UNIT II TRANSPORT LAYER

Introduction – Transport-Layer Protocols: UDP – TCP: Connection Management – Flow control – Congestion Control – Congestion avoidance (DECbit, RED) – SCTP – Quality of Service

1. INTRODUCTION

* Name the services provided by transport layer protocol. **(2)**
* List the advantages of connection oriented services over connection less services. **(2)**

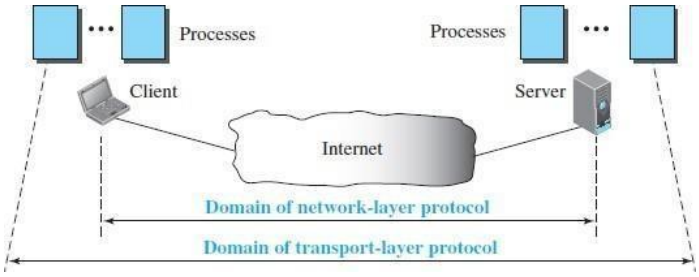
The transport layer is located between the application layer and the network layer. It provides a process-to-processcommunication between two application layers, one at the local host and the other at the remote host.

Transport-Layer Services

The transport layer is responsible for providing services to the application layer; it receives services from the network layer.

Process-to-Process Communication

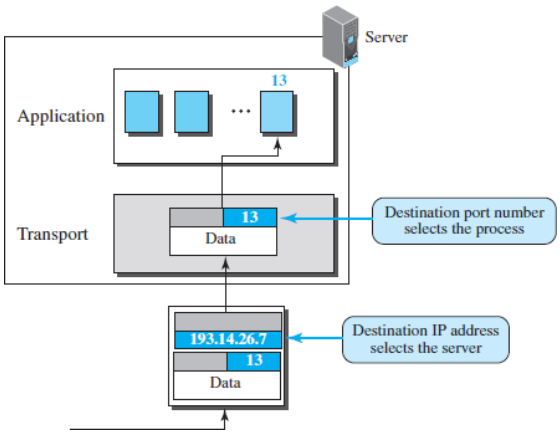
The first duty of a transport-layer protocol is to provide process-to-process communication**.** A process is an application-layer entity (running program) that uses the services of the transport layer. A network-layer protocol can deliver the message



only to the destination computer. However, this is an incomplete delivery. The message still needs to be handed to the correct process. This is where a transport- layer protocol takes over. A transport-layer protocol is responsible for delivery of the message to the appropriate process.

Addressing: Port Numbers

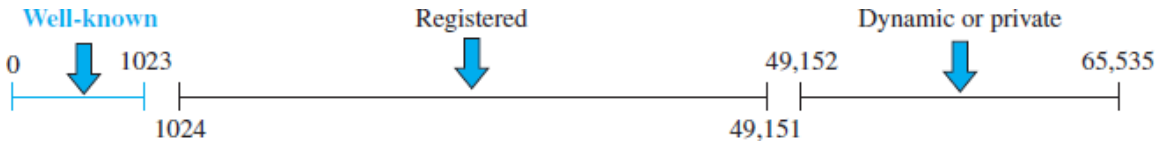
A remote computer can run several server programs at the same time, just as several local computers can run one or more client programs at the same time. For communication, we must define the local host, local process, remote host, and remote process. The local host and the remote host are defined using IP addresses. To define the processes, we need second identifiers, called ***port numbers.*** In the TCP/IP protocol suite, the port numbers are integers between 0 and 65,535 (16 bits).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 for some client/server programs to work properly. TCP/IP has decided to use universal port numbers for servers; these are called ***well- known port numbers.*** There are some exceptions to this rule; for example, there are clients that are assigned well-known port numbers. Every clientprocess knows the well-known port number of the corresponding server process.

ICANN Ranges

ICANN has divided the port numbers into three ranges: well-known,registered, and dynamic (or private), as shown below

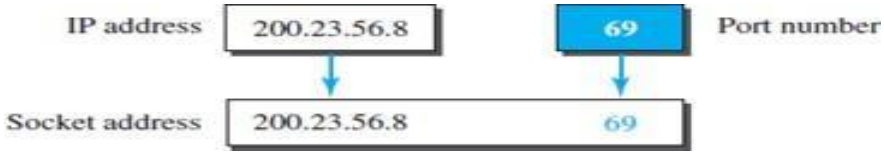


Well-known ports. The ports ranging from 0 to 1023 are assigned and controlled by ICANN. These are the well-known ports. ***Registered ports.*** The ports ranging from 1024 to 49,151 are not assigned or controlled by ICANN. They can only be registered with ICANN to prevent duplication.

Dynamic ports. The ports ranging from 49,152 to 65,535 are neither controlled nor registered. They can be used as temporary or private port numbers.

Socket Addresses

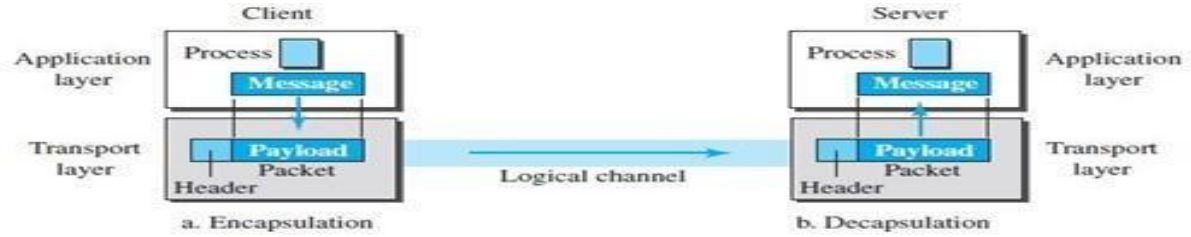
A transport-layer protocol in the TCP suite needs both the IP address and the port number,at each end, to make a connection. The combination of an IP address and a portnumber is called a ***socket address.***



Encapsulation and Decapsulation

Encapsulation happens at the sender site. When a process has a message to send, it passes the message to the transport layer along with a pair of socket addresses and some other pieces of information, which depend on the transport-layer protocol. The transport layer receives the data and adds the transport-layer header. The packets at the transport layer in the Internet are called *user datagrams, segments,* or *packets,* depending on what transport-layer protocol we use. In general discussion, we refer to transport-layer payloads as *packets.*

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. The sender socket address is passed to the process in case it needs to respond to the message received.



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.

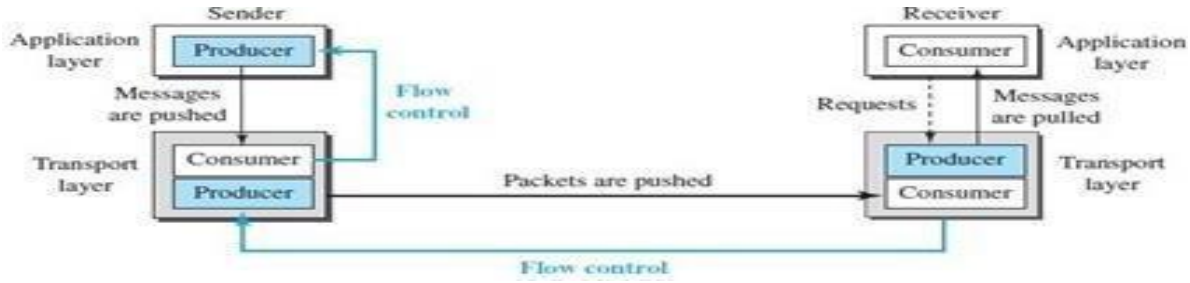
Pushing or Pulling

When the producer *pushes* the items, the consumer may be overwhelmed and there is a need for flow control, in the opposite direction, to prevent discarding of the items. In other words, the consumer needs to warn the producer to stop the delivery and to inform the producer when it is again ready to receive the items. When the consumer pulls the items, it requests them when it is ready. In this case, there is no need for flow control.

Flow Control at Transport Layer

In communication at the transport layer, we are dealing with four entities: sender process, sender transport layer, receiver transport layer, and receiver process. The sending process at the application layer is only a producer. It produces message chunks and pushes them to the transport layer. The sending transport layer has a double role: it is both a consumer and a producer. It consumes the messages pushed by the producer. It encapsulates the messages in packets and pushes them to the receiving transport layer. The receiving transport layer also has a double role: it is the consumer for the packets received from the sender and the producer that decapsulates the messages and delivers

them to the application layer. The last delivery, however, is normally a pulling delivery; the transport layer waits until the application-layer process asks for messages.



Buffers

Although flow control can be implemented in ways several, one of the solutions is normally to use two *buffers*: one at the sending transport layer and the other at the receiving transport layer.

A buffer is a set of memory locations that can hold packets at the sender and receiver. The flow control communication can occur by sending signals from the consumer to the producer.

When the buffer of the sending transport layer is full, it informs the application layer to stop passing chunks of messages; when there are some vacancies, it informs the application layer that it can pass message chunks again. When the buffer of the receiving transport layer is full, it informs the sending transport layer to stop sending packets. When there are some vacancies, it informs the sending transport layer that it can send packets again.

Error Control

In the Internet, since the underlying network layer (IP) is unreliable, we need to make

the transport layer reliable if the application requires reliability. Reliability can be achieved to add error control services to the transport layer. 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.

Sequence Numbers

Packets are numbered sequentially. However, because we need to include the sequence number of each packet in the header, we need to set a limit. If the header of the packet allows *m* bits for the sequence number, the sequence numbers range from 0 to 2*m*-1. For if *m* is 4, the only sequence numbers are 0 through 15, inclusive. However, we can wrap around the sequence. So the sequence numbers in this case are In other words, the sequence numbers are modulo 2*m*.

0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11,

Sliding Window

Since the sequence numbers use modulo 2*m*, a circle can represent the sequence numbers

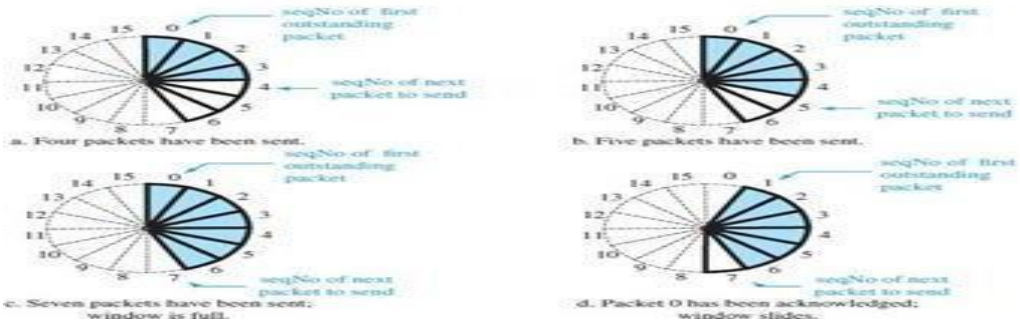
from 0 to 2*m* - 1. The buffer is represented as a set of slices, called the ***sliding window,*** that occupies part of the circle at any time. At the sender site, when a packet is sent, the corresponding slice is marked.

When all the slices are marked, it means that the buffer is full and no further messages can be accepted from the application layer. When an acknowledgment arrives, the corresponding slice is unmarked.

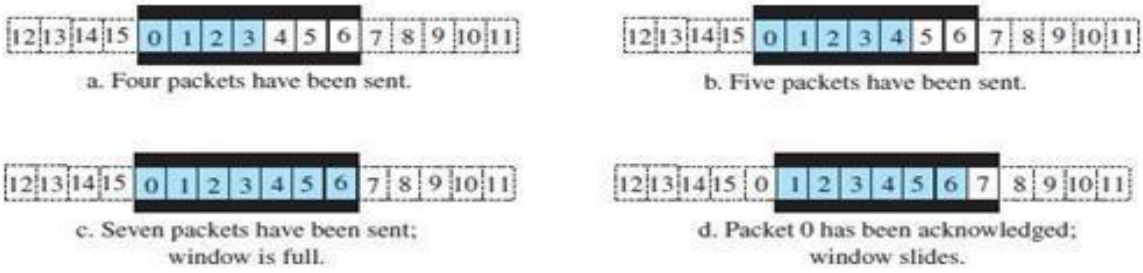
If some consecutive slices from the beginning of the window are unmarked, the window slides over the range of the corresponding sequence numbers to allow more free slices at the end of the window.

Figure shows the sliding window at the sender.

The sequence numbers are in modulo 16 (*m* 4) and the size of the window is 7. Note that the sliding window is just an abstraction: the actual situation uses computer variables to hold the sequence numbers of the next packet to be sent and the last packet sent.



Most protocols show the sliding window using linear representation. The idea is the same, but it normally takes less space on paper.



Connectionless and Connection-Oriented Protocols

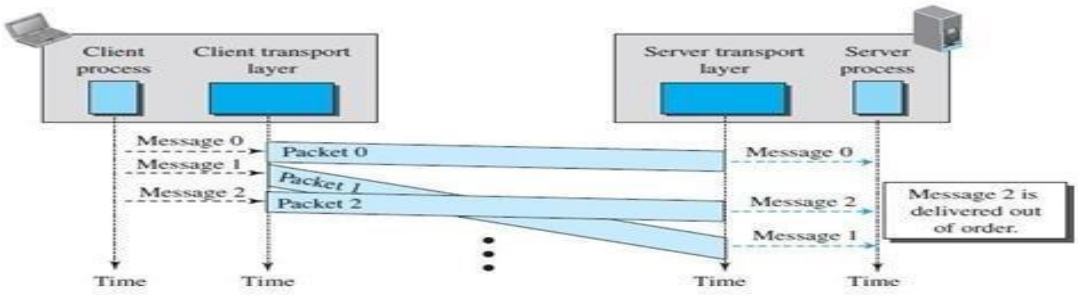
A transport-layer protocol, like a network-layer protocol, can provide two types of services: connectionless and connection- oriented. The nature of these services at the transport layer, however, is different from the ones at the network layer. At the network layer, a connectionless service may mean different paths for different datagrams belonging to the same message.

Connectionless Service

In a connectionless service, the source process (application program) needs to divide its message into chunks of data of the size acceptable by the transport layer and deliver

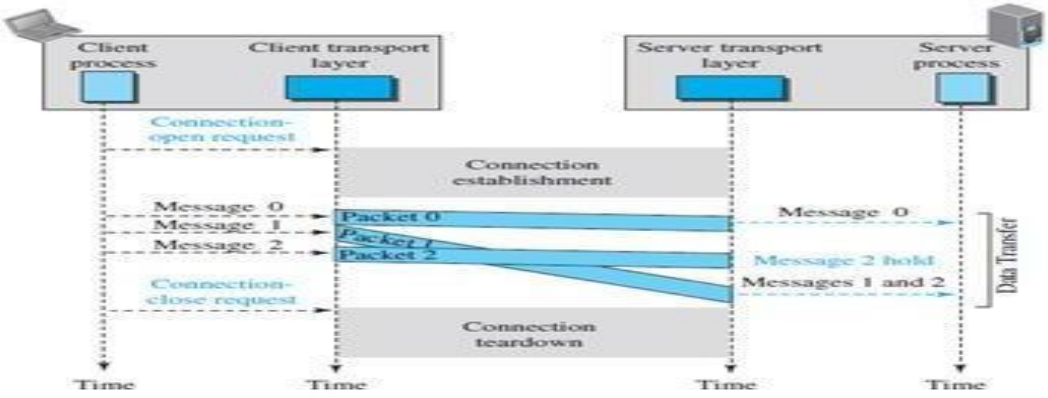
them to the transport layer one by one.

