UNIT I

FUNDAMENTALS & LINK LAYER

Building a network – Requirements – Layering and protocols – Internet Architecture – Network software – Performance ; Link layer Services – Framing – Error Detection – Flow control

1.1. FUNDAMENTALS

- **Computer Network :** The term network meant the set of serial lines used to attach dumb terminals to mainframe computers.

1.1.1. Building a network

- Computer networks are built primarily from general-purpose programmable hardware, and they are not optimized for a particular application like making phone calls or delivering television signals. Instead, they are able to carry many different types of data, and they support a wide, and ever growing, range of applications.
- A network is generated from the Internet to guide our design. It creates a network architecture that identifies the available hardware and software components and shows how they can be arranged to form a complete network system.
- The following things are used to built a network,
  - ✓ Requirements that different applications and different communities of people place on the network.
  - ✓ A network architecture.
  - ✓ The key elements in the implementation of computer networks.

1.1.1.1. Requirements

This is used to built a network.

i. Perspectives

The following contents are the view of network

- An *application programmer* would list the services that his or her application needs—for example, a guarantee that each message the application sends will be delivered without error within a certain amount of time or the abilityto switch gracefully among different connections to the network as the user moves around.
- A *network operator* would list the characteristics of a system that is easy to administer and manage—for example, in which faults can be easily isolated, new devices can be added to the network and configured correctly, and it is easy to account for usage.
- A *network designer* would list the properties of a cost-effective design—for example, that network resources are efficiently utilized and fairly allocated to different users. Issues of performance are also likely to be important.

ii. Scalable Connectivity

- A network must provide connectivity among a set of computers.
- Sometimes a limited network that connects only a few select machines. The reasons of privacy and security, many private networks have the explicit goal of limiting the set of machines that are connected.
- A system that is designed to support growth to an arbitrarily large size is said to *scale*.
- Links, Nodes, and Clouds
  - Connectivity occurs at many different levels.

- At the lowest level, a network can consist of two or more computers (nodes) directly connected by some physical medium *(link)*, such as a coaxial cable or an optical fiber. There are two types of connectivity, *direct connectivity* and *indirect Connectivity*

Direct Connectivity

- As illustrated in Figure 1.1, physical links are sometimes limited to a pair of nodes such a link is said to be *point-to-point*, while in other cases more than two nodes may share a single physical link such a link is said to be *multiple-access*.

  Wireless links, such as those provided by cellular networks and Wi-Fi networks, are an increasingly important class of multiple-access links.

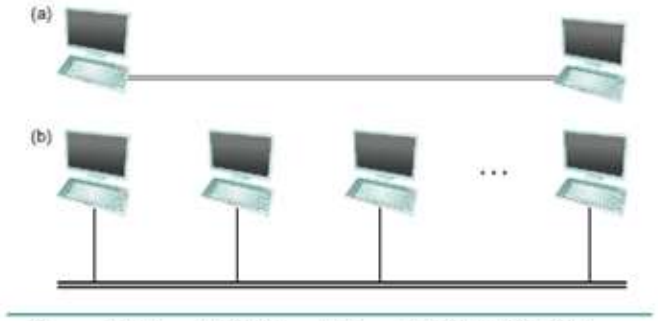


Figure 1.1. Direct Links (a) point-to-point link (b) Muliple Access

- Figure 1.2 shows a set of nodes, each of which is attached to one or more point-to-point links. Those nodes that are attached to at least two links run software that forwards data received on one link out on another. A *switched network* consists of a series of interlinked nodes, called *switches*.

✓ There are two types of switched networks, *circuit switched* and *packet switched*.

- A *circuit-switched network* is made of a set of switches connected by physical links, in which each link is divided into *n* channels. E.g. telephone system.
  - A circuit-switched network first establishes a dedicated circuit across a sequence of links and then allows the source node to send a stream of bits across this circuit to a destination node.

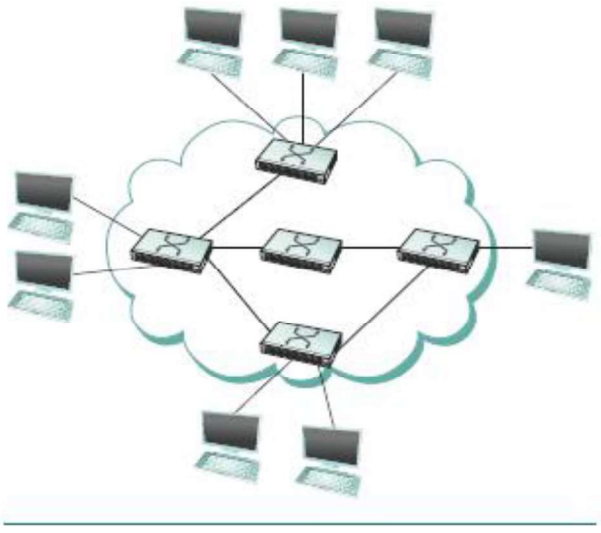


Figure 1.2. Switched Network

- The important feature of *packet-switched networks* is that the nodes in such a network send discrete blocks of data to each other. Each block of data is called as either a *packet* or a *message*. Packet-switched networks use a strategy called *store-and-forward*.
  - Each node in a store-and-forward network first receives a complete packet over some link, stores the packet in its internal memory, and then forwards the complete packet to the nextnode.

✓ Packet switched network is efficient than circuit-switched network

- The cloud in Figure 1.2 distinguishes between the nodes on the inside that *implement* the network (they are commonly called *switches*, and their primary function is to store and forward packets) and the nodes on the outside of the cloud that *use* the network (they are commonly called *hosts*, and they support users and run application programs).

Indirect Connectivity

- A set of computers can be indirectly connected is shown in Figure 1.3. In this situation, a set of independent networks (clouds) are interconnected to form an *internetwork*, or internet for short.
- A node that is connected to two or more networks is commonly called a *router* or *gateway* which is it forwards messages from one network to another.

❖ Address

➢ An address is a byte string that identifies a node; that is, the network can use a node's address to distinguish it from the other nodes connected to the network.

