UNIT – IV

THE TRANSPORT LAYER

The Transport Layer

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

Operating systems today support both multiuser and multiprogramming environments. A remote computer can run several server programs at the same time, just as local computers can run one or more client programs at the same time. For communication, we must define the following:

1. Local host

2. Local process

3. Remote host

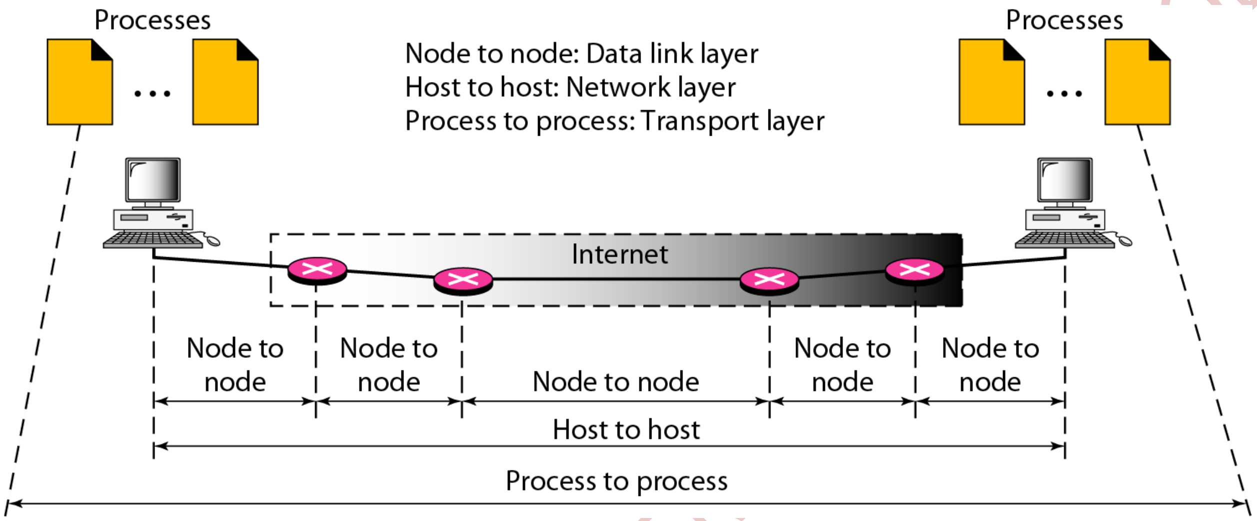
4. Remote process

Addressing

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

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

In the Internet model, the port numbers are 16-bit integers between 0 and 65,535. The client program defines itself with a port number, chosen randomly by the transport layer software running on the client host. This is the ephemeral port number. The server process must also define itself with a port number. This port number, however, cannot be chosen randomly. The Internet has decided to use universal port numbers for servers; these are called well-known port numbers.

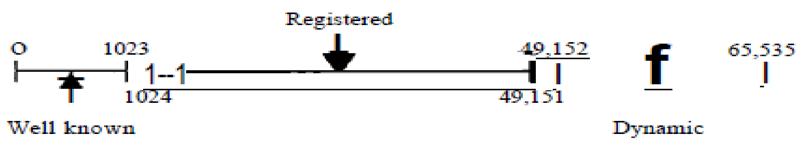


Types of Delivery

lANA Ranges

The lANA (Internet Assigned Number Authority) has divided the port numbers into three ranges: Well-known ports: - The ports ranging from 0 to 1023 are assigned and controlled by lANA. 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.



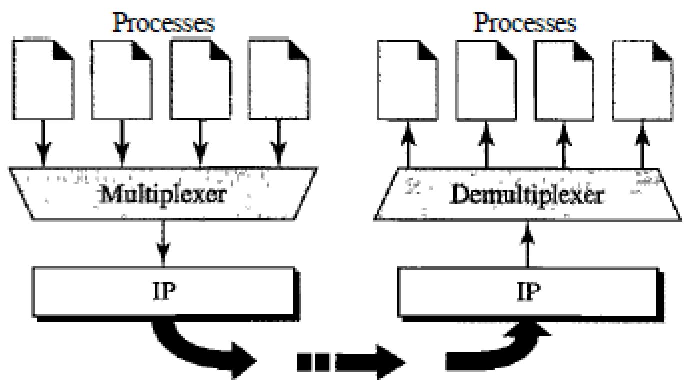
Socket Addresses

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.

