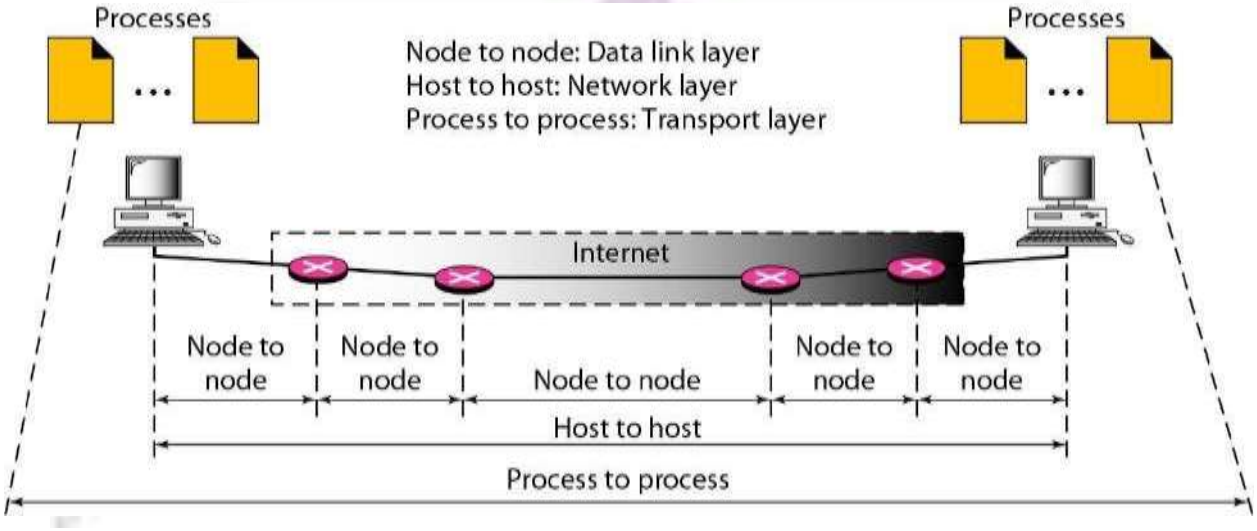
UNIT-4

TRANSPORT LAYER

PROCESS-TO-PROCESS DELIVERY

- The data link layer is responsible for delivery of frames between two neighboring nodes over a link. This is called **node-to-node delivery**.
- The network layer is responsible for delivery of datagrams between two hosts. This is called host-to-host delivery.
- The real communication takes place between two processes (application programs). We need process-to-process delivery.
- At any moment, several processes may be running on the source host and several on the destination host.
- To complete the delivery, we need a mechanism to deliver data from one of these processes running on the source host to the corresponding process running on the destination host.
- 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

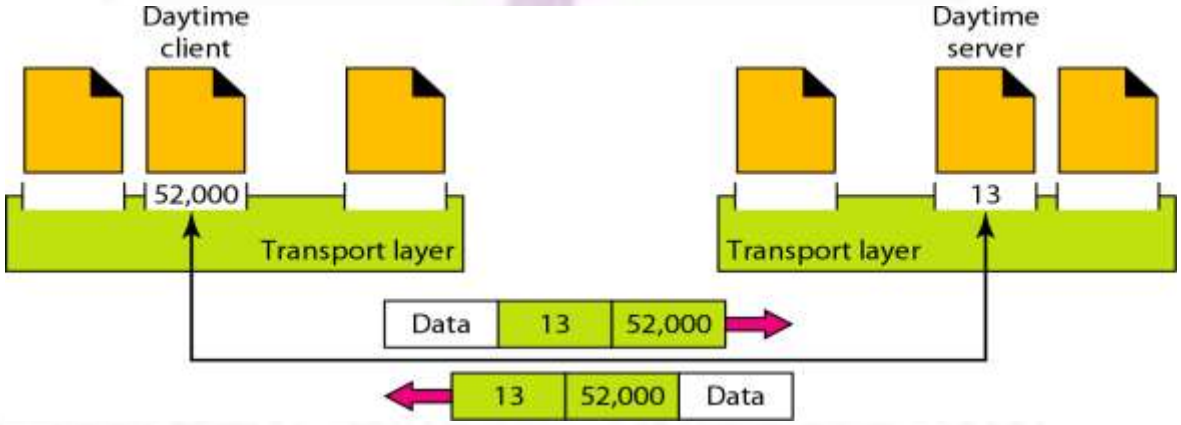


CLIENT/SERVER PARADIGM

- A process on the local host, called a client, needs services from a process usually on the remote host, called a server. Both processes (client and server) have the same name.For example, to get the day and time from a remote machine, we need a Daytime client process

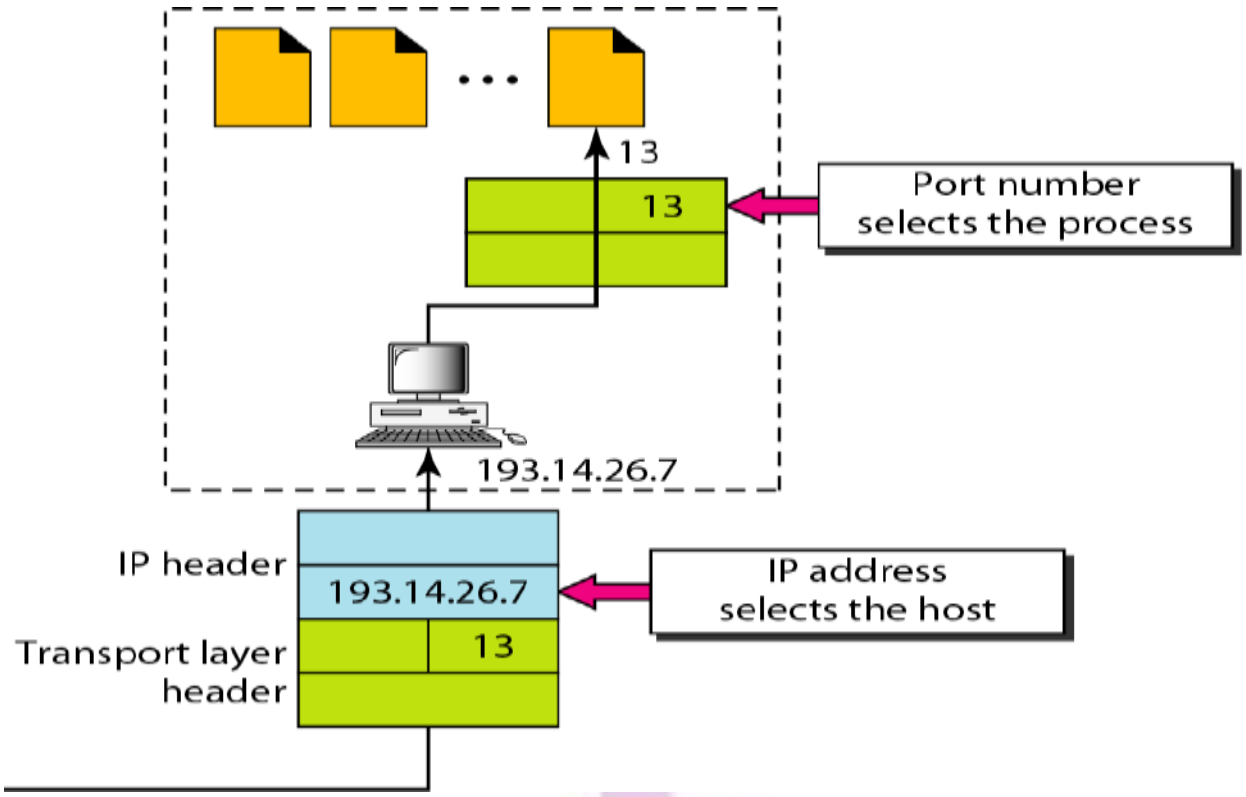
running on the local host and a Daytime server process running on a remote machine.

- Whenever we need to deliver something to one specific destination among many, we need an address. At the transport layer, we need a transport layer address, called a port number, to choose among multiple processes running on the destination host.
- The destination port number is needed for delivery; the source port number is needed for the reply.
- In the Internet model, the port numbers are 16-bit integers between 0 and 65,535.
- The client program defines itself with a port number, chosen randomly by the transport layer software running on the client host. This is the ephemeral port number.
- The server process must also define itself with a port number. However this cannot be chosen randomly.
- If the port number is chosen randomly then the process at the client site that wants to access that server and use its services will not know the port number.
- Of course, one solution would be to send a special packet and request the port number of a specific server, but this requires more overhead.
- To overcome the drawback of additional overhead, the internet has chosen to assign a universal port number called well known port number to the server program.



