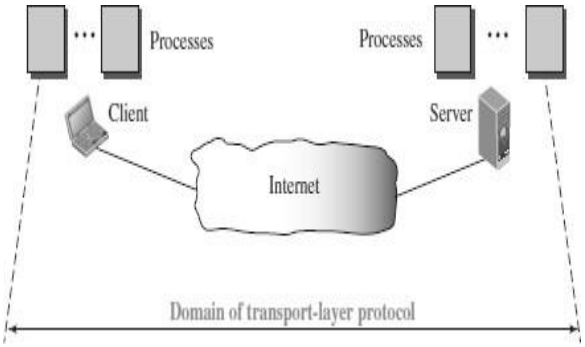
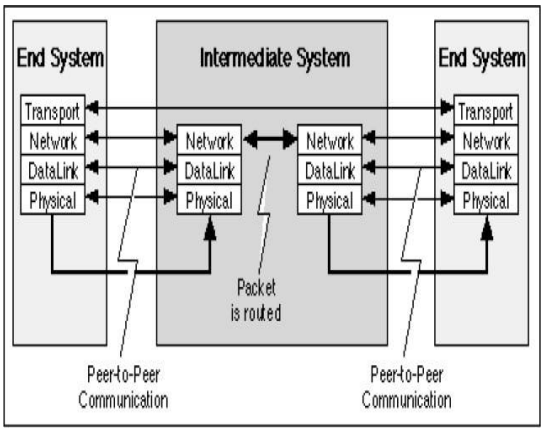
UNIT – IV : TRANSPORT LAYER

Introduction – Transport Layer Protocols – Services – Port Numbers – User Datagram Protocol – Transmission Control Protocol – SCTP.

1. INTRODUCTION

- The transport layer is the fourth layer of the OSI model and is the core of the Internet model.
- It responds to service requests from the session layer and issues service requests to the network Layer.
- The transport layer provides transparent transfer of data between hosts.
- It provides end-to-end control and information transfer with the quality of service needed by the application program.
- It is the first true end-to-end layer, implemented in all End Systems (ES).





TRANSPORT LAYER FUNCTIONS / SERVICES

- The transport layer is located between the network layer and the application layer.
- The transport layer is responsible for providing services to the application layer; it receives services from the network layer.
- The services that can be provided by the transport layer are
  1. Process-to-Process Communication
  2. Addressing : Port Numbers
  3. Encapsulation and Decapsulation
  4. Multiplexing and Demultiplexing
  5. Flow Control
  6. Error Control
  7. Congestion Control

Process-to-Process Communication

- The Transport Layer is responsible for delivering data to the appropriate application process on the host computers.
- This involves multiplexing of data from different application processes, i.e. forming data packets, and adding source and destination port numbers in the header of each Transport Layer data packet.
- Together with the source and destination IP address, the port numbers constitutes a network socket, i.e. an identification address of the process-to-process communication.

Addressing: Port Numbers

- Ports are the essential ways to address multiple entities in the same location.
- Using port addressing it is possible to use more than one network-based application at the same time.
- Three types of Port numbers are used :
  - ✓ ***Well-known ports*** - These are permanent port numbers. They range between 0 to 1023.These port numbers are used by Server Process.
  - ✓ ***Registered ports -*** The ports ranging from 1024 to 49,151 are not assigned or controlled.
  - ✓ ***Ephemeral ports (Dynamic Ports) –*** These are temporary port numbers. They range between 49152–65535.These port numbers are used by Client Process**.**

Encapsulation and Decapsulation

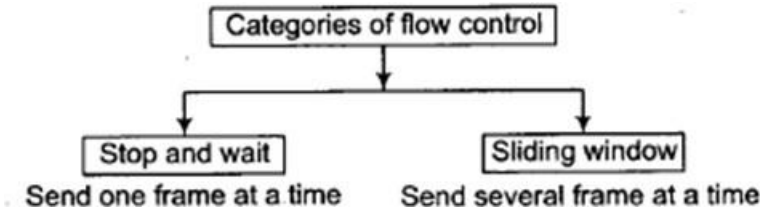
- To send a message from one process to another, the transport-layer protocol encapsulates and decapsulates messages.
- Encapsulation happens at the sender site. The transport layer receives the data and adds the transport-layer header.
- Decapsulation happens at the receiver site. When the message arrives at the destination transport layer, the header is dropped and the transport layer delivers the message to the process running at the application layer.

Multiplexing and Demultiplexing

- Whenever an entity accepts items from more than one source, this is referred to as ***multiplexing*** (many to one).
- Whenever an entity delivers items to more than one source, this is referred to as ***demultiplexing*** (one to many).
- The transport layer at the source performs multiplexing
- The transport layer at the destination performs demultiplexing

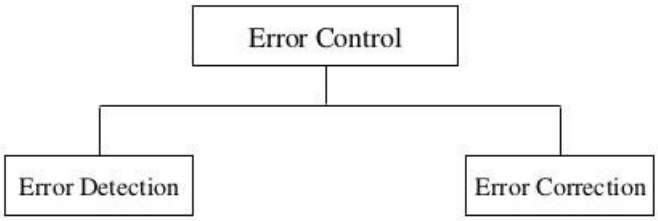
Flow Control

- Flow Control is the process of managing the rate of data transmission between two nodes to prevent a fast sender from overwhelming a slow receiver.
- It provides a mechanism for the receiver to control the transmission speed, so that the receiving node is not overwhelmed with data from transmitting node.



Error Control

- Error control at the transport layer is responsible for
  1. Detecting and discarding corrupted packets.
  2. Keeping track of lost and discarded packets and resending them.
  3. Recognizing duplicate packets and discarding them.
  4. Buffering out-of-order packets until the missing packets arrive.
- Error Control involves Error Detection and Error Correction



Congestion Control

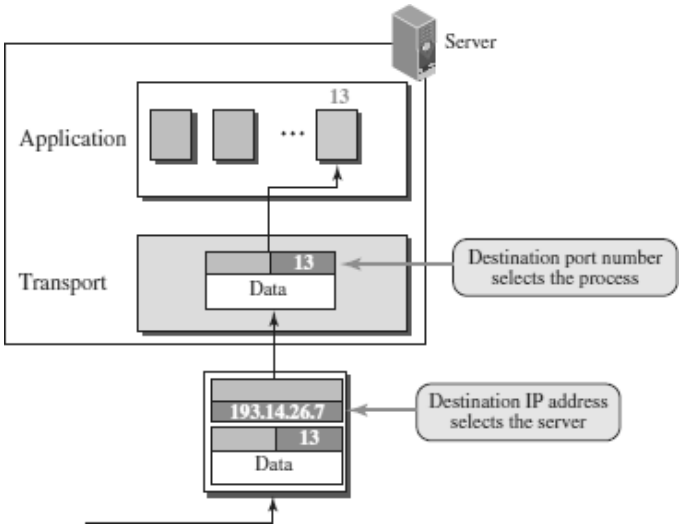
- Congestion in a network may occur if the *load* on the network (the number of packets sent to the network) is greater than the *capacity* of the network (the number of packets a network can handle).
- Congestion control refers to the mechanisms and techniques that control the congestion and keep the load below the capacity.
- Congestion Control refers to techniques and mechanisms that can either prevent congestion, before it happens, or remove congestion, after it has happened
- Congestion control mechanisms are divided into two categories,
  1. Open loop - prevent the congestion before it happens.
  2. Closed loop - remove the congestion after it happens.

---

2. PORT NUMBERS

- A transport-layer protocol usually has several responsibilities.
- One is to create a process-to-process communication.
- Processes are programs that run on hosts. It could be either *server* or *client.*
- A process on the local host, called a *client,* needs services from a process usually on the remote host, called a *server.*
- Processes are assigned a unique 16-bit *port number* on that host.
- Port numbers provide end-to-end addresses at the transport layer
- They also provide multiplexing and demultiplexing at this layer.

