An undertaking of Bhaktapur Municipality
Khwopa College of Engineering
Affiliated to Tribhuvan Univeristy
Libali, Bhaktapur, Nepal

A
Note
on
Computer Network

Compiled By

Er. Dinesh Ghemosu

Table of Contents

5 Transport Layer **1**

5.1 Function of Transport Layer . . . 1
5.2 Function Provided the Upper Layer . . . 1
5.3 Why Transport Layer? . . . 2
5.4 Transport Service Primitives . . . 2
5.5 Introduction and Transport Layer Services . . . 2
5.6 Transport Layer in Internet . . . 3
5.7 Port Address . . . 3
5.8 Socket Address . . . 4
5.9 Muiltiplexing and Demultiplexing . . . 4
5.9.1 How demultiplexing works? . . . 4
5.9.2 Connectionless Multiplexing and Demultipexing . . . 5
5.9.3 connection-oriented Demux . . . 5
5.10 Connectionless versus Connection-oriented Service . . . 7
5.10.1 Connectionless Transport: UDP . . . 7
5.10.1.1 User Datagram . . . 8
5.10.1.2 UDP Features . . . 8
5.10.1.3 Typically Application . . . 9
5.10.2 Connection-Oriented Transport:TCP . . . 9
5.10.2.1 TCP services . . . 9
5.10.2.2 TCP Header Format . . . 12
5.10.2.3 A TCP Connection . . . 13
5.10.2.4 TCP and Flow Control . . . 15
5.10.2.5 Windowing and Window Size . . . 15
5.11 Stream Control Transmission Protocol . . . 15
5.11.1 SCTP Services . . . 15
5.11.1.1 Some SCTP Applications . . . 17
5.12 Congestion in TCP . . . 17
5.12.1 Causes of Congestion . . . 18
5.12.2 Approaches of Congestion Control . . . 18
5.13 Traffic Shaping Algorithms . . . 20
5.13.1 Leaky Bucket Algorithm . . . 20
5.13.2 Tocket Bucket Algorithm . . . 21
5.14 Socket Programming . . . 22
5.14.1 Socket Programming with UDP . . . 23
5.14.2 Socket Programming with TCP . . . 23

5

Transport Layer

5.1 Function of Transport Layer

- Error Handling
- Flow control
- Multiplexing
- Connection set-up and Release
- Congestion Handling
- Segmentation an Reassembly
- Port Addressing

5.2 Function Provided the Upper Layer

-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/hardware within the transport layer that does the work is **transport entity**.

- The transport entity can be located in the operating system kernel, in a library package bound into the network applications, in a separate user process, or even on the network interface card. The first two options are most common on the Internet.

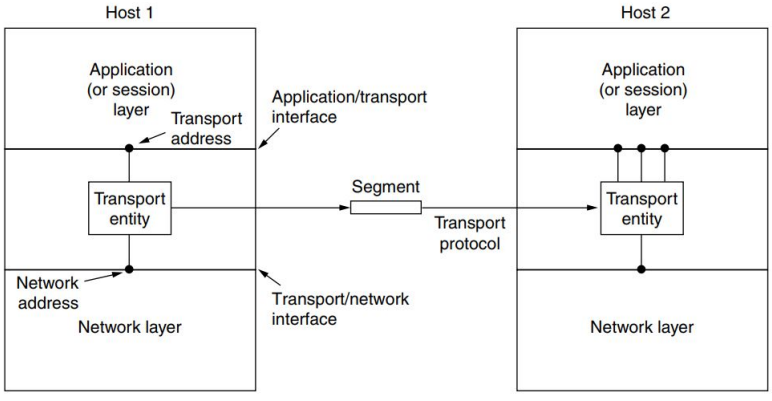

Figure 5.1: The network, transport and application layers.

5.3 Why Transport Layer?

- Networks is under the control of a network operator. If anything goes wrong (such as packets loss, router down etc.) on network, a user has no control over it. So, the transport layer add reliability of the service provided by network.
- Furthermore, the transport primitives can be implemented as calls to library procedure to make them independent of the network primitives.
- In addition, application programmers can write code according to a standard set of primitives and have these programs work on a wide variety of networks, without having to worry about dealing with different network interfaces and levels of reliability.
- As a user, you may want to send and receive email, browse the web, login to another host. So you may want to run several programs or processes at a time. The transport layer allows for processes or applications to communicate with each other.

5.4 Transport Service Primitives

- To allow users to access the transport service, the transport layer must provide some operations to application programs, that is, a transport service interface.
- Each transport service has its own interface.
- The transport service is similar to the network service, but there are also some important differences.
- The main difference is that the network service is intended to model the service offered by real networks.

- Real networks can lose packets, so the network service is generally unreliable.
- The connection-oriented transport service, in contrast, is reliable. Of course, real networks are not error-free, but that is precisely the purpose of the transport layer—to provide a reliable service on top of an unreliable network.
- Network service is used by transport entitles. Application programmer barely sees network primitive while they can see the transport primitives.

Table 5.1: The primitives for a simple transport service.

Primitives	Packet Sent	Meaning
LISTEN	(none)	Block until some process tries to connect
CONNECT	CONNECTION REQUEST	Actively attempt to establish a connection
SEND	DATA	Send information
RECEIVE	(none)	Block until a DATA packet arrive
DISCONNECT	DISCONNECT REQUEST	Request a release of the connection

5.5 Introduction and Transport Layer Services

- A transport layer protocol provide for logical communication between application processes running on different hosts.
- On the sending side, the transport layer converts the application-layer messages it receives from a sending application process into transport-layer packets, known as transport-layer segments in Internet terminology.
- On the receiving side, the transport layer reassembles segments into messages, passes to application layer.
- More than one transport-layer protocol may be available to network applications. For example, the Internet has two protocols—TCP and UDP. Each of these protocols provides a different set of transport-layer services to the invoking application.

- Reliable, in –order unicast delivery (TCP)
  - Connection setup
  - Congestion control
  - Flow control
- Unreliable (“best effort”), unordered unicast or multicast delivery: UDP
- Service not available:
  - Real-time
  - Bandwidth guarantees

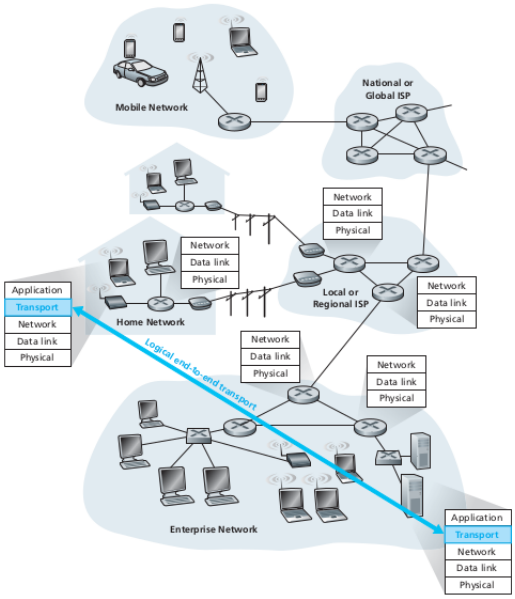

Figure 5.2: The transport layer provides logical rather than physical communication between apllication processes.

– Reliable multicast

Relation between Transport Layer and Network Layer
- **Network Layer**: provides logical communication between hosts
- **Transport Layer**: provide Logical communication between processes

5.6 Transport Layer in Internet

- Generally, a TCP/IP network makes three distinct transport-layer protocols available to the application layer: TCP, UDP and SCTP.
  - **TCP** (Transmission Control Protocol), which provides a reliable, connection-oriented service to the invoking application.
  - **UDP** (User Datagram Protocol), which provides an unreliable, connectionless service to the invoking application.
  - **Stream Control Protocol** : has properties of both TCP and UDP
- In an Internet context, we refer to the transport layer packet as a segment. More specifically, we refer to the transport layer packet for TCP as a segment but often refer to the packet for UDP as a datagram.

5.7 Port Address

- That transport-layer multiplexing requires :
  - that sockets have unique identifiers, and
  - that each segment have special fields that indicate the socket to which the segment is to be delivered.
- These special fields are the source port number field and the destination port number field.
- Port number are the 16 bit integers between 0 and 65,535.
- The IANA (Internet Assigned Number Authority) has divided the port numbers into three ranges.

1. **Well-known port numbers**
   - These are the universal port numbers used by the server ranging form 0 to 1023.
   - Every client process knows the well-known port number of the corresponding server process.

2. **Registered port numbers**
   - Port number ranges from 1024 to 49,151.
   - hese port numbers are assigned to user processes and or applications.

3. **Dynamic port numbers** - These are the port numbers defined the client program temporarily (for

the duration of the request and its completion).
- They are chosen randomly by the transport layer software running on the client host when initiating a connection.
- Port numbers are greater than 49,151.

