Computer Networks

Reference Books:

1. Computer Networking 4th edition, By Behrouz A.Forouzan
2. Computer Networks 5th edition, By A Tanenbaum
3. Computer Networks 5th edition, By Peterson
4. Computer Networking A Top Down Approcah, 6th ed, by Kruose

This PDF contains the notes from the standards books and are only meant for GATE CSE aspirants.

Notes Compiled By-
Manu Thakur
Mtech CSE, IIT Delhi
worstguymanu@gmail.com
https://www.facebook.com/Worstguymanu

Computer Networks

Packet Switching:

To send a message from a source end system to a destination end system, the source breaks long messages into smaller chunks of data known as **packets**. Between source and destination, each packet travels through **communication links** and **packet switches** (routers and link-layer switches).

Packets are transmitted over each communication link at a rate equal to the full transmission rate of the link. So, if a source end system or a packet switch is sending a packet of L bits over a link with transmission rate **R bits/sec**, then the time to transmit the packet is **L/R seconds**, where packet contains L bits.

Store-and-Forward Transmission

Store-and-forward transmission means that the packet switch must receive the entire packet before it can begin to transmit the first bit of the packet onto the **outbound link**.

For Example:

The source has three packets, each consisting of L bits, to send to the destination. The source has

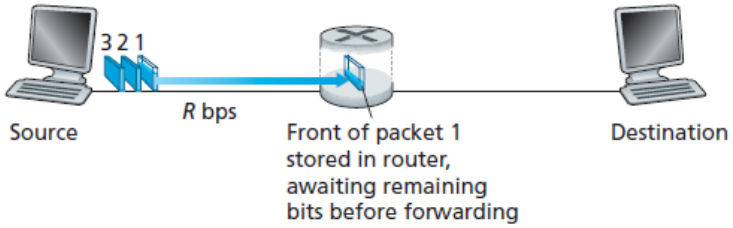


Figure 1.11 ♦ Store-and-forward packet switching

transmitted some of packet 1, and the front of packet 1 has already arrived at the router. Because the router employs **store-and-forwarding**, at this instant of time, the router cannot transmit the bits it has received; instead it must first buffer (i.e., "store") the packet's bits. Only after the router has received all of the packet's bits can it begin to transmit (i.e., "forward") the packet onto the outbound link.

Suppose propagation time is negligible, the source begins to transmit at time 0; at time **L/R seconds**, the **source has transmitted the entire packet**, and the entire packet has been received and stored at the router (since there is no propagation delay). At time L/R seconds, since the router has just received the entire packet, it can begin to transmit the packet onto the outbound link towards the destination; at **time 2L/R**, the router has transmitted the entire packet, and the entire packet has been received by the destination. **Thus, the total delay is 2L/R**.

Note: If the switch instead forwarded bits as soon as they arrive (without first receiving the entire packet), then the total delay would be **L/R** since bits are not held up at the router.

At time L/R, the router begins to forward the first packet. But also at time L/R the source will begin to send the second packet, since it has just finished sending the entire first packet. Thus, at time 2L/R, the destination has received the first packet and the router has received the second packet. **Finally, at time 4L/R the destination has received all three packets**!

Sending one packet from source to destination over a path consisting of **N links** each of rate R (thus, **there are N-1 routers between source and destination**).

$$d_{\text{end-to-end}} = N\frac{L}{R}$$

And for P packets it will be

= **N*(L/R) + (P-1)*(L/R)**

Queuing Delays and Packet Loss

Each packet switch has multiple links attached to it. For each attached link, the packet switch has an **output buffer** (also called an output queue), which stores packets that the router is about to send into that link. If an arriving packet needs to be transmitted onto a link but finds the link busy with the transmission of another packet, the arriving packet must wait in the output buffer. Thus, in addition to the store-and-forward delays, packets suffer output **buffer queuing** delays. These delays are variable and depend on the level of congestion in the network. Since the amount of buffer space is finite, an arriving packet may find that the buffer is completely full with other packets waiting for transmission. In this case, **packet loss** will occur either the arriving packet or one of the already-queued packets will be dropped.

Forwarding Tables and Routing Protocols

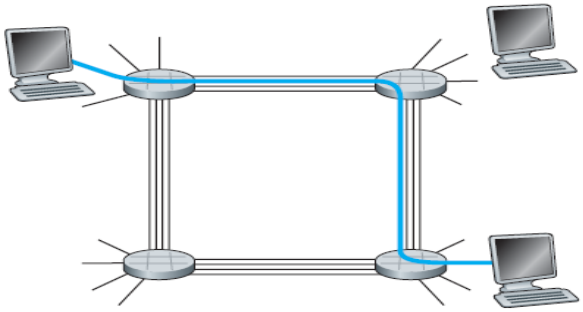
Each router has a forwarding table that maps destination addresses (or portions of the destination addresses) to that router's outbound links. When a packet arrives at a router, the router examines the address and searches its **forwarding table**, using this destination address, to find the appropriate outbound link.

Circuit Switching

There are two fundamental approaches to moving data through a network of links and switches: **circuit switching and packet switching**.
In circuit-switched networks, the resources to provide for communication between the end systems are **reserved** for the duration of the communication session between the end systems.

"In packet-switched networks, these resources are not reserved; a session's messages use the **resources on demand**, and as a consequence, may have to wait (that is, queue) for access to a communication link"

When the network establishes the circuit, it also reserves a constant transmission rate in the network's links (representing a fraction of each link's transmission capacity) for the duration of the connection. Since a given transmission rate has been reserved for this sender-to-receiver connection, the sender **can transfer the data to the receiver at the guaranteed constant rate**. When two hosts want to communicate, the network establishes a dedicated **end-to-end** connection b between the two hosts.



13 ♦ A simple circuit-switched network consisting of four switches and four links

Because each link has four circuits, for each link used by the end-to-end connection, the connection gets one fourth of the link's total transmission capacity for the duration of the connection.

Multiplexing in Circuit-Switched Networks

A circuit in a link is implemented with either **frequency-division multiplexing** (FDM) or **time-division Multiplexing** (TDM). With FDM, the frequency spectrum of a link is divided up among the Connections established across the link.

Example:

How long it takes to send a file of 640,000 bits from Host A to Host B over a circuit-switched network. Suppose that all links in the network use TDM with 24 slots and have a bit rate of 1.536 Mbps. Also suppose that it takes 500 msec to establish an end-to-end circuit before Host A can begin to transmit the file. How long does it take to send the file?

Solution:

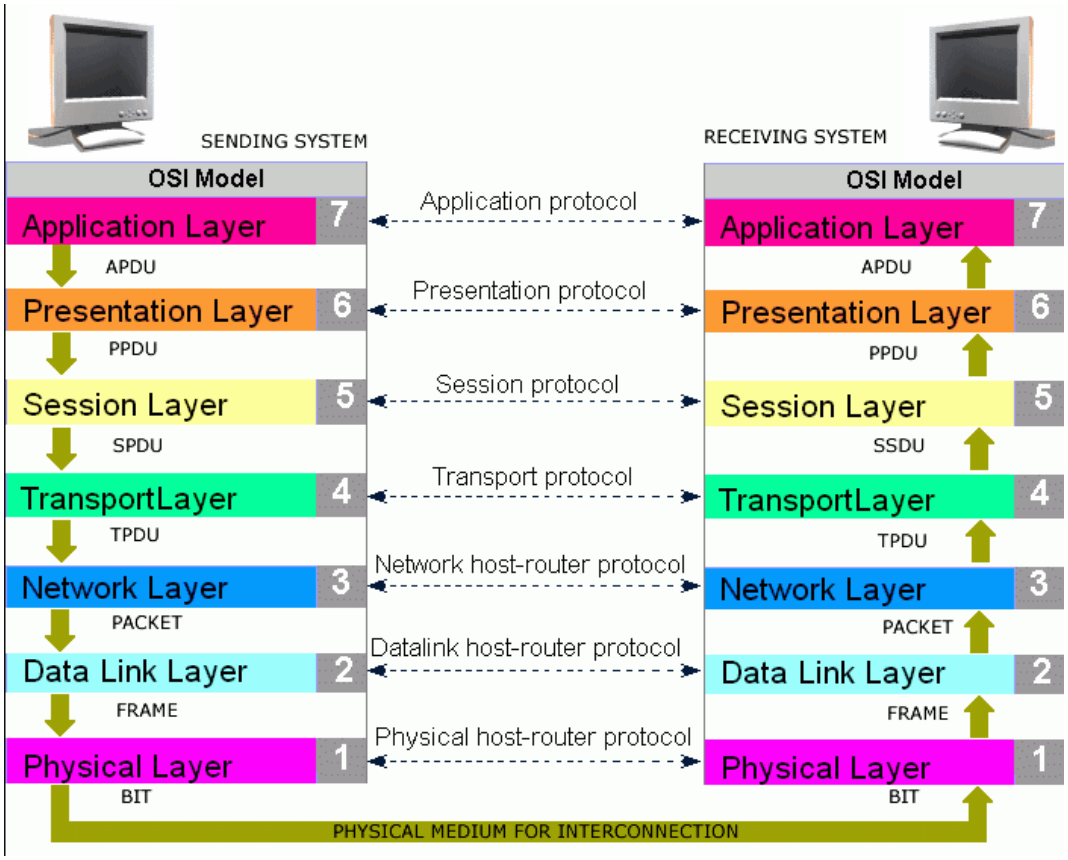
Each circuit has a transmission rate of (1.536 Mbps)/24 = 64 kbps, so it takes (640,000 bits)/(64 kbps) = 10 seconds to transmit the file. To this 10 seconds we add the circuit establishment time(500 ms), giving 10.5 seconds to send the file.

Packet Switching Versus Circuit Switching

Packet switching:

1. **packet switching is not suitable for real-time services**
2. P.S offers better sharing of transmission capacity than circuit switching
3. P.S is simpler, more efficient, and less costly to implement than circuit switching

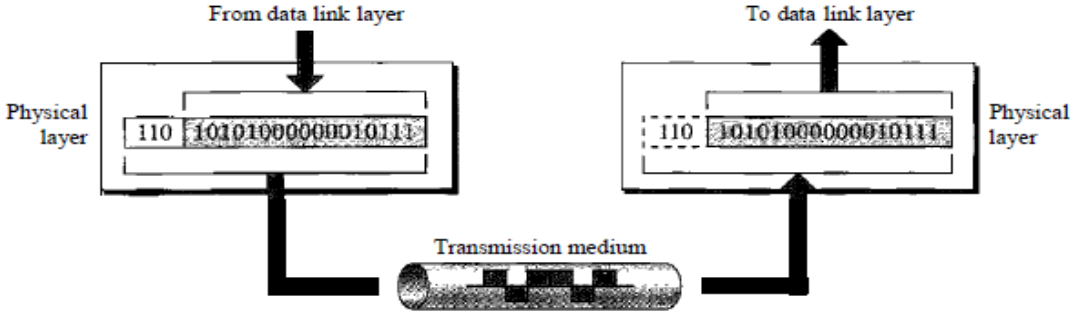
Layered Architecture



OSI reference model

Encapsulation:
A packet (header and data) at level 7 is encapsulated in a packet at level 6. The whole packet at level 6 is encapsulated in a packet at level 5, and so on.
In other words, the data portion of a packet at level N - 1 carries the whole packet (data and header and maybe trailer) from level N. The concept is called encapsulation; level N - 1 is not aware of which part of the encapsulated packet is data and which part is the header or trailer. For level N - 1, the whole packet coming from level N is treated as one integral unit.

Physical Layer
The job of the physical layer is to move the **individual bits** within the frame from one node (hop) to the next. To be transmitted, bits must be encoded into signals--electrical or optical.



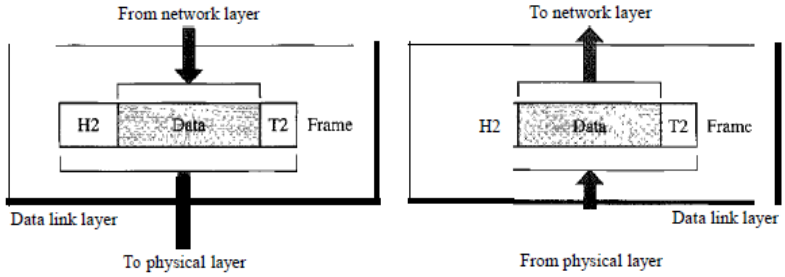
The physical layer is also concerned with the following:

a. Physical characteristics of interfaces and medium.
b. Data rate: The transmission rate-the number of bits sent each second.
c. **Synchronization of bits:** the sender and receiver not only must use the same bit rate but also must be synchronized at the bit level. In other words, the sender and the receiver clocks must be synchronized.
d. Line configuration: In a **point-to-point** configuration, two devices are connected through a dedicated link. In a **multipoint** configuration, a link is shared among several devices.
e. Physical topology: how devices are connected to make a network. For eg: Mesh Topology, Ring topology, Bus topology etc.
f. **Transmission mode**: The physical layer also defines the direction of transmission between two devices: simplex, half-duplex, or full-duplex.

Data Link Layer

The data link layer transforms the physical layer, a raw transmission facility, to a reliable link. It makes the physical layer appear error-free to the upper layer (network layer).

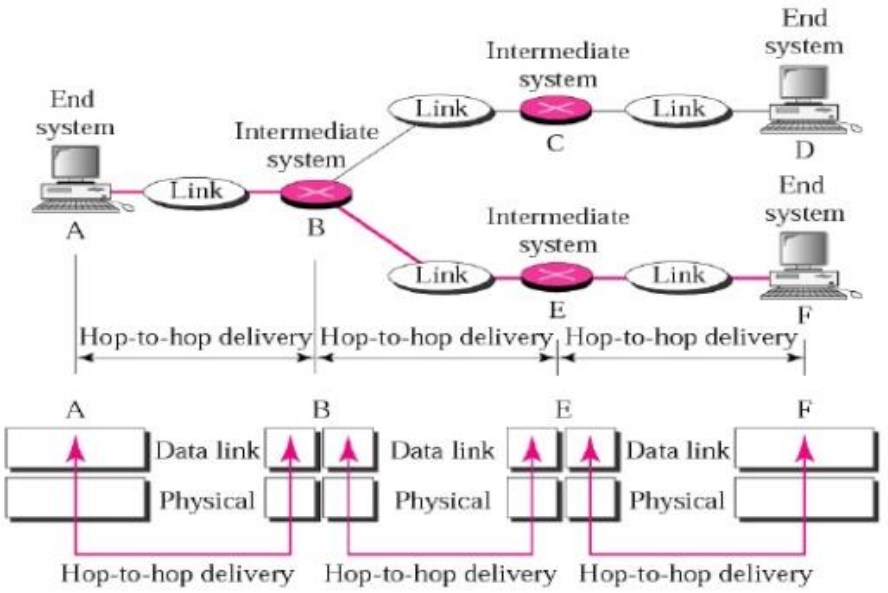
Figure 2.6 *Data link layer*



The data link layer is responsible for moving frames from one hop (node) to the next

Other responsibilities of the data link layer include the following:

i. **Framing**. The data link layer divides the stream of bits received from the network layer into manageable data units called frames.
ii. **Physical addressing**. If frames are to be distributed to different systems on the network, the data link layer adds a header to the frame to define the sender and/or receiver of the frame. If the frame is intended for a system outside the sender's network, the receiver address is the address of the device that connects the network to the next one.
iii. **Flow control**. If the rate at which the data are absorbed by the receiver is less than the rate at which data are produced in the sender, the data link layer imposes a flow control mechanism to avoid overwhelming the receiver.
iv. **Error control**. The data link layer adds reliability to the physical layer by adding mechanisms to detect and retransmit damaged or lost frames. It also uses a mechanism to recognize duplicate frames. Error control is normally achieved through a **trailer** added to the end of the frame.
v. **Access control** when two or more devices are connected to the same link, data link layer protocols are necessary to determine which device has control over the link at any given time.



Hop to Hop Delivery

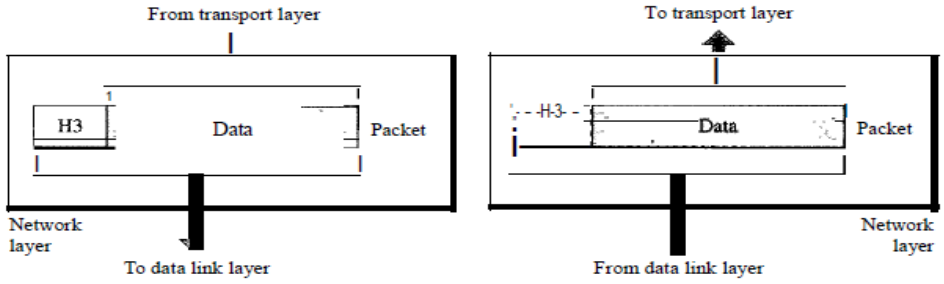
Communication at the data link layer occurs between two adjacent nodes. To send data from A to F, three partial deliveries are made. First, the data link layer at A sends a frame to the data link layer at B (a router). Second, the data link layer at B sends a new frame to the data link layer at E. Finally, the data link layer at E sends a new frame to the data link layer at F. Note that the frames that are exchanged between the three nodes have different values in the headers. The frame from A to B has B as the destination address and A as the source address. The frame from B to E has E as the destination address and B as the source address. The frame from E to F has F as the destination address and E as the source address. The values of the trailers can also be different if error checking includes the header of the frame.

Network Layer

The network layer is responsible for the source-to-destination delivery of a packet

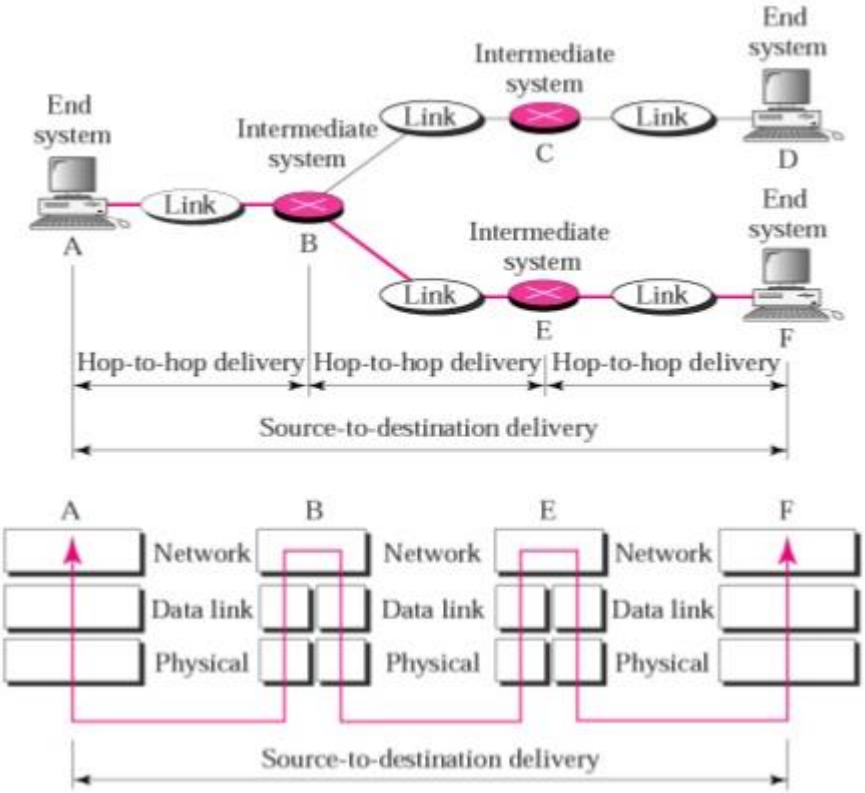
If two systems are connected to the same link, there is usually no need for a network layer. If the Two systems are attached to different networks (links) with connecting devices between the networks (links).
The Internet's network layer is responsible for moving network-layer **packets known as datagrams** from one host to another. The Internet transport-layer protocol (TCP or UDP) in a source host passes a transport-layer segment and a destination address to the network layer.

Figure 2.8 *Network layer*



i. **Logical addressing** the physical addressing implemented by the data link layer handles the addressing problem locally. If a packet passes the network boundary, we need another addressing system to help distinguish the source and destination systems. The network layer adds a header to the packet coming from the upper layer.

ii. **Routing** When independent networks or links are connected to create inter-networks (network of networks) or a large network, the connecting devices (called routers or switches) route or switch the packets to their final destination



The network layer is responsible for the delivery of individual packets from the source host to the destination host.

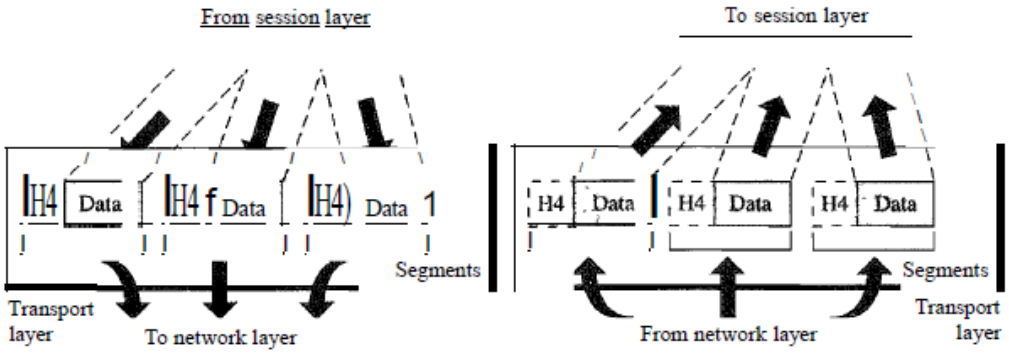
The network layer at A sends the packet to the network layer at B. When the packet arrives at router B, the router makes a decision based on the final destination (F) of the packet. The network layer at B, therefore, sends the packet to the network layer at E. The network layer at E, in tum, sends the packet to the network layer at F.

Transport Layer

The transport layer is responsible for process-to-process delivery of the entire message. A process is an application program running on a host. Whereas the network layer oversees source-to-destination delivery of individual packets, it does not recognize any relationship between those packets. It treats each one independently.

The transport layer, on the other hand, ensures that the whole message arrives intact and in order

Figure 2.10 *Transport layer*



The transport layer is responsible for the delivery of a message from one process to another.

In the Internet there are two transport protocols, **TCP** and **UDP**, either of which can transport application-layer messages. TCP provides a **connection-oriented** service to its applications. The UDP protocol provides a **connectionless** service to its applications. We'll refer to a transport-layer packet as a **segment**.

a. **Service-point addressing** the transport layer header must therefore include a type of address called a service-point address (or **port address**).
b. **Segmentation and reassembly** A message is divided into transmittable segments, with each segment containing a sequence number. These numbers enable the transport layer to reassemble the message correctly upon arriving at the destination and to identify and replace packets that were lost in transmission.
c. **Connection control**. The transport layer can be either connectionless or connection-oriented. A connectionless transport layer treats each segment as an independent packet and delivers it to the transport layer at the destination machine. A connection-oriented Transport layer makes a connection with the transport layer at the destination machine first before delivering the packets. After all the data are transferred, the connection is terminated.
d. **Flow control**. Like the data link layer, the transport layer is responsible for flow control. However, flow control at this layer is performed end to end rather than across a single link.
e. **Error control** the sending transport layer makes sure that the entire message arrives at the receiving transport layer without error (damage, loss, or duplication). Error correction is usually achieved through retransmission.

Session Layer

The session layer allows users on different machines to establish **sessions** between them.

The session layer is responsible for dialog control and synchronization.

a. **Dialog control** the session layer allows two systems to enter into a dialog. It allows the communication between two processes to take place in either half duplex (one way at a time) or full-duplex (two ways at a time) mode.
b. **Synchronization**
The session layer allows a process to add checkpoints, or synchronization points, to a stream of data. For example, if a system is sending a file of 2000 pages, it is advisable to insert checkpoints after every 100 pages.

Presentation Layer

The presentation layer is concerned with the syntax and semantics of the information transmitted.

The presentation layer is responsible for translation, compression, and encryption.

a. **Translation** The processes (running programs) in two systems are usually exchanging information in the form of character strings, numbers, and so on. **The information must be changed to bit streams before being transmitted**.
b. **Encryption** to carry sensitive information, a system must be able to ensure privacy.
c. **Compression** Data compression becomes particularly important in the transmission of multimedia such as text, audio, and video.

Application Layer

The application layer enables the user, whether human or software, to access the network. It uses the following protocols as HTTP, FTP, SMTP, DNS etc. We'll refer to this packet of information at the application layer as a **message**.

How these layers work:

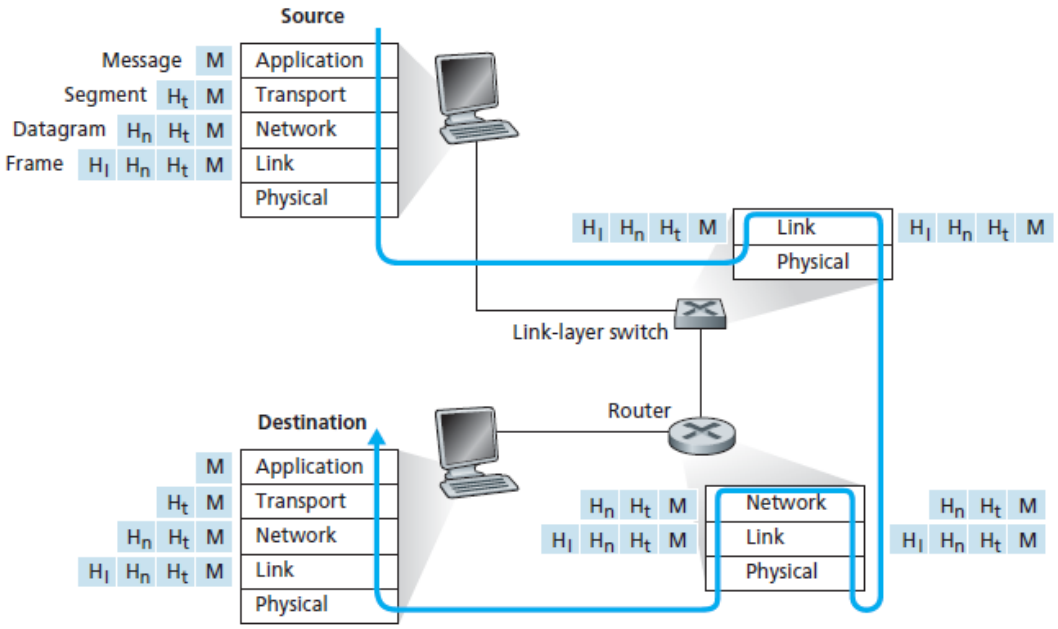


Figure 1.24 ♦ Hosts, routers, and link-layer switches; each contains a different set of layers, reflecting their differences in functionality

Figure 1.24 also illustrates the important concept of **encapsulation**. At the sending host, an **application-layer message** (M) is passed to the **transport layer**. In the simplest case, the transport layer takes the message and appends additional information (so-called transport-layer header information, Ht) that will be used by the receiver-side transport layer.

The application-layer message and the transport-layer header information together constitute the **transport-layer segment**. The transport-layer segment thus encapsulates the application-layer message.

The transport layer then passes the segment to the **network layer**, which adds network-layer header information (Hn) such as source and destination end system addresses, creating a **network-layer datagram**. The datagram is then passed to the **Data link layer**, which (of course!) will add its own link-layer header information and create a link-layer **frame**. Thus, we see that at each layer, a packet as two types of fields: **header fields and a payload field**. **The payload is typically a packet from the layer above**.

Application Layer

The application layer is responsible for providing services to the user.

Domain Name System

The client/server programs can be divided into two categories: those that can be directly used by the user, such as e-mail, and those that support other Application programs. The Domain Name System (DNS) is a supporting program that is used by other programs such as e-mail.

Note: DNS is commonly used by other application-layer protocols—including **HTTP, SMTP, and FTP** to translate user-supplied hostnames to IP addresses.

A user of an e-mail program may know the e-mail address of the recipient; however, the IP protocol needs the IP address.

1. The browser extracts the hostname, www.someschool.edu, from the URL and passes the hostname to the client side of the DNS application.
2. The DNS client sends a query containing the hostname to a DNS server.
3. The DNS client eventually receives a reply, which includes the IP address for the hostname.
4. Once the browser receives the IP address from DNS, it can initiate a TCP connection to the HTTP server process located at port 80 at that IP address.

DNS provides a few other important services in addition to translating hostnames to IP addresses:

1. **Host aliasing**, a host with a complicated hostname can have one or more alias names.
2. **Mail server aliasing** it is highly desirable that e-mail addresses be mnemonic.
3. **Load distribution** DNS is also used to perform load distribution among replicated servers, such as replicated Web servers. Busy sites, such as cnn.com, are replicated over multiple servers, with each server running on a different end system and each having a different IP address.

Note: Time-To-Live field tells how stable the record is. DNS is client-Server application. DNS tree can have root(0) to 127 levels. DNS Organizes name space in a hierarchical structure.

Hyper Text Transfer Protocol (HTTP)

HTTP is implemented in two programs: a client program and a server program.

HTTP is said to be a stateless protocol

Non-Persistent and Persistent Connections:

In many Internet applications, the client and server Communicate for an extended period of time, with the client making a series of requests and the Server responding to each of the requests. Depending on the application and on how the application is being used, the series of requests may be made back-to-back, periodically at regular intervals. When this client-server interaction is taking place over TCP, the application developer needs to make an important decision - **should each request/response pair be sent over a separate TCP connection** known as **non-persistent connections** or should all of the requests and their corresponding responses be sent over the same TCP connection known as **persistent connections**.

Note: The default mode of HTTP uses persistent connections with pipelining.

URL: **Uniform Resource Locator**, This URL consists of three parts: the protocol (http), the DNS name of the host (www.cs.washington.edu), and the path name (index.html). When a user clicks on a hyperlink, the browser carries out a series of steps:

a. The browser determines the URL (by seeing what was selected).
b. The browser asks DNS for the IP address of the server www.cs.washington.edu.
c. DNS replies with 128.208.3.88
d. The browser makes a TCP connection to 128.208.3.88 on port 80, the well-known port for the HTTP protocol.
e. It sends over an HTTP request asking for the page /index.html
f. The www.cs.washington.edu server sends the page as an HTTP response, for example, by sending the file /index.html.
g. If the page includes URLs that are needed for display, the browser fetches the other URLs **using the same process**. In this case, the URLs include multiple embedded images also fetched from www.cs.washington.edu, an embedded video from www.youtube.com, and a script from www.google-analytics.com
h. The browser displays the page /index.html
i. The TCP connections are released if there are no other requests to the same servers for a short period.

Cookies

The World Wide Web was originally designed as a **stateless** entity. A client sends a request; a server Responds. Their relationship is over. Today the Web has other functions; some are listed here.

a. Some websites need to allow access to registered clients only.
b. Websites are being used as electronic stores that allow users to browse through the store.
c. Some websites are used as portals: the user selects the Web pages he wants to see. Eg: Yahoo etc.
d. Some websites are just advertising

For these purposes, the cookie mechanism was devised.

1. When a server receives a request from a client, it stores information about the client in a file or a string.
2. The server includes the cookie in the response that it sends to the client.
3. When the client receives the response, the browser stores the cookie in the cookie directory, which is sorted by the domain server name.

Domain	Path	Content	Expires	Secure
toms-casino.com	/	CustomerID=297793521	15-10-10 17:00	Yes
jills-store.com	/	Cart=1-00501;1-07031;2-13721	11-1-11 14:22	No
aportal.com	/	Prefs=Stk:CSCO+ORCL;Spt:Jets	31-12-20 23:59	No
sneaky.com	/	UserID=4627239101	31-12-19 23:59	No

Figure 7-22. Some examples of cookies.

Cookies are just strings, not executable programs.

Note:

1. A cookie could contain a virus, but since cookies are treated as data, there is no official way for the virus to actually run and do damage.

2. If **Expires** field is absent, the browser discards the cookie when it exits. Such a cookie is called a **non-persistent** cookie. If a time and date are supplied, the cookie is said to be a **persistent** cookie and is kept until it expires.

HTTP:

The Hypertext Transfer Protocol (HTTP) is a **request-response protocol** used mainly to access data on the **World Wide Web**.

HTTP functions as a combination of FTP and SMTP. It is similar to **FTP** because it transfers files and uses the services of **TCP**, but it uses only one connection. **There is no separate control connection**; only data are transferred between the client and the server.
HTTP is like SMTP because the data transferred between the client and the server look like **SMTP** Messages.
In addition, the format of the messages is controlled by **MIME**-like headers. Unlike SMTP, the HTTP Messages(ASCII) are not destined to be read by humans; they are read and interpreted by the HTTP **server** and HTTP **client** (browser).
SMTP messages are **stored and forwarded**, but HTTP messages are delivered immediately.

HTTP uses the services of TCP on well-known port 80. HTTP itself is a stateless protocol.

Table 27.1 *Methods*

Method	*Action*
GET	Requests a document from the server
HEAD	Requests information about a document but not the document itself
POST	Sends some information from the client to the server
PUT	Sends a document from the server to the client
TRACE	Echoes the incoming request
CONNECT	Reserved
OPTION	Inquires about available options

Connect: Connect through a proxy
Delete: Remove the Web page

Note: HTTP uses ASCII characters. A client can directly connect to a server using TELNET, which logs into port 80.

Persistent Versus Non-persistent Connection:

HTTP **1.0** specified a **non-persistent connection**, While a **persistent connection** is the default in version **1.1**.

Non-persistent Connection:

In a **non-persistent** connection, one TCP connection is made for each request/response.

a. The client opens a TCP connection and sends a request.
b. The server sends the response and closes the connection.
c. The client reads the data until it encounters an end-of-file marker; it then closes the connection.

For N different pictures in different files, the connection must be opened and closed N times and the server needs N different buffers.

Persistent Connection:

In a **persistent connection**, the server leaves the connection open for more requests after sending a response. The server can close the connection at the request of a client or if a **time-out** has been reached. **It is also possible to pipeline requests**.

