possible to devise a fool proof way to establish connections safely. Packet lifetime can be bounded to a known maximum using one of the following techniques:

- Restricted subnet design
- Putting a hop counter in each packet
- Time stamping in each packet

Using a 3-way hand shake, a connection can be established. This establishment protocol doesn't require both sides to begin sending with the same sequence number.

The first technique includes any method that prevents packets from looping, combined with some way of bounding delay including congestion over the longest possible path. It is difficult, given that internets may range from a single city to international in scope.

The second method consists of having the hop count initialized to some appropriate value and decremented each time the packet is forwarded. The network protocol simply discards any packet whose hop counter becomes zero. The third method requires each packet to bear the time it was created, with the routers agreeing to discard any packet older than some agreed-upon time.

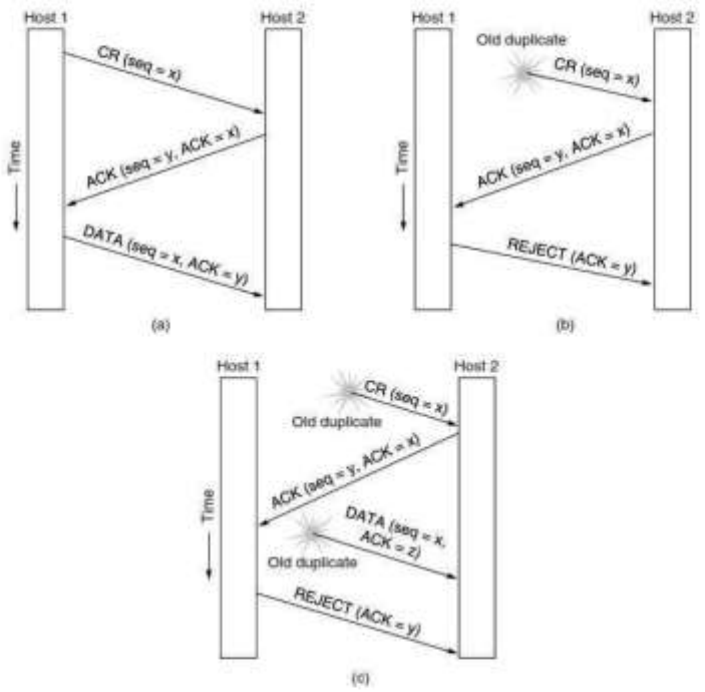


Figure 6-11. Three protocol scenarios for establishing a connection using a three-way handshake. CR denotes CONNECTION REQUEST. (a) Normal operation. (b) Old duplicate CONNECTION REQUEST appearing out of nowhere. (c) Duplicate CONNECTION REQUEST and duplicate ACK.

In fig (A) Tomlinson (1975) introduced the three-way handshake.

- This establishment protocol involves one peer checking with the other that the connection request is indeed current. Host 1 chooses a sequence number, x , and sends a CONNECTION REQUEST segment containing it to host 2. Host 2replies 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

In fig (B) 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 host1an 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.
- The worst case is when both a delayed CONNECTION REQUEST and an ACK are floating around in the subnet.

In fig (C) 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.
- The important thing to realize here is that there is no combination of old segments that can cause the protocol to fail and have a connection set up by accident when no one wants it.

Connection release

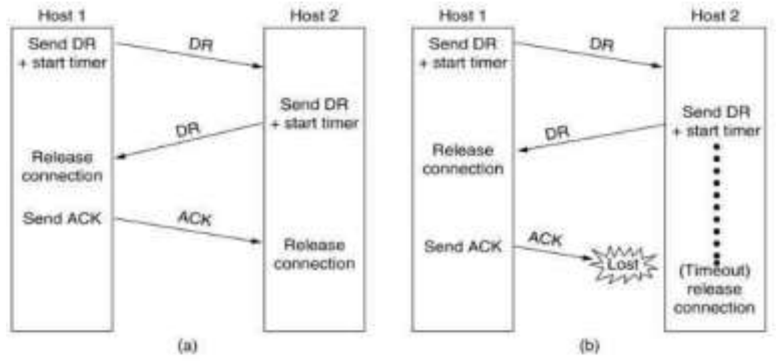
A connection is released using either asymmetric or symmetric variant. But, the improved protocol for releasing a connection is a 3-way handshake protocol.

There are two styles of terminating a connection:

1) Asymmetric release and

 2) Symmetric release.

Asymmetric release is the way the telephone system works: when one party hangs up, the connection is broken. Symmetric release treats the connection as two separate unidirectional connections and requires each one to be released separately.



Four protocol scenarios for releasing a connection. (a) Normal case of a three-way handshake. (b) Final ACK lost.