The transport layer treats each chunk as a single unit without any relation between the chunks. When a chunk arrives from the application layer, the transport layer encapsulates it in a packet and sends it. To show the independency of packets, assume that a client process has three chunks of messages to send to a server process. The chunks are handed over to the connectionless transport protocol in order. However, since there is no dependency between the packets at the transport layer, the packets may arrive out of order at the destination and will be delivered out of order to the server process.

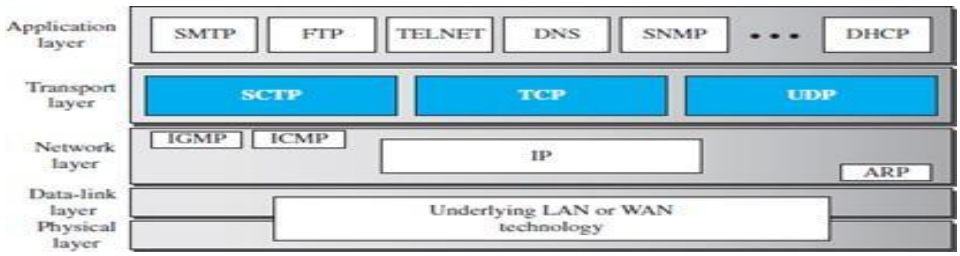


Connection-Oriented Service

In a connection-oriented service, the client and the server first need to establish a logical connection between themselves. The data exchange can only happen after the connection establishment. After data exchange, the connection needs to be torn down.



2.TRANSPORT-LAYER PROTOCOLS



Services

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 errorcontrol 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 that combines the features of UDP and TCP.

Port Numbers

Port numbers provide end-to-end addresses at the transport layer and allow multiplexing and demultiplexing at this layer, just as IP addresses do at the network layer.

Port	*Protocol*	*UDP*	*TCP*	*SCTP*	*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

3.USER DATAGRAM PROTOCOL

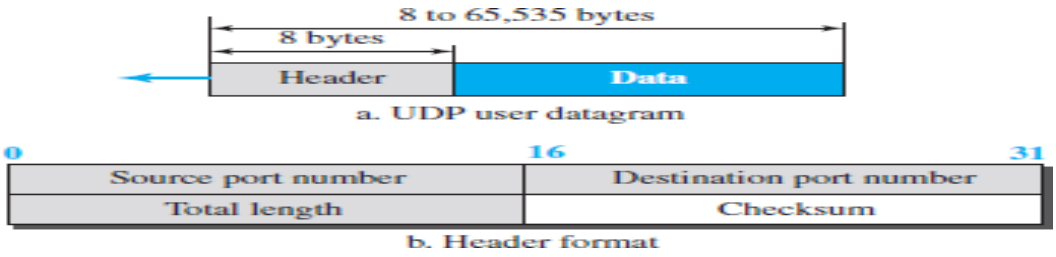
*How does UDP address flow control mechanism? **(2)**

* Name the services provided by User Datagram Protocol**. (2)**

The User Datagram Protocol (UDP) is a connectionless, unreliable transport protocol. It does not add anything to the services of IP except for providing process-to-process communication instead of host-to-host communication. UDP is a very simple protocol using a minimum of overhead. If a process wants to send a small message and does not care much about reliability, it can use UDP. Sending a small message using UDP takes much less interaction between the sender and receiver than using TCP**.**

User Datagram

UDP packets, called ***user datagrams,*** have a fixed-size header of 8 bytes made of four fields, each of 2 bytes (16 bits). The first two fields define the source and destination port numbers. The third field defines the total length of the user datagram, header plus data. The 16 bits can define a total length of 0 to 65,535 bytes. However, the total length needs to be less because a UDP user datagram is stored in an IP datagram with the total length of 65,535 bytes. The last field can carry the optional checksum.



Example

The following is the content of a UDP header in hexadecimal format.

CB84000D001C001C

- **a.** What is the source port number?
- **b.** What is the destination port number?
- **c.** What is the total length of the user datagram?
- **d.** What is the length of the data?
- **e.** Is the packet directed from a client to a server or vice versa?
- **f.** What is the client process?

Solution

- **a.** The source port number is the first four hexadecimal digits (CB84)16, which means that the source port number is 52100.
- **b.** The destination port number is the second four hexadecimal digits (000D)16, which means that the destination port number is 13.
- **c.** The third four hexadecimal digits (001C)16 define the length of the whole UDP packet as 28 bytes.
- **d.** The length of the data is the length of the whole packet minus the length of the header, or 28 − 8 = 20 bytes.
- **e.** Since the destination port number is 13 (well-known port), the packet is from the client to the server.
- **f.** The client process is the Daytime(shown in table above)

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

As mentioned previously, UDP provides a *connectionless service.* This means that 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.

Flow Control

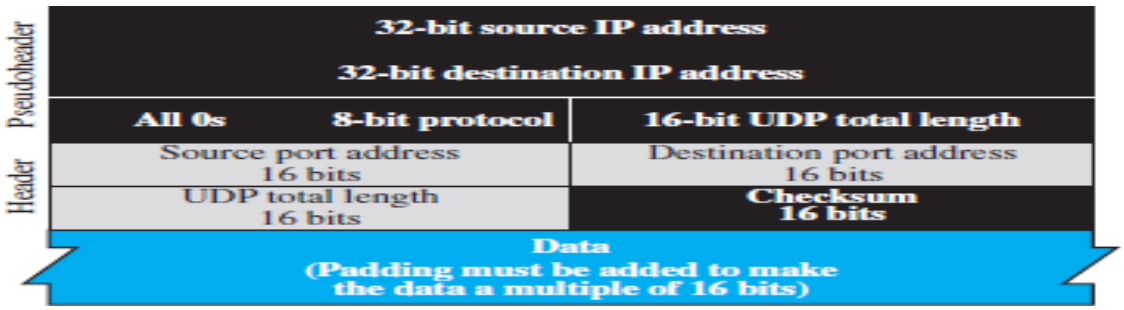
UDP is a very simple protocol. There is no *flow control,* and hence no window mechanism. The receiver may overflow with incoming messages. The lack of flow control means that the process using UDP should provide for this service, if needed.

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 of the IP packet in which the user datagram is to be encapsulated with some fields filled with 0s.



Congestion Control

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

Queuing

We have talked about ports without discussing the actual implementation of them. 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.

UDP Applications

The following shows some typical applications that can benefit more from the services of UDP than from those of TCP.

1. 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.
2. UDP is suitable for a process with internal flow- and error-control mechanisms. For example, the Trivial File TransferProtocol (TFTP) process includes flow and error control. It can easily use UDP.
3. UDP is a suitable transport protocol for multicasting. Multicasting capability is embedded in the UDP software but notin the TCP software.
4. UDP is used for management processes such as SNMP.
5. UDP is used for some route updating protocols such as Routing InformationProtocol (RIP).
6. UDP is normally used for interactive real-time applications that cannot tolerate uneven delay between sections of areceived message

4.TRANSMISSION CONTROL PROTOCOL

*Define slow start in TCP connection **(2)**

* How do fast retransmit mechanism of TCP works? (2)

Transmission Control Protocol (TCP) is a connection-oriented, reliable protocol. TCP explicitly defines connection establishment, data transfer, and connection teardown phases to provide a connection-oriented service. TCP uses a combination of GBN and SR protocols to provide reliability. To achieve this goal, TCP uses checksum (for error detection), retransmission of lost or corrupted packets, cumulative and selective acknowledgments, and timers.

TCP Services

Process-to-Process Communication

As with UDP, TCP provides process-to-process communication using port numbers.

Stream Delivery Service

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.

Sending and Receiving Buffers

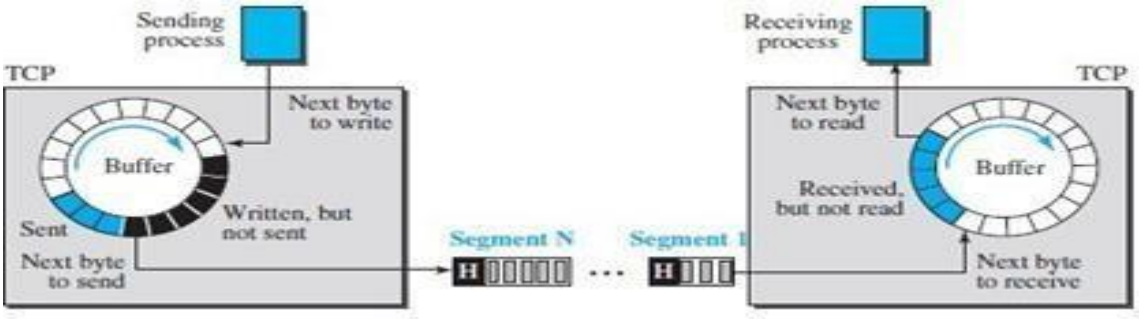
Because the sending and the receiving processes may not necessarily write or read data at the same rate, TCP needs buffers for storage. There are two buffers, the sending buffer and the receiving buffer, one for each direction. The figure shows the movement of the data in one direction. At the sender, the buffer has three types of chambers. The white section contains empty chambers that can be filled by the sending process (producer). The colored area holds bytes that have been sent but not yet acknowledged. The TCP sender keeps these bytes in the buffer until it receives an acknowledgment. The shaded area contains bytes to be sent by the sending TCP. The operation of the buffer at the receiver is simpler. The circular buffer is divided

into two areas (shown as white and colored). The white area contains empty chambers to be filled by bytes received from the network. The colored sections contain received bytes that can be read by the receiving process. When a byte is read by the receiving process, the chamber is recycled and added to the pool of empty chambers.

Segments

Although buffering handles the disparity between the speed of the producing and consuming processes, we need one more step before we can send data. The network layer, as a service provider for TCP, needs to send data in packets, not as a stream of

bytes. At the transport layer, TCP groups a number of bytes together into a packet called a *segment*. TCP adds a header to each segment (for control purposes) and delivers the segment to the network layer for transmission. The segments are encapsulated inan IP datagram and transmitted. This entire operation is transparent to the receiving process.



Note that segments are not necessarily all the same size. In the figure, for simplicity, we show one segment carrying 3 bytes and the other carrying 5 bytes. In reality, segments carry hundreds, if not thousands, of bytes.

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 ownsending and receiving buffer, and segments move in both directions.

Multiplexing and Demultiplexing

Like UDP, TCP performs multiplexing at the sender and demultiplexing at the receiver. However, since TCP is a connection- oriented protocol, a connection needs to be established for each pair of processes.

Connection-Oriented Service

TCP, unlike UDP, is a connection-oriented protocol. 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.

Note that this is a logical connection, not a physical connection. The TCP segment is encapsulated in an IP datagram and can be sent out of order, or lost or corrupted, and then resent. Each may be routed over a different path to reach the destination. There is no physical connection. TCP creates a stream-oriented environment in which it accepts the responsibility of delivering the bytes in order to the other site.

Reliable Service

TCP is a reliable transport protocol. It uses an acknowledgment mechanism to check the safe and sound arrival of data. We will discuss this feature further in the section on error control.

TCP Features

Numbering System

Although the TCP software keeps track of the segments being transmitted or received, there is no field for a segment number value in the segment header. Instead, there are two fields, called the *sequence number* and the *acknowledgment number*. These two fields refer to a byte number and not a segment number.

Byte Number

TCP numbers all data bytes (octets) that are transmitted in a connection. Numbering is independent in each direction. WhenTCP receives bytes of data from a process, TCP stores them in the sending buffer and numbers them. The numbering does not necessarily start from 0. Instead, TCP chooses an arbitrary number between 0 and 2_{32}-1 for the number of the first byte.

Sequence Number

After the bytes have been numbered, TCP assigns a sequence number to each segment that is being sent. The sequence number, in each direction, is defined as follows:

1. The sequence number of the first segment is the ISN (initial sequence number), which is a random number.
2. The sequence number of any other segment is the sequence number of the previous segment plus the number of bytes (real or imaginary) carried by the previous segment.

When a segment carries a combination of data and control information (piggybacking), it uses a sequence number. If a segment does not carry user data, it does not logically define a sequence number. The field is there, but the value is not valid. However, some segments, when carrying only control information, need a sequence number to allow an acknowledgment from the receiver. These segments are used for connection establishment, termination, or abortion. Each of these segments consume onesequence number as though it carries one byte, but there are no actual data. We will elaborate on this issue when we discuss connections.

Acknowledgment Number

As we discussed previously, communication in TCP is full duplex; when a connection is established, both parties can send and receive data at the same time. Each party numbers the bytes, usually with a different starting byte number. The sequence number in each direction shows the number of the first byte carried by the segment.

Each party also uses an acknowledgment number to confirm the bytes it has received. However, the acknowledgment number defines the number of the next byte that the bytes of data being transferred in each connection are numbered by TCP. The numbering starts with an arbitrarily generated number.

Example

Suppose a TCP connection is transferring a file of 5000 bytes. The first byte is numbered 10001. What are the sequence numbers for each segment if data are sent in five segments, each carrying 1000 bytes?

Solution

The following shows the sequence number for each segment: Segment 1 –Sequence Number: 10001 **Range:** 10001 to 11000

Segment 2 -- Sequence Number: 11001 **Range:** 11001 to 12000

Segment 3-- Sequence Number: 12001 **Range:** 12001 to 13000
Segment 4 -- Sequence Number:13001 **Range:** 13001 to 14000
Segment 5 --Sequence Number: 14001 **Range:** 14001 to 15000

The value in the sequence number field of a segment defines the number assigned to the first data byte contained in that segment.

Acknowledgment Number

TCP is full duplex; when a connection is established, both parties can send and receive

data at the same time. Each party numbers the bytes, usually with a different starting byte number. The sequence number in each direction shows the number of the first byte carried by the segment. Each party also uses an acknowledgment number to confirm the bytes it hasreceived.
However, the acknowledgment number defines the number of the next byte that the party expects to receive. In addition, the acknowledgment number is cumulative, which means that the party takes the number of the last byte that it has received, safe and sound, adds 1 to it, and announces this sum as the acknowledgment number. The term *cumulative* here means that if a partyuses 5643 as an acknowledgment number, it has received all bytes from the beginning up to 5642. Note that this does not mean that the party has received 5642 bytes, because the first byte number does not have to be 0.

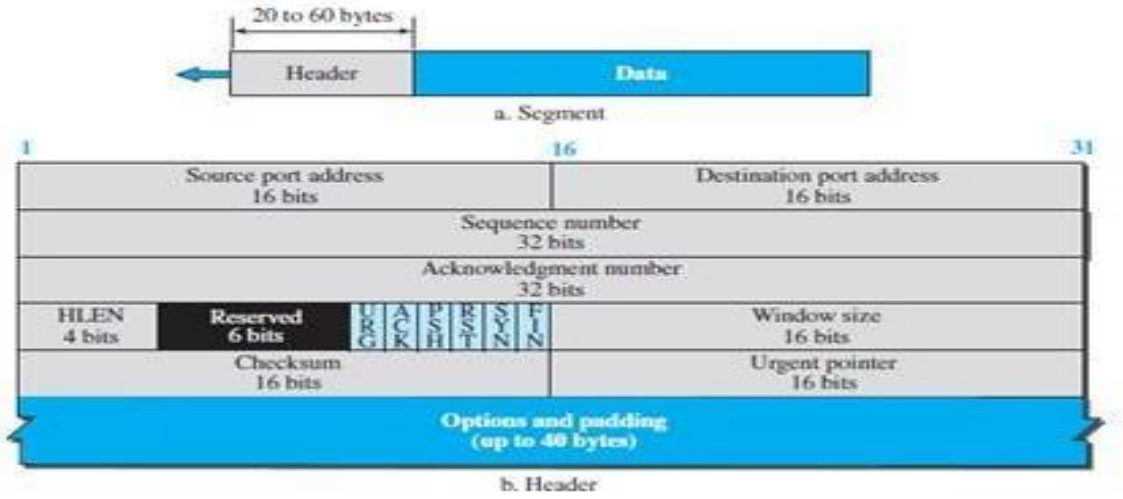
The value of the acknowledgment field in a segment defines the number of the next byte a party expects to receive. Theacknowledgment number is cumulative.