Port number	Process name	Protocol used	Description
20	FTP-DATA	TCP	File transfer—data
21	FTP	TCP	File transfer—control
22	SSH	TCP	Secure Shell
23	TELNET	TCP	Telnet
25	SMTP	TCP	Simple Mail Transfer Protocol
53	DNS	TCP and UDP	Domain Name System
69	TFTP	UDP	Trivial File Transfer Protocol
80	HTTP	TCP and UDP	Hypertext Transfer Protocol
110	POP3	TCP	Post Office Protocol 3
123	NTP	TCP	Network Time Protocol
143	IMAP	TCP	Internet Message Access Protocol
443	HTTPS	TCP	Secure implementation of HTTP

Figure 5.3: Some well-known port numbers.

5.8 Socket Address

- The transport layer protocol in the TCP/IP suite needs two identifiers: IP address and port numbers, at each end to make a make a process-to-process 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 sever socket address defines the server process uniquely.

5.9 Muiltiplexing and Demultiplexing

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 many-to-one relationship and requires multiplexing.
- The protocol accepts messages from different processes through sockets, 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, through the sockets.

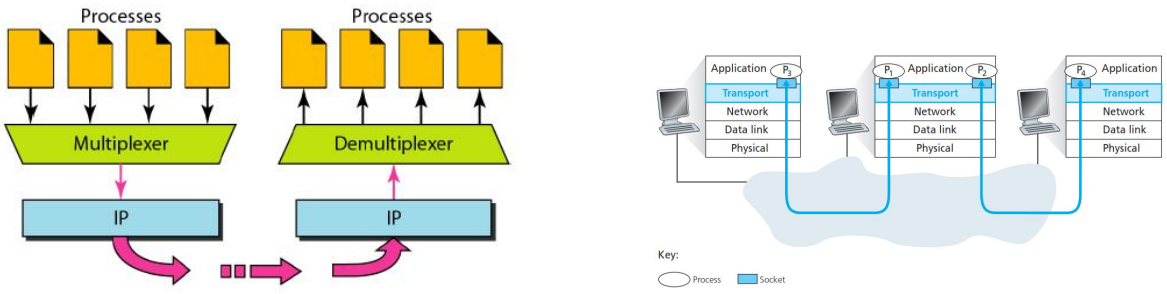

Figure 5.4: Transport layer multiplexing and demultiplexing.

5.9.1 How demultiplexing works?

- Host receives IP datagrams.
  - each datagram has source IP address, destination IP address
  - each datagram carries one transport-layer segment

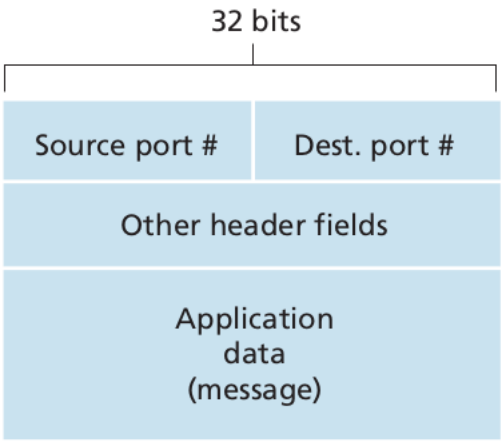

Figure 5.5: Source and destination port-number fields in a tranport-layer segment.

- each segment has source, destination port number
- Host uses IP addresses & port numbers to direct segment to appropriate socket.

5.9.2 Connectionless Multiplexing and Demultipexing

- The python program running in a host can create a UDP socket with the line
  $clientSocket = socket(socket.AF_INET, socket.SOCK_DGRAM)$
- When a UDP socket is created in this manner, the transport layer automatically assigns a port number to the socket.
- Alternatively, we can add a line into our Python program after we create the socket to associate a specific port number (say, 19157) to this UDP socket via the socket bind() method:
  $clientSocket.bind(('', 19157))$
- Typically, the client side of the application lets the transport layer automatically (and transparently) assign the port number, whereas the server side of the application assigns a specific port number.

Example:
Suppose a process in Host A, with UDP port 19157, wants to send a chunk of application data to a process with UDP port 46428 in Host *B.*

- The transport layer in Host A creates a transport-layer segment that includes the application data, the source port number (19157), the destination port number (46428), and two other.
- The transport layer then passes the resulting segment to the network layer. The network layer encapsulates the segment in an IP datagram and makes a best-effort attempt to deliver the segment to the receiving host.
- If the segment arrives at the receiving Host B, the transport layer at the receiving host examines the destination port number in the segment (46428) and delivers the segment to its socket identified by port 46428.
- Note that Host B could be running multiple processes, each with its own UDP socket and associated port number. As UDP segments arrive from the network, Host B directs (demultiplexes) each segment to the appropriate socket by examining the segment's destination port number.

Things to Remember
- Note that a UDP socket is fully identified by a two-tuple consisting of a destination IP address and a destination port number.
- As a consequence, if two UDP segments have different source IP addresses and/or source port numbers, but have the same destination IP address and destination port number, then the two segments will be directed to the same destination process via the same destination socket.

5.9.3 connection-oriented Demux

- TCP socket identified by 4-tuple:
  - source IP address
  - source port number

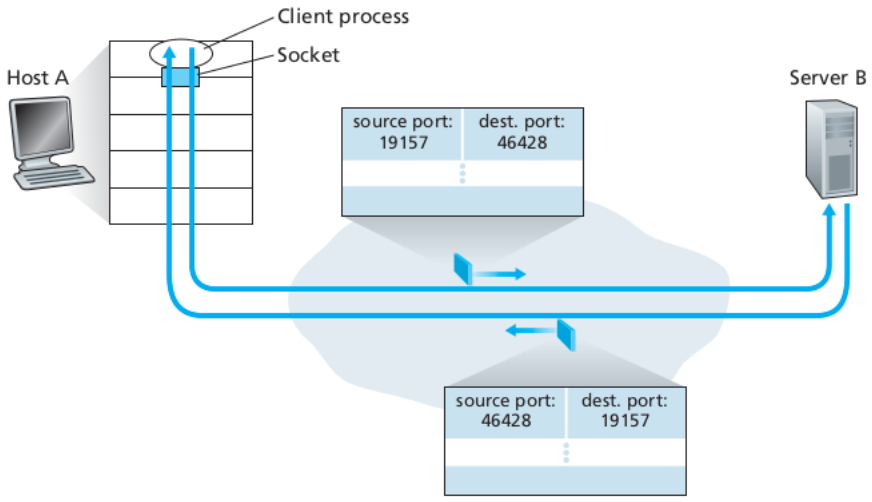

Figure 5.6: The inversion of source and destination port numbers.

- dest IP address
- dest port number

- demux: receiver uses all four values to direct segment to appropriate socke erver host may support many simultaneous TCP sockets:
- each socket identified by its own 4-tuple
- web servers have different sockets for each connecting client
  - non-persistent HTTP will have different socket for each request

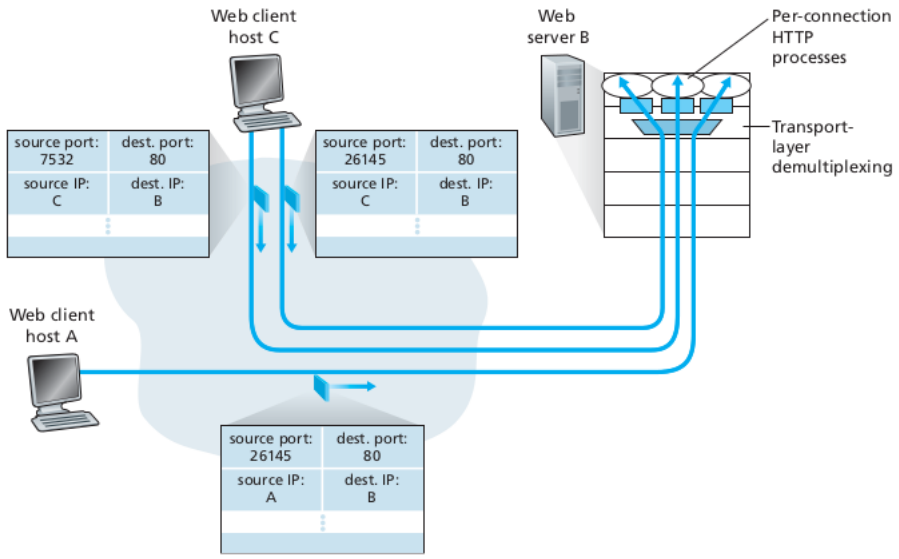

Figure 5.7: Two clients, using the same destination port number (80) to communicate with the same Web server application.