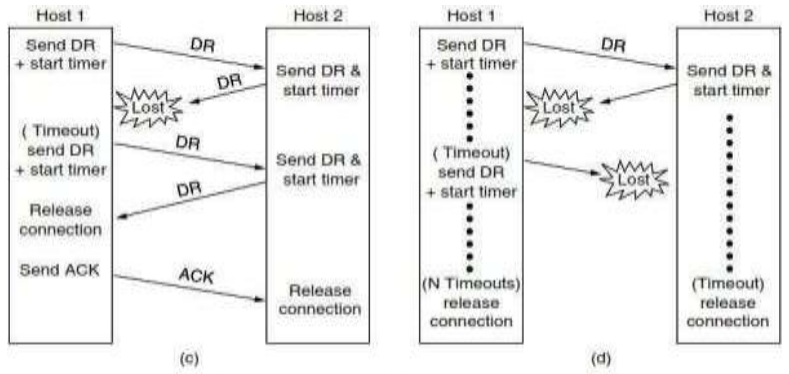


Figure 6-14. Four protocol scenarios for releasing a connection. (a) Normal case of three-way handshake. (b) Final ACK lost. (c) Response lost. (d) Response lost and subsequent DRs lost.

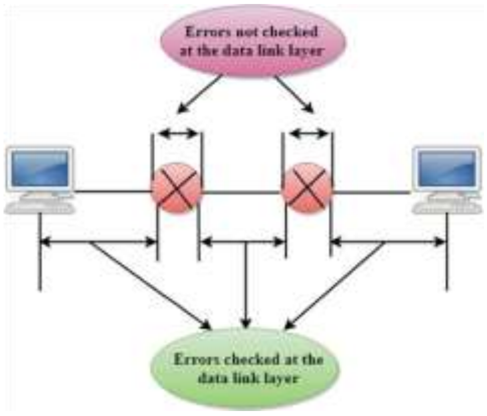
Fig-(a)	Fig-(b)	Fig-(c)	Fig-(d)
One of the user sends a DISCONNECTION REQUEST TPDU in order to initiate connection release. When it arrives, the recipient sends back a DR-TPDU, too, and starts a timer. When this DR arrives, the original sender sends back an ACKTPDU and releases the connection. Finally, when the ACK-TPDU arrives, the receiver also releases the connection.	Initial process is done in the same way as in fig-(a). If the final ACK-TPDU is lost, the situation is saved by the timer. When the timer is expired, the connection is released.	If the second DR is lost, the user initiating the disconnection will not receive the expected response, and will timeout and starts all over again.	Same as in fig (c) except that all repeated attempts to retransmit the DR is assumed to be failed due to lost TPDUs. After 'N' entries, the sender just gives up and releases the connection.

Flow control

Flow control is used to prevent the sender from overwhelming the receiver. If the receiver is overloaded with too much data, then the receiver discards the packets and asking for the retransmission of packets. This increases network congestion and thus, reducing the system performance. The transport layer is responsible for flow control. It uses the sliding window protocol that makes the data transmission more efficient as well as it controls the flow of data so that the receiver does not become overwhelmed. Sliding window protocol is byte oriented rather than frame oriented.

Error Control

- The primary role of reliability is Error Control. In reality, no transmission will be 100 percent error-free delivery. Therefore, transport layer protocols are designed to provide error-free transmission.
- The data link layer also provides the error handling mechanism, but it ensures only node-to-node error-free delivery. However, node-to-node reliability does not ensure the end-to-end reliability.
- The data link layer checks for the error between each network. If an error is introduced inside one of the routers, then this error will not be caught by the data link layer. It only detects those errors that have been introduced between the beginning and end of the link. Therefore, the transport layer performs the checking for the errors end-to-end to ensure that the packet has arrived correctly.


Errors not checked at the data link layer
Errors checked at the data link layer

Sequence Control

- The second aspect of the reliability is sequence control which is implemented at the transport layer.
- On the sending end, the transport layer is responsible for ensuring that the packets received from the upper layers can be used by the lower layers. On the receiving end, it ensures that the various segments of a transmission can be correctly reassembled.

Loss Control

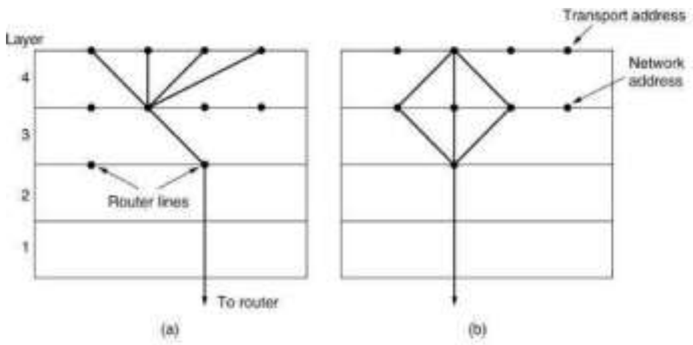
 Loss Control is a third aspect of reliability. The transport layer ensures that all the fragments of a transmission arrive at the destination, not some of them. On the sending end, all the fragments of transmission are given sequence numbers by a transport layer. These sequence numbers allow the receiver’s transport layer to identify the missing segment.

Duplication Control

 Duplication Control is the fourth aspect of reliability. The transport layer guarantees that no duplicate data arrive at the destination. Sequence numbers are used to identify the lost packets; similarly, it allows the receiver to identify and discard duplicate segments.