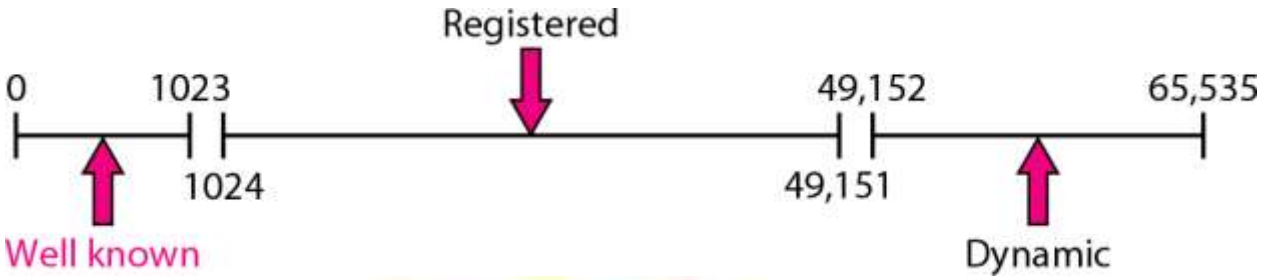
IP Address Vs Port Address: The destination IP address defines the host among the different hosts

in the world. After the host has been selected, the port number defines one of the processes on this particular host.



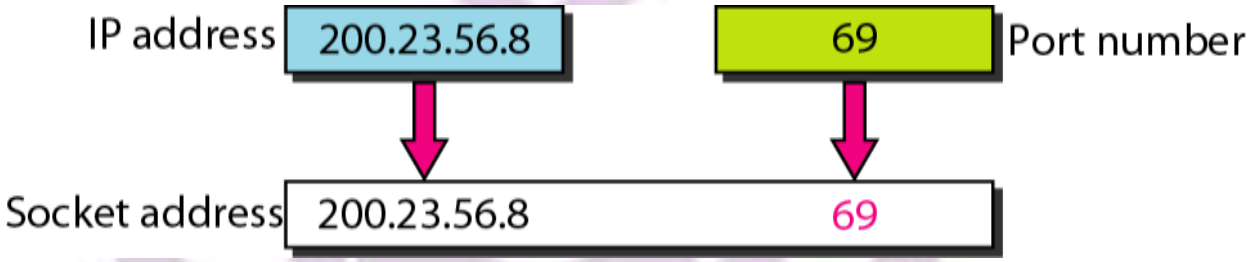
IANA RANGES

- The IANA (Internet Assigned Number Authority) has divided the port numbers into three ranges: well known, registered, and dynamic (or private).
- Well-known ports: The ports ranging from 0 to 1023 are assigned and controlled by IANA. These are the well-known ports.
- Registered ports: The ports ranging from 1024 to 49,151 are not assigned or controlled by IANA. They can only be registered with IANA to prevent duplication.
- Dynamic ports: The ports ranging from 49,152 to 65,535 are neither controlled nor registered. They can be used by any process. These are the ephemeral ports



SOCKET ADDRESSES

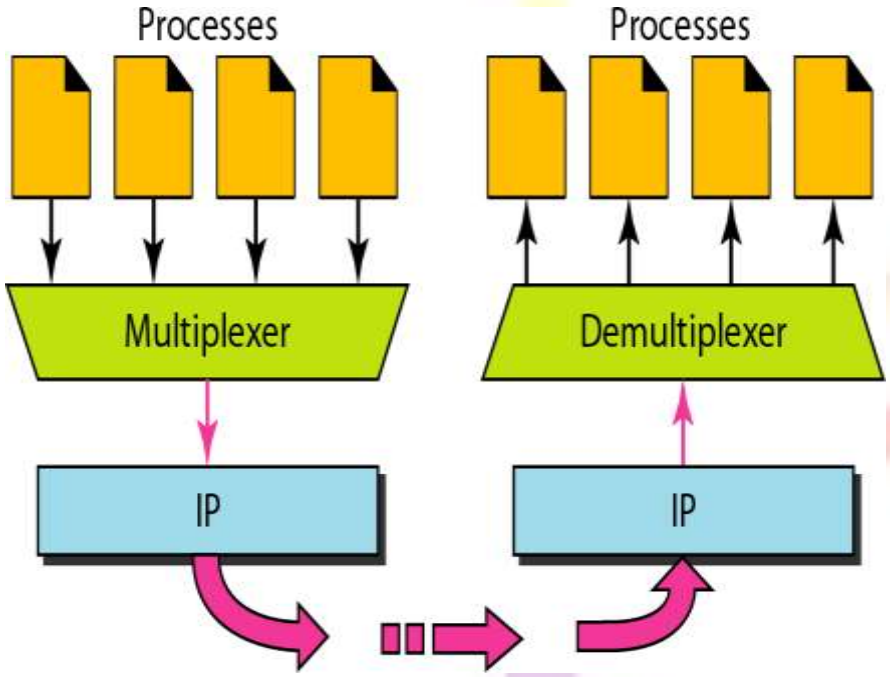
- Process to Process delivery needs two identifiers, IP address, and port number, at each end to make a connection.
- The combination of an IP address and a port number is called a socket address.
- The client socket address defines the client process uniquely just as the server socket address defines the server process uniquely.
- A Transport Layer protocol needs a pair of socket addresses; the client socket address and the server socket address for processes to process communication.



MULTIPLEXING & DEMULTIPLEXING

- Multiplexing: At the sender site, there may be several processes that need to send packets. But, 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

- 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. We will see shortly that one of the transport layer protocols in the Internet model, UDP, is connectionless.
- 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. We will see shortly that TCP and SCTP are connection-oriented protocols.

RELIABLE VERSUS UNRELIABLE

- Reliable: 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

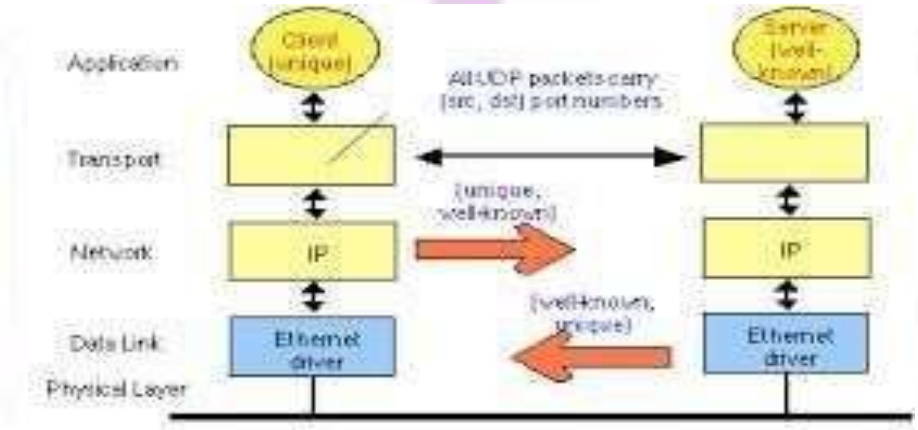
or it needs fast service or the nature of the service does not demand flow and error control (real-time applications), then an unreliable protocol can be used.

- Note : *If the data link layer is reliable and has flow and error control, do we need this at the transport layer, too? The answer is yes. Reliability at the data link layer is between two nodes; we need reliability between two ends.*

UDP:

The **User Datagram Protocol** (**UDP**) is one of the core members of the Internet protocol suite. The protocol was designed by David P. Reed in 1980 and formally defined in RFC 768.

UDP uses a simple connectionless transmission model with a minimum of protocol mechanism. It has no handshaking dialogues, and thus exposes any unreliability of the underlying network protocol to the user's program. There is no guarantee of delivery, ordering, or duplicate protection. UDP provides checksums for data integrity, and port numbers for addressing different functions at the source and destination of the datagram.



With UDP, computer applications can send messages, in this case referred to as datagram, to other hosts on an Internet Protocol (IP) network without prior communications to set up special transmission channels or data paths. UDP is suitable for purposes where error checking and correction is either not necessary or is performed in the application, avoiding the overhead of such

processing at the network interface level. Time-sensitive applications often use UDP because dropping packets is preferable to waiting for delayed packets, which may not be an option in a real-time system.[1] If error correction facilities are needed at the network interface level, an application may use the Transmission Control Protocol (TCP) or Stream Control Transmission Protocol (SCTP)

UDP (User Datagram Protocol) is an alternative communications protocol to Transmission Control Protocol (TCP) used primarily for establishing low-latency and loss tolerating connections between applications on the Internet. Both UDP and TCP run on top of the Internet Protocol (IP) and are sometimes referred to as UDP/IP or TCP/IP. Both protocols send short packets of data, called datagram.

UDP provides two services not provided by the IP layer. It provides port numbers to help distinguish different user requests and, optionally, a checksum capability to verify that the data arrived intact.