- The port numbers are integers between 0 and 65,535 .



ICANN (Internet Corporation for Assigned Names and Numbers) has divided the port numbers into three ranges:

- ✓**Well-known ports**
- ✓**Registered**
- ✓**Ephemeral ports (Dynamic Ports)**
- ✓

WELL-KNOWN PORTS

- These are permanent port numbers used by the servers.
- They range between 0 to 1023.
- This port number cannot be chosen randomly.
- These port numbers are universal port numbers for servers.
- Every client process knows the well-known port number of the corresponding server process.
- For example, while the daytime client process, a well-known client program, can use an ephemeral (temporary) port number, 52,000, to identify itself, the daytime server process must use the well-known (permanent) port number 13.

Some well-known ports

Port	*Protocol*	*Description*
7	Echo	Echoes back a received datagram
9	Discard	Discards any datagram that is received
11	Users	Active users
13	Daytime	Returns the date and the time
17	Quote	Returns a quote of the day
19	Chargen	Returns a string of characters
20	FTP-data	File Transfer Protocol
21	FTP-21	File Transfer Protocol
23	TELNET	Terminal Network
25	SMTP	Simple Mail Transfer Protocol
53	DNS	Domain Name Service
67	DHCP	Dynamic Host Configuration Protocol
69	TFTP	Trivial File Transfer Protocol
80	HTTP	HyperText Transfer Protocol
111	RPC	Remote Procedure Call
123	NTP	Network Time Protocol
161	SNMP-server	Simple Network Management Protocol
162	SNMP-client	Simple Network Management Protocol

EPHEMERAL PORTS (DYNAMIC PORTS)

- The client program defines itself with a port number, called the ***ephemeral port number.***
- The word *ephemeral* means "short-lived" and is used because the life of a client is normally short.
- An ephemeral port number is recommended to be greater than 1023.
- These port number ranges from 49,152 to 65,535 .
- They are neither controlled nor registered. They can be used as temporary or private port numbers.

REGISTERED PORTS

- The ports ranging from 1024 to 49,151 are not assigned or controlled.

3. TRANSPORT LAYER PROTOCOLS

- Three protocols are associated with the Transport layer.
- They are
  - **(1) UDP –User Datagram Protocol**
  - **(2) TCP – Transmission Control Protocol**
  - **(3) SCTP - Stream Control Transmission Protocol**
- Each protocol provides a different type of service and should be used appropriately.

UDP - UDP is an unreliable connectionless transport-layer protocol used for its simplicity and efficiency in applications where error control can be provided by the application-layer process.

TCP - TCP is a reliable connection-oriented protocol that can be used in any application where reliability is important.

SCTP - SCTP is a new transport-layer protocol designed to combine some features of UDP and TCP in an effort to create a better protocol for multimedia communication.

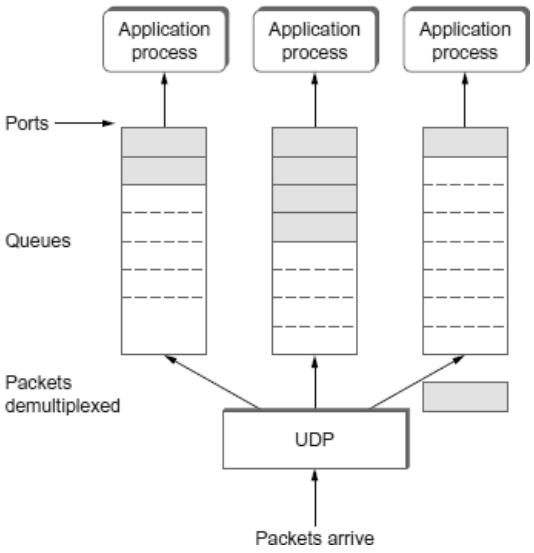
Position of transport-layer protocols in the TCP/IP protocol suite

4. USER DATAGRAM PROTOCOL (UDP)

- User Datagram Protocol (UDP) is a connectionless, unreliable transport protocol.
- UDP adds process-to-process communication to best-effort service provided by IP.
- UDP is a very simple protocol using a minimum of overhead.
- UDP is a simple demultiplexer, which allows multiple processes on each host to communicate.
- UDP does not provide flow control , reliable or ordered delivery.
- UDP can be used to send small message where reliability is not expected.
- Sending a small message using UDP takes much less interaction between the sender and receiver.
- UDP allow processes to indirectly identify each other using an abstract locator called port or mailbox

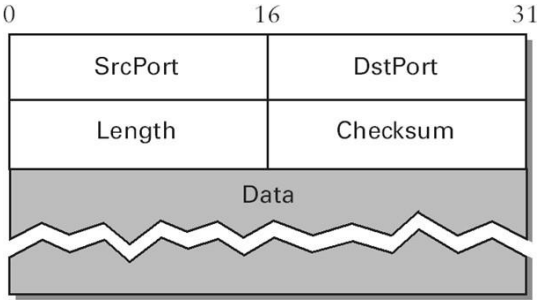
UDP PORTS

- Processes (server/client) are identified by an abstract locator known as port.
- Server accepts message at *well known port*.
- Some well-known UDP ports are 7–Echo, 53–DNS, 111–RPC, 161–SNMP, etc.
- < *port*, *host* > pair is used as key for demultiplexing.
- Ports are implemented as a *message queue*.
- When a message arrives, UDP *appends* it to end of the queue.
- When queue is *full*, the message is discarded.
- When a message is *read*, it is removed from the queue.
- 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 DATAGRAM (PACKET) FORMAT

- UDP packets are known as user *datagrams* .
- These ***user datagrams,*** have a fixed-size header of 8 bytes made of four fields, each of 2 bytes (16 bits).



Source Port Number

- Port number used by process on source host with 16 bits long.
- If the source host is client (sending request) then the port number is an temporary one requested by the process and chosen by UDP.
- If the source is server (sending response) then it is well known port number.

Destination Port Number

- Port number used by process on Destination host with 16 bits long.
- If the destination host is the server (a client sending request) then the port number is a well known port number.
- If the destination host is client (a server sending response) then port number is an temporary one copied by server from the request packet.

Length

- This field denotes the total length of the UDP Packet (Header plus data)
- The total length of any UDP datagram can be from 0 to 65,535 bytes.

Checksum

- UDP computes its checksum over the UDP header, the contents of the message body, and something called the pseudoheader.
- The pseudoheader consists of three fields from the IP header—protocol number, source IP address, destination IP address plus the UDP length field.

Data

- Data field defines tha actual payload to be transmitted.
- Its size is variable**.**

UDP SERVICES

Process-to-Process Communication

- UDP provides process-to-process communication using socket addresses**,** a combination of IP addresses and port numbers.

Connectionless Services

- UDP provides a connectionless service.
- There is no connection establishment and no connection termination .
- Each user datagram sent by UDP is an independent datagram.
- There is no relationship between the different user datagrams even if they are
- coming from the same source process and going to the same destination program.
- The user datagrams are not numbered.
- Each user datagram can travel on a different path.

Flow Control

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

Error Control

- There is no error control mechanism in UDP except for the checksum.
- This means that the sender does not know if a message has been lost or duplicated.
- When the receiver detects an error through the checksum, the user datagram is silently discarded.

- The lack of error control means that the process using UDP should provide for this service, if needed.

Checksum

- UDP checksum calculation includes three sections: a pseudoheader, the UDP header, and the data coming from the application layer.
- The pseudoheader is the part of the header in which the user datagram is to be encapsulated with some fields filled with 0s.

Optional Inclusion of Checksum

- The sender of a UDP packet can choose not to calculate the checksum.
- In this case, the checksum field is filled with all 0s before being sent.
- In the situation where the sender decides to calculate the checksum, but it happens that the result is all 0s, the checksum is changed to all 1s before the packet is sent.
- In other words, the sender complements the sum two times.