Multiplexing:

In networks that use virtual circuits within the subnet, each open connection consumes some table space in the routers for the entire duration of the connection. If buffers are dedicated to the virtual circuit in each router as well, a user who left a terminal logged into a remote machine, there is need for multiplexing. There are 2 kinds of multiplexing:



(a) Upward multiplexing. (b) Downward multiplexing.

(a). UP-WARD MULTIPLEXING:

In the below figure, all the 4 distinct transport connections use the same network connection to the remote host. When connect time forms the major component of the carrier's bill, it is up to the transport layer to group port connections according to their destination and map each group onto the minimum number of port connections.

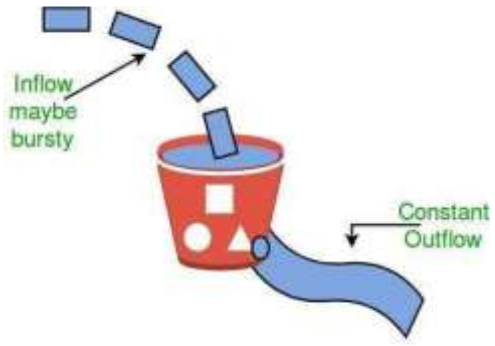
(b). DOWN-WARD MULTIPLEXING:

- If too many transport connections are mapped onto the one network connection, the performance will be poor.
- If too few transport connections are mapped onto one network connection, the service will be expensive. The possible solution is to have the transport layer open multiple connections and distribute the traffic among them on round-robin basis, as indicated in the below figure:

With 'k' network connections open, the effective band width is increased by a factor of 'k'.

Congestion Control

Congestion is a situation in which too many sources over a network attempt to send data and the router buffers start overflowing due to which loss of packets occurs. As a result, the retransmission of packets from the sources increases the congestion further. In this situation, the Transport layer provides Congestion Control in different ways. It uses open-loop congestion control to prevent congestion and closed-loop congestion control to remove the congestion in a network once it occurred. TCP provides AIMD – additive increases multiplicative decrease and leaky bucket technique for congestion control.



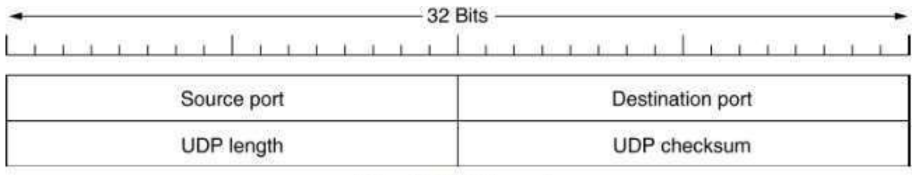
TRANSPORT PROTOCOLS - UDP

The Internet has two main protocols in the transport layer, a connectionless protocol and a connection-oriented one. The protocols complement each other. The connectionless protocol is UDP. It does almost nothing beyond sending packets between applications, letting applications build their own protocols on top as needed.

The connection-oriented protocol is TCP. It does almost everything. It makes connections and adds reliability with retransmissions, along with flow control and congestion control, all on behalf of the applications that use it. Since UDP is a transport layer protocol that typically runs in the operating system and protocols that use UDP typically run in user s pace, these uses might be considered applications.

INTROUCTION TO UDP

- The Internet protocol suite supports a connectionless transport protocol called UDP (User Datagram Protocol). UDP provides a way for applications to send encapsulated IP datagrams without having to establish a connection.
- UDP transmits segments consisting of an 8-byte header followed by the pay-load. The two ports serve to identify the end-points within the source and destination machines.
- When a UDP packet arrives, its payload is handed to the process attached to the destination port. This attachment occurs when the BIND primitive. Without the port fields, the transport layer would not know what to do with each incoming packet. With them, it delivers the embedded segment to the correct application.



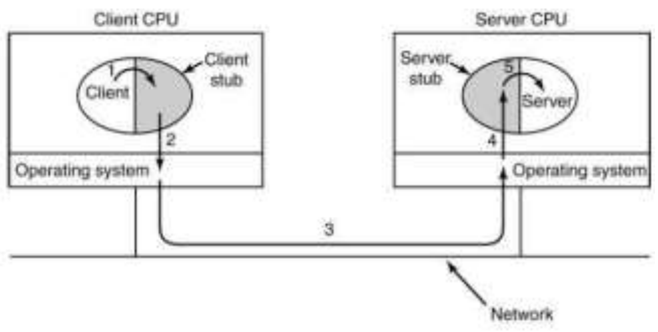
The UDP header.

- Source port, destination port: Identifies the end points within the source and destination machines.
- UDP length: Includes 8-byte header and the data
- UDP checksum: Includes the UDP header, the UDP data padded out to an even number of bytes if need be. It is an optional field

REMOTE PROCEDURE CALL

- In a certain sense, sending a message to a remote host and getting a reply back is like making a function call in a programming language. This is to arrange request-reply interactions on networks to be cast in the form of procedure calls.
- For example, just imagine a procedure named get IP address (host name) that works by sending a UDP packet to a DNS server and waiting or the reply, timing out and trying again if one is not forthcoming quickly enough. In this way, all the details of networking can be hidden from the programmer.
- RPC is used to call remote programs using the procedural call. When a process on machine 1 calls a procedure on machine 2, the calling process on 1 is suspended and execution of the called procedure takes place on 2.