- The server host may support many simultaneous TCP connection sockets, with each socket attached to a process, and with each socket identified by its own four- tuple. When a TCP segment arrives at the host, all four fields (source IP address, source port, destination IP address, destination port) are used to direct (demultiplex) the segment to the appropriate socket.
- The situation is illustrated in Figure [**?**] in which Host C initiates two HTTP sessions to server B, and Host A initiates one HTTP session to B. Hosts A and C and server B each have their own unique IP address—A, C, and B, respectively. Host C assigns two different source port numbers (26145 and 7532) to its two HTTP connections. Because Host A is choosing source port numbers independently of C, it might also assign a source port of 26145 to its HTTP connection. But this is not a problem—server B will still be able to correctly demultiplex the two connections having the same source port number, since the two connections have different source IP addresses.

Table 5.2: Differences between connectionless and connection-oriented services

Connection Oriented Service	Connectionless Service
1. Connection is established between sender and receiver prior to data transfer.	1. No connection is established between sender and receiver prior to data transfer.
2. Acknowledgment is sent to the sender after receiving the packet by the receiver.	2. No acknowledgement
3. Guarantee the delivery of packets. Loss packets are retransmitted.	3. No guarantee of delivery of packets. Lost packets are not known or retransmitted.
4. Reliable ; use flow and error control at transport layer	4. Unreliable: no flow control and error control
6. Slower and complex service	6. Faster service
7. Example: TCP, SCTP	7. Example: UDP

Table 5.3: Differences between UDP and TCP.

UDP	TCP
1. Datagram oriented.	Stream Oriented.
2. Unreliable, connectionles.	2. Reliable, Connection-oriented.
3. Simple	3. Complex
4. Unicast and multicast	4. Only unicast.
5. No windows or ACKs	5. Uses windows or ACKs.
6. Smaller header, less overhead.	6. Full header
7. No sequencing.	7. Sequencing.
8. Useful only for few applications, e.g. mutimedia applications	8. Used for most internet applications
9. Network management (SNMP), routing (RIP), naming (DNS)	9. Web (HTTP), email (SMPT), file transfer (FTP), (telnet) etc.

5.10 Connectionless versus Connection-oriented Service

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

Connection oriented Services

- In connection oriented service, a connection is first established between the sender and receiver. Then data are transferred. At the end, the connection is released.
- In this type of transmission the receiving device sends an acknowledgment, back to the source after a packet or group of packet is received.
- Reliable but slower and extra overhead is required.
- Example: TCP (Transmission control protocol)

Connectionless Services

- In this type of service, the packets are send from one party to another with no need for connection establishment and connection release.
- The packets are not numbered; they may be delayed or lost or arrived out of order. - There is no acknowledgement either.
- Unreliable, but faster and less overhead is required.
- Example: UDP (User datagram protocol)

5.10.1 Connectionless Transport: UDP

- UDP is the connectionless transport protocol in the TCP/IP protocol stack.
  - No connection is established before data transfer.
  - User datagram (segment) sent by UDP is an independent.
- UDP is a simple protocol that exchanges datagram without guaranteed delivery. It relies on higher layer protocols to handle error and retransmit data.
  - No flow error control
  - No retransmission
  - No acknowledgments

Application	Application-Layer Protocol	Underlying Transport Protocol
Electronic mail	SMTP	TCP
Remote terminal access	Telnet	TCP
Web	HTTP	TCP
File transfer	FTP	TCP
Remote file server	NFS	Typically UDP
Streaming multimedia	typically proprietary	UDP or TCP
Internet telephony	typically proprietary	UDP or TCP
Network management	SNMP	Typically UDP
Routing protocol	RIP	Typically UDP
Name translation	DNS	Typically UDP

Figure 5.8: Popular Internet applications and their underlying tranport protocols.

- UDP doesn't use window or Acknowledgement. UDP is designed for applications that do not need to put sequence of segments together.
- The following application layer protocols use UDP: TFTP, SNMP, DHCP, and DNS
Hence,

- Used in transport layer
- Offers unreliable connectionless service
- Provides faster service than that of TCP.
- Offers minimum error checking mechanism.
- Supports multicasting because connectionless.
- Also used by SNMP (Simple Network Management Protocol)

5.10.1.1 User Datagram

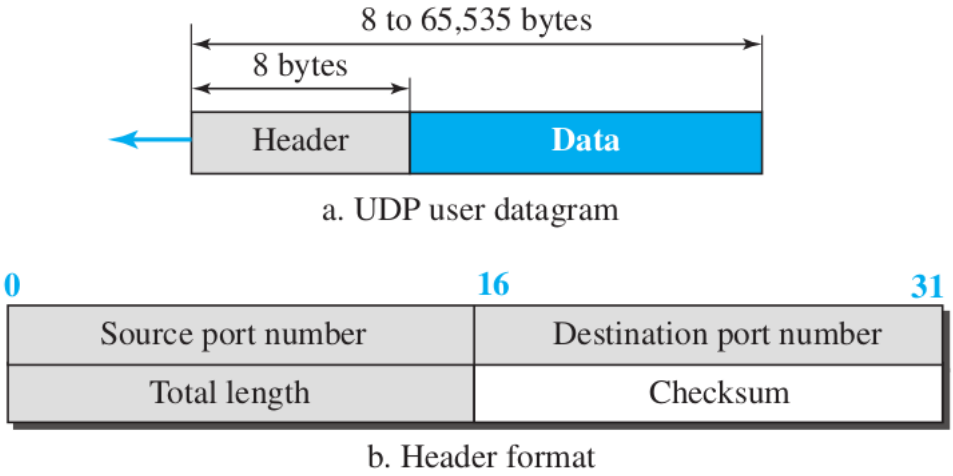

Figure 5.9: User datagram format.

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

- **Source Port**: This is the port number of the application that is originating the user data.
- **Destination Port**: This is the port number pertaining to the destination application.
- **Length**: This field describes the total length of the UDP datagram, including both data and header Information.
- **UDP Checksum**: optional field. Used to detect errors over the entire user datagram

5.10.1.2 UDP Features

Connectionless-Serive

UDP is a connectionless protocol. Each UDP packet is independent from other packets sent by the same appli-

cation program. This feature can be considered as an advantage or disadvantage depending on the application requirements. It is an advantage if, for example, a client application needs to send a short request to a server and to receive a short response. If the request and response can each fit in a single user datagram, a connectionless service may be preferable. The overhead to establish and close a connection may be significant.The connectionless service provides less delay; the connection-oriented service creates more delay. If delay is an important issue for the application, the connectionless service is preferred.
Example:A client-server application such as DNS (see Chapter 26) uses the services of UDP because a client needs to send a short request to a server and to receive a quick response from it. The request and response can each fit in one user datagram. Since only one message is exchanged in each direction, the connectionless feature is not an issue; the client or server does not worry that messages are delivered out of order

Lack of Error Control
UDP does not provide error control; it provides an unreliable service. Most applications expect reliable service from a transport-layer protocol. Although a reliable service is desirable, it may have some side effects that are not acceptable to some applications. When a transport layer provides reliable services, if a part of the message is lost or corrupted, it needs to be resent. This means that the receiving transport layer cannot deliver that part to the application immediately; there is an uneven delay between different parts of the message delivered to the application layer.
Example:Assume we are downloading a very large text file from the Internet. We definitely need to use a transport layer that provides reliable service. We don't want part of the file to be missing or corrupted when we open the file. The delay created between the deliveries of the parts is not an overriding concern for us; we wait until the whole file is composed before looking at it. In this case, UDP is not a suitable transport layer.

Lack of Congestion Control
UDP does not provide congestion control. However, UDP does not create additional traffic in an error-prone network. TCP may resend a packet several times and thus contribute to the creation of congestion or worsen a congested situation. Therefore, in some cases, lack of error control in UDP can be considered an advantage when congestion is a big issue.

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

- UDP is suitable for a process that requires simple request-response communication with little concern for flow and error control, such as DNS. It is not usually used for a process such as FTP that needs to send bulk data.
- 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.
- UDP is a suitable transport protocol for multicasting. Multicasting capability is embedded in the UDP software but not in the TCP software.
- UDP is used for management processes such as SNMP .
- UDP is used for some route updating protocols such as Routing Information Protocol (RIP) .
- UDP is normally used for interactive real-time applications that cannot tolerate uneven delay between sections of a received message .

5.10.2 Connection-Oriented Transport:TCP
- The Transmission Control Protocol (TCP) is a core protocol of the Internet Protocol Suite.
- 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.

5.10.2.1 TCP services
TCP provides follwing services:

- Connection-oriented
- Byte Stream deliver
- Full duplex communication
- Segmentation
- Sequence number
- Acknowledgment number
- Flow control and error control
- Congestion control

Connectin-Oriented
- Connection oriented means that a virtual connection is established before any user data is transferred. The following occurs:

1. The two TCPs establish a connection between them.
2. Data are exchanged in both direction.
3. The connection is terminated.

- If the connection cannot be established,the user program is notified.
- If the connection is ever interrupted,the user program(s) is notified.

Reliable
- TCP provides reliability using error control.
- Error control includes mechanisms for detecting and resending corrupted segments, resending lost segments, storing out-of- order segments until missing segments arrive, and detecting and discarding duplicated segments.
- Error control in TCP is achieved through the use of three simple tool: *Checksum, Acknowledgment, Time-out*.

