UNIT IV TRANSPORT LAYER **9**

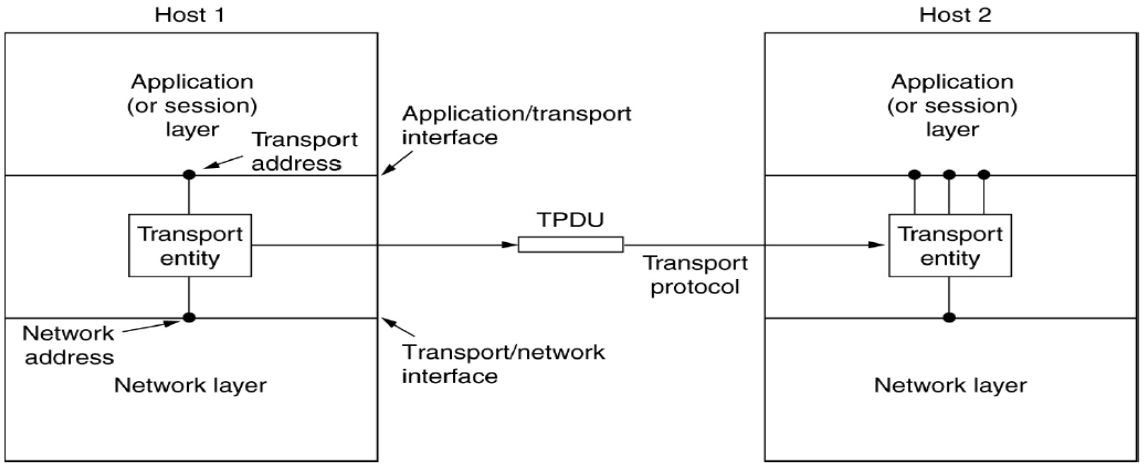
Transport Layer Services - Protocols - UDP - TCP: Transition Diagram, Flow Control, Error Control, Congestion Control - SCTP - QoS: Flow Control to improve QoS - Integrated Services - Differentiated Services - Client Server Programming.

TRANSPORT LAYER

The network layer provides end-to-end packet delivery using datagrams or virtual circuits. The transport layer builds on the network layer to provide **data transport from a process on a source machine to a process on a destination machine** with a desired level of reliability that is independent of the physical networks currently in use.

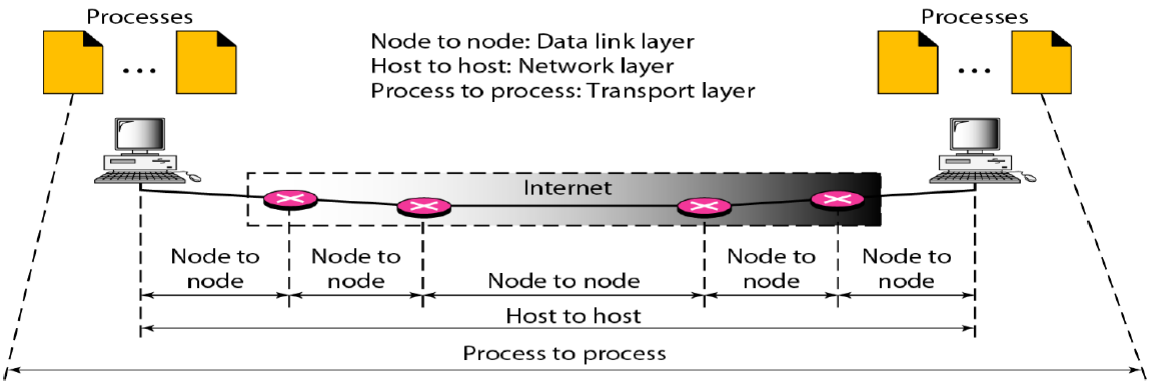
Services Provided to the Upper Layers

The ultimate goal of the transport layer is to provide efficient, reliable, and cost-effective data transmission service to its users, normally processes in the application layer. To achieve this, the transport layer makes use of the services provided by the network layer. The software and/or hardware within the transport layer that does the work is called the **transport entity.**



The network, transport, and application layers.

Types of data deliveries

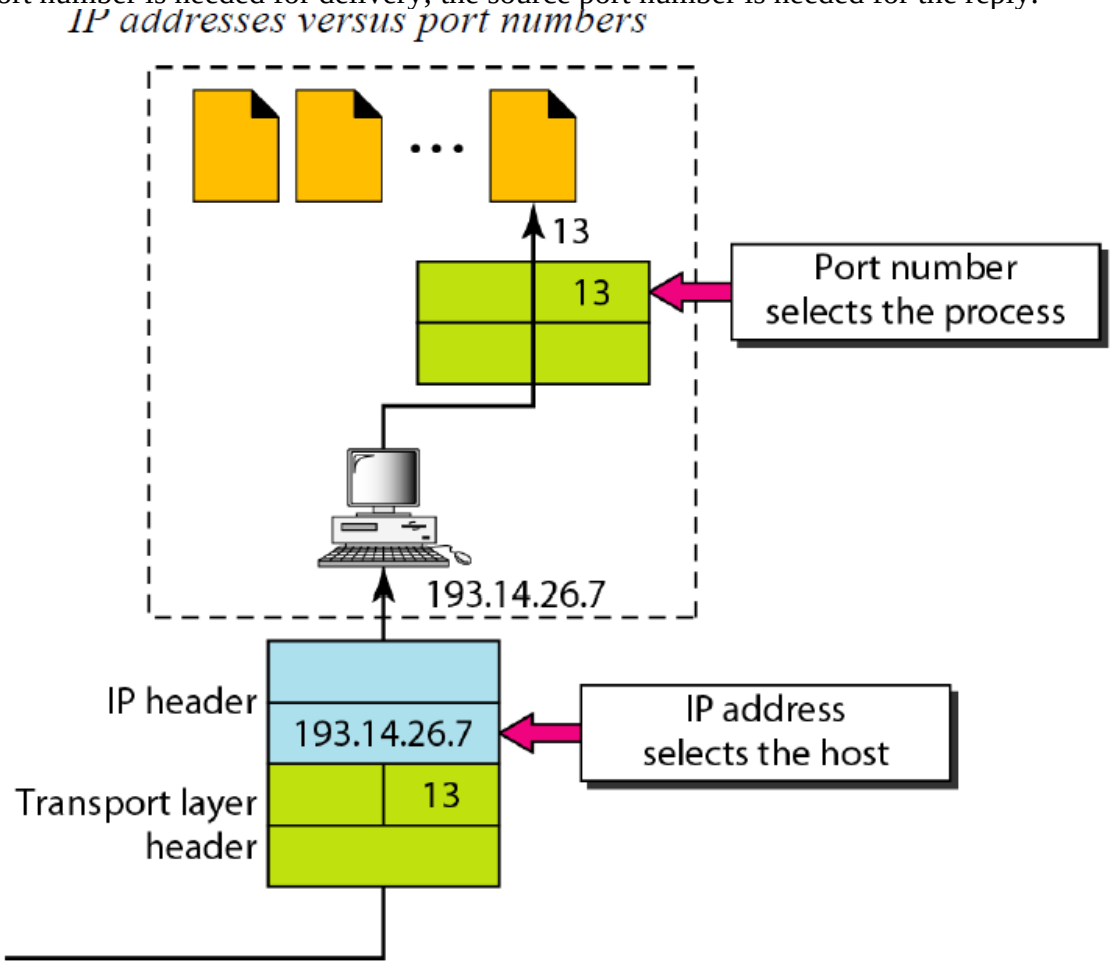


The transport layer is responsible for process-to-process delivery-the delivery of a packet, part of a message, from one process to another. Two processes communicate in a client/server relationship.

Addressing

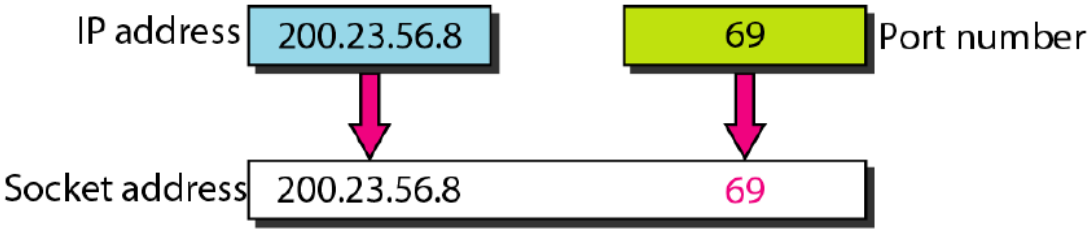
Whenever we need to deliver something to one specific destination among many, we need an address. At the data link layer, we need a MAC address to choose one node among several nodes if the connection is not point-to-point. A frame in the data link layer needs a destination MAC address for delivery and a source address for the next node's reply. At the network layer, we need an IP address to choose one host among millions. A datagram in the network layer needs a destination IP address for delivery and a source IP address for the destination's reply. At the transport layer, we need a transport layer address, called a port number, to choose among multiple processes running on the destination host. The destination port number is needed for delivery; the source port number is needed for the reply.

IP addresses versus port numbers



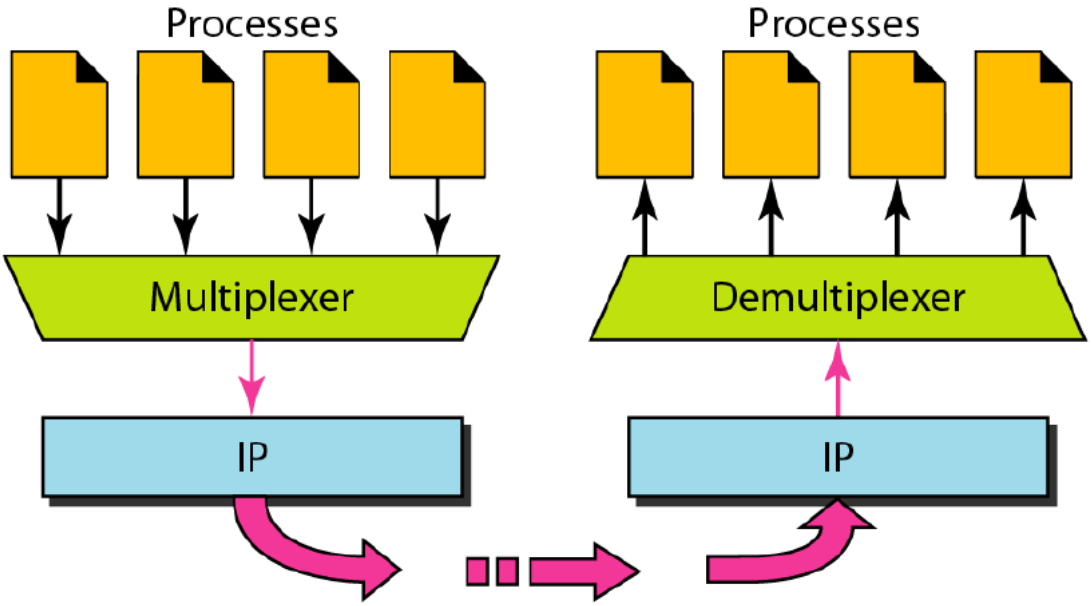
Socket Addresses

Process-to-process delivery needs two identifiers, IP address and the port number, at each end to make a connection. The combination of an IP address and a port number is called a socket address. The client socket address defines the client process uniquely just as the server socket address defines the server process uniquely. A transport layer protocol needs a pair of socket addresses: the client socket address and the server socket address. These four pieces of information are part of the IP header and the transport layer protocol header. The IP header contains the IP addresses; the UDP or TCP header contains the port numbers.



Multiplexing and Demultiplexing

The addressing mechanism allows multiplexing and demultiplexing by the transport Layer.



Multiplexing

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

Demultiplexing

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

Connectionless Versus Connection-Oriented Service

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

Connectionless Service

In a connectionless service, the packets are sent from one party to another with no need for connection establishment or connection release. The packets are not numbered; they may be delayed or lost or may arrive out of sequence. There is no acknowledgment either.

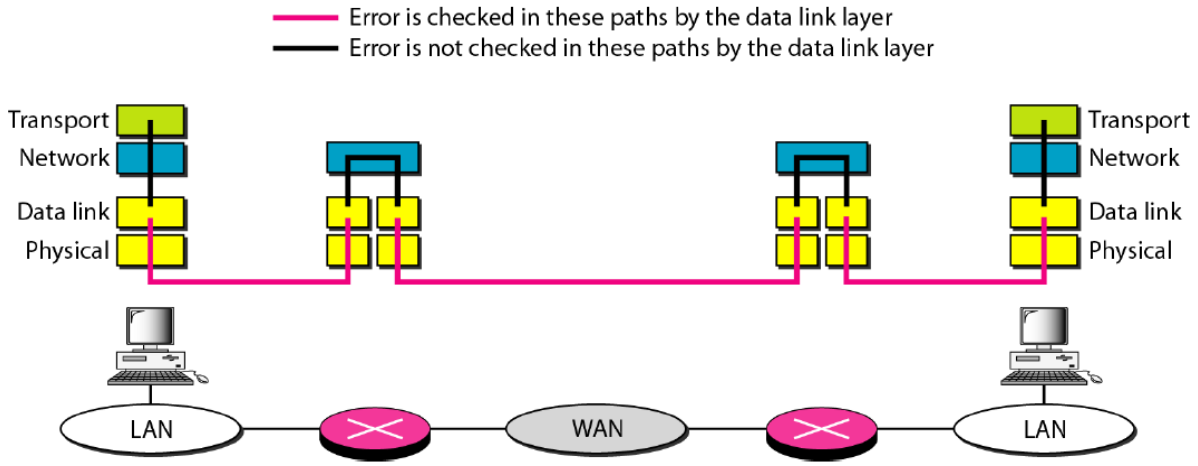
Connection~Oriented *Service*

In a connection-oriented service, a connection is first established between the sender and the receiver. Data are transferred. At the end, the connection is released. The connection-oriented transport service. connections have three phases: establishment, data transfer, and release.

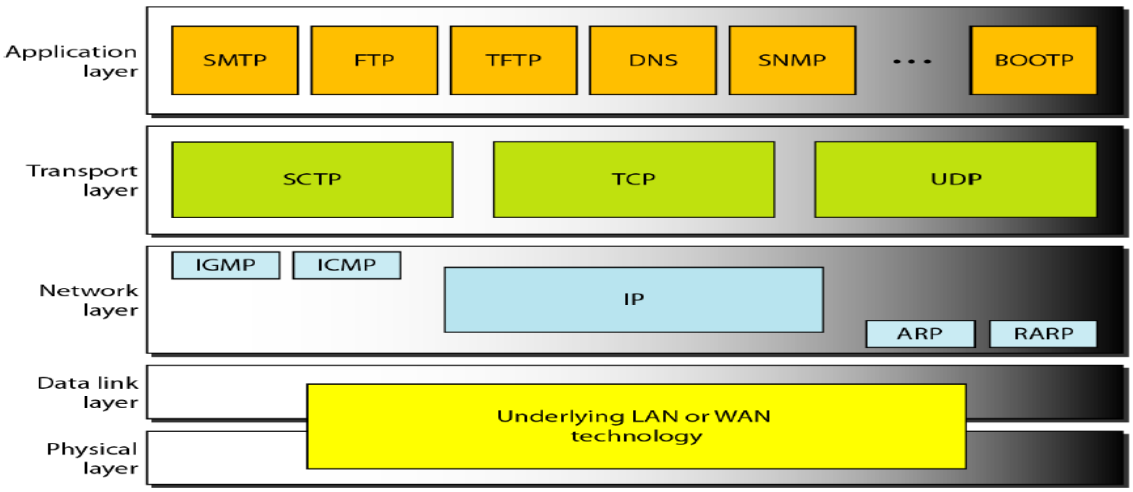
Reliable Versus Unreliable

The transport layer service can be reliable or unreliable. If the application layer program needs reliability, we use a reliable transport layer protocol by implementing flow and error control at the transport layer. This means a slower and more complex service. On the other hand, if the application program does not need reliability because it uses its own flow and error control mechanism or it needs fast service or the nature of the service does not demand flow and error control (real-time applications), then an unreliable protocol can be used.