- Information can be transported from the caller to the callee in the parameters and can come back in the procedure result. No message passing is visible to the application programmer. This technique is known as RPC (Remote Procedure Call) and has become the basis for many networking applications.
- Traditionally, the calling procedure is known as the client and the called procedure is known as the server.
- In the simplest form, to call a remote procedure, the client program must be bound with a small library procedure, called the client stub, that represents the server procedure in the client's address space. Similarly, the server is bound with a procedure called the server stub. These procedures hide the fact that the procedure call from the client to the server is not local.



Steps in making a remote procedure call. The stubs are shaded.

Step 1 is the client calling the client stub. This call is a local procedure call, with the parameters pushed onto the stack in the normal way.

Step 2 is the client stub packing the parameters into a message and making a system call to send the message. Packing the parameters is called marshaling.

Step 3 is the operating system sending the message from the client machine to the server machine.

Step 4 is the operating system passing the incoming packet to the server stub.

Step 5 is the server stub calling the server procedure with the unmarshaled parameters. The reply traces the same path in the other direction.

The key item to note here is that the client procedure, written by the user, just makes a normal (i.e., local) procedure call to the client stub, which has the same name as the server procedure. Since the client procedure and client stub are in the same address space, the parameters are passed in the usual way.

Similarly, the server procedure is called by a procedure in its address space with the parameters it expects. To the server procedure, nothing is unusual. In this way, instead of I/O being done on sockets, network communication is done by faking a normal procedure call. With RPC, passing pointers is impossible because the client and server are in different address spaces.

TCP (TRANSMISSION CONTROL PROTOCOL)

It was specifically designed to provide a reliable end-to end byte stream over an unreliable network. It was designed to adapt dynamically to properties of the inter network and to be robust in the face of many kinds of failures.

Each machine supporting TCP has a TCP transport entity, which accepts user data streams from local processes, breaks them up into pieces not exceeding 64kbytes and sends each piece as a separate IP datagram. When these datagrams arrive at a machine, they are given to TCP entity, which reconstructs the original byte streams. It is up to TCP to time out and retransmits them as needed, also to reassemble datagrams into messages in proper sequence. The different issues to be considered are:

1. The TCP Service Model
2. The TCP Protocol
3. The TCP Segment Header
4. The Connection Management
5. TCP Transmission Policy
6. TCP Congestion Control
7. TCP Timer Management.

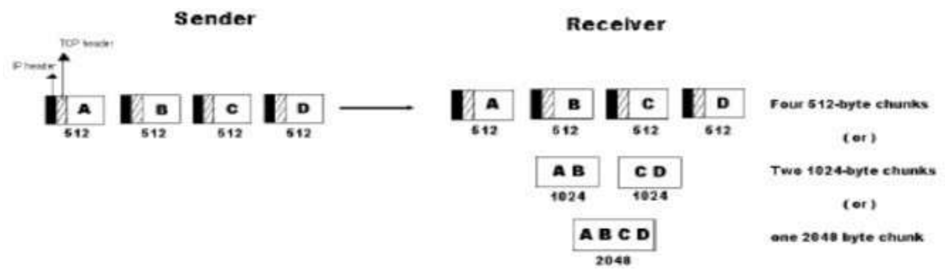
The TCP Service Model

TCP service is obtained by having both the sender and receiver create end points called SOCKETS

Each socket has a socket number(address)consisting of the IP address of the host, called a “PORT” (= TSAP) To obtain TCP service a connection must be explicitly established between a socket on the sending machine and a socket on the receiving machine

All TCP connections are full duplex and point to point i.e., multicasting or broadcasting is not supported. A TCP connection is a byte stream, not a message stream i.e., the data is delivered as chunks

E.g.: 4 * 512 bytes of data is to be transmitted.



Sockets:

A socket may be used for multiple connections at the same time. In other words, 2 or more connections may terminate at same socket. Connections are identified by socket identifiers at same socket. Connections are identified by socket identifiers at both ends. Some of the sockets are listed below:

Eg:

PORT-21	To establish a connection to a host to transfer a file using FTP
PORT-23	To establish a remote login session using TELNET

The TCP Protocol

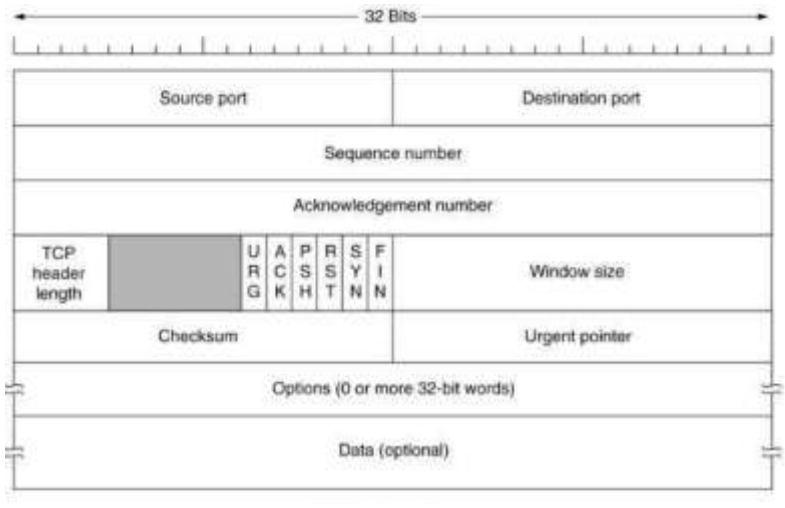
A key feature of TCP, and one which dominates the protocol design, is that every byte on a TCP connection has its own 32-bit sequence number.