Full Duplex Communcation
- TCP offers full duplex communication, in which data can flow in both direction at the same time.
- Each TCP then has a sending and receiving buffer, and segments move in both direction.

Stream Delivery Service
- TCP, unlike UDP, is a stream-oriented protocol. That is, TCP allows the sending process to deliver data as stream of bytes and allows the receiving process to obtain as a stream of bytes.
- TCP creates an environment in which the two processes seem to be connected by an imaginary "tube" that carries their bytes across the Internet. This imaginary environment is depicted in Figure 5.10. The sending process produces (writes to) the stream and the receiving process consumes (reads from) it.

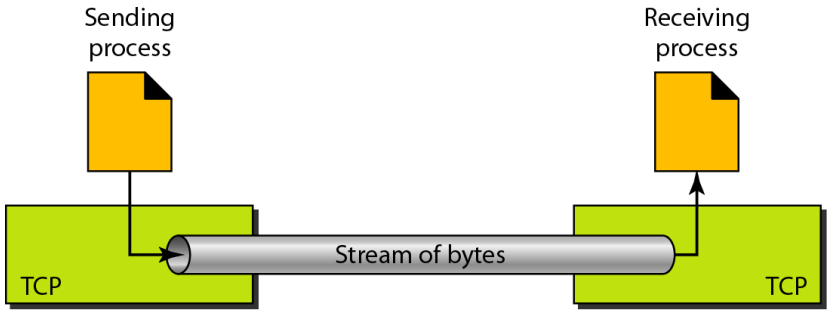

Figure 5.10: Stream delivery in TCP.

Segments
- The IP layer, as a service provider for TCP, needs to send data in packets, not as a stream of bytes.
- At transport layer, TCP groups a number of bytes together into a packet called a segment, and assign a sequence number to each segment.
- TCP adds a header to each segment and delivers the segment to the IP layer for transmission. The segments are encapsulated in IP datagrams and transmitted.

Example:Suppose a TCP connection is transferring a file of 5000bytes. The first byte is numbered 10,001. what

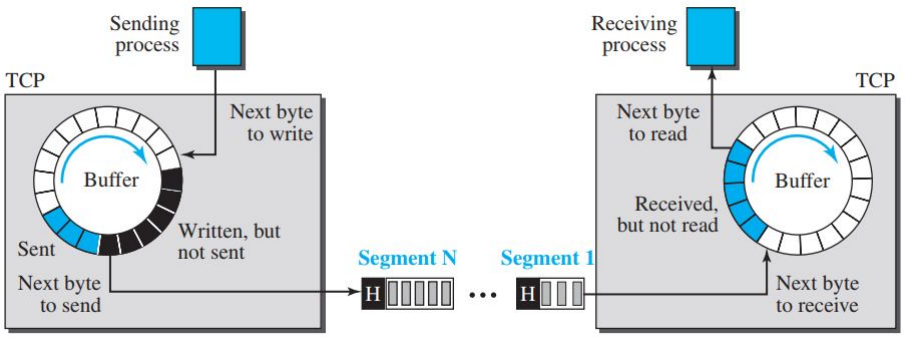

Figure 5.11: Sending and receving buffers.

are the sequence numbers for each segment if data are sent in five segments, each carrying 1000 bytes?
Solution:The following shows the sequence number for each segment:

Segment 1 → Sequence Number: 10,001 (range: 10,001 to 11,000)
Segment 2 → Sequence Number: 11,001 (range: 11,001 to 12,000)
Segment 3 → Sequence Number: 12,001 (range: 12,001 to 13,000)
Segment 4 → Sequence Number: 13,001 (range: 13,001 to 14,000)
Segment 5 → Sequence Number: 14,001 (range: 14,001 to 15,000)

Sequence Number and Acknowledgment Number
Sequence Number
- After the byes have been numbered, TCP assigns a sequence number to each segment that is being sent.
- The value in the sequence number field of a segment defines the number of the first data byte contained in that segment.

Acknowledgment Number
- After receiver receives the bytes, it sends acknowledgment number to the sender. - However, 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. That is if a party uses 5643 as an acknowledgment number, it has received all bytes from the beginning up to 5642. however this does not mean the party has received 5642 bytes because the first byte does not necessarily have to start from 0.

Flow Control
- A TCP connection sets aside a receiver buffer for the connection. When the TCP connection receives bytes that are correct and in sequence, it places data in the receiver buffer. The associated application process will read data from this buffer, but not necessarily at this instant that data arrives. Indeed, the receiving application may be busy with some other task and may not attempt to read the data until longer after it has arrived. If the application is relatively slow at reading data the sender can very easily overflow the receive buffer by sending too much data quickly.
- TCP provides a flow control service to its application to estimate the possibility of the sender overflowing the receive buffer. Flow control is thus a speed-matching service the rate at which the sender is sending against the rate at which receiving application is receiving.
- Flow control at this layer is performed end-to-end rather than across a single link.
- A sliding window is used to make data transmission more efficient as well as control the flow of data so that the receiver does not become overwhelmed.

Error Control
- TCP provides reliability using error control. Error control includes mechanisms for detecting and resending corrupted segments, resending lost segments, storing out-of- order segments until missing segments arrive, and

detecting and discarding duplicated segments.
- Error control in TCP is achieved through the use of three simple tools: *checksum, acknowledgment, and time-out*.

5.10.2.2 TCP Header Format

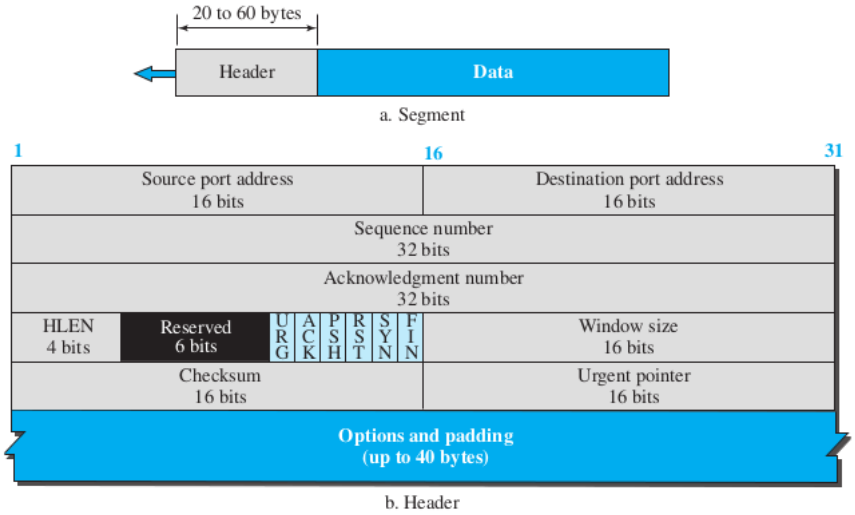

Figure 5.12: TCP Segment Header.

- **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.
- **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.
- **Sequence number**: This 32-bit field defines the number assigned to the first byte of data contained in this segment. As we said before, TCP is a stream transport protocol. To ensure connectivity, each byte to be transmitted is numbered. The sequence number tells the destination which byte in this sequence is the first byte in the segment. During connection establishment (discussed later) each party uses a random number generator to create an initial sequence number (ISN), which is usually different in each direction.
- **Acknowledgment number**. This 32-bit field defines the byte number that the receiver of the segment is expecting to receive from the other party. If the receiver of the segment has successfully received byte number x from the other party, it returns x + 1 as the acknowledgment number. Acknowledgment and data can be piggybacked together.
- **Header length**. This 4-bit field indicates the number of 4-byte words in the TCP header. The length of the header can be between 20 and 60 bytes. Therefore, the value of this field is always between 5 ($5 \times 4 = 20$) and 15 ($15 \times 4 = 60$).
- **Control**: This field defines 6 different control bits or flags, as shown in Figure 5.13. One or more of these bits can be set at a time. These bits enable flow control, connection establishment and termination, connection abortion, and the mode of data transfer in TCP. A brief description of each bit is shown in the figure. We will discuss them further when we study the detailed operation of TCP later in the chapter.
- **Window size:** This field defines the window size of the sending TCP in bytes. Note that the length of this field is 16 bits, which means that the maximum size of the window is 65,535 bytes. This value is normally referred to as the receiving window (rwnd) and is determined by the receiver. The sender must obey the dictation of the receiver in this case.
- **Checksum**: This 16-bit field contains the checksum. The calculation of the checksum for TCP follows the same procedure as the one described for UDP. However, the use of the checksum in the UDP datagram is optional, whereas the use of the checksum for TCP is mandatory. The same pseudoheader, serving the same purpose, is added to the segment.

Encapsulation
A TCP segment encapsulates the data received from the application layer. The TCP segment is encapsulated in an IP datagram, which in turn is encapsulated in a frame at the data-link layer.