TCP has emerged as the dominant protocol used for the bulk of Internet connectivity owing to services for breaking large data sets into individual packets, checking for and resending lost packets and reassembling packets into the correct sequence. But these additional services come at a cost in terms of additional data overhead, and delays called latency.

UDP is an ideal protocol for network applications in which perceived latency is critical such as gaming, voice and video communications, which can suffer some data loss without adversely affecting perceived quality. In some cases, forward error correction techniques are used to improve audio and video quality in spite of some loss.

UDP can also be used in applications that require lossless data transmission when the application is configured to manage the process of retransmitting lost packets and correctly arranging received packets. This approach can help to improve the data transfer rate of large files compared with TCP.

Attributes

A number of UDP's attributes make it especially suited for certain applications.

- It is transaction-oriented, suitable for simple query-response protocols such as the Domain Name System or the Network Time Protocol.
- It provides datagram, suitable for modeling other protocols such as in IP tunneling or Remote Procedure Call and the Network File System.
- It is simple, suitable for bootstrapping or other purposes without a full protocol stack, such as the DHCP and Trivial File Transfer Protocol.
- The lack of retransmission delays makes it suitable for real-time applications such as Voice over IP, online games, and many protocols built on top of the Real Time Streaming Protocol.
- It is stateless, suitable for very large numbers of clients, such as in streaming media applications for example IPTV

- Works well in unidirectional communication, suitable for broadcast information such as in many kinds of service discovery and shared information such as broadcast time or Routing Information Protocol
- UDP provides application multiplexing (via port numbers) and integrity verification (via checksum) of the header and payload.[4] If transmission reliability is desired, it must be implemented in the user's application.

The UDP header consists of 4 fields, each of which is 2 bytes (16 bits).[1] The use of the fields "Checksum" and "Source port" is optional in IPv4 (pink background in table). In IPv6 only the source port is optional

Source port number

- This field identifies the sender's port when meaningful and should be assumed to be the port to reply to if needed. If not used, then it should be zero. If the source host is the client, the port number is likely to be an ephemeral port number. If the source host is the server, the port number is likely to be a well-known port number.[2]
- Destination port number
- This field identifies the receiver's port and is required. Similar to source port number, if the client is the destination host then the port number will likely be an ephemeral port number and if the destination host is the server then the port number will likely be a well-known port number.[2]
- Length

- A field that specifies the length in bytes of the UDP header and UDP data. The minimum length is 8 bytes because that is the length of the header. The field size sets a theoretical limit of 65,535 bytes (8 byte header + 65,527 bytes of data) for a UDP datagram. The practical limit for the data length which is imposed by the underlying IPv4 protocol is 65,507 bytes (65,535 − 8 byte UDP header − 20 byte IP header).[2]

- In IPv6 Jumbo grams it is possible to have UDP packets of size greater than 65,535 bytes.[5] RFC 2675 specifies that the length field is set to zero if the length of the UDP header plus UDP data is greater than 65,535.

- Checksum

- The checksum field is used for error-checking of the header and data. If no checksum is generated by the transmitter, the field uses the value all-zeros.[6] This field is not optional for IPv6.[7]

Reliable byte stream (TCP):

A **reliable byte stream** is a common service paradigm in computer networking; it refers to a byte stream in which the bytes which emerge from the communication channel at the recipient are exactly the same, and in exactly the same order, as they were when the sender inserted them into the channel.

The classic example of a reliable byte stream communication protocol is the Transmission Control Protocol, one of the major building blocks of the Internet.

A reliable byte stream is not the only reliable service paradigm which computer network communication protocols provide, however; other protocols (e.g. SCTP) provide a reliable message stream, i.e. the data is divided up into distinct units, which are provided to the consumer of the data as discrete objects.

Connection-oriented (TCP):

- Flow control: keep sender from overrunning receiver
- Congestion control: keep sender from overrunning network

Characteristics of TCP Reliable Delivery:

TCP provides a **reliable, byte-stream, full-duplex inter-process communications service** to application programs/processes. The service is **connection-oriented** and uses the concept of **port numbers** to identify processes.

Reliable

All data will be delivered correctly to the destination process, without errors, even though the underlying packet delivery service (IP) is unreliable -- see later.

Connection-oriented

Two process which desire to communicate using TCP must first request a **connection**. A connection is closed when communication is no longer desired.

Byte-stream

An application which uses the TCP service is unaware of the fact that data is broken into **segments** for transmission over the network.

Full-duplex

Once a TCP connection is established, application data can flow in both directions simultaneously -- note, however, that many application protocols do not take advantage of this.

Port Numbers

Port numbers identify processes/connections in TCP.

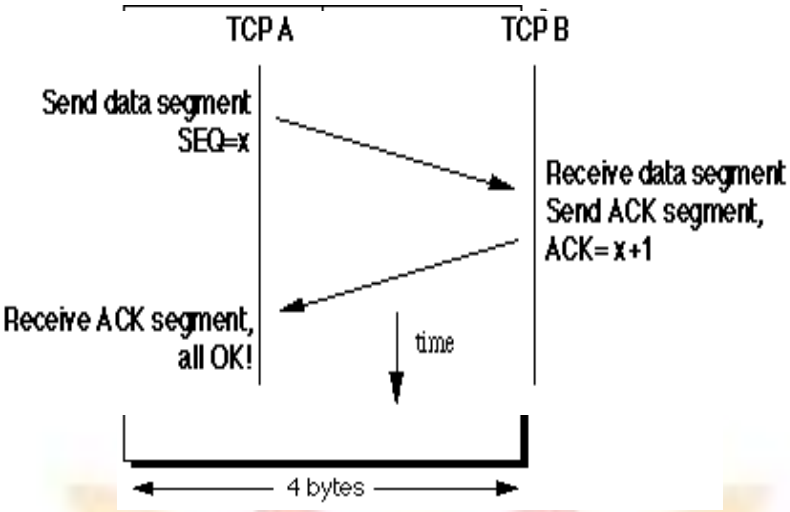
Edge Systems and Reliable Transport

1. An **edge system** is any computer (host, printer, even a toaster...) which is "connected to" the Internet -- that is, it has access to the Internet's packet delivery system, but doesn't itself form part of that delivery system.
2. A **transport service** provides communications between application processes running on edge systems. As we have already seen, application processes communicate with each another using **application protocols** such as HTTP and SMTP. The interface between an application process and the transport service is normally provided using the **socket** mechanism.

Most application protocols require **reliable data transfer**, which in the Internet is provided by the **TCP** transport service/protocol. Note: some applications **do not** require reliability, so the unreliable **UDP** transport service/protocol is also provided as an alternative

TCP Segments

TCP slices (dices?) the incoming byte-stream data into **segments** for transmission across the Internet. A segment is a highly-structured data package consisting of an administrative **header** and some **application data**.



Source and Destination Port Numbers

We have already seen that TCP server processes wait for connections at a pre-agreed port number. At connection establishment time, TCP first allocates a **client port number** -- a port number by which the client, or initiating, process can be identified. Each segment contains both port numbers.

Segment and Acknowledgment Numbers

Every transmitted segment is identified with a 32-bit **Sequence number**[2], so that it can be explicitly acknowledged by the receipient. The Acknowledgment Number identifies the last segment recived by the originator of this segment.

Application Data

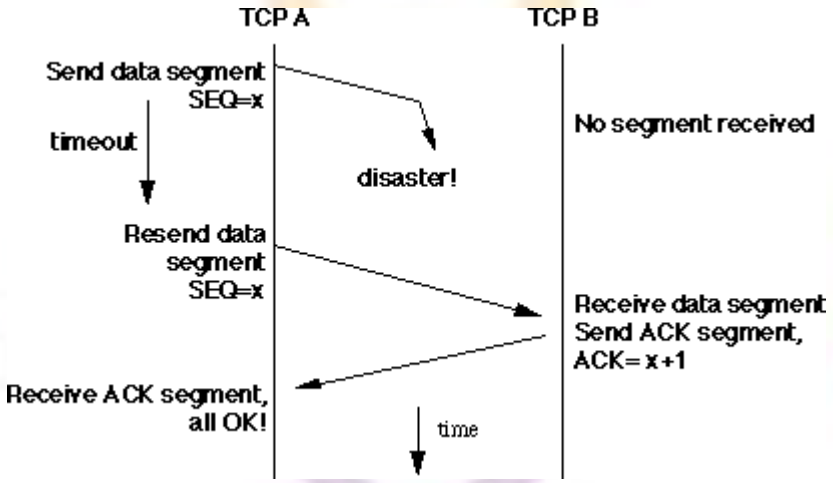
Optional because some segments convey only **control information** -- for example, an ACK segment has a valid acknowledgment number field, but no data. The data field can be any size up to the currently configured **MSS** for the whole segment.