➢ When a source node wants the network to deliver a message to a certain destination node, it specifies the address of the destination node.

➢ If the sending and receiving nodes are not directly connected, then the switches and routers of the network use this address to decide how to forward the message toward the destination. The process of determining systematically how to forward messages toward the destination node based on its address is called *routing*.

➢ The source node wants to send a message to a single destination node (*unicast*). While this is the most common scenario, it is also possible that the source node might want to *broadcast* a message to all the nodes on the network.

➢ A source node might want to send a message to some subset of the other nodes but not all of them, a situation called *multicast*.

iii. Cost-Effective Resource Sharing (key requirement of computer networks**)**

❖ *Multiplexing*, which means that a system resource is shared among multiple users.

❖ At an intuitive level, multiplexing can be explained by analogy to a timesharing computer system, where a single physical processor is shared (multiplexed) among multiple jobs, each of which believes it has its own private processor.

❖ Consider the simple network illustrated in Figure 1.5, where the three hosts on the left side of the network (senders S1–S3) are sending data to the three hosts on the right (receivers R1–R3) by sharing a switched network that contains only one physical link. (For simplicity, assume that host S1 is sending data to host R1, and so on.)

❖ In this state , three flows of data corresponding to the three pairs of hosts are multiplexed onto a single physical link by switch 1 and then *demultiplexed* back into separate flows by switch 2.

❖ There are several different methods for multiplexing multiple flows onto one physical link. They are,

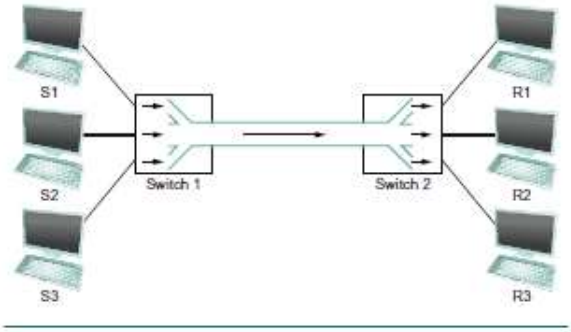


Figure 1.4. Multiplexing multiple logical flows over single physical link

i. *Synchronous time division multiplexing*

ii. *Frequency division multiplexing*

iii. *statistical multiplexing.*

Synchronous Time Division Multiplexing(STDM)

- STDM is to divide time into equal-sized quanta and, in a round-robin fashion, give each flow a chance to send its data over the physical link. E.g, during time quantum 1, data from S1 to R1 is transmitted; during time quantum 2, data from S2 to R2 is transmitted; in quantum 3, S3 sends data to R3. At this point, the first flow (S1 to R1) gets to go again, and the process repeats.

Frequency Division Multiplexing (FDM)

- FDM is to transmit each flow over the physical link at a differentfrequency, much the same way that the signals for different TV stations are transmitted at a different frequency over the airwaves or on a coaxial cable TV link.

Limitations of STDM and FDM

1. if one of the flows (host pairs) does not have any data to send, its share of the physical link that is, its time quantum or its frequency remains idle, even if one of the other flows has data to transmit. For example, S3 had to wait its turn behind S1 and S2 in the previous paragraph, even if S1 and S2 had nothing to send. For computer communication, the amount of time that a link is idle can be very large—for example, consider the amount of time you spend reading a web page (leaving the link idle) compared to the time you spend fetching the page.
2. Both STDM and FDM are limited to situations in which the maximum number of flows is fixed and known ahead of time. It is not practical to resize the quantum or to add additional quanta in the case of STDM or to add new frequencies in the case of FDM.

Statistical Multiplexing

- There are two keys in this technique,
    1. Each flowsends a sequence of packets over the physical link, with a decision made on a packet-by-packet basis as to which flow's packet to send next. If only one flow has data to send, then it can send a sequence of packets back-to-back; however, should more than one of the flows have data to send, then their packets are interleaved on the link. Figure 1.5 depicts a switch multiplexing packets from multiple sources onto a single shared link.

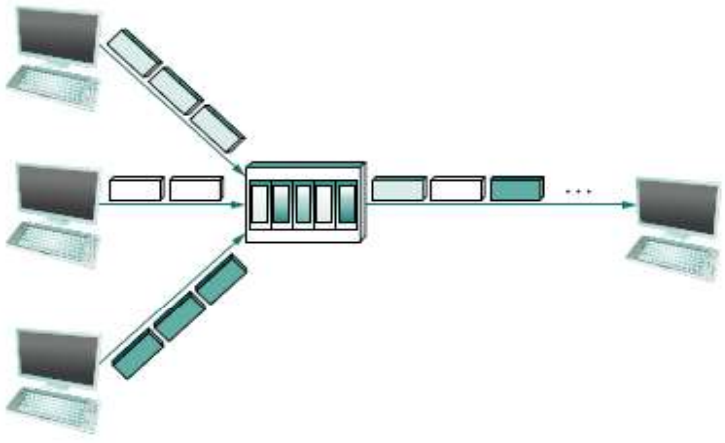


Figure 1.5. A switch multiplexing packets from multiple sources on to one shared link

The decision as to which packet to send next on a shared link can be made in a number of different ways. For example, in a network consistingof switches interconnected by links such as the one in Figure 1.4, the decision would be made by the switch that transmits packets onto the shared link. Each switch in a packet-switched network makes this decision independently, on a packet-by-packet basis. For example, a switch could be designed to service packets on a first-in, first-out (FIFO) basis.

2. To transmit the packets from each of the different flows that are currently sending data through the switch in a round-robin manner. A network that attempts to allocate bandwidth to particular flows is sometimes said to support *quality of service* (QoS).