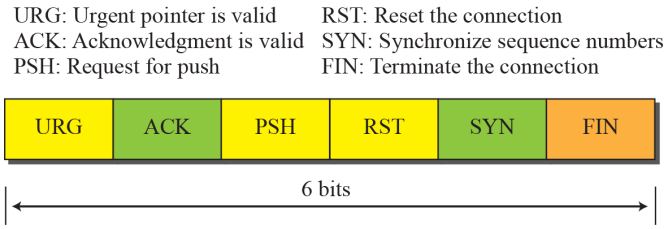

Figure 5.13: Control filed.

5.10.2.3 A TCP Connection

Connection Establishment

- TCP transmits data in full duplex mode.
- The connection establishment in TCP is called *three-way handshaking*.

Three-way Handshake

- A three-way-handshake is a method used in a TCP/IP network to create a connection between a local host/client and server.
- It is a three-step method that requires both the client and server to exchange SYN and ACK (acknowledgment) segment before actual data communication begins.
- A three-way-handshake is also known as a *TCP handshake*.

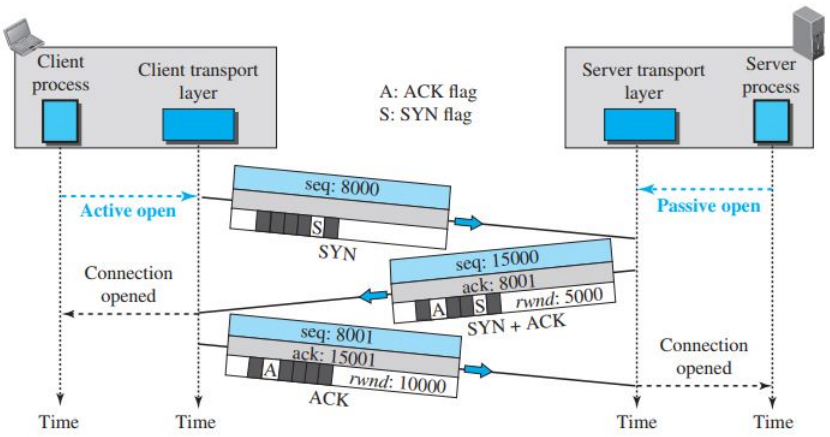

Figure 5.14: Connection establishment using three-way handshaking.

The three Steps:

Step 1:

- The client sends the first segment, a SYN segment, in which only the SYN flag is set.
- This segment is used for synchronization of a sequence number.
- It consumes one sequence number. It is incremented by 1 when data transfer starts. But, a SYN segment does not carry any data.
- The objective of this packet is to ask if the server is open for new connection.

Step 2:

- The target server must have open ports that can accept and initiate new connections.
- When the server receives the first SYN segment, then the server send the second segment, a SYN+ACK segment, with two flag bit set: SYN and ACK.
- A SYN+ACK segment cannot carry data, but consumes one sequence number.

Step 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.
- An ACK segment,, if carrying no data, consumes no sequence number.

Upon completion of this process, the connection is created and the host and server can communicate.

Data Transfer
- After connection is established, bidirectional data takes place.
- The client and server can both send data and acknowledgments.
- The acknowledgment is piggybacked with data.

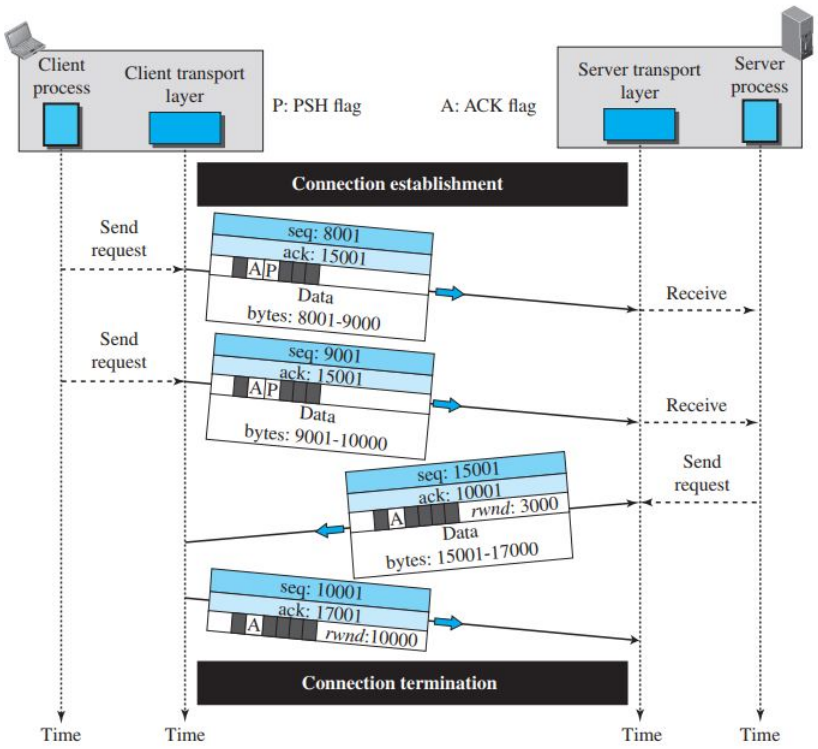

Figure 5.15: Data transfer in TCP.

- After connection is established, the client sends 2000 bytes of the data in two segment.
- The server sends 2000 bytes in one segment.
- The client sends one more segment.
- The first three segments carry both data and acknowledgment, but the last segment carries only an acknowledgment because there are no more data to be sent.

- The data segment sent by the client have the PSH (push) flag set so that the server TCP knows to deliver data to the server process as soon as they are received.

Connection Termination
- Either client or server can close the connection after exchange of data; but usually initiated by the client.
- Mostly it is implemented with three-way handshaking.

Step 1:
- 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.
- It can contain the last chunk of data sent by client.
- The FIN segment consumes one sequence number if it does not carry data.

Step 2:
- The sever TCP, after receiving the *FIN* segment, informs its process of the situation and sends the second segment, a *FIN+ACK* segment, to confirm the receipt of the FIN segment from the client and at the same time to announce the closing of the connection.
- This segment can also contained the last chunk of data from server.
- The *FIN+ACK* segment consumes one sequence number if it does not carry data.

Step 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 cannot carry data and consumes no sequence numbers.

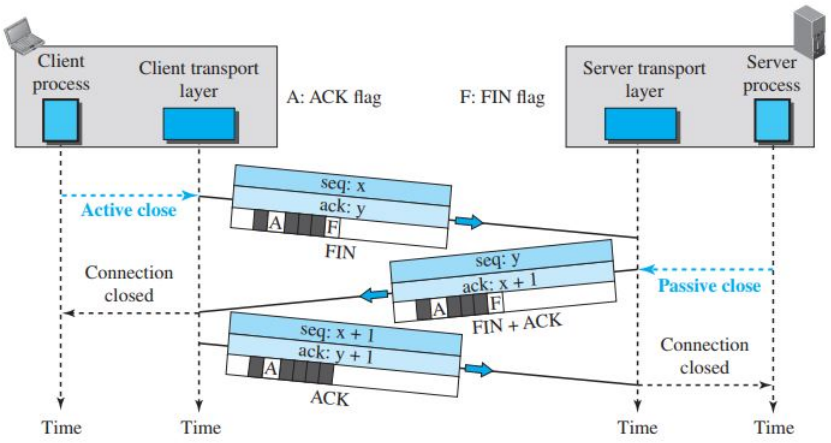

Figure 5.16: Connection Termination using three-way handshaking in TCP.

5.10.2.4 TCP and Flow Control
- Data often is too large to be sent in a single segment.
- TCP splits the data into multiple segments.
- TCP provides flow control through "windowing" to set the pace of how much data is sent at a time.
- How many bytes per window, and how many windows between ACKs.

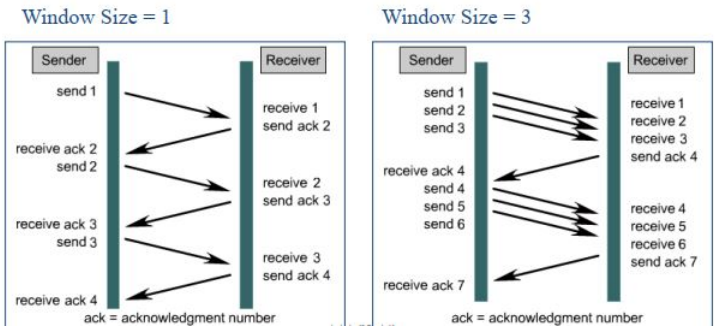

Figure 5.17: TCP and flow control.

5.10.2.5 Windowing and Window Size
- Sliding window refers to the fact that the window size is negotiated dynamically during the TCP.
- If the source receives no acknowledgment, it knows to retransmit at a slower rate.
- Figure 5.18.