Multiplexing and Demultiplexing

The addressing mechanism allows multiplexing and demultiplexing by the transport layer, as shown in Figure 23.6.

Figure 23.6 *Multiplexing and demultiplexing*



Multiplexing

At the sender site, there may be several processes that need to send packets. However, there is only one transport layer protocol at any time. This is a many-to-one relationship and requires multiplexing. The protocol accepts messages from different processes, differentiated by their assigned port numbers. After adding the header, the transport layer passes the packet to the network layer.

Demultiplexing

At the receiver site, the relationship is one-to-many and requires demultiplexing. The transport layer receives datagrams from the network layer. After error checking and dropping of the header, the transport layer delivers each message to the appropriate process based on the port number.

Common properties that a transport protocol can be expected to provide

- Guarantees message delivery
- Delivers messages in the same order they were sent
- Delivers at most one copy of each message
- Supports arbitrarily large messages
- Supports synchronization between the sender and the receiver
- Allows the receiver to apply flow control to the sender
- Supports multiple application processes on each host

Typical limitations of the network on which transport protocol will operate

- Drop messages
- Reorder messages
- Deliver duplicate copies of a given message
- Limit messages to some finite size
- Deliver messages after an arbitrarily long delay

Challenge for Transport Protocols

Develop algorithms that turn the less-than-desirable properties of the underlying network into the high level of service required by application programs

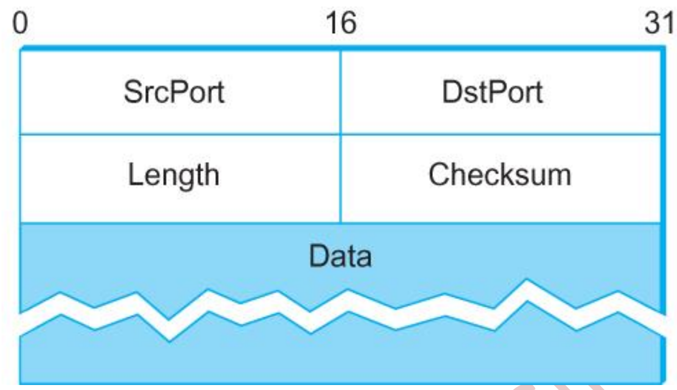
Simple Demultiplexer (UDP – User Datagram Protocol)

- UDP is a connectionless unreliable transport protocol–extends IP's host-to-host delivery service into a process-to-process communication service.
- UDP provides Best effort service. That is it will not provide reliable service.
- UDP does not guarantee reliability or ordering in the way that TCP does.
- Datagrams may arrive out of order, appear duplicated, or go missing without notice.
- Avoiding the overhead of checking whether every packet actually arrived makes UDP faster and more efficient, for applications that do not need guaranteed delivery.
- Time-sensitive applications often use UDP because dropped packets are preferable to delayed packets. UDP's stateless nature is also useful for servers that answer small queries

from huge numbers of clients. Unlike TCP, UDP is compatible with packet broadcast (sending to all on local network) and multicasting (send to all subscribers).

- Common network applications that use UDP include: the Domain Name System (DNS), streaming media applications such as IPTV, Voice over IP (VoIP), Trivial File Transfer Protocol (TFTP) and online games.
- Adds a level of demultiplexing which allows multiple application processes on each host to share the network.

