2.4 TRANSMISSION CONTROL PROTOCOL (TCP)

- TCP is a reliable, connection-oriented, byte-stream protocol.
- 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.
- TCP includes a flow-control mechanism for each of these byte streams that allow the receiver to limit how much data the sender can transmit at a given time.
- 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 congestion-control mechanism. The idea of this mechanism is to prevent sender from overloading the network.
- Flow control is an end to end issue, whereas congestion control is concerned with how host and network interact.

Connection-Oriented Service

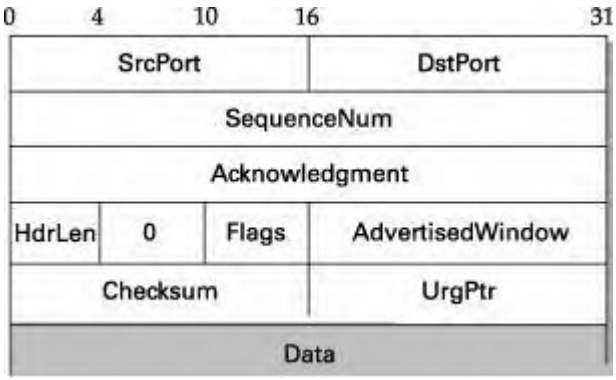
- TCP is a connection-oriented protocol.
- A connection needs to be established for each pair of processes.
- When a process at site A wants to send to and receive data from another process at site B, the following three phases occur:
  1. The two TCP's establish a logical connection between them.
  2. Data are exchanged in both directions.
  3. The connection is terminated.

Reliable Service

- TCP is a reliable transport protocol.
- It uses an acknowledgment mechanism to check the safe and sound arrival of data.

TCP PACKET FORMAT

- Each TCP segment contains the header plus the data.
- The segment consists of a header of 20 to 60 bytes, followed by data from the application program.
- The header is 20 bytes if there are no options and up to 60 bytes if it contains options.



SrcPort and DstPort—port number of source and destination process.

SequenceNum—contains sequence number, i.e. first byte of data segment.

Acknowledgment— byte number of segment, the receiver expects next.

HdrLen—Length of TCP header as 4-byte words.

Flags— contains ***six*** control bits known as flags.

i. **URG** — segment contains urgent data.
ii. **ACK** — value of acknowledgment field is valid.
iii. **PUSH** — sender has invoked the push operation.
iv. **RESET** — receiver wants to abort the connection.
v. **SYN** — synchronize sequence numbers during connection establishment.
vi. **FIN** — terminates the TCP connection.

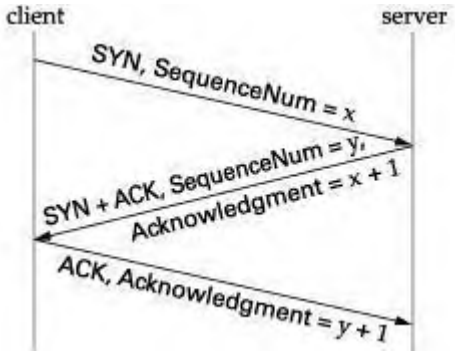
- **Advertised Window—**defines receiver's window size and acts as flow control.
- **Checksum—**It is computed over TCP header, Data, and pseudo header containing IP fields (Length, SourceAddr & DestinationAddr).
- **UrgPtr —** used when the segment contains urgent data. It defines a value that must be added to the sequence number.
- **Options -** There can be up to 40 bytes of optional information in the TCP header.

5.5 TCP CONNECTION MANAGEMENT

- TCP is connection-oriented.
- A connection-oriented transport protocol establishes a logical path between the source and destination.
- All of the segments belonging to a message are then sent over this logical path.
- In TCP, connection-oriented transmission requires three phases: Connection Establishment, Data Transfer and Connection Termination.

Connection Establishment

- While opening a TCP connection the two nodes(client and server) want to agree on a set of parameters.
- The parameters are the starting sequence numbers that is to be used for their respective byte streams.
- Connection establishment in TCP is a *three-way handshaking*.