Error control



Position of UDP, TCP, and SCTP in TCP/IP suite



USER DATAGRAM PROTOCOL (UDP)

The User Datagram Protocol (UDP) is called a connectionless, unreliable transport protocol. It does not add anything to the services of IP except to provide process-to process communication instead of host-to-host communication. Also, it performs very limited error checking. If a process wants to send a small message and does not care much about reliability, it can use UDP.

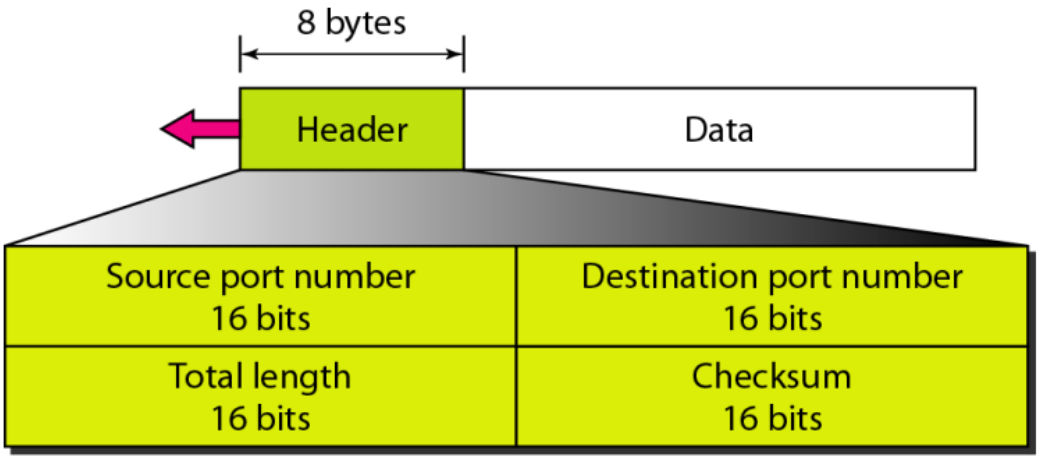
Well-Known Ports for UDP

Port	*Protocol*	*Description*
7	Echo	Echoes a received datagram back to the sender
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
53	Nameserver	Domain Name Service
67	BOOTPs	Server port to download bootstrap information
68	BOOTPc	Client port to download bootstrap information
69	TFTP	Trivial File Transfer Protocol
111	RPC	Remote Procedure Call
123	NTP	Network Time Protocol
161	SNMP	Simple Network Management Protocol
162	SNMP	Simple Network Management Protocol (trap)

User Datagram

UDP packets, called user datagrams, have a fixed-size header of 8 bytes.

Figure ***User datagram format***



The fields are as follows: o Source port number. This is the port number used by the process running on the source host. It is 16 bits long, which means that the port number can range from 0 to 65,535. Ifthe source host is the client (a client sending a request), the port number, in most cases, is an ephemeral port number requested by the process and chosen by the UDP software running on the source host. If the source host is the server (a server sending a response), the port number, in most cases, is a well-known port number.

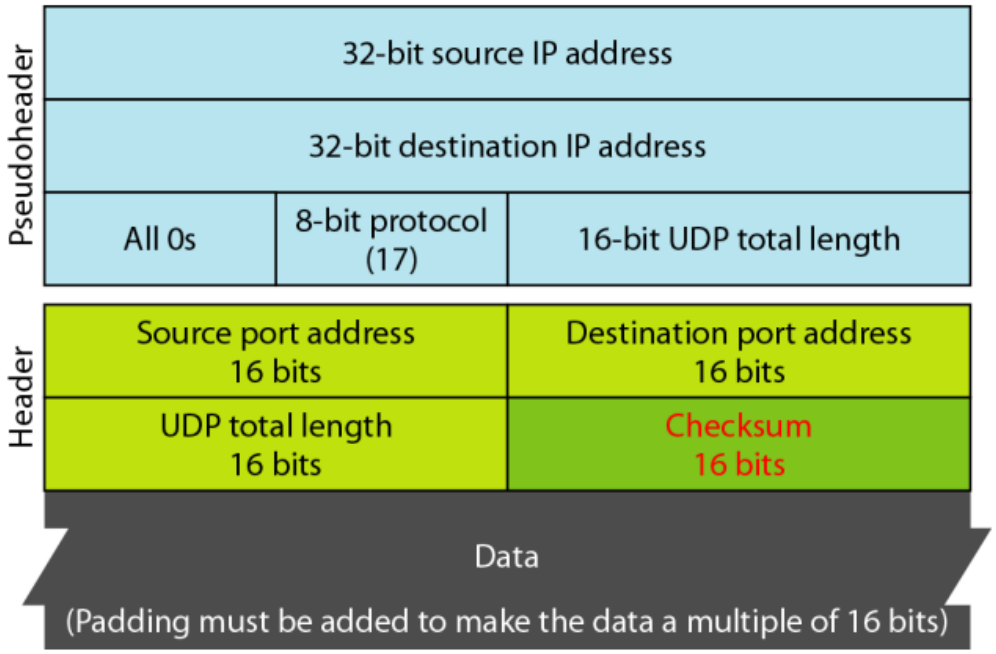
o Destination port number. This is the port number used by the process running on the destination host. It is also 16 bits long. If the destination host is the server (a client sending a request), the port number, in most cases, is a well-known port number. If the destination host is the client (a server sending a response), the port number, in most cases, is an ephemeral port number. In this case, the server copies the ephemeral port number it has received in the request packet. o Length. This is a 16-bit field that 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. How ever, the total length needs to be much less because a UDP user datagram is stored in an IP datagram with a total length of 65,535 bytes.

UDP length = IP length - IP header's length

o Checksum. This field is used to detect errors over the entire user datagram (header plus data).

The UDP checksum calculation is different from the one for IP and ICMP. Here the checksum 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 data gram is to be encapsulated with some fields filled with O.

Figure ***Pseudoheader for checksum calculation***



UDP Operation

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 ifthey are coming from the same source pro cess and going to the same destination program. The user datagrams are not numbered. Also, there is no connection establishment and no connection termination, as is the case for TCP. This means that each user datagram can travel on a different path.

Flow and Error Control

UDP is a very simple, unreliable transport protocol. There is no flow control and hence no window mechanism. The receiver may overflow with incoming messages. 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.

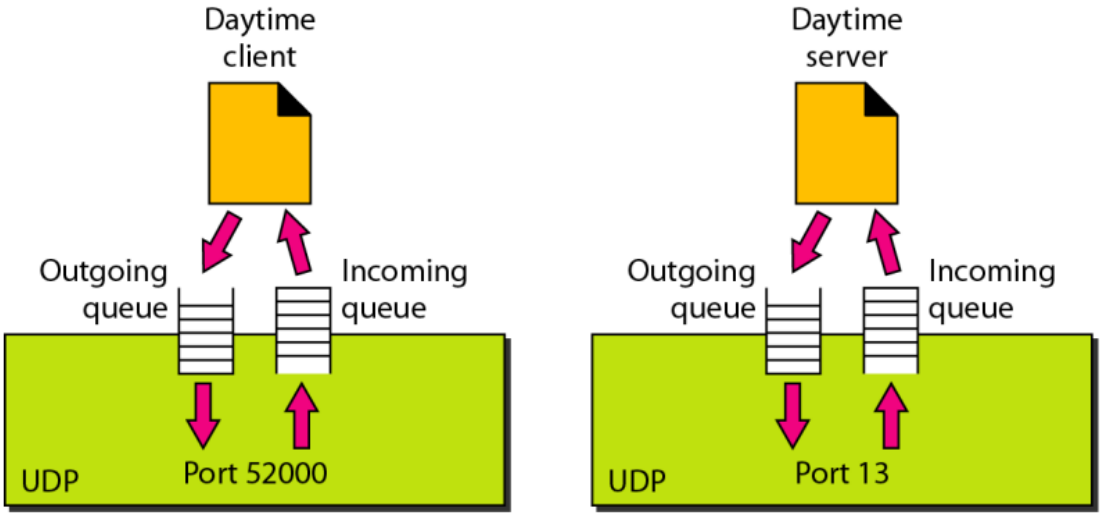
Encapsulation and Decapsulation

To send a message from one process to another, the UDP protocol encapsulates and decapsulates messages in an IP datagram.

Queuing

In UDP, queues are associated with ports.

Figure ***Queues in UDP***



The client process can send messages to the outgoing queue by using the source port number specified in the request. UDP removes the messages one by one and, after adding the UDP header, delivers them to IP. An outgoing queue can overflow. If this happens, the operating system can ask the client process to wait before sending any more messages. When a message arrives for a client, UDP checks to see if an incoming queue has been created for the port number specified in the destination port number field of the user datagram. If there is such a queue, UDP sends the received user datagram to the end of the queue. If there is no such queue, UDP discards the user datagram and asks the ICMP protocol to send a port unreachable message to the server. All the incoming messages for one particular client program, whether coming from the same or a different server, are sent to the same queue. An incoming queue can overflow. If this happens, UDP drops the user datagram and asks for a port unreachable message to be sent to the server.

When a message arrives for a server, UDP checks to see if an incoming queue has been created for the port number specified in the destination port number field ofthe user datagram. If there is such a queue, UDP sends the received user datagram to the end of the queue. Ifthere is no such queue, UDP discards the user datagram and asks the ICMP protocol to send a port unreachable message to the client. All the incoming messages for one particular server, whether coming from the same or a different client, are sent to the same queue. An incoming queue can overflow. Ifthis happens, UDP drops the user datagram and asks for a port unreachable message to be sent to the client. When a server wants to respond to a client, it sends messages to the outgoing queue, using the source port number specified in the request. UDP removes the messages one by one and, after adding the UDP header, delivers them to IP. An outgoing queue can overflow. If this happens, the operating system asks the server to wait before sending any more messages.

Use ofUDP

The following lists some uses of the UDP protocol:

o UDP is suitable for a process that requires simple request-response communication with little concern for flow and error control.

o UDP is suitable for a process with internal flow and error control mechanisms. For example, the Trivial File Transfer Protocol (TFTP) process includes flow and error control. It can easily use UDP.

o UDP is a suitable transport protocol for multicasting. Multicasting capability is embedded in the UDP software but not in the TCP software.

o UDP is used for management processes such as SNMP

o UDP is used for some route updating protocols such as Routing Information Protocol (RIP)

//

TCP

TCP, like UDP, is a process-to-process (program-to-program) protocol. TCP, therefore, like UDP, uses port numbers. Unlike UDP, TCP is a connection oriented protocol; it creates a virtual connection between two TCPs to send data. In addition, TCP uses flow and error control mechanisms at the transport level. In brief, TCP is called a connection-oriented, reliable transport protocol. It adds connection-oriented and reliability features to the services of IP.

TCP Services

1. Process-to-Process Communication

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

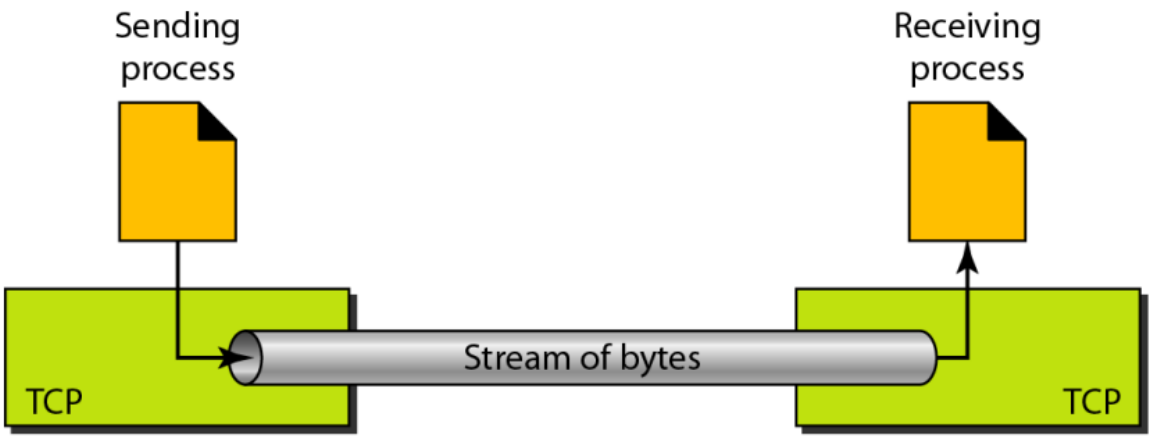
Table ***Well-known ports used by TCP***

Port	*Protocol*	*Description*
7	Echo	Echoes a received datagram back to the sender
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 (data connection)
21	FTP, Control	File Transfer Protocol (control connection)
23	TELNET	Terminal Network
25	SMTP	Simple Mail Transfer Protocol
53	DNS	Domain Name Server
67	BOOTP	Bootstrap Protocol
79	Finger	Finger
80	HTTP	Hypertext Transfer Protocol
111	RPC	Remote Procedure Call

2. Stream Delivery Service

TCP, unlike UDP, is a stream-oriented protocol. TCP, on the other hand, 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 data across the Internet.

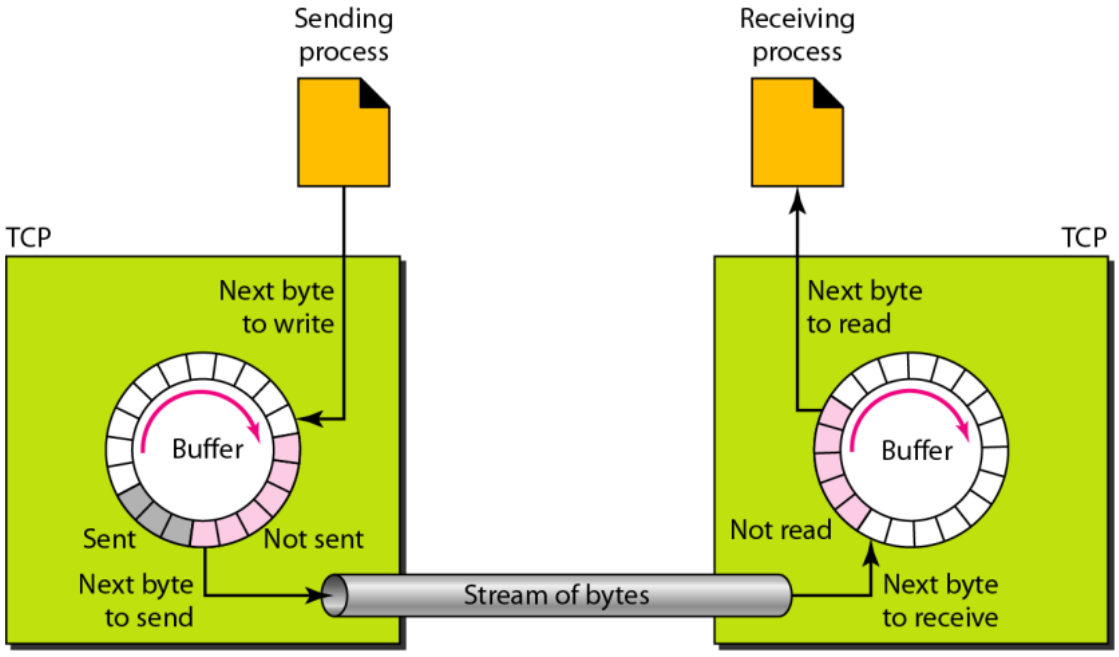
Figure ***Stream delivery***



3. Sending and Receiving Buffers

Because the sending and the receiving processes may not write or read data at the same speed, TCP needs buffers for storage. There are two buffers, the sending buffer and the receiving buffer, one for each direction. One way to implement a buffer is to use a circular array of I-byte locations.