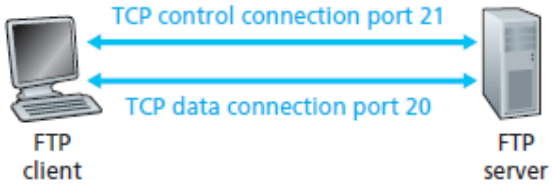
Proxy Server:

HTTP supports proxy servers. A proxy server is a computer that keeps copies of responses to recent requests. The HTTP client sends a request to the proxy server. The proxy server checks its cache. If the response is not stored in the cache, the proxy server sends the request to the corresponding server. Incoming responses are sent to the proxy server and stored for future requests from other clients.
The proxy server reduces the load on the original server, decreases traffic, and improves latency. However, to use the proxy server, the client must be configured to access the proxy instead of the target server.

Network latency is an expression of how much time it takes for a packet of data to get from one designated point to another.

File Transfer Protocol:

File **Transfer Protocol (FTP)** is the standard mechanism provided by TCP/IP for copying a file from one host to another.
FTP differs from other client/server applications in that it establishes **two connections** between the hosts. One connection is used for data transfer, the other for control information (commands and responses). Separation of commands and data transfer makes FTP more efficient.



The control connection remains connected during the entire interactive FTP session. The data connection is opened and then closed for each file transferred.
The data connection, on the other hand, needs more **complex rules** than control line due to the variety of data types transferred.

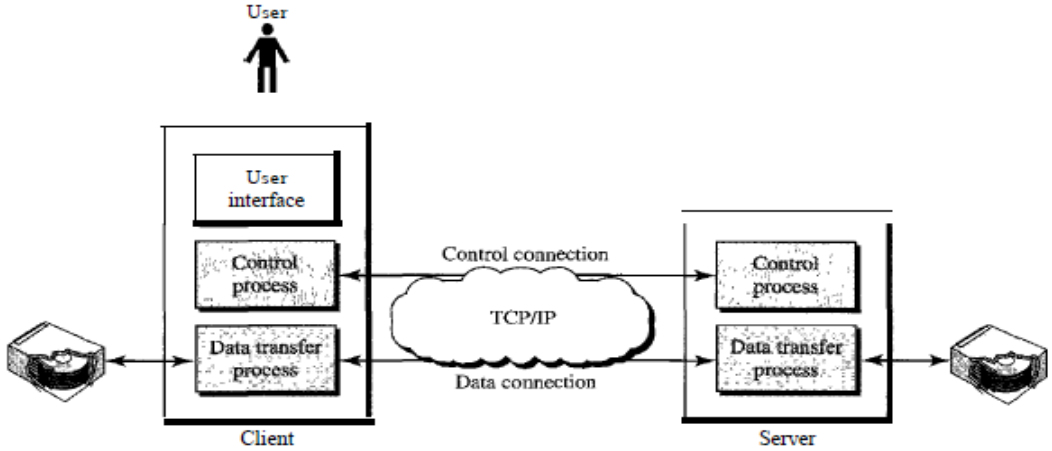
While the control connection is open, the data connection can be opened and closed multiple times if several files are transferred.

The client has three components: user interface, client control process, and the client data transfer process. The server has two components: the server control process and the server data transfer process. The control connection is made between the control processes. The data connection is made between the data transfer processes as shown below in the figure:

Note: FTP uses the same approach as SMTP to communicate across the control connection.

File transfer in FTP means one of three thing (commands):

a. **RETR** A file is to be copied from the server to the client
b. **STOR** file is to be copied from the client to the server
c. **LIST** A list of directory or file names is to be sent from the server to the client



The client must define the type of file to be transferred, the structure of the data, and the transmission mode.

1. HTTP sends request and response header lines into the same TCP connection that carries the transferred file itself **(In-band)**.
2. FTP uses two parallel TCP connections to transfer a file, a control connection and a data connection **(Out-band)**.

TELNET

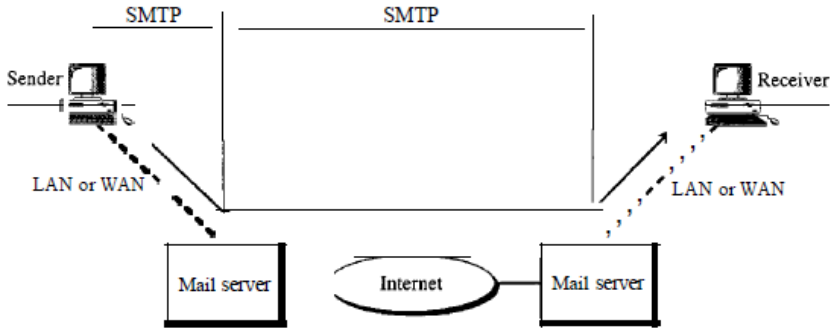
It is the standard TCP/IP protocol for virtual terminal service as proposed by ISO. TELNET Enables the establishment of a connection to a remote system in such a way that the local terminal appears to be a terminal at the remote system.

TELNET is a general-purpose client/server application program.

Simple Mail Transfer Protocol (SMTP):

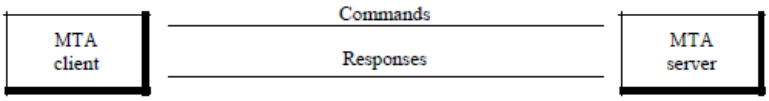
The actual mail transfer is done through **message transfer Agents (MTA)**. To send mail, a system must have the client MTA, and to receive mail, a system must have a server MTA.

Figure 26.16 *SMTP range*



SMTP is used two times, between the sender and the sender's mail server and between the two mail servers.
SMTP uses commands and responses to transfer messages between an MTA client and an MTA Server.

26.17 *Commands and responses*



Keyword	Argument(s)
HELO	Sender's Host Name
MAIL FROM	Sener of the message
RCPTTO	Intended recepient of the mail
DATA	Bpdy of the mail
VRFY	Name of recpient to be verified
EXPN	Mailing list to be expended
HELP	Command Name
SENDFROM	Intended recepient of the mail

Note: We use TELNET to log into port 25 (the well-known port for SMTP).

HTTP V/S SMTP

a. **HTTP** is mainly a **pull** protocol—someone loads information on a Web server and users use HTTP to pull the information from the server at their convenience.
b. **SMTP** is primarily a **push** protocol—the sending mail server pushes the file to the receiving mail server.

POP3

Post Office Protocol, version 3 (**POP3**) is simple and limited in functionality. The **client POP3** software is installed on the recipient computer; the **server POP3** software is installed on the mail server.
POP3 begins when the user agent (the client) opens a TCP connection to the mail server (the server) **on port 110**.
A user agent using POP3 can often be configured (by the user) to "**download and delete**" from the mailbox or to "**download and keep**" at the mailbox.
POP3 progresses through three phases: **authorization**, **transaction**, and **update**. During transaction phase during this phase, the user agent can mark messages for deletion, remove deletion marks, and obtain mail statistics.

IMAP4

IMAP4 is similar to POP3, but it has more features; IMAP4 is more powerful and more Complex. POP3 **does not** allow the user to organize her mail on the server; the user cannot have different folders on the server. In addition, POP3 does not allow the user to **partially check** the contents of the mail before downloading.
IMAP4 provides the following extra functions:

a. A user can check the e-mail header prior to downloading.
b. A user can search the contents of the e-mail for a specific string of characters prior to downloading.
c. A user can partially download e-mail. This is especially useful if bandwidth is limited and the e-mail contains multimedia with high bandwidth requirements.
d. A user can create, delete, or rename mailboxes on the mail server.
e. A user can create a hierarchy of mailboxes in a folder for e-mail storage.

Network Layer

Communication at the network layer is host-to-host (computer-to-computer); Consider a simple network with two hosts, H1 and H2, and several routers on the path between H1 and H2. Suppose that H1 is sending information to H2, The network layer in H1 takes **segments from the transport layer** in H1, encapsulates each segment into a **datagram** (that is, a network-layer packet), and then sends the datagrams to its nearby router, R1. At the receiving host, H2, the network layer receives the datagrams from its nearby router R2, extracts the transport-layer segments, and delivers the segments up to the transport layer at H2.

The primary role of the routers is to forward datagrams from input links to output links.

Forwarding and Routing:

Two important network-layer functions can be identified:

- **Forwarding** When a packet arrives at a router's input link, the router must move the packet to the appropriate output link.
- **Routing** the network layer must determine the route or path taken by packets as they flow from a sender to a receiver. The algorithms that calculate these paths are referred to as **routing algorithms**.

Every router has a **forwarding table**. A router forwards a packet by examining the value of a field in the arriving packet's header.

IPv4 ADDRESSES:

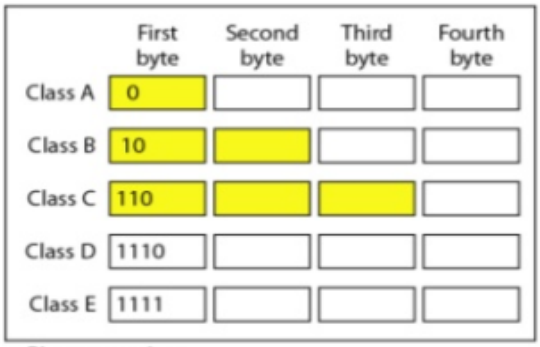
An **IPv4** address is a 32-bit address that *uniquely* and *universally* defines the connection of a device. If a device operating at the network layer has **m connections** to the Internet, it needs to have **m addresses**. **Router** is such a device.

Address Space:
An address space is the total number of addresses used by the protocol. If a protocol uses N bits to define an address, the address space is 2^N because each bit can have two different values (0 or 1) and N bits can have 2^N values.
IPv4 uses **32-bit addresses**, which means that the address space is **2^32**, the actual number is much less because of the restrictions imposed on the addresses.

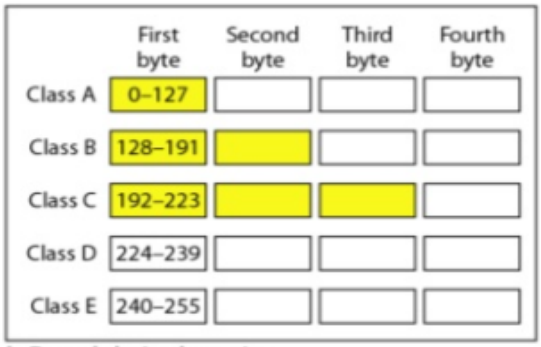
Note:

1. 111.56.045.78 is **invalid** IPV4 address, there must be no leading zero
2. 75.45.301.14 is **invalid** IPV4 address because 301 > 255

Classful Addressing:



a. Binary notation



b. Dotted-decimal notation

One problem with classful addressing is that each class is divided into a **fixed number of blocks** with each block having a fixed size.

Table 19.1 *Number of blocks and block size in classful IPv4 addressing*

Class	*Number of Blocks*	*Block Size*	*Application*
A	128	16,777,216	Unicast
B	16,384	65,536	Unicast
C	2,097,152	256	Unicast
D	1	268,435,456	Multicast
E	1	268,435,456	Reserved

A: 2^7 = 128 B:2^14=16,384 C:2^21=2,097,152 D:2^28 = 2,68,435,456 E: 2^28 = 2,68,435,456

Note:

1. Class D addresses were designed for multicasting as each address is used to define one group of hosts on the internet.
2. Class E addresses were reserved for future use; only a few were used.

In c1assful addressing, a large part of the available addresses were wasted

Netid and Hostid:

In classful addressing, an IP address in class A, B, or C is divided into **netid** and **hostid**. These parts are of varying lengths, depending on the class of the address**. The concept does not apply to classes D and E**.

Class A: 1 Byte defines net-id, and 3 bytes define host-id.
Class B: 2 Bytes define net-id, and 2 bytes define host-id
Class C: 3 Bytes define net-id, and 1 byte define host-id.

Default Mask (The mask can help us to find the net-id and the host-id)

A 32-bit number made of contiguous 1's followed by contiguous 0's. The masks for classes A, B, and C are shown as:

Class	Binary	Dotted-Decimal
A	11111111 00000000 00000000 00000000	255.0.0.0
B	11111111 11111111 00000000 00000000	255.255.0.0
C	11111111 11111111 11111111 00000000	255.255.255.0

The mask can help us to find the netid and the hostid. For example, the mask for a class A address has eight 1s, which means the first 8 bits of any address in class A define the netid; the next 24 bits define the hostid.

Subnetting

It could divide the addresses into several contiguous groups and assign each group to smaller networks (called subnets). Subnetting increases the number of 1's in the mask.

Supernetting

Several networks are combined to create a supernetwork or a **supernet**. An organization that needs 1000 addresses can be granted four contiguous class C blocks. The organization can then use these addresses to create one supernetwork.

Classful addressing, which is almost obsolete, is replaced with classless addressing.

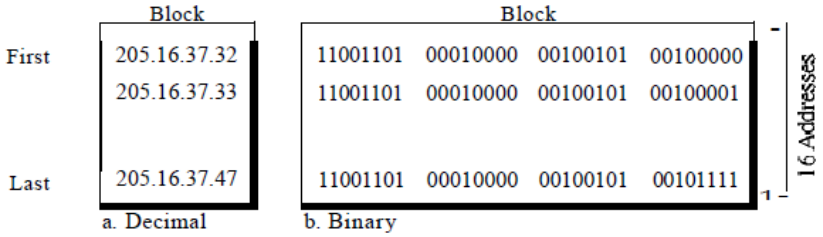
Classless Addressing

There are no classes, but the addresses are still granted in blocks.

Address Blocks: In classless addressing, when an entity, small or large, needs to be connected to the Internet, it is granted a block (range) of addresses. The size of the block (the number of addresses) varies based on the nature and size of the entity.
The Internet authorities impose three restrictions on classless address blocks:

a. The addresses in a block must be contiguous, one after another
b. The number of addresses in a block must be a power of 2 (I, 2, 4, 8, ...)
c. The first address must be evenly divisible by the number of addresses.

A block of 16 addresses granted to a small organization



We can see that the restrictions are applied to this block:

a. The addresses are contiguous.
b. The number of addresses is a power of 2 (16 = 2^4)
c. The first address is divisible by 16.

Note: The first address, when converted to a decimal number, is 3,440,387,360, which when divided by 16 results in 215,024,210.

Mask

A better way to define a block of addresses is to **select any address in the block** and the mask. A mask is a 32-bit number in which the n leftmost bits are **1's** and the 32 - n rightmost bits are 0's.

Note:

a. **In classless addressing the mask for a block can take any value from 0 to 32.**
b. **The address and the /n notation completely define the whole block.**

Note: In 1Pv4 addressing, a block of addresses can be defined as **x.y.z.t/n** in which x.y.z.t defines one of the addresses and the /n defines the **mask**.

First Address: The first address in the block can be found by setting the **32 - n rightmost bits** in the binary notation of the address to **0's.**

Last Address: The last address in the block can be found by setting the rightmost 32 - n bits in the binary notation of the address to **1's.**
Number of Addresses: 2^(32-n)

Example: A block of addresses is granted to a small organization i.e. 205.16.37.39/28. First address, last address and total number of addresses in the granted block are?
Solution: There are 4 host bits as 32-28=4.
39:0010 0111 make last 4 bits 0. Hence first address is: 0010 0000 = 32, and we make last 4 bits as 1 then it will be 0010 1111=47
First address: 205.16.37.32
Last Address: 205.16.37.47
Total Addresses: 2^(32-28) = 16

Second Method to find the first address, last address and total number of addresses is to represent the mask as 32-bit binary.

Given IP: 205.16.37.39/28
Mask: 255.255.255.240 (as last 4 bits are host bits)

First Address: bit by bit ANDing given IP with Mask

Address:	11001101	00010000	00100101	00100111
Mask:	11111111	11111111	11111111	11110000
First address:	11001101	00010000	00100101	00100000

Last Address: bit by bit ORing the given addresses with the complement of the mask

Address:	11001101	00010000	00100101	00100111
Mask complement:	00000000	00000000	00000000	00001111
Last address:	11001101	00010000	*00100101*	*00101111*

Total Addresses: Complementing the mask, interpreting it as a decimal number, and adding 1 to it.

Mask complement:	000000000	00000000	00000000	00001111
Number of addresses:	15 + 1 = 16			

Network Addresses

The first address is called the **network address** and defines the organization network and is not assigned to any device. It defines the organization itself to the rest of the world. First address is the one that is used by **routers** to direct the message sent to the organization from the outside.

Two-Level Hierarchy: No Subnetting

An IP address can define only two levels of hierarchy when not subnetted. The n leftmost bits (**Prefix**) of the address **x.y.z.t/n** define the network (organization N/W) the 32 – n rightmost bits (**suffix**) define the particular host (computer or router) to the network.

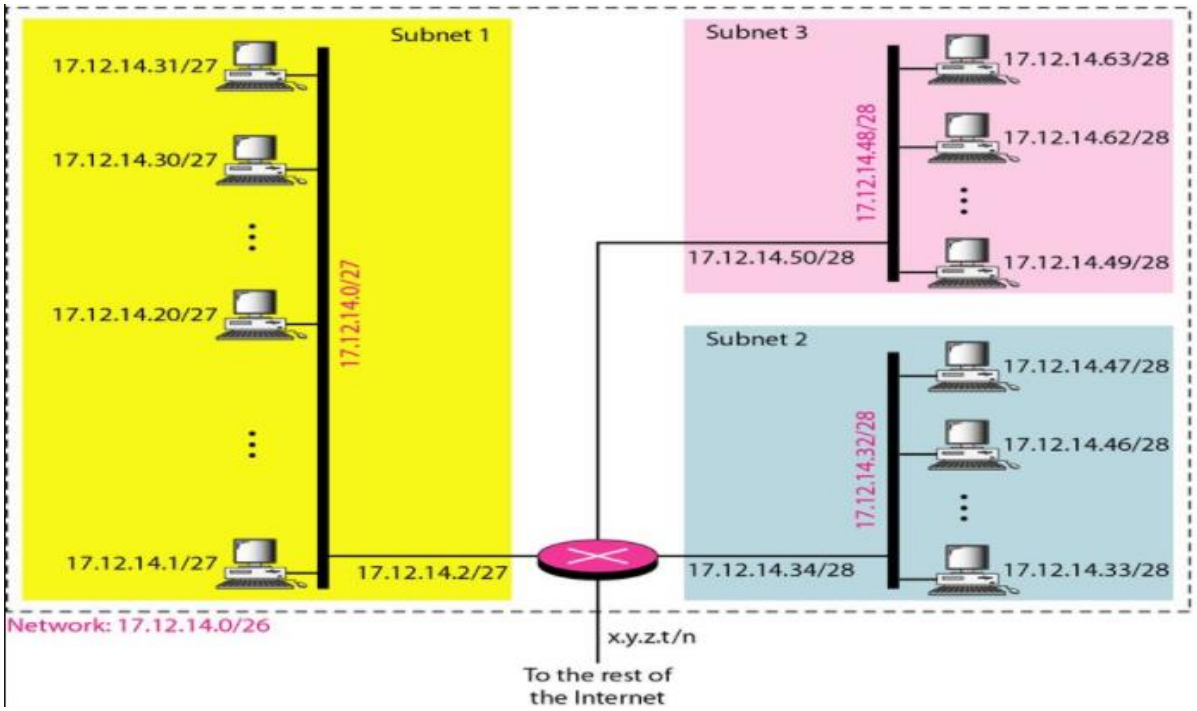
Note:- The prefix is common to all addresses in the network; the suffix changes from one device to another.

Three-Levels ofHierarchy: Subnetting

An organization that is granted a large block of addresses may want to create clusters of networks (called **subnets**) and divide the addresses between the different subnets. The rest of the world still sees the organization as one entity. However, internally there are several subnets. All messages are sent to the router address that connects the organization to the rest of the Internet; the router routes the message to the appropriate subnets. The organization, however, needs to create small sub-blocks of addresses, each assigned to specific subnets. **The organization has its own mask; each subnet must also have its own**.
As an example, suppose an organization is given the block **17.12.40.0/26**, which contains 64 addresses. The organization has three offices and needs to divide the addresses into three subblocks of 32, 16, and 16 addresses. We can find the new masks by using the following arguments:

a. Suppose the **mask for the first subnet** (subnet mask) is n1, then 2^(32- n1) must be 32, which means that n1 =27.
b. Suppose the mask for the second subnet is n2, then 2^(32- n2) must be 16, which means that n2 = 28
c. Suppose the mask for the third subnet is n3, then 2^(32- n3) must be 16, which means that n3 =28.

This means that we have the masks 27, 28, 28 with the organization mask being 26.



Organization is given the block **17.12.40.0/26:**

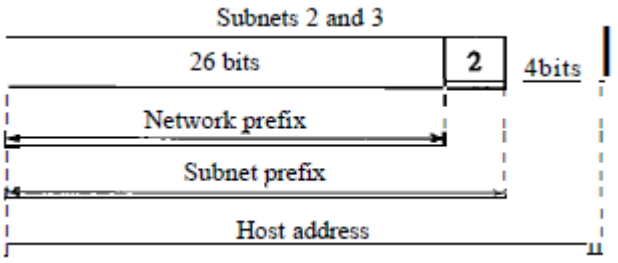
1st Customer: 32 addresses are needed, hence mask will be /27
Range: 17.12.40.0 to 17.12.40.31

2nd Customer: 16 addresses needed, hence mask will be /28
Range: 17.12.40.32 to 17.12.14.47

3rd Customer: 16 addresses needed, hence mask will be /28
Range: 17.12.14.48 to 17.12.14.53

Note:
a. 17.21.14.0/27, 17.12.14.32/28, and 17.12.14.48/28 are subnet address.
b. Applying the mask of the network, **/26**, to any of the addresses gives us the network address 17.12.14.0/26

In this case we need **2 bits** for subnetting, there will **3-level hierarchy:**



Example:
An ISP is granted a block of addresses starting with 190.100.0.0/16 (65,536 addresses). The ISP needs to distribute these addresses to three groups of customers as follows:

a. The first group has 64 customers; each needs 256 addresses.
b. The second group has 128 customers; each needs 128 addresses
c. The third group has 128 customers; each needs 64 addresses.

Design the subblocks and find out how many addresses are still available after these allocations.
Solution: 190.100.0.0/16, total IPs available 2^16

1st Group: 256 addresses are needed, hence mask will be **/24** there are 64 customers
1st Customer: 190.100.0.0/24 to 190.100.0.255
2nd Customer: 190.100.1.0/24 to 190.1001.255

64th Customer: 190.100.63.0/24 to 190.100.63.255/24

2nd Group: Each customer needs 128 addresses, hence mask **/25** will be required
1st Customer: 190.100.64.0/25 to 190.100.64.127/25 (128 addresses)
2nd Customer: 190.100.64.128/25 to 190.100.64.255/25 (128 addresses)

127th Customer: 190.100.127.0/25 to 190.100.127.127/25
128th Customer: 190.100.127.128/25 to 190.100.127.255/25

3rd Group: Each customer needs 64 addresses, hence mask **/26** will be required.
1st Customer: 190.100.128.0/26 to 190.100.128.63/26 (64 addresses)
2nd Customer: 190.100.128.64/26 to 190.100.128.127/26 (64 addresses)

128th Customer: 190.100.159.128/26 to 190.100.159.255/26 (64 addresses)

Total addresses assigned: 64*256 + 128*128 + 128*64 = 40,960
Total available addresses: 2^16 = 65536
Waste: 65536 - 40960 = 24,576

Network Address Translation (NAT):

To solve the problem of shortage of addresses, NAT enables a user to have a large set of addresses internally and one address, or a small set of addresses, externally. To separate the addresses used inside the home or business and the ones used for the Internet, the Internet authorities have reserved three sets of addresses as private addresses:

Table 19.3 *Addresses for private networks*

Range			*Total*
10.0.0.0	to	10.255.255.255	2^{24}
172.16.0.0	to	172.31.255.255	2^{20}
192.168.0.0	to	192.168.255.255	2^{16}

Note: Any organization can use an address out of this set without permission from the Internet authorities. They are unique inside the organization, but they are not unique globally. No router will forward a packet that has one of these addresses as the destination address.
The site must have only one single connection to the global Internet through a router that runs the NAT software.

The router that connects the network to the global address uses one private address and one global address.

All the outgoing packets go through the NAT router, which replaces the source address in the packet with the global NAT address. All incoming packets also pass through the NAT router, which replaces the destination address in the packet (the NAT router global address) with the appropriate private address.

IPv6 ADDRESSES

Despite all short-term solutions, such as classless addressing, Dynamic Host configuration Protocol (DHCP), and NAT, address depletion is still a long-term problem for the Internet.

An IPv6 address is 128 bits long.

IPv6 specifies hexadecimal colon notation, in this notation, 128 bits is divided into eight sections, each 2 bytes in length. Two bytes in hexadecimal notation requires four hexadecimal digits.

The leading zeros of a section (four digits between two colons) can be omitted. Only the leading zeros can be dropped, not the trailing zeros.

Original

FDEC: 0074 : 0000 : 0000 : 0000 : BOFF : 0000 : FFFO

Abbreviated FDEC: 74 : 0 : 0 : 0 : BOFF : 0 : FFFO

More abbreviated FDEC : 74 : : BOFF : 0 : FFF0

Gap

If there are consecutive sections consisting of zeros only. We can remove the zeros altogether and replace them with a double semicolon. Note that this type of abbreviation is **allowed only once** per address. If there are two runs of zero sections, only one of them can be abbreviated.

Example: Expand the address 0:15: :1:12:1213 to its original.
Solution: every digit is an hexadecimal digit, and only head zeroes are dropped.

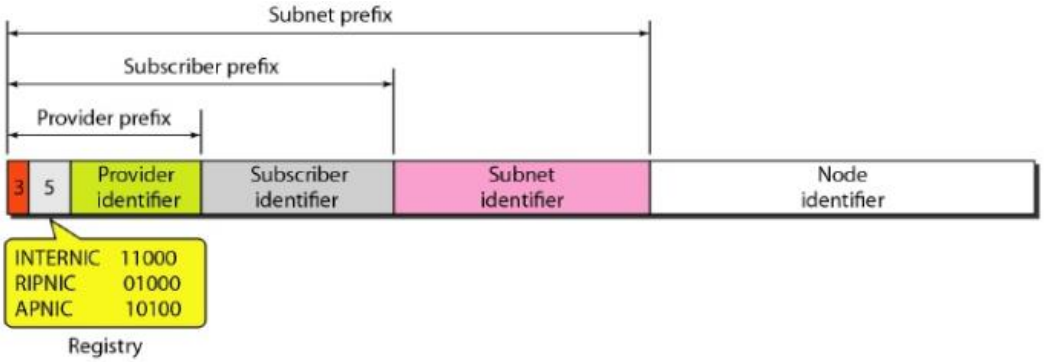
0000:0015:0000:0000:0000:0001:0012:1213

Address Space:

IPv6 has a much larger address space; **2^128**. A few leftmost bits, called the **type prefix**, in each address define its category. The type prefix is variable in length.

Unicast Addresses:

A **unicast** address defines a single computer. The packet sent to a unicast address must be delivered to that specific computer. IPv6 defines two types of unicast addresses: **geographically** based and **provider-based**. The p**rovider-based address is generally used by a normal host as a unicast address**.



Prefixes for provider-based unicast address

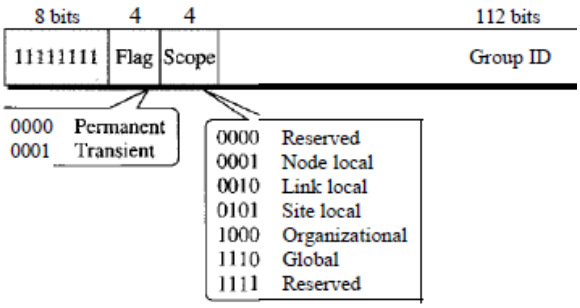
- **Type identifier** This 3-bit field defines the address as a provider-based address.
- **Registry identifier** This 5-bit field indicates the agency that has registered the address**.**
- **Provider identifier** variable-length(16 bits recommended) field identifies the provider for Internet access
- **Subscriber identifier** When an organization subscribes to the Internet through a provider, it is assigned a subscriber identification (24 bits)
- **Subnet identifier** It defines a specific subnetwork under the territory of the subscriber (32 bits)

- **Node identifier** the identity of the node connected to a subnet (48 bits). A length of 48 bits is recommended for this field to make it compatible with the 48-bit link (**physical**) address used by Ethernet.

Multicast Addresses

Multicast addresses are used to define a group of hosts instead of just one. A packet sent to a multicast address must be delivered to each member of the group.

Figure 19.17 *Multicast address in IPv6*



The second field is a flag that defines the group address as either **permanent** or **transient**. A permanent group address is defined by the Internet authorities and can be accessed at all times. A transient group address, on the other hand, is used only temporarily.

Anycast Addresses

An anycast address, like a multicast address, also defines a group of nodes. However, a packet destined for an anycast address is delivered to only one of the members of the anycast group, the nearest one (the one with the shortest route).

Reserved Addresses

These addresses start with eight 0s (type prefix is 00000000).

a. An **unspecified address** is used when a host does not know its own address and sends an inquiry to find its address.

b. A **loopback address** is used by a host to test itself without going into the network.

c. A **compatible address** is used during the transition from IPv4 to IPv6. It is used when a computer using IPv6 wants to send a message to another computer using IPv6, but the message needs to pass through a part of the network that still operates in IPv4.

d. A **mapped address** is also used during transition. However, it is used when a computer that has migrated to IPv6 wants to send a packet to a computer still using IPv4.

Local Addresses They provide addressing for private networks without being connected to the Internet.

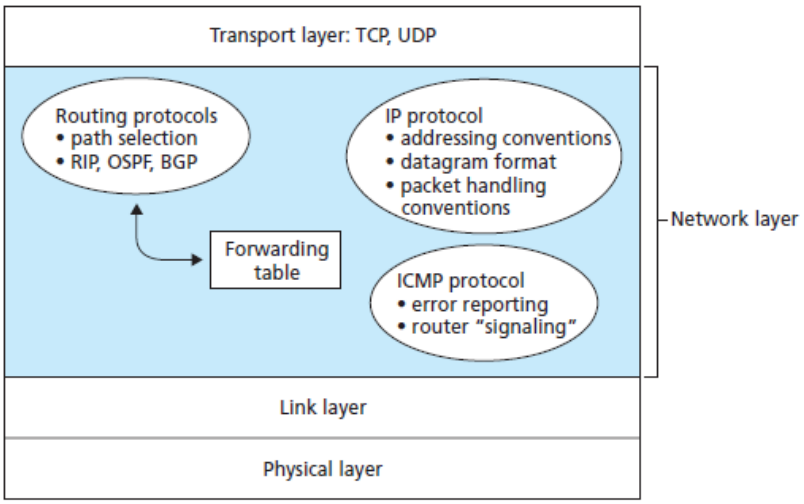
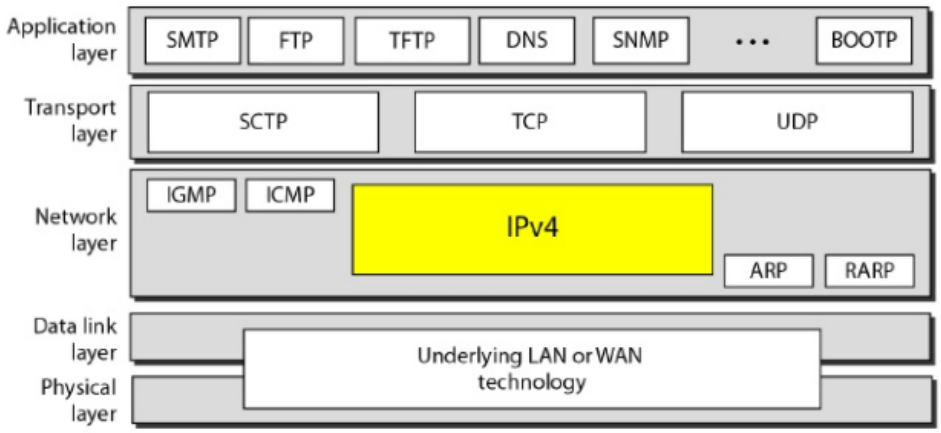


Figure 4.12 ♦ A look inside the Internet's network layer

Internet's network layer has three major components:

a. The first component is the **IP protocol**
b. The second major component is the **routing component**, which determines the path a datagram follows from source to destination. The routing protocols compute the **forwarding tables** that are used to forward packets through the network.
c. The final component of the network layer is a facility to **report errors** in datagrams.

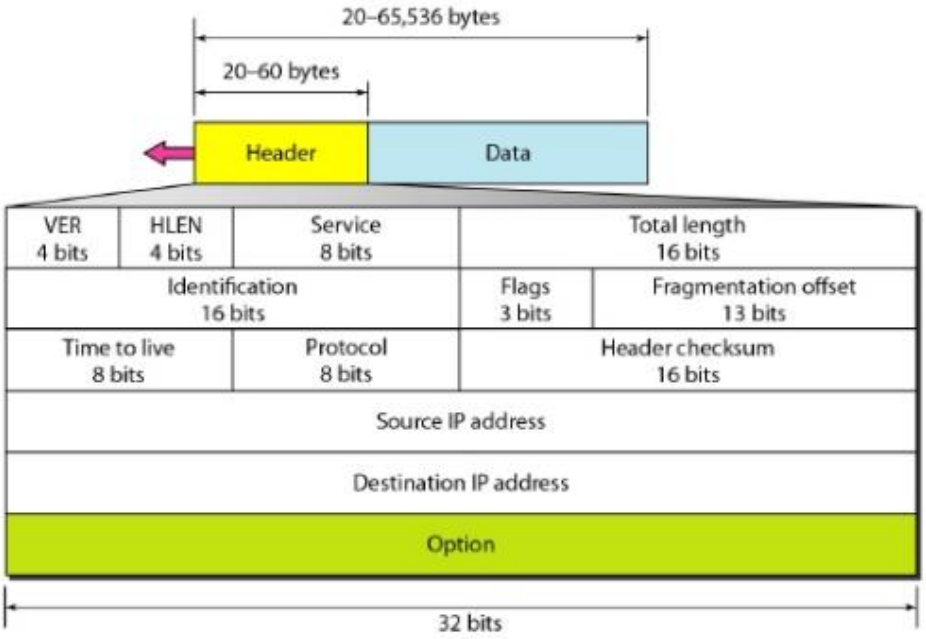
IPV4 Datagram Format:



Position of IPv4 in TCP/IP protocol suite

IPv4 is an **unreliable** and **connectionless** datagram protocol a **best-effort** delivery service. The term best-effort means that IPv4 provides no error control or flow control (except for **error detection on the header**). If reliability is important, IPv4 must be paired with a reliable protocol such as TCP.