Segment

A packet in TCP is called a *segment.*

Format

The format of a segment is shown in Figure below. The segment consists of a header of 20 to 60 bytes, followed by data fromthe application program. The header is 20 bytes if there are no options and up to 60 bytes if it contains options. We will discuss some of the header fields in this section. The meaning and purpose of these will become clearer as we proceed through the section.



Source port address. This is a 16-bit field that defines the port number of the application program in the host that is sending thesegment.

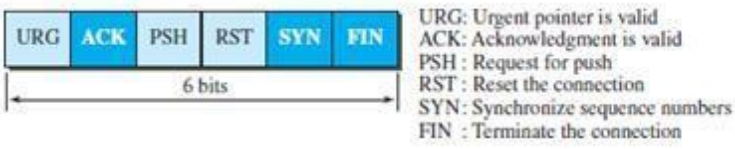
Destination port address. This is a 16-bit field that defines the port number of the application program in the host that isreceiving the segment.

Sequence number. This 32-bit field defines the number assigned to the first byte of data contained in this segment. As we said before, TCP is a stream transport protocol. To ensure connectivity, each byte to be transmitted is numbered. The sequence number tells the destination which byte in this sequence is the first byte in the segment. During connection establishment (discussed later) each party uses a random number generator to create an **initial sequence number** (ISN), which is usually different in each direction.

Acknowledgment number. This 32-bit field defines the byte number that the receiver of the segment is expecting to receivefrom the other party. If the receiver of the segment has successfully received byte number *x* from the other party, it returns *x*
=1 as the acknowledgment number. Acknowledgment and data can be piggybacked together.

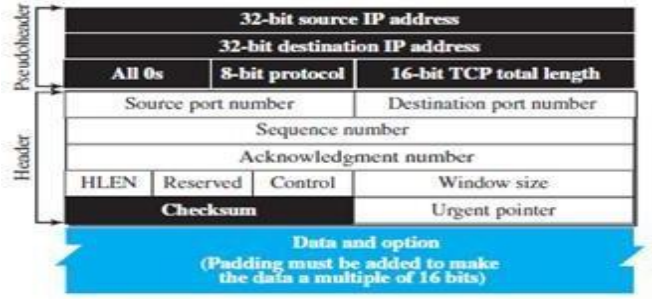
Header length. This 4-bit field indicates the number of 4-byte words in the TCP header. The length of the header can be between 20 and 60 bytes. Therefore, the value of this field is always between 5 (5 * 4 = 20) and 15 (15 *4 = 60).

Control. This field defines 6 different control bits or flags, as shown in Figure. One or more of these bits can be set at a time.
These bits enable flow control, connection establishment and termination, connection abortion, and the mode of datatransfer in TCP. A brief description of each bit is shown in the figure.



Window Size-This field defines the window size of the sending TCP in bytes. Note that the length of this field is 16 bits, which means that the maximum size of the window is 65,535 bytes. This value is normally referred to as the receiving window (*rwnd*) and is determined by the receiver. The sender must obey the dictation of the receiver in this case.

Checksum. This 16-bit field contains the checksum. The calculation of the checksum for TCP follows the same procedure as theone described for UDP. However, the use of the checksum in the UDP datagram is optional, whereas the use of the checksumfor TCP is mandatory. The same pseudoheader, serving the same purpose, is added to the segment.



Urgent pointer. This 16-bit field, which is valid only if the urgent flag is set, is used when the segment contains urgent data. It defines a value that must be added to the sequence number to obtain the number of the last urgent byte in the data section
of the segment.
Options. There can be up to 40 bytes of optional information in the TCP header.

5. TCP CONNECTION MANAGEMENT

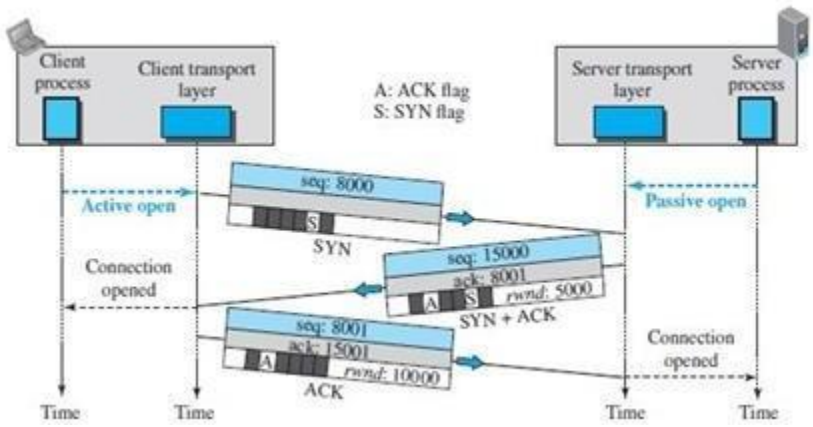
TCP is connection-oriented protocol establishes a logical path between the source and destination. All of the segments belonging to a message are then sent over this logical path. Using a single logical pathway for the entire message facilitates the acknowledgment process as well as retransmission of damaged or lost frames.
In TCP, connection-oriented transmission requires three phases: connection establishment, data transfer, and connectiontermination.

Connection Establishment

TCP transmits data in full-duplex mode. When two TCPs in two machines are connected, they are able to send segments to eachother simultaneously.

Three-Way Handshaking

The connection establishment in TCP is called ***three-way handshaking.*** The process starts with the server. The server program tells its TCP that it is ready to accept a connection. This request is called a *passive open*. Although the server TCP is ready to accept a connection from any machine in the world, it cannot make the connection itself. The client program issues a request for an *active open*. A client that wishes to connect to an open server tells its TCP to connect to a particular server.



The three steps in this phase are as follows.

1. The client sends the first segment, a SYN segment, in which only the SYN flag is set. This segment is for synchronization of sequence numbers. The client in our example chooses a random number as the first sequence number and sends this number to the server. This sequence number is called the *initial sequence number (ISN).* Note that this segment does not contain an acknowledgment number. It does not define the window size either; a window size definition makes sense only when a segment includes an acknowledgment. Note that the SYN segment is a control segment and carries no data. However, it consumes one sequence number because it needs to be acknowledged. We can say that the SYN segment carries one imaginary byte.

2. The server sends the second segment, a SYN + ACK segment with two flag bits set as: SYN and ACK. This segment has a dual purpose. First, it is a SYN segment for communication in the other direction. The server uses this segment to initialize a sequence number for numbering the bytes sent from the server to the client. The server also acknowledges the receipt of the SYN segment from the client by setting the ACK flag and displaying the next sequence number it expects to receive from the client. Because the segment contains an acknowledgment, it also needs to define the receive window size, *rwnd* (to be used bythe client), as we will see in the flow control section. Since this segment is playing the role of a SYN segment, it needs to be acknowledged. It, therefore, consumes one sequence number.

3. The client sends the third segment. This is just an ACK segment. It acknowledges the receipt of the second segment with the ACK flag and acknowledgment number field. Note that the ACK segment does not consume any sequence numbers if it does not carry data, but some implementations allow this third segment in the connection phase to carry the first chunk of data from the client.In this case, the segment consumes as many sequence numbers as the number of data bytes.

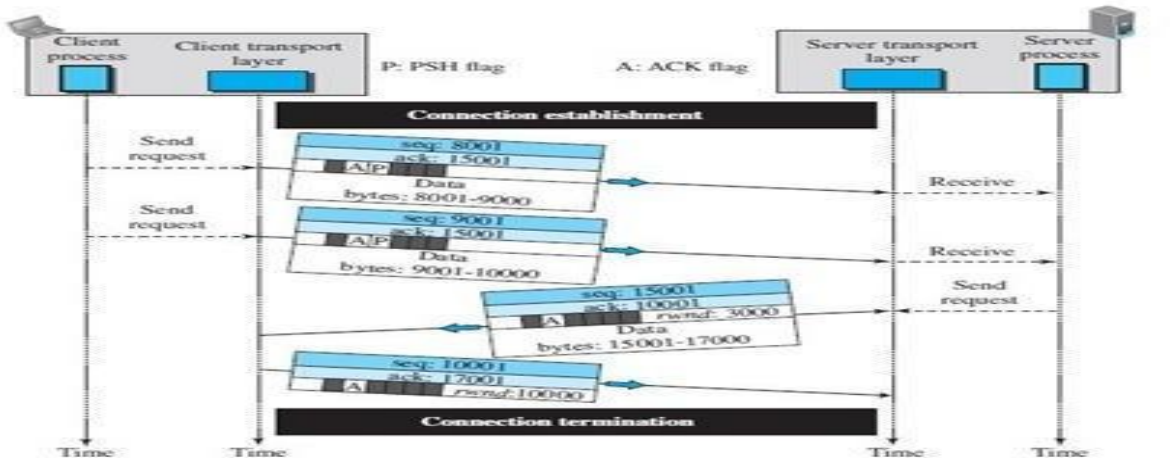
SYN Flooding Attack

The connection establishment procedure in TCP is susceptible to a serious security problem called ***SYN flooding attack.*** This happens when one or more malicious attackers send a large number of SYN segments to a server pretending that each of them is coming from a different client by faking the source IP addresses in the datagrams.

Data Transfer

After connection is established, bidirectional data transfer can take place. The client

and server can send data and acknowledgments in both directions. In this example, after a connection is established, the client sends 2,000 bytes of data intwo segments. The server then sends 2,000 bytes in one segment. The client sends one more segment. The first three segments carry both data and acknowledgment, but the last segment carries only an acknowledgment because there is no more data to be sent. Note the values of the sequence and acknowledgment numbers. The data segments sent by the client have the PSH (push) flag set so that the server TCP knows to deliver data to the server process as soon as they are received. The segment from the server, on the other hand,does not set the push flag. Most TCP implementations have the option to set or not to set this flag.



The application program at the sender can request a *push* operation. This means that the sending TCP must not wait for the window to be filled. It must create a segment and send it immediately. The sending TCP must also set the push bit (PSH) to let the receiving TCP know that the segment includes data that must be delivered to the receiving application program as soon as possible and not to wait for more data to come.

Connection Termination

Either of the two parties involved in exchanging data (client or server) can close the connection, although it is usually initiated by the client. Most implementations today allow two options for connection termination: three-way handshaking and four-way handshaking with a half-close option.

Three-Way Handshaking

Most implementations today allow *three-way handshaking* for connection termination, as shown in Figure

1. In this situation, the client TCP, after receiving a close command from the client process, sends the first segment, a FIN segment in which the FIN flag is set. Note that a FIN segment can include the last chunk of data sent by the client or it can be just a control segment as shown in the figure. If it is only a control segment, it consumes only one sequence number because itneeds to be acknowledged.

2. The server TCP, after receiving the FIN segment, informs its process of the situation and sends the second segment, a FIN

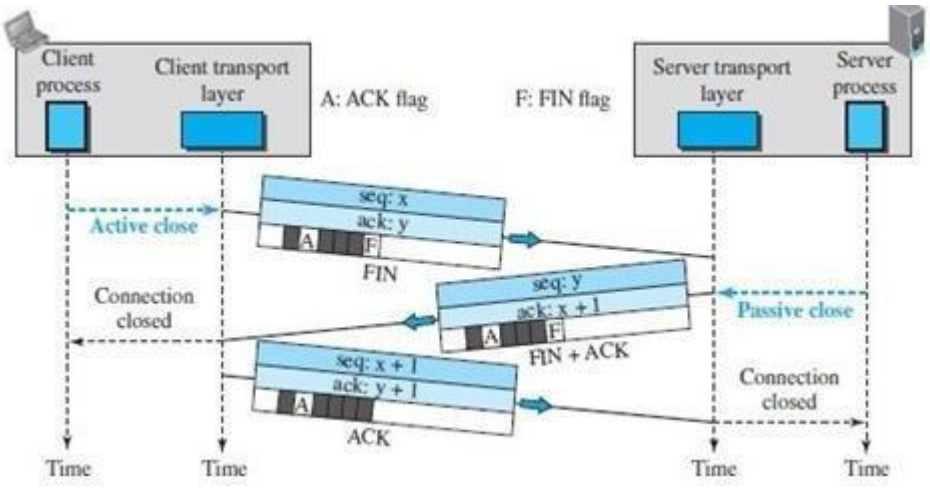
+ ACK segment, to confirm the receipt of the FIN segment from the client and at the

same time to announce the closing of the connection in the other direction. This segment can also contain the last chunk of data from the server. If it does not carry data, itconsumes only one sequence number because it needs to be acknowledged.

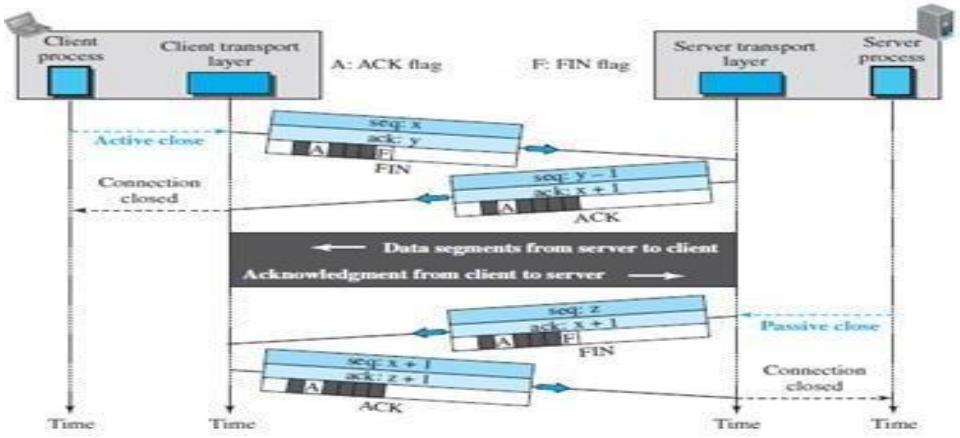
3. The client TCP sends the last segment, an ACK segment, to confirm the receipt of the FIN segment from the TCP server. Thissegment contains the acknowledgment number, which is one plus the sequence number received in the FIN segment from the server. This segment cannot carry data and consumes no sequence numbers.