Figure ***Sending and receiving buffers***



4. Full-Duplex Communication

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

5. Connection-Oriented Service

   TCP, unlike UDP, is a connection-oriented protocol. When a process at site A wants to send and receive data from another process at site B, the following occurs:

   1. The two TCPs establish a connection between them.

   2. Data are exchanged in both directions.

   3. The connection is terminated

6. Reliable Service

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

TCP Features

1. 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 the byte number and not the segment number.

- Byte Number

  TCP numbers all data bytes that are transmitted in a connection. Numbering is independent in each direction. When TCP receives bytes of data from a process, it stores them in the sending buffer and numbers them. The numbering does not necessarily start from O. Instead, TCP generates a random number between 0 and 232 - 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 for each segment is the number of the first byte carried in that segment.

- Acknowledgment Number

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

2. Flow Control

TCP, unlike UDP, provides flow control. The receiver of the data controls the amount of data that are to be sent by the sender. This is done to prevent the receiver from being over whelmed with data. The numbering system allows TCP to use a byte-oriented flow control.

3. Error Control

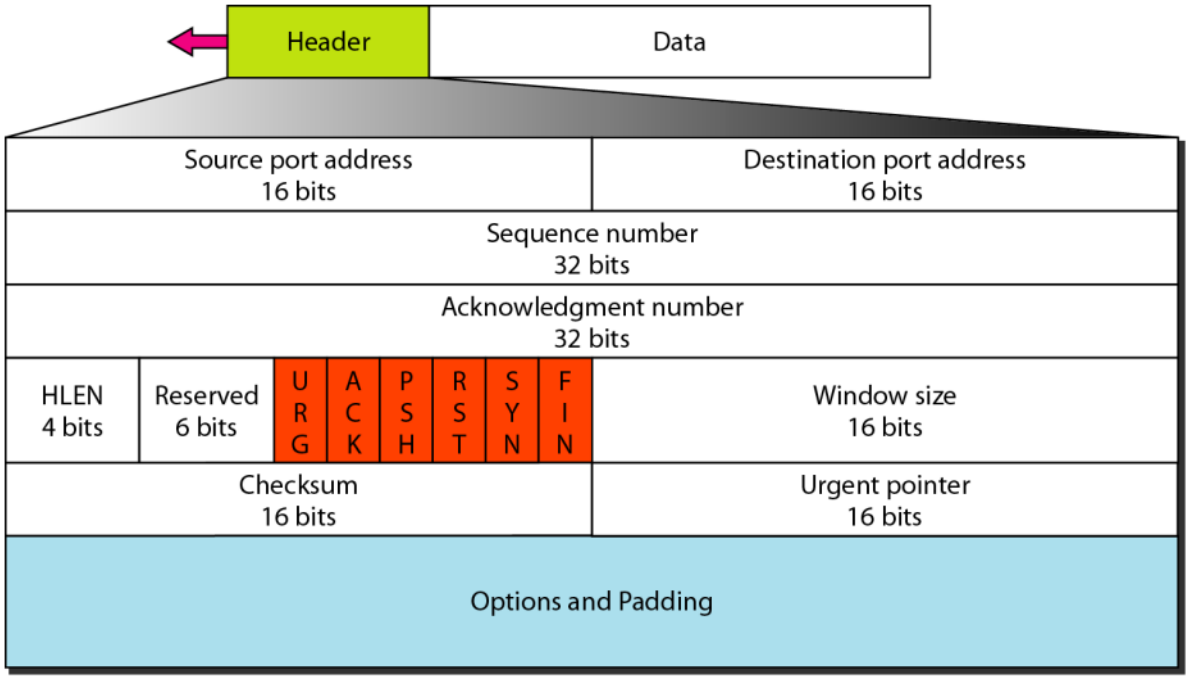
To provide reliable service, TCP implements an error control mechanism. Although error control considers a segment as the unit of data for error detection (loss or corrupted segments), error control is byte-oriented.

Congestion Control TCP, unlike UDP, takes into account congestion in the network. The amount of data sent by a sender is not only controlled by the receiver (flow control), but is also determined by the level of congestion in the network.

Segment

A packet in TCP is called a segment.

Figure ***TCP segment format***



o Source port address.

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

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

o Sequence number.

This 32-bit field defines the number assigned to the first byte of data contained in this segment. The sequence number tells the destination which byte in this sequence comprises the first byte in the segment. During connection establishment, each party uses a random number generator to create an initial sequence number (ISN), which is usually different in each direction.

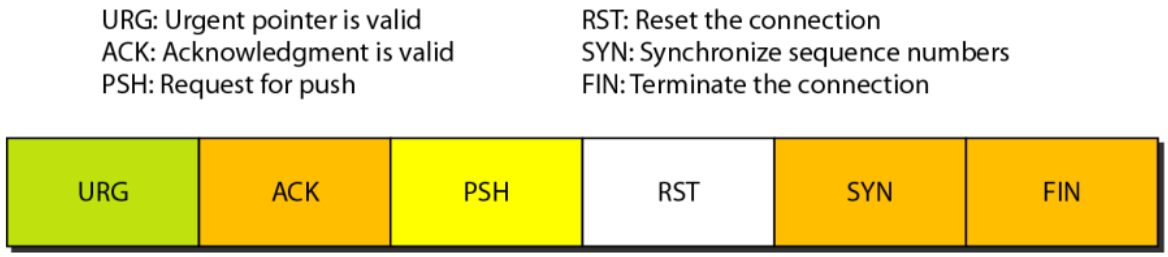
o Acknowledgment number. This 32-bit field defines the byte number that the receiver of the segment is expecting to receive from the other party. If the receiver of the segment has successfully received byte number x from the other party, it defines x + I as the acknowledgment number. Acknowledgment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

o 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 can be between 5 (5 x 4 =20) and 15 (15 x 4 =60).

o Reserved. This is a 6-bit field reserved for future use.

o Control. This field defines 6 different control bits or flags

Figure ***Control field***



These bits enable flow control, connection establishment and termination, connection abortion, and the mode of data transfer in TCP.

Table ***Description of flags in the control field***

Flag	*Description*
URG	The value of the urgent pointer field is valid.
ACK	The value of the acknowledgment field is valid.
PSH	Push the data.
RST	Reset the connection.
SYN	Synchronize sequence numbers during connection.
FIN	Terminate the connection.

- **Window size.**

  This field defines the size of the window, in bytes, that the other party must maintain. 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.

- **Urgent pointer**. This l6-bit field, which is valid only if the urgent flag is set, is used when the segment contains urgent data. It defines the number 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.

A TCP Connection

TCP is connection-oriented. A connection-oriented transport protocol establishes a virtual path between the source and destination. All the segments belonging to a mes sage are then sent over this virtual path. Using a single virtual 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:

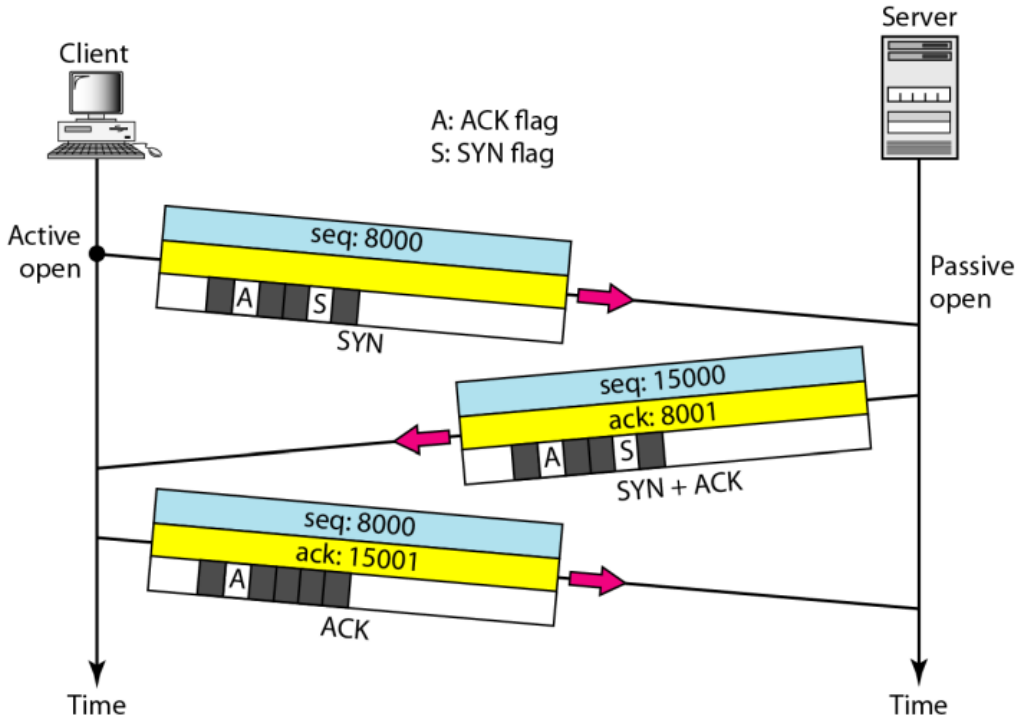
1. connection establishment,
2. data transfer, and
3. connection termination.

1. **Connection Establishment**

TCP transmits data in full-duplex mode. When two TCPs in two machines are connected, they are able to send segments to each other simultaneously. This implies that each party must initialize communication and get approval from the other party before any data are transferred. **Three-Way Handshaking**

The connection establishment in TCP is called three way handshaking.

Figure ***Connection establishment using three-way handshaking***



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. It consumes one sequence number. When the data transfer starts, the sequence number is incremented by 1. We can say that the SYN segment carries no real data, but we can think of it as containing 1 imaginary byte. A SYN segment cannot carry data, but it consumes one sequence number.

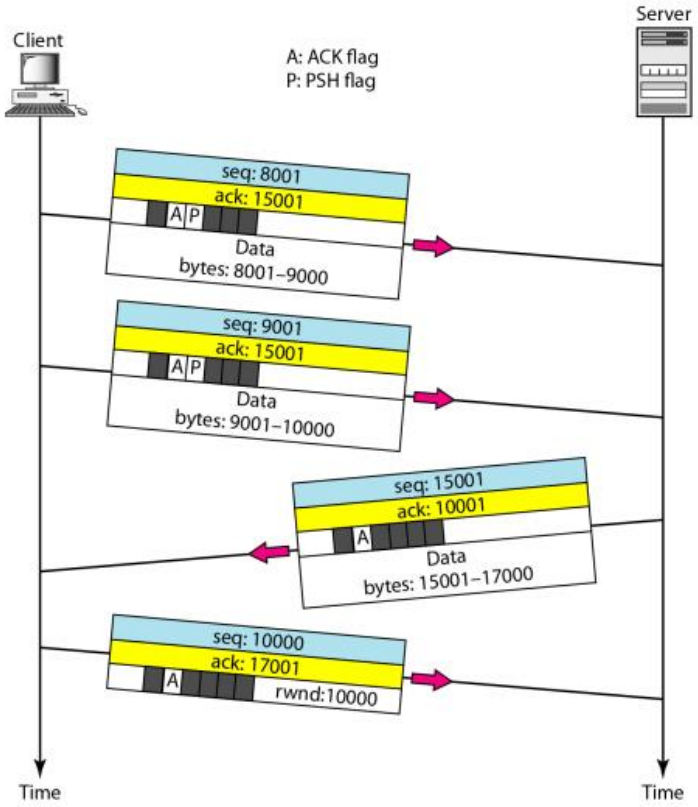
2. The server sends the second segment, a SYN + ACK segment, with 2 flag bits set: SYN and ACK. This segment has a dual purpose. It is a SYN segment for commu nication in the other direction and serves as the acknowledgment for the SYN segment. It consumes one sequence number. A SYN + ACK segment cannot carry data, but does consume 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 sequence number in this segment is the same as the one in the SYN segment; the ACK segment does not consume any sequence numbers. An ACK segment, if carrying no data, consumes no sequence number

2. Data Transfer

After connection is established, bidirectional data transfer can take place. The client and server can both send data and acknowledgments.

Figure ***Data transfer***



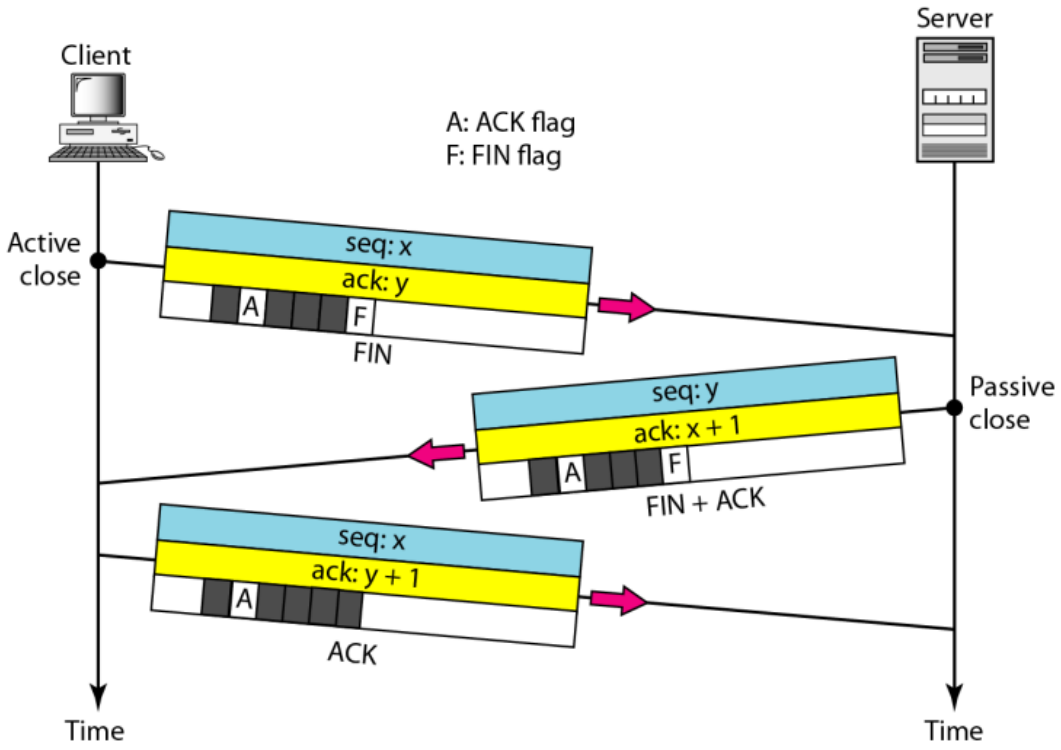
3. Connection Termination

Any of the two parties involved in exchanging data (client or server) can close the connec tion, 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.

Figure *Connection termination using three-way handshaking*



1. In a normal 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.

2. 2. The server TCP, after receiving the FIN segment, informs its process of the situa tion 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, it consumes only one sequence number.