IPv4 is also a connectionless protocol for a packet-switching network that uses the datagram approach. Each datagram is handled **independently**, and each datagram can follow a different route to the destination. This implies that datagrams sent by the same source to the same destination could arrive out of order. Also, some could be lost or corrupted during transmission.



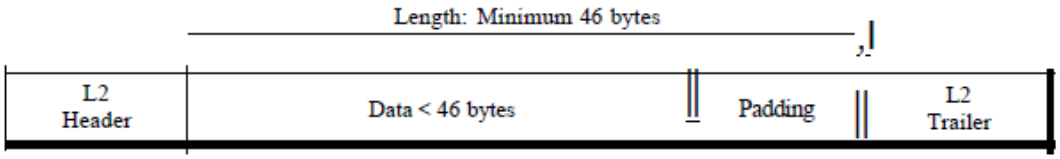
IPV4 Datagram Format

A datagram is a **variable-length** packet consisting of two parts: **header and data**. The header is **20 to 60 bytes** in length and contains information essential to routing and delivery.

a. **Version (ver)** 4 bits specify the IP protocol version of the datagram IPV4 or IPV6
b. **Header length (HLEN)** This 4-bit field defines the total length of the datagram header in bytes. This field is needed because the length of the header is variable (between 20 and 60 bytes). When there are no options, the header length is 20 bytes, and the value of this field is 5 (5 x 4 = 20). When the option field is at its maximum size, the value of this field is 15 (15 x 4 = 60 bytes).
c. **Total Length It** defines the total length (**header plus data**) of the IPv4 datagram in bytes. To find the length of the data coming from the upper layer, subtract the header length from the total length.

Since the field length is 16 bits, the total length of the IPv4 datagram is limited to **65,535** (2^16 - 1) bytes.

Figure 20.7 *Encapsulation of a small datagram in an Ethernet frame*



Data is less than 46 bytes, so padding is done as the minimum and maximum restriction of the size of that can be encapsulated in a frame (**46 to 1500 bytes**).

d. **Identifier, flags, fragmentation offset** these three fields have to do with IP fragmentation.

IPv6, does not allow for fragmentation at routers

e. **Time to live**

The time-to-live (TTL) field is included to ensure that datagrams do not circulate forever. This field is decremented by one each time the datagram is processed by a router. If the TTL field reaches 0, the datagram must be dropped. This value is approximately 2 times the maximum number of routes

between any two hosts.
Another use of this field is to intentionally limit the journey of the packet. For example, if the source wants to confine the packet to the local network, it can **store 1** in this field. When the packet arrives at the first router, this value is decremented to 0, and the datagram is discarded.

f. **Protocol** this 8-bit field defines the higher-level protocol that uses the services of the IPv4 layer. An IPv4 datagram can encapsulate data from several higher-level protocols such as TCP, UDP, ICMP, and IGMP.
g. **Header Checksum** The header checksum aids a router in detecting bit errors in a received IP datagram. A router computes the header checksum for each received IP datagram and detects an error condition if the checksum carried in the datagram header does not equal the computed checksum. Routers typically discard datagrams for which an error has been detected. **Note that the checksum must be recomputed and stored again at each router**, as the **TTL field, and possibly the options field as well**, may change.
h. **Source and destination IP addresses** when a source creates a datagram, it inserts its IP address into the source IP address field and inserts the address of the ultimate destination into the destination IP address field. Often the source host determines the destination address via a **DNS lookup**.
i. **Options** The options fields allow an IP header to be extended.
j. **Data (payload)** the data field of the IP datagram contains the transport-layer segment (TCP or UDP) to be delivered to the destination. However, the data field can carry other types of data, such as ICMP messages.

Note: - An IP datagram has a total of 20 bytes of header (assuming no options). If the datagram carries a TCP segment, then each (non-fragmented) datagram carries a total of 40 bytes of header (**20 bytes of IP header plus 20 bytes of TCP header**) along with the application-layer message.

IP Datagram Fragmentation:

A datagram can travel through different networks. Each router **decapsulates** the IPv4 datagram from the frame it receives, processes it, and then encapsulates it in another frame. The format and size of the received frame depend on the protocol used by the physical network through which the frame has just travelled. The format and size of the sent frame depend on the protocol used by the physical network through which the frame is going to travel.

Maximum Transfer Unit (MTU):

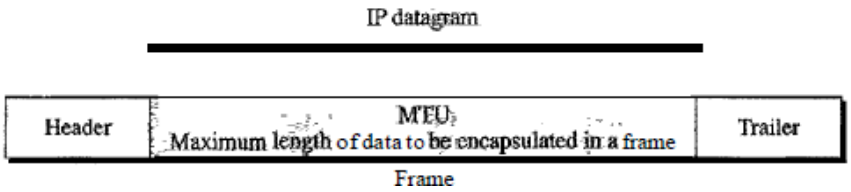
The maximum amount of data that a link-layer frame can carry is called the **maximum transmission unit (MTU)**. Because each IP datagram is encapsulated within the link-layer frame for transport from one router to the next router, the MTU of the link-layer protocol places a hard limit on the length of an IP datagram. When a datagram is encapsulated in a frame, the total size of the datagram must be less than this maximum size, A router that interconnects several links, each running different data link-layer protocols with different MTUs. Suppose router receives

an IP datagram from one link. Router checks its forwarding table to determine the outgoing link, and this outgoing link has an MTU that is smaller than the length of the IP datagram. The solution is to fragment the data in the IP datagram into two or **smaller IP datagrams**, **encapsulate each of these smaller IP datagrams in a separate link-layer frame**; and send these frames over the outgoing link. Each of these smaller datagrams is referred to as a **fragment**.

Note:

a. **Fragments need to be reassembled before they reach the transport layer at the destination.**
b. **The value of the MTU depends on the physical network protocol**

Figure 20.9 *Maximum transfer unit (MTU)*



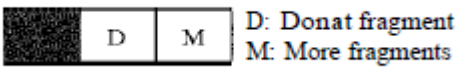
To allow the destination host to perform these reassembly tasks, the designers of IPV4 put **identification**, **flag**, and **fragmentation offset** fields in the IP datagram header. When a datagram is fragmented, each fragment has its own header with most of the fields repeated, but with some changed. A fragmented datagram may itself be fragmented if it encounters a network with an even smaller MTU.

Note: **In IPv4, a datagram can be fragmented by the source host or any router in the path** although there is a tendency to limit fragmentation only at the source. The reassembly of the datagram, however, is **done only by the destination host** because each fragment becomes an independent datagram. The fragmented datagram can travel through different routes.
When a datagram is fragmented, required parts of the header must be copied by all fragments. **The option field may or may not be copied**.

The host or router that fragments a datagram must change the values of three fields: **flags**, **fragmentation offset**, and **total length**.

Fields Related to Fragmentation:

a. **Identification** (16 bits) the combination of the **identification and source IPv4** address must uniquely define a datagram as it leaves the source host. To guarantee uniqueness, the IPv4 protocol uses a **counter** to label the datagrams. The counter is initialized to a positive number. When the IPv4 protocol sends a datagram, it copies the current value of the counter to the identification field and increments the counter by 1.
b. **Flags (3 bits)**



D (Don't fragment): If its value is 1 the machine must not fragment the datagram. If it cannot pass the datagram through any available physical network, it discards the datagram and sends an ICMP error message to the source host.

M (more fragment): If its value is 1, it means the datagram is not the last fragment; there are more fragments after this one. If its value is 0, it means this is the last or only fragment.

Fragmentation offset (13 bits):

It shows the **relative position** of this fragment with respect to the whole datagram. Offset is measured in units of 8 bytes **(8 is scaling factor)**. This is done because the length of the offset field is only 13 bits and cannot represent a sequence of bytes greater than 8191 but the total length of IP datagram possible is 2^16 (2^13*2^3).

Example: A datagram of 4,000 bytes (20 bytes of IP header plus 3,980 bytes of IP payload) arrives at a router and must be forwarded to a link with an MTU of 1,500 bytes. (Kruose)

Solution:

3,980 data bytes in the original datagram must be allocated to three separate fragments (each of which is also an IP datagram). Suppose that the original datagram is stamped with an identification Number of 777.

The amount of original payload data in all but the last fragment be a multiple of 8 bytes

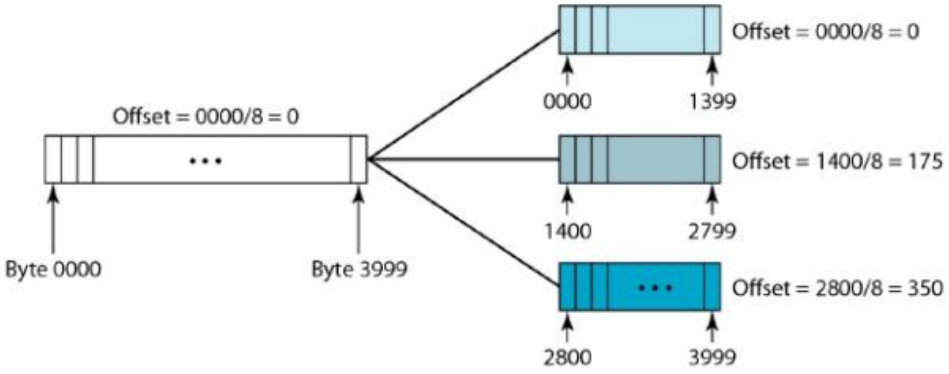
Fragment	Bytes	ID	Offset	Flag
1st fragment	1,480 bytes in the data field of the IP datagram	identification = 777	offset = 0 (meaning the data should be inserted beginning at byte 0)	flag = 1 (meaning there is more)
2nd fragment	1,480 bytes of data	identification = 777	offset = 185 (meaning the data should be inserted beginning at byte 1,480. Note that 185 · 8 = 1,480)	flag = 1 (meaning there is more)
3rd fragment	1,020 bytes (= 3,980–1,480–1,480) of data	identification = 777	offset = 370 (meaning the data should be inserted beginning at byte 2,960. Note that 370 · 8 = 2,960)	flag = 0 (meaning this is the last fragment)

At the destination, the payload of the datagram is passed to the transport layer only after the IP layer has fully reconstructed the original IP datagram. If one or more of the fragments does not arrive at the destination, the incomplete datagram is discarded and not passed to the transport layer. If TCP is being used at the transport layer, then TCP will recover from this loss by having the source retransmit the data in the original datagram.

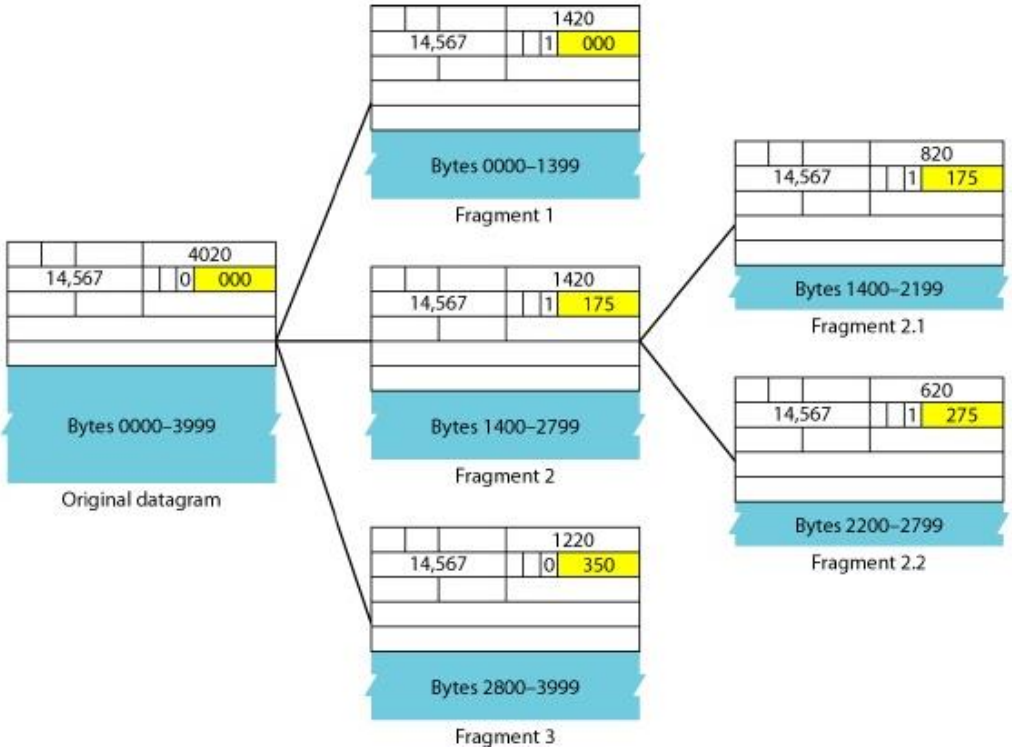
NOTE:

a. **Fragmentation also has its costs.**
b. **Fragmentation can be used to create lethal DoS attacks, whereby the attacker sends a series of bizarre and unexpected fragments, and few other attacks also.**

Example 2: A datagram with a data size of 4000 bytes fragmented into three fragments. Following is the solution:



Detailed fragment example, if a fragment itself gets fragmented:



Fragement2: Identification no.: 14,567 Total Length: 1420, offset value=175 MF=1, second fragment is itself further divided into two fragments

Fragment 2.1: identification: 14567, Total Length=820, Payload: 800 Bytes (1400-2199) offset=1400/8=175

Fragment 2.2: identification: 14567, Total Length=620, Payload: 600 Bytes (2200-2799) offset=2200/8=175 **(first Byte number is divided by 8)**

Note:

1. The first fragment has an offset field value of zero.
2. Divide the length of the first fragment by 8. The second fragment has an offset value equal to that result.
3. Divide the total length of the first and second fragments by 8. The third fragment has an offset value equal to that result.

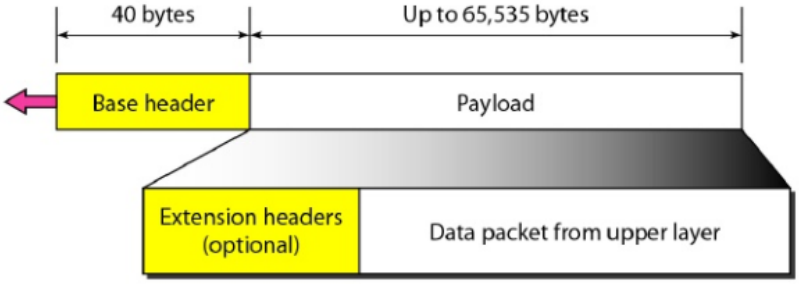
IPv6 Datagram Format:

IPv4 has some deficiencies that make it unsuitable for the fast-growing Internet:

a. Despite all short-term solutions, such as subnetting, classless addressing, and NAT, address depletion is still a long-term problem in the Internet.
b. The Internet must accommodate real-time audio and video transmission.
c. The Internet must accommodate encryption and authentication of data for some applications. No encryption or authentication is provided by IPv4.

Advantages of IPV6:

a. **Larger address space**: An IPv6 address is 128 bits long, 2^128 address space
b. **Better header format** IPv6 uses a new header format in which options are separated from the base header and inserted, when needed, between the base header and the upper-layer data.
c. **New options:** IPv6 has new options to allow for additional functionalities.
d. **Allowance for extension** IPv6 is designed to allow the extension of the protocol if required by new technologies.
e. **Support for resource allocation:** the type-of-service field has been removed but a mechanism has been added to enable the source to request special handling of the packet. This mechanism can be used to support traffic such as real-time audio and video.
f. **Support for more security** the encryption and authentication.



IPv6 datagram header and payload

Packet Format:

Each packet is composed of a mandatory base header followed by the payload. The payload consists of two parts: optional extension headers and data from an upper layer. **The base header occupies fixed 40 bytes**, whereas the extension headers and data from the upper layer contain up to **65,535 bytes** of information.

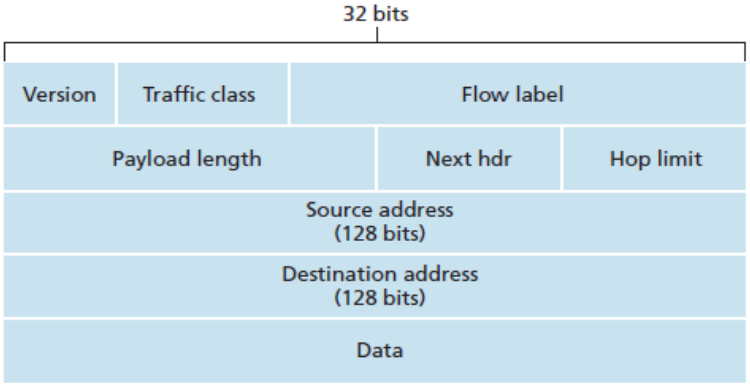


Figure 4.24 ♦ IPv6 datagram format

Base Header Base address with its eight fields.

a. **Version** This 4-bit field defines the version number of the IP. For IPv6, the value is 6.
b. **Traffic Class** this 8-bit field is similar in spirit to the TOS field we saw in IPv4 (**Kruose**). In Forozuan **Priority** (4 bit) is given at the place of Traffic Class. The priority field of the IPv6 packet defines the priority of each packet with respect to other packets from the same source. The size of Traffic Class field is 8 bits. Traffic Class field is similar to the IPv4 Type of Service (ToS) field. The Traffic Class field indicates the IPv6 packet's class or priority.
c. **Flow Label** A sequence of packets, sent from a particular source to a particular destination that needs special handling by routers is called a flow of packets. The combination of the source address and the value of the flow label uniquely defines a flow of packets. To a router, a flow is a sequence of packets that share the same characteristics.
d. **Payload length:** The 16-bits payload length field defines the length of the IP datagram excluding the base header.
e. **Next header** The field uses the same values as the protocol field in the IPv4 header., the Next header field tells which transport protocol handler (e.g., TCP, UDP) to pass the packet to. This field identifies the protocol to which the contents (data field) of this datagram will be delivered.
f. **Hop limit** same as TTL field in IPV4, The contents of this field are decremented by one by each router that forwards the datagram. If the hop limit count reaches zero, the datagram is discarded.
g. **Source and destination addresses** 128-bits source/destination address
h. **Data:** This is the payload portion of the IPv6 datagram.

Fragmentation/Reassembly:

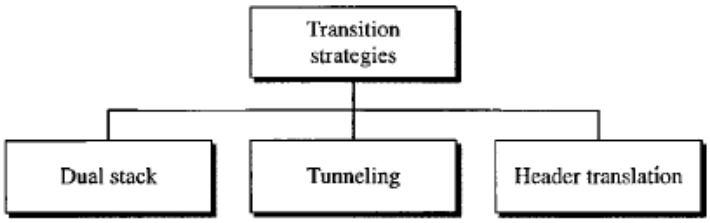
IPv6 does not allow for fragmentation and reassembly at intermediate routers; these operations can be performed only by the source and destination. If an IPv6 datagram received by a router is too large to be forwarded over the outgoing link, the router simply drops the datagram and sends a "Packet Too Big" ICMP error message back to the sender. The sender can then resend the data, using a smaller IP datagram size.

Header checksum because the transport-layer (for example, TCP and UDP) and link-layer (for example, Ethernet) protocols in the Internet layers perform **checksumming**, the designers of IP probably felt that this functionality was sufficiently redundant in the network layer that it could be removed.

Options An options field is no longer a part of the standard IP header. However, it has not gone away.

Transitioning from IPv4 to IPv6

Three transition strategies



a. **Dual Stack** is recommended that all hosts, before migrating completely to version 6, have a dual stack of protocols. In other words, a station must run IPv4 and IPv6 simultaneously until all the Internet uses IPv6.
   To determine which version to use when sending a packet to a destination, the source host queries the DNS. If the DNS returns an IPv4 address, the source host sends an IPv4 packet. If the DNS returns an IPv6 address, the source host sends an IPv6 packet.
b. **Tunneling** is a strategy used when two computers using IPv6 want to communicate with each other and the packet must pass through a region that uses IPv4. To pass through this region, the packet must have an IPv4 address. So the IPv6 packet is encapsulated in an IPv4 packet when it enters the region, and it leaves its capsule when it exits the region.
c. **Header Translation** The sender wants to use IPv6, but the receiver does not understand IPv6. The header of the IPv6 packet is converted to an IPv4 header.

Network Layer: Address Mapping

We need protocols to create a mapping between physical and logical addresses. IP packets use logical (host-to-host) addresses. These packets, however, need to be encapsulated in a frame, which needs physical addresses (node-to-node).
Lack of flow and error control in the Internet Protocol has resulted in another protocol, **ICMP** that provides alerts.

ADDRESS MAPPING:

An internet is made of a combination of physical networks connected by internetworking devices such as routers. A packet starting from a source host may pass through several different physical networks before finally reaching the destination host. The hosts and routers are recognized at the network level by their logical (IP) addresses. However, packets pass through physical networks to reach these hosts and routers. At the physical level, the hosts and routers are recognized by their physical addresses.
A physical address is a local address. It must be unique locally, but is **not necessarily unique universally**. It is called a physical address because it is usually (but not always) implemented in hardware. An example of a physical address is the **48-bit MAC address** in the Ethernet protocol.

ARP (Logical to Physical Address)

Anytime a host or a router has an IP datagram to send to another host or router, it has the logical (IP) address of the receiver. **The logical (IP) address is obtained from the DNS**. The IP datagram must be encapsulated in a frame to be able to pass through the physical network. This means that the sender needs the physical address of the receiver.

The host or the router sends an **ARP query packet**. The packet includes the physical and IP addresses of the sender and the IP address of the receiver. The query is broadcast over the network. Every host or router on the network receives and processes the ARP query packet, but only the intended recipient recognizes its IP address and sends back an **ARP response packet**.

The response packet contains the recipient's IP and physical addresses. The **packet is unicast directly to the inquirer** by using the physical address received in the query packet.

Cache Memory

Using ARP is **inefficient** if a system needs to broadcast an ARP request for each IP packet it needs to send to system B. It could have broadcast the IP packet itself. **ARP can be useful if the ARP reply is cached** A system that receives an ARP reply stores the mapping in the cache memory. **Before sending an ARP request**, the system first checks its cache to see if it can find the mapping. An ARP packet is encapsulated directly into a data link frame.

An ARP request is broadcast; an ARP reply is unicast

Mapping Physical to Logical Address (RARP, BOOTP, and DHCP)

There are occasions in which a host knows its physical address, but needs to know its logical address. This may happen in two cases:

1. A diskless station is just booted. The station can find its physical address by checking its interface, but it does not know its IP address.
2. An organization does not have enough IP addresses to assign to each station; it needs to assign IP addresses on demand. The station can send its physical address and ask for a short time lease.

Reverse Address Resolution Protocol (RARP)

Finds the logical address for a machine that knows only its physical address. **Each host or router is assigned one or more logical (IP) addresses**, which are unique and independent of the physical address of the machine. To create an IP datagram, a host or a router needs to know its own IP address or addresses. The IP address of a machine is usually read from its configuration file stored on a disk file. However, a diskless machine is usually booted from ROM, which has minimum booting information. **A RARP request is created and broadcast on the local network**. Another machine on the local network **that knows all the IP addresses** will respond with a RARP reply. The requesting machine must be running a RARP client program; the responding machine must be running a RARP server program.

Note: RARP is almost obsolete. Two protocols, BOOTP and DHCP, are replacing RARP.

The Bootstrap Protocol (BOOTP)

It is a client/server protocol designed to provide physical address to logical address mapping. **BOOTP is an application layer protocol**. The administrator may put the client and the server on the same network or on different networks. BOOTP messages are encapsulated in a UDP packet, and the UDP packet itself is encapsulated in an IP packet.

The reader may ask how a client can send an IP datagram when it knows neither its own IP address (the source address) nor the server's IP address (the destination address**). The client simply uses all 0's as the source address and all 1's as the destination address**.

BOOTP is a static configuration protocol

Binding between the physical and IP addresses is static and fixed in a table until changed by the administrator.

The Dynamic Host Configuration Protocol (DHCP)

BOOTP is **not a dynamic configuration protocol**.
When a client requests its IP address, the BOOTP server consults a table that matches the physical address of the client with its IP address. This implies that the binding between the physical address And the IP address of the client already exists. **The binding is predetermined**.

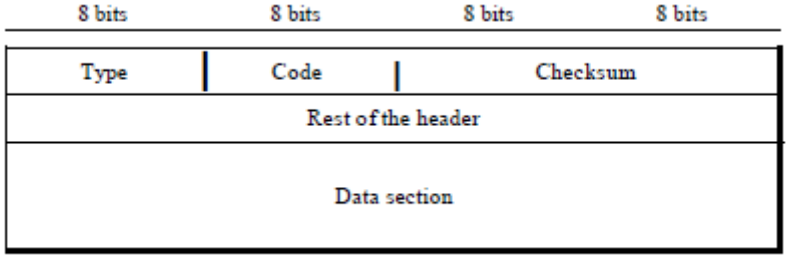
DHCP provides static and dynamic address allocation that can be manual or automatic.

When a DHCP client sends a request to a DHCP server, the server first checks its **static database**. If an entry with the requested physical address exists in the static database, the permanent IP address of the client is returned. If the entry does not exist in the static database, the server selects an IP address from the available pool, assigns the address to the client, and adds the entry to the dynamic database.

The Internet Control Message Protocol (ICMP)

The IP protocol has no **error-reporting** or **error-correcting** mechanism, has been designed to compensate for the above two deficiencies.

Types of Messages: ICMP messages are divided into two broad categories: **error-reporting** messages and **query** messages. The **error-reporting messages may be reported** problems by a router or a host destination) may encounter when it processes an IP packet.
The **query messages**, which occur in pairs, help a host or a network manager get specific information from a router or another host. For example, nodes can discover their neighbours.



General format of ICMP Message

An ICMP message has an 8-byte header and a variable-size data section. Data section carries error message or other information for query.

Error Reporting

ICMP uses the source IP address to send the error message to the source (originator) of the datagram. **ICMP always reports error messages to the original source.**

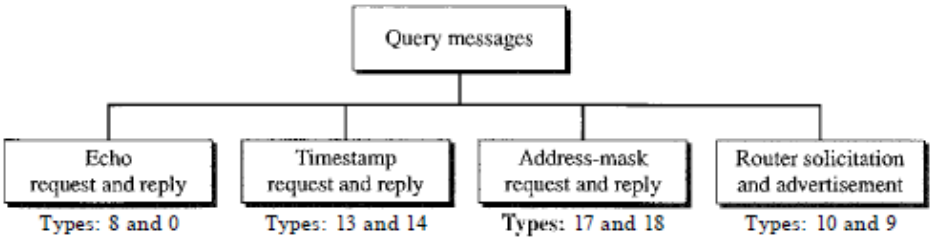
Five types of errors are handled:

a. **Destination Unreachable** when a router cannot route a datagram or a host cannot deliver a datagram, the datagram is discarded and the **router or the host**.

b. **Source quench** a router or a host has a limited-size queue (buffer) for incoming datagrams waiting **to be forwarded** (in the case of a router) or **to be processed** (in the case of a host). When a router or host discards a datagram **due to congestion**, it sends a source-quench message to the sender of the datagram.
c. **Time exceeded** in two cases, either TTL reaches 0 or when not all fragments that make up a message arrive at the destination host within a certain time limit.
d. **Parameter problems** an ambiguous or missing value in any field of the datagram.
e. **Redirection** When a router needs to send a packet destined for another network

QUERY:

ICMP can diagnose some network problems. This is accomplished through the query messages. A node sends a message that is answered in a specific format by the destination node. A query message is encapsulated in an IP packet, which in tum is encapsulated in a data link layer frame.



a. **Echo Request and Reply** the echo-request and echo-reply messages are designed for diagnostic purposes. These can be used to check if any communication happening at IP level. Because ICMP messages are encapsulated in IP datagrams. E.g. ping command
b. **Timestamp Request and Reply** two machines (hosts or routers) can determine the round-trip time needed for an IP datagram to travel between them or to synchronize the clocks in two machines.
c. **Address-Mask Request and Reply** a host may know its IP address, but it may not know the corresponding mask.

Traceroute:

It uses two ICMP messages, time exceeded and destination unreachable, to find the route of a packet. This is a program at the application level that uses the services of UDP.
The **traceroute** program uses the ICMP messages and the TTL field in the IP packet to find the route.

IGMP

The Internet Group Management Protocol (IGMP) is one of the necessary, but not sufficient protocols that is involved in **multicasting**. IGMP is a companion to the IP protocol.

IGMP is not a multicasting routing protocol; it is a protocol that manages group membership

Network Layer: Forwarding, and Routing

Forwarding:

1. **Next-Hop Method Versus Route Method** the routing table holds only the address of the next hop instead of information about the complete route (**route method**).

22.2 *Route method versus next-hop method*

a. Routing tables based on route

Destination	Route
HostB	R1, R2, host B

Routing table for host A

Destination	Route
HostB	R2, host B

Routing table for R1

Destination	Route
HostB	HostB

Routing table for R2

b. Routing tables based on next hop

Destination	Next hop
Host B	R1

Destination	Next hop
HostB	R2

Destination	Next hop
Host B	

Network-Specific Method v/s Host-Specific Method:

Instead of having an entry for every destination host connected to the same physical network (host-specific method), we have only one entry that defines the address of the destination network itself.

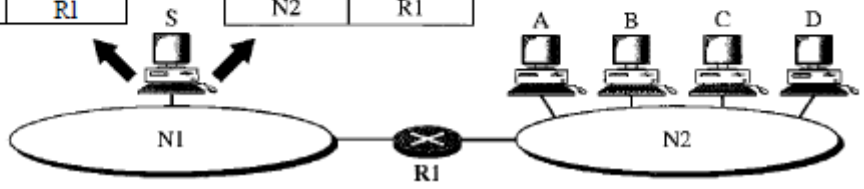
22.3 *Host-specific versus network-specific method*

Routing table for host S based on host-specific method

Destination	Next hop
A	R1
B	R1
C	R1
D	R1

Routing table for host S based on network-specific method

Destination	Next hop
N2	R1

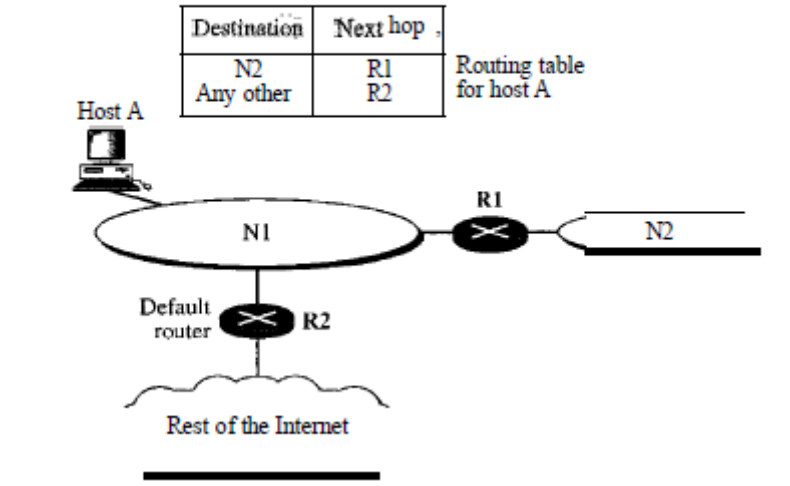


We treat all hosts connected to the same network as one single entity.

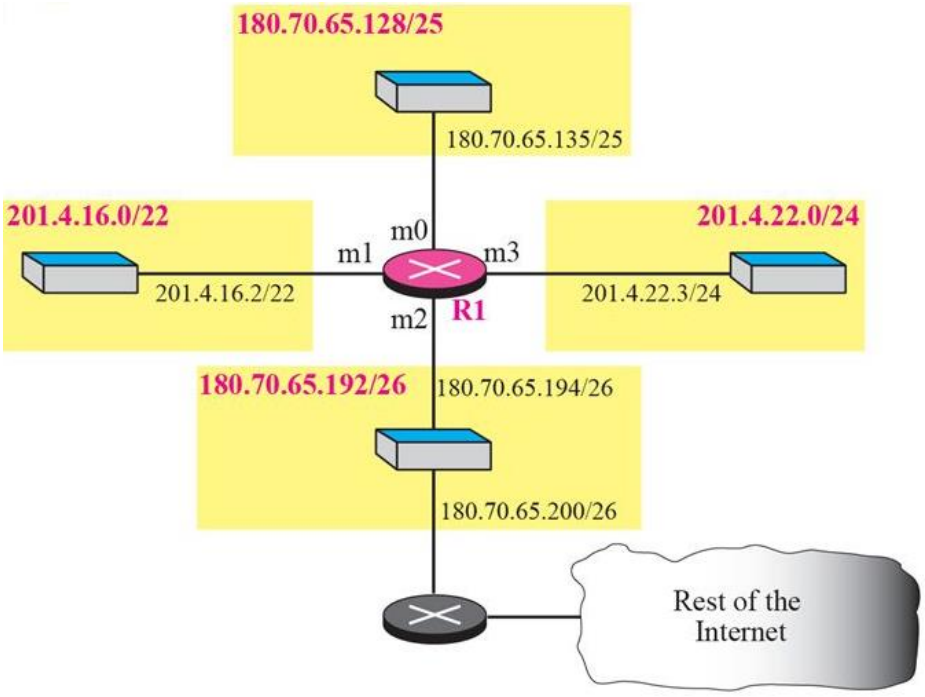
Default Method

Host A is connected to a network with two routers. Router R1 routes the packets to hosts connected to network N2. However, for the rest of the Internet, router R2 is used. So instead of listing all networks in the entire Internet, host A can just have one entry called the **default**.

Default method



Forwarding Process following is the configuration given for router R1.



shows the corresponding table.

Table 22.1 *Routing table for router R1 in Figure 22.6*

Mask	*Network Address*	*Next Hop*	*Interface*
/26	180.70.65.192	-	m2
/25	180.70.65.128	-	m0
/24	201.4.22.0	-	m3
/22	201.4.16.0	....	m1
Any	Any	180.70.65.200	m2

Example: Show the forwarding process if a packet arrives at R1 in with the destination address 180.70.65.140.
Solution: The router performs the following steps:

1. The first mask (/26) is applied to the destination address. The result is 180.70.65.128, which does not match the corresponding network address.
2. The second mask (/25) is applied to the destination address. The result is 180.70.65.128, which matches the corresponding network address, and the interface number m0 are passed to ARP for further processing.