TCP Operation

When a segment is received correct and intact at its destination, an **acknowledgment** (ACK) segment is returned to the sending TCP. This ACK contains the sequence number of the last byte correctly received, incremented by 1[3]. ACKs are cumulative -- a single ACK can be sent for several segments if, for example, they all arrive within a short period of time.

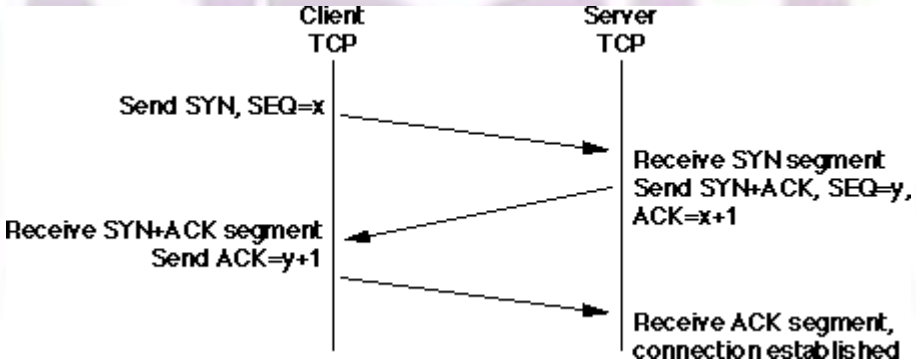
The network service can fail to deliver a segment. If the sending TCP waits for **too long**[4] for an acknowledgment, it times out and resends the segment, on the assumption that the datagram has been lost.

In addition, the network can potentially deliver duplicated segments, and can deliver segments out of order. TCP buffers or discards out of order or duplicated segments appropriately, using the byte count for identification.



TCP Connections:

An application process requests TCP to establish, or open, a (reliable) connection to a server process running on a specified edge-system, and awaiting connections at a known port number. After allocating an unused client-side port number[5], TCP initiates an exchange of connection establishment "control segments":



- This exchange of segments is called a **3-way handshake** (for obvious reasons), and is necessary because any one of the three segments can be lost, etc. The **ACK** and **SYN** segment names refer to "control bits" in the TCP header: for example, if the **ACK** bit is set, then this is an **ACK** segment.
- Each TCP chooses an **random initial sequence number** (the **x** and **y** in this example). This is crucial to the protocol's operation if there's a small chance that "old" segments (from a closed connection) might be interpreted as valid within the current connection.
- A connection is **closed** by another 3-way handshake of control segments. It's possible for a connection to be **half open** if one end requests close, and the other doesn't respond with an appropriate segment.

Optional: TCP Flow Control, Congestion Control and Slow Start

TCP attempts to make the best possible use of the underlying network, by sending data at the highest possible rate that won't cause segment loss. There are two aspects to this:

Flow Control

The two TCPs involved in a connection each maintain a **receive window** for the connection, related to the size of their **receive buffers**. For TCP "**A**", this is the maximum number of bytes that TCP "**B**" should send to it before "blocking" and waiting for an ACK. All TCP segments contain a **window** field, which is used to inform the other TCP of the sender's receive window size -- this is called "advertising a window size". At any time, for example, TCP **B** can have multiple segments "**in-flight**" -- that is, sent but not yet ACK'd -- up to TCP **A**'s advertised window.

Congestion Avoidance and Control

When a connection is initially established, the TCPs know nothing at all about the speed, or capacity, of the networks which link them. The built-in "**slow start**" algorithm controls the rate at which segments are initially sent, as TCP tentatively discovers reasonable numbers for the connection's **Round Trip Time (RTT)** and its variability. TCP also slowly increases the number of segments "in-flight", since this increases the utilisation of the network.

Every TCP in the entire Internet is attempting to make full use of the available network, by increasing the number of "in-flight" segments it has outstanding. Ultimately there will come a point where the sum of the traffic, in some region of the network exceeds one or more router's buffer space, at which time segments will be dropped. When TCP "times out", and has to resend a dropped segment, it takes this as an indication that it (and all the other TCPs) have pushed the network just a little too hard. TCP immediately reduces its **congestion window** to a low value, and slowly, slowly allows it to increase again as ACKs are received.

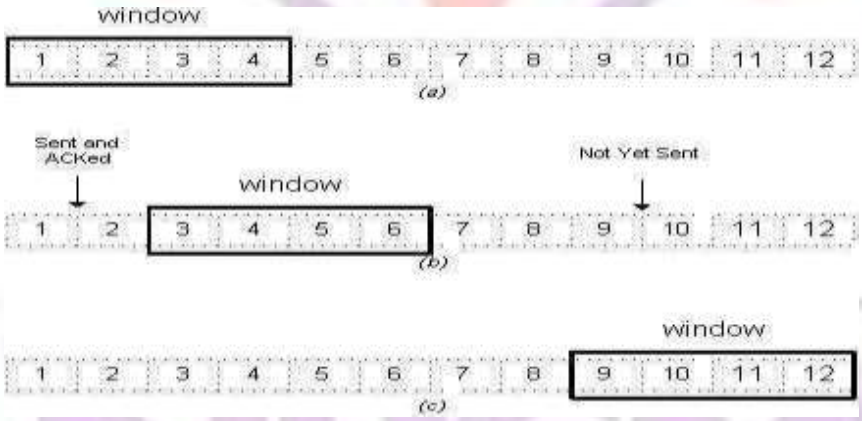
Flow control:

- Based on window mechanism

One of TCP's primary functions is to properly match the transmission rate of the sender to that of the receiver and the network. It is important for the transmission to be at a high enough rate to ensure good performance, but also to protect against overwhelming the network or receiving host.

TCP's 16-bit window field is used by the receiver to tell the sender how many bytes of data the receiver is willing to accept. Since the window field is limited to a maximum of 16 bits, this provides for a maximum window size of 65,535 bytes.

The window size advertised by the receiver tells the sender how much data, starting from the current position in the TCP data byte stream can be sent without waiting for further acknowledgements. As data is sent by the sender and then acknowledged by the receiver, the window slides forward to cover more data in the byte stream. This concept is known as a "sliding window".



The window boundary is eligible to be sent by the sender. Those bytes in the stream prior to the window have already been sent and acknowledged. Bytes ahead of the window have not been sent and must wait for the window to "slide" forward before they can be transmitted by the sender. A receiver can adjust the window size each time it sends acknowledgements to the sender. The maximum transmission rate is ultimately bound by the receiver's ability to accept and process data.

ERROR CONTROL:

Retransmission:

TCP is relegated to rely mostly upon implicit signals it learns from the network and remote host. TCP must make an educated guess as to the state of the network and trust the information from the remote host in order to control the rate of data flow. This may seem like an awfully tricky problem, but in most cases TCP handles it in a seemingly simple and straightforward way.

A sender's implicit knowledge of network conditions may be achieved through the use of a **timer**. For each TCP segment sent the sender expects to receive an acknowledgement within some period of time otherwise an error in the form of a timer expiring signals that that something is wrong.

Somewhere in the end-to-end path of a TCP connection a segment can be lost along the way. Often this is due to congestion in network routers where excess packets must be dropped. TCP not only must correct for this situation, but it can also **learn** something about network conditions from it.

Whenever TCP transmits a segment the sender starts a timer which keeps track of how long it takes for an acknowledgment for that segment to return. This timer is known as the **retransmission timer**. If an acknowledgement is returned before the timer expires, which by default is often initialized to 1.5 seconds, the timer is reset with no consequence. If however an acknowledgement for the segment does not return within the timeout period, the sender would retransmit the segment and double the retransmission timer value for each consecutive timeout up to a maximum of about 64 seconds. If there are serious network problems, segments may take a few minutes to be successfully transmitted before the sender eventually times out and generates an error to the sending application.

Fundamental to the timeout and retransmission strategy of TCP is the measurement of the **round-trip time** between two communicating TCP hosts. The round-trip time may vary during the TCP connection as network traffic patterns fluctuate and as routes become available or unavailable.

A TCP option negotiated in the TCP connection establishment phase sets the number of bits by which the window is right-shifted in order to increase the value of the window. TCP keeps track of when data is sent and at what time acknowledgements covering those sent bytes are returned. TCP uses this information to calculate an estimate of round trip time. As packets are sent and acknowledged, TCP adjusts its round-trip time estimate and uses this information to come up with a reasonable timeout value for packets sent. If acknowledgements return quickly, the round- trip time is short and the retransmission timer is thus set to a lower value. This allows TCP to quickly retransmit data when network response time is good, alleviating the need for a long delay between the occasional lost segment. The converse is also true. TCP does not retransmit data too quickly during times when network response time is long.