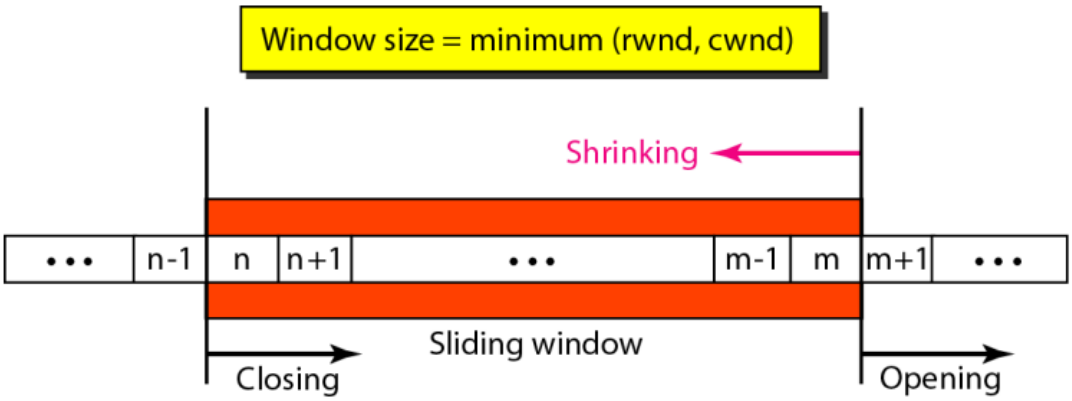
3. 3. The client TCP sends the last segment, an ACK segment, to confirm the receipt of the FIN segment from the TCP server. This segment contains the acknowledgment number, which is 1 plus the sequence number received in the FIN segment from the server. This segment cannot carry data and consumes no sequence numbers.

Flow Control

TCP uses a sliding window, to handle flow control. The sliding window protocol used by TCP, however, is something between the Go-Back-N and Selective Repeat sliding window. The sliding window protocol in TCP looks like the Go-Back-N protocol because

it does not use NAKs; it looks like Selective Repeat because the receiver holds the out-of-order segments until the missing ones arrive.

Figure ***Sliding window***



The window spans a portion of the buffer containing bytes received from the process. The bytes inside the window are the bytes that can be in transit; they can be sent without worrying about acknowledgment. The imaginary window has two walls: one left and one right.

The window is opened, closed, or shrunk. These three activities, are in the control of the receiver (and depend on congestion in the network), not the sender. The sender must obey the commands of the receiver in this matter.

- Opening a window means moving the right wall to the right. This allows more new bytes in the buffer that are eligible for sending.
- Closing the window means moving the left wall to the right. This means that some bytes have been acknowledged and the sender need not worry about them anymore.
- Shrinking the window means moving the right wall to the left. This is strongly discouraged and not allowed in some implementations because it means revoking the eligibility of some bytes for sending.

The size of the window at one end is determined by the lesser of two values: receiver window (rwnd) or congestion window (cwnd). The receiver window is the value advertised by the opposite end in a segment containing acknowledgment. It is the number of bytes the other end can accept before its buffer overflows and data are discarded. The congestion window is a value determined by the network to avoid congestion.

Error Control

TCP provides reliability using error control. Error control includes mechanisms for detecting corrupted segments, lost segments, out-of-order segments, and duplicated segments. Error control also includes a mechanism for correcting errors after they are detected. Error detection and correction in TCP is achieved through the use of three simple tools:

- checksum,
- acknowledgment, and
- time-out.

Checksum

Each segment includes a checksum field which is used to check for a corrupted segment. If the segment is corrupted, it is discarded by the destination TCP and is considered as lost. TCP uses a 16-bit checksum that is mandatory in every segment.

Acknowledgment

TCP uses acknowledgments to confirm the receipt of data segments. Control segments that carry no data but consume a sequence number are also acknowledged. ACK segments are never acknowledged.

Retransmission

The heart of the error control mechanism is the retransmission of segments. When a segment is corrupted, lost, or delayed, it is retransmitted. In modern implementations, a segment is retransmitted on two occasions: when a retransmission timer expires or when the sender receives three duplicate ACKs.

Retransmission After RTO

A recent implementation of TCP maintains one retrans mission time-out (RTO) timer for all outstanding (sent, but not acknowledged) seg ments. When the timer matures, the earliest outstanding segment is retransmitted even though lack of a received ACK can be due to a delayed segment, a delayed ACK, or a lost acknowledgment.

The value of RTO is dynamic in TCP and is updated based on the round-trip time (RTT) of segments. An RTI is the time needed for a segment to reach a destination and for an acknowledgment to be received

Retransmission After Three Duplicate ACK Segments

The previous rule about retransmission of a segment is sufficient ifthe value ofRTO is not very large. Sometimes, however, one segment is lost and the receiver receives so many out-of-order segments that they cannot be saved (limited buffer size). To alleviate this situation,

most imple mentations today follow the three-duplicate-ACKs rule and retransmit the missing segment immediately. This feature is referred to as fast retransmission

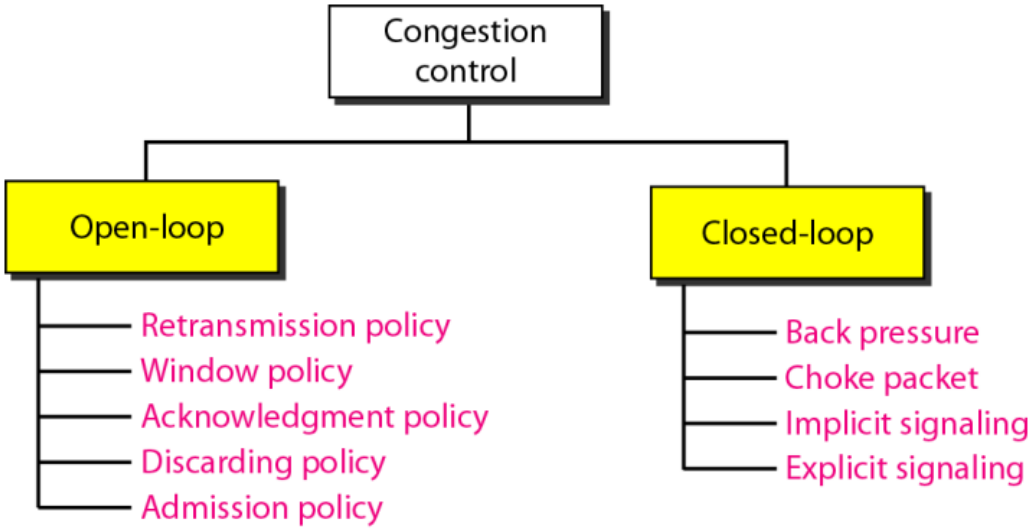
Out-of-Order Segments

When a segment is delayed, lost, or discarded, the segments following that segment arrive out of order. Originally, TCP was designed to discard all out-of-order segments, resulting in the retransmission of the missing segment and the following segments

CONGESTION CONTROL

Congestion control refers to techniques and mechanisms that can either prevent conges tion, before it happens, or remove congestion, after it has happened. In general, we can divide congestion control mechanisms into two broad categories: open-loop congestion control (prevention) and closed-loop congestion control (removal).

Figure ***Congestion control categories***



Open-Loop Congestion Control

In open-loop congestion control, policies are applied to prevent congestion before it happens. In these mechanisms, congestion control is handled by either the source or the destination.

Retransmission Policy

Retransmission is sometimes unavoidable. If the sender feels that a sent packet is lost or corrupted, the packet needs to be retransmitted. Retransmission in general may increase congestion in the network. However, a good retransmission policy can prevent congestion. The retransmission policy and the retransmission timers must be designed to optimize efficiency and at the same time prevent congestion.

Window Policy

The type of window at the sender may also affect congestion. The Selective Repeat window is better than the Go-Back-N window for congestion control. In the Go-Back-N window, when the timer for a packet times out, several packets may be resent, although some may have arrived safe and sound at the receiver. This duplication may make the congestion worse. The Selective Repeat window, on the other hand, tries to send the specific packets that have been lost or corrupted.

Acknowledgment Policy

The acknowledgment policy imposed by the receiver may also affect congestion. If the receiver does not acknowledge every packet it receives, it may slow down the sender and help prevent congestion. A receiver may send an acknowledgment only if it has a packet to be sent or a special timer expires. A receiver may decide to acknowledge only N packets at a time.

Discarding Policy

A good discarding policy by the routers may prevent congestion and at the same time may not harm the integrity of the transmission.

Admission Policy

An admission policy, which is a quality-of-service mechanism, can also prevent conges tion in virtual-circuit networks. Switches in a flow first check the resource requirement of a flow before admitting it to the network. A router can deny establishing a virtual circuit connection if there is congestion in the network or ifthere is a possibility offuture congestion.

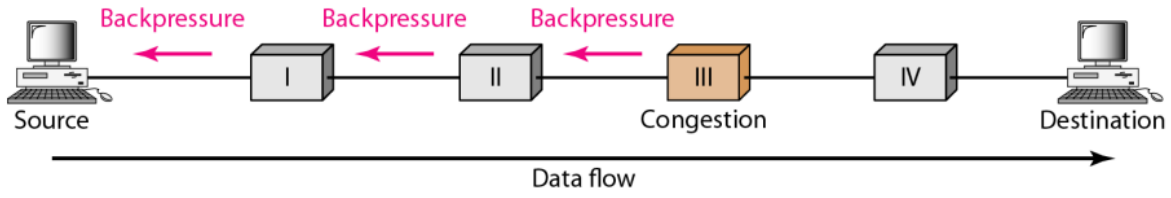
Closed-Loop Congestion Control

Closed-loop congestion control mechanisms try to alleviate congestion after it happens.

Backpressure

The technique of backpressure refers to a congestion control mechanism in which a congested node stops receiving data from the immediate upstream node or nodes. This may cause the upstream node or nodes to become congested, and they, in turn, reject data from their upstream nodes or nodes. And so on. Backpressure is a node-to-node congestion control that starts with a node and propagates, in the opposite direction of data flow, to the source.

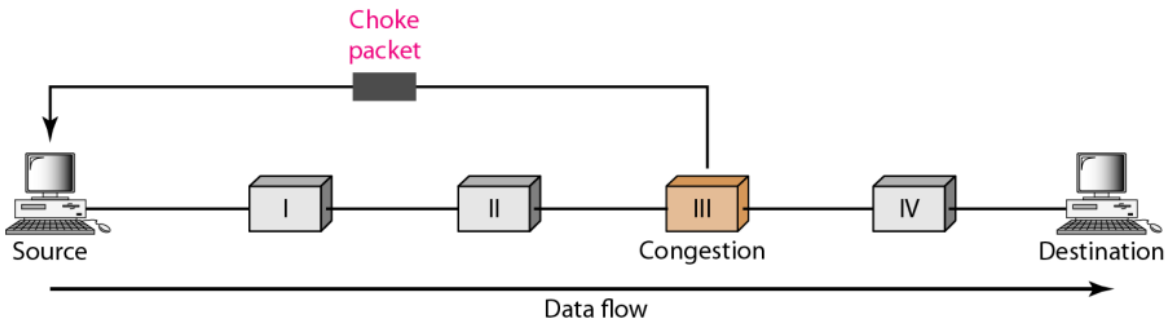
Figure *Backpressure method for alleviating congestion*



Choke Packet

A choke packet is a packet sent by a node to the source to inform it of congestion. In the choke packet method, the warning is from the router, which has encountered congestion, to the source station directly. The intermediate nodes through which the packet has traveled are not warned.

Figure *Choke packet*



Implicit Signaling

In implicit signaling, there is no communication between the congested node or nodes and the source. The source guesses that there is a congestion somewhere in the network from other symptoms. For example, when a source sends several packets and there is no acknowledgment for a while, one assumption is that the network is congested. The delay in

receiving an acknowledgment is interpreted as congestion in the network; the source should slow down.

Explicit Signaling

The node that experiences congestion can explicitly send a signal to the source or destination.

Backward Signaling

A bit can be set in a packet moving in the direction opposite to the congestion. This bit can warn the source that there is congestion and that it needs to slow down to avoid the discarding of packets.

Forward Signaling

A bit can be set in a packet moving in the direction of the congestion. This bit can warn the destination that there is congestion. The receiver in this case can use policies, such as slowing down the acknowledgments, to alleviate the congestion.

Congestion Control in TCP

Congestion Window

the sender's window size is determined not only by the receiver but also by congestion in the network. The sender has two pieces of information: the receiver-advertised window size and the congestion window size. The actual size of the window is the minimum of these two.

Actual window size=minimum (rwnd, cwnd)

Congestion Policy

TCP's general policy for handling congestion is based on three phases: slow start, congestion avoidance, and congestion detection. In the slow-start phase, the sender starts with a very slow rate of transmission, but increases the rate rapidly to reach a threshold. When the threshold is reached, the data rate is reduced to avoid congestion. Finally if congestion is detected, the sender goes back to the slow-start or congestion avoidance phase based on how the congestion is detected.

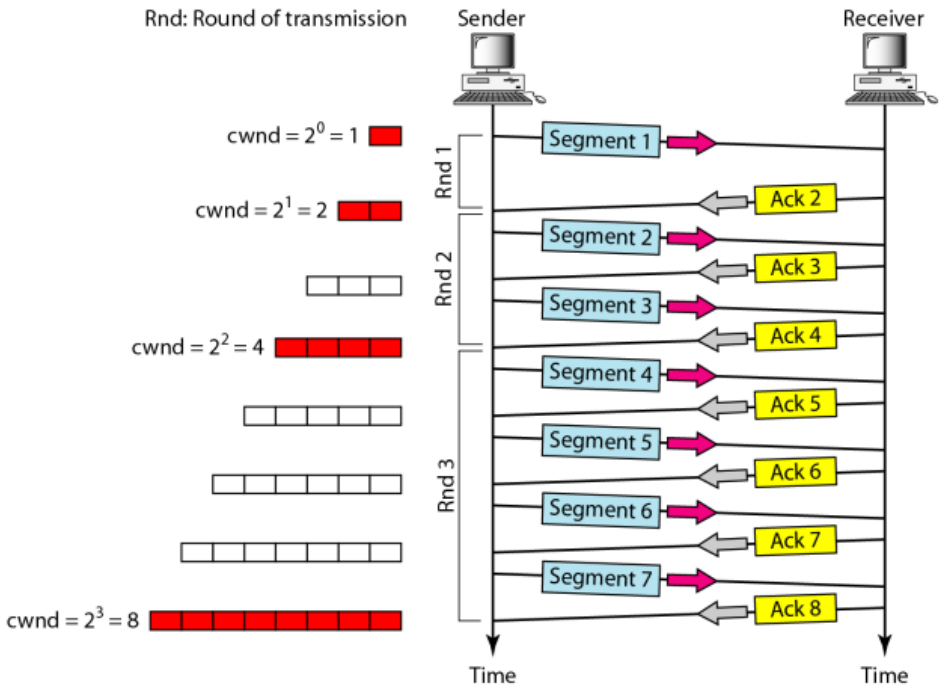
Slow Start: Exponential Increase

This algorithm is based on the idea that the size of the con gestion window (cwnd) starts with one maximum segment size (MSS). The MSS is determined during connection establishment by using an option of the same name. The size of the window increases one MSS each time an acknowledgment is received.

The sender starts with cwnd =1 MSS. This means that the sender can send only one segment. After receipt of the acknowledgment for segment 1, the size of the congestion window is increased by 1, which means that cwnd is now 2. Now two more segments can be sent. When

each acknowledgment is received, the size of the window is increased by 1 MSS. When all seven segments are acknowledged, cwnd = 8.