Congestion Control

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

Encapsulation and Decapsulation

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

Queuing

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

Multiplexing and Demultiplexing

- In a host running a transport protocol suite, there is only one UDP but possibly several processes that may want to use the services of UDP.
- To handle this situation, UDP multiplexes and demultiplexes.

APPLICATIONS OF UDP

- UDP is used for management processes such as SNMP.
- UDP is used for route updating protocols such as RIP.
- UDP is a suitable transport protocol for multicasting. Multicasting capability is embedded in the UDP software
- UDP is suitable for a process with internal flow and error control mechanisms such as Trivial File Transfer Protocol (TFTP).
- UDP is suitable for a process that requires simple request-response communication with little concern for flow and error control.
- UDP is normally used for interactive real-time applications that cannot tolerate uneven delay between sections of a received message.

---

5. 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.

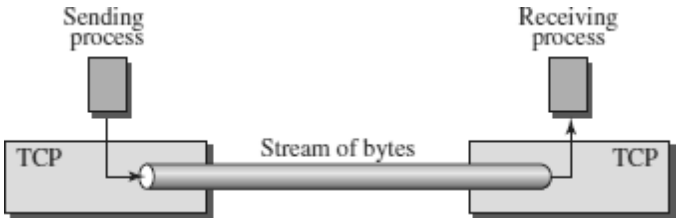
TCP SERVICES

Process-to-Process Communication

- TCP provides process-to-process communication using port numbers.

Stream Delivery Service

- TCP is a stream-oriented protocol.
- TCP allows the sending process to deliver data as a stream of bytes and allows the receiving process to obtain data as a stream of bytes.
- TCP creates an environment in which the two processes seem to be connected by an imaginary "tube" that carries their bytes across the Internet.
- The sending process produces (writes to) the stream and the receiving process consumes (reads from) it.



Full-Duplex Communication

- TCP offers full-duplex service, where data can flow in both directions at the same time.
- Each TCP endpoint then has its own sending and receiving buffer, and segments move in both directions.

Multiplexing and Demultiplexing

TCP performs multiplexing at the sender and demultiplexing at the receiver.

Connection-Oriented Service

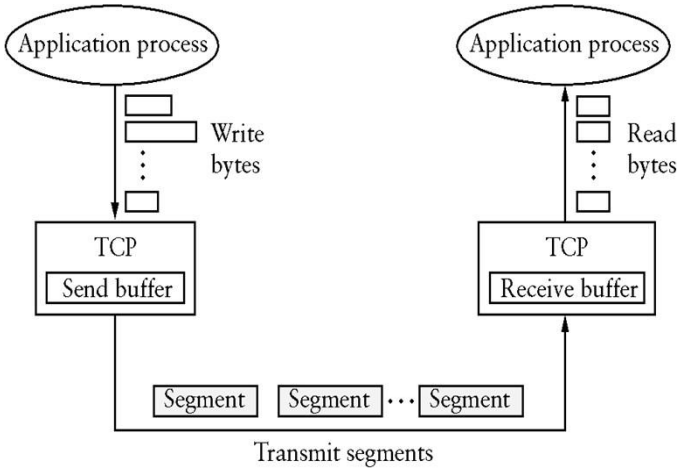
- TCP is a connection-oriented protocol.
- A connection needs to be established for each pair of processes.
- When a process at site A wants to send to and receive data from another process at site B, the following three phases occur:
  1. The two TCP's establish a logical connection between them.
 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

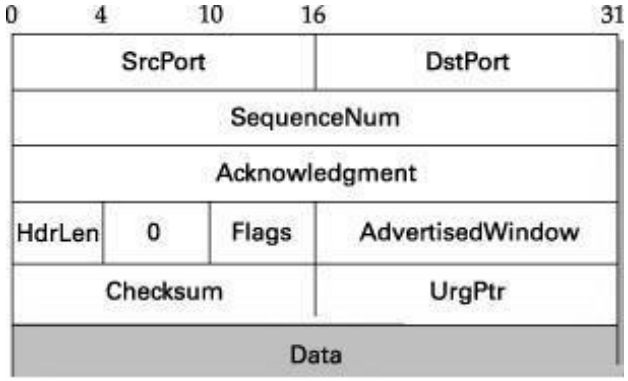
- A packet in TCP is called a segment.
- Data unit exchanged between TCP peers are called ***segments*.**
- A TCP segment encapsulates the data received from the application layer.
- The TCP segment is encapsulated in an IP datagram, which in turn is encapsulated in a frame at the data-link layer.



- 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 does not, itself, transmit individual bytes over the Internet.
- 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.
- TCP connection supports byte streams flowing in both directions.
- The packets exchanged between TCP peers are called segments, since each one carries a segment of the byte stream.

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.

- **URG** — segment contains urgent data.
- **ACK** — value of acknowledgment field is valid.
- **PUSH** — sender has invoked the push operation.
- **RESET** — receiver wants to abort the connection.
- **SYN** — synchronize sequence numbers during connection establishment.
- **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.

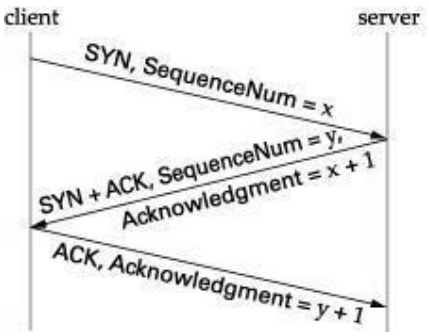
Options - There can be up to 40 bytes of optional information in the TCP header.

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).

- 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,"
- A timer is scheduled for each of the first two segments, and if the expected response is not received, the segment is retransmitted.

Data Transfer

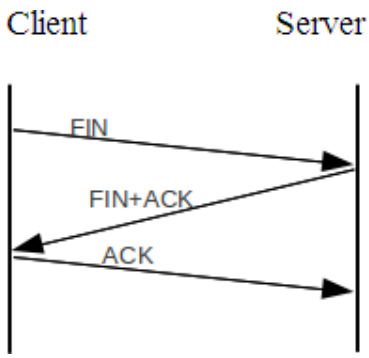
- After connection is established, bidirectional data transfer can take place.
- The client and server can send data and acknowledgments in both directions.
- The data traveling in the same direction as an acknowledgment are carried on the same segment.
- The acknowledgment is piggybacked with the data.

Connection Termination

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

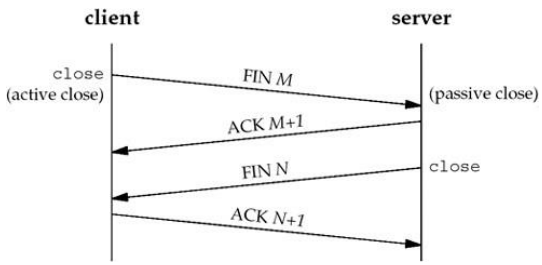
Three-way Close and Half-Close

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

Half-Close—Client stops sending but receives data.