Half- Close

In TCP, one end can stop sending data while still receiving data. This is called a ***halfclose.*** Either the server or the client can issue a half-close request. It can occur when the server needs all the data before processing can begin. A good example is sorting. When the client sends data to the server to be sorted, the server needs to receive all the data before sorting can start. Thismeans the client, after sending all data, can close the connection in the client-to-server direction. However, the server-to-client direction must remain open to return the sorted data. The server, after receiving the data, still needs time for sorting; itsoutbound direction must remain open

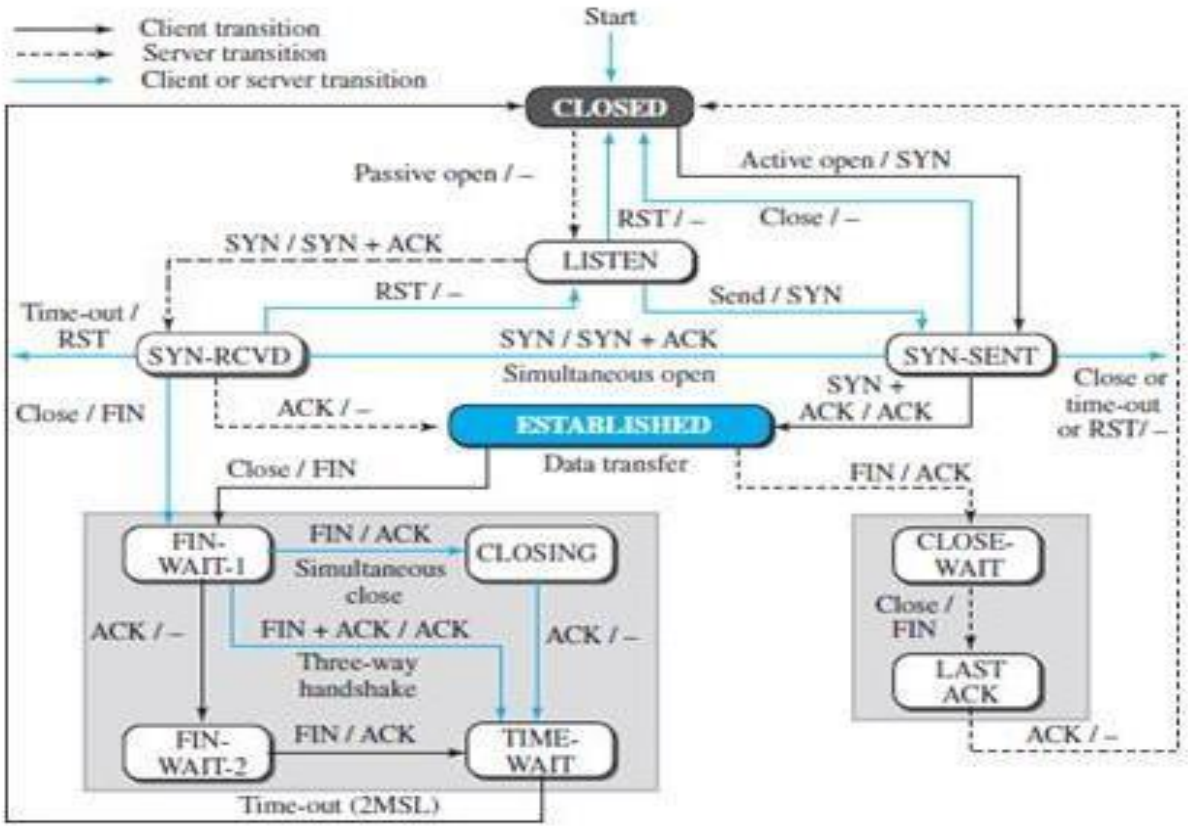


The data transfer from the client to the server stops. The client half-closes the connection by sending a FIN segment. The server accepts the half-close by sending the ACK segment. The server, however, can still send data. When the server has sent all of the processed data, it sends a FIN segment, which is acknowledged by an ACK from the client. After half-closing the connection, data can travel from the server to the client and acknowledgments can travel from the client to the server. The client cannot send any more data to the server. Half close is shown below.



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) as shown in Figure. The rounded-corner rectangles represent the states. The transition from one state to another is shown using directed lines. Each line has two strings separated by a slash. The first string is the input, what TCP receives. The second is the output, what TCP sends. The dotted black lines in the figure represent the transition that a server normally goes through; the solid black lines show the transitions that a client normally goes through. However, in some situations, a server transitions through a solid line or a client transitions through a dotted line. The colored lines show special situations. Note that the rounded-corner rectangle marked *ESTABLISHED* is in fact two sets of states, a set for the client and another for the server, that are used for flow and error control.The states of TCP is shown in table below.



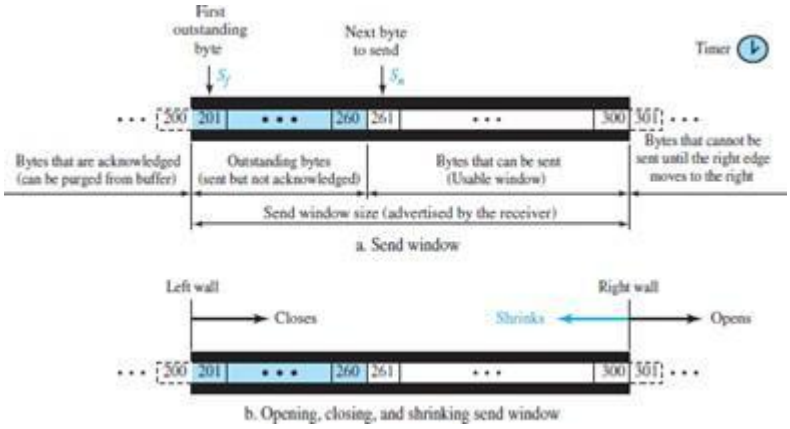
State	Description
CLOSED	No connection exists
LISTEN	Passive open received; waiting for SYN
SYN-SENT	SYN sent; waiting for ACK
SYN-RCVD	SYN + ACK sent; waiting for ACK
ESTABLISHED	Connection established; data transfer in progress
FIN-WAIT-1	First FIN sent; waiting for ACK
FIN-WAIT-2	ACK to first FIN 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

Windows in TCP

TCP uses two windows (send window and receive window) for each direction of data transfer, which means fourwindows for a bidirectional communication.

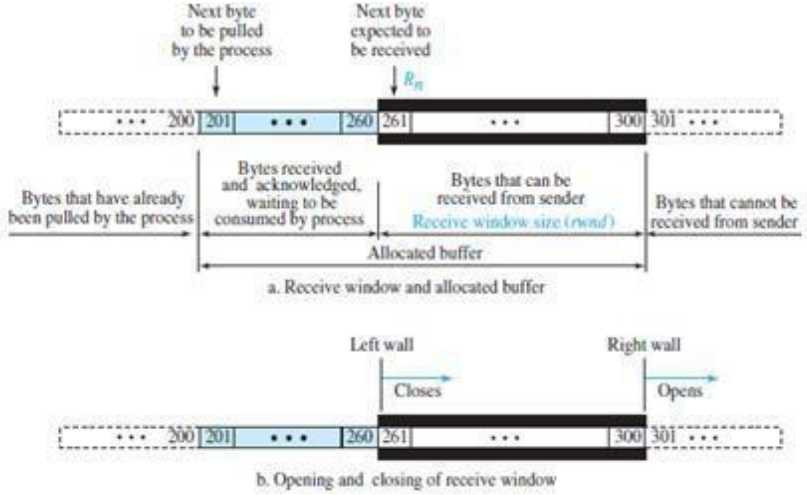
Send Window

The figure shows how a send window *opens*, *closes*, or *shrinks*.



Receive Window

The figure also shows how the receive window opens and closes.



Silly Window Syndrome

A serious problem can arise in the sliding window operation when either the sending application program creates data slowly or the receiving application program consumes data slowly, or both. Any of these situations results in the sending of data in very small segments, which reduces the efficiency of the operation. For example, if TCP sends segments containing only 1 byte of data, it means that a 41-byte datagram (20 bytes of TCP header and 20 bytes of IP header) transfers only 1 byte of user data. Here the overhead is 41/1, which indicates that we are using the capacity of the network very inefficiently. The inefficiency is even worse after accounting for the data- link layer and physical-layer overhead. This problem is called the ***silly window syndrome.***

Syndrome Created by the Sender

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. How long should the sending TCP wait? If it waits too long, it may delay the process. If it does not wait long enough, it may end up sending small segments. Nagle found an elegant solution. **Nagle's algorithm** issimple:

1. The sending TCP sends the first piece of data it receives from the sending application program even if it is only 1 byte.
2. After sending the first segment, the sending TCP accumulates data in the output buffer and waits until either the receiving TCP sends an acknowledgment or until enough data have accumulated to fill a maximum-size segment. At this time, the sending TCP can send the segment.
3. Step 2 is repeated for the rest of the transmission. Segment 3 is sent immediately if an acknowledgment is received for segment 2, or if enough data have accumulated to fill a maximum-size segment.

Syndrome Created by the Receiver

The receiving TCP may create a silly window syndrome if it is serving an application program that consumes data slowly, for example, 1 byte at a time. Suppose that the sending application program creates data in blocks of 1 kilobyte, but the receiving application program consumes data 1 byte at a time. Also suppose that the input buffer of the receiving TCP is 4 kilobytes. The sender sends the first 4 kilobytes of data. The receiver stores it in its buffer. Now its buffer is full. It advertises a window
size of zero, which means the sender should stop sending data. The receiving application reads the first byte of data from the input buffer of the receiving TCP. Now there is 1 byte of space in the incoming buffer. The receiving TCP announces a window size of 1 byte, which means that the sending TCP, which is eagerly waiting to send data, takes this advertisement as good news and sends a segment carrying only 1 byte of data. The procedure will continue. One byte of data is consumed and a segment carrying 1 byte of data is sent. Again we have an efficiency problem and the silly window syndrome.

Two solutions have been proposed to prevent the silly window syndrome created by an application program that consumes data more slowly than they arrive. **Clark's solution** is to send an acknowledgment as soon as the data arrive, but to announce a window size of zero until either there is enough space to accommodate a segment of

maximum size or until at least half of the receive buffer is empty. The second solution is to delay sending the acknowledgment. This means that when a segment arrives, it is not acknowledged immediately. The receiver waits until there is a decent amount of space in its incoming buffer before acknowledging the arrived segments. The delayed acknowledgment prevents the sending TCP from sliding its window. After the sending TCP has sent the data in the window, it stops. This kills the syndrome. Delayed acknowledgment also has another advantage: it reduces traffic. The receiver does not have to acknowledge each segment. However, there also is a disadvantage in that the delayed acknowledgment may result in the sender unnecessarily retransmitting the unacknowledged segments.

Acknowledgment Type

Cumulative Acknowledgment (ACK) TCP was originally designed to acknowledge receipt of segments cumulatively. The receiver advertises the next byte it expects to receive, ignoring all segments received and stored out of order. This is sometimes referred to as *positive cumulative acknowledgment,* or ACK.

Selective Acknowledgment (SACK) More and more implementations are adding another type of acknowledgment called*selective acknowledgment,* or SACK. A SACK does not replace an ACK, but reports additional information to the sender. A SACK reports a block of bytes that is out of order, and also a block of bytes that is duplicated, i.e., received more than once.

Retransmission

The heart of the error control mechanism is the retransmission of segments. When a segment is sent, it is stored in a queue until it is acknowledged. When the retransmission timer expires or when the sender receives three duplicate ACKs for the first segment in the queue, that segment is retransmitted.

Retransmission after RTO

The sending TCP maintains one **retransmission time-out (RTO)** for each connection. When the timer matures, i.e. times out, TCP resends the segment in the front of the queue (the segment with the smallest sequence number) and restarts the timer.

Note that again we assume *Sf□Sn*. We will see later that the value of RTO is dynamic in TCP and is updated based on the **round-trip time (RTT)** of segments. RTT is the time needed for a segment to reach a destination and for an acknowledgment tobe received.

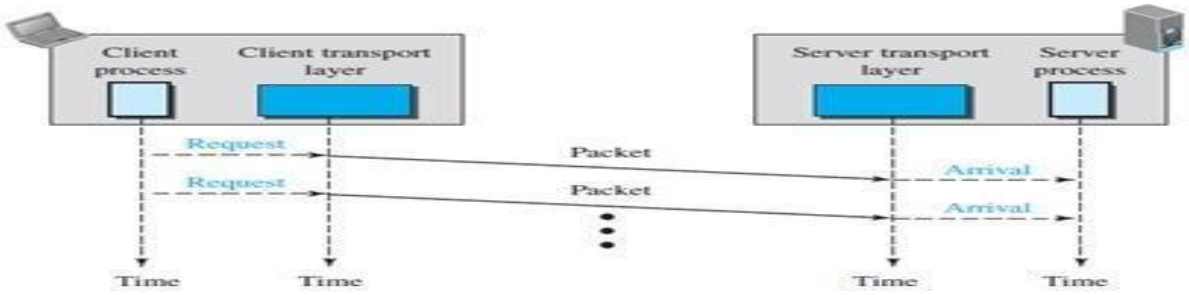
*****List out the transport layer protocol service primitives and their meaning**. (2)**

***** Explain Stop and Wait protocol mechanism. **(2)**

6.FLOW CONTROL

Simple Protocol

Our first protocol is a simple connectionless protocol with neither flow nor error control. We assume that the receiver can immediately handle any packet it receives. In other words, the receiver can never be overwhelmed with incoming packets. The transport layer at the sender gets a message from its application layer, makes a packet out of it, and sends the packet. The transport layer at the receiver receives a packet from its network layer, extracts the message from the packet, and delivers the message to its application layer. The transport layers of the sender and receiver provide transmission services for their application layers.



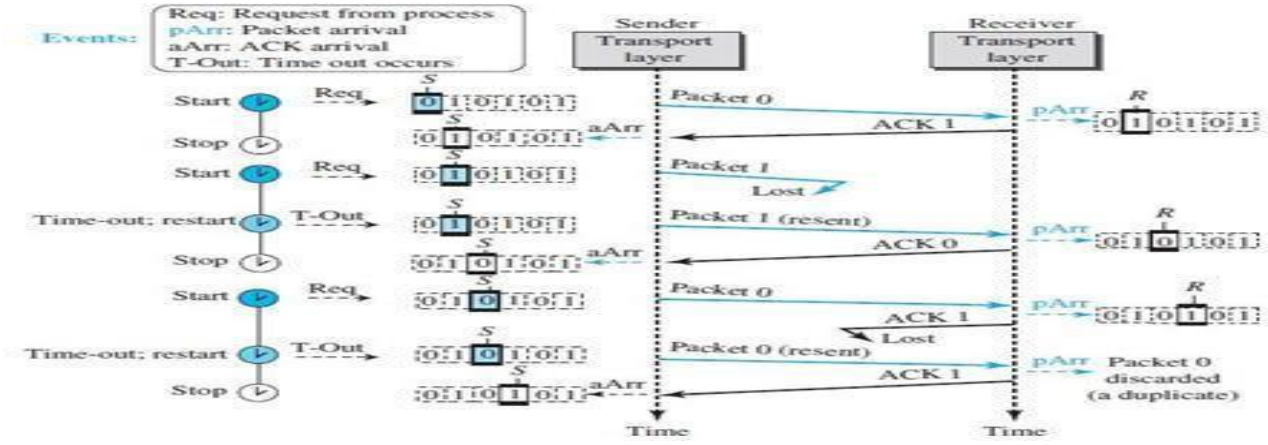
Stop-and-Wait Protocol

Our second protocol is a connection-oriented protocol called the **Stop-and-Wait protocol,** which uses both flow and error control. Both the sender and the receiver use a sliding window of size 1. The sender sends one packet at a time and waits for an acknowledgment before sending the next one. To detect corrupted packets, we need to add a checksum to each data packet.

When a packet arrives at the receiver site, it is checked. If its checksum is incorrect, the packet is corrupted and silently discarded. Every time the sender sends a packet, it starts a timer. If an acknowledgment arrives before the timer expires, the timer is stopped and the sender sends the next packet (if it has one to send). If the timer expires, the sender resends the previous packet, assuming that the packet was either lost or corrupted. This means that the sender needs to keep a copy of the packet until its acknowledgment arrives. Note that only one packet and one acknowledgment can be in the channels at any time.