Figure ***Slow start, exponential increase***

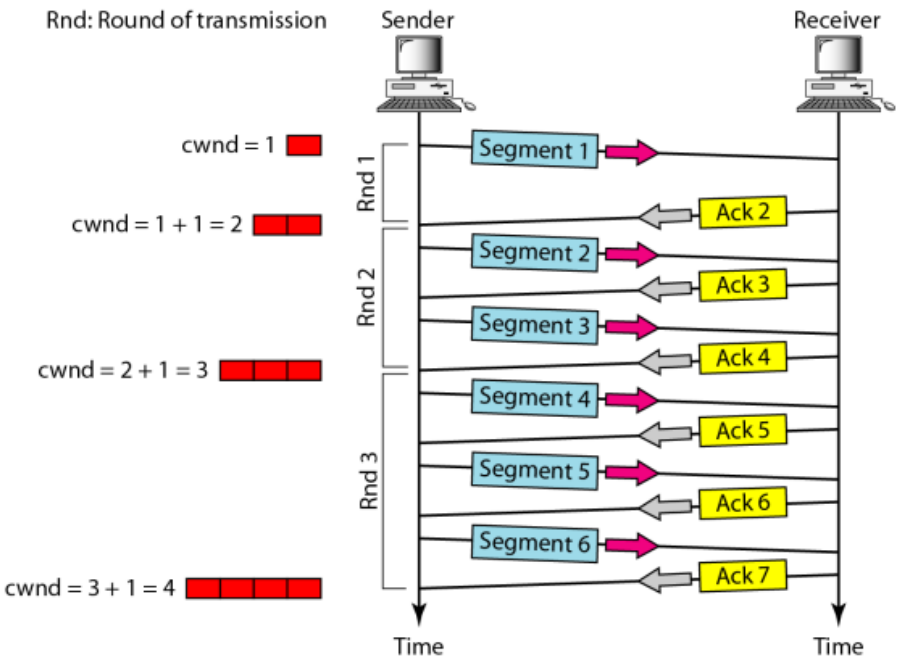


Slow start cannot continue indefinitely. There must be a threshold to stop this phase. The sender keeps track of a variable named ssthresh (slow-start threshold). When the size of window in bytes reaches this threshold, slow start stops and the next phase starts. In most implementations the value ofssthresh is 65,535 bytes.

Congestion Avoidance: Additive Increase

If we start with the slow-start algorithm, the size of the congestion window increases exponentially. To avoid congestion before it happens, one must slow down this exponential growth. TCP defines another algo rithm called congestion avoidance, which undergoes an additive increase instead of an exponential one. When the size of the congestion window reaches the slow-start threshold, the slow-start phase stops and the additive phase begins. In this algorithm, each time the whole window of segments is acknowledged (one round), the size of the congestion window is increased by 1.

Figure ***Congestion avoidance, additive increase***



Start	➡	*cwnd=1*
After round 1	➡	*cwnd*= 1+1=2
After round 2	➡	*cwnd=2*+ 1=3
After round 3	➡	*cwnd=3*+ 1=4

In the congestion avoidance algorithm, the size of the congestion window increases additively until congestion is detected.

Congestion Detection: Multiplicative Decrease

If congestion occurs, the congestion window size must be decreased. The only way the sender can guess that congestion has occurred is by the need to retransmit a segment. However, retransmission can occur in one of two cases: when a timer times out or when three ACKs are received. In both cases, the size of the threshold is dropped to one-half, a multiplicative decrease. Most TCP implementations have two reactions:

I. If a time-out occurs, there is a stronger possibility of congestion; a segment has probably been dropped in the network, and there is no news about the sent segments.

II. In this case TCP reacts strongly: a. It sets the value of the threshold to one-half of the current window size. b. It sets cwnd to the size of one segment. c. It starts the

slow-start phase again. 2. If three ACKs are received, there is a weaker possibility of congestion; a segment may have been dropped, but some segments after that may have arrived safely since three ACKs are received. This is called fast transmission and fast recovery. In this case, TCP has a weaker reaction: a. It sets the value of the threshold to one-half of the current window size. b. It sets cwnd to the value of the threshold (some implementations add three segment sizes to the threshold). c. It starts the congestion avoidance phase.

SCTP

Stream Control Transmission Protocol (SCTP) is a new reliable, message-oriented transport layer protocol. SCTP combines the best features of UDP and TCP. SCTP is a reliable message oriented protocol. It preserves the message boundaries and at the same time detects lost data, duplicate data, and out-of-order data. It also has congestion control and flow control mechanisms.

SCTP Services

1. Process-to-Process Communication

SCTP uses all well-known ports in the TCP space.

Table ***Some SCTP applications***

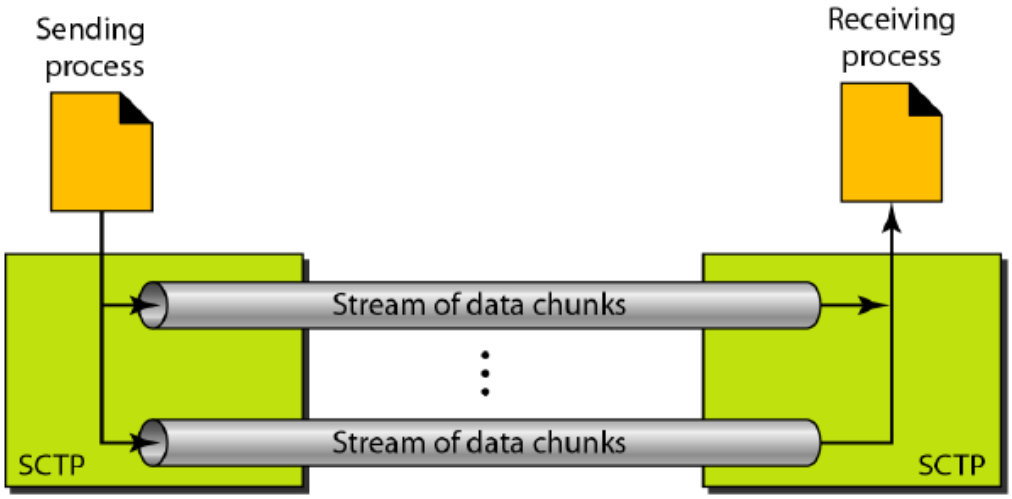
Protocol	*Port Number*	*Description*
IUA	9990	ISDN over IP
M2UA	2904	SS7 telephony signaling
M3UA	2905	SS7 telephony signaling
H.248	2945	Media gateway control
H.323	1718, 1719, 1720, 11720	IP telephony
SIP	5060	IP telephony

---

2. Multiple Streams

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

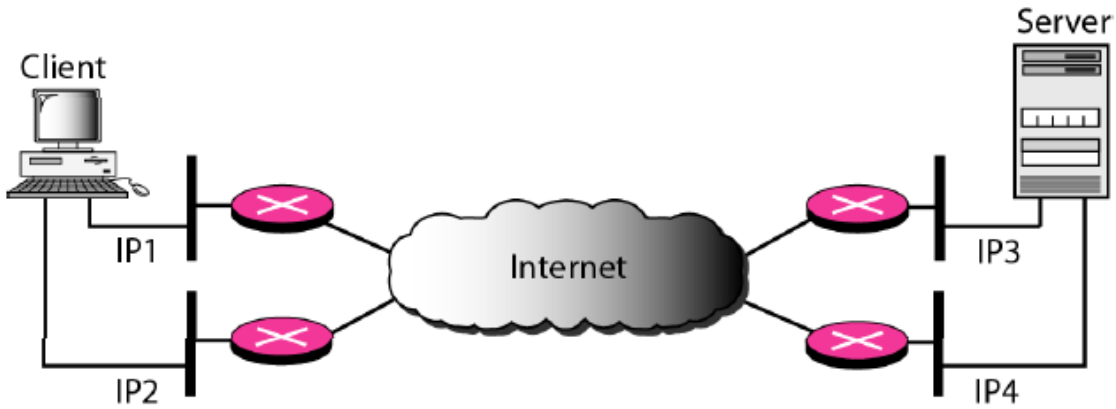
Figure *Multiple-stream concept*



3. *Multihoming*

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

Figure *Multihoming concept*



4. *Full-Duplex Communication*

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

5. Connection-Oriented Service

Like TCP, SCTP is a connection-oriented protocol. However, in SCTP, a connection is called an association. When a process at site A wants to send and receive data from another process at site B, the following occurs:

1. The two SCTPs establish an association between each other.

2. Data are exchanged in both directions.

3. The association is terminated.

6. Reliable Service

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

SCTP Features

1. Transmission Sequence Number

The unit of data in TCP is a byte. Data transfer in TCP is controlled by numbering bytes by using a sequence number. On the other hand, the unit of data in SCTP is a DATA chunk which mayor may not have a one-to-one relationship with the message coming from the 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. 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 In TCP, there is only one stream in each connection.

2. Stream Identifier (SI)

In SCTP, there may be several streams in each association. Each stream in SCTP needs to be identified by 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.

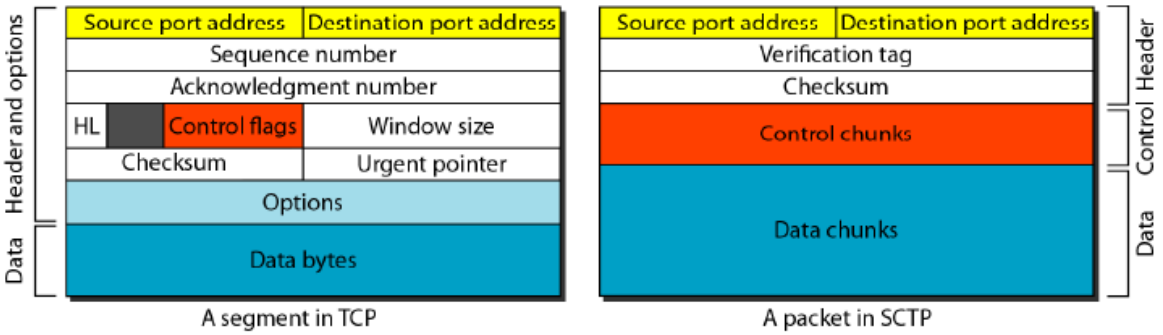
3. Stream Sequence Number

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).

4. *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 is carried as control chunks. Several control chunks and data chunks can be packed together in a packet.

Figure *Comparison between a TCP segment and an SCTP packet*



1. The control information in TCP is part of the header; the control information in SCTP is included in the control chunks.
2. The data in a TCP segment treated as one entity; an SCTP packet can carry several data chunks; each can belong to a different stream.
3. The options section, which can be part of a TCP segment, does not exist in an SCTP packet. Options in SCTP are handled by defining new chunk types.
4. The mandatory part of the TCP header is 20 bytes, while the general header in SCTP is only 12 bytes. The SCTP header is shorter due to the following:
    a. An SCTP sequence number (TSN) belongs to each data chunk and hence is located in the chunk's header.
    b. The acknowledgment number and window size are part of each control chunk.
    c. There is no need for a header length field (shown as HL in the TCP segment) because there are no options to make the length of the header variable; the SCTP header length is fixed (12 bytes).

d. There is no need for an urgent pointer in SCTP.

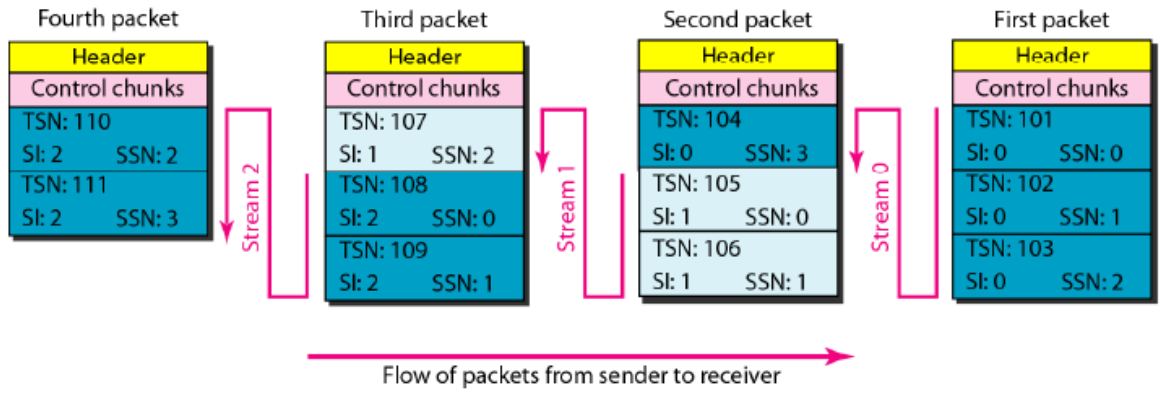
5. The checksum in TCP is 16 bits; in SCTP, it is 32 bits.

    a. The verification tag in SCTP is an association identifier, which does not exist in TCP. A unique verification tag is needed to define each association.

    b. TCP includes one sequence number in the header, which defines the number of the first byte in the data section. An SCTP packet can include several different data chunks. TSNs, SIs, and SSNs define each data chunk.

    c. Some segments in TCP that carry control information (such as SYN and FIN) need to consume one sequence number; control chunks in SCTP never use a TSN, SI, or SSN. These three identifiers belong only to data chunks, not to the whole packet.

Figure ***Packet, data chunks, and streams***



Data chunks are identified by three items: TSN, SI, and SSN.

- TSN is a cumulative number identifying the association;
- SI defines the stream;
- SSN defines the chunk in a stream.

5. 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. In SCTP, however, the control information is carried by control chunks, which do not need a TSN. These control chunks are acknowledged by another control chunk of the appropriate type (some need no

acknowledgment). For example, an INIT control chunk is acknowledged by an INIT ACK chunk. There is no need for a sequence number or an acknowledgment number.

6. Flow Control

Like TCP, SCTP implements flow control to avoid overwhelming the receiver.

7. Error Control

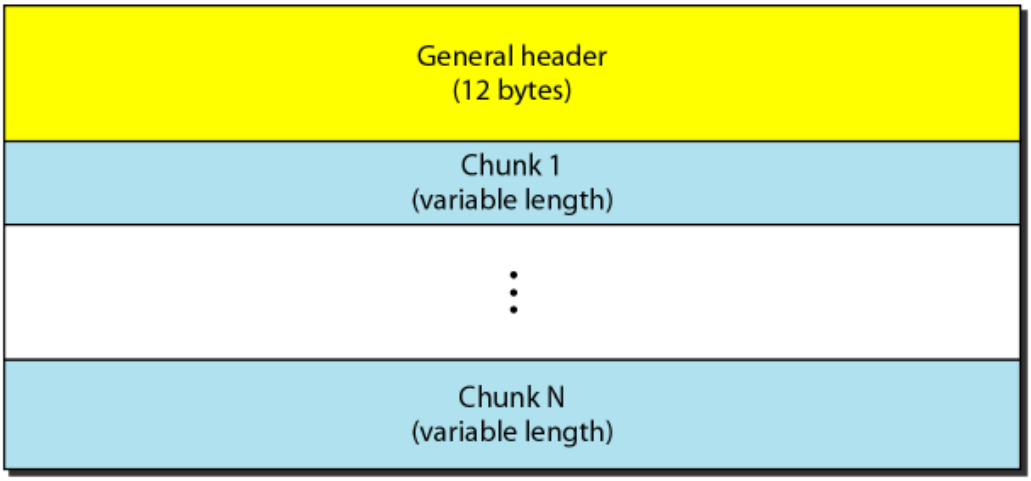
Like TCP, SCTP implements error control to provide reliability. TSN numbers and acknowledgment numbers are used for error control.

8. Congestion Control

Like TCP, SCTP implements congestion control to determine how many data chunks can be injected into the network.

Packet Format

Figure ***SCTP packet format***

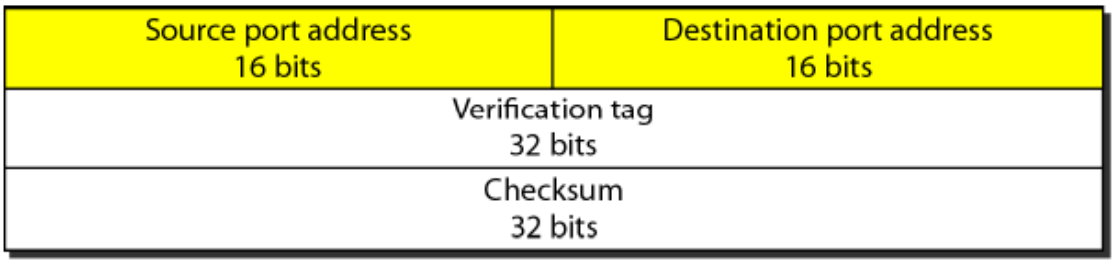


An SCTP packet has a mandatory general header and a set of blocks called chunks. There are two types of chunks: control chunks and data chunks. A control chunk controls and maintains the association; a data chunk carries user data. In a packet, the control chunks come before the data chunks.