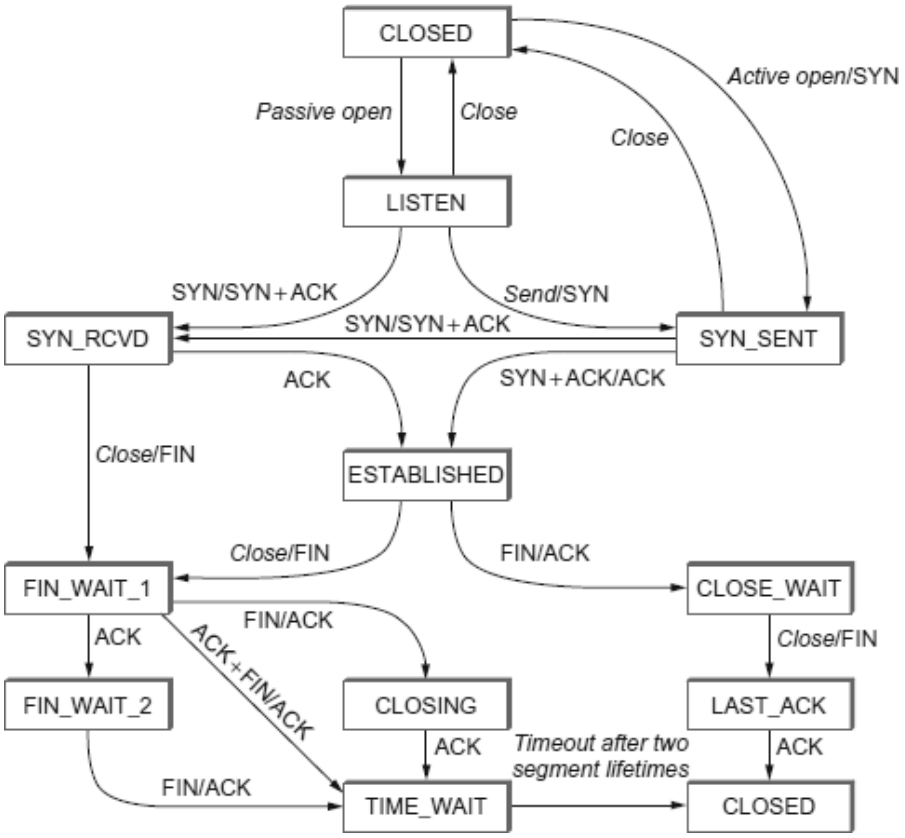


- Client half-closes the connection by sending a FIN segment.
- Server sends an ACK segment.
- Data transfer from client to the server *stops*.
- After sending all data, server sends FIN segment to client, which is acknowledged by the client.

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:

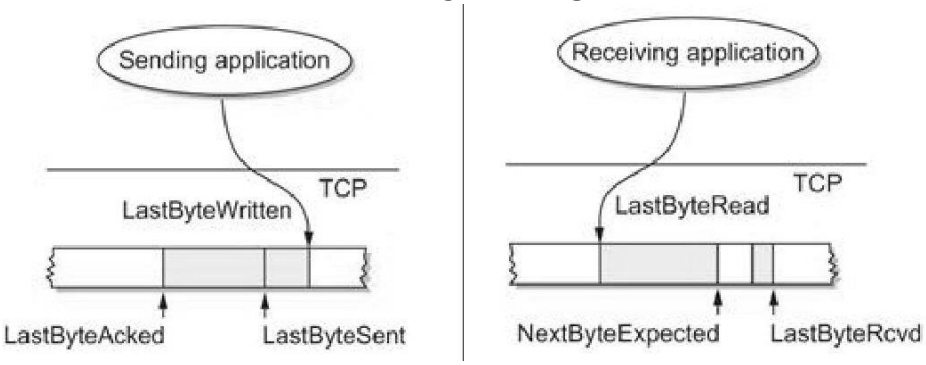
ESTABLISHED → CLOSE_WAIT → LAST_ACK → CLOSED

Simultaneous close:

ESTABLISHED → FIN_WAIT_1 → CLOSING → TIME_WAIT → CLOSED

TCP FLOW CONTROL

- TCP uses a variant of sliding window known as adaptive flow control that:
  - o guarantees *reliable* delivery of data
  - o ensures *ordered* delivery of data
  - ○enforces *flow control* at the sender
- Receiver advertises its window size to the sender using AdvertisedWindow field.
- Sender thus cannot have *unacknowledged* data greater than AdvertisedWindow.



Send Buffer

- Sending TCP maintains *send buffer* which contains 3 segments

  (1) acknowledged data

  (2) unacknowledged data

  (3) data to be transmitted.

- Send buffer maintains three *pointers*

  (1) LastByteAcked, (2) LastByteSent, and (3) LastByteWritten

  such that:

  $$LastByteAcked \leq LastByteSent \leq LastByteWritten$$

- A byte can be sent only *after* being written and only a sent byte *can be* acknowledged.
- Bytes to the *left* of LastByteAcked are not kept as it had been acknowledged.

Receive Buffer

- Receiving TCP maintains *receive* buffer to hold data even if it arrives out-of-order.
- Receive buffer maintains three *pointers* namely

  (1) LastByteRead, (2) NextByteExpected, and (3) LastByteRcvd

  such that:

  $$LastByteRead \leq NextByteExpected \leq LastByteRcvd + 1$$

- A byte *cannot* be read until that byte and all preceding bytes have been received.
- If data is received *in order*, then NextByteExpected = LastByteRcvd + 1
- Bytes to the *left* of LastByteRead are not buffered, since it is read by the application.

Flow Control in TCP

- Size of *send* and *receive* buffer is *MaxSendBuffer* and *MaxRcvBuffer* respectively.
- Sending TCP prevents *overflowing* of send buffer by maintaining

  $$LastByteWritten - LastByteAcked \leq MaxSendBuffer$$

- Receiving TCP avoids *overflowing* its receive buffer by maintaining

  $$LastByteRcvd - LastByteRead \leq MaxRcvBuffer$$

- Receiver *throttles* the sender by having AdvertisedWindow based on *free* space

available for buffering.

AdvertisedWindow = MaxRcvBuffer − ((NextByteExpected − 1) – LastByteRead)

- Sending TCP *adheres* to AdvertisedWindow by computing EffectiveWindow that *limits* how much data it should send.

EffectiveWindow = AdvertisedWindow − (LastByteSent − LastByteAcked)

- When data arrives, LastByteRcvd moves to its right and AdvertisedWindow shrinks.
- Receiver acknowledges only, if preceding bytes have arrived.
- AdvertisedWindow *expands* when data is *read* by the application.
    - o If data is read as *fast* as it arrives then

AdvertisedWindow = MaxRcvBuffer

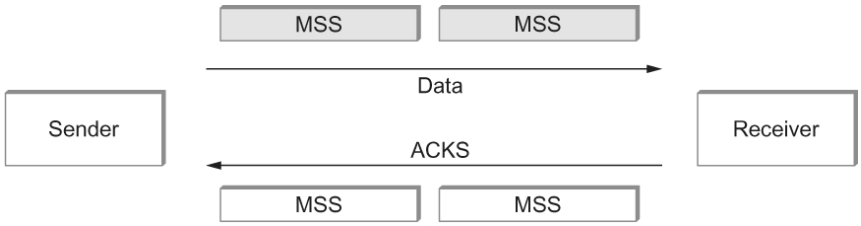
    - o If data is read *slowly*, it eventually leads to a AdvertisedWindow of size 0.
- AdvertisedWindow field is designed to allow sender to keep the pipe *full*.

TCP TRANSMISSION

- TCP has three mechanism to trigger the transmission of a segment.
- They are
    - o Maximum Segment Size (MSS) - Silly Window Syndrome
    - o Timeout - Nagle's Algorithm

Silly Window Syndrome

- When either the sending application program creates data slowly or the receiving application program consumes data slowly, or both, problems arise.
- Any of these situations results in the sending of data in very small segments, which reduces the efficiency of the operation.
- This problem is called the silly window syndrome.
- The sending TCP may create a silly window syndrome if it is serving an application program that creates data slowly, for example, 1 byte at a time.
- The application program writes 1 byte at a time into the buffer of the sending TCP.
- The result is a lot of 1-byte segments that are traveling through an internet.
- The solution is to prevent the sending TCP from sending the data byte by byte.
- The sending TCP must be forced to wait and collect data to send in a larger block.