Format for UDP header



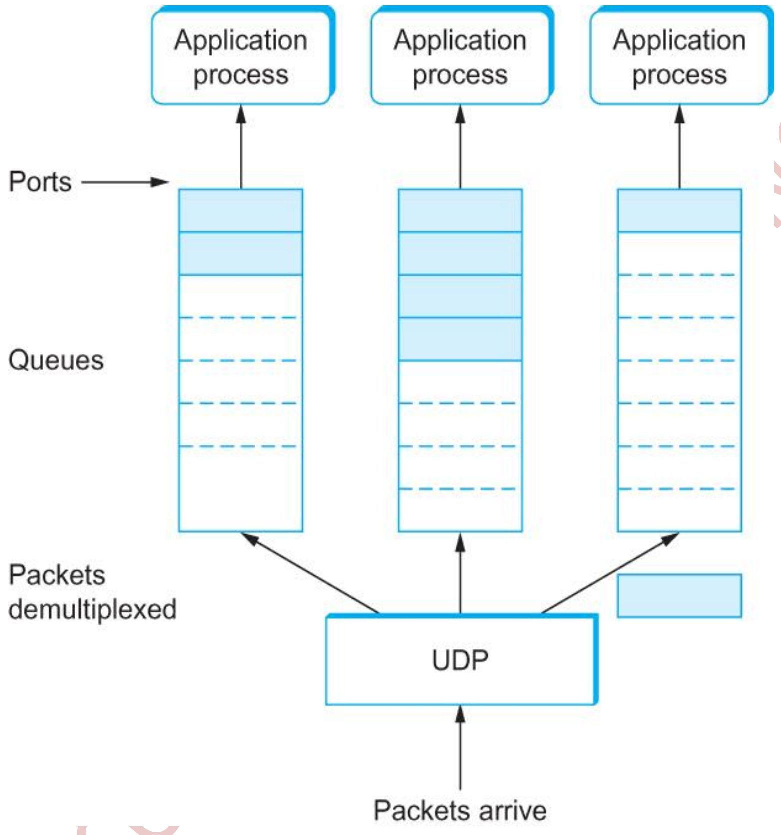
Format for UDP header

- The source port, much like the source port in TCP, identifies the process on the originating system.
- The destination port identifies the receiving process on the receiving machine
- Generally, server will use accept messages at a *well-known port*. That is, each server receives its messages at some fixed port that is widely published, for example, the Domain Name Server (DNS) receives messages at well-known port 53 on each host, the mail service listens for messages at port 25, and the Unix talk program accepts messages at well-known port 517, http at port no 80 and so on. Client process can use ephemeral port numbers.
- The length field contains the length of the UDP datagram.
- The checksum field is used by UDP to verify the correctness of the UDP header and data(contents of the message body, sometimes called the *pseudoheader*).

- The pseudoheader consists of three fields from the IP header—protocol number, source IP address, and destination IP address— plus the UDP length field

UDP Message Queue

When a message arrives, the protocol (e.g., UDP) appends the message to the end of the queue. Should the queue be full, the message is discarded. There is no flow-control mechanism that tells the sender to slow down. When an application process wants to receive a message, one is removed from the front of the queue. If the queue is empty, the process blocks until a message becomes available.



UDP Message Queue

Use of UDP

The following lists some uses of the UDP protocol:

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

Reliable Byte Stream (TCP)

In contrast to UDP, Transmission Control Protocol (TCP) offers the following services

- Reliable
- Connection oriented; it creates a virtual connection between two TCPs to send data.
- Byte-stream service
- TCP guarantees the reliable, in-order delivery of a stream of bytes. It is a full-duplex protocol, meaning that each TCP connection supports a pair of byte streams, one flowing in each direction
- Flow control involves preventing senders from overrunning the capacity of the receivers
- TCP supports a demultiplexing mechanism that allows multiple application programs on any given host to simultaneously carry on a conversation with their peers.
- TCP also implements a highly tuned congestion-control mechanism. The idea of this mechanism is to throttle how fast TCP sends data, not for the sake of keeping the sender from overrunning the receiver, but to keep the sender from overloading the network.

Difference between flow control and congestion control:

Flow control involves preventing senders from overrunning the capacity of receivers. *Congestion control* involves preventing too much data from being injected into the network, thereby causing switches or links to become overloaded. Thus, flow control is an end-to-end issue, while congestion control is concerned with how hosts and networks interact.