- ***General Header***

The general header (packet header) defines the endpoints 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.

Figure *General header*



There are four fields in the general header:

- Source port address. This is a 16-bit field that defines the port number of the process sending the packet.
- Destination port address. This is a 16-bit field that defines the port number of the process receiving the packet.
- Verification tag. This is a number that matches a packet to an association.
- Checksum. This 32-bit field contains a CRC-32 checksum.

Chunks

Control information or user data are carried in chunks.

Table ***Chunks***

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 cumulative TSN

An SCTP Association

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

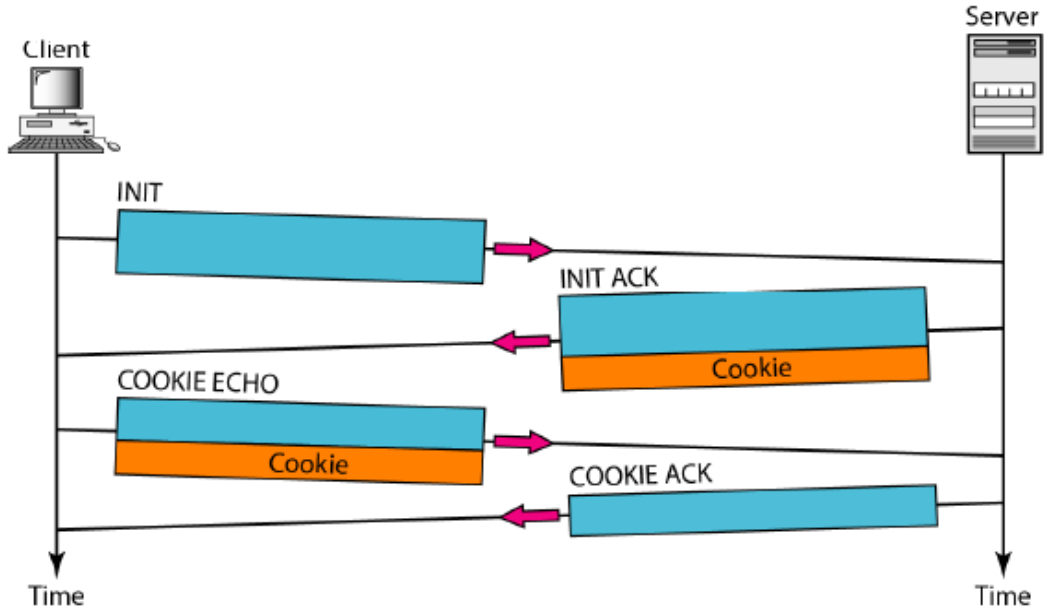
Association Establishment

Association establishment in SCTP requires a four-way handshake.

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

1. The client sends the first packet, which contains an INIT chunk.
2. The server sends the second packet, which contains an INIT ACK chunk.
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 COOKIE ECHO chunk. SCTP allows the inclusion of data chunks with this packet.

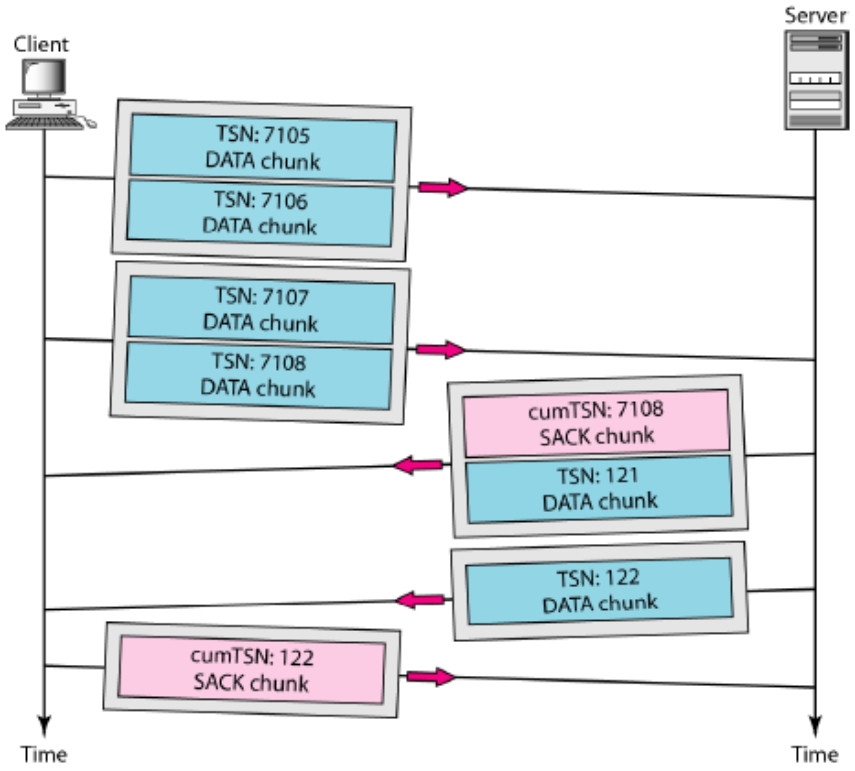
Figure *Four-way handshaking*



Data Transfer

After the association is established, bidirectional data transfer can take place. The client and the server can both send data. 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 is fragmented. In SCTP, only DATA chunks consume TSNs; DATA chunks are the only chunks that are acknowledged.

Figure *Simple data transfer*

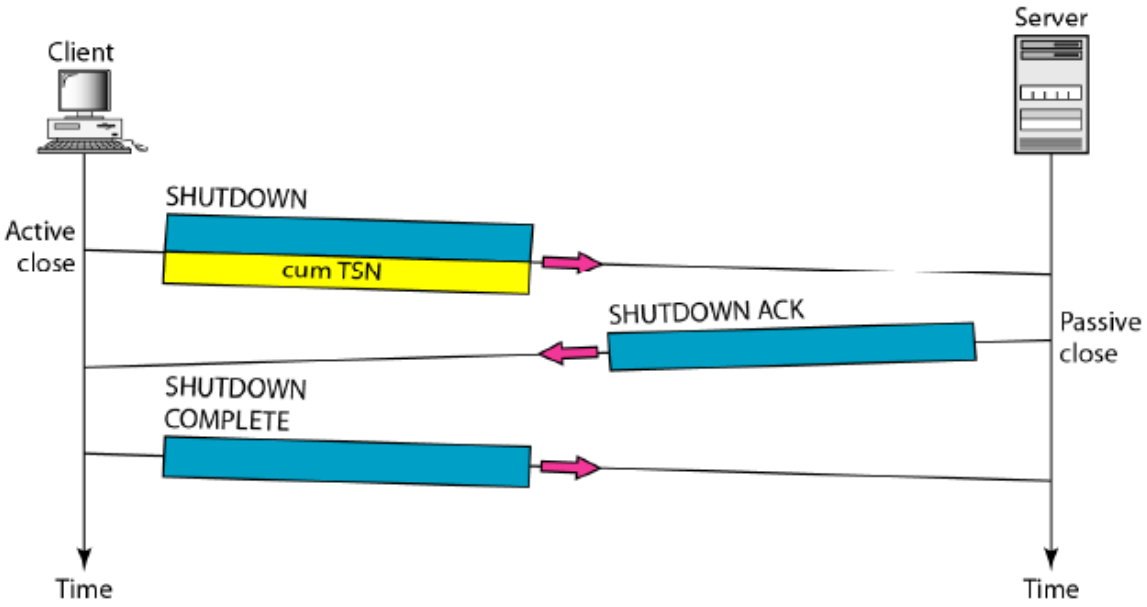


1. The client sends the first packet carrying two DATA chunks with TSNs 7105 and 7106.
2. The client sends the second packet carrying two DATA chunks with TSNs 7107 and 7108.
3. The third packet is from the server. It contains the SACK chunk needed to acknowledge the receipt of DATA chunks from the client. Contrary to TCP, SCTP acknowledges the last in-order TSN received, not the next expected. The third packet also includes the first DATA chunk from the server with TSN 121.
4. After a while, the server sends another packet carrying the last DATA chunk with TSN 122, but it does not include a SACK chunk in the packet because the last DATA chunk received from the client was already acknowledged.
5. Finally, the client sends a packet that contains a SACK chunk acknowledging the receipt of the last two DATA chunks from the server.

Association Termination

In SCTP, like TCP, either of the two parties involved in exchanging data (client or server) can close the connection. 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.

Figure *Association termination*



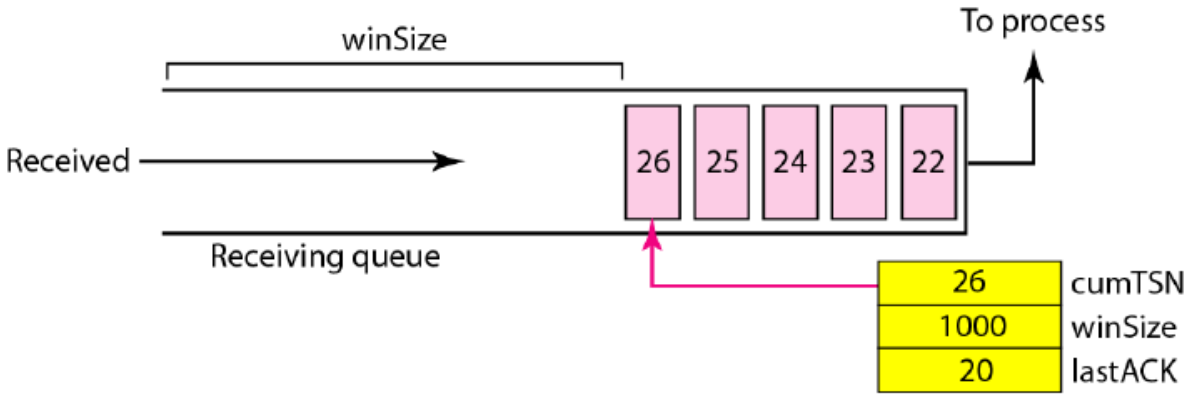
Flow Control

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 by the process. The first variable holds the last TSN received, *cumTSN*. The second variable holds the available buffer size, *winsize. The third variable holds the last accumulative acknowledgment, lastACK.*

Figure *Flow control, receiver site*

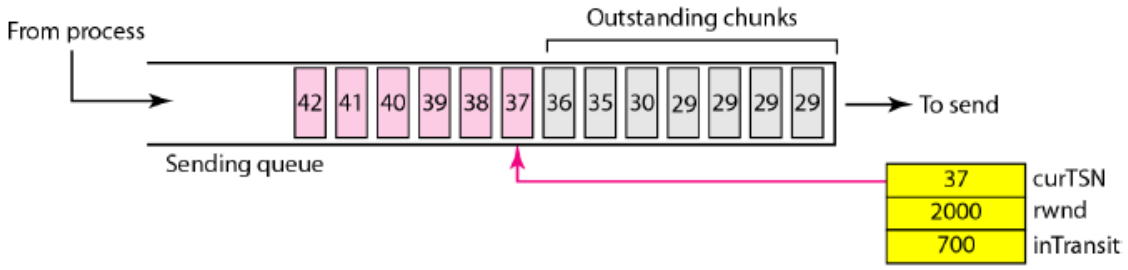


1. When the site receives a data chunk, it stores it at the end of the buffer (queue) and subtracts the size of the chunk from *winSize.* The TSN number of the chunk is stored in the *cumTSN* variable.
2. When the process reads a chunk, it removes it from the queue and adds the size of the removed chunk 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.

- ***Sender Site***

The sender has one buffer (queue) and three variables: *curTSN, rwnd,* and *inTransit*

Figure *Flow control, sender site*



The buffer holds the chunks produced by the process that either have been sent or are ready to be sent. The first variable, *curTSN*, refers to the next chunk to be sent. All chunks in the queue with a TSN less than this value have been sent, but not acknowledged; they are outstanding. The second variable, *rwnd*, holds the last value advertised by the receiver (in bytes). The third variable, *inTransit*, holds the number of bytes in transit, bytes sent but not yet acknowledged.

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 1 and now points to the next chunk to be sent. The value of *inTransit* is incremented by the size of the data in the transmitted chunk.

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 any more. The value of *inTransit* is reduced by the total size of the discarded chunks. The value of *rwnd* is updated with the value of the advertised window in the SACK.

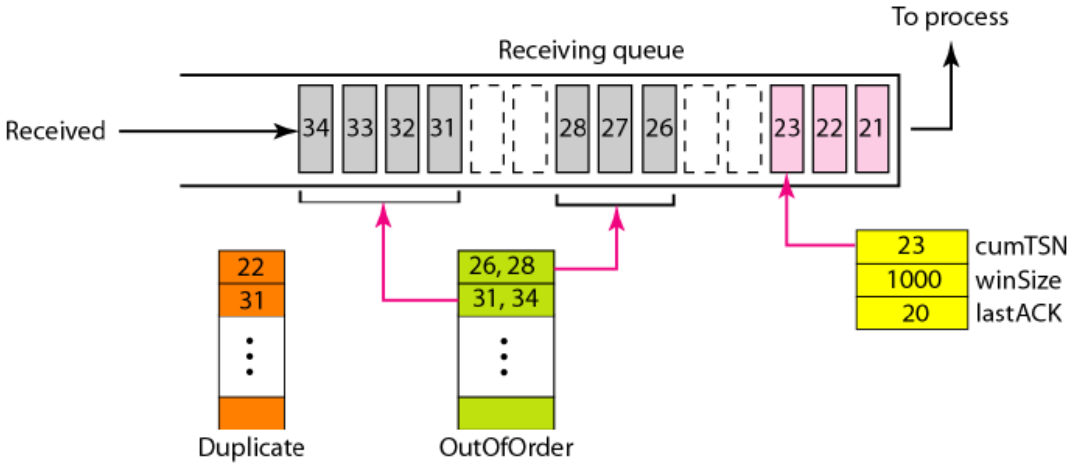
Error Control

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.

- ***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.

Figure *Error control, receiver site*



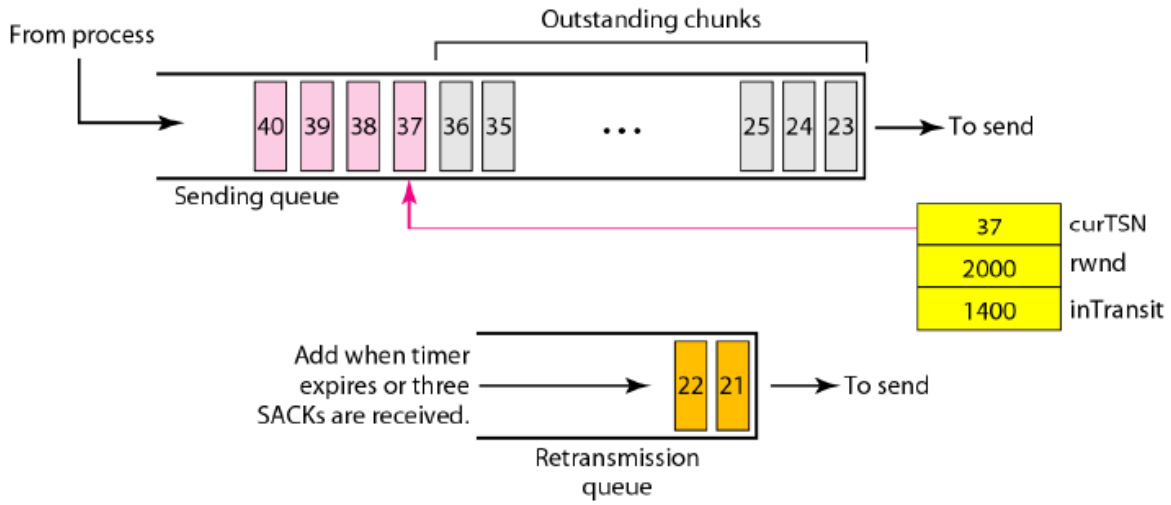
The last acknowledgment sent was for data chunk 20. The available window size is 1000 bytes. Chunks 21 to 23 have been received in order. The first out-of-order block contains chunks 26 to 28. The second out-of-order block contains chunks 31 to 34. A variable holds

the value of *cumTSN*. An array of variables keeps track of the beginning and the end of each block that is out of order. An array of variables holds the duplicate chunks received. Note that there is no need for storing duplicate chunks in the queue; they will be discarded.