Nagle's Algorithm

- If there is data to send but is less than MSS, then we may want to wait some amount of time before sending the available data
- If we wait too long, then it may delay the process.
- If we don't wait long enough, it may end up sending small segments resulting in Silly Window Syndrome.
- The solution is to introduce a timer and to transmit when the timer expires
- Nagle introduced an algorithm for solving this problem

```
When the application produces data to send
    if (both the available data and the window) = MSS
        Send the full segment
    else
        if (there is unACKed data)
            Buffer the new data until an ACK arrives
        else
            Send all the new data now
```

TCP CONGESTION CONTROL

- Congestion occurs if load (number of packets sent) is greater than capacity of the network (number of packets a network can handle).
- When load is less than network capacity, throughput increases proportionally.
- When load exceeds capacity, queues become full and the routers discard some packets and throughput declines sharply.
- When too many packets are contending for the same link
    - The queue overflows
    - Packets get dropped
    - Network is congested
- Network should provide a congestion control mechanism to deal with such a situation.
- TCP maintains a variable called *CongestionWindow* for each *connection.*
- TCP Congestion Control mechanisms are:

1. Additive Increase / Multiplicative Decrease (AIMD)
2. Slow Start
3. Fast Retransmit and Fast Recovery

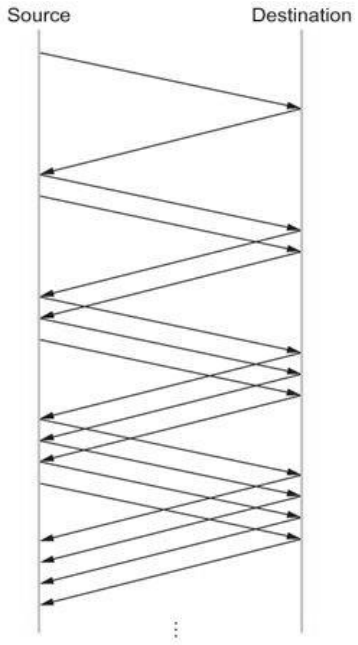
Additive Increase / Multiplicative Decrease (AIMD)

- TCP source *initializes* CongestionWindow based on congestion level in the network.
- Source *increases* CongestionWindow when level of congestion goes down and *decreases* the same when level of congestion goes up.
- TCP interprets *timeouts* as a sign of congestion and reduces the rate of transmission.
- On timeout, source reduces its CongestionWindow by half, i.e., *multiplicative decrease*. For example, if CongestionWindow = 16 packets, after timeout it is 8.
- Value of CongestionWindow is never less than maximum segment size (MSS).
- When ACK arrives CongestionWindow is *incremented* marginally, i.e., *additive increase.*

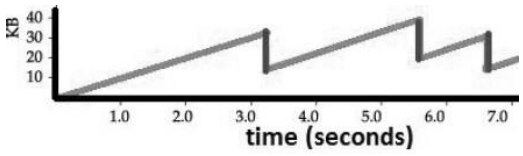
  Increment = MSS × (MSS/CongestionWindow)

  CongestionWindow += Increment

- For *example*, when ACK arrives for 1 packet, 2 packets are sent. When ACK for both packets arrive, 3 packets are sent and so on.
- CongestionWindow increases and decreases throughout *lifetime* of the connection.



- When CongestionWindow is plotted as a function of time, a *saw-tooth* pattern results.



CongestionWindow Trace

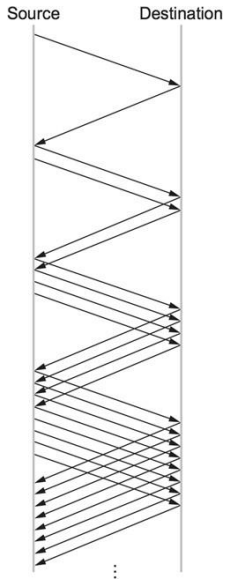
Slow Start

- Slow start is used to increase CongestionWindow *exponentially* from a cold start.
- Source TCP *initializes* CongestionWindow to one packet.
- TCP *doubles* the number of packets sent every RTT on successful transmission.
- When ACK arrives for first packet TCP adds 1 packet to CongestionWindow and sends two packets.
- When two ACKs arrive, TCP increments CongestionWindow by 2 packets and sends four packets and so on.
- Instead of sending entire permissible packets at once (bursty traffic), packets are sent in a phased manner, i.e., *slow start*.
- Initially TCP has no idea about congestion, henceforth it increases CongestionWindow rapidly until there is a timeout. On timeout:

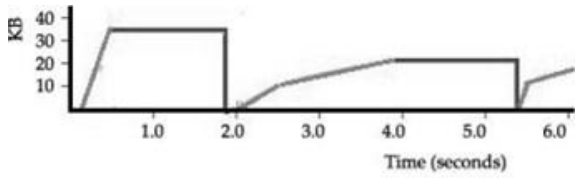
  CongestionThreshold = CongestionWindow/ 2

  CongestionWindow = 1

- Slow start is repeated until CongestionWindow reaches CongestionThreshold and thereafter 1 packet per RTT.



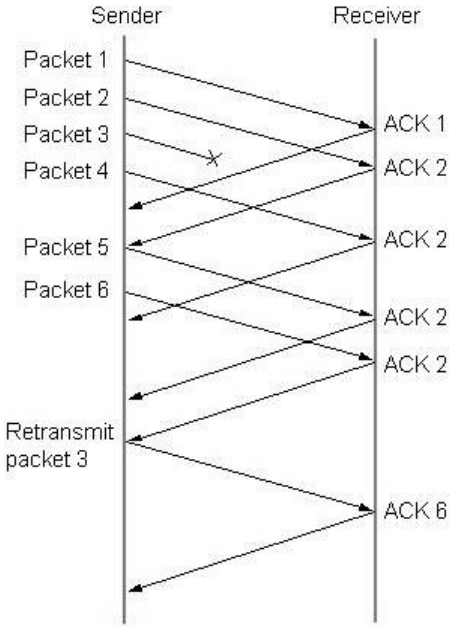
- The congestion window trace will look like



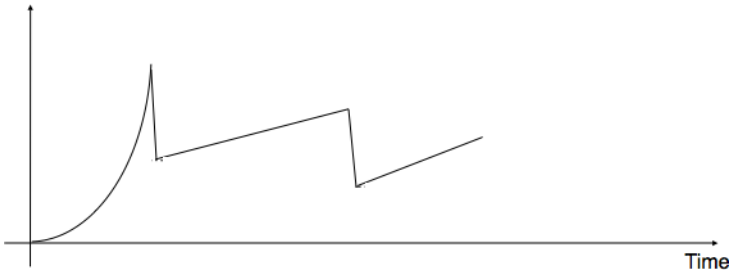
CongestionWindow Trace

Fast Retransmit And Fast Recovery

- TCP timeouts led to long periods of time during which the connection went dead while waiting for a timer to expire.
- Fast retransmit is a heuristic approach that *triggers* retransmission of a dropped packet sooner than the regular timeout mechanism. It *does not* replace regular timeouts.
- When a packet arrives out of order, receiving TCP resends the same acknowledgment (*duplicate ACK*) it sent last time.
- When *three duplicate* ACK arrives at the sender, it infers that corresponding packet may be lost due to congestion and retransmits that packet. This is called ***fast retransmit*** before regular timeout.
- When packet loss is detected using fast retransmit, the slow start phase is replaced by additive increase, multiplicative decrease method. This is known as ***fast recovery***.
- Instead of setting CongestionWindow to one packet, this method uses the ACKs that are still in pipe to clock the sending of packets.
- Slow start is only used at the beginning of a connection and after *regular* timeout. At other times, it follows a pure AIMD pattern.
-