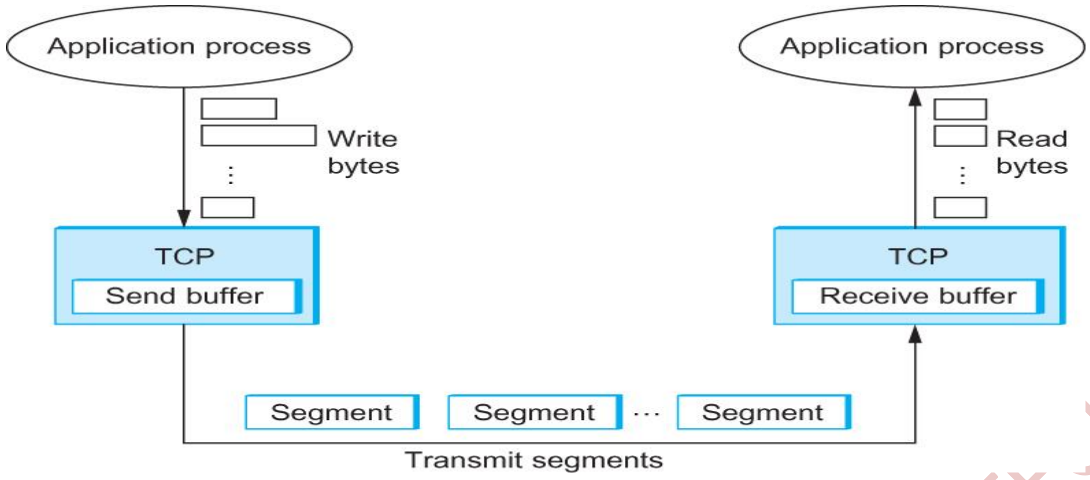
End-to-end Issues

At the heart of TCP is the sliding window algorithm. As TCP runs over the Internet rather than a point-to-point link, the following issues need to be addressed by the sliding window algorithm

- TCP supports logical connections between processes that are running on two different computers in the Internet
- TCP connections are likely to have widely different RTT times
- Packets may get reordered in the Internet
- TCP needs a mechanism using which each side of a connection will learn what resources the other side is able to apply to the connection
- TCP needs a mechanism using which the sending side will learn the capacity of the network

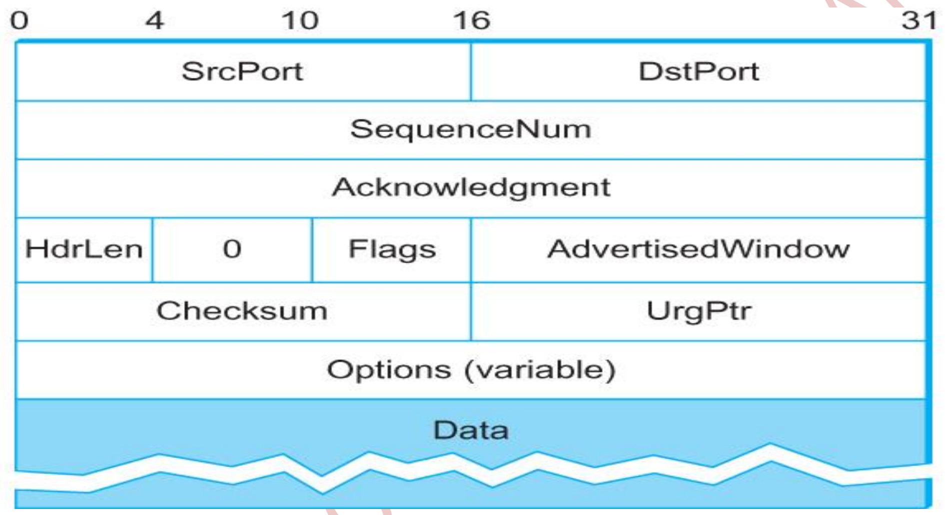
TCP Segment

- TCP is a byte-oriented protocol, which means that the sender writes bytes into a TCP connection and the receiver reads bytes out of the TCP connection.
- TCP on the source host buffers enough bytes from the sending process to fill a reasonably sized packet and then sends this packet to its peer on the destination host.
- TCP on the destination host then empties the contents of the packet into a receive buffer, and the receiving process reads from this buffer at its leisure.
- The packets exchanged between TCP peers are called *segments*.



How TCP manages a byte stream

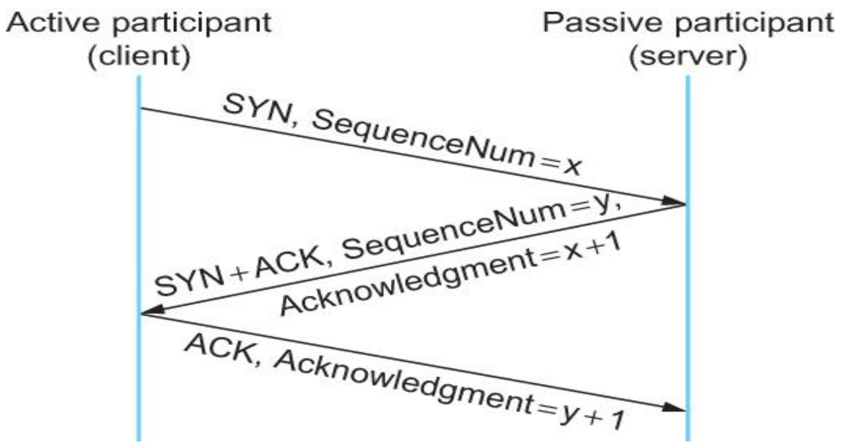
TCP Header



- The SrcPort and DstPort fields identify the source and destination ports, respectively.
- The Acknowledgment, SequenceNum, and AdvertisedWindow fields are all involved in TCP's sliding window algorithm.
- The SequenceNum field contains the sequence number for the first byte of data carried in that segment.
- The Acknowledgment and AdvertisedWindow fields carry information about the flow of data going in the other direction.
- The 6-bit Flags field is used to relay control information between TCP peers.
- The possible flags include SYN, FIN, RESET, PUSH, URG, and ACK.
- The SYN and FIN flags are used when establishing and terminating a TCP connection, respectively.