- When the Internet began, the lines between routers were mostly 56-kbps leased lines, so a host blasting away at full speed took over 1 week to cycle through the sequence numbers.
- The basic protocol used by TCP entities is the sliding window protocol.
- When a sender transmits a segment, it also starts a timer.
- When the segment arrives at the destination, the receiving TCP entity sends back a segment (with data if any exist, otherwise without data) bearing an acknowledgement number equal to the next sequence number it expects to receive.
- If the sender's timer goes off before the acknowledgement is received, the sender transmits the segment again.

THE TCP SEGMENT HEADER

Every segment begins with a fixed-format, 20-byte header. The fixed header may be followed by header options. After the options, if any, up to 65,535 - 20 - 20 = 65,495 data bytes may follow, where the first 20 refer to the IP header and the second to the TCP header. Segments without any data are legal and are commonly used for acknowledgements and control messages.

A TCP segment's header field can be anything from 20 to 60 bytes long. Here 40 bytes are used for the options field, which is located at the end of the TCP header. A header is 20 bytes if there are no options field; otherwise, it can be up to 60 bytes.



TCP Header.

Header Fields

Source port- It is a 16-bit field that holds the port address of the application sending the data.

Destination Port- It is a 16-bit field that holds the port address of the application receiving the data.

Sequence Number- It is used to keep track of the bytes sent. Each byte in a TCP stream is uniquely identified by the TCP sequence number, which is a four-byte number.

Acknowledgment number- It is a 32-bit field that contains the acknowledgment number or the byte number that the receiver expects to receive next. It works as an acknowledgment for the previous data received successfully.

Header Length (HLEN)- The header length is a 4-bit field that specifies the length of the TCP header. It helps in knowing from where the actual data begins.

Flags- There are six control flags or bits:

URG: It indicates an urgent pointer. If URG is set, then the data is processed urgently.

ACK: It represents the acknowledgment field in a segment. If the ACK is set to 0, the data packet does not contain an acknowledgment.

RST: It Resets the connection. If RST is set, then it requests to restart a connection.

PSH: If this field is set, the receiving device is requested to push the data directly to the receiver without buffering it.

SYN: It initiates and establishes a connection between the hosts. If SYN is set, the device wants to establish a secure connection; else, not.

FIN: It is used to terminate a connection. If FIN is 1, the device wants to terminate the connection; else, not.

Checksum - A checksum is a sequence of numbers and letters used to detect errors in data. It is a 16-bit field that is optional in UDP but mandatory in TCP/IP.

Window size - It is a 16-bit field. This field specifies the size of data that the receiver can accept.

Urgent pointer - This field (valid only If the URG flag is set to 1) is used to indicate urgently needed data and must be received as soon as possible. It specifies a value that will be appended to the sequence number to get the last urgent byte's sequence number.

TCP 3-WAY HANDSHAKE PROCESS

TCP 3-way handshake process is used for establishing and terminating the connection between the client and server.

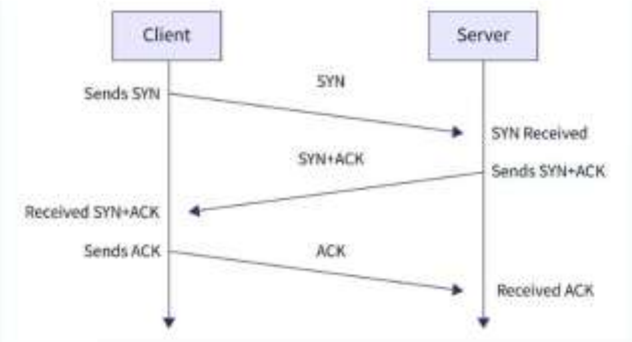
Steps of a 3-Way Handshake for Establishing the Connection

The three steps involved in establishing a connection using the 3-way handshake process in TCP are as follows:

1. The client sends the SYN (synchronize) message to the server: When a client requests to connect to a server, it sends the message to the server with the SYN flag set to 1. The message also includes:
    - The sequence number (any random 32-bit number).
    - The ACK (which is set to 0 in this case).
    - The window size.
    - The maximum segment size. For example, if the window size is 3000 bits and the maximum segment size is 300 bits, the connection can send a maximum of 10 data segments (3000/300 = 10).
2. The server responds with the SYN and the ACK (synchronize-acknowledge) message to the client: After receiving the synchronization request, the server sends the client an acknowledgment by changing the ACK flag to '1'. The ACK's acknowledgment number is one higher than the sequence number received. If the client sends an SYN with a sequence number of 2000, the server will send the ACK using acknowledgment number = 20001. If the server wants to create the connection, it sets

   the SYN flag to '1' and transmits it to the client. The SYN sequence number used here will be different from the SYN used by the client. The server also informs the client of its window size and maximum segment size. After this step is completed, the connection is established from the client to the server.