- For example, packets 1 and 2 are received whereas packet 3 gets lost.
  - Receiver sends a duplicate ACK for packet 2 when packet 4 arrives.
  - Sender receives 3 duplicate ACKs after sending packet 6 retransmits packet 3.
  - When packet 3 is received, receiver sends cumulative ACK up to packet 6.
- The congestion window trace will look like



TCP CONGESTION AVOIDANCE

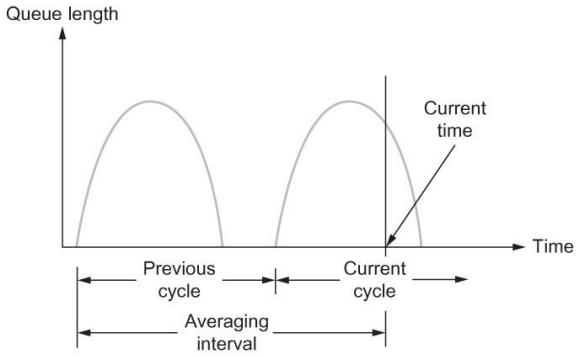
- Congestion avoidance mechanisms *prevent* congestion before it actually occurs.
- These mechanisms predict when congestion is about to happen and then to reduce the rate at which hosts send data just before packets start being discarded.
- TCP *creates* loss of packets in order to determine bandwidth of the connection.
- Routers *help* the end nodes by intimating when congestion is likely to occur.
- Congestion-avoidance mechanisms are:
  - DEC bit - Destination Experiencing Congestion Bit
  - RED - Random Early Detection

Dec Bit - Destination Experiencing Congestion Bit

- The first mechanism developed for use on the Digital Network Architecture (DNA).
- The idea is to evenly split the responsibility for congestion control between the routers and the end nodes.
- Each router monitors the load it is experiencing and explicitly notifies the end nodes when congestion is about to occur.
- This notification is implemented by setting a binary congestion bit in the packets that flow through the router; hence the name DECbit.

- The destination host then copies this congestion bit into the ACK it sends back to the source.
- The Source checks *how many* ACK has DEC bit set for previous window packets.
- If less than 50% of ACK have DEC bit set, then source *increases* its congestion window by 1 packet
- Otherwise, *decreases* the congestion window by 87.5%.
- Finally, the source adjusts its sending rate so as to avoid congestion.
- *Increase by 1, decrease by 0.875* rule was based on AIMD for stabilization.
- A single congestion bit is added to the packet header.
- Using a queue length of 1 as the trigger for setting the congestion bit.
- A router sets this bit in a packet if its average queue length is greater than or equal to 1 at the time the packet arrives.

Computing average queue length at a router using DEC bit



- Average queue length is measured over a time interval that includes the

***last busy + last idle cycle + current busy* cycle.**

- It calculates the average queue length by *dividing* the curve area with time interval.

Red - Random Early Detection

- The second mechanism of congestion avoidance is called as R*andom Early Detection (RED).*

- Each router is programmed to monitor its own queue length, and when it detects that there is congestion, it notifies the source to adjust its congestion window.
- RED differs from the DEC bit scheme by two ways:

a. In DECbit, explicit notification about congestion is sent to source, whereas RED implicitly notifies the source by dropping a few packets.

b. DECbit may lead to tail drop policy, whereas RED drops packet based on drop probability in a random manner. Drop each arriving packet with some ***drop probability*** whenever the queue length exceeds some *drop level.* This idea is *called* ***early random drop***.

Computation of average queue length using RED

- ***AvgLen = (1 − Weight) × AvgLen + Weight × SampleLen***

where $0 < \text{Weight} < 1$ and

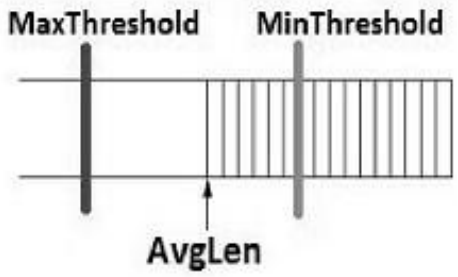
SampleLen – is the length of the queue when a sample measurement is made.

- The queue length is measured every time a new packet arrives at the gateway.
- RED has two queue length thresholds that trigger certain activity: ***MinThreshold and MaxThreshold***
- When a packet arrives at a gateway it compares Avglen with these two values according to the following rules.

if AvgLen ≤ MinThreshold
⟶ queue the packet

if MinThreshold < AvgLen < MaxThreshold
⟶ calculate probability P
⟶ drop the arriving packet with probability P

if AvgLen ≥ MaxThreshold
⟶ drop the arriving packet



6. STREAM CONTROL TRANSMISSION PROTOCOL (SCTP)

- Stream Control Transmission Protocol (SCTP) is a reliable, message-oriented transport layer protocol.
- SCTP has mixed features of TCP and UDP.
- SCTP maintains the message boundaries and detects the lost data, duplicate data as well as out-of-order data.
- SCTP provides the Congestion control as well as Flow control.
- SCTP is especially designed for internet applications as well as multimedia communication.

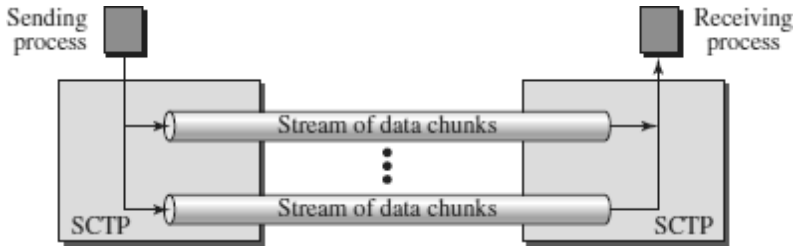
SCTP SERVICES

Process-to-Process Communication

- SCTP provides process-to-process communication.

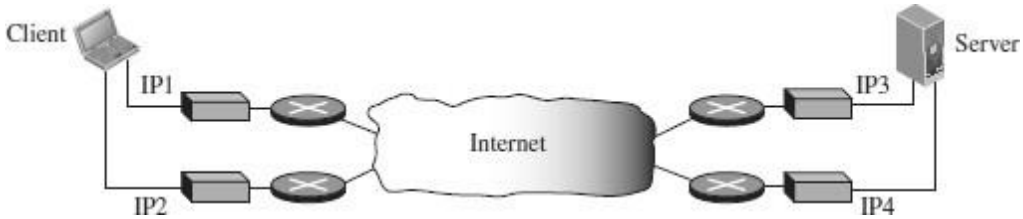
Multiple Streams

- SCTP allows multistream service in each connection, which is called ***association*** in SCTP terminology.
- If one of the streams is blocked, the other streams can still deliver their data.



Multihoming

- An SCTP association supports multihoming service**.**
- The sending and receiving host can define multiple IP addresses in each end for an association.
- In this fault-tolerant approach, when one path fails, another interface can be used for data delivery without interruption.



Full-Duplex Communication

- SCTP offers full-duplex service, where data can flow in both directions at the same time. Each SCTP then has a sending and receiving buffer and packets are sent in both directions.

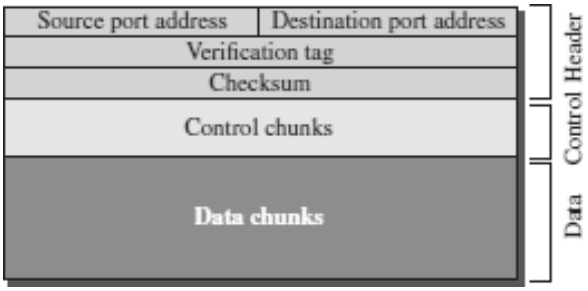
Connection-Oriented Service

- SCTP is a connection-oriented protocol.
- In SCTP, a connection is called an *association*.
- If a client wants to send and receive message from server , the steps are :
  Step1: The two **SCTPs** establish the connection with each other.
  Step2: Once the connection is established, the data gets exchanged in both the directions.
  Step3: Finally, the association is terminated.