5.11 Stream Control Transmission Protocol
- Stream Control Transmission Protocol (SCTP) is a new transport-layer protocol designed to combine some features of UDP and TCP in an effort to create a better pro- tocol for multimedia communication.

5.11.1 SCTP Services
- SCTP, like UDP or TCP, provides process-to-process communication.
- SCTP allows multistream service in each connecetion, which is called ***association***. If one stream is blocked, the other stream can still deliver their data. (Figure 5.19).

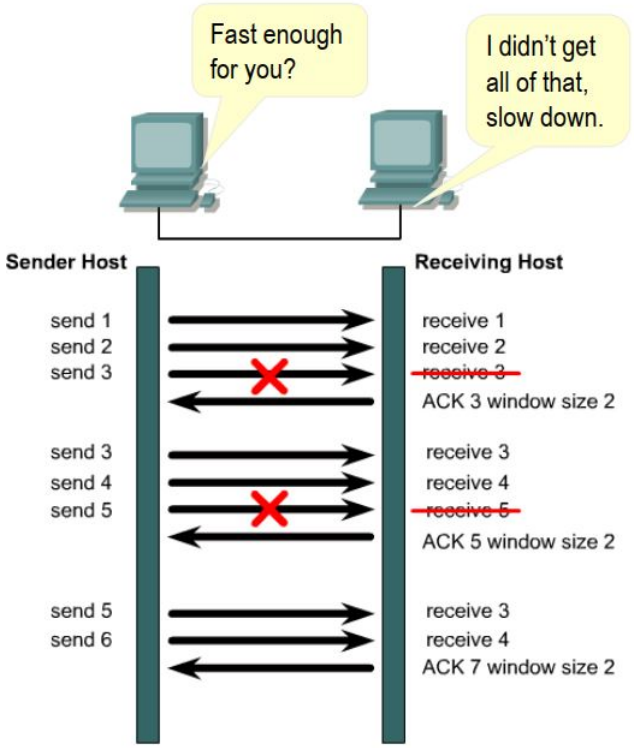

Figure 5.18: Windowing and window size.

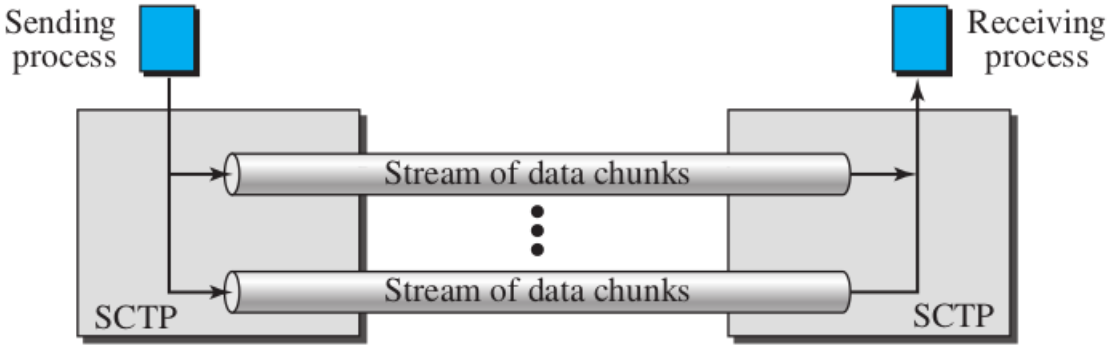

Figure 5.19: Multistream concept.

- An SCTP association, on the other hand, supports multihoming service. The sending and receiving host can define multiple IP addresses in each end for an association. In this fault-tolerant approach, when one path fails, another interface can be used for data delivery without interruption. This fault-tolerant feature is very helpful when we are sending and receiving a real-time payload such as Internet telephony. (Figure 5.20).

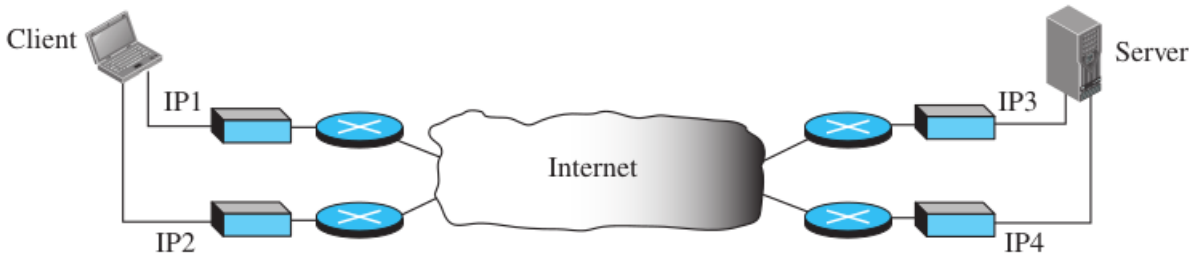

Figure 5.20: Multistream concept.

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

- Like TCP, SCTP is a connection-oriented protocol. However, in SCTP, a connection is called an association.
- SCTP, like TCP, is a reliable transport protocol. It uses an acknowledgment mechanism to check the safe and sound arrival of data.

5.11.1.1 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

Figure 5.21: Some SCTP appliations.

5.12 Congestion in TCP

- Too many packets are present in a subnet or a part of subnet, causes the packet delay and loss that degrades the performance. This situation is called congestion.
- Network and transport layer share the responsibility of handling congestion.
- Most effective way to control traffic congestion is to reduce the load that the transport layer is placing on the network.
- When number of packet sent by the host into the network is within its carrying capacity, and the number delivered is proportional to the number sent. However, as traffic increase too far, the routers are no longer able to cope, and they begin losing packet.
- At very high traffics, performance collapses completely and almost no packet are delivered.

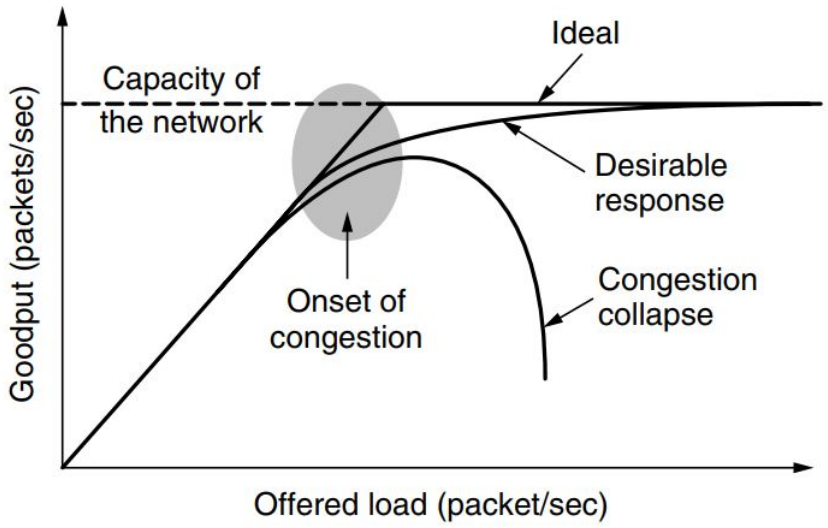

Figure 5.22: With too much traffic, performance drops sharply.

Difference between flow control and congestion control

- Congestion control has to do with making sure the network is able to carry the offered traffic. It is a global issue, involving the behaviour of all the hosts and routers.
- Flow control, in contrast, relates to the traffic between a particular sender and a particular receiver. Its job is to make sure that a fast sender cannot continually transmit data faster than the receiver is able to absorb it.

5.12.1 Causes of Congestion

- When there are more input lines and less or single output lines.
- When there is slow router i.e. if routers CPU's are slow.
- If the router has no free buffers i.e. insufficient memory to hold queue of packets.
- If the components used in subnet (link, router, switches, etc.) have different traffics carrying and switching capacities, then congestion occurs.
- If the bandwidths of the lines are slow, it can't carry large volume of packets and caused congestion. Hence, congestion cannot be eradicated but can be controlled.

5.12.2 Approaches of Congestion Control

- The presence of congestion means that the load is (temporarily) greater than the resources (in a part of a network) can handle.
- Two solutions: increase the resources or decrease the load.
- These solution are usually applied on different time scales to either prevent congestion or react to it once it has occurred.

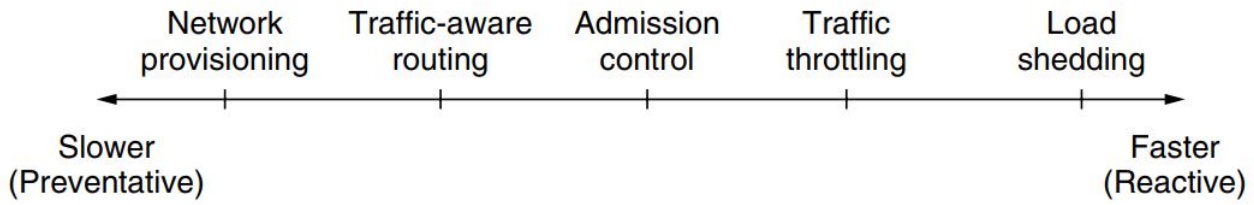

Figure 5.23: Timescale of apporaches of congestion control.