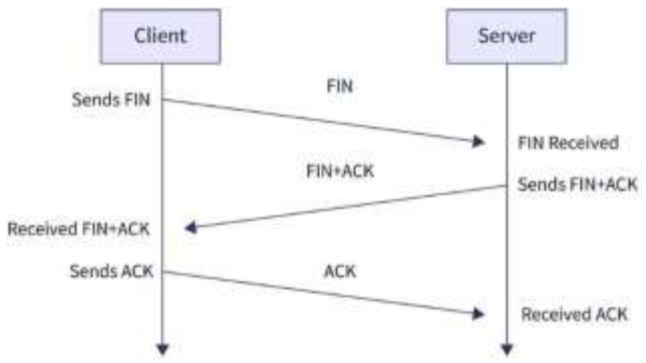
3. The client sends the ACK (acknowledge) message to the server: The client will set the ACK flag to '1' after receiving the SYN from the server and transmits it with an acknowledgment number 1 greater than the server's SYN sequence number. The SYN flag has been set to '0' in this case. The connection between the server and the client is now formed after this phase is completed.



Steps of a 3-Way Handshake for Terminating the Connection

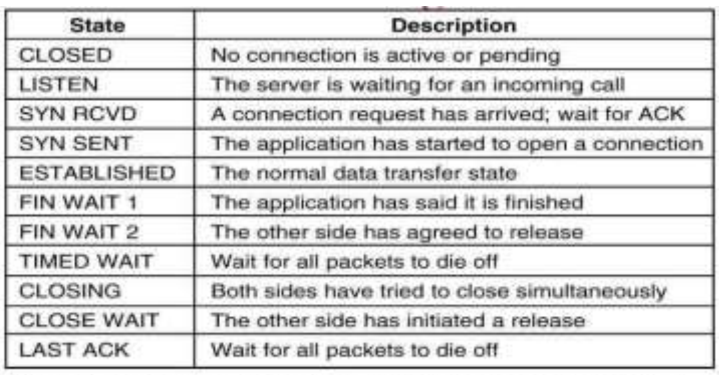
Most implementations today allow three-way and four-way handshaking with a half-close option for connection termination. Here we only mentioned the steps of three-way handshaking for connection termination. The three steps involved in terminating a connection using the 3-way handshake process in TCP are as follows:

1. The client sends the FIN (finish) message to the server: When the client decides to disconnect from the network, it transmits the message to the server with a random sequence number and sets the FIN flag to '1'. ACK is set to 0 in this case.
2. The server responds with the FIN and the ACK (finish-acknowledge) message to the client: After receiving the request, the server acknowledges the client's termination request by changing the ACK flag to '1'. The ACK's acknowledgment number is one higher than the sequence number received. If the client sends a FIN with a sequence number of 2000, the server will send the ACK using acknowledgment number = 20001. If the server also decides to terminate the connection, it sets the FIN flag to '1' and transmits it to the client. The FIN sequence number used here will be different from the FIN used by the client. After this step is completed, the connection between the client to the server is disconnected.
3. The client sends the ACK (acknowledge) message to the server: The client will set the ACK flag to '1' after receiving the FIN from the server and transmits it with an acknowledgment number 1 greater than the server's FIN sequence number. The FIN flag is set to '0' in this case. After this step is completed, the connection is also disconnected from the server to the client.



TCP Connection Management Modeling

The steps required establishing and release connections can be represented in a finite state machine with the 11 states listed in Fig. 4.13. In each state, certain events are legal. When a legal event happens, some action may be taken. If some other event happens, an error is reported.