Figure 1.5 that since the switch has to multiplex three incoming packet streams onto one outgoing link, it is possible that the switch will receive packets faster than the shared link can accommodate. The switch is forced to buffer these packets in its memory. a switch receive packets faster than it can send them for an extended period of time, then the switch will run out of buffer space, and some packets will have to be dropped. When a switch is operating in this state, it is said to be *congested*.

iv. Support for Common Services

- When two application programs need to communicate with each other, a lot of complicated things must happen beyond simply sending a message from one host to another. One option would be for application designers to build all that complicated functionality into each application program. Since many applications need common services, it is much more logical to implement those common services once and then to let the application designer build the application using those services.
- The challenge for a network designer is to identify the right set of common services.
- ***Goal:*** To hide the complexity of the network from the application without overly constraining the application designer.
- To create a common services, use a cloud to abstractly represent connectivity among a set of computers, and a channel as connecting one process to another.
- Figure 1.6 shows a pair of application-level processes communicating over a logical channel that is, in turn, implemented on top of a cloud that connects a set of hosts. Here the channel as being like a pipe connecting two applications, so that a sending application can put data in one end and expect that data to be

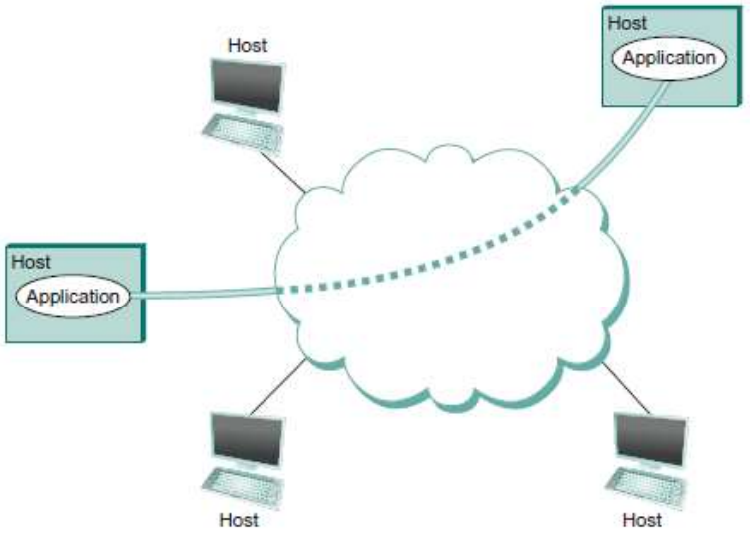


Figure 1.6. Process Communicating over an abstract channel

delivered by the network to the application at the other end of the pipe. The challenge is to recognize what functionality the channels should provide to application programs. For example,

- Does the application require a guarantee that messages sent over the channel are delivered, or is it acceptable if some messages fail to arrive?
- Is it necessary that messages arrive at the recipient process in the same order in which they are sent, or does the recipient not care about the order in which messages arrive?
- Does the network need to ensure that no third parties are able to eavesdrop on the channel, or is privacy not a concern?

❖ ***How to design abstract channel?***

1. *Identifying Common Communication Patterns :* first understanding the communication needs of a representative collection of applications, then extracting their common communication requirements, and finally incorporating the functionality that meets these requirements in the network.
   - For example File Transfer Protocol (FTP) or Network File System (NFS), whether whole files are transferred across the network or only single blocks of the file are read/ written at a given time. The process that requests access to the file is called the *client*, and the process that supports access to the file is called the *server*.
   - Reading a file involves the client sending a small request message to a server and the server responding with a large message that contains the data in the file.Writing works in the opposite way—the client sends a large message containing the data to be written to the server, and the server responds with a small message confirming that the write to disk has taken place.
   - A digital library is a more sophisticated application than file transfer, but it requires similar communication services. For example, the *Association for Computing Machinery* (ACM) operates a large digital library of computer science literature at http://portal.acm.org/dl.cfm
   - The two video applications videoconferencing and video on demand as a representative sample, we might decide to provide the following two types of channels: *request/reply* channels and *message stream* channels. The request/reply channel would be used by the file transfer and digital library applications.
2. *Reliability :* Before reliability evaluation First understanding how networks can fail?. There are three general classes of failure
   - First, as a packet is transmitted over a physical link, *bit errors* may be introduced into the data; that is, a 1 is turned into a 0 or *vice versa*. Sometimes single bits are corrupted, but more often than not a *burst error* occurs several consecutive bits are corrupted. Once detected, it is sometimes possible to correct for such errors if we know which bit or bits are corrupted, we can simply flip them—while in other cases the damage is so bad that it is necessary to discard the entire packet. In such a case, the sender may be expected to retransmit the packet.
   - The second class of failure is at the packet, rather than the bit, level; that is, a complete packet is lost by the network. One reason this can happen is that the packet contains an uncorrectable bit error and therefore has to be discarded. For example, a switch that is forwarding it from one link to another is

so overloaded that it has no place to store the packet and therefore is forced to drop it. This is the problem of congestion mentioned.

- The third class of failure is at the node and link level; that is, a physical link is cut, or the computer it is connected to crashes. This can be caused by software that crashes, a power failure, or a reckless backhoe operator.
- Failures due to misconfiguration of a network device are also common.

❖ The challenge is to fill in the gap between what the application expects and what the underlying technology can provide. This is sometimes called the *semantic gap*.

v. Manageability

Managing a network includes making changes as the network grows to carry more traffic or reach more users, and troubleshooting the network when things go wrong or performance isn't as desired.

1.1.2. Data Flow

Communication between two devices can be simplex, half-duplex, or full-duplex as shown in following Figure.

i. *Simplex :* In simplex mode, the communication is unidirectional, as on a one-way street. Only one of the two devices on a link can transmit; the other can only receive (see Fig a). Keyboards and traditional monitors are examples of simplex devices. The keyboard can only introduce input; the monitor can only accept output. The simplex mode can use the entire capacity of the channel to send data in one direction.

ii. *Half-Duplex* : In half-duplex mode, each station can both transmit and receive, but not at the same time. When one device is sending, the other can only receive, and vice versa (see Fig b).