- The ACK flag is set any time the Acknowledgment field is valid, implying that the receiver should pay attention to it.
- The URG flag signifies that this segment contains urgent data. When this flag is set, the UrgPtr field indicates where the nonurgent data contained in this segment begins.
- The urgent data is contained at the front of the segment body, up to and including a value of UrgPtr bytes into the segment.
- The PUSH flag signifies that the sender invoked the push operation, which indicates to the receiving side of TCP that it should notify the receiving process of this fact.
- The RESET flag signifies that the receiver has become confused, it received a segment it did not expect to receive—and so wants to abort the connection.
- The Checksum field is used in exactly the same way as for UDP—it is computed over the TCP header, the TCP data, and the pseudoheader, which is made up of the source address, destination address, and length fields from the IP header.

Connection Establishment/Termination in TCP


Active participant (client)
Passive participant (server)
SYN, SequenceNum=x
SYN+ACK, SequenceNum=y, Acknowledgment=x+1
ACK, Acknowledgment=y+1

Three-Way Handshake

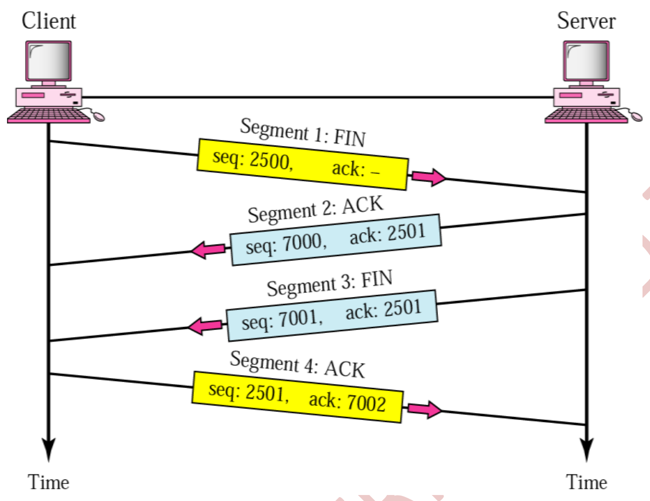
The algorithm used by TCP to establish a connection is called a *three- way handshake*. The three-way handshake involves the exchange of three messages between the client and the server

- First, the client (the active participant) sends a segment to the server (the passive participant) stating the initial sequence number it plans to use (Flags = SYN,ACK is not set at the first time SequenceNum = *x*).
- The server then responds with a single segment that both acknowledges the client's sequence number (Flags =SYN + ACK = *x* + 1,Seq No) and states its own beginning sequence number. That is, both the SYN and ACK bits are set in the Flags field of this second message.
- Finally, the client responds with a third segment that acknowledges the server's sequence number (Flags = ACK, where Ack = *y* + 1). The reason that each side acknowledges a sequence number that is one larger than the one sent is that the Acknowledgment field actually identifies the "next sequence number expected," thereby implicitly acknowledging all earlier sequence numbers.

3.3.5 Connection Termination in TCP(four way handshaking)

- Client TCP sends first segment , a FIN segment
- The server TCP sends second segment , an ack segment, to confirm receipt of FIN. In this segment it uses ack number which is one plus than the sequence number received in the FIN segment.

- The server TCP can continue sending data in the server to client direction. When it does not have any data to send, this server also sends third segment called FIN segment.
- The client then fourth segment , an ack segment to confirm the FIN. This contain ack number which is one plus than the sequence number received in the FIN segment.