Sequence Numbers

To prevent duplicate packets, the protocol uses sequence numbers and acknowledgment numbers. A field is added to the packet header to hold the sequence number of that packet. One important consideration is the range of the sequence numbers. Since we want to minimize the packet size, we look for the smallest range that provides unambiguous communication.



Go-Back-*N* Protocol (GBN)

The key to Go-back-*N* is that we can send several packets before receiving

acknowledgments, but the receiver can only buffer one packet. We keep a copy of the sent packets until the acknowledgments arrive.

Sequence Numbers

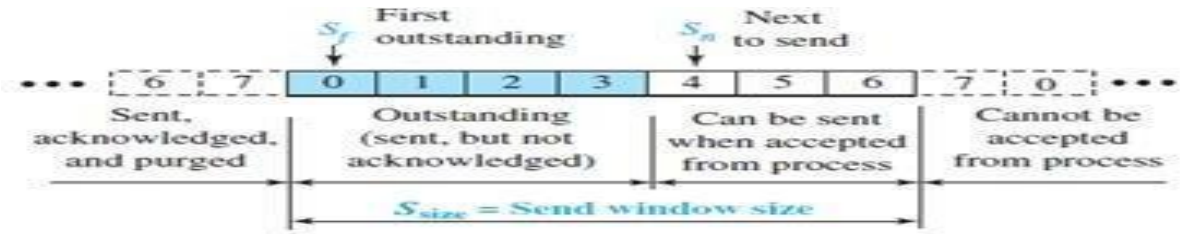
As we mentioned before, the sequence numbers are modulo 2*m*, where *m* is the size of the sequence number field in bits.

Acknowledgment Numbers

An acknowledgment number in this protocol is cumulative and defines the sequence number of the next packet expected. For example, if the acknowledgment number (ackNo) is 7, it means all packets with sequence number up to 6 have arrived, safe and sound, and the receiver is expecting the packet with sequence number 7.

Send Window

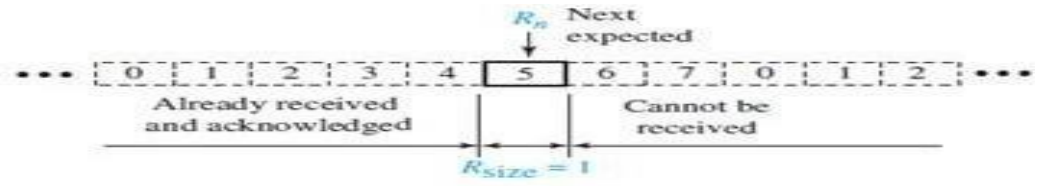
The send window is an imaginary box covering the sequence numbers of the data packets that can be in transit or can be sent. In each window position, some of these sequence numbers define the packets that have been sent; others define those that can be sent. Figure below shows a sliding window of size 7 (*m* 3) for the Go-Back-*N* protocol.



The send window at any time divides the possible sequence numbers into four regions. The first region, left of the window, defines the sequence numbers belonging to packets that are already acknowledged. The sender does not worry about these packets and keeps no copies of them. The second region, colored, defines the range of sequence numbers belonging to the packets that have been sent, but have an unknown status. The sender needs to wait to find out if these packets have been received or were lost. We call these *outstanding* packets. The third range, white in the figure, defines the range of sequence numbers for packets that can be sent; however, the corresponding data have not yet been received from the application layer. Finally, the fourth region, right of the window, defines sequence numbers that cannot be used until the window slides. The window itself is an abstraction; three variables define its size and location at any time. We call these variables *Sf* (send window,the first outstanding packet), *Sn* (send window, the next packet to be sent), and *S*size (send window, size). The variable *Sf* defines the sequence number of the first (oldest) outstanding packet. The variable *Sn* holds the sequence number that will be assigned to the next packet to be sent. Finally, the variable *S*size defines the size of the window, which is fixed in our protocol.

Receive Window

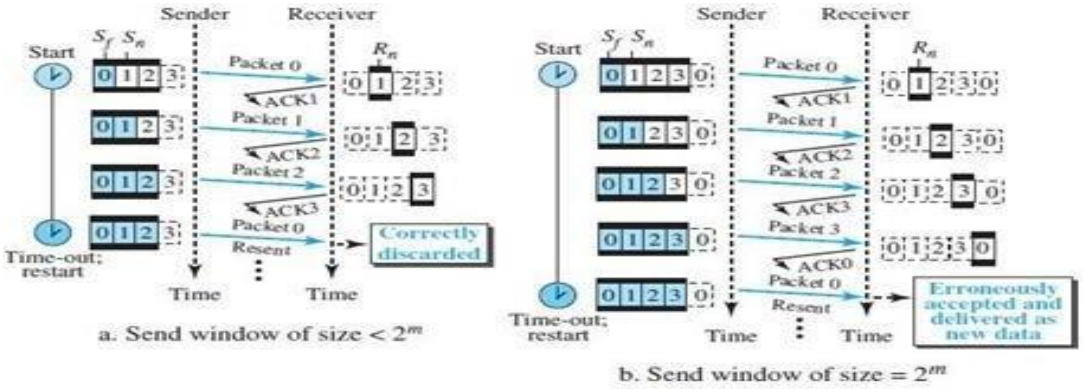
The receive window makes sure that the correct data packets are received and that the correct acknowledgments are sent. In Go- Back-*N*, the size of the receive window is always 1. The receiver is always looking for the arrival of a specific packet. Any packet arriving out of order is discarded and needs to be resent.



Send Window Size

We can now show why the size of the send window must be less than 2*m*. As an example,we choose *m* 2, which means the size of the window can be 2*m* - 1, or 3.

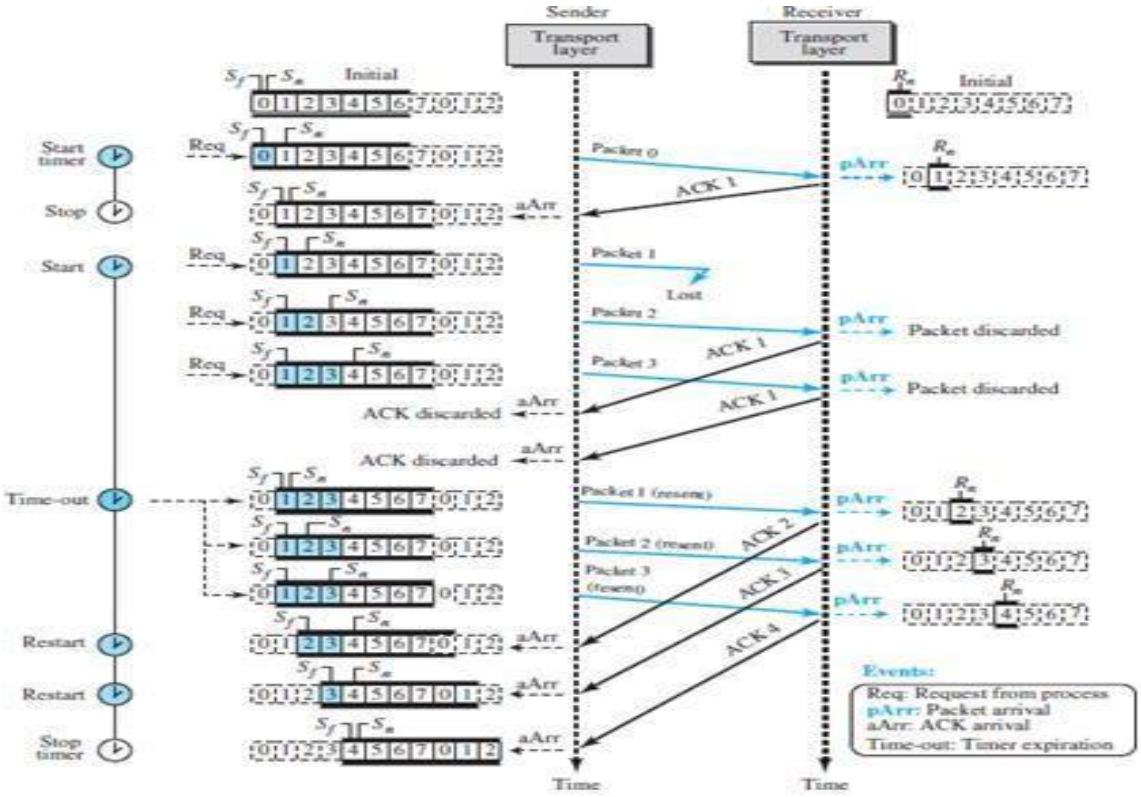
Figure below compares a window size of 3 against a window size of 4. If the size of the window is 3 (less than 2*m*) and all three acknowledgments are lost, the only timer expires and all three packets are resent. The receiver is now expecting packet 3, not packet 0, so the duplicate packet is correctly discarded. On the other hand, if the size of the window is 4 (equal to 22) and all acknowledgments are lost, the sender will send a duplicate of packet 0. However,this time the window of the receiver expects to receive packet 0 (in the next cycle),so it accepts packet 0, not as a duplicate, but as thefirst packet in the next cycle. This is an error. This shows that the size of the send window must be less than 2*m*.



In the Go-Back-*N* protocol, the size of the send window must be less than 2*m*;the size of the receive window is always 1.

Go-Back-N versus Stop-and-Wait

The reader may find that there is a similarity between the Go-Back-*N* protocol and the Stop and Wait protocol. The Stop-and- Wait protocol is actually a Go-Back-*N* protocol in which there are only two sequence numbers and the send window size is 1. In other words, *m* = 1 and 2*m* - 1 =1. In Go-Back-*N*, we said that the arithmetic is modulo 2*m*; in Stop-and-Wait it is modulo 2, which is the same as 2*m* when *m* = 1.

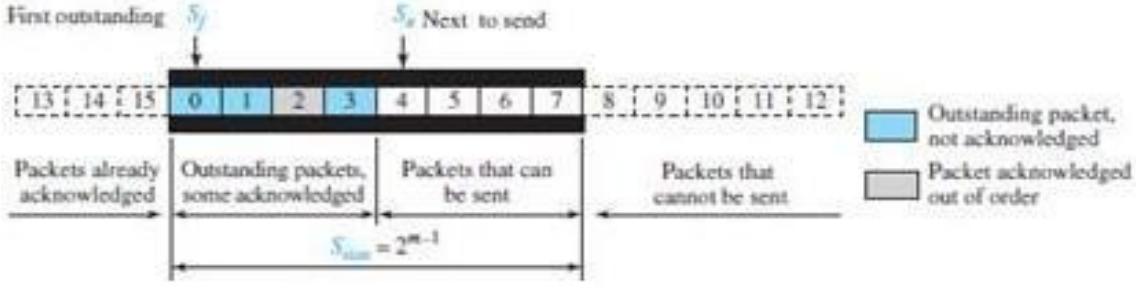


Selective-Repeat Protocol

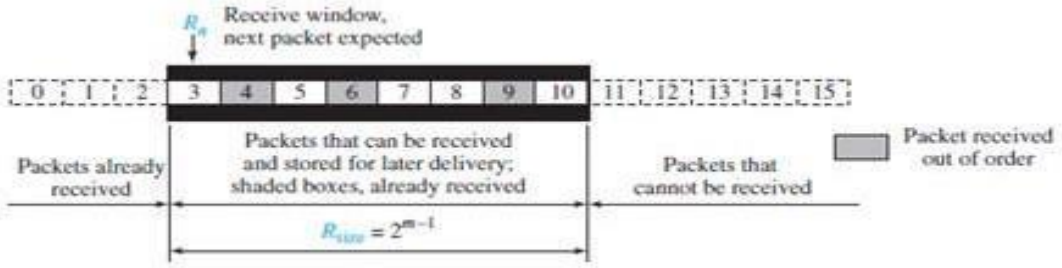
The **Selective-Repeat (SR) protocol,** which, as the name implies, resends only selective packets.

Windows

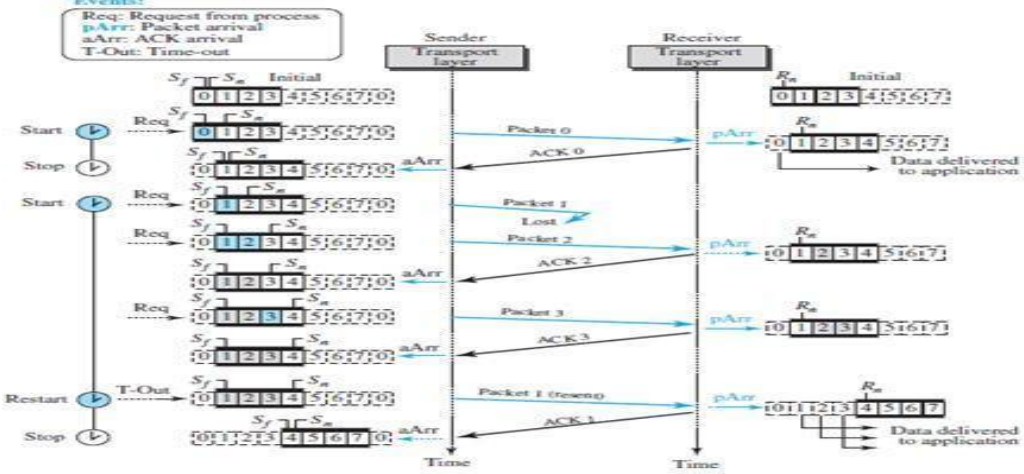
The send window maximum size can be 2*m*-1. For example, if *m* =4, the sequence numbers go from 0 to 15, but the maximum size of the window is just 8. The receive window in Selective-Repeat is totally different from the one in Go- Back-*N*. The size of the receive window is the same as the size of the send window (maximum 2*m*-1). The Selective-Repeat protocol allows as many packets as the size of the receive window to arrive out of order and be kept until there is a set of consecutive packets to be delivered to the application layer. Because the sizes of the send window and receive window are the same, all the packets in the send packet can arrive out of order and be stored until they can be delivered. The send window in selective repeat is given below



The receive window is shown below:



The flow diagram of selective repeat is given below

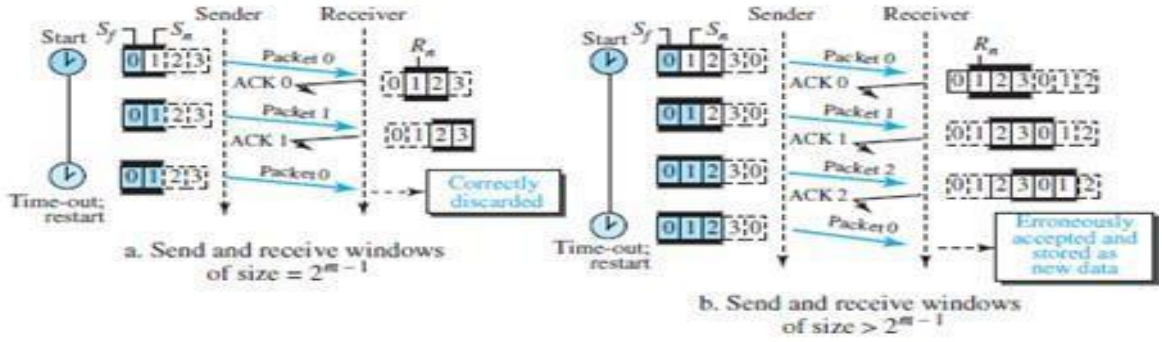


At the sender, packet 0 is transmitted and acknowledged. Packet 1 is lost. Packets 2 and 3 arrive out of order and are acknowledged. When the timer times out, packet 1 (the only unacknowledged packet) is resent and is acknowledged. The send window then slides. At the receiver site we need to distinguish between the acceptance of a packet and its delivery to the application layer. At the second arrival, packet 2 arrives and is stored and marked (shaded slot), but it cannot be delivered because packet 1 is missing. At the next arrival, packet 3 arrives and is marked and stored, but still none of the packets can be delivered. Only at the last arrival, when finally a copy of packet 1 arrives, can packets 1, 2, and 3 be delivered to the application layer. There are two conditions for the delivery of packets to the application layer: First, a set of consecutive packets must have arrived. Second, the set starts from the beginning of the window. After the first arrival, there was only one packet and it started from the beginning of the window. After the last arrival, there are three packets and the first one starts from the beginning of the window. The key is that a reliable transport layer promises to deliver packets in order.