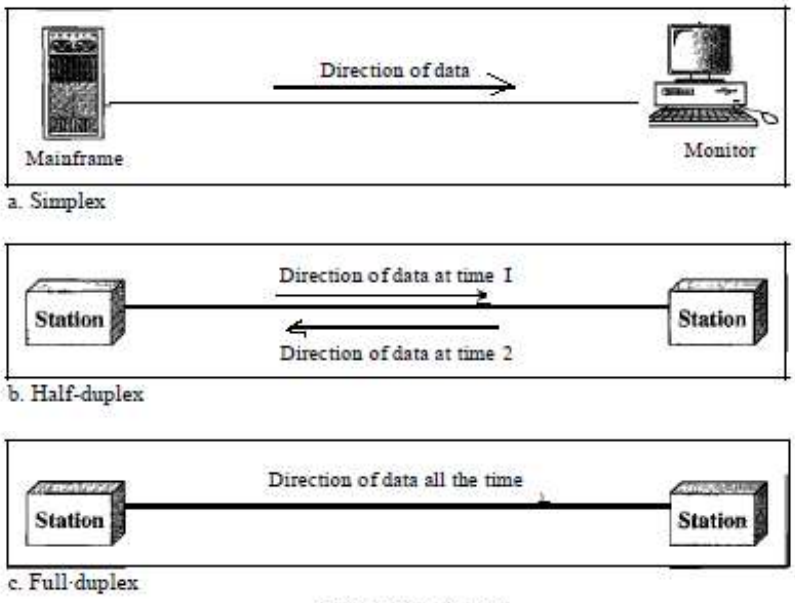


Fig Data Flow

The half-duplex mode is like a one-lane road with traffic allowed in both directions. When cars are traveling in one direction, cars going the other way must wait.

In a half-duplex transmission, the entire capacity of a channel is taken over by whichever of the two devices is transmitting at the time.

Walkie-talkies and CB (citizens band) radios are both half-duplex systems. The half-duplex mode is used in cases where there is no need for communication in both directions at the same time; the entire capacity of the channel can be utilized for each direction.

iii. *Full-Duplex :* In full-duplex mode (also called duplex), both stations can transmit and receive simultaneously (see Fig c). The full-duplex mode is like a two-way street with traffic flowing in both directions at the same time. In full-duplex mode, signals going in one direction share the capacity of the link with signals going in the other direction. This sharing can occur in two ways:

a. Either The link must contain two physically separate transmission paths, one for sending and the other for receiving;

b. Or The capacity of the channel is divided between signals traveling in both directions.

One common example of full-duplex communication is the telephone network. When two people are communicating by a telephone line, both can talk and listen at the same time.

1.1.3. Networks

A network is a set of devices (often referred to as *nodes)* connected by communication links. A node can be a computer, printer, or any other device capable of sending and/or receiving data generated by other nodes on the network.

1. *Distributed Processing* : Most networks use distributed processing, in which a task is divided among multiple computers.
2. *Network Criteria :* A network must be able to meet a certain number of criteria. The most important of these are performance, reliability, and security.
   a. *Performance :*Performance can be measured in many ways, including transit time and response time.
      Transmit time is the amount of time required for a message to travel from one device to another.
      Response time is the elapsed time between an inquiry and a response. The performance of a network depends on a number of factors
      i. the number of users
      ii. the type of transmission medium
      iii. the capabilities of the connected hardware
      iv. the efficiency of the software.
      Performance is often evaluated by two networking metrics:
      i. throughput
      ii. delay.
      Network need more throughput and less delay. If we try to send more data to the network, we may increase throughput but we increase the delay because of traffic congestion in the network.
   b. *Reliability :* In addition to accuracy of delivery, network reliability is measured by the frequency of failure, the time it takes a link to recover from a failure, and the network's robustness in a catastrophe.
   c. *Security :* Network security issues include protecting data from unauthorized access, protecting data from damage and development, and implementing policies and procedures for recovery from breaches and data losses.

1.1.4. Physical Structures -Network attributes

Type of Connection

A network is two or more devices connected through links. A link is a communications pathway that transfers data from one device to another. For visualization purposes, it is simplest to imagine any link as a line drawn between two points.

For communication to occur, two devices must be connected in some way to the same link at the same time. There are two possible types of connections: point-to-point and multipoint.

i. *Point-to-Point* : A point-to-point connection provides a dedicated link between two devices. The entire capacity of the link is reserved for transmission between those two devices.

Most point-to-point connections use an actual length of wire or cable to connect the two ends, but other options, such as microwave or satellite links, are also possible (see Fig 1-6a). When you change television channels by infrared remote control, you are establishing a point-to-point connection between the remote control and the television's control system.

ii. *Multipoint* : A multipoint (also called multidrop) connection is one in which more than two specific devices share a single link (see Fig 1-6b).In a multipoint environment, the capacity of the channel is shared, either spatially or temporally.

If several devices can use the link simultaneously, it is a *spatially shared* connection.

If users must take turns, it is a *timeshared* connection.

1.1.5. 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. The topology of a network is the geometric representation of the relationship of all the links and linking devices (usually called nodes) to one another. There are four basic topologies possible: mesh, star, bus, and ring (see Fig).

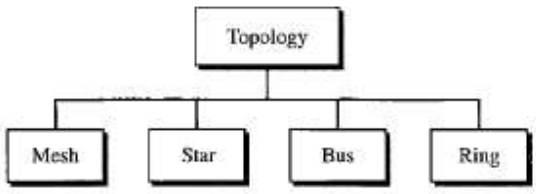


Fig. Categories of Topology

i. ***Mesh Toplogy :*** 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.

To find the number of physical links in a fully connected mesh network with *n* nodes, we first consider that each node must be connected to every other node. Node 1 must be connected to *n* - I nodes, node 2 must be connected to *n* – 1 nodes, and finally node *n* must be connected to *n* - 1 nodes.

In a mesh topology, we need ***n(n -1) /2*** *duplex-mode links*.

To accommodate that many links, every device on the network must have ***n – 1*** *input/output (VO) ports* (see Fig) to be connected to the other *n* - 1 stations.

Usage:

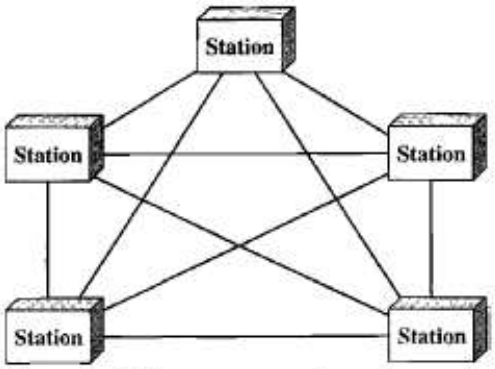


Fig . A fully connected mesh topology

Tthe connection of telephone regional offices in which each regional office needs to be connected to every other regional office.