Reliable Service

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

SCTP PACKET FORMAT



An SCTP packet has a mandatory general header and a set of blocks called chunks.

General Header

- The *general header* (packet header) defines the end points of each association to which the packet belongs
- It guarantees that the packet belongs to a particular association
- It also preserves the integrity of the contents of the packet including the header itself.
- There are four fields in the general header.

Source port

This field identifies the sending port.

Destination port

This field identifies the receiving port that hosts use to route the packet to the appropriate endpoint/application.

Verification tag

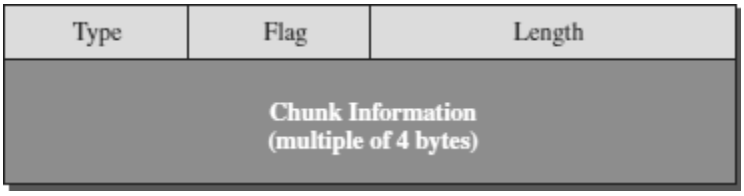
A 32-bit random value created during initialization to distinguish stale packets from a previous connection.

Checksum

The next field is a checksum. The size of the checksum is 32 bits. SCTP uses CRC-32 Checksum.

Chunks

- Control information or user data are carried in chunks.
- Chunks have a common layout.
- The first three fields are common to all chunks; the information field depends on the type of chunk.



- The type field can define up to 256 types of chunks. Only a few have been defined so far; the rest are reserved for future use.
- The flag field defines special flags that a particular chunk may need.
- The length field defines the total size of the chunk, in bytes, including the type, flag, and length fields.

Types of Chunks

- An SCTP association may send many packets, a packet may contain several chunks, and chunks may belong to different streams.
- SCTP defines two types of chunks - Control chunks and Data chunks.
- A control chunk controls and maintains the association.
- A data chunk carries user data.

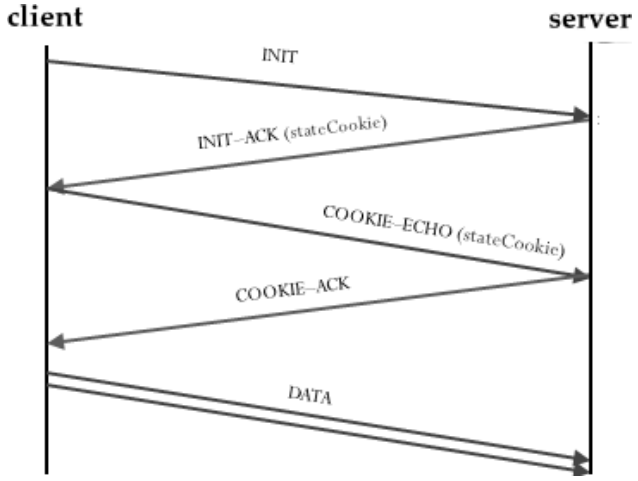
Type	*Chunk*	*Description*
0	DATA	User data
1	INIT	Sets up an association
2	INIT ACK	Acknowledges INIT chunk
3	SACK	Selective acknowledgment
4	HEARTBEAT	Probes the peer for liveliness
5	HEARTBEAT ACK	Acknowledges HEARTBEAT chunk
6	ABORT	Aborts an association
7	SHUTDOWN	Terminates an association
8	SHUTDOWN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

SCTP ASSOCIATION

- SCTP is a connection-oriented protocol.
- A connection in SCTP is called an *association* to emphasize multihoming.
- SCTP Associations consists of three phases:
  - Association Establishment
  - Data Transfer
  - Association Termination

Association Establishment

- Association establishment in SCTP requires a four-way handshake.
- In this procedure, a client process wants to establish an association with a server process using SCTP as the transport-layer protocol.
- The SCTP server needs to be prepared to receive any association (passive open).
- Association establishment, however, is initiated by the client (active open).



- The client sends the first packet, which contains an INIT chunk.
- The server sends the second packet, which contains an INIT ACK chunk. The INIT ACK also sends a cookie that defines the state of the server at this moment.
- The client sends the third packet, which includes a COOKIE ECHO chunk. This is a very simple chunk that echoes, without change, the cookie sent by the server. SCTP allows the inclusion of data chunks in this packet.
- The server sends the fourth packet, which includes the COOKIE ACK chunk that acknowledges the receipt of the COOKIE ECHO chunk. SCTP allows the inclusion of data chunks with this packet.

Data Transfer

- The whole purpose of an association is to transfer data between two ends.
- After the association is established, bidirectional data transfer can take place.
- The client and the server can both send data.
- SCTP supports piggybacking.

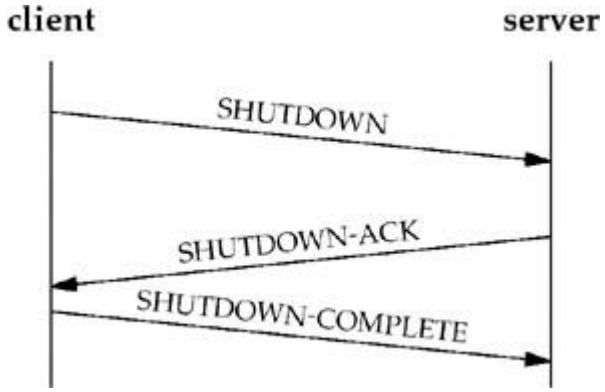
- Types of SCTP data Transfer :

1. **Multihoming Data Transfer**
   - Data transfer, by default, uses the primary address of the destination.
   - If the primary is not available, one of the alternative addresses is used.
   - This is called Multihoming Data Transfer.

2. **Multistream Delivery**
   - SCTP can support multiple streams, which means that the sender process can define different streams and a message can belong to one of these streams.
   - Each stream is assigned a stream identifier (SI) which uniquely defines that stream.
   - SCTP supports two types of data delivery in each stream: *ordered* (default) and *unordered.*

Association Termination

- In SCTP,either of the two parties involved in exchanging data (client or server) can close the connection.
- SCTP does not allow a "half closed" association. If one end closes the association, the other end must stop sending new data.
- If any data are left over in the queue of the recipient of the termination request, they are sent and the association is closed.
- Association termination uses three packets.



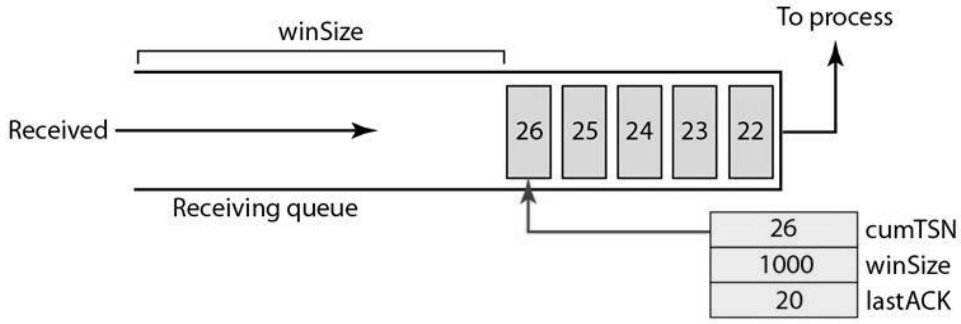
SCTP FLOW CONTROL

- Flow control in SCTP is similar to that in TCP.
- Current SCTP implementations use a byte-oriented window for flow control.

Receiver Site

- The receiver has one buffer (queue) and three variables.