Longest Mask Matching:
The routing table is sorted from the longest mask to the shortest mask. In other words, if there are three masks /27, /26, and /24, the mask /27 must be the first entry and /24 must be last.

Routing Table:

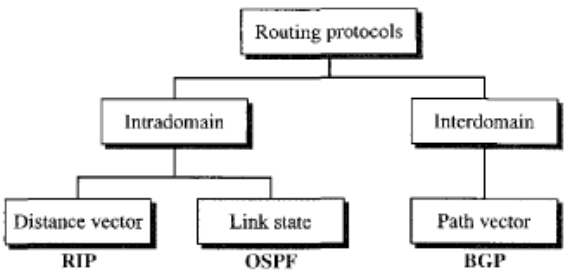
A host or a router has a routing table with an entry for each destination, or a combination of destinations, to route IP packets. The routing table can be either **static** or **dynamic**.

Static Routing Table A static routing table contains information entered manually. The administrator enters the route for each destination into the table. When a table is created, it cannot update automatically when there is a change in the Internet. The table must be manually altered by the administrator.

Dynamic Routing Table
A dynamic routing table is updated periodically by using one of the dynamic routing protocols such as RIP, OSPF, or BGP.
A routing protocol is a combination of rules and procedures that lets routers in the internet inform each other of changes. The routing protocols also include procedures for combining information received from other routers.

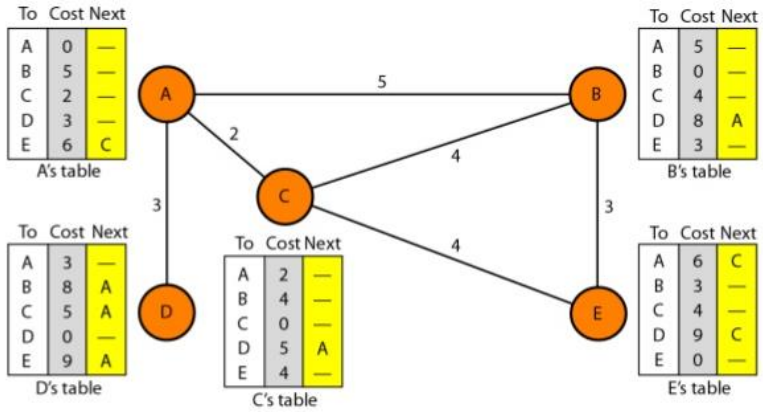
Optimization:
When a router receives a packet, to which network should it pass the packet? The decision is based on optimization: Which of the available pathways is the optimum pathway?
One approach is to assign a cost for passing through a network. **We call this cost a metric**. However, the metric assigned to each network depends on the type of protocol.
The **Routing Information Protocol (RIP)**, treat all networks as equals. The cost of passing through a network is the same; **it is one hop count**.
Open Shortest Path First (OSPF), allow the administrator to assign a cost for passing through a network based on the type of service required.

Popular routing protocols



Distance Vector Routing:

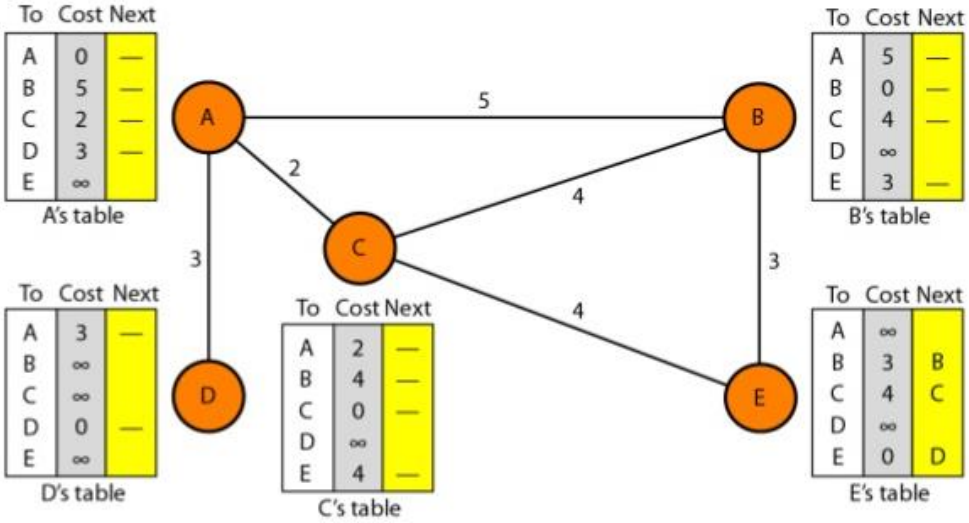
The least-cost route between any two nodes is the route with minimum distance. **Each node maintains a vector** (table) of minimum distances to every node. The table at each node also guides The packets to the desired node by showing the next stop in the route (next-hop routing).



22.14 Distance vector routing tables

Initialization:

The tables in Figure 22.14 are **stable**; each node knows how to reach any other node And the cost. At the beginning, however, this is not the case. Each node can know only the distance between itself and its immediate neighbours those directly connected to it.



Initialization of tables in distance vector routing

Sharing:

The whole idea of distance vector routing is the sharing of information **between neighbours**.

Immediate neighbours, can improve their routing tables if they help each other.

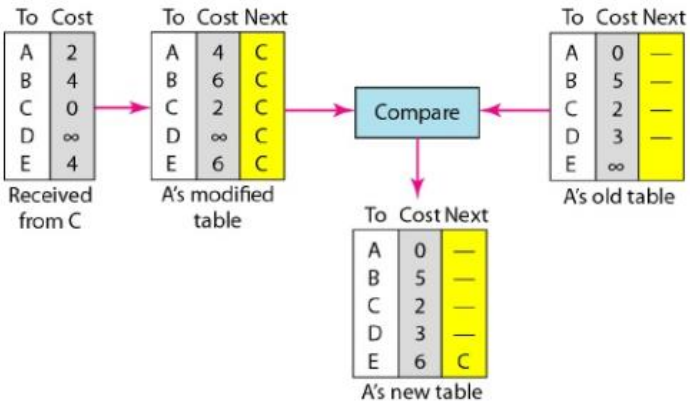
For each node is to send its **entire table to the neighbor** and let the neighbor decide what part to use and what part to discard. When the neighbor receives a table, third column needs to be replaced with the sender's name**. A node therefore can send only the first two columns of its table to any neighbor**.

In distance vector routing, each node shares its routing table with its immediate neighbors periodically and when there is a change.

Updating:

When a node receives a two-column table from a neighbour, it needs to update its routing table.

1. The receiving node needs to add the cost between itself and the sending node to each value in the second column. If node C claims that its distance to a destination is **x**, and the distance between A and C is **y**, then the distance between A and that destination, via C, is **x + y**.
2. The receiving node needs to add the name of the sending node to each row as the third column if the receiving node uses information from any row.
3. The receiving node needs to compare each row of its old table with the corresponding row of the modified version of the received table.
   a. If the **next-node entry is different**, the receiving node chooses the row with the smaller cost. **If there is a tie, the old one is kept**.
   b. If the **next-node entry is the same** (same node is advertising new value now), the receiving node chooses the new row. For example, suppose node C has previously advertised a route to node X with distance 3. Suppose that now there is no path between C and X; node C now advertises this route with a distance of infinity. Node A must not ignore this value even though its old entry is smaller. The old route does not exist anymore. The new route has a distance of infinity.



Points to be noted here:

1. When we add any number to infinity, the result is still infinity.
2. If A needs to reach itself via C, it needs to go to C and come back, a distance of 4.

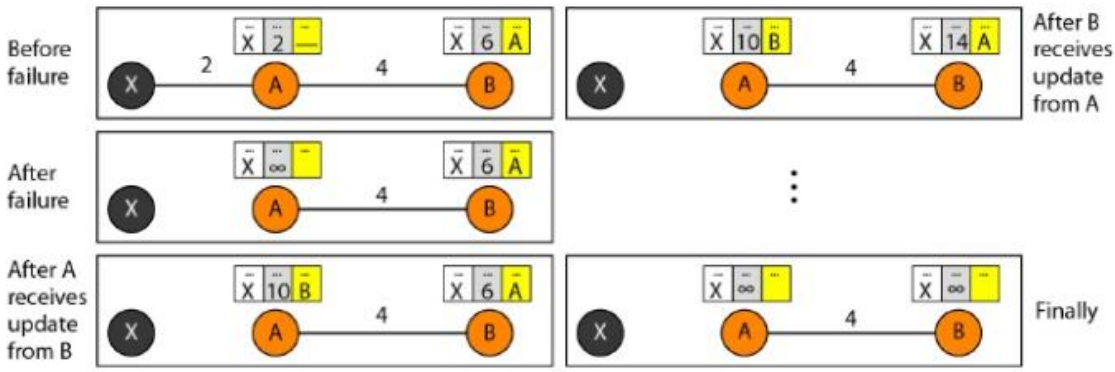
When to Share

When does a node send its **partial** routing table (only two columns) to all its **immediate neighbours**?

a. **Periodic Update** A node sends its routing table, normally every 30 s, in a periodic update.
b. **Triggered Update** A node sends its two-column routing table if there is a **change in its routing table**. This is called a **triggered** update. The **update** can result from the following:
   1. A node receives a table from a neighbour, resulting in changes in its own table after updating.
   2. A node detects some failure in the neighbouring links which results in a distance change to infinity.

Two-Node Loop Instability

A problem with distance vector routing is **instability**, which means that a network using this protocol can become unstable.



Two-node instability

Both nodes A and B know how to reach node X. But suddenly, the link between A and X fails. Node A changes its table. If A can send its table to B immediately, everything is fine. However, the system Becomes **unstable** if B sends its routing table to A before receiving A's routing table. Node A receives the update and, assuming that B has found a way to reach X, immediately updates its routing table. Based on the **triggered update strategy**.

A sends its new update to B. Now B thinks that something has been changed around A and updates its routing table (Because **Next Node entry is same** in B's table, and B chooses new row). The cost of reaching X increases gradually **until it reaches infinity**. At this moment, both A and B know that X **cannot** be reached. However, **during this time the system is not stable**. Node A thinks that the route to X is via B; node B thinks that the route to X is via A. If A receives a **packet** destined for X, it goes to B and then comes back to A. Similarly, if B receives a packet destined for X, it goes to A and comes back to B. **Packets bounce between A and B**, creating a two-node loop problem.

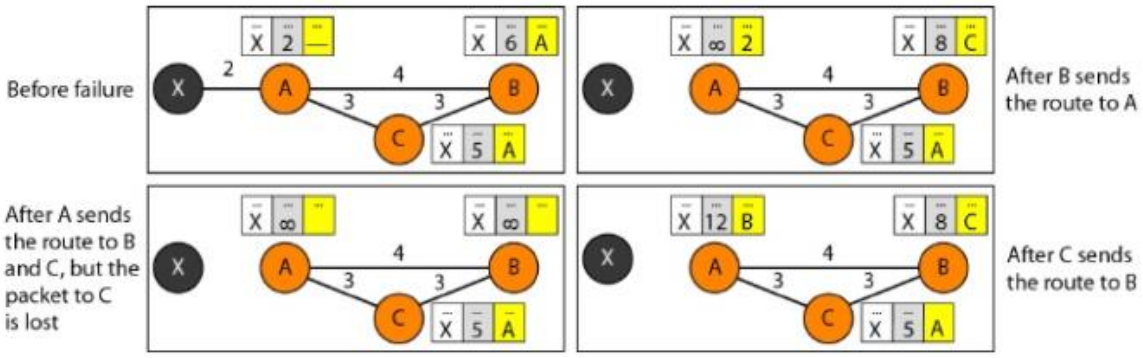
Solutions to Instability:

1. **Defining Infinity:**
   The first obvious solution is to redefine infinity to a smaller number such as 100. In most implementations of Distance vector protocol, Infinity is defined as 16, this means that the distance vector routing cannot be used in large systems. The size of the network, in each direction, **cannot exceed 15 hops**.

2. **Split Horizon**
   In this strategy, flooding the table through each interface doesn't happen. If, according to its table, node B thinks that the optimum route to reach X is via A, it does not need to advertise this piece of information to A; the information has come from A (**A already knows**). Taking information from node A, modifying it, and sending it back to node A creates the confusion. In this case, node A keeps the value of infinity as the distance to X. Later when node A sends its routing table to B, node B also corrects its routing table. The system becomes stable after the first update: both node A and B know that X is not reachable.

3. **Split Horizon and Poison Reverse**

Three-Node Instability



The two-node instability can be avoided by using the split horizon strategy combined with poison reverse. However, if the instability is between three nodes, stability cannot be guaranteed.

Suppose, after finding that X is not reachable, node A sends a packet to Band C to inform them of the situation. Node B immediately updates its table, but the packet to C is lost in the network and never reaches C. Node C remains in the dark and still thinks that there is a route to X via A with a distance of 5. After a while, node C sends to Bits routing table, which includes the route to X. Node B is totally fooled here. It receives information on the route to X from C, and according to the algorithm, it updates its table, showing the route to X via C with a cost of 8. **This information has come from C, not from A, so after a while node B may advertise this route to A**. Now A is fooled and updates its table to show that A can reach X via B with a cost of 12. Of course, the loop continues; now A advertises the route to X to C, with increased cost, but not to B. Node C then advertises the route to B with an increased cost. Node B does the same to A. And so on. The loop stops when the cost in each node reaches infinity.

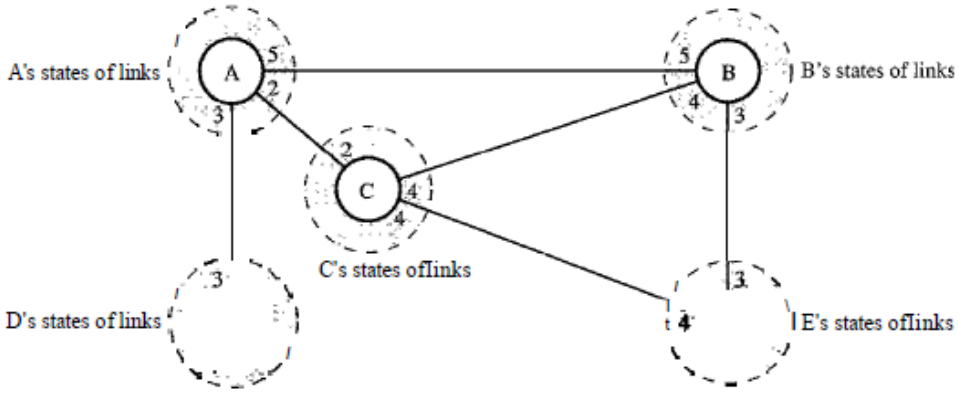
Routing Information Protocol (RIP)

It is an **intradomain routing protocol** used inside an **autonomous system**. It is a very simple protocol based on distance vector routing.

Link State Routing

In link state routing, if **each node in the domain** has the entire topology of the domain the list of nodes and links, how they are connected including the type, cost (metric), and Condition of the links (up or down)-the node can use **Dijkstra's algorithm** to build a routing table. If there are changes in any point in the network (a link is down, for example), the topology must be updated for each node. Each node has partial knowledge: it knows the state (type, condition, and cost) **of its links**.

Figure 22.21 *Link state knowledge*



Node A knows that it is connected to node B with metric 5, to node C with metric 2, and to node D with metric 3. Node C knows that it is connected to node A with metric 2, to node B with metric 4, and to node E with metric 4. Node D knows that it is connected only to node A with metric 3. And so on. Although there is an **overlap in the knowledge**, the overlap guarantees the creation of a common topology-a picture of the whole domain for each node.

Building Routing Tables

Four sets of actions are required to ensure that **each node has the routing table** showing the least-cost node to every other node.

1. Creation of the states **of the links by each node**, called the **link state packet** (LSP).
2. Spread of **LSPs to every other router**, called flooding, in an efficient and reliable way.
3. Formation of a shortest path tree for each node
4. Calculation of a routing table based on the shortest path tree

Creation of Link State Packet (LSP)

A link state packet can carry a large amount of information. For the moment, however, we assume that it carries a minimum amount of data: the node identity, the **list of links**, a **sequence number**, and **age**. The first two, node identity and the list of links, are needed to make the topology. The third, sequence number, facilitates flooding and distinguishes new LSPs from old ones. The fourth, age, prevents old LSPs from remaining in the domain for a long time. LSPs are generated on two occasions:

1. **When there is a change in the topology of the domain**. Triggering of LSP spreading is the main way of quickly informing any node in the domain to update its topology.
2. **On a periodic basis** the period in this case is much longer compared to distance vector routing. It is done to ensure that old information is removed from the domain. A longer period ensures that flooding does not create too much traffic on the network.

Flooding of LSPs After a node has prepared an LSP, it must be spread to all other nodes, not only to its neighbours.

The process is called flooding and based on the following:

1. The creating node sends a copy of the LSP **out of each interface**.
2. A node that receives an LSP compares it with the copy it may already have. If the newly arrived LSP is older than the one it has (found by checking the sequence number), it discards the LSP. **If it is newer, the node does the following**:
   a. It discards the old LSP and keeps the new one.
   b. It sends a copy of it out of each interface **except the one from which the packet arrived**. **This guarantees that flooding stops somewhere in the domain** (where a node has only one interface).

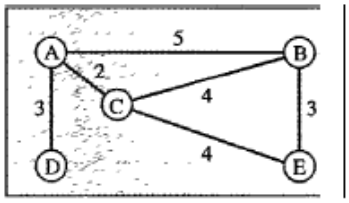
Formation of Shortest Path Tree: Dijkstra Algorithm

After receiving all LSPs, each node will have a copy of the whole topology. However, the topology is not sufficient to find the shortest path to every other node; a shortest path tree is needed.

A tree is a graph of nodes and links; one node is called the **root**. All other nodes can be reached from the root through only one single route. What we need **for each node is a shortest path tree with that node as the root**.

The Dijkstra algorithm creates a shortest path tree from a graph. The algorithm divides the nodes into two sets: **tentative** and **permanent**. It finds the neighbors of a current node, makes them tentative, examines them, and if they pass the criteria, makes them permanent.

Example of formation of shortest path tree

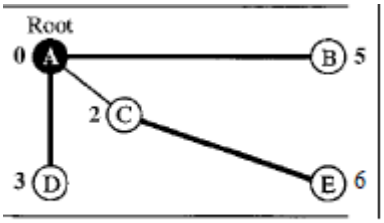


Topology

Routing table for A:

Table 22.2 *Routing table for node A*

Node	*Cost*	*Next Router*
A	0	-
B	5	-
C	2	-
D	3	-
E	6	C



6. Move E to permanent list (tentative list is empty).

Open Shortest Path First (OSPF)

It protocol is an **intradomain** routing protocol based on link state routing. Its domain is also **an autonomous system**.

Data Link Layer

The two main functions of the data link layer are data link control and media access control. The Services Provided by the Link Layer:

a. **Framing** almost all link-layer protocols encapsulate each network-layer datagram within a link-layer frame before transmission over the link. A frame consists of a data field, in which the network-layer datagram is inserted, and a number of header fields.
b. **Link access** when multiple nodes share a single broadcast link a medium access control (MAC) protocol specifies the rules by which a frame is transmitted onto the link.
c. **Reliable delivery** when a link-layer protocol provides reliable delivery service, it guarantees to move each network-layer datagram across the link without error. However, link-layer reliable delivery can be considered an unnecessary overhead for low bit-error links, including fibre, coax, and many twisted-pair copper links. For this reason, many wired link-layer protocols do not provide a reliable delivery service.
d. Error detection and correction

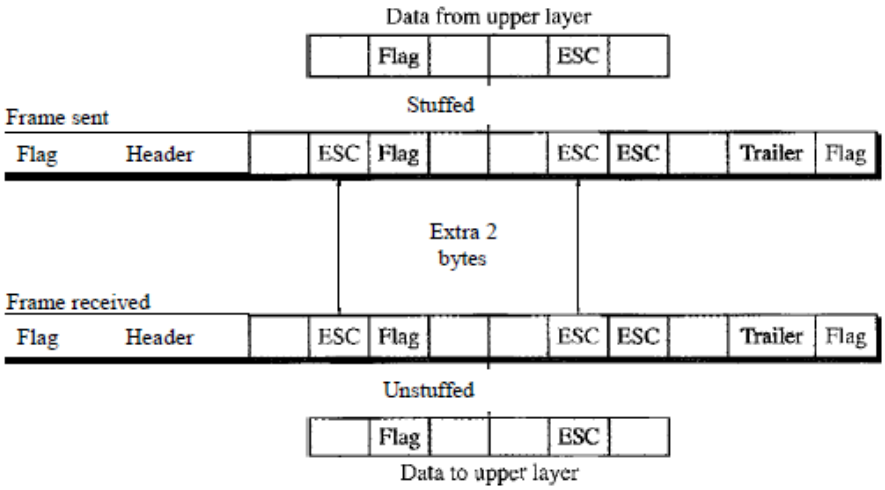
FRAMING:-

Fixed-Size Framing In fixed-size framing, there is no need for defining the boundaries of the frames; the size itself can be used as a delimiter.

Variable-Size Framing:
In variable-size framing, we need a way to define the end of the frame and the beginning of the next. Two approaches were used for this purpose: a **character-oriented** approach and a **bit-oriented approach**.

Character-Oriented Protocols (Byte Stuffing)
In a character-oriented protocol, data to be carried are 8-bit characters. To separate one frame from the next, an **8-bit flag** is added at the beginning and the end of a frame to signal the start or end of a frame.

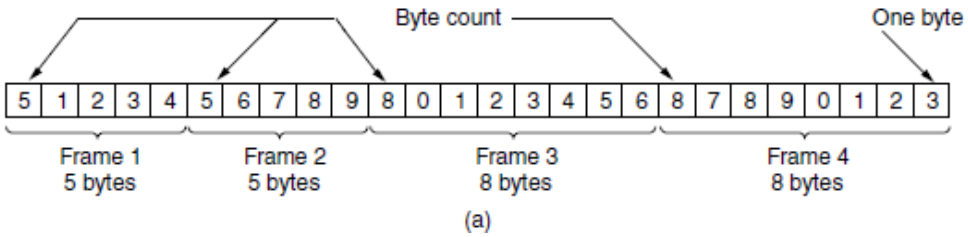
11.2 *Byte stuffing and unstuffing*



Byte stuffing is the process of adding 1 extra byte if there is a flag or escape character in the text.
The escape and flag characters that are part of the text must also be marked by another escape character.

Byte count:

The first framing method uses a field **in the header to specify the number of bytes in the frame**. Note that count itself counts itself.



(a)

Byte Count

Bit stuffing:

Each frame begins and ends with a special bit pattern, **01111110**. This pattern is a flag byte. Whenever **the sender's data link layer encounters five consecutive 1s in the data, it automatically stuffs a 0 bit into the outgoing bit stream**.

When the receiver sees five consecutive incoming 1 bits, followed by a 0 bit, it automatically destuffs (i.e., deletes) the 0 bit.

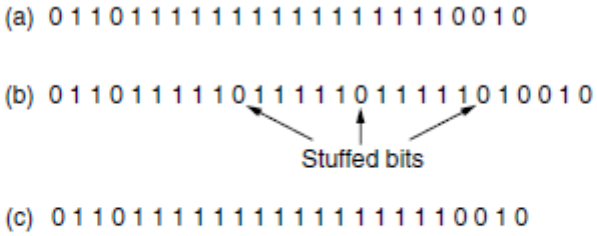


Figure 3-5. Bit stuffing. (a) The original data. (b) The data as they appear on the line. (c) The data as they are stored in the receiver's memory after destuffing.

Flow Control:-

Each receiving device has a block of memory, called a **buffer**, reserved for storing incoming data until they are processed. If the buffer begins to fill up, the receiver must be able to tell the sender to halt transmission until it is once again able to receive.

Flow control refers to a set of procedures used to restrict the amount of data That the sender can send before waiting for acknowledgment.

Error Control:-

Error control is both error detection and error correction. The data link layer is often implemented simply: Any time an error is detected in an exchange, specified frames are retransmitted. This process is called **automatic repeat request** (ARQ).

Stop-and-Wait Automatic Repeat Request

To detect and correct corrupted frames, we need to add redundancy bits to our data. When the frame arrives at the receiver site, it is checked and if it is corrupted, **it is silently discarded**. **The detection of errors in this protocol is manifested by the silence of the receiver**.

The **received** frame could be the **correct one**, or **a duplicate**, or a **frame out of order**. **The solution is to number the frames**. When the receiver receives a data frame that is out of order, this means that

frames were either lost or duplicated. The **corrupted and lost** frames need to be resent in this protocol.
Sender keeps a copy of the sent frame. At the same time, it starts a timer. If the timer expires and there is no ACK for the sent frame, the frame is resent, the copy is held, and the timer is restarted.

Error correction in Stop-and-Wait ARQ is done by keeping a copy of the sent frame And retransmitting of the frame when the timer expires.

Since an ACK frame can also be corrupted and lost, it too needs **redundancy bits and a sequence number**.

Sequence Numbers:
The protocol specifies that frames need to be numbered. This is done by using sequence numbers. A field is added to the data frame to hold the sequence number of that frame.

The sequence numbers of course can **wrap around**. For example, if we decide that the **sequence field is m bits long**, the sequence numbers start from **0, go to 2^m - 1**, and then are repeated.

Assume that the sender has sent the frame numbered x. Three things can happen:

a. The frame arrives safe and sound at the receiver site; the receiver sends an acknowledgment. The acknowledgment arrives at the sender site, causing the sender to send the next frame numbered x + 1.
b. The frame arrives safe and sound at the receiver site; the receiver sends an acknowledgment, but the **acknowledgment is corrupted or lost**. The sender resends the frame (numbered x) after the time-out. Note that the frame here is a duplicate. The receiver can recognize this fact because it expects frame x + 1 but frame x was received.
c. The frame is corrupted or never arrives at the receiver site; the sender resends the frame (numbered x) after the time-out.

We can see that there is a need for sequence numbers **x and x + 1** because the receiver needs to distinguish between case 1 and case 2. **But there is no need for a frame to be numbered x + 2**. In case 1, the frame can be numbered x again because frames x and x + 1 are acknowledged and there is no ambiguity at either site. It means sequence is 0, 1, 0, 1, 0 ,…

In Stop-and-Wait ARQ we use sequence numbers to number the frames. The sequence numbers are based on modulo 2 arithmetic.

Acknowledgment Numbers:
The acknowledgment numbers always announce the sequence Number of the next frame expected by the receiver. For example, if frame 0 has arrived safe and sound, the receiver sends an **ACK frame with acknowledgment 1** (meaning frame 1 is expected next). If frame 1 has arrived safe and sound, the receiver sends an ACK frame with acknowledgment 0 (meaning frame 0 is expected).

In Stop-and-Wait ARQ the acknowledgment number always announces in Modulo-2 arithmetic the sequence number of the next frame expected.

Design:
The sender has a control variable, which we call **Sn (sender, next frame to send)**, that holds the sequence number for the next frame to be sent (0 or 1).

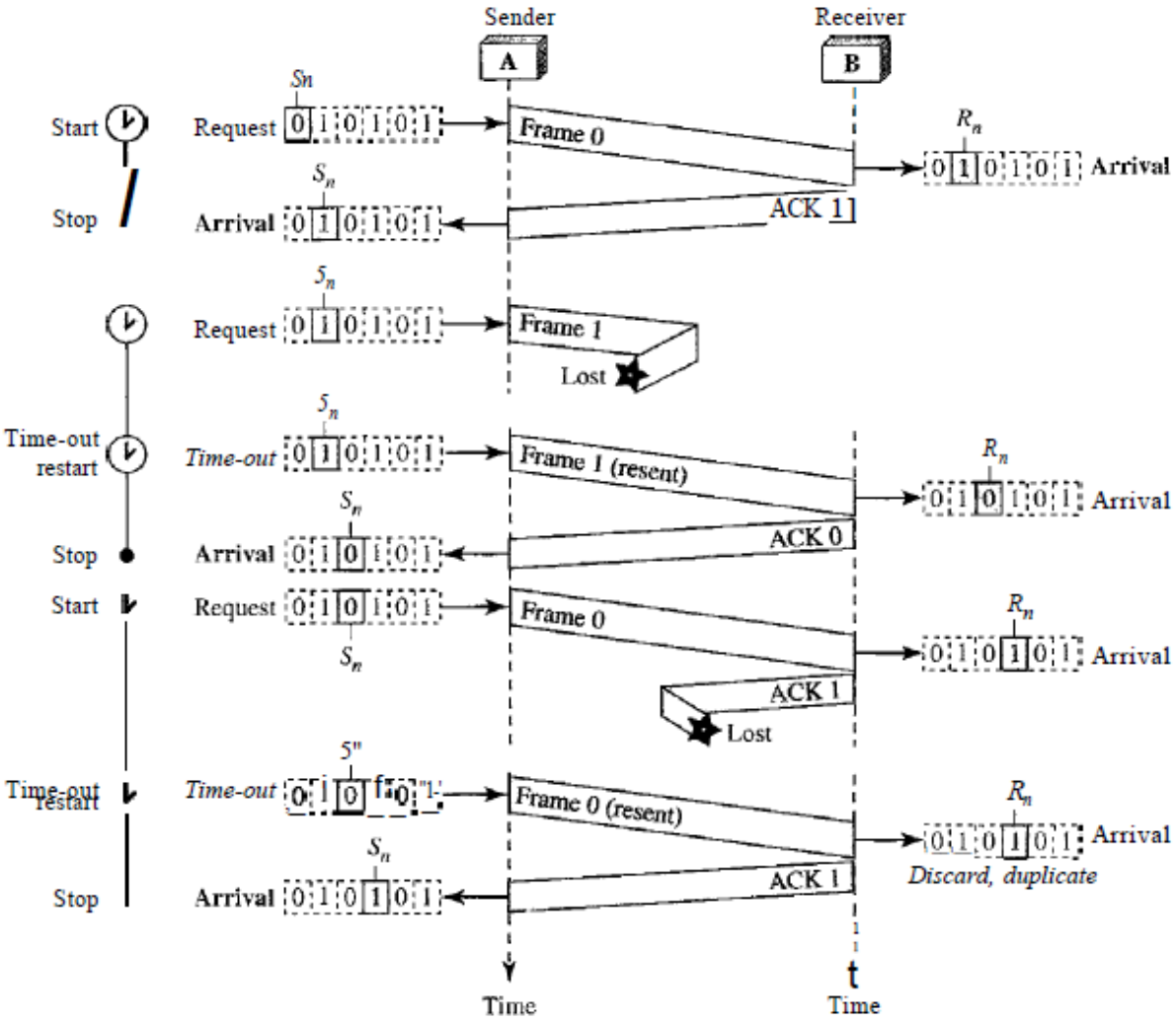
The receiver has a control variable, which we call **Rn (receiver, next frame expected)**, that holds the number of the next frame expected. Three events can happen at the sender site; one event can happen at the receiver site.

a. Variable Sn points to the slot that matches the sequence number of the frame that has been sent, but not acknowledged;
b. Rn points to the slot that matches the sequence number of the expected frame.

Efficiency:-

The **Stop-and-Wait** is very **INEFFICENT** if our channel is **thick and long**. By thick, we mean that our channel has a **large bandwidth**; by long, we mean the round-trip delay is long. The product of these two is called the **bandwidth-delay** product.
The bandwidth-delay product is a measure of the number of bits we can send out of our system while waiting for news from the receiver.

The bandwidth-delay product then is the volume of the pipe in bits.



Flow diagram Stop&Wait ARQ

Bandwidth Delay Product = BWW*Round Trip Delay

Example Assume that, in a Stop-and-Wait ARQ system, the bandwidth of the line is 1 Mbps, and 1 bit takes 20 ms to make a round trip. What is the bandwidth-delay product? If the system data frames are 1000 bits in length, what is the utilization percentage of the link?

Solution: The bandwidth-delay product is

$$(1 \times 10^6) \times (20 \times 10^{-3}) = 20{,}000 \text{ bits}$$

The system can send 20,000 bits during the time it takes for the data to go from the sender to the receiver and then back again.
However, the system sends only 1000 bits. We can say that the

Link utilization = 1000/20,000 = 5%

For a link with a high bandwidth or long delay, the use of Stop-and-Wait ARQ wastes The capacity of the link

Que: What is the utilization percentage of the link in above example if we have a protocol that can send up to 15 frames before stopping and worrying about the acknowledgments?
Sol:

Bandwidth Delay Product = 20,000 bits, 15 frames can be sent by sender.

Link utilization = 15,000/20,000 = 75%

Question:

RTT = 30ms B.W= 1Gbps (10^9 bits per second) with a packet size, L, of 1,000 bytes (8,000 bits) per packet, including both header fields and data, the time needed to actually transmit the packet into the 1 Gbps link is

$$d_{trans} = \frac{L}{R} = \frac{8000 \text{ bits/packet}}{10^9 \text{ bits/sec}} = 8 \text{ microseconds}$$

If the sender begins sending the packet at t = 0, then at t = L/R = 8 microseconds, the last bit enters the channel at the sender side.
The last bit of the packet emerging at the receiver at **t = RTT/2 + L/R = 30.008 ms**

Assuming for simplicity that ACK packets are extremely small (so that we can ignore their transmission time) and that the receiver can send an ACK as soon as the last bit of a data packet is received, the ACK emerges back at the sender at **t = RTT + L/R = 30.008 msec**. At this point, the sender can now transmit the next message. Thus, in 30.008 msec, the sender was sending for only **0.008 msec**. **Utilization** of Sender:

$$U_{\text{sender}} = \frac{L/R}{RTT + L/R} = \frac{.008}{30.008} = 0.00027$$

Go-Back-N Automatic Repeat Request

To improve the efficiency of transmission (filling the pipe), multiple frames must be in transition while waiting for acknowledgment. In a **Go-Back-N (GBN)** protocol, the sender is allowed to transmit multiple packets (when available) without waiting for an acknowledgment, but is **constrained to have no more than some maximum allowable number, N, of unacknowledged packets in the pipeline**.