- ***Sender Site***

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

Figure *Error control, sender site*



The sending queue holds chunks 23 to 40. The chunks 23 to 36 have already been sent, but not acknowledged; they are outstanding chunks. The *curTSN* points to the next chunk to be sent (37). We assume that each chunk is 100 bytes, which means that 1400 bytes of data (chunks 23 to 36) is in transit. The sender at this moment has a retransmission queue. When a packet is sent, a retransmission timer starts for that packet (all data chunks in that packet). Some implementations use one single timer for the entire association, but we continue with our tradition of one timer for each packet for simplification. When the retransmission timer for a packet expires, or four duplicate SACKs arrive that declare a packet as missing, the chunks in that packet are moved to the retransmission queue to be resent. These chunks are considered lost, rather than outstanding. The chunks in the retransmission queue have priority. In other words, the next time the sender sends a chunk, it would be chunk 21 from the retransmission queue.

Congestion Control

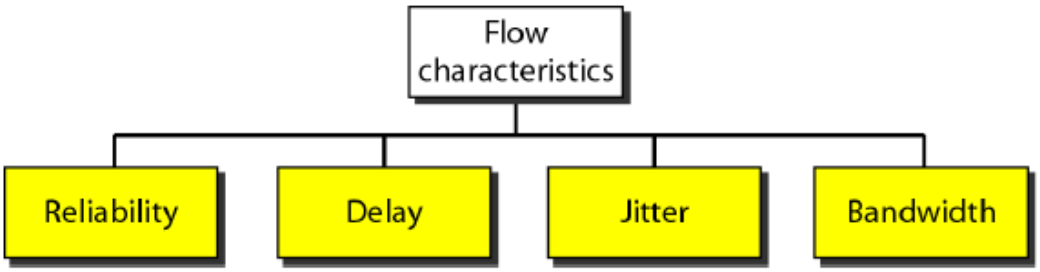
SCTP, like TCP, is a transport layer protocol with packets subject to congestion in the network. The SCTP designers have used the same strategies for congestion control for TCP. SCTP has slow start (exponential increase), congestion avoidance (additive increase), and congestion detection (multiplicative decrease) phases. Like TCP, SCTP also uses fast retransmission and fast recovery.

QUALITY OF SERVICE

Flow Characteristics

Traditionally, four types of characteristics are attributed to a flow: reliability, delay, jitter, and bandwidth.

Figure *Flow characteristics*



Reliability

Reliability is a characteristic that a flow needs. Lack of reliability means losing a packet or acknowledgment, which entails retransmission.

Jitter

Jitter is the variation in delay for packets belonging to the same flow. Jitter is defined as the variation in the packet delay. High jitter means the difference between delays is large; low jitter means the variation is small.

Bandwidth

Different applications need different bandwidths. In video conferencing we need to send millions of bits per second to refresh a color screen while the total number of bits in an e-mail may not reach even a million.

TECHNIQUES TO IMPROVE QoS

1. scheduling,
2. traffic shaping,
3. admission control, and
4. resource reservation.

1. Scheduling

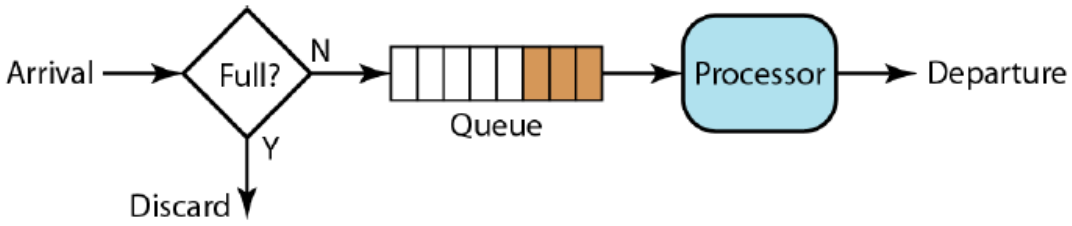
Packets from different flows arrive at a switch or router for processing. A good scheduling technique treats the different flows in a fair and appropriate manner.

- FIFO queuing,
- priority queuing, and
- weighted fair queuing

- ***FIFO Queuing***

In first-in, first-out (FIFO) queuing, packets wait in a buffer (queue) until the node (router or switch) is ready to process them. If the average arrival rate is higher than the average processing rate, the queue will fill up and new packets will be discarded. A FIFO queue is familiar to those who have had to wait for a bus at a bus stop.

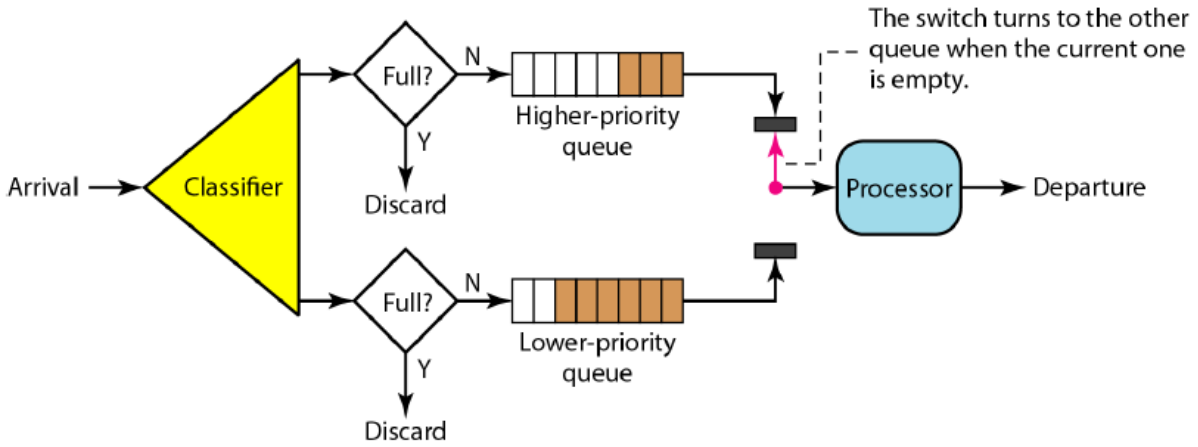
Figure ***FIFO queue***



- ***Priority Queuing***

In priority queuing, packets are first assigned to a priority class. Each priority class has its own queue. The packets in the highest-priority queue are processed first. Packets in the lowest-priority queue are processed last.

Figure ***Priority queuing***

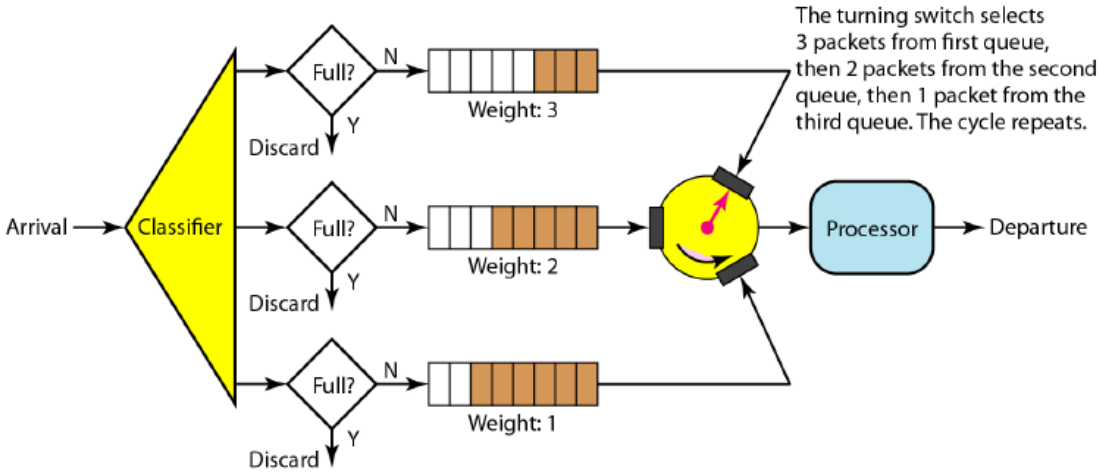


A priority queue can provide better QoS than the FIFO queue because higher priority traffic, such as multimedia, can reach the destination with less delay. However, there is a potential drawback. If there is a continuous flow in a high-priority queue, the packets in the lower-priority queues will never have a chance to be processed. This is a condition called *starvation.*

- ***Weighted Fair Queuing***

A better scheduling method is weighted fair queuing. In this technique, the packets are still assigned to different classes and admitted to different queues. The queues, however, are weighted based on the priority of the queues; higher priority means a higher weight. The system processes packets in each queue in a round-robin fashion with the number of packets selected from each queue based on the corresponding weight.

Figure ***Weighted fair queuing***



2. Traffic Shaping

Traffic shaping is a mechanism to control the amount and the rate of the traffic sent to the network. Two techniques can shape traffic:

- leaky bucket and
- token bucket.

- ***Leaky Bucket***

If a bucket has a small hole at the bottom, the water leaks from the bucket at a constant rate as long as there is water in the bucket. The rate at which the water leaks does not depend on the rate at which the water is input to the bucket unless the bucket is empty. The input rate can vary, but the output rate remains constant. Similarly, in networking, a technique called leaky bucket can smooth out bursty traffic. Bursty chunks are stored in the bucket and sent out at an average rate.

Figure *Leaky bucket*

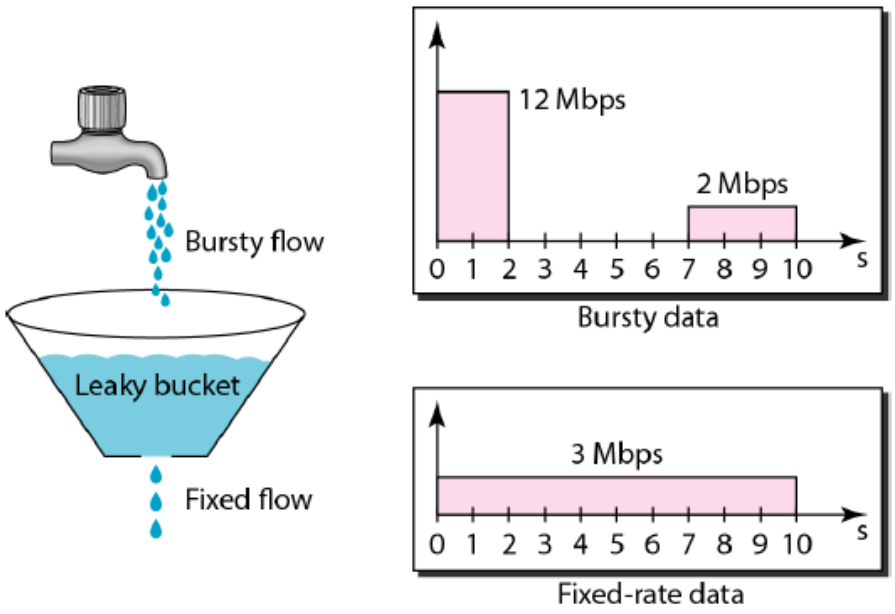
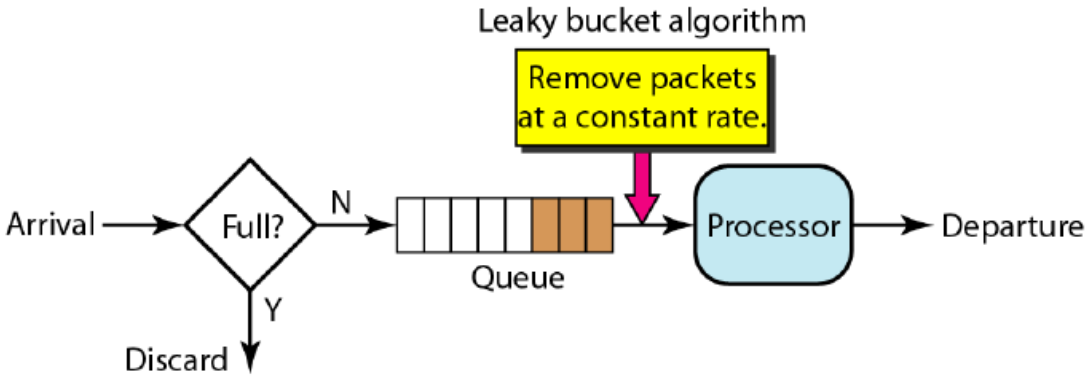


Figure *Leaky bucket implementation*



A FIFO queue holds the packets. If the traffic consists of fixed-size packets the process removes a fixed number of packets from the queue at each tick of the clock. If the traffic consists of variable-length packets, the fixed output rate must be based on the number of bytes or bits.

The following is an algorithm for variable-length packets:

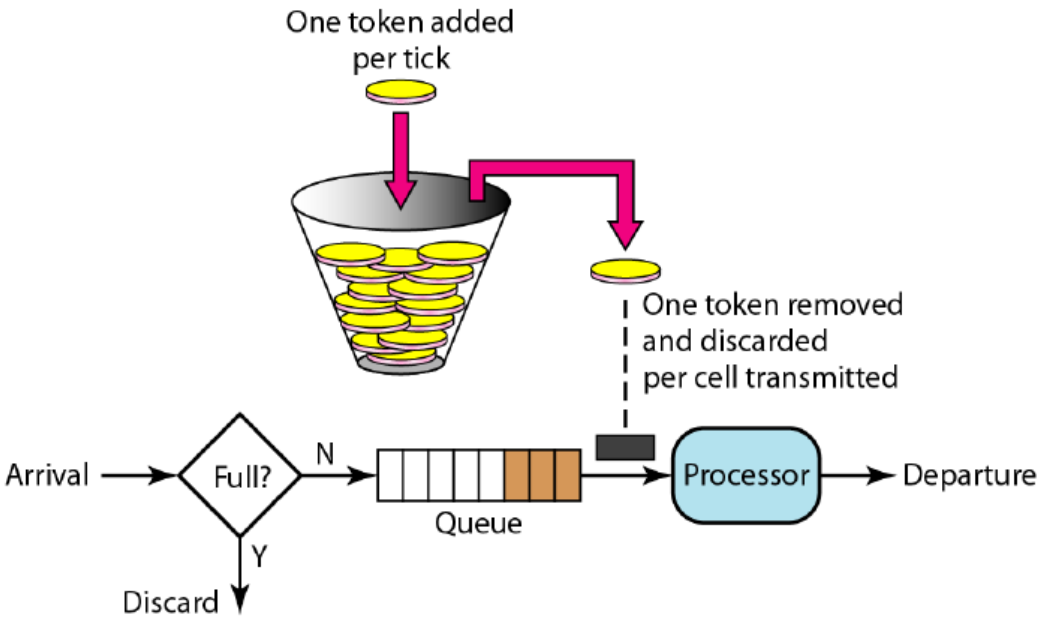
1. Initialize a counter to *n* at the tick of the clock.
2. If *n* is greater than the size of the packet, send the packet and decrement the counter by the packet size. Repeat this step until *n* is smaller than the packet size.
3. Reset the counter and go to step 1.