Advantages:

a. The use of dedicated links guarantees that each connection can carry its own data load, thus eliminating the traffic problems that can occur when links must be shared by multiple devices.
b. A mesh topology is robust. If one link becomes unusable, it does not incapacitate the entire system.
c. There is the advantage of privacy or security. When every message travels along a dedicated line, only the intended recipient sees it. Physical boundaries prevent other users from gaining access to messages.
d. point-to-point links make fault identification and fault isolation easy.

Disadvantages:

a. every device must be connected to every other device, installation and reconnection are difficult.
b. The sheer bulk of the wiring can be greater than the available space (in walls, ceilings, or floors) can accommodate.
a. The hardware required to connect each link (I/O ports and cable) can be prohibitively expensive.

ii. ***Star Topology*** : 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. 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 (see Fig) .

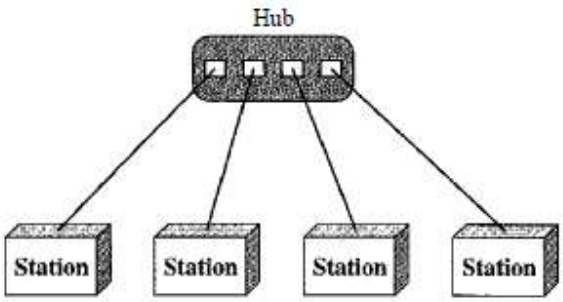


Fig A star topology connecting four stations

Usage:

The star topology is used in local-area networks (LANs). High-speed LANs often use a star topology with a central hub.

Advantages:

a. A star topology is less expensive than a mesh topology.
a. It easy to install and reconfigure. Because each device needs only one link and one I/O port to connect it to any number of others.
b. Less cabling needs to be housed, and additions, moves, and deletions involve only one connection: between that device and the hub.
c. Robustness.
d. To easy fault identification and fault isolation. Because If one link fails, only that link is affected. All other links remain active. As long as the hub is working, it can be used to monitor link problems and bypass defective links.

Disadvantages:

a. The dependency of the whole topology on one single point, the hub. If the hub goes down, the whole system is dead.

b. Although a star requires far less cable than a mesh, each node must be linked to a central hub. For this reason, often more cabling is required in a star than in some other topologies .

iii. ***Bus Topology*** **:** This is a multipoint connection. One long cable acts as a *backbone* to link all the devices in a network (see Fig). 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 that either splices into the main cable or punctures the sheathing of a cable to create a contact with the metallic core. As a signal travels along the backbone, some of its energy is transformed into heat. So there is a limit on the number of taps a bus can support and on the distance between those taps.

Backbone cable can be laid along the most efficient path, then connected to the nodes by drop lines of various lengths.

Usage:

Bus topology was the one of the first topologies used in the design of early local area networks.

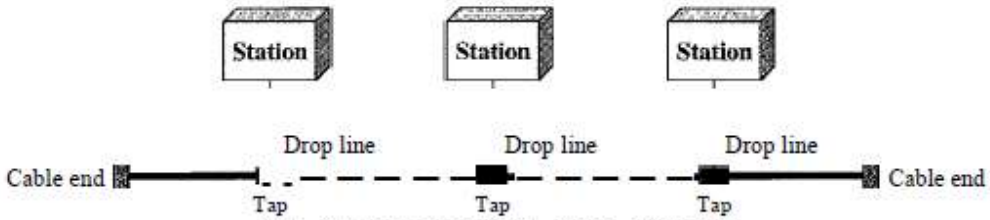


Fig A bus connecting three stations

Advantages:

a. Easy to instal. In this way, a bus uses less cabling than mesh or star topologies.

b. Backbone cable stretches through the entire facility. Each drop line has to reach only as far as the nearest point on the backbone.

Disadvantages:

a. Difficult to reconnect and fault isolation.

b. A bus is usually designed to be optimally efficient at installation. It can therefore be difficult to add new devices.

c. Signal reflection at the taps can cause degradation in quality. This degradation can be controlled by limiting the number and spacing of devices connected to a given length of cable. Adding new devices may therefore require modification or replacement of the backbone.

d. A fault or break in the bus cable stops all transmission, even between devices on the same side of the problem. The damaged area reflects signals back in the direction of origin, creating noise in both directions.

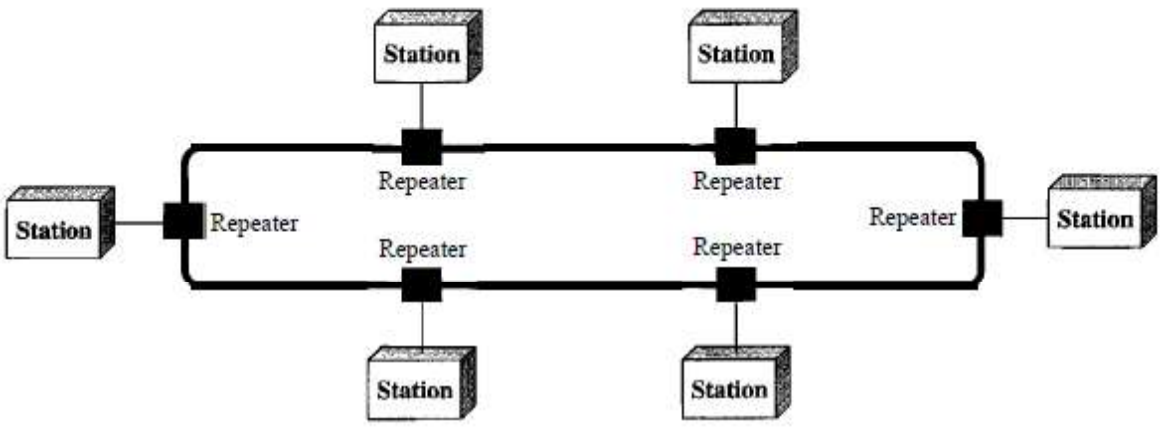


Fig.. A ring connecting six stations

iv. ***Ring Topology*** : In 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. Each device in the ring incorporates a repeater. When a device receives a signal intended for another device, its repeater regenerates the bits and passes them along (see Fig).