Sequence Numbers:

If the header of the frame allows **m bits for the sequence number**, the sequence numbers range from 0 to (2^m – 1).
For example, if m is 4, the only sequence numbers are 0 through 15 inclusive. However, we can repeat the sequence. So the sequence numbers are:

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

In the Go-Back-N Protocol, the sequence numbers are modulo 2^m where m Is the size of the sequence number field in bits.

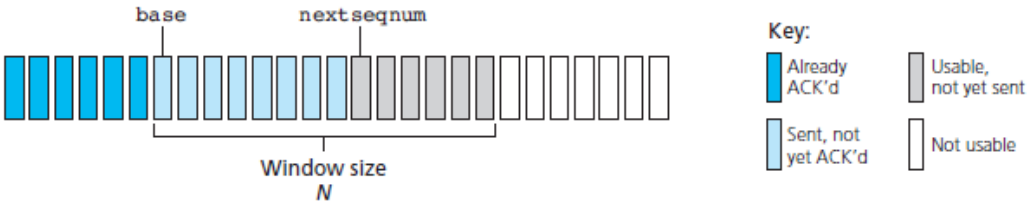
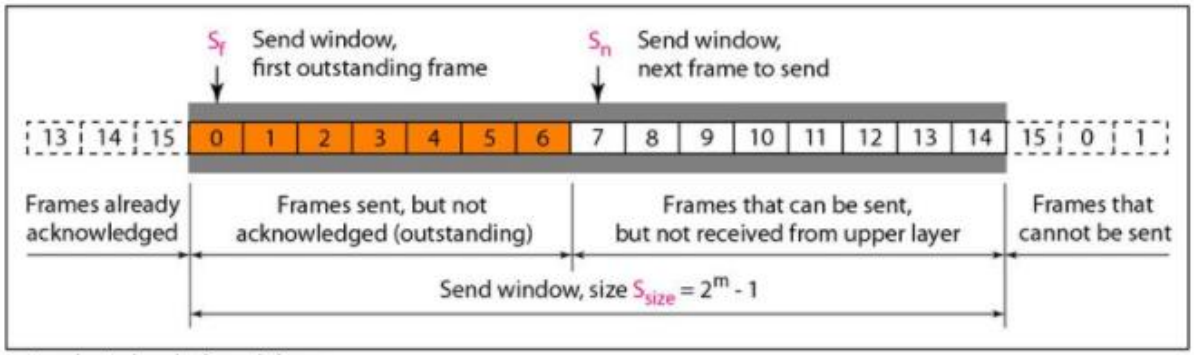
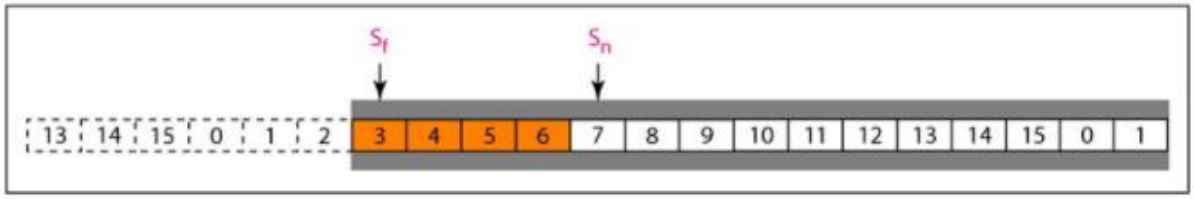


Figure 3.19 ♦ Sender's view of sequence numbers in Go-Back-N



a. Send window before sliding



b. Send window after sliding

Send window for Go-Back-N ARQ

The window at any time divides the possible sequence numbers into four regions.

a. The first region, from the far left to the left wall of the window, defines the sequence numbers belonging to frames **that are already acknowledged**. The sender does not worry about these frames and keeps no copies of them.

b. The second region, defines the range of sequence numbers belonging to the frames that are sent and have an unknown status. The sender needs to wait to find out if these frames have been received or were lost. We call these **outstanding frames**.
c. The third range, defines the range of sequence numbers for frames that can be sent. However, the corresponding data packets have not yet been received from the network layer.
d. The fourth region defines sequence numbers that cannot be used until the window slides.

The variable **Sf** defines the sequence number of the first (oldest) outstanding frame. The variable **Sn** holds the sequence number that will be assigned to the next frame to be sent. Finally, the variable **Ssize** defines the size of the window, which is fixed in our protocol.

The send window is an abstract concept defining an imaginary box of size $2^m - 1$ with three variables: S_f, S_n, and $Ssize'$

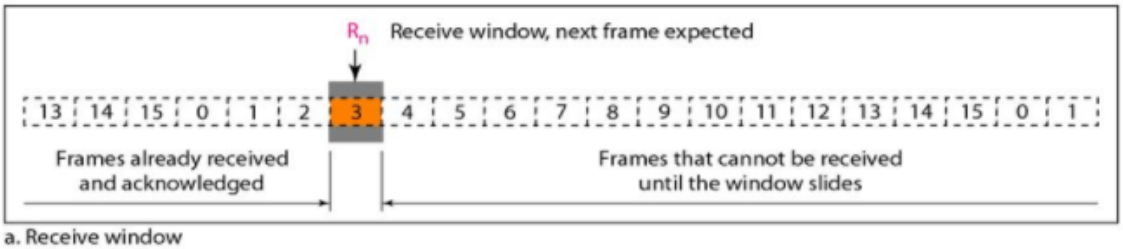
The send window can slide one or more slots when a valid acknowledgment arrives.

In above image, frames 0, 1, and 2 are acknowledged, so the window has slide to the right three slots. Note that the value of **Sf is 3** because frame 3 is now the first outstanding frame.

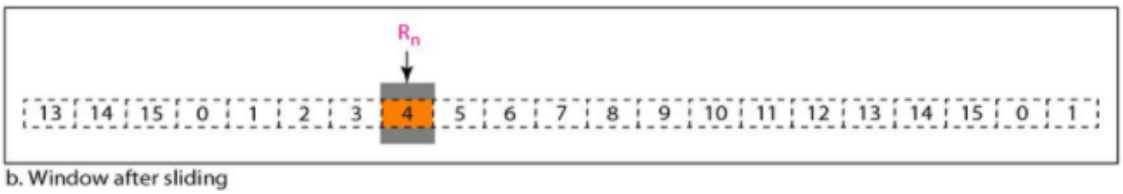
The receive window makes sure that the correct data frames are received and that the correct acknowledgments are sent.

The size of the receive window is always 1.

The receiver is always looking for the arrival of a specific frame. **Any frame arriving out of order is discarded and needs to be resent**.



a. Receive window



b. Window after sliding

The receive window is an abstract concept defining an imaginary **box of size 1 with one single variable Rn**. The window slides when a correct frame has arrived; sliding occurs one slot at a time.

Note: Only a frame with a sequence number **matching the value of Rn is accepted** and acknowledged.

Timers

Although there can be a timer for each frame that is sent, **in our protocol we use only one**. The reason is that the timer for the first outstanding frame always expires first; **we send all outstanding frames when this timer expires**.

Acknowledgment

The receiver sends a positive acknowledgment if a frame has arrived safe and sound and in order**. If a frame is damaged or is received out of order, the receiver is silent and Will discard all subsequent frames until it receives the one it is expecting**. **The silence of the receiver causes the timer of the unacknowledged frame at the sender site to expire.** This, in turn, causes the sender to go back and resend all frames, beginning with the one with the expired timer. The receiver does not have to acknowledge each frame received. It can send one cumulative acknowledgment for several frames.

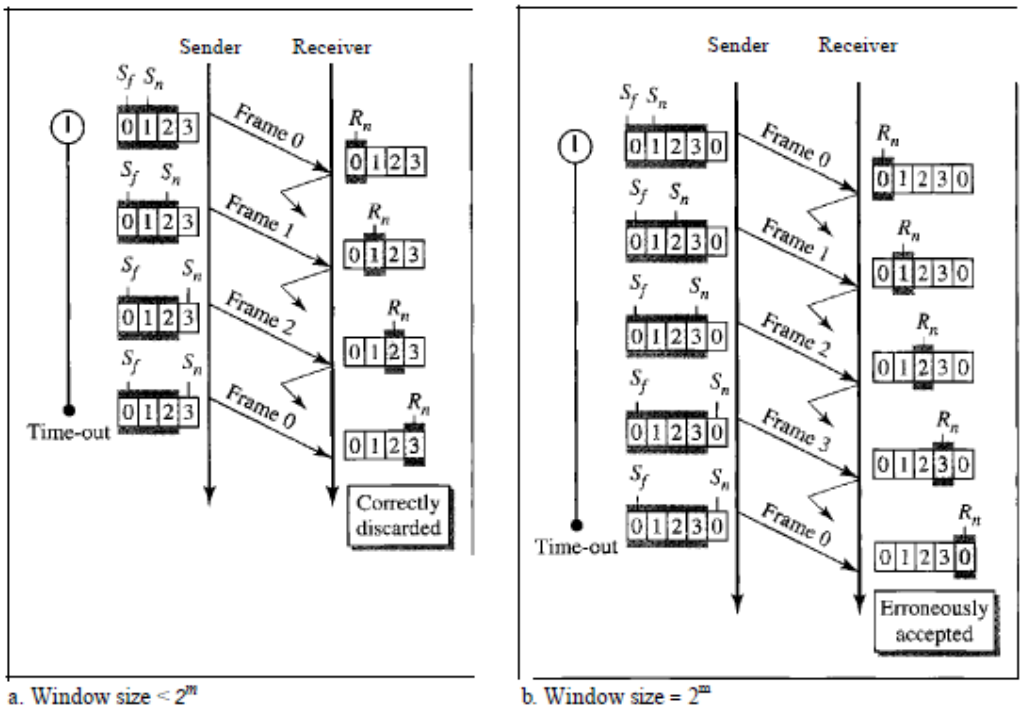
Design:

Multiple frames can be in transit in the forward direction, and multiple acknowledgments in the reverse direction.

Send Window Size

We choose m =2, which means the size of the window can be (2^m – 1), or 3. if the size of the window is 4 (**equal to 2^2**) and all acknowledgments are lost, the sender will send a duplicate of frame 0. However, this time the window of the receiver expects to receive frame 0, so it accepts frame 0, not as a duplicate, but as the first frame in the next cycle. This is an error.

Figure 11.15 *Window size for Go-Back-N ARQ*



a. Window size < 2^m b. Window size = 2^m

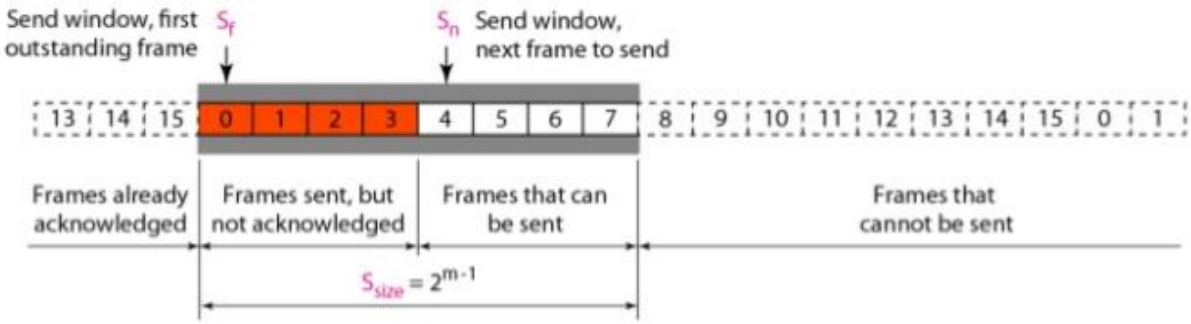
In Go-Back-N ARQ, the size of the send window must be less than 2^m the size of the receiver window is always 1.

Selective Repeat (SR)

Go-Back-N ARQ simplifies the process at the receiver site. The receiver keeps track of only one variable, and there is **no need to buffer out-of-order frames**; they are simply discarded. In S/R only the damaged frame is resent. This mechanism is called **Selective Repeat ARQ**. It is **more efficient** for noisy links, but the processing at the receiver is more complex.

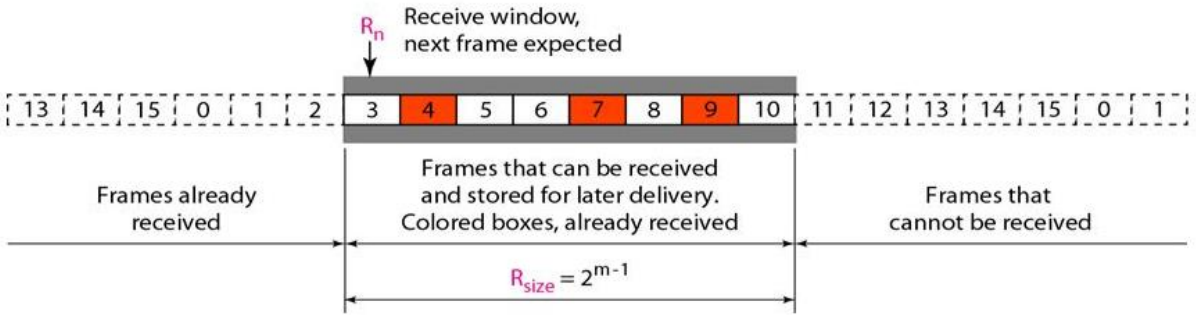
Windows

The size of the send window is much smaller; it is maximum up to 2^(m- 1). **The receive window is the same size as the send window**. For example, if m = 4, the sequence numbers go from 0 to 15, but the size of the window is just 8.



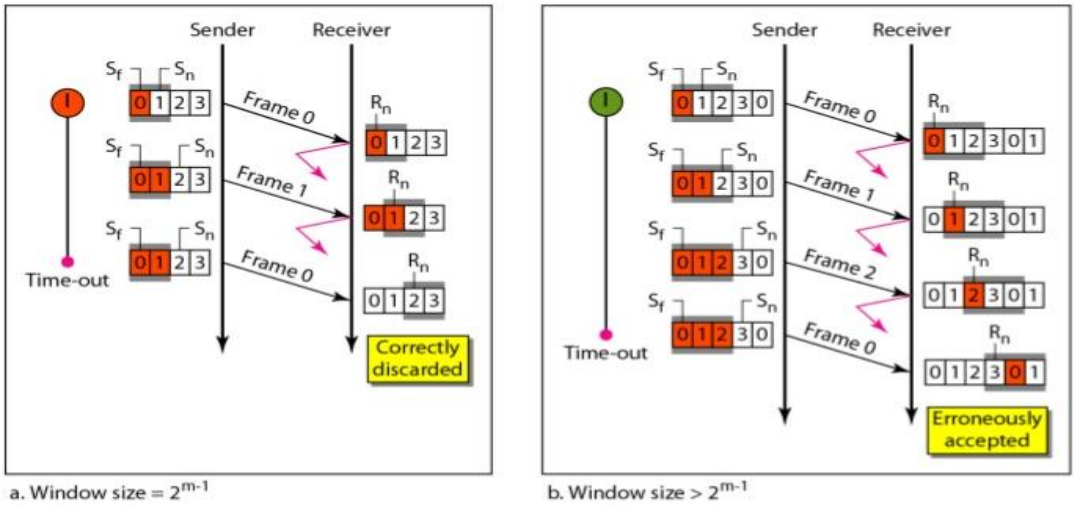
Send window for Selective Repeat ARQ

Because the sizes of the send window and receive window are the same, all the frames in the send frame can arrive out of order and be stored until they can be delivered.



Receive window for Selective Repeat ARQ

Design



In Selective Repeat ARQ, the size of the sender and receiver window must be at most one-half of 2^m.

If a valid NAK frame arrives, we just resend the corresponding frame. If a valid ACK arrives, we use a loop to purge the buffers, stop the corresponding timer and move the left wall of the window. The time-out event is simpler here; only the frame which times out is resent.

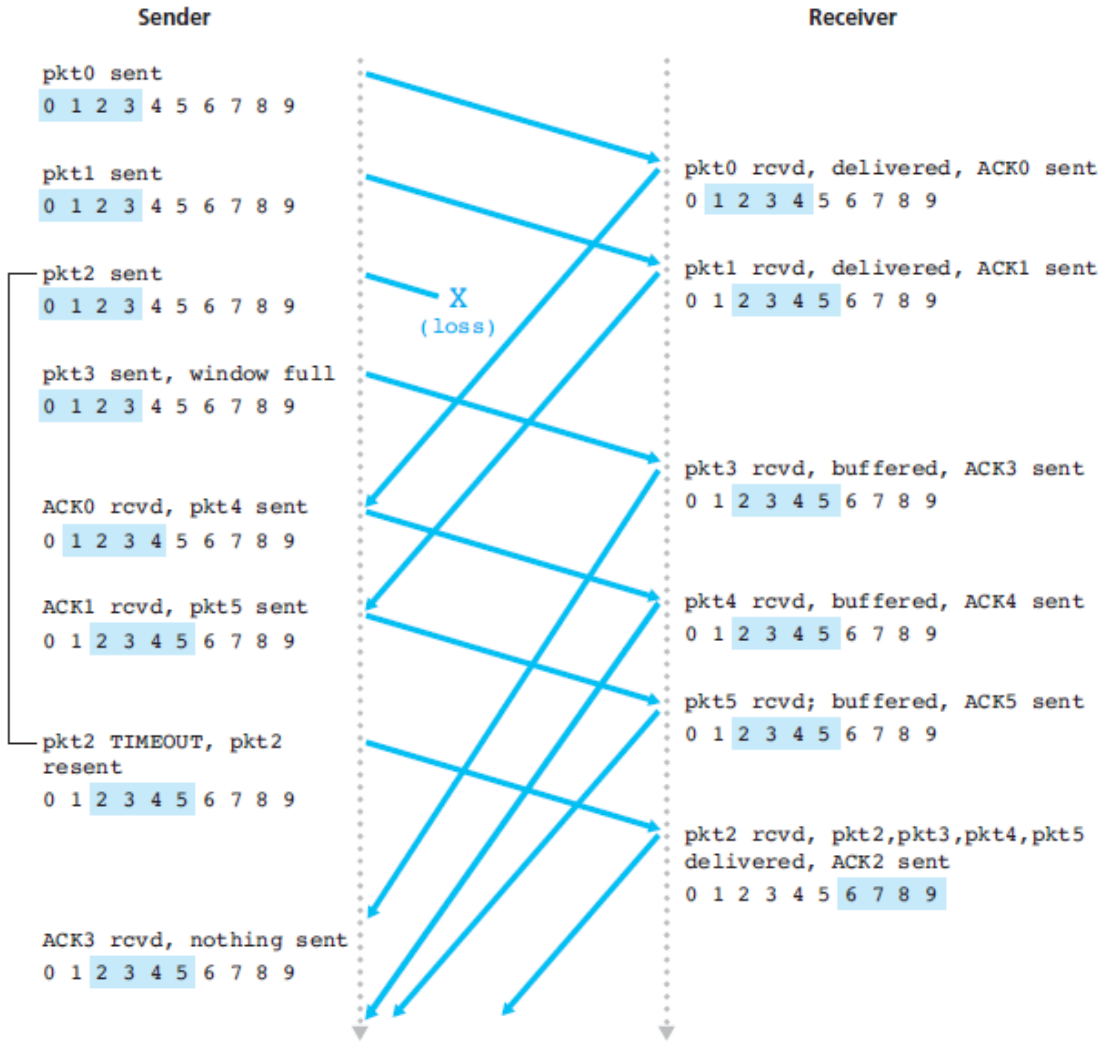


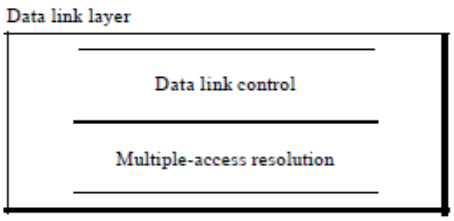
Figure 3.26 ♦ SR operation

Piggybacking

The three protocols we discussed in this section are all **unidirectional**: data frames flow in only one direction although control information such as ACK and NAK frames can travel in the other direction. A technique called **piggybacking** is used to improve the efficiency of the bidirectional protocols. When a frame is carrying data from A to B, it can also carry control information about arrived (or lost) frames from B; when a frame is carrying data from B to A, it can also carry control information about the arrived (or lost) frames from A.

Multiple Access

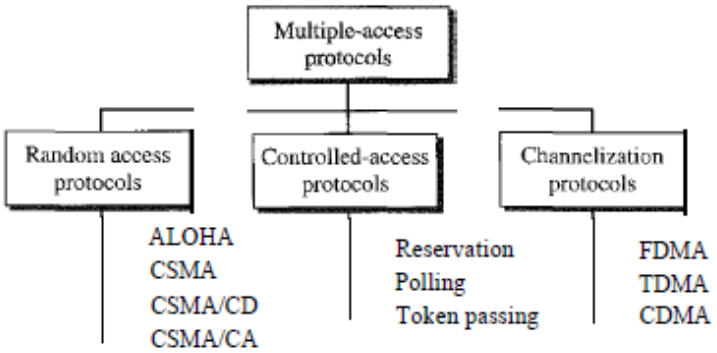
Data link layer divided into two functionality-oriented sublayers



The upper sublayer that is responsible for flow and error control is called the **logical link control** (LLC) layer; the lower sublayer that is mostly responsible for multiple access resolution is called the **media access control** (MAC) layer.

A **point-to-point** link consists of a single sender at one end of the link and a single receiver at the other end of the link. A **broadcast link**, can have multiple sending and receiving nodes all connected to the same, single, shared broadcast channel.

Taxonomy ofmultiple-access protocols discussed in this chapter



Random Access Protocols

In R.A method, no station is superior to another station and none is assigned the control over another. A transmitting node always transmits at the full rate of the channel, namely, R bps. There is no scheduled time for a station to transmit. When there is a collision, each node involved in the collision repeatedly retransmits its frame (that is, packet) until its frame gets through without a collision. But when a node experiences a collision, it doesn't necessarily retransmit the frame right away. Instead it waits a random delay before retransmitting the frame. Each node involved in a collision chooses independent random delays.

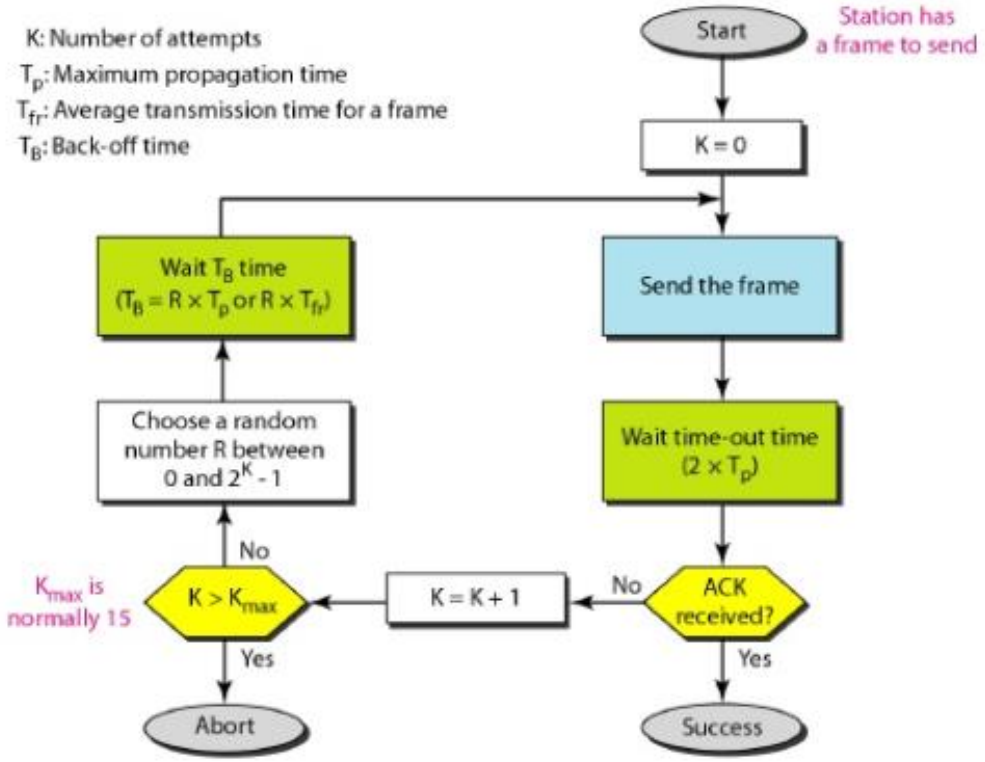
Pure Aloha

The idea is that each station sends a frame whenever it has a frame to send. We need to resend the frames that have been destroyed during transmission. The pure ALOHA protocol relies on **acknowledgments from the receiver**.

If the acknowledgment does not arrive after a time-out period, the station assumes that the frame (or the acknowledgment) has been destroyed and resends the frame.

A collision involves two or more stations. If all these stations try to resend their frames after the time-out, the frames will collide again. Pure ALOHA dictates that when the time-out period passes, each station waits a random amount of time before resending its frame.

After a maximum number of retransmission attempts **Kmax** a station must give up and try later



Procedure for Pure Aloha

Note: The time-out period is equal to the maximum possible round-trip propagation delay.

Example: The stations on a wireless ALOHA network are a maximum of 600 km apart. If we assume that signals propagate at 3 x 10^8 m/s, we find Tp = (6 x 10^5) / (3 x 10^8) = 2ms. Now we can find the value of TB for different values of K.

a. For K = 1, the range is {0, 1}. The station needs to generate a random number with a value of 0 or 1. This means that TB is either 0ms (0 x 2) or 2ms (1 x 2), based on the outcome of the random variable.
b. For K =2, the range is {0, 1, 2, 3}. This means that TB can be 0, 2, 4, or 6 ms, based on the outcome of the random variable.
c. We need to mention that if K > 10, it is normally set to 10.

Vulnerable time

Let us find the length of time, the **vulnerable time**, in which there is a possibility of collision.

in pure ALOHA, is 2 times the frame transmission time.

$$\text{Pure ALOHA vulnerable time} = 2 \times T_{fr}$$

Where Tfr is transmission time of a frame.

At any given time, the probability that a node is transmitting a **frame is p**. suppose this frame begins transmission **at time t0**. In order for this frame to be successfully transmitted, no other nodes can begin their transmission in the interval of time [t0 – 1, t0]. Such a transmission would overlap with the beginning of the transmission of node i's frame. The probability that all other nodes do not begin a transmission in this **interval is (1 – p)^(N-1)**. Similarly, no other node can begin a transmission while node i is transmitting, as such a transmission would overlap with the latter part of node i's transmission. The probability that all other nodes do not begin a transmission in this interval is also (1 – p)^(N-1). So, **the probability that a given node has a successful transmission is p(1 – p)^2(N-1)**.

Note: Maximum efficiency of the pure ALOHA protocol is only 1/(2e)—exactly half that of slotted ALOHA.

Example: A pure ALOHA network transmits 200-bit frames on a shared channel of 200 kbps. What is the requirement to make this frame collision-free?
Solution: Average frame transmission time Tfr is 200 bits/200 kbps or 1ms. The vulnerable time is 2x1 ms =2ms.
It means no station should send later than 1ms before this station starts transmission and no station should start sending during the 1ms period when this station is sending.

Throughput: (S=Gxe^(-2G))

Throughput Let us call G the average number of frames generated by the system during one frame transmission time. Then it can be proved that the average number of successful transmissions for pure ALOHA is $S = G \times e^{-2G}$. The maximum throughput S_{max} is 0.184, for $G = \frac{1}{2}$. In other words, if one-half a frame is generated during one frame transmission time (in other words, one frame during two frame transmission times), then 18.4 percent of these frames reach their destination successfully. This is an expected result because the vulnerable time is 2 times the frame transmission time. Therefore, if a station generates only one frame in this vulnerable time (and no other stations generate a frame during this time), the frame will reach its destination successfully.

The maximum throughput Smax =0.184 when G =(1/2), as S = Gxe^(-2G)

Example: A pure ALOHA network transmits 200-bit frames on a shared channel of 200 kbps. What is the throughput if the system (all stations together) produces?

a. 1000 frames per second
b. 500 frames per second
c. 250 frames per second

Solution:

Tt = 200/200 kbps = 1ms

a. If the system creates 1000 frames per second, this is 1 frame per millisecond. The load is 1. In this case S = G x e^{-2G} or S = 0.135 (13.5 percent). This means that the throughput is 1000 x 0.135 = 135 frames. Only 135 frames out of 1000 will probably survive.
b. If the system creates 500 frames per second, this is (1/2) frame per millisecond. The load is *(1/2)*. In this case S = G x e^{-2G} or S = 0.184 (18.4 percent). This means that the throughput is 500 x 0.184 = 92 and that only 92 frames out of 500 will probably survive. Note that this is the maximum throughput case, percentagewise.
c. If the system creates 250 frames per second, this is (1/4) frame per millisecond. The load is (1/4). In this case S = G x e^{-2G} or S = 0.152 (15.2 percent). This means that the throughput is 250 x 0.152 = 38. Only 38 frames out of 250 will probably survive.

Slotted ALOHA

Pure ALOHA has a vulnerable time of **2 x Tfr** . This is so because there is no rule that defines when the station can send.

SlottedALOHA vulnerable time = Tfr

Throughput: S=Gxe^(-G) Smax=0.368

Therefore, if a station generates only one frame in this vulnerable time (and no other station generates a frame during this time), the frame will reach its destination successfully.

- All frames consist of exactly L bits.
- Time is divided into slots of size L/R seconds (that is, a slot equals the time to transmit one frame).
- Nodes start to transmit frames only at the beginnings of slots.
- The nodes are synchronized so that each node knows when the slots begin.
- If two or more frames collide in a slot, then all the nodes detect the collision event before the slot ends.

- When the node has a fresh frame to send, it waits until the beginning of the next slot and transmits the entire frame in the slot.
- If there isn't a collision, the node has successfully transmitted its frame and thus need not consider retransmitting the frame. (The node can prepare a new frame for transmission, if it has one.)
- If there is a collision, the node detects the collision before the end of the slot. The node retransmits its frame in each subsequent slot with probability p until the frame is transmitted without a collision.

Slotted ALOHA allows a node to transmit continuously at the full rate, R, when that node is the only active node.

already suffered a collision.) Suppose there are N nodes. Then the probability that a given slot is a successful slot is the probability that one of the nodes transmits and that the remaining $N - 1$ nodes do not transmit. The probability that a given node transmits is p; the probability that the remaining nodes do not transmit is $(1 - p)^{N-1}$. Therefore the probability a given node has a success is $p(1 - p)^{N-1}$. Because there are N nodes, the probability that any one of the N nodes has a success is $Np(1 - p)^{N-1}$.

Example 12.4

A slotted ALOHA network transmits 200-bit frames using a shared channel with a 200-kbps bandwidth. Find the throughput if the system (all stations together) produces

a. 1000 frames per second
b. 500 frames per second
c. 250 frames per second

Solution

This situation is similar to the previous exercise except that the network is using slotted ALOHA instead of pure ALOHA. The frame transmission time is *200/200* kbps or 1 ms.

a. In this case G is 1. So S = G x e^{-G} or $S = 0.368$ (36.8 percent). This means that the throughput is 1000 x 0.0368 = 368 frames. Only 368 out of 1000 frames will probably survive. Note that this is the maximum throughput case, percentagewise.
b. Here G is $\frac{1}{2}$. In this case S = G x e^{-G} or S = 0.303 (30.3 percent). This means that the throughput is 500 x 0.0303 = 151. Only 151 frames out of 500 will probably survive.
c. Now G is $\frac{1}{4}$. In this case S = G x e^{-G} or S = 0.195 (19.5 percent). This means that the throughput is 250 x 0.195 = 49. Only 49 frames out of 250 will probably survive.

Carrier Sense Multiple Access (CSMA)

In both slotted and pure ALOHA, a node's decision to transmit is made independently of the activity of the other nodes attached to the broadcast channel.

CSMA can reduce the possibility of collision, but it cannot eliminate it.

The possibility of collision still exists because of propagation delay; when a station sends a frame, it still takes time (although very short) for the first bit to reach every station.

Vulnerable Time

The vulnerable time for CSMA is the propagation time Tp. This is the time needed for a signal to propagate from one end of the medium to the other. When a station sends a frame, and any other station tries to send a frame during this time, a collision will result. But if the first bit of the frame reaches the end of the medium, every station will already have heard the bit and will refrain from sending.

Vulnerable Time = Tp

Persistence Methods:

What should a station do if the channel is busy? What should a station do if the channel is idle?

1-Persistent after the station finds the line idle, it sends its frame immediately (with probability 1). This method has the highest chance of collision because two or more stations may find the line idle and send their frames immediately.

Nonpersistent a station that has a frame to send senses the line. If the line is idle, it sends immediately. If the line is not idle, it waits a random amount of time and then senses the line again. The nonpersistent approach reduces the chance of collision because it is unlikely that two or more stations will wait the same amount of time and retry to send simultaneously. However, this method

Reduces the efficiency of the network because the medium remains idle when there may be stations with frames to send.

p-Persistent the p-persistent method is used if the channel has time slots with a slot duration equal to or greater than the **maximum propagation time**. The p-persistent approach combines the advantages of the other two strategies. It reduces the chance of collision and improves efficiency. In this method, after the station finds the line idle it follows these steps:
1. With probability p, the station sends its frame.
2. With probability q = 1 - p, the station waits for the beginning of the next time slot and checks the line again.
    a. If the line is idle, it goes to step 1.
    b. If the line is busy, it acts as though a collision has occurred and uses the backoff procedure

Carrier Sense Multiple Access with Collision Detection (CSMA/CD)

The CSMA method does not specify the procedure following a collision. Carrier sense multiple access with collision detection (CSMA/CD) augments the algorithm to handle the collision.

Minimum Frame Size:
We need a restriction on the frame size. Before sending the last bit of the frame, the sending station must detect a collision, if any, and abort the transmission. This is so because the station, once the entire frame is sent, does not keep a copy of the frame and does not monitor the line for collision detection. Therefore, the frame transmission time **Tfr** must be at least two times the maximum propagation time **Tp**.
If the two stations involved in a collision are the maximum distance apart, the signal from the first takes time Tp to reach the second, and **the effect of the collision** takes another time Tp to reach the first. So the requirement is that the **first station must still be transmitting after 2Tp**.