Network Provisioning

- Build a network that is well matched to the traffic it carries.
- Increasing bandwidth of low bandwidth link.
- Adding resources i.e. spare routers where there is serious congestion.

Traffic Aware Routing

- Routers are tailed to change the path away from the heavily used paths.
- Take an account of load or average queuing delay along with link bandwidth and propagation delay to calculate the link cost.
- Least-weight paths will be set that are more lightly loaded.
- But, this slow down the routing as routing tables oscillates widely.

Admission Control

- Sometimes it is not possible to increase capacity.
- The only way then to beat back the congestion is to decrease the load. In a virtual-circuit network, new connections can be refused if they would cause the network to become congested. This is called admission control.
- By analogy, in the telephone system, when a switch gets overloaded it practices admission control by not giving dial tones.

Traffic Throttling

- When congestion is imminent, routers in the network tell the sender to throttle back their transmissions and slow down.
- Can be used in both datagram and virtual circuit network.
- Routers must determine when congestion is happening by continuously monitoring the resources it is using.

  - The utilization of output links,

- Buffering of queued packets inside the router (most useful measure), and
- The number of packets that are lost due to insufficient buffering.

- Different schemes to feedback are:

- Choke packets
- Explicit Congestion Notification
- Hop-by-hop backpressure

Choke Packets
- The most direct way to notify a sender of congestion is to tell it directly.
- In this approach, the router selects a congested packet and sends a choke packet back to the source host.
- When the source host gets the choke packet, it is required to reduce the traffic sent to the specified destination, for example, by 50%.

Explicit Congestion Notification
- Instead of generating additional packets to warn of congestion, a router can tag any packet it forwards (by setting a bit in the packet's header) to signal that it is experiencing congestion.
- When the network delivers the packet, the destination can note that there is congestion and inform the sender. The sender can then throttle its transmissions as before.

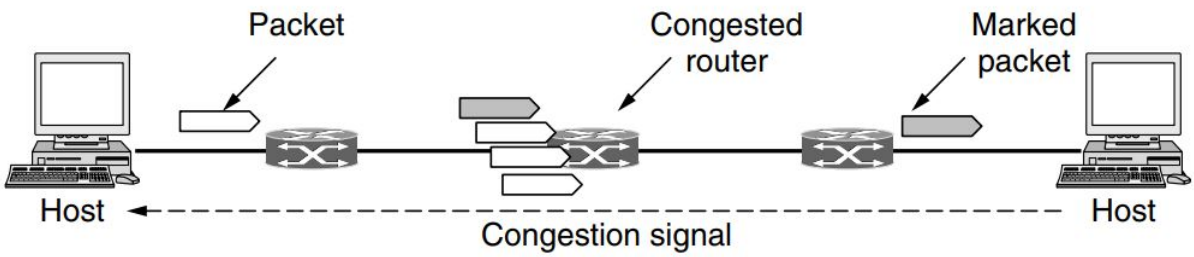

Figure 5.24: Explicit congestion notification.

Hop-by-Hop Backpressure
- It is a congestion control mechanism in which a congested node stops receiving data from the immediate node or nodes.
- It is a node-to-node congestion control that starts with a node and propagates, in the opposite direction of data flow, to the source.
- Can be applied to only a virtual circuit networks, in which each nodes knows the upstream node from which a flow of data is coming.

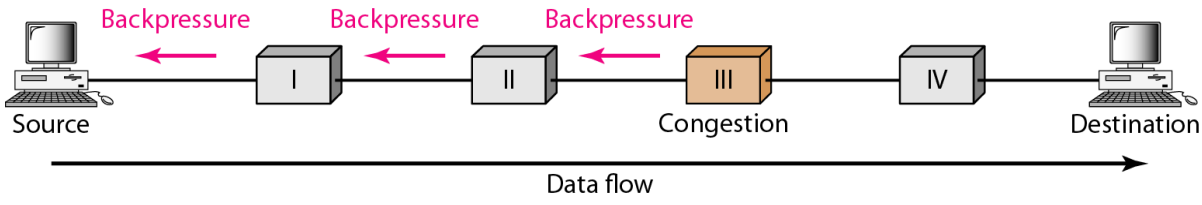

Figure 5.25: Hop-by-hop backpressure.

Load Shedding
- A fancy way of saying that when routers are being inundated by packets that they cannot handle, they just throw them away.
- To implement an intelligent discard policy, applications must mark their packets to indicate to the network how important they are. Then, when packets have to be discarded, routers can first drop packets from the least important class, then the next most important class, and so on.

5.13 Traffic Shaping Algorithms

- Traffic shaping is a bandwidth management technique used on computer networks which delays of all datagrams to bring them into compliance with a desired traffic profile.
- It is a mechanism to control the amount and the rate of the traffic sent to the network.
- It helps to regulate the data transmission and reduces congestion.
- There are two types of traffic shaping algorithms:

i. Leaky Bucket Algorithm
ii. Token Bucket Algorithm

5.13.1 Leaky Bucket Algorithm

- Imagine a bucket with a small hole in the bottom. No matter at what rate water enters the bucket, the outflow is at constant rate, when there is any water in the bucket and zero when the bucket is empty. Also once the bucket is full any additional water entering it spills over the sides and is lost.

- The same idea can be applied to the packets conceptually; each host is connected to the network by an interface, containing a leaky bucket i.e. finite interval queue. If a packet arrives at the queue when it is full, the packet is discarded; if one or more processes within the host try to send a packet when a maximum number are already in queue, the new packet is discarded.

- A simple leaky bucket algorithm can be implemented using FIFO queue. A FIFO queue holds the packets. If the traffic consists of fixed-size packets (e.g., cells in ATM networks), 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.

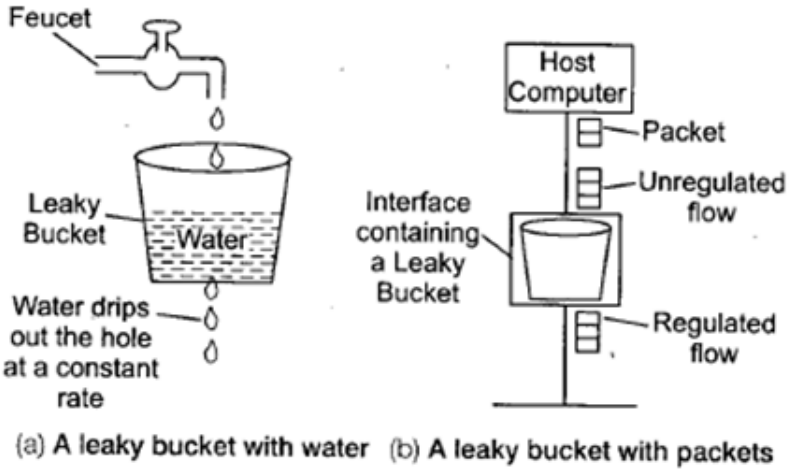

Figure 5.26: Leaky bucket algorithm.

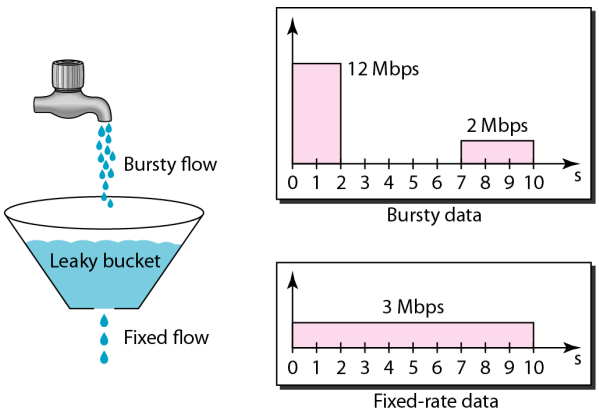

Figure 5.27: Leaky bucket algorithm data rate domostration.

Alorithm

Step-1: Initialize a counter to n at the tick of the clock.

Step-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.
Step-3: Reset the counter and go to step 1.