If a TCP data segment is lost in the network, a receiver will never even know it was once sent. However, the sender is waiting for an acknowledgement for that segment to return. In one case, if an acknowledgement doesn't return, the sender's retransmission timer expires which causes a retransmission of the segment. If however the sender had sent at least one additional segment after the one that was lost and that later segment is received correctly, the receiver does not send an acknowledgement for the later, out of order segment.

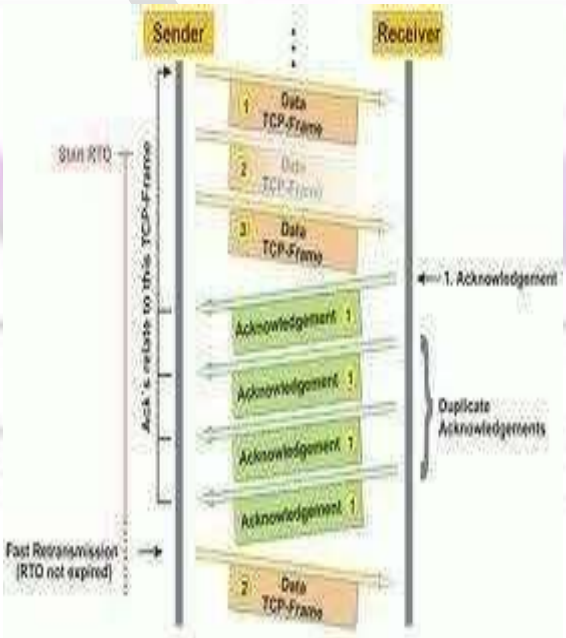
The receiver cannot acknowledgement out of order data; it must acknowledge the last contiguous byte it has received in the byte stream prior to the lost segment. In this case, the receiver will send an acknowledgement indicating the last contiguous byte it has received. If that last

contiguous byte was already acknowledged, we call this a duplicate ACK. The reception of duplicate ACKs can implicitly tell the sender that a segment may have been lost or delayed. The sender knows this because the receiver only generates a duplicate ACK when it receives other, out of order segments. In fact, the Fast Retransmit algorithm described later uses duplicate ACKs as a way of speeding up the retransmission process.

A TCP sender uses a timer to recognize lost segments. If an acknowledgement is not received for a particular segment within a specified time (a function of the estimated Round-trip delay time), the sender will assume the segment was lost in the network, and will retransmit the segment.

Duplicate acknowledgement is the basis for the fast retransmit mechanism which works as follows: after receiving a packet (e.g. with sequence number 1), the receiver sends an acknowledgement by adding 1 to the sequence number (i.e., acknowledgement number 2) which means that the receiver receives the packet number 1 and it expects packet number 2 from the sender. Let's assume that three subsequent packets have been lost. In the meantime the receiver receives packet numbers 5 and 6. After receiving packet number 5, the receiver sends an acknowledgement, but still only for sequence number 2. When the receiver receives packet number 6, it sends yet another acknowledgement value of 2. Because the sender receives more than one acknowledgement with the same sequence number (2 in this example) this is called duplicate acknowledgement.

The fast retransmit enhancement works as follows: if a TCP sender receives a specified number of acknowledgements which is usually set to three duplicate acknowledgements with the same acknowledge number (that is, a total of four acknowledgements with the same acknowledgement number), the sender can be reasonably confident that the segment with the next higher sequence number was dropped, and will not arrive out of order. The sender will then retransmit the packet that was presumed dropped before waiting for its timeout.



TCP Congestion control:

The standard fare in TCP implementations today can be found in RFC 2581 [2]. This reference document specifies four standard congestion control algorithms that are now in common use. Each of the algorithms noted within that document was actually designed long before the standard was published [9], [11]. Their usefulness has passed the test of time.

The four algorithms, Slow Start, Congestion Avoidance, Fast Retransmit and Fast Recovery are described below.

Slow Start

Slow Start, a requirement for TCP software implementations is a mechanism used by the sender to control the transmission rate, otherwise known as sender-based flow control. This is accomplished through the return rate of acknowledgements from the receiver. Inother words, the rate of acknowledgements returned by the receiver determine the rate at which the sender can transmit data.

When a TCP connection first begins, the Slow Start algorithm initializes a **congestion window** to one segment, which is the maximum segment size (MSS) initialized by the receiver during the connection establishment phase. When acknowledgements are returned by the receiver, the congestion window increases by one segment for each acknowledgement returned. Thus, the sender can transmit the minimum of the congestion window and the advertised window of the receiver, which is simply called the **transmission window**.

Slow Start is actually not very slow when the network is not congested and network response time is good. For example, the first successful transmission and acknowledgement of a TCP segment increases the window to two segments. After successful transmission of these two segments and acknowledgements completes, the window is increased to four segments. Then eight segments, then sixteen segments and so on, doubling from there on out up to the maximum window size advertised by the receiver or until congestion finally does occur.

At some point the congestion window may become too large for the network or network conditions may change such that packets may be dropped. Packets lost will trigger a timeout at the sender. When this happens, the sender goes into congestion avoidance mode as described in the next section.

SCTP

Stream Control Transmission Protocol (SCTP) is a new reliable, message-oriented transport layer protocol. SCTP, however, is mostly designed for Internet applications that have recently been introduced. These new applications need a more sophisticated service than TCP can provide. SCTP combines the best features of UDP and TCP. SCTP is a reliable message-oriented protocol.

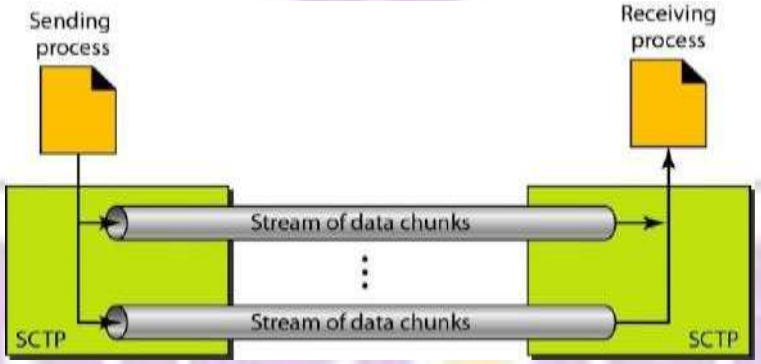
SCTP SERVICES

Process-to-Process Communication

SCTP uses all well-known ports in the TCP space. Table 23.4 lists some extra port numbers used by SCTP.

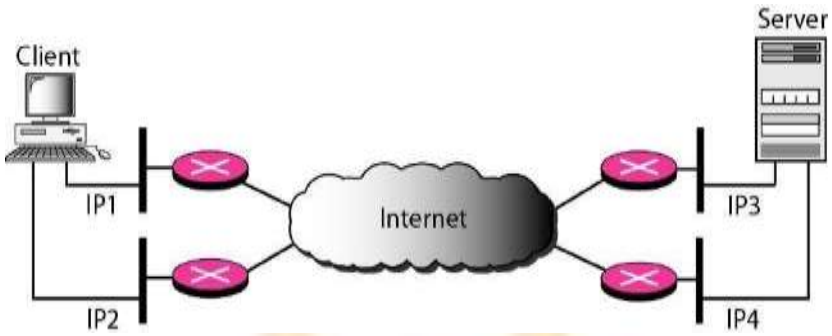
Multiple Streams

We learned in the previous section that TCP is a stream-oriented protocol. Each connection between a TCP client and a TCP server involves one 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.



Multihoming

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 below shows the idea of multihoming.



In Figure above, the client is connected to two local networks with two IP addresses. The server is also connected to two networks with two IP addresses. The client and the server can make an association, using four different pairs of IP addresses

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.

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.

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

- Transmission Sequence Number: The unit of data in SCTP is a DATA chunk, Data transfer in SCTP is controlled by numbering the data chunks. SCTP uses a transmission sequence number (TSN) to number the data chunks (same as sequence number in TCP).
- Stream Identifier: 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

- 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).

SCTP PACKETS

Data are carried as data chunks, control information is carried as control chunks. The SCTP packets are discussed in comparison with TCP packets.

1. The control information in TCP is part of the header; the control information in SCTP is included in the control chunks. There are several types of control chunks; each is used for a different purpose.