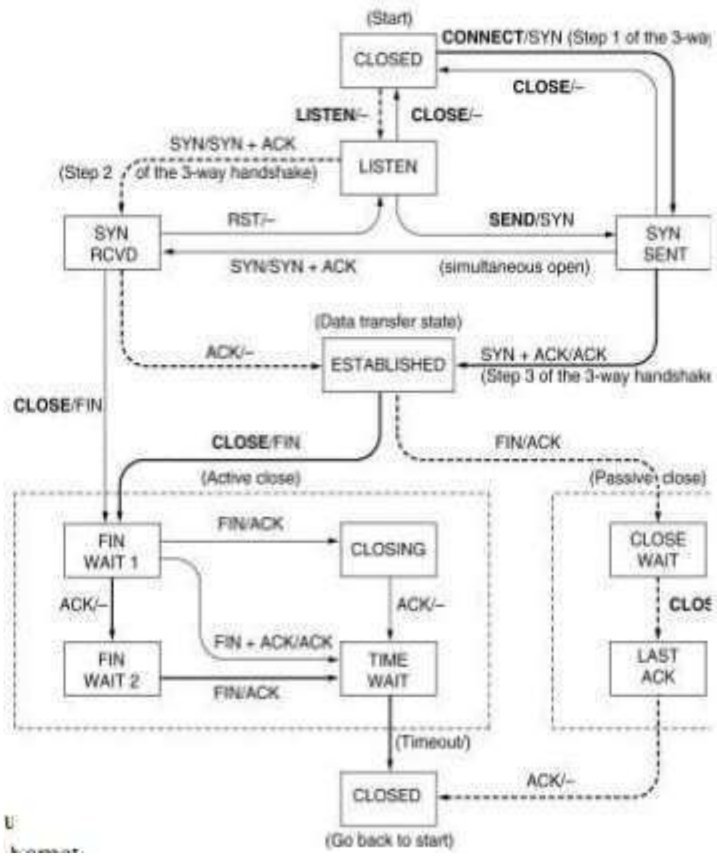


State	Description
CLOSED	No connection is active or pending
LISTEN	The server is waiting for an incoming call
SYN RCVD	A connection request has arrived; wait for ACK
SYN SENT	The application has started to open a connection
ESTABLISHED	The normal data transfer state
FIN WAIT 1	The application has said it is finished
FIN WAIT 2	The other side has agreed to release
TIMED WAIT	Wait for all packets to die off
CLOSING	Both sides have tried to close simultaneously
CLOSE WAIT	The other side has initiated a release
LAST ACK	Wait for all packets to die off

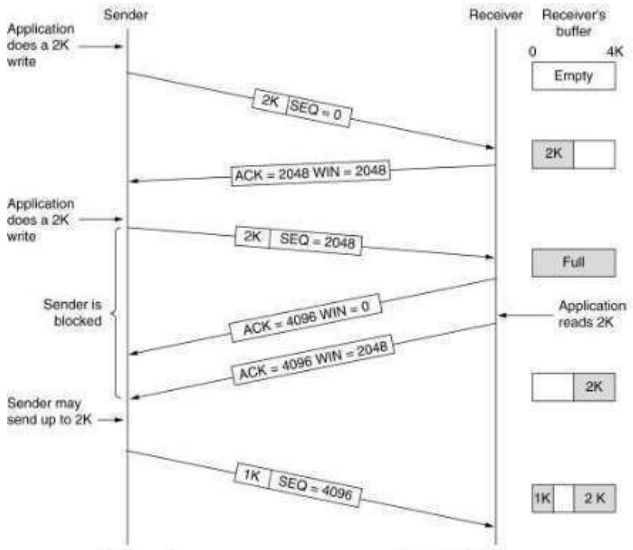
The states used in the TCP connection management finite state machine

TCP Connection management from server's point of view:

1. The server does a LISTEN and settles down to see who turns up.
2. When a SYN comes in, the server acknowledges it and goes to the SYNRCVD state
3. When the servers SYN is itself acknowledged the 3-way handshake is complete and server goes to the ESTABLISHED state. Data transfer can now occur.
4. When the client has had enough, it does a close, which causes a FIN to arrive at the server [dashed box marked passive close].
5. The server is then signaled.
6. When it too, does a CLOSE, a FIN is sent to the client.
7. When the client's acknowledgement shows up, the server releases the connection and deletes the connection record.



TCP Transmission Policy



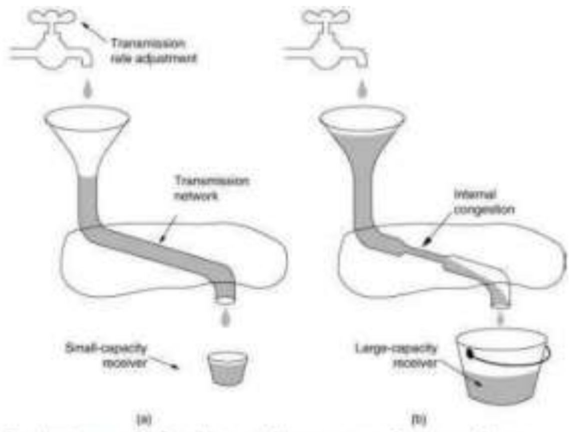
Window management in TCP.

1. In the above example, the receiver has 4096-byte buffer.
2. If the sender transmits a 2048-byte segment that is correctly received, the receiver will acknowledge the segment.
3. Now the receiver will advertise a window of 2048 as it has only 2048 of buffer space, now.
4. Now the sender transmits another 2048 bytes which are acknowledged, but the advertised window is'0'.

5. The sender must stop until the application process on the receiving host has removed some data from the buffer, at which time TCP can advertise a layer window.

TCP CONGESTION CONTROL:

TCP does to try to prevent the congestion from occurring in the first place in the following way:
When a connection is established, a suitable window size is chosen and the receiver specifies a window based on its buffer size. If the sender sticks to this window size, problems will not occur due to buffer overflow at the receiving end. But they may still occur due to internal congestion within the network. Let's see this problem occurs.



(a) A fast network feeding a low capacity receiver.
(b) A slow network feeding a high-capacity receiver.

In fig (a): We see a thick pipe leading to a small- capacity receiver. As long as the sender does not send more water than the bucket can contain, no water will be lost.
In fig (b): The limiting factor is not the bucket capacity, but the internal carrying capacity of the n/w. if too much water comes in too fast, it will backup and some will be lost.