Window Sizes

We can now show why the size of the sender and receiver windows can be at most one-half of $2m$. For an example, we choose $m=2$, which means the size of the window is $2m/2$ or $2(m\text{-}1) = 2$.

If the size of the window is 2 and all acknowledgments are lost, the timer for packet 0 expires and packet 0 is resent. However, the window of the receiver is now expecting packet 2, not packet 0, so this duplicate packet is correctly discarded (the sequence number 0 is not in the window). When the size of the window is 3 and all acknowledgments are lost, the sender sends a duplicate of packet 0. However, this time, the window of the receiver expects to receive packet 0 (0 is part of the window), so it accepts packet 0, not as a duplicate, but as a packet in the next cycle. This is clearly an error.



Example :

Assume a sender sends 6 packets: packets 0, 1, 2, 3, 4, and 5. The sender receives an ACK with ack No =3. What is the interpretation if thesystem is using GBN or SR?

Solution

If the system is using GBN, it means that packets 0, 1, and 2 have been received uncorrupted and the receiver is expecting packet 3. If the system is using SR, it means that packet 3 has been received uncorrupted; the ACK does not say anything about other packets.

7.CONGESTION CONTROL

The idea of TCP congestion control is for each source to determine how much capacity is available in the network, so it knows how many packets it can safely have in transit. Once a given source has this many packets in their transit, it uses the arrival of an ACK as a signal that one of it s packets has the left the network, & that it is therefore safe to insert a new packet into thenetwork without adding to the level of congestion. By using ACKs to pace the transmission of packets, TCP is said to self- clocking.

ADDITIVE INCREASE/MULTIPLICATIVE DECREASE

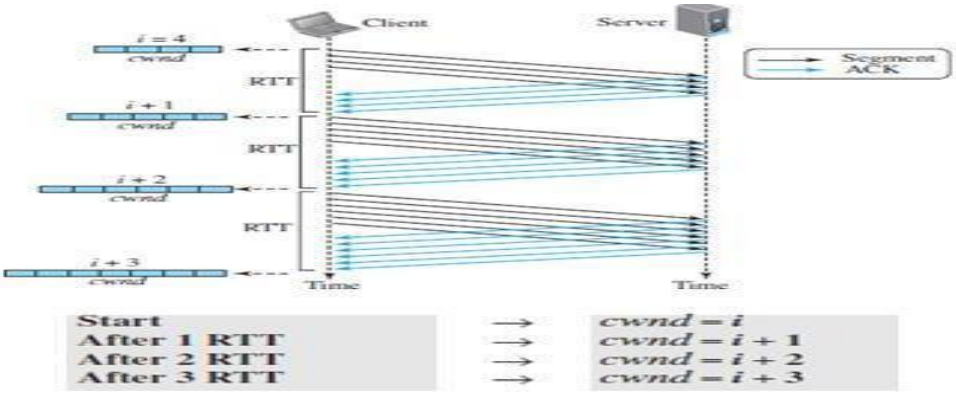
TCP maintains a new state variable for each connection, called Congestion Window, which is used by the source to limit how much data it is allowed to have in transit at a given time .The congestion window is congestion controls counterpart to flow controls advertised window. TCP is modified such that the maximum number of bytes of

unacknowledged data allowed is now the minimum of the congestion window and the advertised window .TCP‟s effective window is revised as follows:

Max Window =MIN (Congestion Window, Received window(Advertised Window))

Effective Window =Max Window – (Last Byte Sent- Last Byte Acked)

The problem, of course , is TCP comes to learn an appropriate value for Congestion Window . Unlike the Advertised Window, which is sent by the receiving side of the connection, there is no one to send a suitable Congestion Window based on the level of congestion it perceives to exist in the network. This involves decreasing the congestion goes up & increasing the congestion window when the level of congestion goes down. Taken together,, the mechanism is commonly called additive increase/multiplicative decrease (AIMD).

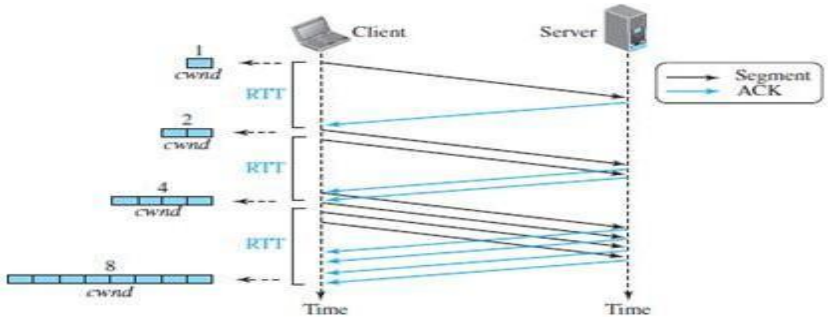


SLOW START: EXPONENTIAL INCREASE

The **slow-start algorithm** is based on the idea that the size of the congestion window (*cwnd*) starts with one maximum segment size (MSS), but it increases one MSS each time an acknowledgment arrives. The MSS is a value negotiated during the connection establishment, using an option of the same name. The name of this algorithm is misleading; the algorithm starts slowly, but grows exponentially. The sender starts with *cwnd* = 1. This means that the sender can send only one segment.

After the first ACK arrives, the acknowledged segment is purged from the window, which means there is now one empty segment slot in the window. The size of the congestion window is also increased by 1 because the arrival of the acknowledgment is a good sign that there is no congestion in the network. The size of the window is now

2. After sending two segments and receiving two individual acknowledgments for them, the size of the congestion window now becomes 4, and soon.



The size of the congestion window in this algorithm is a function of the number of ACKs arrived and can be determined asfollows.

If an ACK arrives, *cwnd* = *cwnd* + **1.**

If we look at the size of the *cwnd* in terms of round-trip times (RTTs), we find that the growth rate is exponential in terms ofeach round trip time, which is a very aggressive approach:

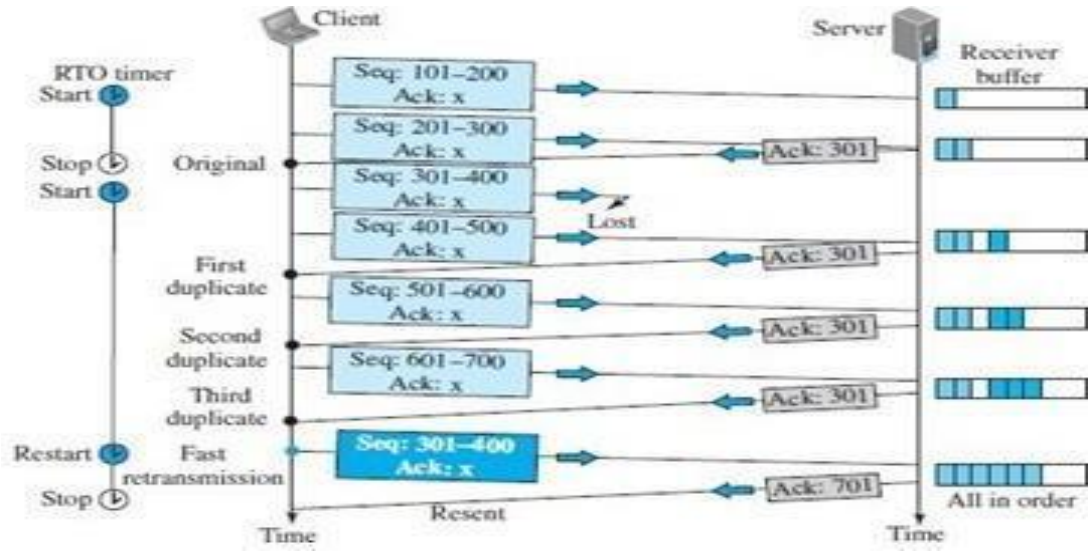
Start	→	$cwnd = 1 \rightarrow 2^0$
After 1 RTT	→	$cwnd = cwnd + 1 = 1 + 1 = 2 \rightarrow 2^1$
After 2 RTT	→	$cwnd = cwnd + 2 = 2 + 2 = 4 \rightarrow 2^2$
After 3 RTT	→	$cwnd = cwnd + 4 = 4 + 4 = 8 \rightarrow 2^3$

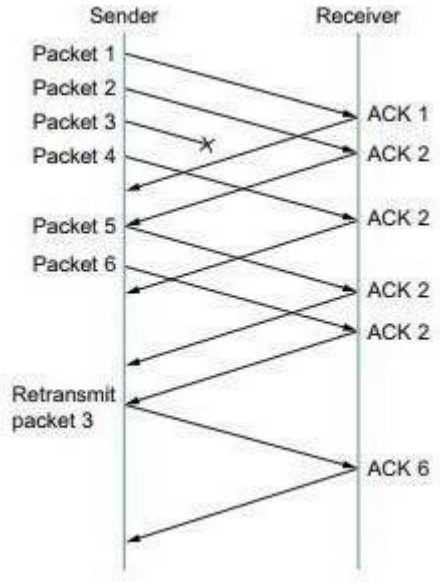
A slow start cannot continue indefinitely. There must be a threshold to stop this phase. The sender keeps track of a variable named *ssthresh* (slow-start threshold). When the

size of the window in bytes reaches this threshold, slow start stops and the next phase starts. For each ACK, the *cwnd* is increased by only 1. Hence, if two segments are acknowledged cumulatively, thesize of the *cwnd* increases by only 1, not 2. The growth is still exponential, but it is not a power of 2.

FAST RETRANSMIT AND FAST RECOVERY

The idea of fast retransmit is straight forward. Every time a data packet arrives at the receiving side, the receiver responds with an acknowledgement, even if this sequence number has already been acknowledged. Thus, when a packet arrives out of order – that is, TCP cannot yet acknowledge the data the packet contains because earlier data has not yet arrived – TCP resends the same acknowledgement it sent the last time. This second transmission of the acknowledgement is called a duplicate ACK.When the sending side sees a duplicate ACK , it knows that the other side must have received a packet out of order, which suggests that an earlier packet has might have been lost. Since it is also possible that the earlier packet has only been delayed rather than lost , the sender waits un til it sees some number of duplicate ACKs and then retransmits the missing packet. In practice, TCP waits until it has seen three duplicate ACKs before retransmitting the packet.





Fast Recovery

The **fast-recovery** algorithm is optional in TCP. The old version of TCP did not use it, but the new versions try to use it. It starts when three duplicate ACKs arrive, which is interpreted as light congestion in the network. Like congestion avoidance, this algorithm is also an additive increase, but it increases the size of the congestion window when a duplicate ACK arrives (after the three duplicate ACKs that trigger the use of this algorithm). We can say

If a duplicate ACK arrives, *cwnd* = *cwnd* + (1 / *cwnd).*

9.STREAM CONTROL TRANSMISSION PROTOCOL (SCTP)

* Define Multihoming. **(2)**

* Illustrate the header packet of SCTP**. (2)**

Stream Control Transmission Protocol (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.

SCTP Services

Before discussing the operation of SCTP, let us explain the services offered by SCTP to the application-layer processes.

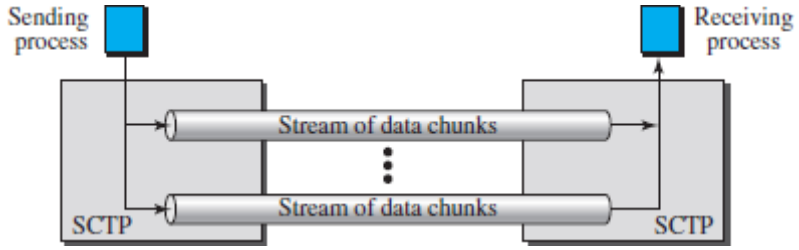
Process-to-Process Communication

SCTP, like UDP or TCP, provides process-to-process communication.

Multiple Streams

We learned that TCP is a stream-oriented protocol. Each connection between a TCP client and a TCP server involves a single stream. The problem with this approach is that a loss at any point in the stream blocks the delivery of the rest of the data. This can be acceptable when we are transferring text; it is not when we are sending real-time data such as audio or video. 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. Figure shows the idea of multiple-stream delivery.



Multihoming

A TCP connection involves one source and one destination IP address. This means that even if the sender or receiver is a multihomed host (connected to more than one physical address with multiple IP addresses), only one of these IP addresses per end can be utilized during the connection. An SCTP association, on the other hand, 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. This fault-tolerant feature is very helpful when we are sending and receiving a real-time payload such as Internet telephony. Figure shows the idea of multihoming.

Full-Duplex Communication

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

Connection-Oriented Service

Like TCP, SCTP is a connection-oriented protocol. However, in SCTP, a connection is called an *association*.

Reliable Service

SCTP, like TCP, is a reliable transport protocol. It uses an acknowledgment mechanism to check the safe and sound arrival of data. We will discuss this feature further in the section on error control.

SCTP Features

The following shows the general features of SCTP.

Transmission Sequence Number (TSN)

The unit of data in SCTP is a data chunk, which may or may not have a one-to-one relationship with the message coming fromthe process because of fragmentation . Data transfer in SCTP is controlled by numbering the data chunks.

SCTP uses a **transmission sequence number (TSN)** to number the data chunks. In other words, the TSN in SCTP plays a role analogous to the sequence number in TCP. TSNs are 32 bits long and randomly initialized between 0 and 2_{32} - 1. Each data chunk must carry the corresponding TSN in its header.

Stream Identifier (SI)

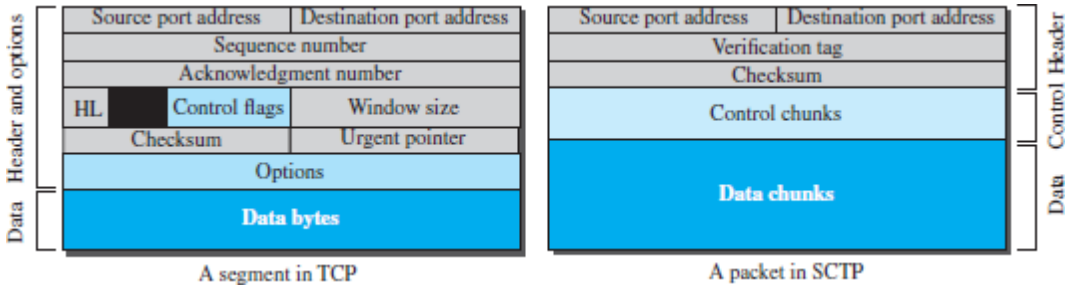
In SCTP, there may be several streams in each association. Each stream in SCTP needs to be identified using a **stream identifier (SI).** Each data chunk must carry the SI in its header so that when it arrives at the destination, it can be properly placed in its stream. The SI is a 16-bit number starting from 0.