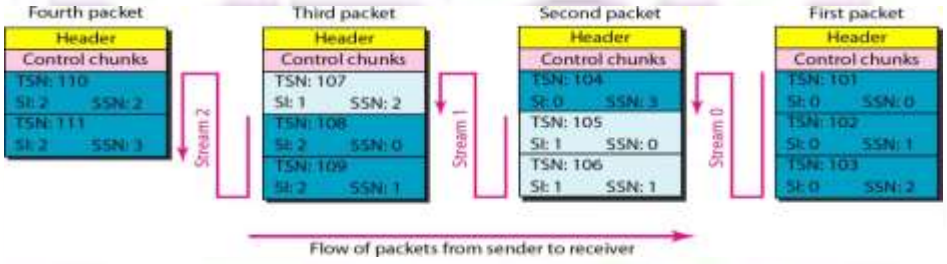
2. The data in a TCP segment treated as one entity; an SCTP packet can carry several data chunks; each can belong to a different stream.
3. The options section, which can be part of a TCP segment, does not exist in an SCTP packet. Options in SCTP are handled by defining new chunk types.
4. The mandatory 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.
6. The verification tag in SCTP is an association identifier, which does not exist in TCP. In TCP, the combination of IP and port addresses defines a connection; in SCTP we may have multihorning using different IP addresses. A unique verification tag is needed to define each association.

7. 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.
8. 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.

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 carried by 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 it delivers 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.

We also assume that the network allows only three data chunks per packet, which means that we need. four packets as shown in Figure 23.30. Data chunks in stream 0 are carried in the first packet and part of the second packet; those in stream 1 are carried in the second and third packets; those in stream 2 are carried in the third and fourth packets.



Note that each data chunk needs three identifiers: TSN, SI, and SSN. TSN is a cumulative number and is used, as we will see later, for flow control and error control. SI defines the stream to which the chunk belongs. SSN defines the chunk's order in a particular stream. In our example, SSN starts from 0 for each stream.

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 (for example, a SYN segment needs to be acknowledged by an ACK segment).

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.

Flow Control

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

Error Control

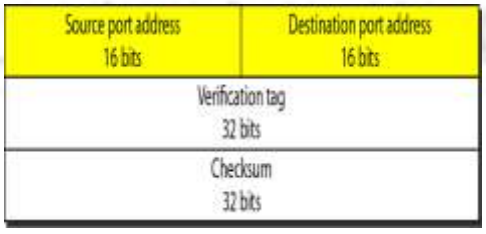
Like TCP, SCTP implements error control to provide reliability. TSN numbers and acknowledgment numbers are used for error control.

Congestion Control

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

PACKETS 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. The following fig shows the general header.
- **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. The fonnat of the general header is given here.



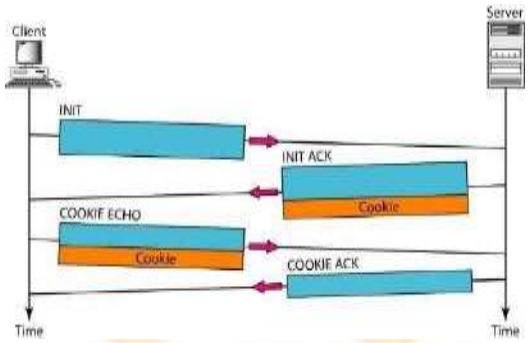
- **Source port address**. This is a 16-bit field that defines the port number of the process sending the packet.
- **Destination port address**. This is a 16-bit field that defines the port number of the process receiving the packet.
- **Verification tag**. This is a number that matches a packet to an association. This prevents a packet from a previous association from being mistaken as a packet in this association. It serves as an identifier for the association; it is repeated in every packet during the association. There is a separate verification used for each direction in the association.
- **Checksum.** This 32-bit field contains a CRC-32 checksum. Note that the size of the checksum is increased from 16 (in UDP, TCP, and IP) to 32 bits to allow the use of the CRC-32 checksum

An SCTP Association

SCTP, like TCP, is a connection-oriented protocol. However, a connection in SCTP is called an *association* to emphasize multihoming. An 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). SCTP association establishment is shown in Figure fig. 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.

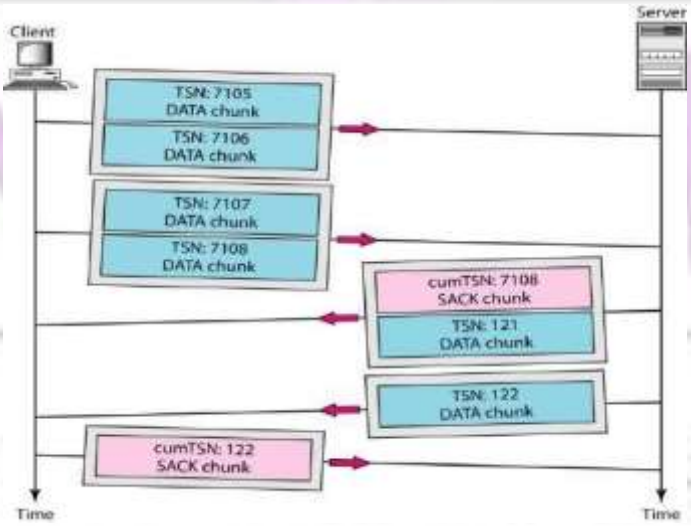
Cookie: To overcome sync flooding (malicious attack) designer has chosen cookie. After receiving init from client. The sever sends cookie to the client, The client runs the cookie and sends the response of cookie along with cookie for its identification.



Data Transfer

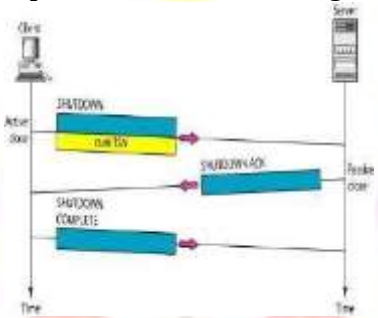
The whole purpose of an association is to transfer data between two ends. After the association is established, bidirectional data transfer can take place. The client and the server can both send data. Like TCP, SCTP supports piggybacking. For better illustration consider following figure.

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. However, unlike TCP, SCTP does not allow a halfclose situation. 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 below.



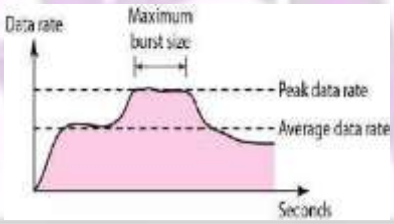
Note: Flow control and error control same as in TCP IP (Refer TCP)

DATA TRAFFIC

- The main focus of congestion control and quality of service is data traffic. In congestion control we try to avoid traffic congestion. In quality of service, we try to create an appropriate environment for the traffic. So, before talking about congestion control and quality of service.

PARAMETERS OF DATA TRAFFIC

- **Traffic Descriptor:** Traffic descriptors are qualitative values that represent a data flow

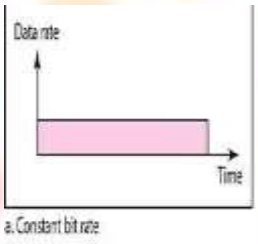


- **Average Data Rate:** The average data rate is the number of bits sent during a period of time, divided by the number of seconds in that period.
- **Peak Data Rate:** The peak data rate defines the maximum data rate of the traffic. In Figure above it is the maximum y axis value.
- **Maximum Burst Size:** Although the peak data rate is a critical value for the network, it can usually be ignored if the duration of the peak value is very short.

- **Effective Bandwidth:** The effective bandwidth is the bandwidth that the network needs to allocate for the flow of traffic. The effective bandwidth is a function of three values: average data rate, peak data rate, and maximum burst size.

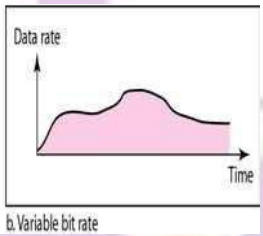
TRAFFIC PROFILES

- **Constant Bit Rate:** A constant-bit-rate (CBR), or a fixed-rate, traffic model has a data rate that does not change. In this type of flow, the average data rate and the peak data rate are the same. The maximum burst size is not applicable This type of traffic is very easy to handle since network knows how much bandwidth to allocate for this type of flow.



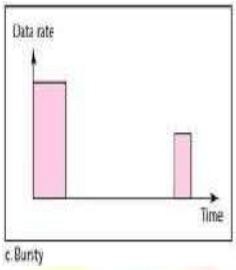
a. Constant bit rate

- In the **variable-bit-rate (VBR),** the rate of the data flow changes in time, with the changes smooth instead of sudden and sharp. Here the type of flow, the average data rate and the peak data rate are different. The maximum burst size is usually a small value This type of traffic is more difficult to handle than constant-bit-rate traffic, but it normally does not need to be reshaped.



b. Variable bit rate

- In the **bursty data** category, the data rate changes suddenly in a very short time. It may jump from zero, for example, to 1 Mbps in a few microseconds and vice versa. The average bit rate and the peak bit rate are very different values in this type of flow. The maximum burst size is significant. This is the most difficult type of traffic for a network to handle because the profile is very unpredictable. To handle this type of traffic, the network normally needs to reshape it, using reshaping techniques unpredictable.