Example: A network using CSMA/CD has a bandwidth of 10 Mbps. If the maximum propagation time (and ignoring the time needed to send a jamming signal) is 25.6 microsec, what is the minimum size of the frame?
Solution:
The frame transmission time is Tfr = 2 x Tp =51.2 microsec. This means, in the worst case, a station needs to transmit for a period of 51.2 ~s to detect the collision. The minimum size of the frame is 10 Mbps x 51.2 microsec =512 bits or 64 bytes. This is actually the minimum size of the frame for Standard Ethernet.

In ALOHA, we first transmit the entire frame and then wait for an acknowledgment. In CSMA/CD, transmission and collision detection is a continuous process. We do not send the entire frame and then look for a collision.

CSMA/CD Efficiency

$$\text{Efficiency} = \frac{1}{1 + 5d_{\text{prop}}/d_{\text{trans}}}$$

Time Division Multiplexing (TDM) In TDMA the stations share the bandwidth of the channel in time. Divide the time into slots and assign each slot to every channel in round robin fashion.
Each station needs to know the beginning of its slot and the location of its slot. **In TDMA, the bandwidth is just one channel that is timeshared between different stations**. At one time only one station will be accessing medium hence no collision. Slots are reserved stations though many times a station doesn't have any data to send.

Length of one slot = Tt + Tp
Efficiency = Tt/Tt + Tp

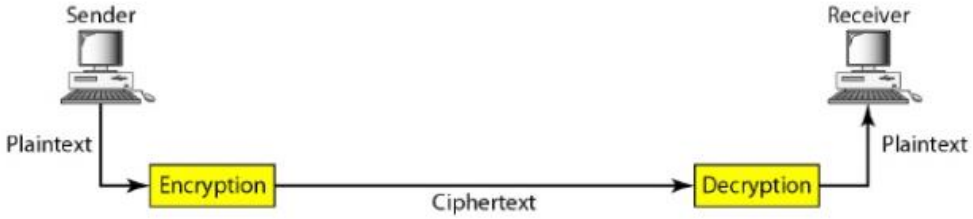
Poling:

Tp time is needed for polling to select a system. No reservations here unlike TDMA. Not fare sharing a system may participate in polling many times.

Efficiency = Tt/Tt + Tp + Tpoll

Note: If Tp is not given then ignore it.

CRYPTOGRAPHY

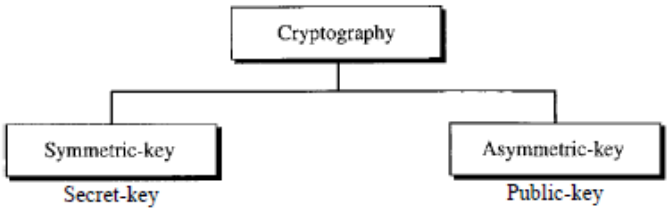


Plaintext and Ciphertext: An encryption algorithm transforms the plaintext into ciphertext; a decryption algorithm transforms the ciphertext back into plaintext. Encryption and decryption algorithms as **ciphers**.

Key:
A key is a number (or a set of numbers) that the cipher, as an algorithm, operates on. To encrypt a message, we need an encryption algorithm, an encryption key, and the plaintext. These create the ciphertext. To decrypt a message, we need a decryption algorithm, a decryption key, and the ciphertext.

We can divide all the cryptography algorithms (ciphers) into two groups: **symmetric key** (also called secret-key) cryptography algorithms and **asymmetric** (also called public-key) cryptography algorithms.

Categories of cryptography



Symmetric-Key Cryptography

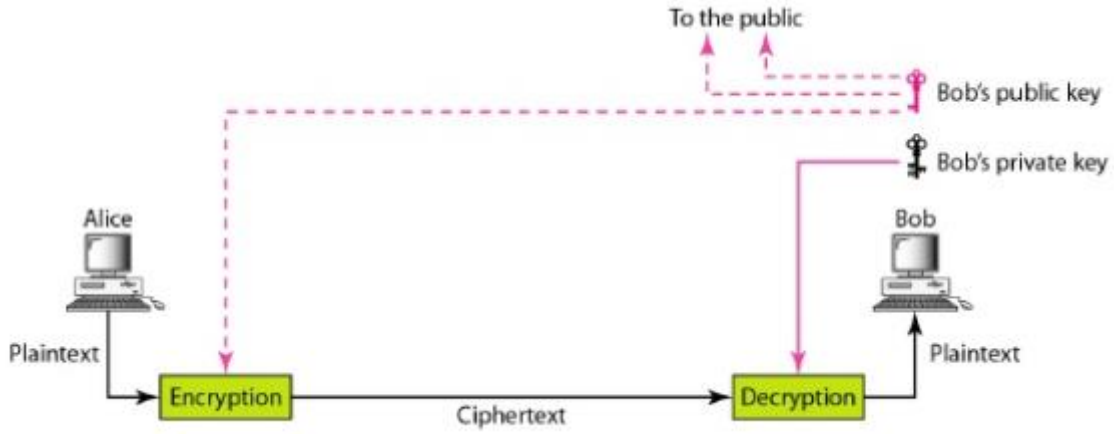
In symmetric-key cryptography, **the same key is used by both parties**. The sender uses this key and an encryption algorithm to encrypt data; the receiver uses the same key and the corresponding decryption algorithm to decrypt the data.

Symmetric-key cryptography

In symmetric-key cryptography, the same key is used by the sender (for encryption) and the receiver (for decryption). The key is shared.

Asymmetric-Key Cryptography

There are two keys: **a private key** and **a public key**. The private key is kept by the receiver. The public key is announced to the public. **Sender uses the public key to encrypt the message**. When the message is received by receiver, **the private key is used to decrypt the message**.



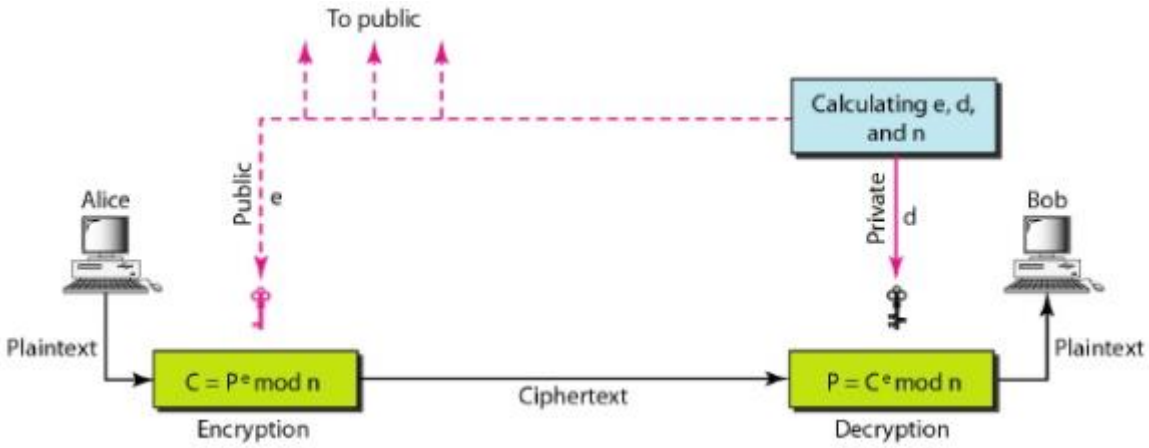
Asymmetric-key cryptography

In public-key encryption/decryption, the public key that is used for encryption is different from the private key that is used for decryption.

Note: - In symmetric-key cryptography, the same key locks and unlocks the box. In asymmetric-key cryptography, one key locks the box, but another key is needed to unlock it.

ASYMMETRIC-KEY CRYPTOGRAPHY:

RSA:- It uses two numbers, e and d, as the public and private keys.



Selecting Keys

Bob (receiver) use the following steps to select the private and public keys:

1. Bob chooses two very large **prime numbers p and q**. The larger the values, the more difficult it is to break RSA.
2. Calculate value of **n = pxq**
3. Calculate another number **ϕ(n)=(p-1)x(q-1)**
4. Choose a number **e less than n that has no common factors (other than 1) with ϕ(n)**.
5. Find a number d such that **e*d = 1 mod ϕ(n)** or **e*d mod ϕ(n) = 1**

In RSA, e and n are announced to the public; d and ϕ are kept secret.

Suppose Alice wants to send Bob a bit pattern represented by the integer **number m** (with m < n).

Ciphertext c = m^e mod n
Plaintext m = c^d mod n

Example: Alice now wants to send the letters l, o, v, and e to Bob. Interpreting each letter as a number between 1 and 26 (with a being 1, and z being 26).

Plaintext Letter	m: numeric representation	m^e	Ciphertext $c = m^e \bmod n$
l	12	248832	17
o	15	759375	15
v	22	5153632	22
e	5	3125	10

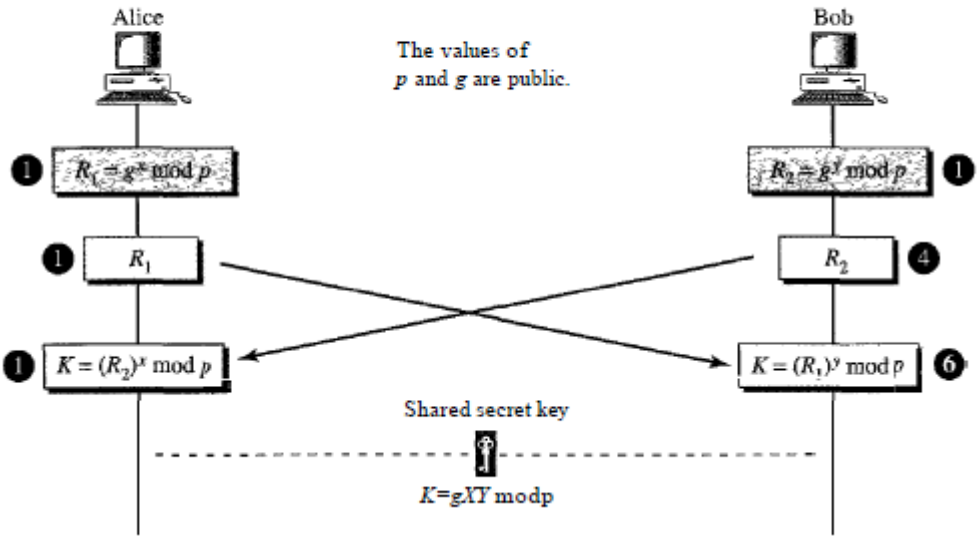
Table 8.2 ♦ Alice's RSA encryption, $e = 5$, $n = 35$

$$(m^d \bmod n)^e \bmod n = m^{de} \bmod n = m^{ed} \bmod n = (m^e \bmod n)^d \bmod n$$

Diffie-Hellman:-

RSA is a public-key cryptosystem that is often used to encrypt and decrypt symmetric keys. **Diffie-Hellman**, on the other hand, was originally designed for **key exchange**. In the Diffie-Hellman cryptosystem, two parties create a symmetric session key to exchange data without having to remember or store the key for future use.

Figure 30.26 *Diffie-Hellman method*



Before establishing a symmetric key, the two parties need to choose two numbers **p and g**. The first number, **p, is a large prime number. The second number g is a random number. These two numbers need not be confidential**

- Step 1: Alice chooses a large random number x and calculates $R1 = g^x \bmod p$.
- Step 2: Bob chooses another large random number y and calculates $R_2 = gY \bmod p$.
- Step 3: Alice sends $R1$ to Bob. Note that Alice does not send the value of x; she sends only $R1$.
- Step 4: Bob sends R_2 to Alice. Again, note that Bob does not send the value of y, he sends only R_2.
- Step 5: Alice calculates $K = (R_2)^x \bmod p$.
- Step 6: Bob also calculates $K = (R_1)^y \bmod p$.

The symmetric key for the session is K.

$$(g^x \bmod p)^y \bmod p = (gY \bmod p)X \bmod p = g^{xy} \bmod p$$

The symmetric (shared) key in the Diffie-Hellman protocol is $K = g^{xy} \bmod p$.

Example 30.10

Let us give a trivial example to make the procedure clear. Our example uses small numbers, but note that in a real situation, the numbers are very large. Assume $g = 7$ and $p = 23$. The steps are as follows:

1. Alice chooses $x = 3$ and calculates $R_1 = 7^3 \bmod 23 = 21$.
2. Bob chooses $y = 6$ and calculates $R_2 = 7^6 \bmod 23 = 4$.
3. Alice sends the number 21 to Bob.
4. Bob sends the number 4 to Alice.
5. Alice calculates the symmetric key $K = 4^3 \bmod 23 = 18$.
6. Bob calculates the symmetric key $K = 21^6 \bmod 23 = 18$.

The value of K is the same for both Alice and Bob; $g^{xy} \bmod p = 7^{18} \bmod 23 = 18$.

Man-in-the-Middle Attack
Two keys, instead of one, are created: one between Alice and Eve and one between Eve and Bob. Eve receives messages from Alice and sends it to Bob.

Authentication
The man-in-the-middle attack can be avoided if Bob and Alice first **authenticate** each other.

Network Security

a. **Message Confidentiality (Privacy)**
   The transmitted message must make sense to only the intended receiver. To all others, the message must be garbage.
b. **Message Integrity**
   Message integrity means that the data must arrive at the receiver exactly as they were sent. There must be no changes during the transmission, neither accidentally nor maliciously. As more and more monetary exchanges occur over the Internet, integrity is crucial.
c. **Message Authentication**
   In message authentication the receiver needs to be sure of the sender's identity and that an imposter has not sent the message.
d. **Message Nonrepudiation**
   A sender must not be able to deny sending a message that he or she, in fact, did send. For example, when a customer sends a message to transfer money from one account to another, the bank must have proof that the customer actually requested this transaction.
e. **Entity Authentication**
   The entity or user is verified prior to access to the system resources.

Questions:

a. A(n) **message digest** can be used to preserve the **integrity** of a document or a message.
b. A(n) **hash function** creates a message digest out of a message.
c. A **conventional** signature is included in the document; a **digital** signature is a separate entity.

d.

Digital signature cannot provide _____ for the message.

- (●) **A)** integrity
- ✓ ( ) **B)** confidentiality
- ( ) **C)** nonrepudiation
- ( ) **D)** authentication

e. If **confidentiality** is needed, a cryptosystem must be applied over the scheme.
f. The secret key between members needs to be created as a **session** key when two members contact KDC.

Digital Signature:

Although a MAC (message authentication code) use a keyed hash function which can provide message integrity and message authentication, it has a drawback. It needs a symmetric key that must be established between the sender and the receiver.
A digital signature can use a pair of asymmetric keys (a public one and a private one) which proves authenticity of sender.

Inclusion
A **conventional signature** is included in the document; it is part of the document. When we sign a document digitally, **we send the signature as a separate document**. The sender sends two documents: **the message and the signature**. The recipient receives both documents and verifies that the signature belongs to the supposed sender. If this is proved, the message is kept; otherwise, it is rejected.

Verification Method
In conventional signature, when the recipient receives a document, she compares the signature on the document with the signature on file. If they are the same, the document is authentic. The recipient needs to have a copy of this signature on file for comparison. In digital signature, the recipient receives the message and the signature. A copy of the signature is not stored anywhere. The recipient needs to apply a verification technique to the combination of the message and the signature to verify the authenticity.

Relationship
In conventional signature, there is normally a one-to-many relationship between a signature and documents. A person, for example, has a signature that is used to sign many checks, many documents, etc. In digital signature, there is a one-to-one relationship between a signature and a message. Each message has its own signature. The signature of one message cannot be used in another message.

Need for Keys
In conventional signature a signature is like a private "key" belonging to the signer of the document. The signer uses it to sign a document; no one else has this signature. The copy of the signature is on file like a public key; anyone can use it to verify a document, to compare it to the original signature.

"In digital signature, the signer uses her private key, applied to a signing algorithm, to sign the document. The verifier, on the other hand, uses the public key of the signer, applied to the verifying algorithm, to verify the document."

A digital signature needs a public-key system.

Process

Digital signature can be achieved in two ways: signing the document or signing a digest of the document.

In a cryptosystem, we use the private and public keys of the receiver; in digital signature, we use the private and public key of the sender.

Services

A digital signature can provide three out of the five services we mentioned for a security system: **message integrity**, **message authentication**, and **nonrepudiation**. Note that a digital signature scheme does not provide confidential communication. If confidentiality is required, the message and the signature must be encrypted using either a secret-key or public-key **cryptosystem**.

Message Integrity

The integrity of the message is preserved even if we sign the whole message because we cannot get **the same signature if the message is changed**.

Certification Authority:

Bob wants two things:

a. he wants people to know his public key, and
b. He wants no one to accept a public key forged as his.

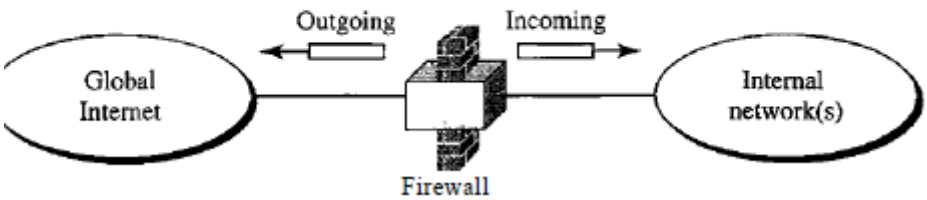
Bob can go to a certification authority that binds a public key to an entity and issues a certificate. The CA has a well-known public key itself that cannot be forged. It then asks for Bob's public key and writes it on the certificate. To prevent the certificate itself from being forged, the CA signs the certificate with its private key. Now Bob can upload the signed certificate. Anyone who wants Bob's public key downloads the signed certificate and uses the public key of the center to extract Bob's public key.

**********FIREWALLS**********

FIREWALLS

A firewall is a (combination of hardware and software) device (usually a router or a computer) installed between the internal network of an organization and the rest of the Internet. It is designed to forward some packets and filter (not forward) others.

Firewall



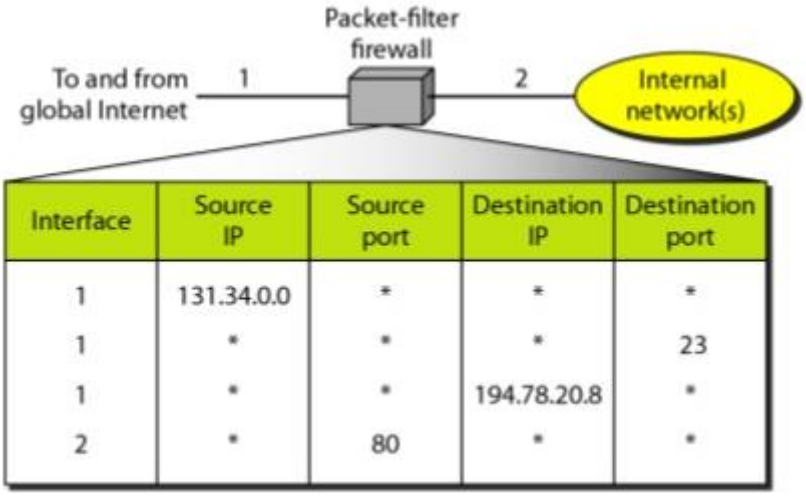
A firewall has three goals:

a. All traffic from outside to inside, and vice versa, passes through the firewall
b. Only authorized traffic, as defined by the local security policy, will be allowed to pass.
c. The firewall itself is immune to penetration

A firewall is usually classified as a **packet-filter firewall** or a **proxy-based firewall (Application Gateways)**

Packet Filter Firewall:

It can forward or block packets based on the information in the **network layer** and **transport layer** headers: source and destination IP addresses, source and destination port addresses, and type of protocol (TCP or UDP). A packet-filter firewall is a router that uses a filtering table to decide which packets must be discarded (not forwarded).



Interface	Source IP	Source port	Destination IP	Destination port
1	131.34.0.0	*	*	*
1	*	*	*	23
1	*	*	194.78.20.8	*
2	*	80	*	*

Packet-filter firewall

The following packets are filtered:

1. Incoming packets from network 131.34.0.0 are blocked (security precaution). Note that the * (asterisk) means "any."
2. Incoming packets destined for any internal TELNET server (port 23) are blocked
3. Incoming packets destined for internal host 194.78.20.8 are blocked. The organization wants this host for internal use only.
4. Outgoing packets destined for an HTTP server (port 80) are blocked. The organization does not want employees to browse the Internet.

A **packet-filter** firewall filters at the network or transport layer.

Policy	Firewall Setting
No outside Web access.	Drop all outgoing packets to any IP address, port 80
No incoming TCP connections, except those for organization's public Web server only.	Drop all incoming TCP SYN packets to any IP except 130.207.244.203, port 80
Prevent Web-radios from eating up the available bandwidth.	Drop all incoming UDP packets—except DNS packets.
Prevent your network from being used for a smurf DoS attack.	Drop all ICMP ping packets going to a "broadcast" address (eg 130.207.255.255).
Prevent your network from being tracerouted	Drop all outgoing ICMP TTL expired traffic

Proxy Firewall (Application Gateway)

Packet-level filtering allows an organization to perform filtering based on the information available in the **network layer and transport layer** headers and **on the basis of the contents** of IP and TCP/UDP headers, including IP addresses, port numbers, and acknowledgment bits.

However, sometimes we need to filter a message based on the information available in the message itself (at the application layer).

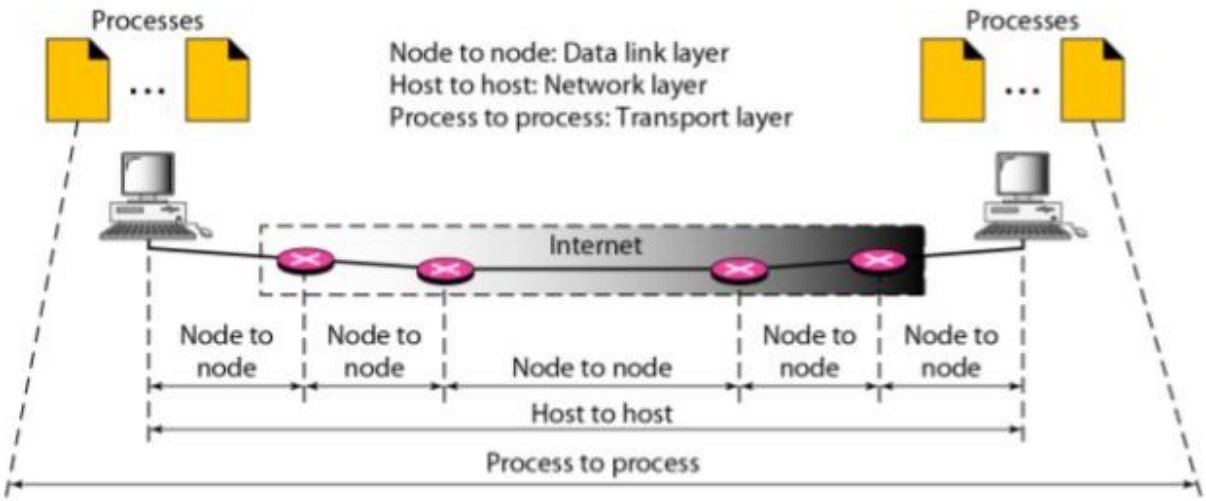
A proxy firewall filters at the application layer.

TRANSPORT LAYER

PROCESS-TO-PROCESS DELIVERY

The transport layer is responsible for process-to-process delivery-the delivery of a packet, part of a message, from one process to another.

The transport layer is responsible for process-to-process delivery.



Types of Data Deliveries

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.

The port numbers are 16-bit integers between 0 and 65,535.

Two Important things:

1. The client program defines itself with a port number, chosen **randomly** by the transport layer software running on the client host.
2. The server process must also define itself with a port number. This port number, however, **cannot be chosen randomly**.

The Internet has decided to use **universal port numbers** for servers; these are called **well-known** port numbers.

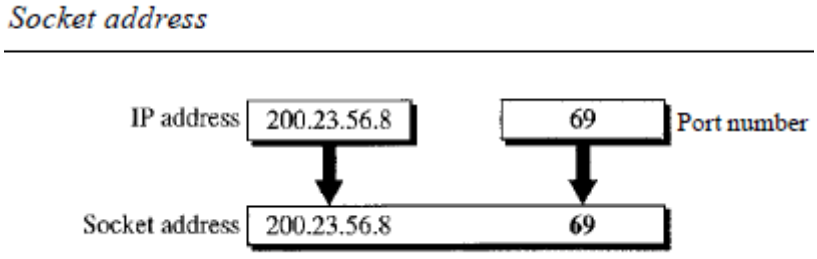
IANA Ranges

1. **Well-known ports** the ports ranging from **0 to 1023** are **assigned** and **controlled** by IANA.
2. **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.
3. **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 (Temporal)** ports.

Socket Addresses

Process-to-process delivery needs two identifiers, **IP address** and the **port number** each end to make a connection. The combination of an IP address and a port number is called a **socket address**.

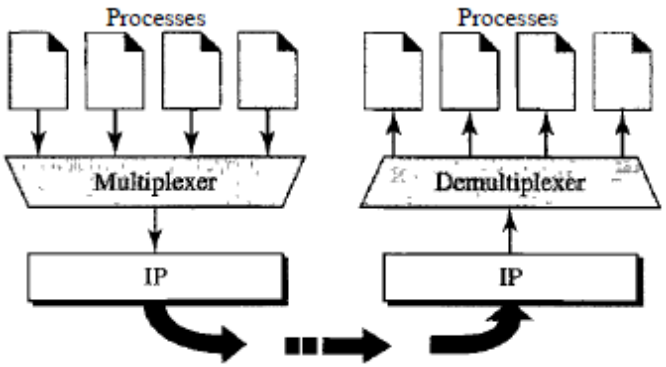
A transport layer protocol needs a pair of socket addresses: the client socket address and the server socket address. These four pieces of information are part of the IP header and the transport layer protocol header. **The IP header contains the IP addresses; the UDP or TCP header contains the port numbers.**



Multiplexing and Demultiplexing

The addressing mechanism allows **multiplexing** and **demultiplexing** by the transport layer.

Multiplexing 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 a **many-to-one** relationship and requires multiplexing. After adding the header, the transport layer passes the packet to the network layer.

Demultiplexing

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

Connectionless Versus Connection-Oriented Service

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

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**. **UDP**, is connectionless.
b. **Connection Oriented Service** connection is first established between the sender and the receiver. Data are transferred. At the end, the connection is released. TCP and STCP are connection oriented.

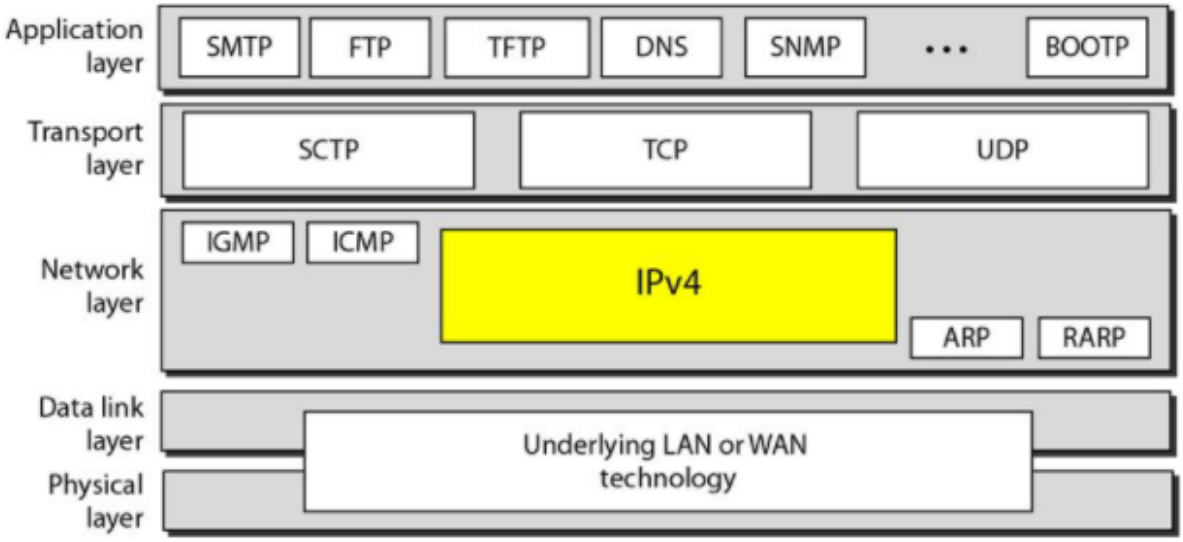
Reliable Versus Unreliable

The transport layer service can be **reliable** or **unreliable**. If the application layer program needs reliability, we use a reliable transport layer protocol by implementing **flow and error control** at the transport layer. This means a **slower and more complex service**.

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.

UDP is connectionless and unreliable; TCP and SCTP are connection-oriented and reliable.

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. **Because the network layer in the Internet is unreliable (best-effort delivery), we need to implement reliability at the transport layer**.



Position of UDP TCP and SCTP in TCP/IP Suite

USER DATAGRAM PROTOCOL (UDP)

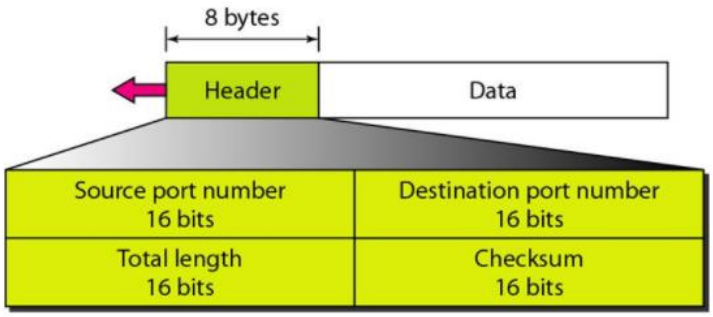
The User Datagram Protocol (UDP) is called a **connectionless**, **unreliable** transport protocol. It does not add anything to the services of IP **except to provide process-to-process communication instead of host-to-host communication**. Also, it performs very limited error checking. If a process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

Note:

a. Some well-known port numbers used by UDP
b. Some port numbers can be used by both UDP and TCP
c. FTP can use port 21 with either UDP or TCP

User Datagram Format:-

UDP packets, called **user datagrams**, have a fixed-size header of 8 bytes. Below figure shows the format of a user datagram:



User Datagram Format

1. **Source port number** Source port can used by client/server both. It is 16 bits long (0 to 65535)

2. **Destination port number** this is the port number used by the process running on the destination host. It is also 16 bits long. If the destination host is the client (a server sending a response), In this case, **the server copies the ephemeral port number it has received in the request packet**.

3. **Length** This is a **16-bit field that defines the total length of the user datagram, header plus data**. The 16 bits can define a total length of 0 to 65,535 bytes. However, the total length needs to be much less because a UDP user datagram is stored in an IP datagram with a total length of 65,535 bytes. The length field in a UDP user datagram is actually not necessary. Because

$$\text{UDP length} = \text{IP length} - \text{IP header's length}$$

4. **Checksum** this field is used to detect errors over the entire user datagram (**header plus data**).

Note: The calculation of the checksum and its inclusion in a user datagram are optional

UDP Operation

Connectionless Services

There is no relationship between the different user datagrams even if they are coming from the same source process and going to the same destination program. The user datagrams are not numbered. Also, there is no connection establishment and no connection termination. The process that uses **UDP cannot send a stream of data** to UDP and expect UDP to chop them into different related user datagrams. Each request must be small enough to fit into one user datagram.

Flow and Error Control

UDP is a very simple, unreliable transport protocol. There is no flow control and hence no window mechanism. The receiver may overflow with incoming messages. There is no error control mechanism in UDP except for the checksum. When the receiver detects an error through the checksum, **the user datagram is silently discarded**.

Encapsulation and Decapsulation

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

Use of UDP

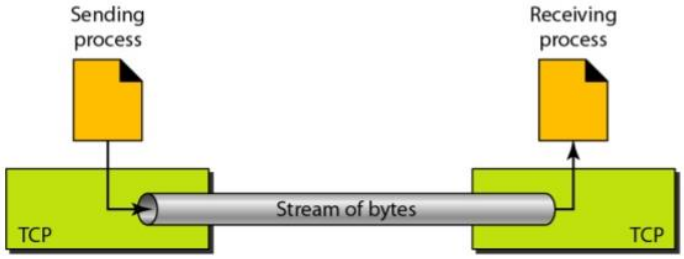
1. UDP is suitable for a process that requires simple request-response communication.
2. UDP is suitable for a process with internal flow and error control mechanisms.
3. **UDP is a suitable transport protocol for multicasting**. Multicasting capability is embedded in the UDP software but not in the TCP software
4. **UDP is used for some route updating protocols such as Routing Information Protocol** (RIP)

Transmission Control Protocol (TCP)

TCP offers several additional services to applications. It provides reliable data transfer. Using flow control, sequence numbers, acknowledgments, and timers TCP ensures that data is delivered from sending process to receiving process, correctly and in order. TCP thus converts IP's unreliable service between end systems into a reliable data transport service between processes. TCP also provides Congestion control.
TCP is called a connection-oriented, it creates a virtual connection between two TCPs to send data.

TCP Services:

Process-to-Process Communication
Like UDP, TCP provides process-to-process communication using port numbers. TCP, unlike UDP, is a stream-oriented protocol.
It allows the sending process to deliver data as a stream of bytes and allows the receiving process to obtain data as a stream of bytes. TCP creates an environment in which the two processes seem to be connected by an imaginary "tube" that carries their data across the Internet.



Stream Delivery

The sending process produces (writes to) the stream of bytes, and the receiving process consumes (reads from) them.

Sending and Receiving Buffers
Because the sending and the receiving processes may not write or read data at the same speed, TCP needs buffers for storage. There are two buffers, the **sending buffer** and the **receiving buffer**, one for each direction.
One way to implement a buffer is to use a circular array of 1-byte locations.

Segments
The IP layer, as a service provider for TCP, needs to send data in **packets**, not as a stream of bytes. At the transport layer, TCP groups a number of bytes together into a packet called a **segment**. TCP adds a header to each segment and delivers the segment to the IP layer for transmission. The segments are encapsulated in IP datagrams and transmitted. This entire operation is transparent to the receiving process.

Note: - that the segments are not necessarily the same size.

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

Connection-Oriented Service:
TCP, unlike UDP, is a connection-oriented protocol. When a process at site A wants to send and receive data from another process at site B, the following occurs:
1. The two TCPs establish a connection between them.
2. Data are exchanged in both directions.
3. The connection is terminated.