Stream Sequence Number (SSN)

When a data chunk arrives at the destination SCTP, it is delivered to the appropriate stream and in the proper order. This means that, in addition to an SI, SCTP defines each data chunk in each stream with a **stream sequence number (SSN).**

Packets

In TCP, a segment carries data and control information. Data are carried as a collection of bytes; control information is defined by six control flags in the header. The design of SCTP is totally different: data are carried as data chunks, control information as control chunks. Several control chunks and data chunks can be packed together in a packet. A packet in SCTP plays the same role as a segment in TCP.



In SCTP, we have data chunks, streams, and packets. An association may send many packets, a packet may contain several chunks, and chunks may belong to different streams. To make the definitions of these terms clear, let us suppose that process A needs to send 11 messages to process B in three streams. The first four messages are in the first stream, the second three messages are in the second stream, and the last four messages are in the third stream. Although a message, if long, can be carriedby several data chunks, we assume that each message fits into one data chunk. Therefore, we have 11 data chunks in three streams. The application process delivers 11 messages to SCTP, where each message is earmarked for the appropriate stream.

Although the process could deliver one message from the first stream and then another from the second, we assume that itdelivers all messages belonging to the first stream first, all messages belonging to the second stream next, and finally, all messages

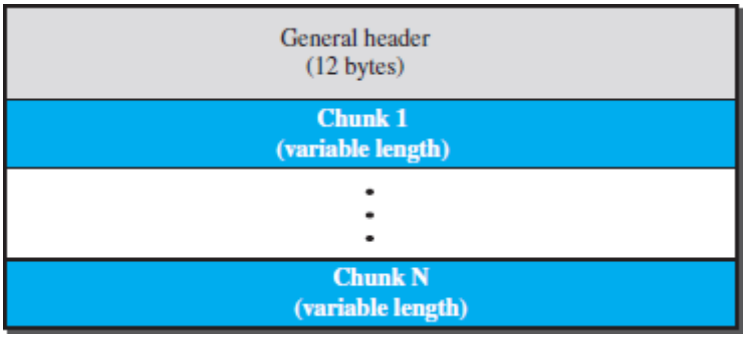
belonging to the last stream. Data chunks in stream 0 are carried in the first and part of the second packet; those in stream 1 are carried in the second and the third packet; those in stream 2 are carried in the third and fourth packet.
Note that each data chunk needs three identifiers: TSN, SI, and SSN. TSN is a cumulative number

Acknowledgment Number

TCP acknowledgment numbers are byte-oriented and refer to the sequence numbers. SCTP acknowledgment numbers are chunk-oriented. They refer to the TSN. A second difference between TCP and SCTP acknowledgments is the control information. Recall that this information is part of the segment header in TCP. To acknowledge segments that carry only control information, TCP uses a sequence number and acknowledgment number.

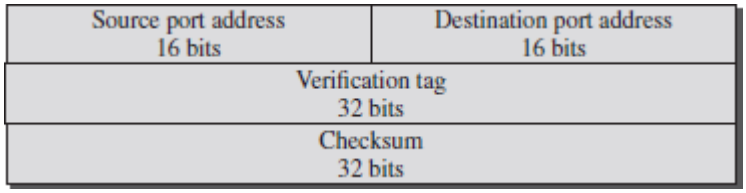
Packet Format

An SCTP packet has a mandatory general header and a set of blocks called chunks. There are two types of chunks: control chunks and data chunks. A control chunk controls and maintains the association; a data chunk carries user data. In a packet, the control chunks come before the data chunks



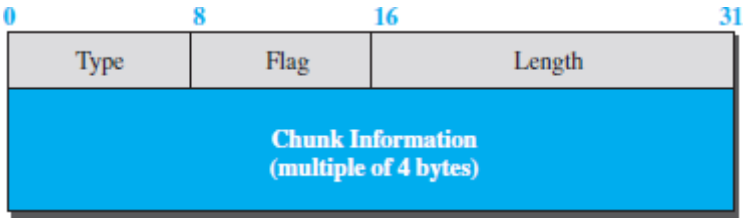
General Header

The *general header* (packet header) defines the end points of each association to which the packet belongs, guarantees that the packet belongs to a particular association, and preserves the integrity of the contents of the packet including the header itself. The format of the general header is shown in Figure.



Chunks

Control information or user data are carried in chunks. Chunks have a common layout, as shown in Figure



Types of Chunks

SCTP defines several types of chunks, as shown in Table

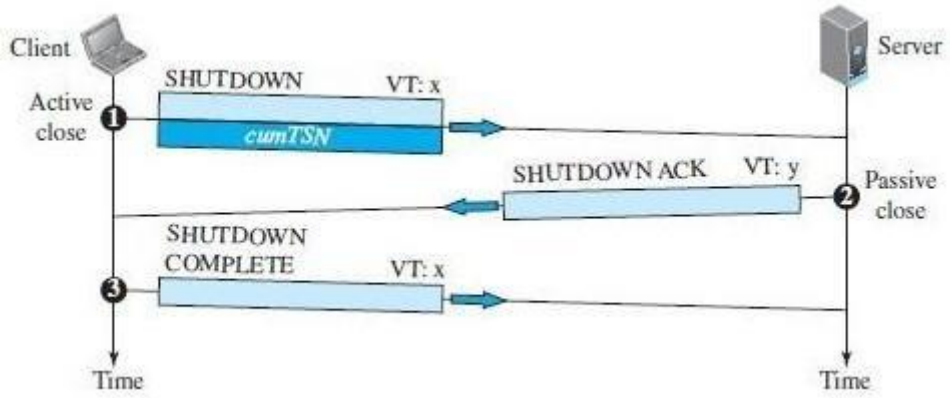
Type	*Chunk*	*Description*
0	DATA	User data
1	INIT	Sets up an association
2	INIT ACK	Acknowledges INIT chunk
3	SACK	Selective acknowledgment
4	HEARTBEAT	Probes the peer for liveliness
5	HEARTBEAT ACK	Acknowledges HEARTBEAT chunk
6	ABORT	Aborts an association
7	SHUTDOWN	Terminates an association
8	SHUTDOWN 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

An SCTP Association

SCTP, like TCP, is a connection-oriented protocol. However, a connection in SCTP is called an *association* to emphasizemultihoming.

Association Establishment

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



The steps, in a normal situation, are as follows:

1. The client sends the first packet, which contains an INIT chunk. The **verification tag (VT)** of this packet (defined in the general header) is 0 because no verification tag has yet been defined for this direction (client to server). The INIT tag includes an **initiation tag** to be used for packets from the other direction (server to client). The chunk also defines the **initial TSN** for this direction and advertises a value for *rwnd*. The value of *rwnd* is normally advertised in a SACK chunk; it is done here because SCTP allows the inclusion of a DATA chunk in the third and fourth packets; the server must be aware of the available client buffer size. Note that no other chunks can be sent with the first packet.

2. The server sends the second packet, which contains an INIT ACK chunk. The verification tag is the value of the initial tagfield in the INIT chunk. This chunk initiates the tag to be used in the other direction, defines the initial TSN, for data flow from server to client, and sets the server's *rwnd*. The value of *rwnd* is defined to allow the client to send a DATA chunk with the third packet. The INIT ACK also sends a cookie that defines the state of the server at this moment.

3. 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.

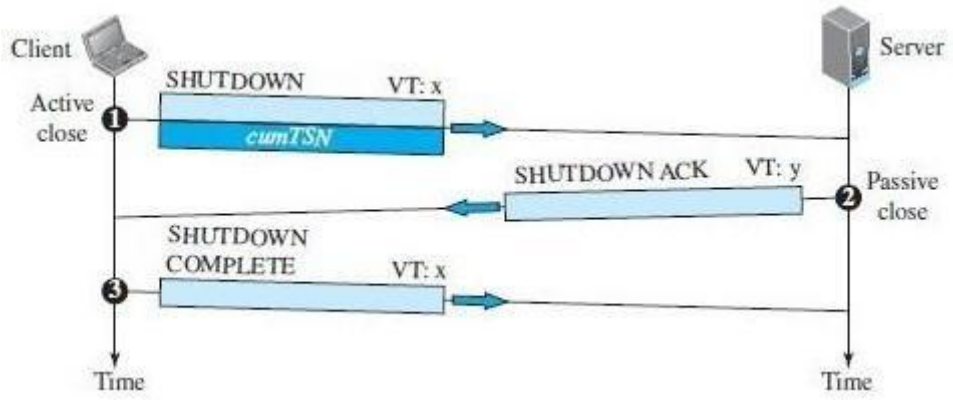
4. The server sends the fourth packet, which includes the COOKIE ACK chunk that acknowledges the receipt of the COOKIEECHO chunk. SCTP allows the inclusion of data chunks with this packet.

Data Transfer

The whole purpose of an association is to transfer data between two ends. After the association is established, bidirectional datatransfer can take place. The client and the server can both send data. Like TCP, SCTP supports piggybacking. There is a major difference, however, between data transfer in TCP and SCTP. TCP receives messages from a process as a stream of bytes without recognizing any boundary between them. The process may insert some boundaries for its peer use, but TCP treats that mark as part of the text. In other words, TCP takes each message and appends it to its buffer. A segment can carry parts of two different messages. The only ordering system imposed by TCP is the byte numbers. SCTP, on the other hand, recognizes and maintains boundaries. Each message coming from the process is treated as one unit and inserted into a DATA chunk unless it isfragmented. In this sense, SCTP is like UDP, with one big advantage: data chunks are related to each other.

Association Termination

In SCTP, like TCP, either of the two parties involved in exchanging data (client or server) can close the connection. However, unlike TCP, SCTP does not allow a "halfclosed" 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, as shown in Figure

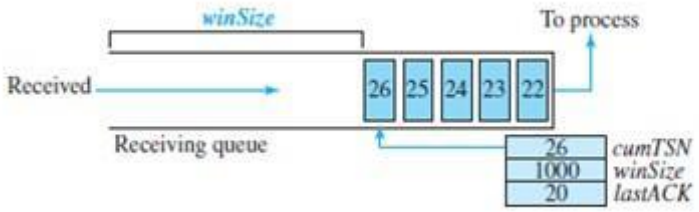


Flow Control

Flow control in SCTP is similar to that in TCP. In TCP, we need to deal with only one unit of data, the byte. In SCTP, we need to handle two units of data, the byte and the chunk. The values of *rwnd* and *cwnd* are expressed in bytes; the values of TSN and acknowledgments are expressed in chunks.

Receiver Site

The receiver has one buffer (queue) and three variables. The queue holds the received data chunks that have not yet been read bythe 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 cumulative acknowledgment, *lastACK*.



1. 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.

2. When the process reads a chunk, it removes it from the queue and adds the size of the removed chunk to *winSize* (recycling). 3.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. The value of *lastACK* is then updated to hold the value of *cumTSN*.

Sender Site

The sender has one buffer (queue) and three variables: *curTSN, rwnd,* and *inTransit,* as shown in Figure below. We assume eachchunk is 100 bytes long. The buffer holds the chunks produced by the process that have either 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 beensent 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 is the procedure used by the sender.

1. 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 one 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.
2. 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 totalsize of the discarded chunks. The value of *rwnd* is updated with the value of the advertised window in the SACK.

Error Control

SCTP, like TCP, 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

In our design, 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

Sender Site

At the sender site, our design demands two buffers (queues): a sending queue and a retransmission queue.

8.CONGESTION AVOIDANCE (DECBIT, RED)

- TCP repeatedly increases the load it imposes on the network in an effort to find the point at which congestion occurs, and then it backs off from this point.
- An appealing alternative, but one that has not yet been widely adopted, is to predict when congestion is about to happen and then to reduce the rate at which hosts send data just before packets start being discarded.

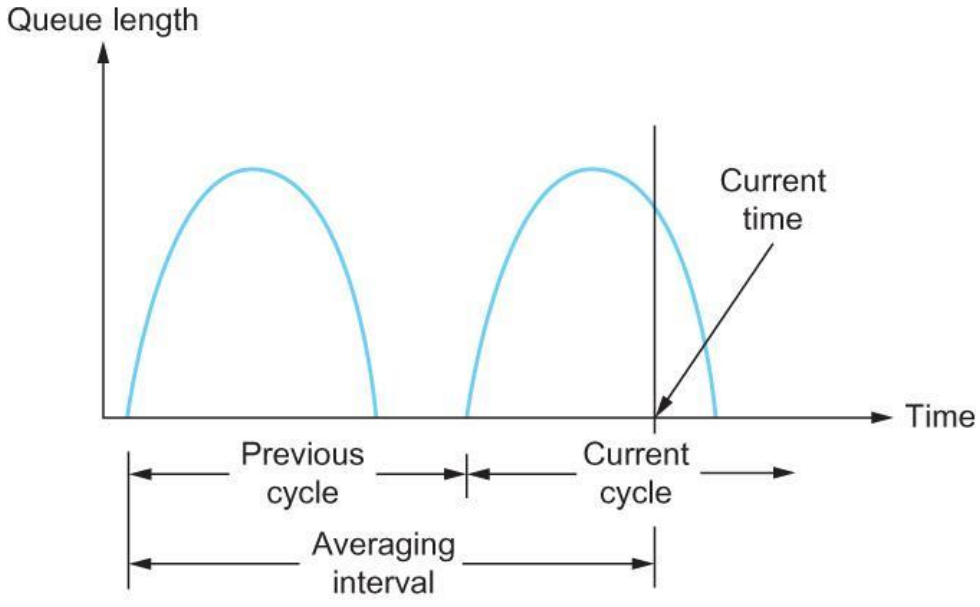
We call such a strategy congestion avoidance, to distinguish it from congestion control

Congestion Avoidance Mechanism

- DEC Bit
  - The first mechanismwas developed for use on the Digital Network Architecture (DNA), a connectionless network with a connection-oriented transport protocol.
  - This mechanism could, therefore, also be applied to TCP and IP.
  - As noted above, the idea here is to more 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.
- Finally, the source adjusts its sending rate so as to avoid congestion
- A single congestion bit is added to the packet header. 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.
- This average queue length is measured over a time interval that spans the last busy+idle cycle, plus the current busy cycle.
- Essentially, the router calculates the area under the curve and divides this value by the time interval to compute the average queue length.
- Using a queue length of 1 as the trigger for setting the congestion bit is a trade-off between significant queuing (and hence higher throughput) and increased idle time (and hence lower delay).
- In other words, a queue length of 1 seems to optimize the power function.
- The source records how many of its packets resulted in some router setting the congestion bit.
- In particular, the source maintains a congestion window, just as in TCP, and watches to see what fraction of the last window's worth of packets resulted in the bit being set.
- If less than 50% of the packets had the bit set, then the source increases its congestion window by one packet.
- If 50% or more of the last window's worth of packets had the congestion bit set, then the source decreases its congestion window to 0.875 times the previous value.
- The value 50% was chosen as the threshold based on analysis that showed it to correspond to the peak of the power curve. The "increase by 1, decrease by 0.875" rule was selected because additive increase/multiplicative decrease makes the mechanism stable.