CONGESTION

- An important issue in a packet-switched network is congestion. Congestion in a network may occur if the number of packets sent to the network-is greater than the number of packets a network can handle.
- Congestion control refers to the mechanisms and techniques to control the congestion and keep the load below the capacity

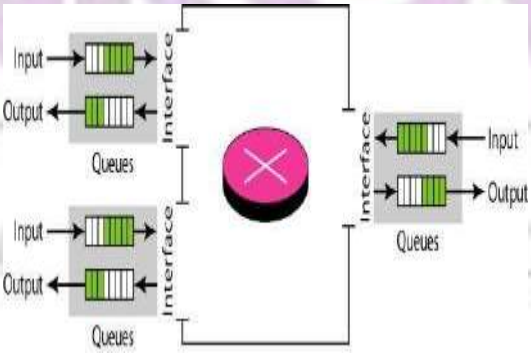
Reasons for existence of congestion:

- congestion happens on a freeway because any abnonnality in the flow.
- Congestion in a network or internetwork occurs because routers and switches have queues-buffers that hold the packets before and after processing
- When a packet arrives at the incoming interface, it undergoes three steps before departing, as shown in Figure.

1. The packet is put at the end of the input queue while waiting to be checked.

2. The processing module of the router removes the packet from the input queue once it reaches the front of the queue and uses its routing table and the destination address to find the route.

3. The packet is put in the appropriate output queue and waits its turn to be sent.

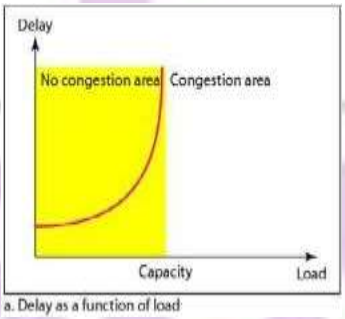


Awareness:

- First, if the rate of packet arrival is higher than the packet processing rate, the input queues become longer and longer.
- Second, if the packet departure rate is less than the packet processing rate, the output queues become longer and longer

NETWORKPER FORMANCE

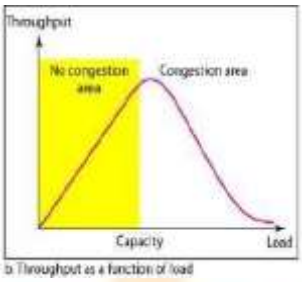
- Note that when the load is much less than the capacity of the network, the delay is at a minimum. This minimum delay is composed of propagation delay and processing delay, both of which are negligible.
- However, when the load reaches the network capacity, the delay increases sharply because we now need to add the waiting time in the queues (for all routers in the path) to the total delay. Note that the delay becomes infinite when the load is greater than the capacity
- Delay has a negative effect on the load and consequently the congestion.



a. Delay as a function of load

- We can define throughput in a network as the number of packets passing through the network in a unit of time.
- Notice that when the load is below the capacity of the network, the throughput increases proportionally with the load. We expect the throughput to remain constant after the load reaches the capacity, but instead the throughput declines sharply.
- The reason is the discarding of packets by the routers. When the load exceeds the capacity, the queues become full and the routers have to discard some packets.

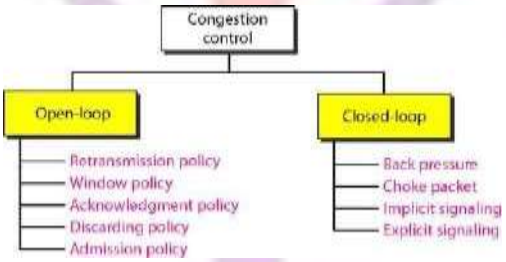
- Discarding packets does not reduce the number of packets in the network because the sources retransmit the packets, using time-out mechanisms, when the packets do not reach the destinations.



b. Throughput as a function of load

CONGESTION CONTROL

Congestion control refers to techniques and mechanisms that can either prevent congestion, 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).



OPEN LOOP CONTROL

- **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.

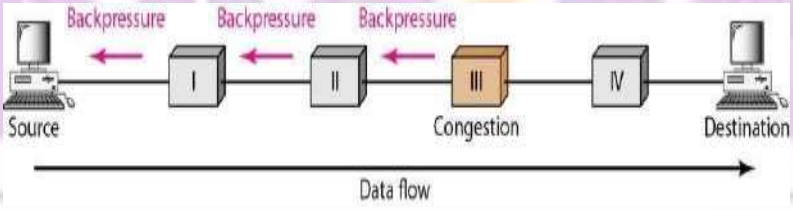
- **Acknowledgment Policy:** The acknowledgment policy imposed by the receiver may also affect congestion. If the receiver does not acknowledge every packet it receives, it may

slow down the sender and help prevent congestion. Several approaches are used in this case. 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. We need to know that the acknowledgments are also part of the load in a network. Sending fewer acknowledgments means imposing less load on the network.

- **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. For example, in audio transmission, if the policy is to discard less sensitive packets when congestion is likely to happen, the quality of sound is still preserved and congestion is prevented or alleviated.
- **Admission Policy:** An admission policy, which is a quality-of-service mechanism, can also prevent congestion 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 if there is a possibility of future congestion.

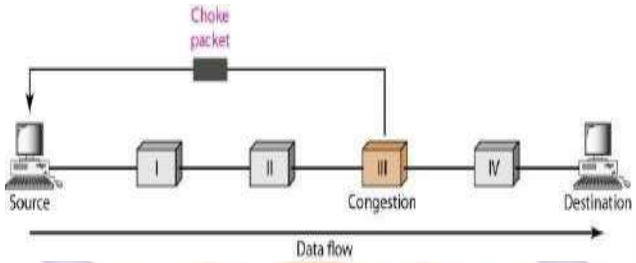
CLOSED LOOP CONTROL

- 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. The backpressure technique can be applied only to virtual circuit networks, in which each node knows the upstream node from which a flow of data is corning. Figure 24.6 shows the idea of backpressure.



- **Choke Packet:** A choke packet is a packet sent by a node to the source to inform it of congestion. Note the difference between the backpressure and choke packet methods.
- In backpressure, the warning is from one node to its upstream node, although the warning may eventually reach the source station.

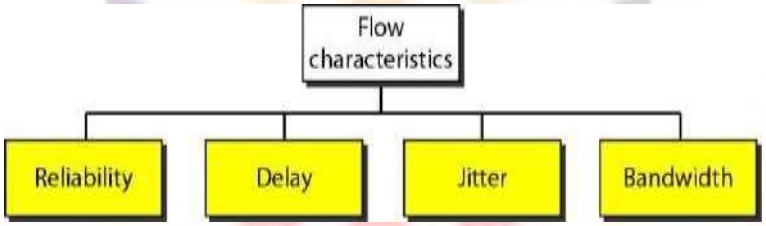
- In the choke packet method, the warning is from the router, which has encountered congestion, to the source station directly. The intermediate nodes through which the packet has traveled are not warned.



- **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. The explicit signaling method, however, is different from the choke packet method.
- In the choke packet method, a separate packet is used for this purpose; in the explicit signaling method, the signal is included in the packets that carry data.
- Explicit signaling, congestion control, can occur in either the forward or the backward direction.
- **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.

QUALITY OF SERVICE

Quality of service (QoS) is an internetworking issue that has been discussed more than defined. We can informally define quality of service as something a flow seeks to attain. The QoS can be judged by flow characteristics and flow class. The following fig. shows different flow characteristics.



- **Reliability:** Reliability is a characteristic that a flow needs. Lack of reliability means losing a packet or acknowledgment, which entails retransmission. However, the sensitivity of application programs to reliability is not the same.
- **Delay:** Source-to-destination delay is another flow characteristic. Again applications can tolerate delay in different degrees.
- **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.
    - For example, if four packets depart at times 0, 1, 2, 3 and arrive at 20, 21, 22, 23, all have the same delay, 20 units of time. On the other hand, if the above four packets arrive at 21, 23, 21, and 28, they will have different delays: 21,22, 19, and 24.
- **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.
- **Flow Classes-** based flow characteristic, we can classify flows into groups, with each group similar characteristics. These classes will be used by other protocols such as ATM (Not there in syllabus)

1 *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).
- IP was originally designed for best-effort delivery. Which means that every user receives the same level of services. This type of delivery does not guarantee the minimum of a service, such as bandwidth, to applications such as real-time audio and video
- If such an application accidentally gets extra bandwidth, it may be detrimental to other applications, resulting in congestion.
- Integrated Services, sometimes called IntServ, is afiow-based QoS model, which means that a user needs to create a flow, a kind of virtual circuit, from the source to the destination
- Signaling: IP is a connectionless, datagram, packet-switching protocol. we can implement a flow-based model over a IP signaling protocol 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.