TCP Features:-

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

Byte Number
TCP numbers all data bytes that are transmitted in a connection. **The numbering does not necessarily start from 0**. Instead, **TCP generates a random number between 0 and 2^32 - 1** for the number of the first byte. For example, if the random number happens to be 1057 and the total data to be sent are 6000 bytes, the bytes are numbered from 1057 to 7056.

> The bytes of data being transferred in each connection are numbered by TCP. The numbering starts with a randomly generated number.

Sequence Number
After the bytes have been numbered, TCP assigns a sequence number to each segment that is being sent. S**equence number for each segment is the number of the first byte carried in that segment**.

Example:
Suppose a TCP connection is transferring a file of 5000 bytes. The first byte is numbered 10,001 what are the sequence numbers for each segment if data are sent in five segments, each carrying 1000 bytes?
Sol: 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)

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

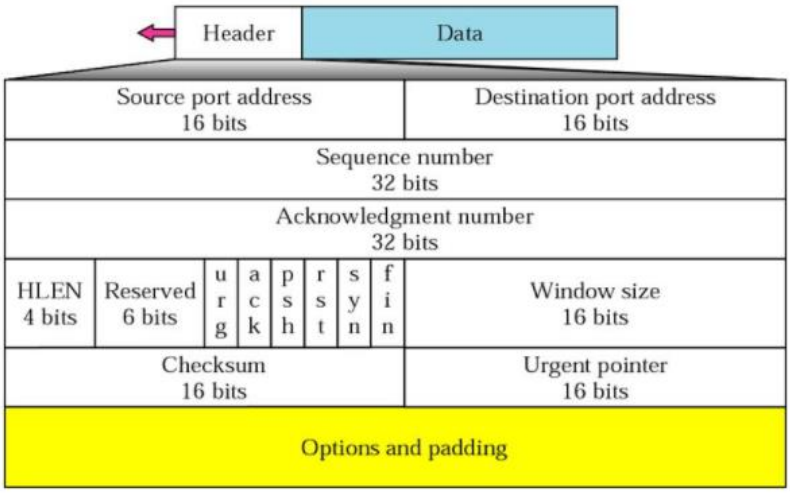
Note:-
When a segment carries a combination of data and control information (piggybacking), it uses a sequence number. **If a segment does not carry user data, it does not logically define a sequence number. The field is there, but the value is not valid**. However**, some segments**, when carrying only control information, need a sequence number to allow an acknowledgment from the receiver. These segments are used for connection establishment, termination, or abortion. **Each of these segments consumes one sequence number as though it carried 1 byte, but there are no actual data. If the randomly generated sequence number is x, the first data byte is numbered x + 1.** The byte x is considered a **phony byte** that is used for a control segment to open a connection.

Acknowledgment Number

Communication in TCP is full duplex; when a connection is established, both parties can send and receive data at the same time. Each party numbers the bytes, usually with a different starting byte number. The sequence number in each direction shows the number of the first byte carried by the segment. Each party also uses an acknowledgment number to confirm the bytes it has received. However, the acknowledgment number defines the number of the next byte that the party expects to receive.

The term cumulative here means that if a party uses 5643 as an acknowledgment number, it has received all bytes from the beginning up to 5642. Note that this does not mean that the party has received 5642 bytes because the first byte number does not have to start from 0.

TCP segment format



The segment consists of a **20- to 60-byte header**, followed by data from the application program. The header is 20 bytes if there are no options and up to 60 bytes if it contains options.

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.

Acknowledgment number this 32-bit field defines the byte number that the receiver of the segment is expecting to receive from the other party. If the receiver of the segment has successfully received byte number x from the other party, it defines x + 1 as the acknowledgment number.

Header length the length of the header can be between 20 and 60 bytes. Therefore, the value of this field can be between 5 (5 x 4 =20) and 15 (15 x 4 =60). **4 is scaling factor**.

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

Control this field defines 6 different control bits or flags, one or more of these bits can be set at a time

These bits enable flow control, connection establishment and termination, connection abortion, and TCP.

Table 23.3 *Description offlags in the controlfield*

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

Window size
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 inclusion of the checksum in the UDP datagram is optional, whereas the **inclusion of the checksum for TCP is mandatory**.

TCP Connection:

A TCP connection is also always point-to-point, that is, b/w a single sender and a single receiver.

TCP is connection-oriented. A connection-oriented transport protocol establishes a virtual path between the source and destination.
TCP operates at a higher level. TCP uses the services of IP to deliver individual segments to the receiver, but it controls the connection itself. If a segment is lost or corrupted, it is retransmitted. Unlike TCP, **IP is unaware of this retransmission**.

Connection Establishment

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

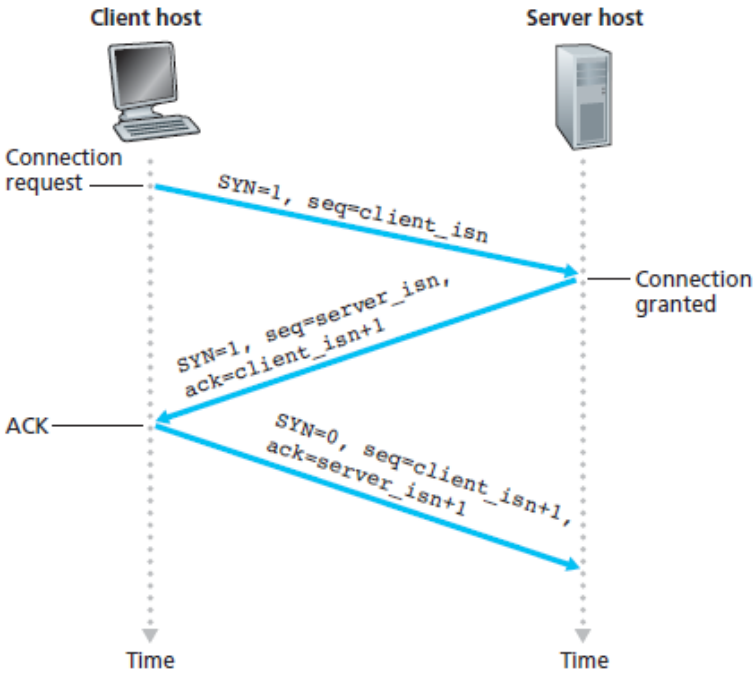


Figure 3.39 ♦ TCP three-way handshake: segment exchange

a. The client sends the first segment, a **SYN segment**, in which only the SYN flag is set. This segment is for synchronization of sequence numbers. **It consumes one sequence number**.

> A SYN segment cannot carry data, but it consumes one sequence number.

b. The server sends the second segment, a SYN +ACK segment, with 2 flag bits set: SYN and ACK. **It consumes one sequence number.**

> A SYN + ACK segment cannot carry data, but does consume one sequence number.

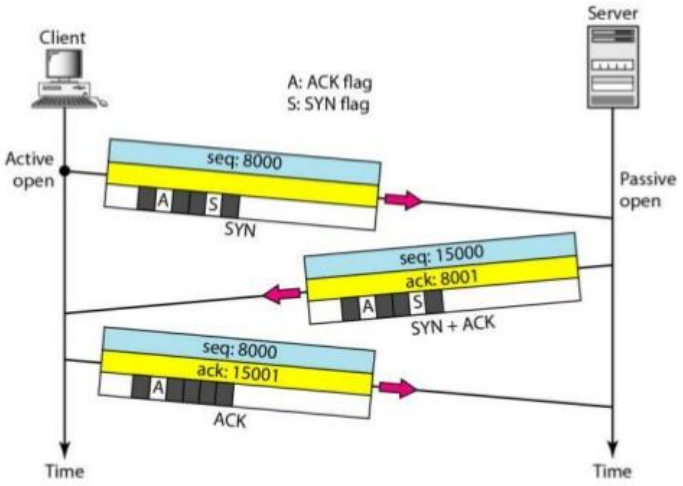
c. The client sends the third segment. This is just an ACK segment. Sequence number for this segment is client_isn + 1. But this same sequence client_isn+1 will be used for data transfer. Hence we can say that **this ack segment doesn't consume any sequence number**.

SYN Flooding Attack

A malicious attacker sends a large number of SYN segments to a server, pretending that each of them is coming from a different client by faking the source IP addresses in the datagrams. The server, assuming that the clients are issuing an active open, allocates the necessary resources. This **SYN flooding attack** belongs to a type of security attack known as **a denial-of-service attack**, in which an attacker monopolizes a system with so many service requests that **the system collapses and denies service** to every request.

2. Data Transfer

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



3-way handshaking connection establishment

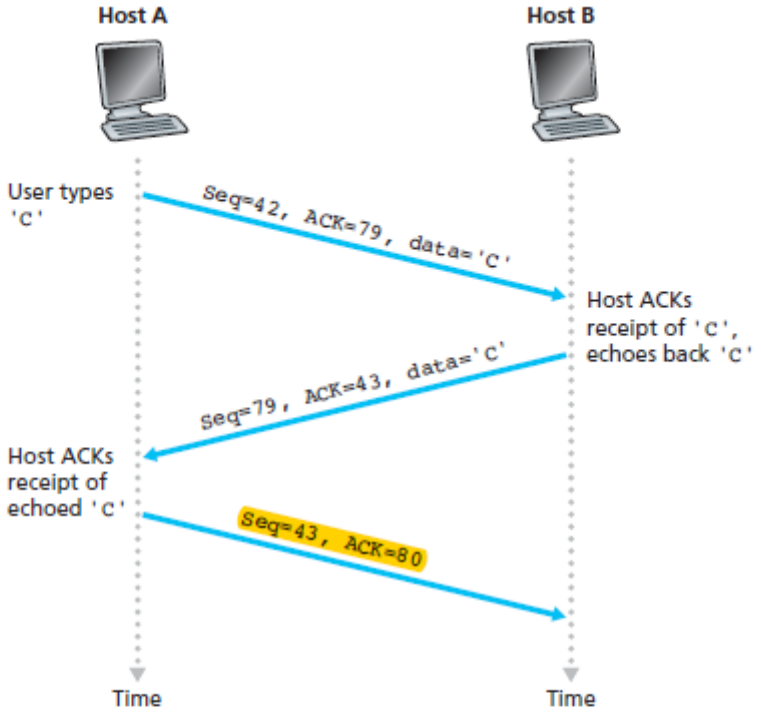


Figure 3.31 ♦ Sequence and acknowledgment numbers for a simple Telnet application over TCP

The third segment is sent from the client to the server. Its sole purpose is to acknowledge the data it has received from the server. (Recall that the second segment contained data—the letter 'C'—from the server to the client.) This segment has an empty data field (that is, the acknowledgment is not being piggybacked with any client-to-server data). The segment has 80 in the acknowledgment number field because the client has received the stream of bytes up through byte sequence number 79 and it is now waiting for bytes 80 onward. You might think it odd that this segment also has a sequence number since the segment contains no data. But because TCP has a sequence number field, the segment needs to have some sequence number.

Data Transfer

Pushing Data

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

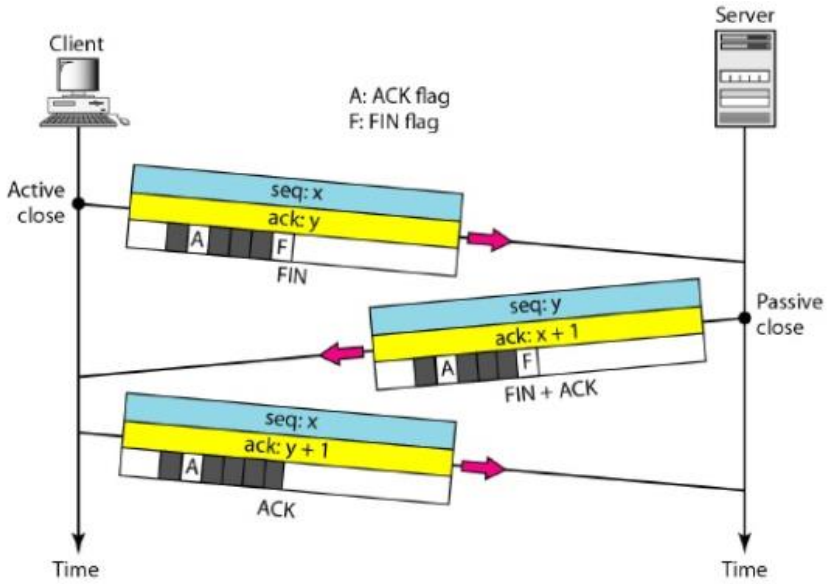
Urgent Data

On occasion an application program needs to send urgent bytes. This means that the sending application program wants a piece of data to be read out of order by the receiving application program.

Suppose sender wants to abort the process, but it has already sent a huge amount of data. **If it issues an abort command (control +C)**, these two characters will be stored at the end of the receiving TCP buffer. It will be delivered to the receiving application program after all the data have been processed. The solution is to send a segment with the **URG** bit set.

Connection Termination

Any of the two parties involved in exchanging data (client or server) can close the connection, **although it is usually initiated by the client.**



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.

   Note:-that a FIN segment can include the last chunk of data sent by the client.

   > The FIN segment consumes one sequence number ifit does not carry data.

2. The server TCP, after receiving the FIN segment, informs its process of the situation and sends the second segment, a FIN +ACK segment, to confirm the receipt of the FIN segment from the client and at the same time to announce the closing of the connection in the other direction. This segment can also contain the last chunk of data from the server.

The FIN + ACK segment consumes one sequence number if it does not carry data.

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

Half-Close

In TCP, one end can stop sending data while still receiving data. This is called a **half-close**. Although either end can issue a half-close, it is normally initiated by the client. It can occur when the server needs all the data before processing can begin.

The client half-closes the connection by sending a FIN segment. The server accepts the half-close by sending the ACK segment. The data transfer from the client to the server stops. The server, however, can still send data. When the server has sent all the processed data, it sends a FIN segment, which is acknowledged by an ACK from the client.

After half-closing of connection, data can travel from the server to the client and acknowledgement can travel from the client to the server. The client cannot send any more data to the server.

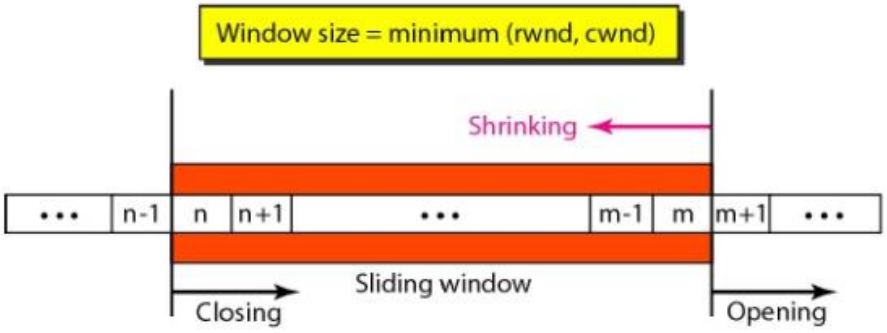
Flow Control

TCP uses a sliding window protocol to handle flow control. This is something **between the Go-Back-N and Selective Repeat** sliding window.

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

Note: Difference b/w sliding protocols used by TCP and Data Link Layer

1. The sliding window of TCP is byte-oriented, but at data link layer is frame-oriented
2. TCP's sliding window is of variable size; at the data link layer was of fixed size



Sliding Window

The window is **opened**, **closed**, or **shrunk**.

a. Opening a window means moving the right wall to the right. This allows more new bytes in the buffer that are eligible for sending.
b. Closing the window means moving the left wall to the right. This means that some bytes have been acknowledged and the sender need not worry about them anymore.
c. Shrinking the window means moving the right wall to the left.

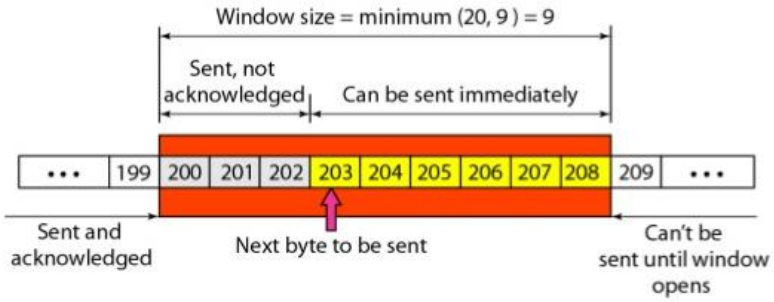
The size of the window at one end is determined by the lesser of two values **receiver window (rwnd)** or **congestion window (cwnd)**.
The **receiver window** is the value advertised by the opposite end in a segment containing acknowledgment. It is the number of bytes the other end can accept before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

Example what is the value of the receiver window (rwnd) for host A if the receiver, host B, has a buffer size of 5000 bytes and 1000 bytes of received and unprocessed data?
Solution
The value of rwnd =5000 - 1000 =4000. Host B can receive only 4000 bytes of data before Overflowing its buffer. Host B advertises this value in its next segment to A.

Example
What is the size of the window for host A if the value of rwnd is 3000 bytes and the value of cwnd is 3500 bytes?
Solution
The size of the window is the smaller of rwnd and cwnd, which is 3000 bytes

Example:
The sender has sent bytes up to 202. We assume that **cwnd** is 20. The receiver has sent an ack number of 200 with an **rwnd** of 9 bytes.

The size of the sender window is the minimum of rwnd and cwnd, or 9 bytes. Bytes 200 to 202 are sent, but not acknowledged. Bytes 203 to 208 can be sent without worrying about acknowledgment. Bytes 209 and above cannot be sent.



Error Control:

Checksum each segment includes a checksum field which is used to check for a corrupted segment. If the segment is corrupted, it is discarded by the destination TCP and is considered as lost.

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

ACK segments do not consume sequence numbers and are not acknowledged.

Retransmission

When a segment is corrupted, lost, or delayed, it is retransmitted. Segment is retransmitted on two occasions: when a **retransmission timer expires** or when the sender receives **three duplicate** ACKs.

> In modern implementations, a retransmission occurs if the retransmission timer expires or three duplicate ACK segments have arrived.

> No retransmission timer is set for an ACK segment.

Retransmission after RTO

A recent implementation of TCP maintains one **retransmission time-out** (RTO) timer for all outstanding (sent, but not acknowledged) segments. Note that **no time-out timer is set for a segment that carries only an acknowledgment**. T**he value of RTO is dynamic in TCP and is updated based on the round-trip time (RTT) of segments**. An RTT is the time needed for a segment to reach a destination and for an acknowledgment to be received.

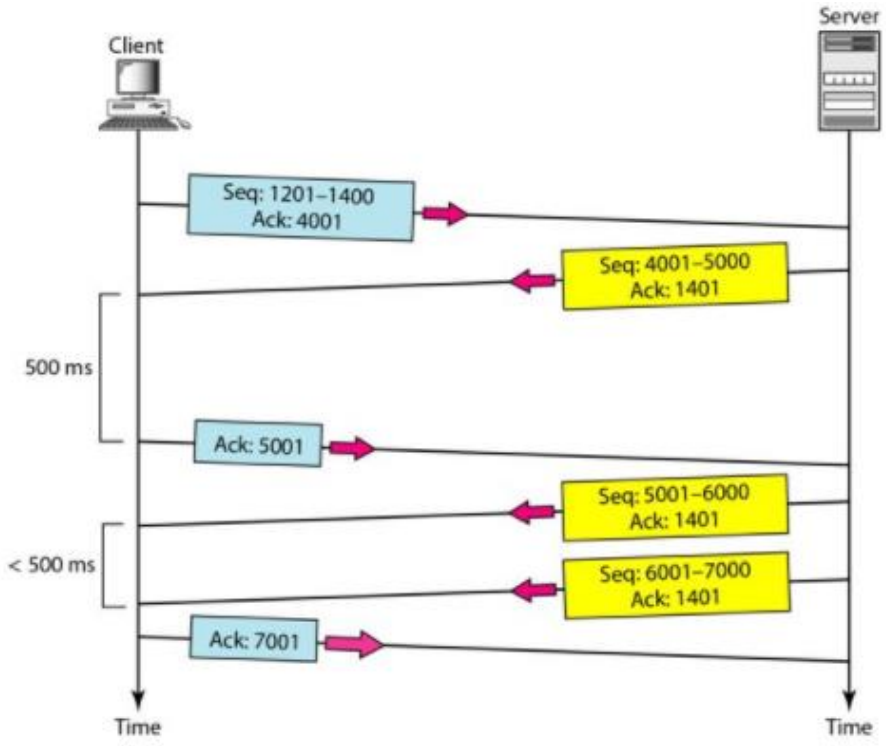
Retransmission After Three Duplicate ACK Segments

One segment is lost and the receiver receives so many out-of-order segments that they cannot be saved (limited buffer size). To remove this situation, most implementations today follow the three-duplicate-ACKs rule and retransmit the missing segment immediately. This feature is referred to as **fast retransmission**.

Out of Order Segments

When a segment is delayed, lost, or discarded, the segments following that segment arrive out of order. TCP **stores them temporarily and flag them as out-of-order segments** until the missing segment arrives.

Normal Operation:

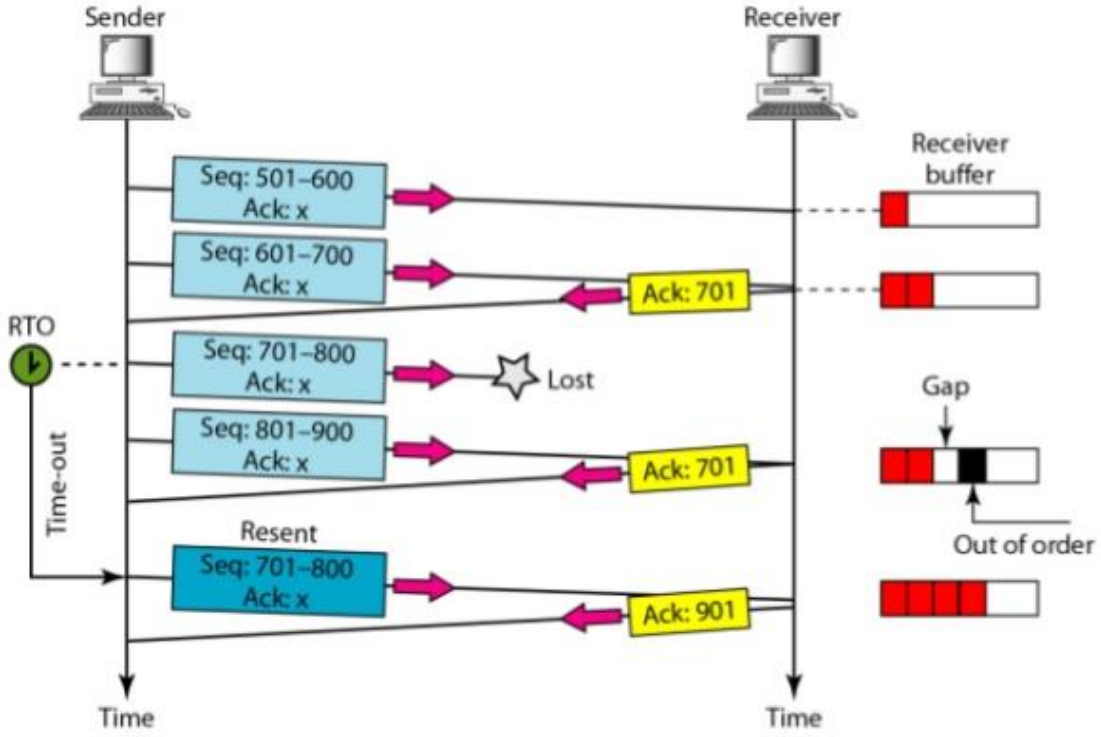


When the client receives the first segment from the server, it does not have any more data to send; it sends only an ACK segment. However, the acknowledgment needs to be delayed for 500 ms to see if any more segments arrive. When the timer matures, it triggers an acknowledgment. This is so because the client has no knowledge if other segments are coming; it cannot delay the acknowledgment forever. When the next segment arrives, another acknowledgment timer is set.
Note: acknowledgment is cumulative

SCTP: (Stream Control Transmission Protocol)

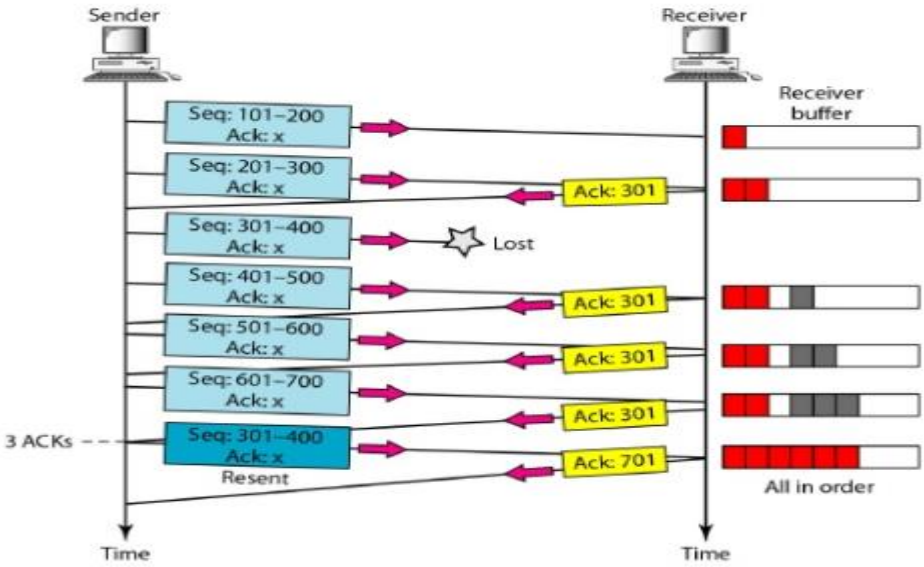
a. UDP is a message-oriented protocol.
b. TCP is a byte-oriented protocol,
c. SCTP combines best features of UDP and TCP, SCTP is a reliable message oriented protocol.

Lost Segment:
A lost segment and a corrupted segment are treated the same way by the receiver. A lost segment is discarded somewhere in the network; a corrupted segment is discarded by the receiver itself. Both are considered lost.



We are assuming that data transfer is unidirectional. The sender sends segments 1 and 2, which are acknowledged immediately by an ACK. Segment 3, however, is lost. The receiver receives segment 4, Which is out of order. The receiver stores the data in the segment in its buffer but leaves a gap to indicate that there is no continuity in the data. The receiver immediately sends an acknowledgment to the sender, displaying the next byte it expects. Note that the receiver stores bytes 801 to 900, but never delivers these bytes to the application until the gap is filled.
Note: TCP starts RTO timer for each sent segment, if ACK not comes sender retransmits it

Fast Retransmission



TCP Congestion Control

The TCP congestion-control mechanism operating at the sender keeps track of an additional variable, the congestion window. The congestion window, denoted **cwnd**, imposes a constraint on the rate at which a TCP sender can send traffic.
The amount of unacknowledged data at a sender may not exceed the minimum of cwnd and rwnd, that is:

$$\text{LastByteSent} - \text{LastByteAcked} \leq \min\{\text{cwnd}, \text{rwnd}\}$$

Choke Packet

A choke packet is a packet sent by a node to the source to inform it of congestion.

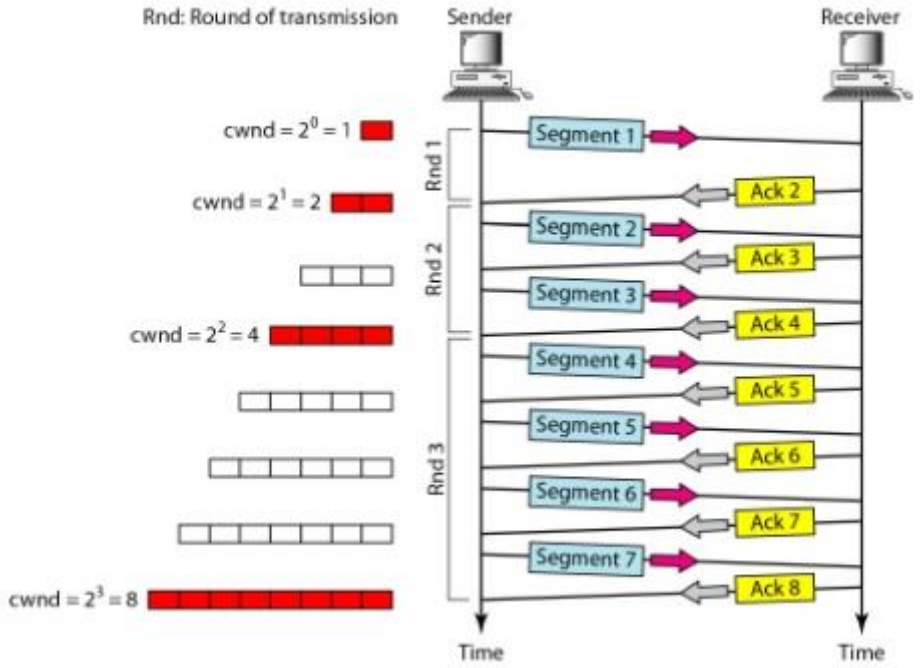
The sender has two pieces of information: the receiver-advertised window size and the congestion window size. The actual size of the window is the minimum of these two.

$$\text{Actual window size} = \text{minimum (rwnd, cwnd)}$$

Slow Start: Exponential Increase

This algorithm is based on the idea that the size of the congestion window **(cwnd) starts with one maximum segment size (MSS)**. The MSS is determined during connection establishment by using an option of the same name. The value of cwnd begins at 1 MSS and **increases by 1 MSS every time a transmitted segment is first acknowledged**. This process results in a doubling of the sending rate every RTT. Thus, the TCP send rate starts slow but grows exponentially during the slow start phase.

The sender starts with **cwnd =1 MSS**. This means that the sender can send only one segment. After receipt of the acknowledgment for segment 1, the size of the congestion window is increased by 1, which means that cwnd is now 2. Now two more segments can be sent. **When each cknowledgment is received, the size of the window is increased by 1 MSS**. When all seven segments are acknowledged, cwnd = 8.



Slow Start, exponential increase

If we look at the size of **cwnd** in terms of rounds (acknowledgment of the whole window of segments), we find that the rate is exponential as shown below:

Start	➡	$cwnd = 1$
After round 1	➡	$cwnd = 2^1 = 2$
After round 2	➡	$cwnd = 2^2 = 4$
After round 3	➡	$cwnd = 2^3 = 8$

There must be a threshold to stop this phase. The sender keeps track of a variable named **ssthresh** (**slow-start threshold**). When the size of window in bytes reaches this threshold, slow start stops and the next phase starts. **In most implementations the value of ssthresh is 65,535 bytes.**

In the slow-start algorithm, the size of the congestion window increases exponentially until it reaches a threshold.

Congestion Avoidance: Additive Increase

In this algorithm, each time the **whole window of segments is acknowledged** (one round), the size of the congestion window is increased by 1.

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

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

Congestion Detection: Multiplicative Decrease

If congestion occurs, the congestion window size (**cwnd**) must be decreased. Retransmission can occur in one of two cases: when **a timer times out** or when **three ACKs are received**.

In both cases, the size of the threshold is dropped to one-half, a multiplicative decrease.

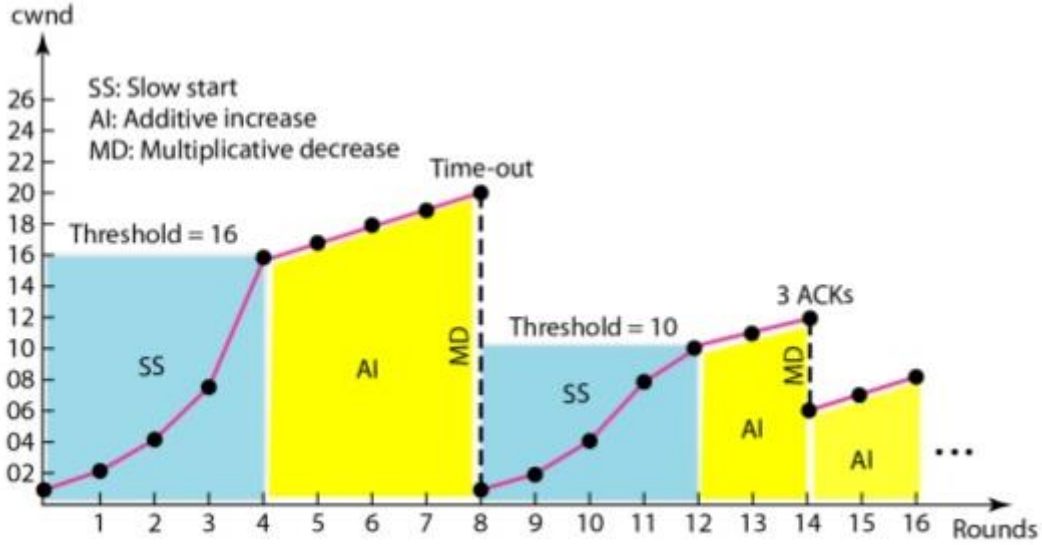
Most TCP implementations have two reactions:

1. If a **time-out occurs**, there is a stronger possibility of congestion; a segment has probably been dropped in the network, and there is no news about the sent segments. In this case TCP reacts strongly:
   a. The TCP sender sets the value of **cwnd to 1** and begins the slow start process anew. It also sets the value of a second state variable, **ssthresh (slow start threshold) to cwnd/2**—half of the value of the congestion window value when congestion was detected.
   b. It starts the slow-start phase again.

2. If **three duplicate ACKs** are received, there is a weaker possibility of congestion; a segment may have been dropped, but some segments after that may have arrived safely since three ACKs are received. This is called **fast transmission and fast recovery**.
   a. It sets the **value of the threshold to one-half of the current window size** same as time-out.
   b. It sets cwnd to the value of the threshold.
   c. It starts the congestion avoidance phase.

Example:

We assume that the maximum window size is 32 segments. The threshold is set to 16 segments (**one-half of the maximum window size**). In the slow-start phase the window size starts from 1 and grows exponentially until it reaches the threshold. After it reaches the threshold, the congestion avoidance (**additive increase**) procedure allows the window size to increase linearly until a timeout occurs or the maximum window size is reached. Time-out occurs when the window size is 20. At this moment, the multiplicative decrease procedure takes over and reduces the threshold to one-half of the previous window size. The previous window size was 20 when the time-out happened so the new threshold is now 10.