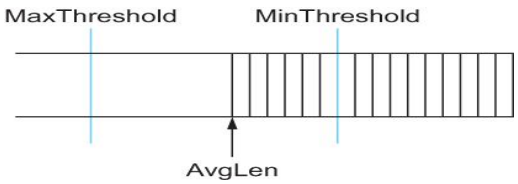


Computing average queue length at a router

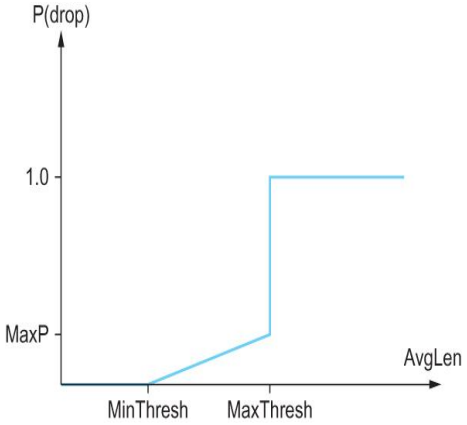
- Random Early Detection (RED)
  - A second mechanism, called random early detection (RED), is similar to the DECbit scheme in that each router is programmed to monitor its own queue length, and when it detects that congestion is imminent, to notify the source to adjust its congestion window. RED, invented by Sally Floyd and Van Jacobson in the early 1990s, differs from the DECbit scheme in two major ways:
  - The first is that rather than explicitly sending a congestion notification message to the source, RED is most commonly implemented such that it implicitly notifies the source of congestion by dropping one of its packets.
  - The source is, therefore, effectively notified by the subsequent timeout or duplicate ACK.
  - RED is designed to be used in conjunction with TCP, which currently detects congestion by means of timeouts (or some other means of detecting packet loss such as duplicate ACKs).
  - As the "early" part of the RED acronym suggests, the gateway drops the packet earlier than it would have to, so as to notify the source that it should decrease its congestion window sooner than it would normally have.
  - In other words, the router drops a few packets before it has exhausted its buffer space completely, so as to cause the source to slow down, with the hope that this will mean it does not have to drop lots of packets later on.
  - The second difference between RED and DECbit is in the details of how RED decides when to drop a packet and what packet it decides to drop.
  - To understand the basic idea, consider a simple FIFO queue. Rather than wait for the queue to become completely full and then be forced to drop each arriving packet, we could decide to drop each arriving packet with some drop probability whenever the queue length exceeds some drop level.

    This idea is called early random drop. The RED algorithm defines the details of how to monitor the queue length and when to drop a packet

    - First, RED computes an average queue length using a weighted running average similar to the one used in the original TCP timeout computation. That is, AvgLen is computed as
      - AvgLen = (1 − Weight) × AvgLen + Weight × SampleLen
      - where 0 < Weight < 1 and SampleLen is the length of the queue when a sample measurement is made.
    - In most software implementations, the queue length is measured every time a new packet arrives at the gateway.
    - In hardware, it might be calculated at some fixed sampling interval.
    - Second, RED has two queue length thresholds that trigger certain activity: MinThreshold and MaxThreshold.
    - When a packet arrives at the gateway, RED compares the current AvgLen with these two thresholds, according to the following rules:
      - if AvgLen δ MinThreshold
        - queue the packet
      - if MinThreshold < AvgLen < MaxThreshold
        - calculate probability P
        - drop the arriving packet with probability P
      - if MaxThreshold δ AvgLen
        - drop the arriving packet
    - P is a function of both AvgLen and how long it has been since the last packet was dropped.
    - Specifically, it is computed as follows:
      - TempP = MaxP × (AvgLen − MinThreshold)/(MaxThreshold − MinThreshold)
      - P = TempP/(1 − count × TempP)



RED thresholds on a FIFO queue



Drop probability function for RED

- Source-based Congestion Avoidance
  - The general idea of these techniques is to watch for some sign from the network that some router's queue is building up and that congestion will happen soon if nothing is done about it.
  - For example, the source might notice that as packet queues build up in the network's routers, there is a measurable increase in the RTT for each successive packet it sends.
  - One particular algorithm exploits this observation as follows:
    - The congestion window normally increases as in TCP, but every two round-trip delays the algorithm checks to see if the current RTT is greater than the average of the minimum and maximum RTTs seen so far.
    - If it is, then the algorithm decreases the congestion window by one-eighth.
  - A second algorithm does something similar. The decision as to whether or not to change the current window size is based on changes to both the RTT and the window size.
  - The window is adjusted once every two round-trip delays based on the product
    - (CurrentWindow − OldWindow)×(CurrentRTT − OldRTT)
    - If the result is positive, the source decreases the window size by one-eighth;
    - if the result is negative or 0, the source increases the window by one maximum packet size.
    - Note that the window changes during every adjustment; that is, it oscillates around its optimal point.

10.QUALITY OF SERVICE

Approaches to QoS Support

- fine-grained approaches, which provide QoS to individual applications or flows
- coarse-grained approaches, which provide QoS to large classes of data or aggregated traffic

■ **Integrated Services (RSVP)**

- The term “Integrated Services” (often called IntServ for short) refers to a body of work that was produced by the IETF around 1995–97.
- The IntServ working group developed specifications of a number of *service classes designed to meet the needs of some of the* application types described above.
- It also defined how RSVP could be used to make reservations using these service classes.
- Service Classes
    - Guaranteed Service

The network should guarantee that the maximum delay that any packet will experience has some specified value

- Controlled Load Service

Overview of Mechanisms

Flowspec

With a best-effort service we can just tell the network where we want our packets to go and leave it at that, a real-time service involves telling the network something more about the type of service we require.The set of information that we provide to the network is referred to as a *flowspec.*

Admission Control

When we ask the network to provide us with a particular service, the network needs to decide if it can in fact provide that service. The process of deciding when to say no is called *admission control*.

Resource Reservation

We need a mechanism by which the users of the network and the components of the network itself exchange information such as requests for service, flowspecs, and admission control decisions. We refer to this process as *resource reservation*

The aim of the controlled load service is to emulate a lightly loaded network for those applications that request the service, even though the network as a whole may in fact be heavily loaded

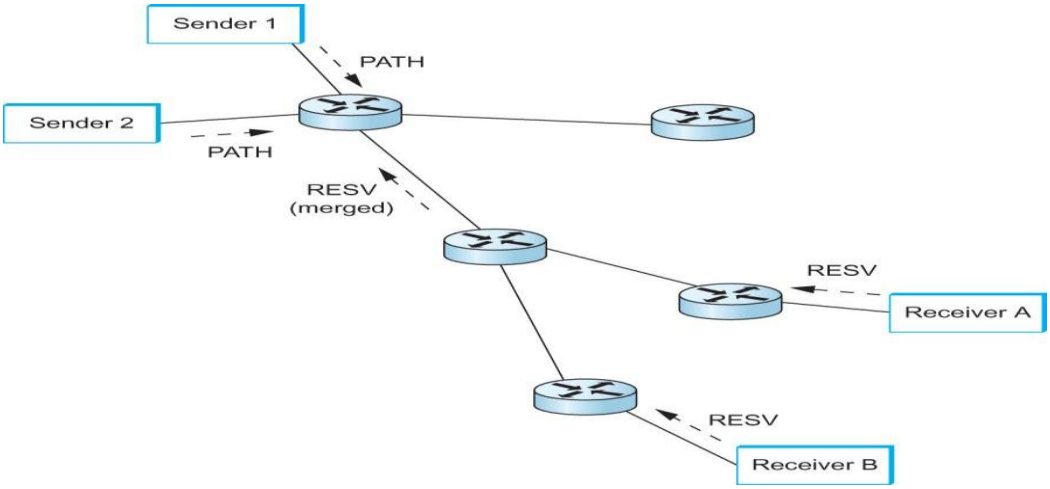
- **Overview of Mechanisms**
    - Packet Scheduling
- Finally, when flows and their requirements have been described, and admission control decisions have been made, the network switches and routers need to meet the requirements of the flows.
- A key part of meeting these requirements is managing the way packets are queued and scheduled for transmission in the switches and routers.
- This last mechanism is *packet scheduling.*
- Flowspec
    - There are two separable parts to the flowspec:
- The part that describes the flow’s traffic characteristics (called the *TSpec)* and
- The part that describes the service requested from the network (the *RSpec).*
- The RSpec is very service specific and relatively easy to describe.
- For example, with a controlled load service, the RSpec is trivial: The application just requests controlled load service with no additional parameters.
    - With a guaranteed service, you could specify a delay target or bound Tspec

- We need to give the network enough information about the bandwidth used by the flow to allow intelligent admission control decisions to be made
  - For most applications, the bandwidth is not a single number It varies constantly
  - A video application will generate more bits per second when the scene is changing rapidly than when it is still Just knowing the long term average bandwidth is not enough
  - Flowspec
    - Suppose 10 flows arrive at a switch on separate ports and they all leave on the same 10 Mbps link
    - If each flow is expected to send no more than 1 Mbps
- No problem
    - If these are variable bit applications such as compressed video
- They will occasionally send more than the average rate
    - If enough sources send more than average rates, then the total rate at which data arrives at the switch will be more than 10 Mbps
    - This excess data will be queued
    - The longer the condition persists, the longer the queue will get
    - One way to describe the Bandwidth characteristics of sources is called a Token Bucket Filter
    - The filter is described by two parameters
- A token rate *r*
- A bucket depth *B*
    - To be able to send a byte, a token is needed
    - To send a packet of length *n*, *n* tokens are needed
    - Initially there are no tokens
    - Tokens are accumulated at a rate of *r* per second
    - No more than *B* tokens can be accumulated
    - We can send a burst of as many as *B* bytes into the network as fast as we want, but over significant long interval we cannot send more than *r* bytes per second
    - This information is important for admission control algorithm when it tries to find out whether it can accommodate new request for service

■ **Integrated Services (RSVP)**

- **Admission Control**
  - The idea behind admission control is simple: When some new flow wants to receive a particular level of service, admission control looks at the TSpec and RSpec of the flow and tries to decide if the desired service can be provided to that amount of traffic, given the currently available resources, without causing any previously admitted flow to receive worse service than it had requested. If it can provide the service, the flow is admitted; if not, then it is denied.
- **Reservation Protocol**
  - While connection-oriented networks have always needed some sort of setup protocol to establish the necessary virtual circuit state in the switches, connectionless networks like the Internet have had no such protocols.
  - However we need to provide a lot more information to our network when we want a real-time service from it.

- While there have been a number of setup protocols proposed for the Internet, the one on which most current attention is focused is called Resource Reservation Protocol (RSVP). One of the key assumptions underlying RSVP is that it should not detract from the robustness that we find in today's connectionless networks.
- Because connectionless networks rely on little or no state being stored in the network itself, it is possible for routers to crash and reboot and for links to go up and down while end-to-end connectivity is still maintained.
- RSVP tries to maintain this robustness by using the idea of *soft state in the routers.*
- Another important characteristic of RSVP is that it aims to support multicast flows just as effectively as unicast flows
- Initially, consider the case of one sender and one receiver trying to get a reservation for traffic flowing between them.
- There are two things that need to happen before a receiver can make the reservation.
- First, the receiver needs to know what traffic the sender is likely to send so that it can make an appropriate reservation. That is, it needs to know the sender's TSpec.
- Second, it needs to know what path the packets will follow from sender to receiver, so that it can establish a resource reservation at each router on the path. Both of these requirements can be met by sending a message from the sender to the receiver that contains the TSpec.
- Obviously, this gets the TSpec to the receiver. The other thing that happens is that each router looks at this message (called a PATH message) as it goes past, and it figures out the *reverse path that will be used to send* reservations from the receiver back to the sender in an effort to get the reservation to each router on the path.
- Having received a PATH message, the receiver sends a reservation back "up" the multicast tree in a RESV message.
- This message contains the sender's TSpec and an RSpec describing the requirements of this receiver.
- Each router on the path looks at the reservation request and tries to allocate the necessary resources to satisfy it. If the reservation can be made, the RESV request is passed on to the next router.
- If not, an error message is returned to the receiver who made the request. If all goes well, the correct reservation is installed at every router between the sender and the receiver.
- As long as the receiver wants to retain the reservation, it sends the same RESV message about once every 30 seconds.



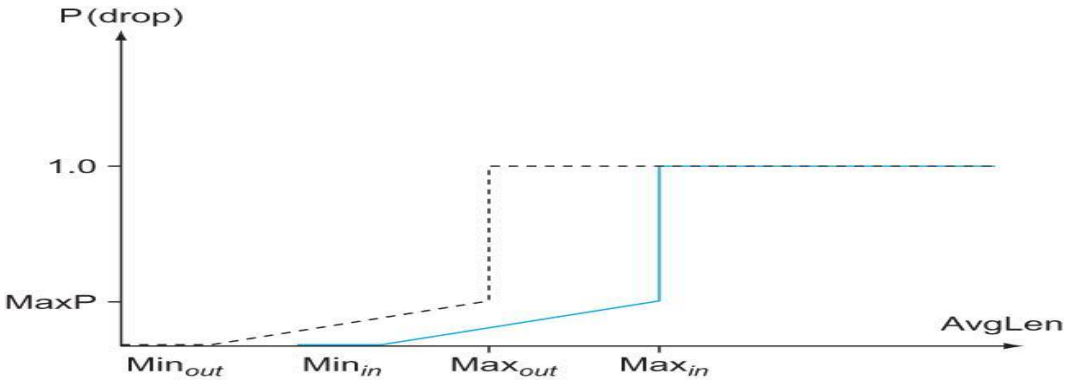
Making reservations on a multicast tree

Packet Classifying and Scheduling

- Once we have described our traffic and our desired network service and have installed a suitable reservation at all the routers on the path, the only thing that remains is for the routers to actually deliver the requested service to the data packets. There are two things that need to be done:
- Associate each packet with the appropriate reservation so that it can be handled correctly, a process known as *classifying packets.*
- Manage the packets in the queues so that they receive the service that has been requested, a process known as packet *scheduling.*

Differentiated Services

- Whereas the Integrated Services architecture allocates resources to individual flows, the Differentiated Services model (often called DiffServ for short) allocates resources to a small number of classes of traffic.
- In fact, some proposed approaches to DiffServ simply divide traffic into two classes.
- Suppose that we have decided to enhance the best-effort service model by adding just one new class, which we'll call "premium."
- Clearly we will need some way to figure out which packets are premium and which are regular old best effort.
- Rather than using a protocol like RSVP to tell all the routers that some flow is sending premium packets, it would be much easier if the packets could just identify themselves to the router when they arrive. This could obviously be done by using a bit in the packet header—if that bit is a 1, the packet is a premium packet; if it's a 0, the packet is best effort
- **The Expedited Forwarding (EF) PHB**
- One of the simplest PHBs to explain is known as "expedited forwarding" (EF). Packets marked for EF treatment should be forwarded by the router with minimal delay and loss.
- The only way that a router can guarantee this to all EF packets is if the arrival rate of EF packets at the router is strictly limited to be less than the rate at which the router can forward EF packets.
- **The Assured Forwarding (AF) PHB**
- The "assured forwarding" (AF) PHB has its roots in an approach known as "RED with In and Out" (RIO) or "Weighted RED," both of which are enhancements to the basic RED algorithm.
- For our two classes of traffic, we have two separate drop probability curves. RIO calls the two classes "in" and "out" for reasons that will become clear shortly.
- Because the "out" curve has a lower MinThreshold than the "in" curve, it is clear that, under low levels of congestion, only packets marked "out" will be discarded by the RED algorithm. If the congestion becomes more serious, a higher percentage of "out" packets are dropped, and then if the average queue length exceeds Min_{in}, RED starts to drop "in" packets as well.



- RED with In and Out drop probabilities