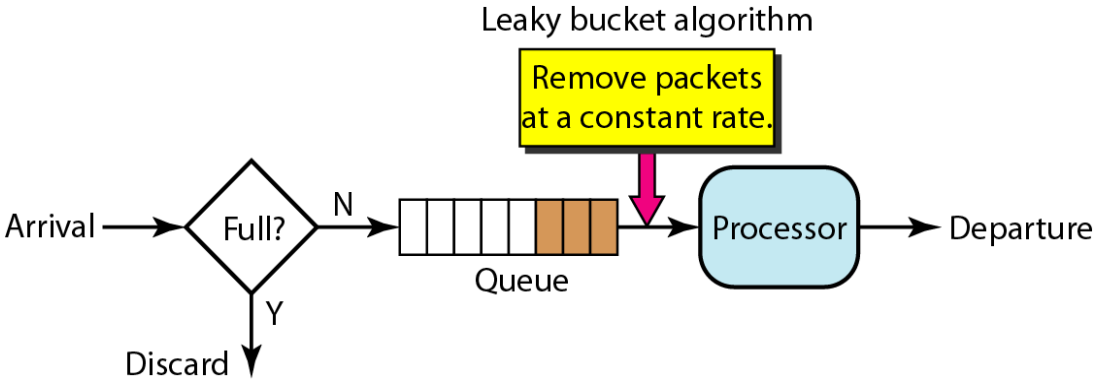

Figure 5.28: Leaky bucket algorithm.

Numeric Example:

Let **n** = 1000

Packet =.

200	700	500	450	400	200

Since n > front of Queue i.e. n>200
Therefore, n= 1000-200 = 800
Packet size of 200 is sent to the network

200	700	500	450	400

Now Again n > front of queue i.e. n > 400
Therefore, n= 800-400 = 400
Packet size of 400 is sent to the network

200	700	500	450

Since n < front of queue .
There fore, the procedure is stop.

And we initialize **n = 1000** on another tick of clock.

This procedure is repeated until all the packets is sent to the network.

5.13.2 Tocket Bucket Algorithm

- A leaky bucket algorithm is based on the rigid output pattern at the average rate no matter how burst traffic is. In leaky bucket, there are chances of loss of packet as packet is filled in bucket and overflow if bucket if full. To minimize such limitation of bucket, token bucket algorithm was introduced.

- In this algorithm the buckets holds token to transmit a packet, the host must capture and destroy one token.
- Tokens are generated by a clock rate of one token every t sec. for a packet to be generated.

- This implantation of basic token bucket algorithm is just a variable counts token. The counter is incremented by one at every t sec and decremented by one whenever one packet is sent.
- When counter hits zero, no packet can be sent.

Algorithm:
Step 1: A token is added at every *t* time.

Step 2: The bucket can hold at most *b*-tokens. If a token arrive when bucket is full, it is discarded.
Step 3: When a packet of *m* bytes arrived *m* tokens are removed from the bukcet and the packet is sent to the network.
Step 4: If less than *n* token are available, no tokens are removed from the bucket and the packet is considered to be *non conformant*. The non conformant packet may be enquequed for subsequent transmission when sufficient token have been accumulated in the bucket.

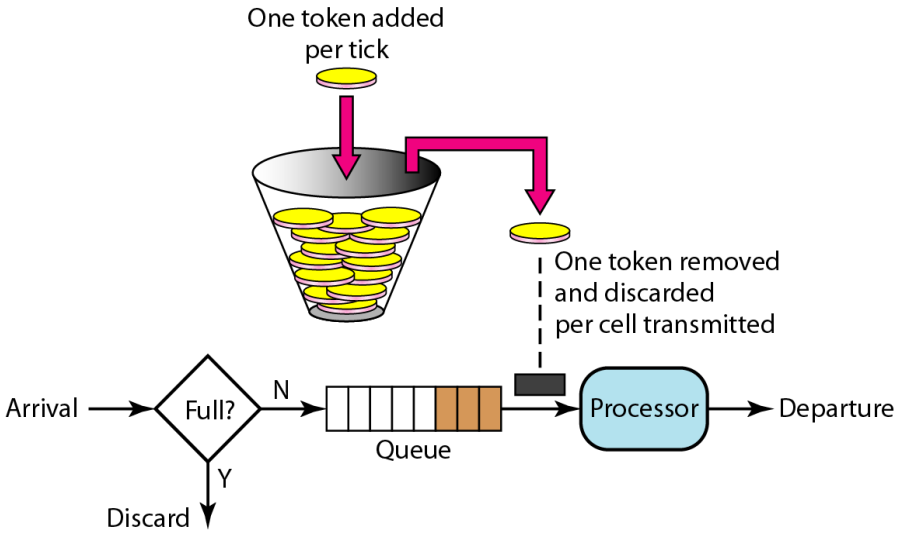

Figure 5.29: Token bucket algorithm.

Token Bucket	**Leaky Bucket**
1. Token dependent	1. Token independent.
2. If bucket is full, token are discarded, but not the packet.	2. If bucket is full, packet or data is discarded.
3. Packets can only transmitted when there are enough token.	3. Packets are transmitted continuosuly.
4. It allows large bursts of be sent faster rate after that constant rate.	4. It sends the packets at constant rate.
5. It saves token to send large bursts.	5. It does not save token.

5.14 Socket Programming

- **Goal**: learn how to build client/server applications that communicate using sockets.
- **Socket**: door between application process and end-to-end transport protocol (TCP or UDP).

- A typical network application consists of a pair of programs—a client program and a server program—residing in two different end systems.
- When these two programs are executed, a client process and a server process are created, and these processes communicate with each other by reading from, and writing to, sockets.
- When creating a network application, the developer's main task is therefore to write the code for both the client and server programs.

- Two sockets types for two transport services:
UDP: unreliable datagram
TCP: reliable, byte stream-oriented

We'll use the following simple client-server application to demonstrate socket programming for both UDP and TCP:

1. Client reads a line of characters (data) from its keyboard and sends the data to the server.
2. The server receives the data and converts characters to uppercase.
3. The server sends the modified data to the client.
4. The client receives the modified data and displays the line on its screen.

5.14.1 Socket Programming with UDP

UDP: no "connection" between client & server
- no handshaking before sending data
- sender explicitly attaches IP destination address and port number to each packet
- receiver extracts sender IP address and port number from received packet.

UDP: transmitted data may be lost or received out-of-order
- Application viewpoint:
- UDP provides unreliable transfer of groups of bytes ("datagrams") between client and server.

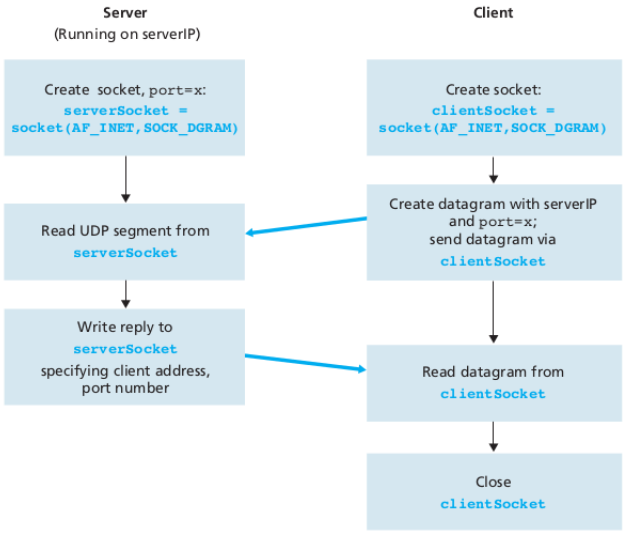

Figure 5.30: The client-server application using UDP.

5.14.2 Socket Programming with TCP

client must contact server
- server process must first be running.
- server must have created socket (door) that welcomes client's contact.

client contacts server by:
- Creating TCP socket, specifying IP address, port number of server process.
- when client creates socket: client TCP establishes connection to server TCP.
- when contacted by client, server TCP creates new socket for server process to communicate with that particular client allows server to talk with multiple clients source port numbers used to distinguish clients.

Application viewport:
- TCP provides reliable, in-order byte-stream transfer ("pipe") between client and server.

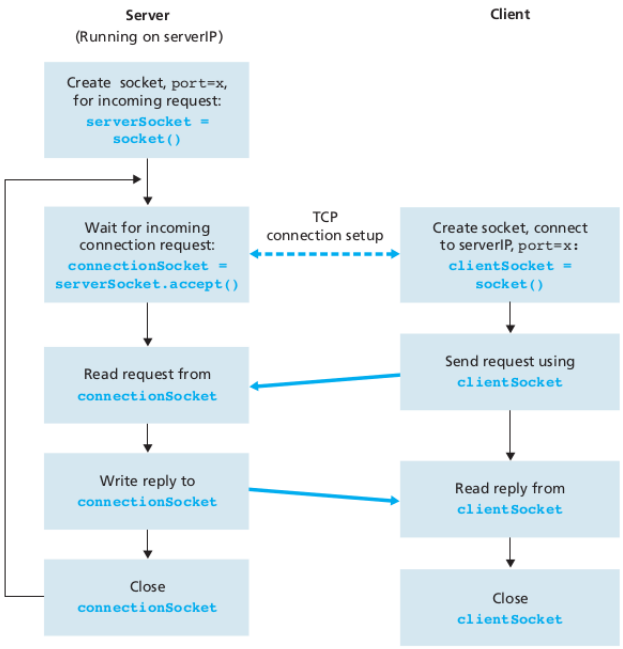

Figure 5.31: The client-server application using TCP.