- When a connection is established, the sender initializes the congestion window to the size of the max segment in use our connection.
- It then sends one max segment. if this max segment is acknowledged before the timer goes off, it adds one segments worth of bytes to the congestion window to make it two maximum size segments and sends 2 segments.
- As each of these segments is acknowledged, the congestion window is increased by one max segment size.
- When the congestion window is 'n' segments, if all 'n' are acknowledged on time, the congestion window is increased by the byte count corresponding to 'n' segments.
- The congestion window keeps growing exponentially until either a time out occurs or the receiver's window is reached.
- The internet congestion control algorithm uses a third parameter, the "threshold" in addition to receiver and congestion windows.

TCP TIMER MANAGEMENT:

TCP uses 3 kinds of timers:

1. Retransmission timer
2. Persistence timer
3. Keep-Alive timer.

Retransmission timer:

When a segment is sent, a timer is started. If the segment is acknowledged before the timer expires, the timer is stopped. If on the other hand, the timer goes off before the acknowledgement comes in, the segment is retransmitted and the timer is started again. The algorithm that constantly adjusts the time-out interval, based on continuous measurements of n/w performance was proposed by JACOBSON and works as follows:

- for each connection, TCP maintains a variable RTT, that is the best current estimate of the round trip time to the destination inn question.
- When a segment is sent, a timer is started, both to see how long the acknowledgement takes and to trigger a retransmission if it takes too long.
- If the acknowledgement gets back before the timer expires, TCP measures how long the measurements took say M

Persistence timer:

- It is designed to prevent the following deadlock:
- The receiver sends an acknowledgement with a window size of ‘0’ telling the sender to wait later, the receiver updates the window, but the packet with the update is lost now both the sender and receiver are waiting for each other to do something
- when the persistence timer goes off, the sender transmits a probe to the receiver the response to the probe gives the window size
- if it is still zero, the persistence timer is set again and the cycle repeats
- if it is non zero, data can now be sent

Keep-Alive timer: When a connection has been idle for a long time, this timer may go off to cause one side to check if other side is still there. If it fails to respond, the connection is terminated.

Stop & Wait ARQ, Go back N and Selective Repeat Comparison

Factors	Stop and Wait ARQ	Go back N	Selective Repeat	Remarks
Efficiency	1 / (1+2a)	N / (1+2a)	N / (1+2a)	Go back N and Selective Repeat gives better efficiency than Stop and Wait ARQ.
Window Size	Sender Window Size = 1 Receiver Window Size = 1	Sender Window Size = N Receiver Window Size = 1	Sender Window Size = N Receiver Window Size = N	Buffer requirement in Selective Repeat is very large. If the system does not have lots of memory, then it is better to choose Go back N.

Minimum number of sequence numbers required	2	N+1	2 x N	Selective Repeat requires large number of bits in sequence number field.
Retransmissions required if a packet is lost	Only the lost packet is retransmitted	The entire window is retransmitted	Only the lost packet is retransmitted	Selective Repeat is far better than Go back N in terms of retransmissions required.

Bandwidth Requirement	Bandwidth requirement is Low	Bandwidth requirement is high because even if a single packet is lost, entire window has to be retransmitted.	Bandwidth requirement is moderate	Selective Repeat is better than Go back N in terms of bandwidth requirement.
		Thus, if error rate is high, it wastes a lot of bandwidth.		

CPU usage	Low	Moderate	High due to searching and sorting required at sender and receiver side	Go back N is better than Selective Repeat in terms of CPU usage.
Level of difficulty in Implementation	Low	Moderate	Complex as it requires extra logic and sorting and searching	Go back N is better than Selective Repeat in terms of implementation difficulty.

Acknowledgements	Uses independent acknowledgement for each packet	Uses cumulative acknowledgements (but may use independent acknowledgements as well)	Uses independent acknowledgement for each packet	Sending cumulative acknowledgements reduces the traffic in the network but if it is lost, then the ACKs for all the corresponding packets are lost.
Type of Transmission	Half duplex	Full duplex	Full duplex	Go back N and Selective Repeat are better in terms of channel usage.

<u>Conclusions-</u>

- Go back N is more often used than other protocols.
- SR protocol is less used because of its complexity.
- Stop and Wait ARQ is less used because of its low efficiency.

- Depending on the context and resources availability, Go back N or Selective Repeat is employed.
- Selective Repeat and Stop and Wait ARQ are similar in terms of retransmissions.

- Go back N and Selective Repeat are similar in terms of efficiency if sender window sizes are same.
- SR protocol may be considered as a combination of advantages of Stop and Wait ARQ and Go back N.
- SR protocol is superior to other protocols but because of its complexity, it is less used. Important Notes- Note-01:

Protocols at data link layer like HDLC (Low level protocols) use Go back N. This is because-

1. Bandwidth is high
2. CPU is very busy doing routing job
3. Error rate is low since out of order packets are not possible in wired medium

Note-02:

Protocols at transport layer like TCP (High level protocols) use selective repeat.