TCP moves to slow start again and starts with a window size of 1, and TCP moves to additive increase when the new threshold is reached. When the window size is 12, a three-ACKs event happens. The multiplicative decrease procedure takes over again. The threshold is set to 6 and TCP goes to the

additive increase phase this time. It remains in this phase **until another time-out or another three ACKs happen**.

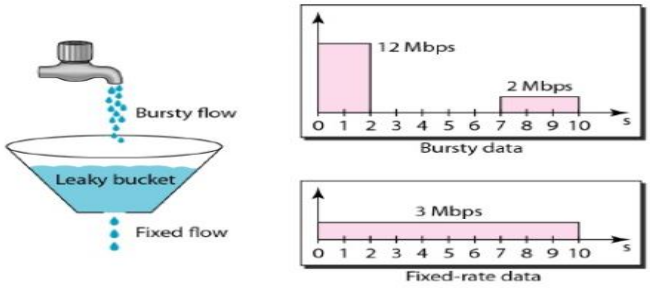
Traffic Shaping

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

a. **Leaky Bucket**

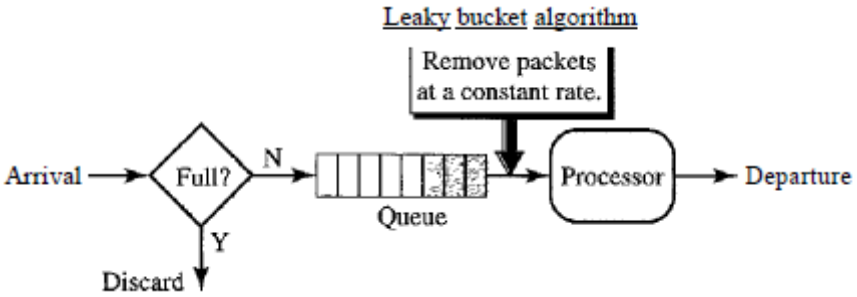
If a bucket has a small hole at the bottom, the water leaks from the bucket at a constant rate as long as there is water in the bucket. The rate at which the water leaks does not depend on the rate at which the water is input to the bucket unless the bucket is empty. **The input rate can vary, but the output rate remains constant**.

Similarly, in networking, a technique called leaky bucket can smooth out **bursty traffic**. **Bursty chunks are stored in the bucket and sent out at an average rate**.



In the figure, we assume that the network has committed a bandwidth of 3 Mbps for a host. **The use of the leaky bucket shapes the input traffic to make it conform to this commitment**. the host sends a burst of data at a rate of 12 Mbps for 2s, for a total of 24 Mbits of data. The host is silent for 5s and then sends data at a rate of 2 Mbps for 3s, for a total of 6 Mbits of data. In all, the host has sent 30 Mbits of data in l0s. The leaky bucket smooths the traffic by sending out data at a rate of 3 Mbps during the same 10 s. Without the leaky bucket, the beginning burst may have hurt the network by consuming more bandwidth than is set aside for this host. We can also see that the **leaky bucket may prevent congestion**.

A simple leaky bucket implementation is shown in the below figure. **A FIFO queue holds the packets**. If the traffic consists of fixed-size packets the process removes a fixed number of packets from the queue at each tick of the clock. If the traffic consists of variable-length packets, the fixed output rate must be based on the number of bytes or bits.



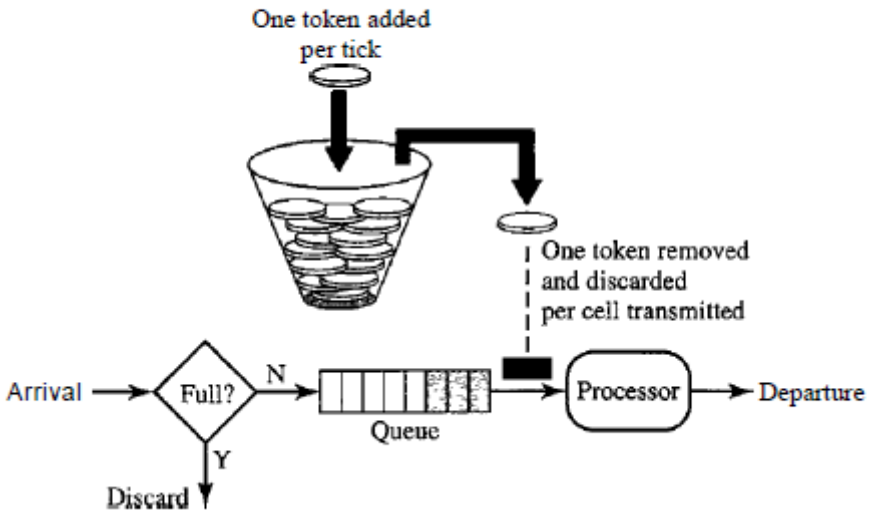
A leaky bucket algorithm shapes bursty traffic into fixed-rate traffic by averaging the data rate. It may drop the packets if the bucket is full.

Token Bucket

The l**eaky bucket is very restrictive**. It does not credit an idle host. For example, if a host is not sending for a while, its bucket becomes empty. Now if the host has bursty data, **the leaky bucket allows only an average rate**. The token bucket algorithm allows **idle hosts to accumulate credit for the future in the form of tokens**.
For each tick of the clock, the system sends **n tokens to the bucket**. The system removes one token for every cell (or byte) of data sent. For example, **if n is 100 and the host is idle for 100 ticks, the bucket collects 10,000 tokens. Now the host can consume all these tokens in one tick with 10,000 cells, or the host takes 1000 ticks with 10 cells per tick**. In other words, **the host can send bursty data as long as the bucket is not empty**.
The token bucket can easily be implemented with a counter. The token is initialized to zero. Each time a token is added, the counter is incremented by 1. Each time a unit of data is sent, the counter is decremented by 1. When the counter is zero, the host cannot send data.

The token bucket allows bursty traffic at a regulated maximum rate

Token bucket



If C is the maximum capacity of bucket and p is the token arrival rate and M is the maximum output rate then the burst length S can be calculated as:

C + pS = MS

Example:-
Consider a frame relay network having a capacity of 1Mb of data is arriving at the rate of 25Mbps for 40msec. The Token arrival rate is 2Mbps and the capacity of bucket is 500kb with maximum output rate with 25Mbps. Calculate

1. Burst length
2. Total output time

Solution:

1. C + pS = MS C=500kb, p=2Mbps M= max token output rate = 25Mbps
   500kb + 2Mbps*S = 25Mbps*S
   S = 500kb/23*10^6 = 500/23x1000 = 21.73 ms = 22 ms

2. For 22ms, the output rate is 25Mbps after that the output rate becomes 2mbps i.e. token arrival rate. Therefore, for another 500Kb the time will be taken
   500kb/2mbps = 500/2000 = 250msec
   Therefore total output time = 22+250 = 272 ms

Example:

Example

Consider a frame relay network having a capacity of 1Mb and data is input at the rate of 25mbps.Calculate
1. What is the time needed to fill the bucket.
2. If the output rate is 2 mbps , the time needed to empty the bucket.

Ans.
Here ,
C is Capacity of bucket = 1mb
Data input rate = 25 mbps
output rate = 2mbps.

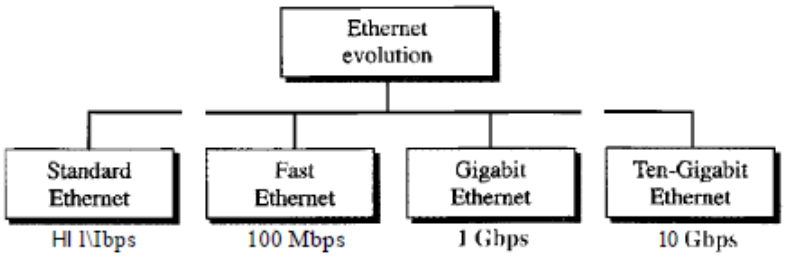
1. T = C/input rate = 1/25 = 40 msec

2 T = C/output rate = ½ = 500 msec

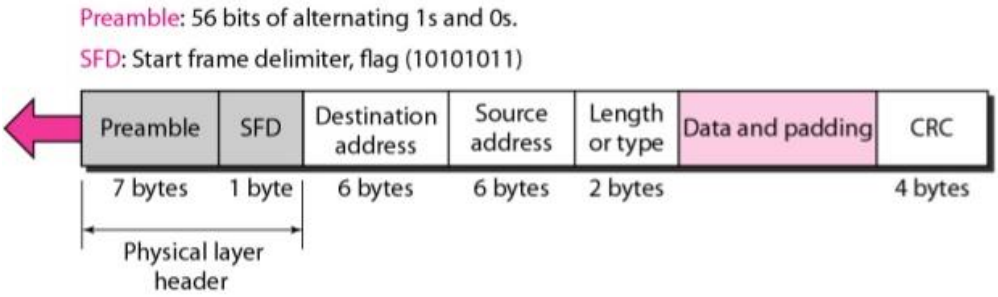
Physical Layer

Standard Ethernet:

Ethernet evolution through four generations



802.3 MAC Frame Format



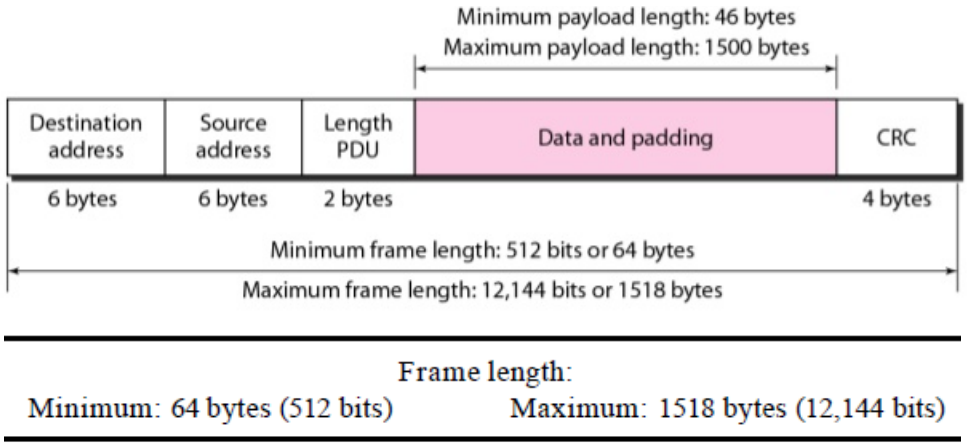
The Ethernet frame contains seven fields: preamble, SFD, DA, SA, length or type of protocol data unit (PDU), upper-layer data, and the CRC.

Note:- Ethernet does not provide any mechanism for acknowledging received frames. Acknowledgments must be implemented at the higher layers.

1. **Preamble** the preamble is actually added at the physical layer and is not (formally) part of the frame. It alerts the receiving system to the coming frame and enables it to synchronize its input timing.
2. **Start frame delimiter (SFD)** it signals the beginning of the frame.
3. **Destination address (DA)** the DA field is 6 bytes and contains the physical address of the destination station or stations to receive the packet.
4. **Source address (SA)** The SA field is also 6 bytes and contains the physical address of the sender of the packet. We will discuss addressing shortly.
5. **Length or type t**he original Ethernet used this field as the type field to define the upper-layer protocol using the MAC frame. The IEEE standard used it as the length field to define the number of bytes in the data field.
6. **Data** this field carries data encapsulated from the upper-layer protocols**. It is a minimum of 46 and a maximum of 1500 bytes**.
7. **CRC** the last field contains error detection information.

Frame Length

Ethernet has imposed restrictions on both the minimum and maximum lengths of a frame.



Frame length:

Minimum: 64 bytes (512 bits) Maximum: 1518 bytes (12,144 bits)

MAC Addressing

Each station on an Ethernet network (such as a PC, workstation, or printer) has its own **network interface card (NIC)**. The NIC fits inside the station and provides the station with a **6-byte physical address**.

Example of an Ethernet address in hexadecimal notation

06:01 :02:01:2C:4B

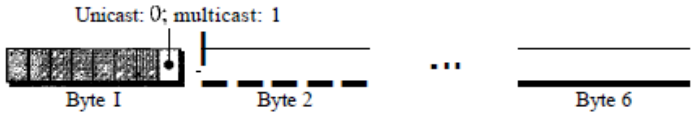
6 bytes = 12 hex digits = 48 bits

Unicast, Multicast, and Broadcast Addresses

A **source address** is always a unicast address-the frame comes from only one station. The destination address, however, can be unicast, multicast, or broadcast.

If the **least significant bit of the first byte** in a destination address is 0, the address is unicast; otherwise, it is multicast.

Unicast and multicast addresses



The least significant bit of the first byte defines the type of address. If the bit is 0, the address is unicast; otherwise, it is multicast.

The broadcast address is a special case of the multicast address; the recipients are all the stations on the LAN.

A broadcast destination address is forty-eight 1's.

Define the type of the following destination addresses:
a. 4A:30:10:21:1O:1A (A=1010 even, hence unicast)
b. 47:20:1B:2E:08:EE (7 = 0111, odd, multicast)
c. FF:FF:FF:FF:FF:FF (All 1's, hence broadcast)

Access Method: CSMA/CD

Standard Ethernet uses 1-persistent CSMA/CD.

Slot Time

In an Ethernet network, the round-trip time required for a frame to travel from one end of a maximum-length network to the other plus the time needed to send the jam sequence is called the slot time.

Slot time = round-trip time + time required to send the jam sequence

Slot Time and Collision:

We assume that the sender sends a minimum-size packet of 512 bits. Before the sender can send the entire packet out, the signal travels through the network and reaches the end of the network. If there is another signal at the end of the network (worst case), a collision occurs. The sender has the opportunity to abort the sending of the frame and to send a jam sequence to inform other stations of the collision. **The round-trip time plus the time required to send the jam sequence should be less than the time needed for the sender to send the minimum frame**.

The sender needs to be aware of the collision before it has sent the entire frame.

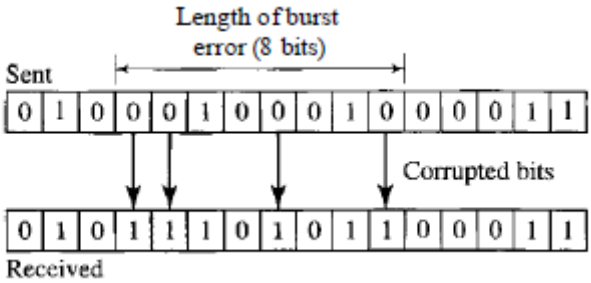
In the second case, the **sender sends a frame larger than the minimum size** (between 512 and 1518 bits). In this case, **if the station has sent out the first 512 bits and has not heard a collision, it is guaranteed that collision will never occur during the transmission of this frame**.

Error Detection and Correction

Data can be corrupted during transmission.
Some applications require that errors be detected and corrected.

Single-Bit Error 1 bit of a given data unit (such as a byte, character, or packet) is changed from 1 to 0 or from 0 to 1.

Burst Error A burst error means that 2 or more bits in the data unit have changed. Note that a burst error does not necessarily mean that the errors occur in consecutive bits. **The length of the burst is measured from the first corrupted bit to the last corrupted bit**. Some bits in between may not have been corrupted.



Redundancy
To detect or correct errors, we need to send extra (redundant) bits with data.

Error Detection

a. The receiver has (or can find) a list of valid codewords.
b. The original codeword has changed to an invalid one.

Hamming Distance:

The Hamming distance between two words (of the same size) is **the number of differences between the corresponding bits**. We show the **Hamming distance between two words x and y as d(x, y)**.

The Hamming distance can easily be found if we apply the **XOR operation on the two words and count the number of 1's in the result**.

Note that the Hamming distance is a value greater than zero.

Example 10.4

Let us find the Hamming distance between two pairs of words.

1. The Hamming distance *d(000, 011)* is 2 because 000 ⊕ 011 is 011 (two Is).
2. The Hamming distance *d(10101, 11110)* is 3 because 10101 ⊕ 11110 is 01011 (three Is).

Minimum Hamming Distance

smallest Hamming distance between all possible pairs. We use d_{min} to define the minimum Hamming distance in a coding scheme. To find this value, we find the Hamming distances between all words and select the smallest one.

The minimum Hamming distance is the smallest Hamming distance between all possible pairs in a set of words.

Example: Find the minimum Hamming distance of the coding scheme.

Codewords
000
011
101
110

Solution

We first find all Hamming distances.

$d(000, 011) = 2$ $d(000, 101) = 2$ $d(0aO, 110) = 2$ $d(Oll, 101) = 2$
$d(Oll, 110) = 2$ $d(W1, 110) = 2$

The d_{min} in this case is 2.

Example 2:

Codeword
00000
01011
10101
11110

$d(00000, 01011) = 3$ $d(00000, 10101) = 3$ $d(00000, 11110) = 4$
$d(01011, 10101) = 4$ $d(OlO11, 11110) = 3$ $d(10101, 11110) = 3$

The d_{min} in this case is 3.

Three Parameters

Any coding scheme needs to have at least **three parameters**:

1. codeword size n
2. dataword size k
3. minimum Hamming distance dmin

A coding scheme C is written as C(n, k) with a separate expression for dmin.

For example, we can call our first coding scheme C(3, 2) with dmin =2 and our second coding scheme C(5, 2) with dmin = 3.

Minimum Distance for Error Detection

Find the minimum Hamming distance in a code if we want to be able to detect up to **s errors**. If s errors occur during transmission, **the Hamming distance between the sent codeword and received codeword is s**. **If our code is to detect up to s errors**, **the minimum distance between the valid codes must be s + 1**, so that the received codeword does not match a valid codeword. In other words**, if the minimum distance between all valid codewords is s + 1, the received codeword cannot be erroneously mistaken for another codeword**.

Note: Although a code with dmin =s + 1, may be able to detect **more than s errors in some special** cases, only s or fewer errors are guaranteed to be detected.

> To guarantee the detection of up to s errors in all cases, the minimum Hamming distance in a block code must be $d_{min} = s + 1$.

Minimum Distance for Error Correction

> To guarantee correction of up to t errors in all cases, the minimum Hamming distance in a block code must be $d_{min} = 2t + 1$.

Example:

A code scheme has a Hamming distance dmin = 4. What is the error detection and correction capability of this scheme?

Solution:

This code guarantees the detection of up to three **errors (s == 3)**, but it can correct up to **one error**. In other words, **if this code is used for error correction, part of its capability is wasted**. **Error correction codes need to have an odd minimum distance (3, 5, 7 ...)**.

Linear Block Codes:-

> In a linear block code, the exclusive OR (XOR) of any two valid codewords creates another valid codeword.

Codeword
00000
01011
10101
11110

Exoring of 2nd code and 3rd creates 4th code word. This scheme is also a linear block code. We can create all four codewords by XORing two other codewords.

Minimum Distance for Linear Block Codes:

It is simple to find the minimum Hamming distance for a linear block code. The minimum Hamming distance is **the number of 1's in the nonzero valid codeword with the smallest number of 1's**.

Let us now show some linear block codes:

Simple Parity-Check Code: (Even Parity)

In this code, **a k-bit dataword is changed to an n-bit codeword where n = k + 1**. **The extra bit, called the parity bit**, is selected **to make the total number of 1's in the codeword even**.

The **minimum Hamming distance for this category is dmin =2**, which means that the code is a single-bit error-detecting code; **it cannot correct any error**.

A simple parity-check code is a single-bit error-detecting code in which $n = k + 1$ with $d_{min} = 2$.

Datawords	*Codewords*
1000	10001
1001	10010
1010	10100
1011	10111
1100	11000
1101	11011
1110	11101
1111	11110

K = 4 and n=5

r0 = a3 + a2 + a1 + a0 (modulo 2)

If the number of 1s is even, the result is 0; if the number of 1s is odd, the result is 1. In both cases, the total number of 1s in the codeword is even.

The checker at the receiver does the same thing as the generator in the sender with one exception:

The syndrome is 0 when the number of 1's in the received codeword is even; otherwise, it is 1.

S0 = b3 + b2 + b1 + b0 + q0 (modulo 2)

A simple parity-check code can detect an odd number of errors.

Two-dimensional parity check:

The dataword is organized in a table (rows and columns). The two-dimensional parity check can **detect up to three errors that occur anywhere in the table**.

Error Correcttion v/s Detection

In error detection we are looking only to see if any error has occurred, we are not interested in number of errors. Single bit error is same as burst error for us.

In error correction, we need to know the exact no. of bits that are corrupted and their exact location in the message.

Note:- An error detecting code can detect only the types of errors for which it is designed; other types of errors for which it is designed, other types of errors may remain undetected.

To calculate a codeword that is capable of correcting all **single-bit errors** in an 8-bit data element. In the codeword, **there are m data bits and r redundant (check) bits, giving a total of n codeword bits**

n= m+r

so

$$(m+r+1) \leq 2^r$$

Example: We has a dataword of 7 bits, calculate values of k and n that satisfy the requirement

7 + 4 + 1 <= 2^4 hence there will be 4 redundant bits required.

m=7, r =4 hence C(n,r) = C(11, 4)

Codeword	r	r	0	r	0	0	0	r	1	0	1	0
Bit No.	1	2	3	4	5	6	7	8	9	10	11	12

Note:-Bits at positions of the power 2 as 1,2,4,8 …. will contain redundant bits

r1	r2	d2	d3	r4	d5	d6	d7	r8	d9	d10	d11

Check bit 1 governs positions 1, 3, 5, 7, 9: Value = 1 (0 0 0 1)
Check bit 2 governs positions 2, 3, 6, 7, 10: Value = 0 (0 0 0 0)
Check bit 4 governs positions 4, 5, 6, 7, 12 = 0 (0 0 0 0)
Check bit 8 governs positions 8, 9, 10, 11, 12 = 0 (1 0 1 0)

How to correct 1 bit error:

Assume the codeword is corrupted to give 1010 0000 1010. Check parity:

Bit 1: $B1 \oplus B3 \oplus B5 \oplus B7 \oplus B9 = 1 \oplus 1 \oplus 0 \oplus 0 \oplus 1 = 1$
Bit 2: $B2 \oplus B3 \oplus B6 \oplus B7 \oplus B10 = 0 \oplus 1 \oplus 0 \oplus 0 \oplus 0 = 1$
Bit 4: $B4 \oplus B5 \oplus B6 \oplus B7 \oplus B12 = 0 \oplus 0 \oplus 0 \oplus 0 \oplus 0 = 0$
Bit 8: $B8 \oplus B9 \oplus B10 \oplus B11 \oplus B12 = 0 \oplus 1 \oplus 0 \oplus 1 \oplus 0 = 0$

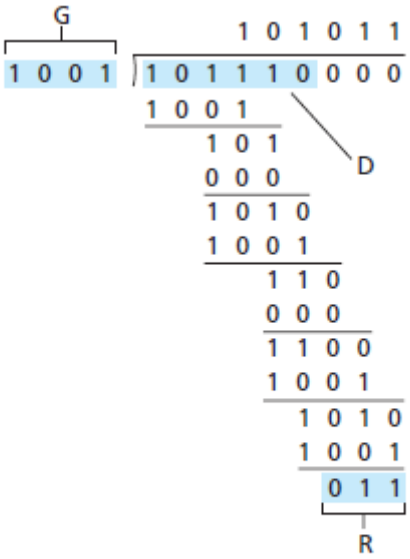
Thus the error is in bit position 1 + 2 = 3, and bit 3 can be inverted to give the correct codeword:

1000 0000 1010

CYCLIC CODES

In a cyclic code, if a codeword is cyclically shifted (rotated), the result is another codeword. For example, if 1011000 is a codeword and we cyclically left-shift, then 0110001 is also a codeword.

Cyclic Redundancy Check (Polynomial Codes)



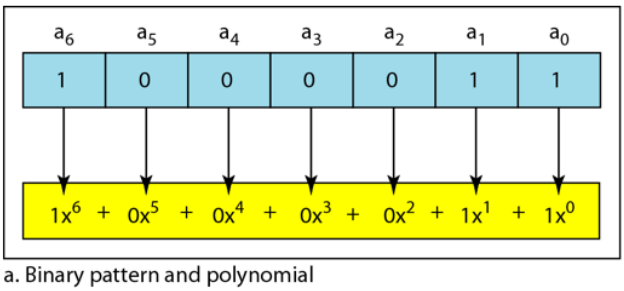
A sample CRC calculation

Generator: x^3 + 1= 1001
Dataword: 101110
We added 3 0's in dataword because generator has 4 digits.
Codeword: Dataword + remainder = **101110-011**

Note: In each step, a copy of the divisor is XORed with the 4 bits of the dividend.

Received codeword =c(x) + e(x)
(c(x) + e(x))/g(x) = C(x)/g(x) + e(x)/g(x), if e(x) completely divides e(x), errors will be undetected.

In a cyclic code, those *e(x)* errors that are divisible by *g(x)* are not caught.



a. Binary pattern and polynomial

Note: If the generator has more than one term and the coefficient of x^0 is 1, all single errors can be caught.

Que: Which of the following g(x) values guarantees that a single-bit error is caught? For each case, what is the error that cannot be caught?

a. x+1 b. x^3 c.1

Solution:

a. No x^i can be divisible by x+1, Any single-bit error can be caught.

b. If i is equal or greater than 3, x^i is divisible by g(x), all single-bit errors in positions 1 to 3 are caught.
c. All values of i makes x^i divisible by g(x). **No single bit error can be caught.** g(x) is useless.

A generator that contains a factor of $x + 1$ can detect all odd-numbered errors.

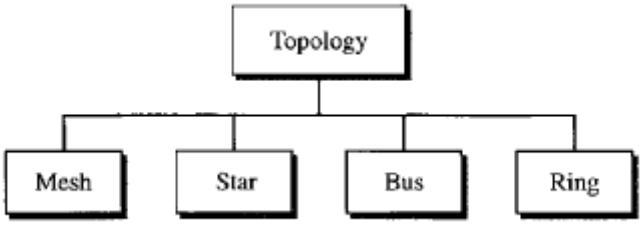
Que:- x^6 + 1, can detect all burst errors with a length less than or equal to 6 bits.

A good polynomial generator needs to have the following characteristics:

1. It should have at least two terms.
2. The coefficient of the term x^0 should be 1.
3. It should not divide X^t + 1, for t between 2 and n - 1.
4. It should have the factor x + 1.

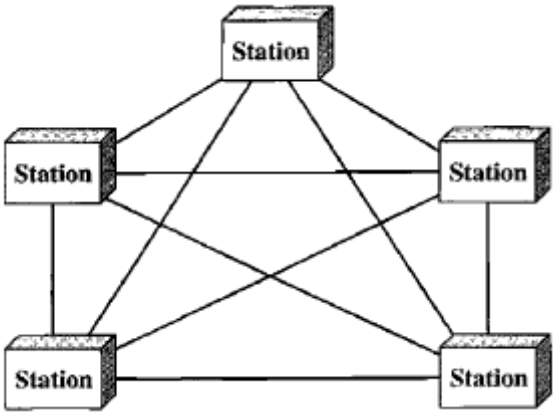
Physical Topology

The term physical topology refers to the way in which a network is laid out physically. Two or more devices connect to a link; two or more links form a topology.



Mesh Topology: (Point to Point)

In a mesh topology, every device has a **dedicated point-to-point link** to every other device. The term dedicated means that the link carries traffic only between the two devices it connects. Every device on the network must have n – 1 input/output (VO) ports. **Total physical links will be n(n-1)/2.**

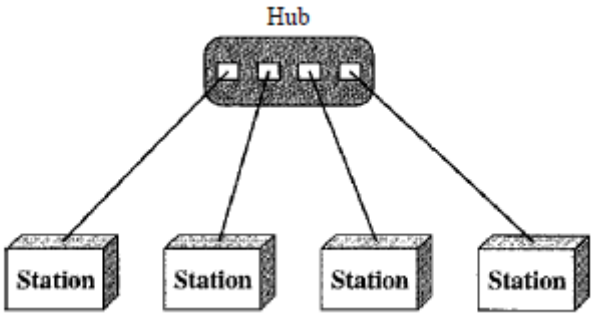


Star Topology (Point to Point)

In a star topology, **each device has a dedicated point-to-point link only to a central controller**, usually called a **hub**. The devices are not directly linked to one another. Unlike a mesh topology, a star topology does not allow direct traffic between devices. The controller acts as an exchange: If one device wants to send data to another, it sends the data to the controller, which then relays the

data to the other connected device. In a star, each device needs **only one link and one I/O port** to connect it to any number of others. Eg: LAN netoworks

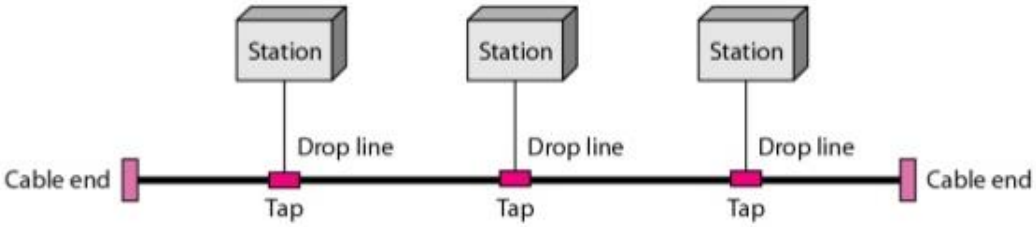
Note: Star Topology is less expensive than Mess Technology. If the hub goes down, the whole system is dead.



Star Topology

Bus Topology (Multipoint)

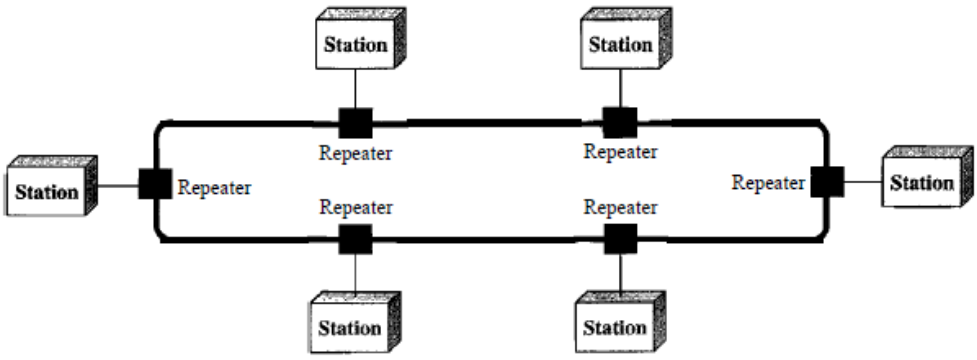
One long cable acts as a **backbone** to link all the devices in a network.



Nodes are connected to the bus cable by drop lines and taps. A **drop line** is a connection running between the device and the main cable. A **tap** is a connector. Advantages of a bus topology include ease of installation.

Ring Topology:

A ring topology, each device has a dedicated point-to-point connection with only the two devices on either side of it. A signal is passed along the ring in one direction, from device to device, until it reaches its destination.



Networking Devices:

1. Repeater → Physical Layer
2. Router → Network Layer
3. Gateway → All Layer

4. Bridge → Data Link Layer

1. Passive Hubs

A passive hub is just a connector. It connects the wires coming from different branches. In a star-topology Ethernet LAN, a passive hub is just a point where the signals coming from different stations collide; the hub is the collision point.

2. Repeaters

A repeater is a device that operates only in the physical layer. A repeater receives a signal and, before it becomes too weak or corrupted, regenerates the original bit pattern. The repeater then sends the refreshed signal. **A repeater can extend the physical length of a LAN**. A repeater does not actually connect two LANs; it connects two segments of the same LAN.

A repeater connects segments of a LAN.

A repeater forwards every frame; it has no filtering capability.

A repeater is a regenerator, not an amplifier.

3. Bridges

A bridge operates in both the physical and the data link layer. As a physical layer device, it regenerates the signal it receives. As a data link layer device, the bridge can check the physical (MAC) addresses (source and destination) contained in the frame. **A bridge has filtering capability**. It can check the destination address of a frame and decide if the frame should be forwarded or dropped. If the frame is to be forwarded, the decision must specify the port. A bridge has a table that maps addresses to ports.

4. **Routers**

A router is a three-layer device that routes packets based on their logical addresses (host-to-host addressing). A router normally connects LANs and WANs in the Internet and has a routing table that is used for making decisions about the route. The routing tables are normally dynamic and are updated using routing protocols.

5. Gateway

A gateway is normally a computer that operates in all five layers of the Internet or seven layers of OSI model. A gateway takes an application message, reads it, and interprets it. This means that it can be used as a connecting device between two internetworks that use different models.

Que:

In a token bucket network transmission spreed is $20*(10^6)$ bps and maximum rate can only be sent for at most 10 sec at a time , and at most 150 Mb can be sent over any 15 sec window, then the value for token input rate is _____ mbps

Solution:

Consider a token bucket with maximum rate $R = 20*(10^6)$ *bps*

suppose we want to make sure that the maximum rate can only be sent for at-most 10 *seconds* at a time , and at- most (maximum) *150 Mb* can be sent over any 15 second window.

Then the required value for which the new tokens are added at the rate of r Mbps which we have to calculate in this question .

Capacity of the token bucket (b) = 150 Mbps
Maximum possible transmission rate (M) = $20*(10^6)$ *bps*
So the maximum burst time = b/(M-r) = 10

Duration = b/M-r, where b is the initial capacity, M is outgoing rate and r is incoming rate

now put b= 150 Mbps, M = 20 Mbps , we need to find r

b= (M-r) 10

b = 10M - 10r

10r = (200 - 150)

r = 50 /10
r = 5

The value for token input rate is 5 Mbps

Points to be remembered:

q1. The Simplest Protocol and the Stop-and-Wait Protocol are for _____ channels.
Ans: Noiseless

Q2: The Stop-And-Wait ARQ, Go-Back-N ARQ, and the Selective Repeat ARQ are for _____ channels.
Ans: Noisy

Q3. The ____ Protocol has neither flow nor error control.
Ans: Simplest Protocol

Q4: The _____Protocol has flow control, but not error control.
Ans: Stop & wait

Q5: The _____Protocol, adds a simple error control mechanism to the _____Protocol.
Ans: Stop-and-Wait ARQ; Stop-and-Wait