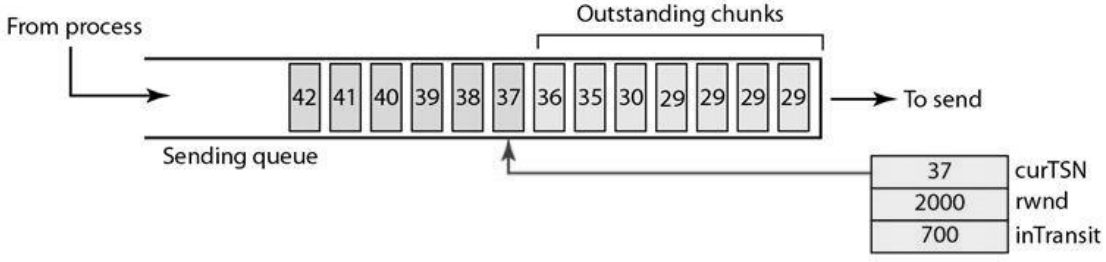
- The queue holds the received data chunks that have not yet been read by the process.
- The first variable holds the last TSN received, cumTSN.
- The second variable holds the available buffer size; winsize.
- The third variable holds the last accumulative acknowledgment, lastACK.
- The following figure shows the queue and variables at the receiver site.



- When the site receives a data chunk, it stores it at the end of the buffer (queue) and subtracts the size of the chunk from winSize.
- The TSN number of the chunk is stored in the cumTSN variable.
- When the process reads a chunk, it removes it from the queue and adds the size of the removed chunk to winSize (recycling).
- When the receiver decides to send a SACK, it checks the value of lastAck; if it is less than cumTSN, it sends a SACK with a cumulative TSN number equal to the cumTSN.
- It also includes the value of winSize as the advertised window size.

Sender Site

- The sender has one buffer (queue) and three variables: curTSN, rwnd, and inTransit.
- We assume each chunk is 100 bytes long. The buffer holds the chunks produced by the process that either have been sent or are ready to be sent.
- The first variable, curTSN, refers to the next chunk to be sent.
- All chunks in the queue with a TSN less than this value have been sent, but not acknowledged; they are outstanding.
- The second variable, rwnd, holds the last value advertised by the receiver (in bytes).
- The third variable, inTransit, holds the number of bytes in transit, bytes sent but not yet acknowledged.
- The following figure shows the queue and variables at the sender site.



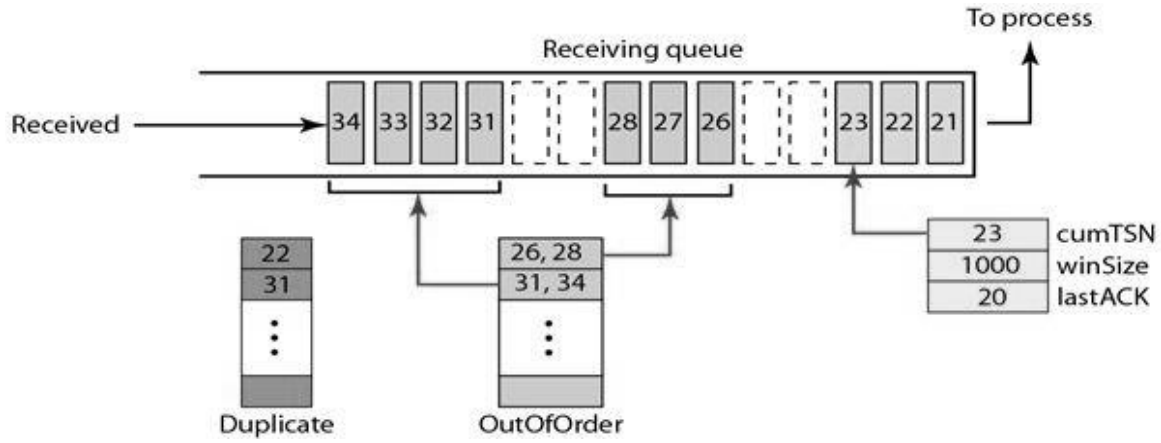
- A chunk pointed to by curTSN can be sent if the size of the data is less than or equal to the quantity rwnd - inTransit.
- After sending the chunk, the value of curTSN is incremented by 1 and now points to the next chunk to be sent.
- The value of inTransit is incremented by the size of the data in the transmitted chunk.
- When a SACK is received, the chunks with a TSN less than or equal to the cumulative TSN in the SACK are removed from the queue and discarded. The sender does not have to worry about them anymore.
- The value of inTransit is reduced by the total size of the discarded chunks.
- The value of rwnd is updated with the value of the advertised window in the SACK.

SCTP ERROR CONTROL

- SCTP is a reliable transport layer protocol.
- It uses a SACK chunk to report the state of the receiver buffer to the sender.
- Each implementation uses a different set of entities and timers for the receiver and sender sites.

Receiver Site

- The receiver stores all chunks that have arrived in its queue including the out-of-order ones. However, it leaves spaces for any missing chunks.
- It discards duplicate messages, but keeps track of them for reports to the sender.
- The following figure shows a typical design for the receiver site and the state of the receiving queue at a particular point in time.

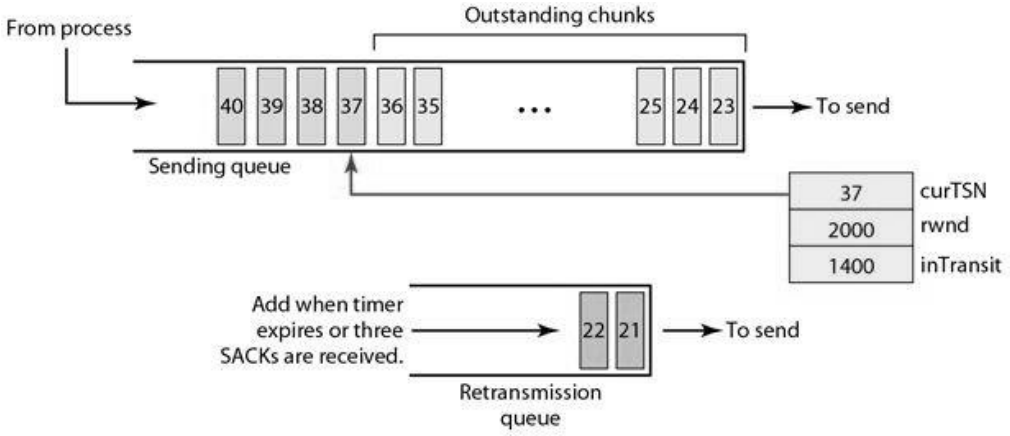


- The available window size is 1000 bytes.
- The last acknowledgment sent was for data chunk 20.
- Chunks 21 to 23 have been received in order.
- The first out-of-order block contains chunks 26 to 28.
- The second out-of-order block contains chunks 31 to 34.
- A variable holds the value of cumTSN.

- An array of variables keeps track of the beginning and the end of each block that is out of order.
- An array of variables holds the duplicate chunks received.
- There is no need for storing duplicate chunks in the queue and they will be discarded.

Sender Site

- At the sender site, it needs two buffers (queues): a sending queue and a retransmission queue.
- Three variables were used - rwnd, inTransit, and curTSN as described in the previous section.
- The following figure shows a typical design.



- The sending queue holds chunks 23 to 40.
- The chunks 23 to 36 have already been sent, but not acknowledged; they are outstanding chunks.
- The curTSN points to the next chunk to be sent (37).
- We assume that each chunk is 100 bytes, which means that 1400 bytes of data (chunks 23 to 36) is in transit.
- The sender at this moment has a retransmission queue.
- When a packet is sent, a retransmission timer starts for that packet (all data chunks in that packet).
- Some implementations use one single timer for the entire association, but other implementations use one timer for each packet.

SCTP CONGESTION CONTROL

- SCTP is a transport-layer protocol with packets subject to congestion in the network.
- The SCTP designers have used the same strategies for congestion control as those used in TCP.

NOTE : REFER TCP CONGESTION CONTROL