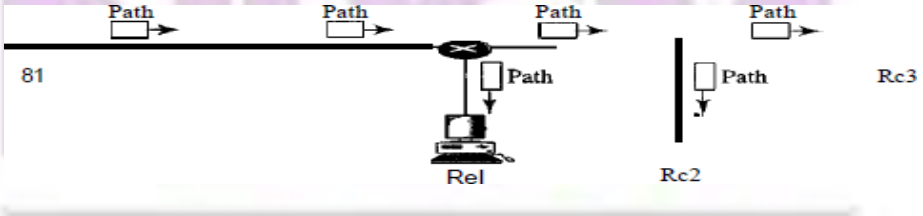
Service Classes

- 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.
- Only the first, the sum of the delays in the routers, can be guaranteed by the router. This type of service guarantees the packets within delivery time. (sum of delays of routers= Tspec)
- 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.

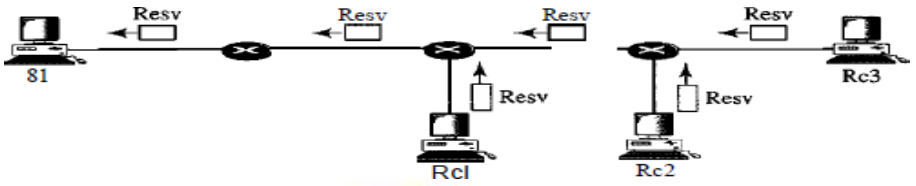
- Good examples of these types of applications are file transfer, e-mail, and Internet access. The controlled load service is a qualitative type of service in that the application requests the possibility of low-loss or no-loss packets.

RSVP

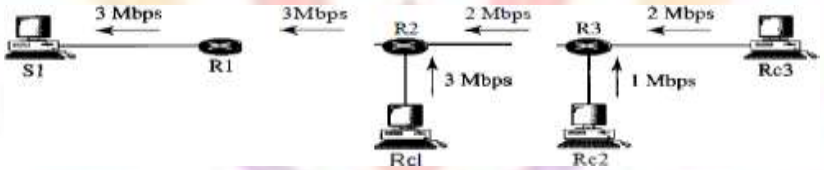
- The Resource Reservation Protocol (RSVP) is a signaling protocol to help IP create a flow and consequently make a resource reservation.
- Multicast Trees: RSVP is different signaling system designed for multicasting. It can also used for unicasting. The reason for this design is to enable RSVP to provide 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
- RSVP Messages: RSVP has several types of messages. However, for our purposes, we discuss only two of them: Path and Resv.
- Path Messages: Receivers make the reservation using 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 below shows path messages.



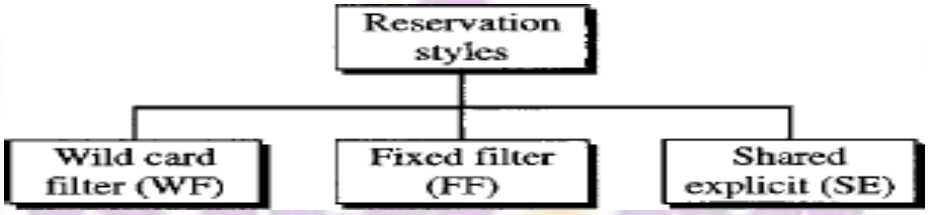
- Resv Messages: After a receiver has received a Path message, it sends a Resv message.
- The Resv message travels toward the sender (upstream) and makes a resource reservation on the routers that support RSVP.
- If a router does not support RSVP on the path, it routes the packet based on the best- effort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



- Reservation Merging: In RSVP, the resources are not reserved for each receiver in a flow; the reservation is merged.
- In Figure, Rc3 requests a 2-Mbps bandwidth while Rc2 requests a I-Mbps bandwidth. Router R3, which needs to make a bandwidth reservation, merges the two requests.
- The reservation is made for 2 Mbps, the larger of the two, because a 2-Mbps input reservation can handle both requests. The same situation is true for R2.



- 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, as shown in Fig.



- 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. 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.

2 *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.

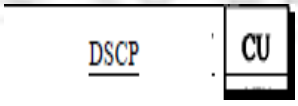
-The routers do not have to store information about flows. The applications, or hosts, define the type of service they need each time they send a packet.

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.

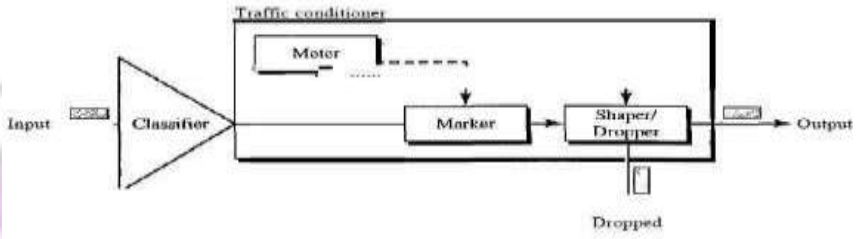
DS Field

- 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 as shown in Fig.



- 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 services like 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, as shown in Fig.

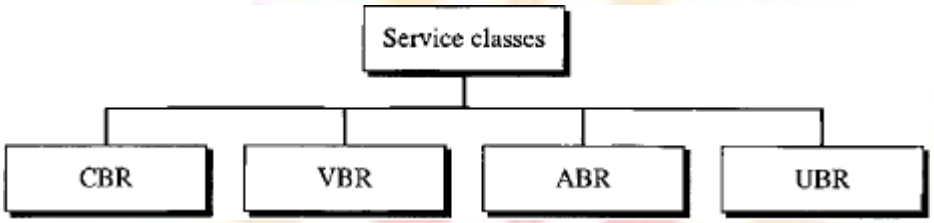


- Meters: The meter checks to see if the incoming flow matches the negotiated traffic profile. The meter also sends this result to other components. The meter can use several tools such as a token bucket to check the profile.
- Marker: A marker can remark a packet that is using best-effort delivery (OSCP: 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.

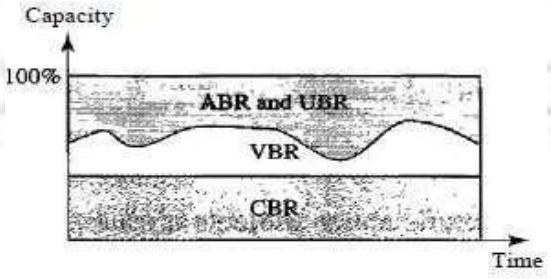
- Dropper: A dropper, which works as a shaper with no buffer, discards packets if the flow severely violates the negotiated profile.

QoS in ATM

- The QoS in ATM is based on the class, user-related attributes, and network-related attributes.
- The ATM Forum defines four service classes: CBR, VBR, ABR, and UBR



- CBR The constant-bit-rate (CBR) class is designed for customers who need realtime audio or video services.
- VBR The variable-bit-rate (VBR) class is divided into two subclasses: real-time(VBR- RT) and non-real-time (VBR-NRT).
- VBR-RT is designed for those users who need real-time services (such as voice and video transmission) and use compression techniques to create a variable bit rate. VBR-NRT is designed for those users who do not need real-time services but use compression techniques to create a variable bit rate.
- ABR The available-bit-rate (ABR) class delivers cells at a minimum rate. If more network capacity is available, this minimum rate can be exceeded. ABR is particularly suitable for applications that are bursty.
- UBR The unspecified-bit-rate (UBR) class is a best-effort delivery service that does not guarantee anything. Below fig. Show relationship between different classes to the capacity of the network



ATM Attributes

- ATM defines two sets of attributes. User-related attributes and network related attributes.
- User-related attributes are those attributes that define how fast the user wants to send data. These are negotiated at the time of contract between a user and a network. Few of user-related attributes are given
- SCR The sustained cell rate (SCR) is the average cell rate over a long time interval. The actual cell rate may be lower or higher than this value, but the average should be equal to or less than the SCR
- PCR The peak cell rate (PCR) defines the sender's maximum cell rate. The user‘s cell rate can sometimes reach this peak, as long as the SCR is maintained.
- MCR The minimum cell rate (MCR) defines the minimum cell rate acceptable to the sender.
- CVDT The cell variation delay tolerance (CVDT) is a measure of the variation in cell transmission times.
- Network-Related Attributes: The network-related attributes are those that define characteristics of the network. The following are some network-related attributes
- CLR The cell loss ratio (CLR) defines the fraction of cells lost (or delivered so late that they are considered lost) during transmission.

$$CLR = \frac{1}{100} = 10^{-2}$$

- CTD The cell transfer delay (CTD) is the average time needed for a cell to travel from source to destination. The maximum CTD and the minimum CTD are also considered attributes.
- CDV The cell delay variation (CDV) is the difference between the CTD maximum and the CTD minimum.
- CER The cell error ratio (CER) defines the fraction of the cells delivered in error.