Advantages:

a. A ring is relatively easy to install and reconfigure. Each device is linked to only its immediate neighbors (either physically or logically). To add or delete a device requires changing only two connections. The only constraints are media and traffic considerations (maximum ring length and number of devices).

b. Fault isolation is simplified. Generally in a ring, a signal is circulating at all times. If one device does not receive a signal within a specified period, it can issue an alarm. The alarm alerts the network operator to the problem and its location.

Disadvantages:

i. Unidirectional traffic. In a simple ring, a break in the ring (such as a disabled station) can disable the entire network. This weakness can be solved by using a dual ring or a switch capable of closing off the break.

v. **Hybrid Topology** : A network can be hybrid. For example, we can have a main star topology with each branch connecting several stations in a bus topology as shown in Figure.

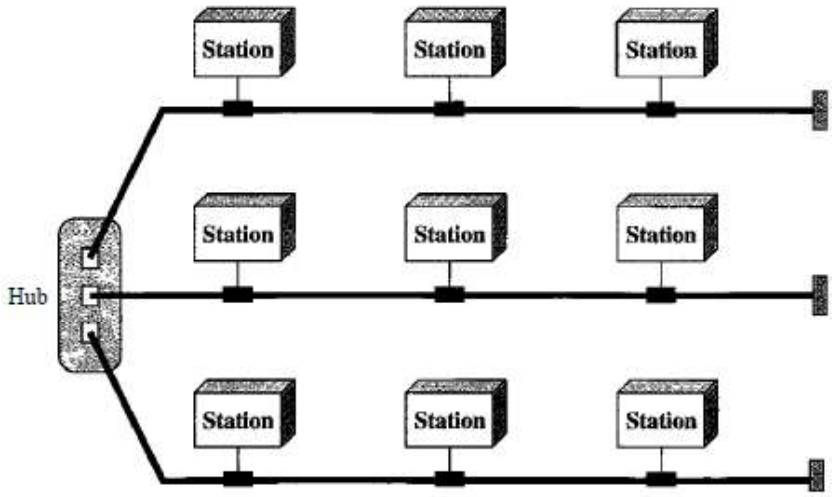


Fig .A hybrid topology : a star backbone with three bus networks

1.1.6. Categories of Networks

The categories are determined by its size. A Two primary categories:

i. Local-Area Networks

ii. Wide-Area Networks

Metropolitan Area Networks : Networks of a size in between are normally referred to as metropolitan area networks and span tens of miles.

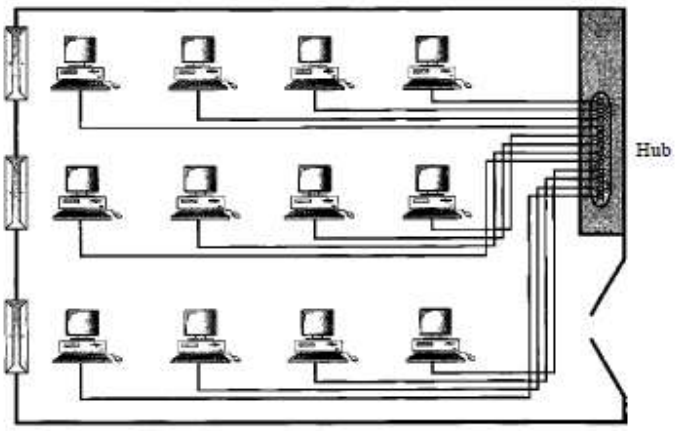


Fig . An isolated LAN connecting 12 computers

i. ***Local Area Network :*** A LAN is privately owned and links the device in a single office. building, or campus (see Fig).Depending on the needs of an organization and the type of technology used, a LAN can be as simple as two PCs and a printer in someone's home office; or it can extend throughout a company and include audio and video peripherals. Currently, LAN size is limited to a few kilometers.

LANs are designed to allow resources to be shared between personal computers or workstations. The resources to be shared can include hardware (e.g., a printer), software (e.g., an application program), or data. One of the computers may be given a largecapacity disk drive and may become a server to clients. Software can be stored on this central server and used as needed by the whole group.

LANs are distinguished from other kinds of networks by three characteristics:

1. Their size
2. Their transmission technology
3. Their topology.

Early LANs : Data rate - 4 to 16 megabits per second (Mbps) range. Speed - 100 or 1000 Mbps

Usage :

A common example of a LAN, found in many business environments, links a workgroup of task-related computers, for

example, engineering workstations or accounting PCs.

ii. ***Wide Area Network :*** It provides long-distance transmission of data, image, audio, and video information over large geographic areas that may comprise a country, a continent, or even the whole world.

Separation of the pure communication aspects of the network (the subnet) from the application aspects (the hosts), greatly simplifies the complete network design.

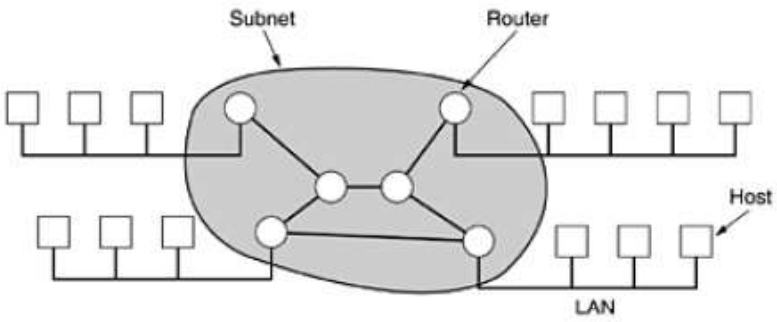


Fig Relation between hosts o LANa and Subnet

In most wide area networks, the subnet consists of two distinct components: *transmission lines* and *switching elements*.