3.3.6 State Transition Diagram

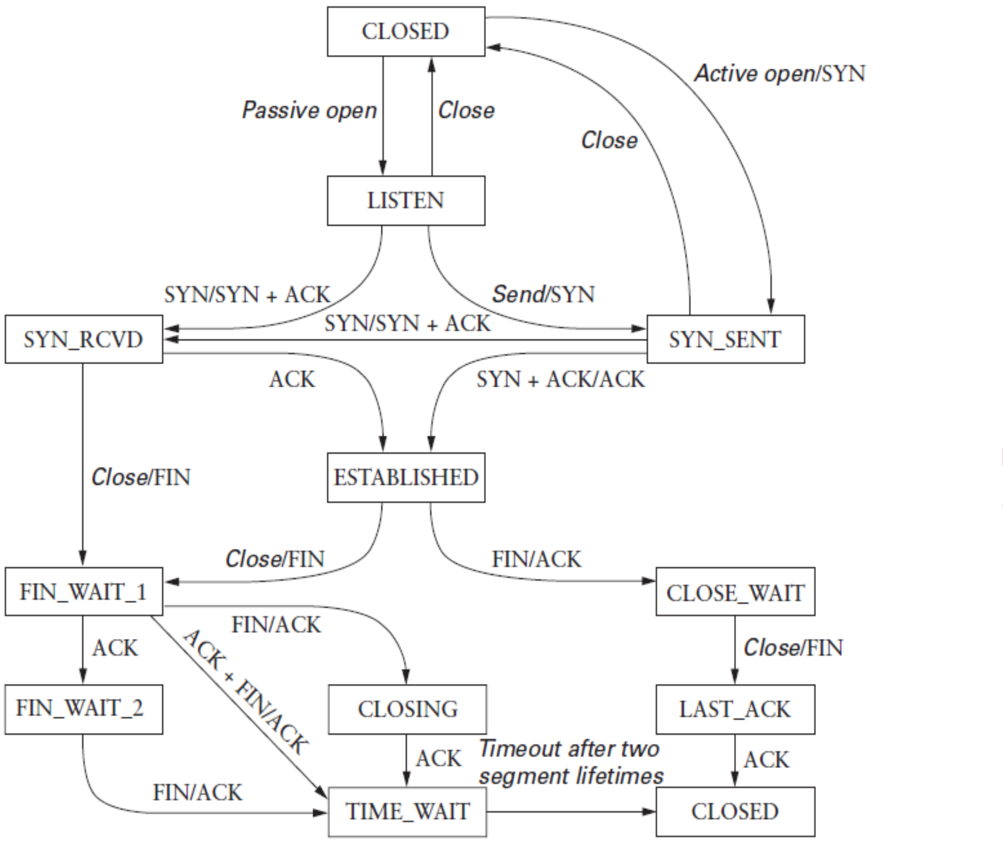


Figure 5.7 TCP state transition diagram.

Table 15.2 *States for TCP*

State	*Description*
CLOSED	No connection exists
LISTEN	Passive open received; waiting for SYN
SYN-SENT	SYN sent; waiting for ACK
SYN-RCVD	SYN+ACK sent; waiting for ACK
ESTABLISHED	Connection established; data transfer in progress
FIN-WAIT-1	First FIN sent; waiting for ACK
FIN-WAIT-2	ACK to first FIN received; waiting for second FIN
CLOSE-WAIT	First FIN received, ACK sent; waiting for application to close
TIME-WAIT	Second FIN received, ACK sent; waiting for 2MSL time-out
LAST-ACK	Second FIN sent; waiting for ACK
CLOSING	Both sides decided to close simultaneously

- The client does an active open, which causes its TCP to send a SYN segment to the server and to move to the SYN SENT state.
- When the SYN segment arrives at the server, it moves to the SYN RCVD state and responds with a SYN+ACK segment.

- The arrival of this segment causes the client to move to the ESTABLISHED state and to send an ACK back to the server.
- When this ACK arrives, the server finally moves to the ESTABLISHED state.
- When the client process has no more data to sent, it issues active close command to its TCP. TCP issues FIN segment and enters into FIN-WAIT -1 state.
- After receiving ACK from the server, client enters into FIN-WAIT-2 state.
- when the client receives a FIN segment, it sends an ACK segment and goes to the TIME – WAIT state.
- The client remains in this state for 2MSL. When the timer expires, it goes to the CLOSED state.
- The server process issues a passive open command.
- The server TCP goes to the LISTEN state.
- When it receives a SYN segment, TCP sends SYN + ACK segment and goes to SYN-RCVD state.
- After receiving the ACK segment, TCP goes to the ESTABLISHED state.
- The server upon receiving the FIN segment from the client, it sends ACK segment and goes to the CLOSE-WAIT state.
- After receiving passive close command from its process, the server sends FIN segment to the clientand goes to LAST – ACK state.
- When the Ack segment is received fromthe client,the server goes to CLOSED state.