- ***Token Bucket***

The leaky bucket is very restrictive. It does not credit an idle host. For example, if a host is not sending for a while, its bucket becomes empty. Now if the host has bursty data, the leaky bucket allows only an average rate. The time when the host was idle is not taken into account. On the other hand, the token bucket algorithm allows idle hosts to accumulate credit for the future in the form of tokens. For each tick of the clock, the system sends *n* tokens to the bucket. The system removes one token for every cell (or byte) of data sent. For example, if *n* is 100 and the host is idle for 100 ticks, the bucket collects 10,000 tokens. Now the host can consume all these tokens in one tick with 10,000 cells, or the host takes 1000 ticks with 10 cells per tick. In other words, the host can send bursty data as long as the bucket is not empty.

A leaky bucket algorithm shapes bursty traffic into fixed-rate traffic by averaging the data rate. It may drop the packets if the bucket is full. The token bucket allows bursty traffic at a regulated maximum rate.

Figure ***Token bucket***



- ***Combining Token Bucket and Leaky Bucket***

The two techniques can be combined to credit an idle host and at the same time regulate the traffic. The leaky bucket is applied after the token bucket; the rate of the leaky bucket needs to be higher than the rate of tokens dropped in the bucket.

3. Admission Control

Admission control refers to the mechanism used by a router, or a switch, to accept or reject a flow based on predefined parameters called flow specifications. Before a router accepts a flow for processing, it checks the flow specifications to see if its capacity (in terms of bandwidth, buffer size, CPU speed, etc.) and its previous commitments to other flows can handle the new flow.

4. Resource Reservation

A flow of data needs resources such as a buffer, bandwidth, CPU time, and so on. The quality of service is improved if these resources are reserved beforehand.

INTEGRATED SERVICES

Two models have been designed to provide quality of service in the Internet:

- Integrated Services and
- Differentiated Services.

Both models emphasize the use of quality of service at the network layer (IP), although the model can also be used in other layers such as the data link.

Integrated Services, sometimes called IntServ, is *a flow-based* QoS model, which means that a user needs to create a flow, a kind of virtual circuit, from the source to the destination and inform all routers of the resource requirement.

Signaling

IP is a connectionless, datagram, packet-switching protocol. To implement a flow-based model over a connectionless protocol, a signaling protocol to be run over IP that provides the signaling mechanism for making a reservation. This protocol is called Resource Reservation Protocol (RSVP).

Flow Specification

When a source makes a reservation, it needs to define a flow specification. A flow spedfication has two parts: Rspec (resource specification) and Tspec (traffic specification). Rspec defines the resource that the flow needs to reserve (buffer, bandwidth, etc.). Tspec defines the traffic characterization of the flow.

Admission

After a router receives the flow specification from an application, it decides to admit or deny the service. The decision is based on the previous commitments of the router and the current availability of the resource.

Service Classes

Two classes of services have been defined for Integrated Services: guaranteed service and controlled-load service.

- ***Guaranteed Service Class***

This type of service is designed for real-time traffic that needs a guaranteed minimum end-to-end delay. The end-to-end delay is the sum of the delays in the routers, the propagation delay in the media, and the setup mechanism. Only the first, the sum of the delays in the routers, can be guaranteed by the router. **This type of service guarantees that the packets will arrive within a certain delivery time and are not discarded if flow traffic stays within the boundary of Tspec. Guaranteed services are quantitative services, in which the amount of end-to-end delay and the data rate must be defined by the application.**

- ***Controlled-Load Service Class***

This type of service is designed for applications that can accept some delays, but are sensitive to an overloaded network and to the danger of losing packets. Good examples of these types of applications are file transfer, e-mail, and Internet access. The controlled load service is a **qualitative type of service** in that the application requests the possibility of low-loss or no-loss packets.

RSVP

In the Integrated Services model, an application program needs resource reservation. The Resource Reservation Protocol (RSVP) is a signaling protocol to help IP create a flow and consequently make a resource reservation.

Multicast Trees

RSVP can be also used for unicasting because unicasting is just a special case of multicasting with only one member in the multicast group. The reason for this design is to enable RSVP to provide resource reservations for all kinds of traffic including multimedia which often uses multicasting.

Receiver-Based Reservation

In RSVP, the receivers, not the sender, make the reservation. This strategy matches the other multicasting protocols. For example, in multicast routing protocols, the receivers, not the sender, make a decision to join or leave a multicast group.

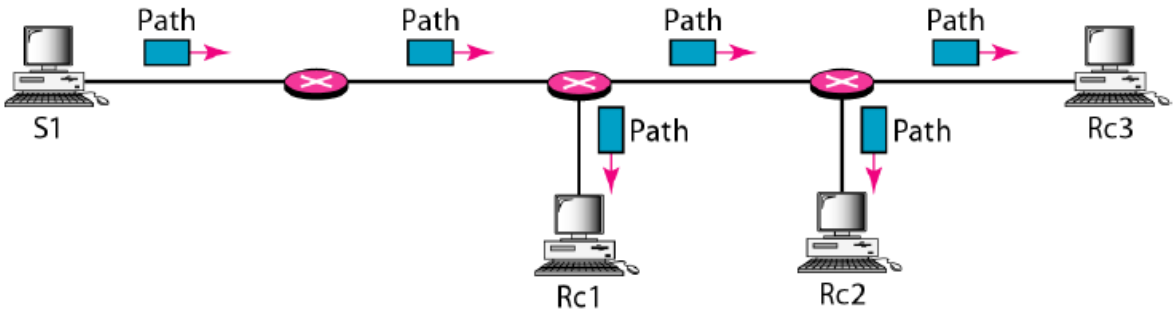
RSVP Messages

RSVP has several types of messages:

1. **Path Messages**

   The receivers in a flow make the reservation in RSVP. However, the receivers do not know the path traveled by packets before the reservation is made. The path is needed for the reservation. To solve the problem, RSVP uses Path messages. A Path message travels from the sender and reaches all receivers in the multicast path. On the way, a Path message stores the necessary information for the receivers. A Path message is sent in a multicast environment; a new message is created when the path diverges.

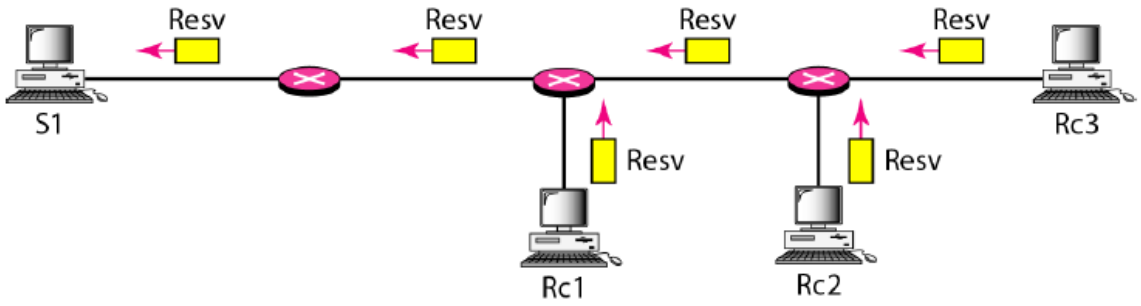
Figure ***Path messages***



Resv Messages

After a receiver has received a Path message, it sends a *Resv* message. The Resv message travels toward the sender (upstream) and makes a resource reservation on the routers that support RSVP. If a router does not support RSVP on the path, it routes the packet based on the best-effort delivery methods.

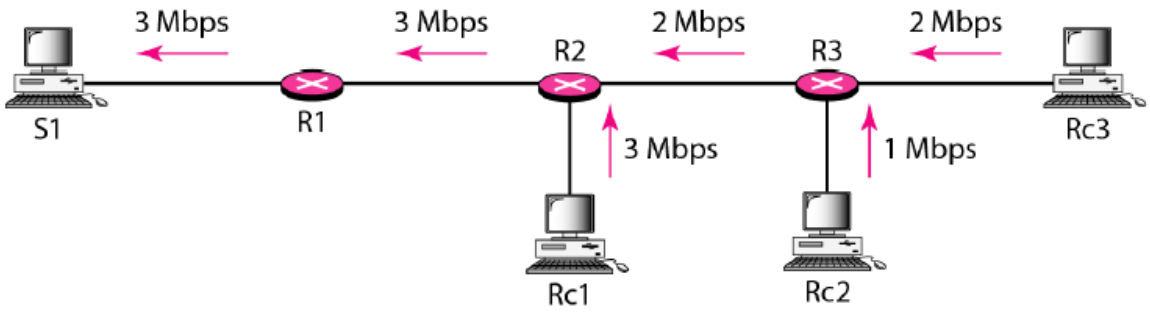
Figure *Resv messages*



Reservation Merging

In RSVP, the resources are not reserved for each receiver in a flow; the reservation is merged.

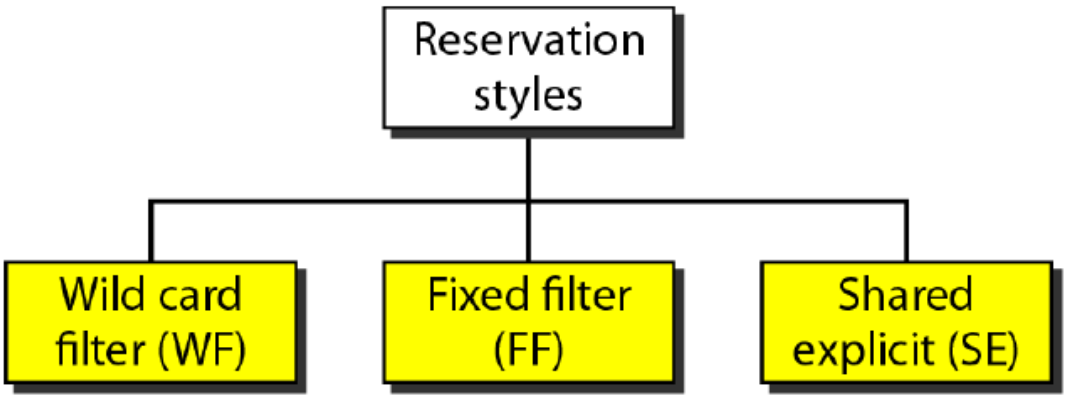
Figure *Reservation merging*



Reservation Styles

When there is more than one flow, the router needs to make a reservation to accommodate all of them. RSVP defines three types of reservation styles.

Figure *Reservation styles*



- **Wild Card Filter Style**
  In this style, the router creates a single reservation for all senders. The reservation is based on the largest request. This type of style is used when the flows from different senders do not occur at the same time.
- **Fixed Filter Style**
  In this style, the router creates a distinct reservation for each flow. This means that if there are *n* flows, *n* different reservations are made. This type of style is used when there is a high probability that flows from different senders will occur at the same time.
- **Shared Explicit Style**
  In this style, the router creates a single reservation which can be shared by a set of flows.

Soft State

The reservation information (state) stored in every node for a flow needs to be refreshed periodically. This is referred to as a *soft state* as compared to the *hard state* used in other virtual-circuit protocols such as ATM or Frame Relay, where the information about the flow is maintained until it is erased. The default interval for refreshing is currently 30 s.

Problems with Integrated Services

There are at least two problems with Integrated Services that may prevent its full implementation in the Internet:

- scalability and
- service-type limitation.

Scalability

The Integrated Services model requires that each router keep information for each flow. As the Internet is growing every day, this is a serious problem.

Service-Type Limitation

The Integrated Services model provides only two types of services, guaranteed and control-load. Those opposing this model argue that applications may need more than these two types of services.

DIFFERENTIATED SERVICES

Differentiated Services (DS or Diffserv) was introduced by the IETF (Internet Engineering Task Force) to handle the shortcomings of Integrated Services. Two fundamental changes were made:

1. The main processing was moved from the core of the network to the edge of the network. This solves the scalability problem. The routers do not have to store information about flows. The applications, or hosts, define the type of service they need each time they send a packet.
2. The per-flow service is changed to per-class service. The router routes the packet based on the class of service defined in the packet, not the flow. This solves the service-type limitation problem. We can define different types of classes based on the needs of applications. Differentiated Services is a class-based QoS model designed for IP.

DS Field

In Diffserv, each packet contains a field called the DS field. The value of this field is set at the boundary of the network by the host or the first router designated as the boundary router. IETF proposes to replace the existing TOS (type of service) field in IPv4 or the class field in IPv6 by the DS field.

Figure ***DS field***

The DS field contains two subfields: DSCP and CU. The DSCP (Differentiated Services Code Point) is a 6-bit subfield that defines the per-hop behavior (PHB). The 2-bit CU (currently unused) subfield is not currently used. The Diffserv capable node (router) uses the DSCP 6 bits as an index to a table defining the packet-handling mechanism for the current packet being processed.

Per-Hop Behavior

The Diffserv model defines per-hop behaviors (PHBs) for each node that receives a packet. So far three PHBs are defined: DE PHB, EF PHB, and AF PHB.

- **DE PHB The DE PHB (default PHB)** is the same as best-effort delivery, which is compatible with TOS.
- **EF PHB The EF PHB (expedited forwarding PHB)** provides the following services:
    - Low loss

- Low latency
- Ensured bandwidth

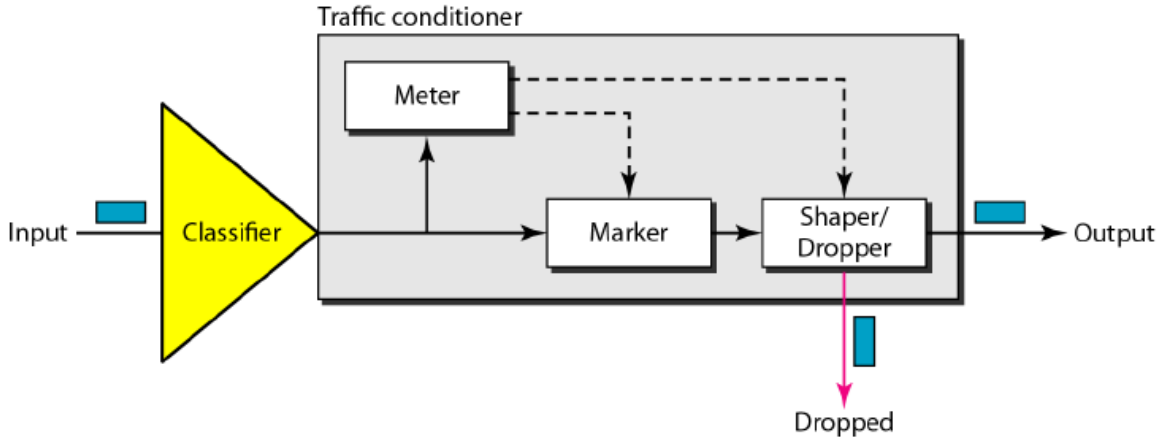
This is the same as having a virtual connection between the source and destination.

- **AF PHB The AF PHB (assured forwarding PHB)** delivers the packet with a high assurance as long as the class traffic does not exceed the traffic profile of the node. The users of the network need to be aware that some packets may be discarded.

Traffic Conditioner

To implement Oiffserv, the OS node uses traffic conditioners such as meters, markers, shapers, and droppers.

Figure ***Traffic conditioner***



- **Meters**

The meter checks to see if the incoming flow matches the negotiated traffic profile. The meter also sends this result to other components. The meter can use several tools such as a token bucket to check the profile.

- **Marker**

A marker can remark a packet that is using best-effort delivery (OSCP: 000000) or down-mark a packet based on information received from the meter. Downmarking (lowering the class of the flow) occurs if the flow does not match the profile. A marker does not up-mark (promote the class) a packet.

- **Shaper**

A shaper uses the information received from the meter to reshape the traffic if it is not compliant with the negotiated profile. Dropper A dropper, which works as a shaper with no buffer, discards packets if the flow severely violates the negotiated profile.