Transmission lines : It move bits between machines. They can be made of copper wire, optical fiber, or even radio links.

Switching elements : These are specialized computers that connect three or more transmission lines.

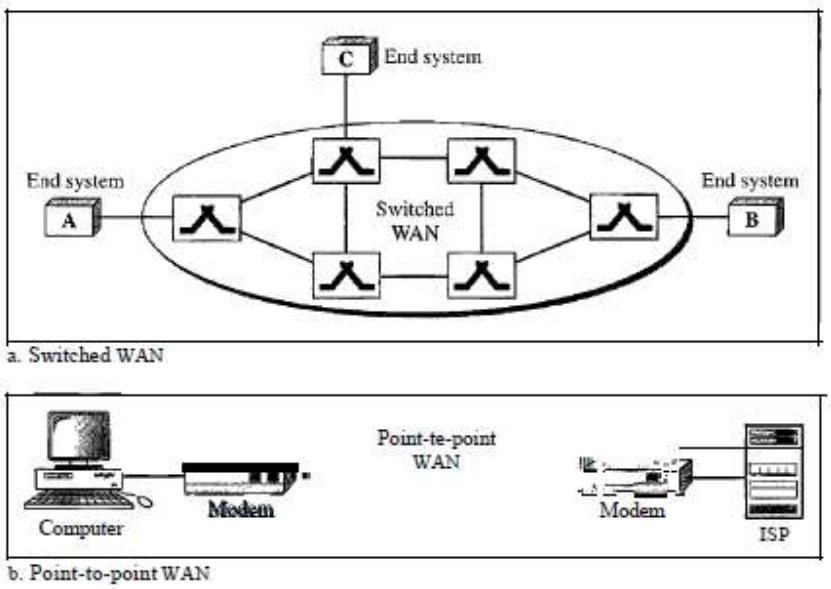


Fig WANs : a switched WAN and Point-to-Point WAN

Router : When data arrive on an incoming line, the switching element must choose an outgoing line on which to forward them. These switching computers have been called by *router.*

From Fig., The *switched WAN* connects the end systems, which usually comprise a router (internetworking connecting device) that connects to another LAN or WAN.

The *point-to-point WAN* is normally a line leased from a telephone or cable TV provider that connects a home computer or a small LAN to an Internet service provider (lSP). This type of WAN is often used to provide Internet access.

When a packet is sent from one router to another via one or more intermediate routers, the packet is received at each intermediate router in its entirety, stored there until the required output line is free, and then forwarded. A subnet organized according to this principle is called a store-and-forward or packet-switched subnet(see Fig.). Nearly all wide area networks (except those using satellites) have store-and-forward subnets. When the packets are small and all the same size, they are often called cells.

In Fig ., all the packets follow the route ACE, rather than ABDE or ACDE. In some networks all packets from a given message must follow the same route; in others each packet is routed separately. Of course, if ACE is the best route, all packets may be sent along it, even if each packet is individually routed.

Usage :

Long-distance transmission of data, image, audio, and video information.

iii. Metropolitan Area Network : A metropolitan area network (MAN) is a network with a size between a LAN and a WAN. It covers a city.

Usage :

The cable television network available in many cities.

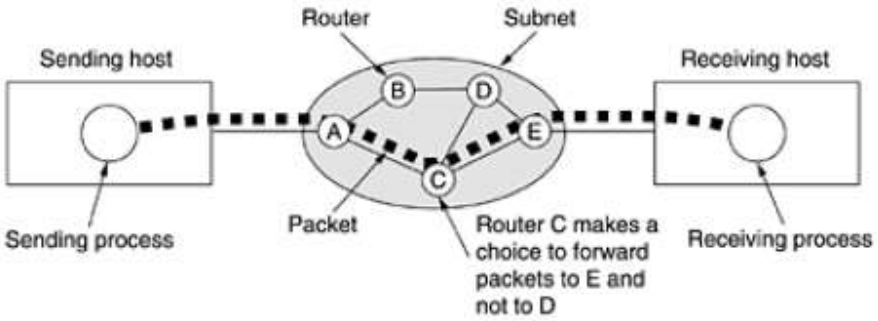


Fig . A stream of packets from sender to receiver

1.1.7. NETWORK ARCHITECTURES

Network designers have developed general blueprints—usually called *network architectures*—that guide the design and implementation of networks.

1.1.7.1. Layering and protocols

a. ***Layering***

- The idea of an abstraction is to define a model that can capture some important aspect of the system, encapsulate this model in an object that provides an interface that can be manipulated by other components of the

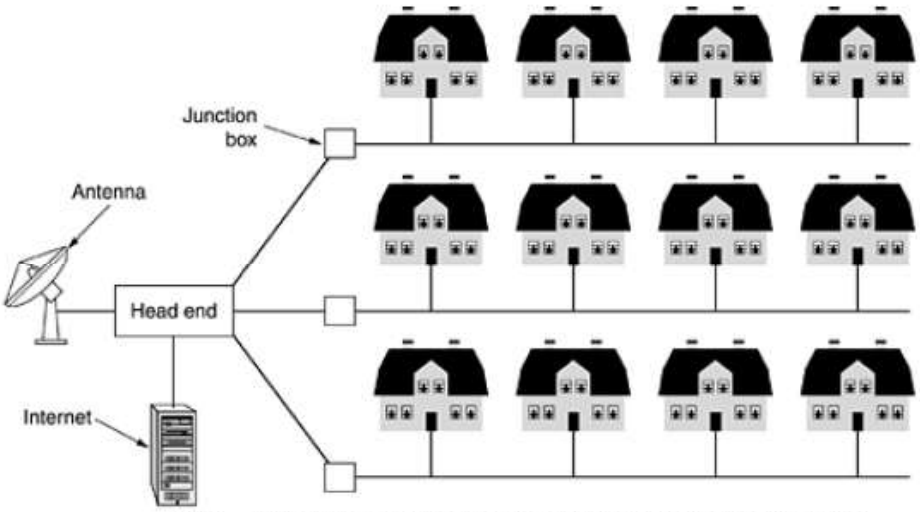


Fig A Metropolitan area network based on cable TV

system, and hide the details of how the object is implemented from the users of the object.