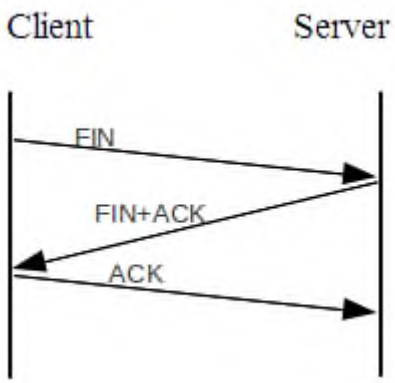


1. Client sends a SYN segment to the server containing its initial sequence number (Flags = SYN, SequenceNum = *x*)
2. Server responds with a segment that acknowledges client's segment and specifies its initial sequence number (Flags = SYN + ACK, ACK = *x* + 1 SequenceNum = *y*).
3. Finally, client responds with a segment that acknowledges server's sequence number (Flags = ACK, ACK = *y* + 1).

Connection Termination

- Connection termination or teardown can be done in two ways :

Three-way Close and Half-Close ***Three-way Close***—Both client and server close *simultaneously.*

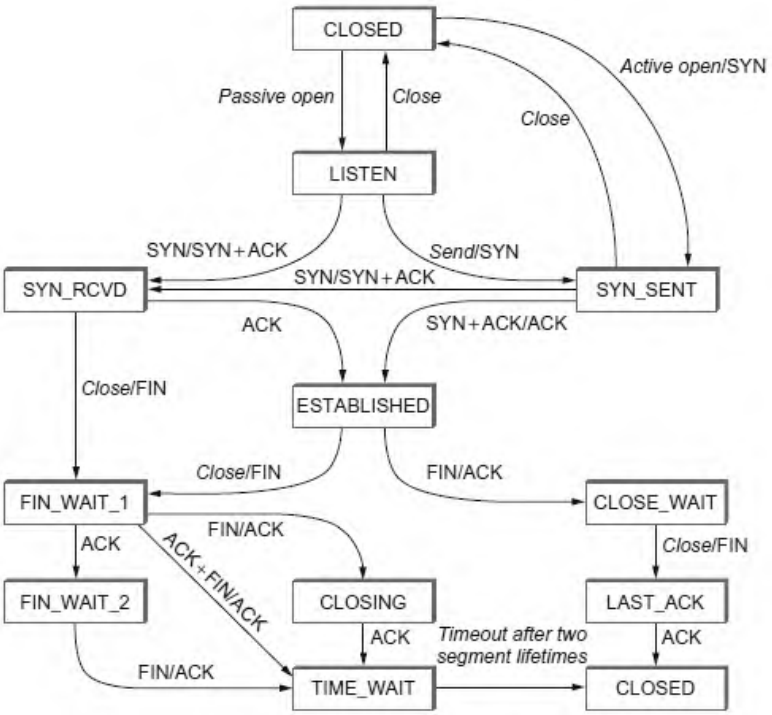


- Client sends a FIN segment.
- The FIN segment can include last chunk of data.
- Server responds with FIN + ACK segment to inform its closing.
- Finally, client sends an ACK segment

STATE TRANSITION DIAGRAM

- To keep track of all the different events happening during connection establishment, connection termination, and data transfer, TCP is specified as the finite state machine (FSM).
- The transition from one state to another is shown using directed lines.
- States involved in opening and closing a connection is shown above and below ESTABLISHED state respectively.
- States Involved in TCP :

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



Opening a TCP Connection

1. Server invokes a *passive* open on TCP, which causes TCP to move to LISTEN state
2. Client does an *active* open, which causes its TCP to send a SYN segment to the server and move to SYN_SENT state.
3. When SYN segment arrives at the server, it moves to SYN_RCVD state and *responds* with a SYN + ACK segment.
4. Arrival of SYN + ACK segment causes the client to move to ESTABLISHED state and sends an ACK to the server.
5. When ACK arrives, the server finally moves to ESTABLISHED state.

Closing a TCP Connection

1. Client / Server can independently close its half of the connection or simultaneously. Transitions from ESTABLISHED to CLOSED state are: ***One side closes***: ESTABLISHED → FIN_WAIT_1 → FIN_WAIT_2 → TIME_WAIT → CLOSED

Other side closes: ESTABLISHED → CLOSE_WAIT → LAST_ACK → CLOSED

Simultaneous close: ESTABLISHED → FIN_WAIT_1 → CLOSING → TIME_WAIT → CLOSED