- Abstractions naturally lead to layering. for example, a simple network as having two layers of abstraction sandwiched between the application program and the underlying hardware, as illustrated in Figure 1.7. The layer immediately above the hardware in this case might provide host-to-host connectivity, abstracting away the fact that there may be an arbitrarily complex network topology between any two hosts. The next layer up builds on the available

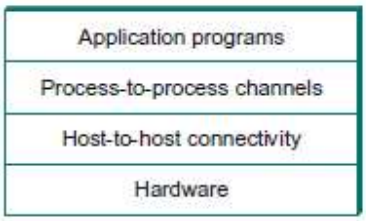
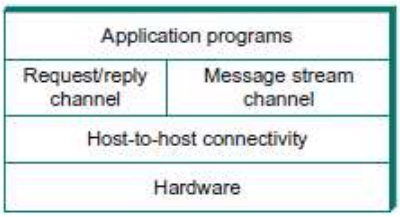


Figure 1.7. Example of Layered Network System

host-to-host communication service and provides support for process-to-process channels.

- Layering provides two features.
  - First, it decomposes the problem of building a network into more manageable components
  - Second, it provides a more modular design.
- Alternative Abstraction is a multilevel networking system, as illustrated in Figure 1.8. consider the two types of channels. One provides *a request/reply service* and one supports *a message stream service*.



Fgure 1.8. Layered system with alternative abstractions available at a given layer

b. Protocols

- A *protocol* provides a communication service that higher-level objects use to exchange messages.
- ***Key elements of Protocol:***
- ✓ **Syntax –** It refers to the structure or format of data meaning the order in which they are presented.
- ✓ **Semantics –** It refers to the meaning of each section of bit. How to do interpretation.
- ✓ **Timing –** When data should be sent and how fast they can be sent.
- Each protocol defines two different interfaces. This situation is illustrated in Figure 1.9.
  - *service interface:* A protocol defines a communication service that it exports locally (the service interface). For example, a request/ reply protocol would support operations by which an application can send and receive messages. An implementation of the HTTP protocol could support an operation to fetch a page of hypertext from a remote server. An application such as a web browser would invoke such an operation whenever the browser needs to obtain a new page.
  - *peer interface :*A set of rules governing the messages that the protocol exchanges with its peer(s) to implement this service (the peer interface). In the case of HTTP, for example, the protocol specification defines in detail how a *GET* command is formatted, what arguments can be used with the command, and how a web server should respond when it receives such a command.

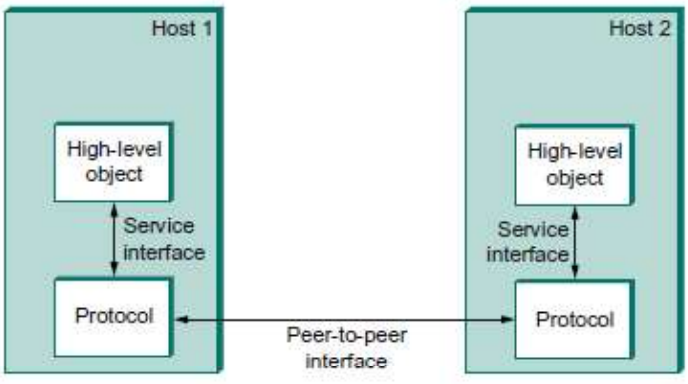


Figure 1. 9 Service Interfaces and peer Interfaces

❖ *protocol graph*

- ✓ Protocols that make up a network system with a *protocol graph*. The nodes of the graph correspond to protocols, and the edges represent a *depends on* relation.
- ✓ For example, Figure 1.10 illustrates a protocol graph for the hypothetical layered system contains the following protocols RRP (Request/Reply Protocol) and MSP (Message Stream Protocol) implement two different types of process-to-process channels, and both depend on the Host-to-Host Protocol (HHP) which provides a host-to-host connectivity service.
- ✓ Suppose that the file access program on host 1 wants to send a message to its peer on host 2 using the communication service offered by RRP. In this case, the file application asks RRP to send the message on its behalf. To communicate with its peer, RRP invokes the services of HHP, which in turn transmits the message to its peer on the other machine. Once the message has arrived at the instance of HHP on host 2, HHP passes the message up to RRP, which in turn delivers the message to the file application. In this particular case, the application is said toemploy the services of the *protocol stack* RRP/HHP.

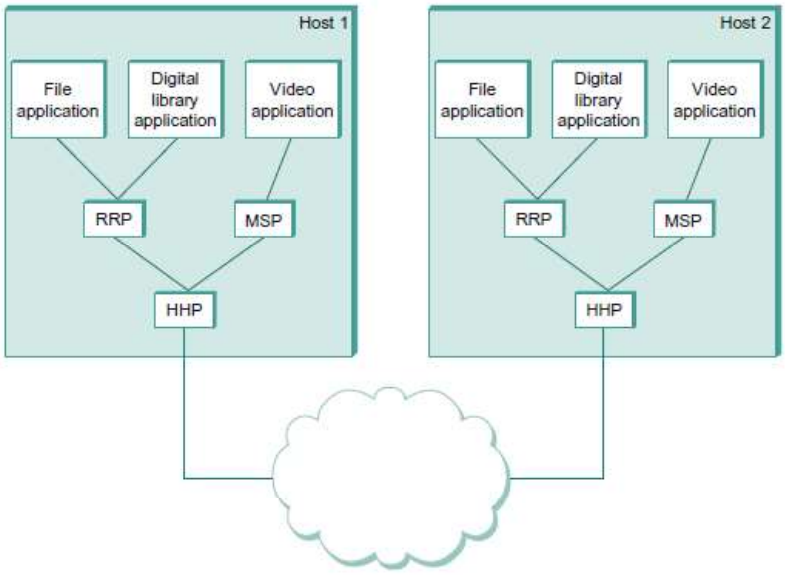


Figure 1.10 Example of a protocol graph

- ✓ *protocol* is used in two different ways.
  - o abstract interfaces - the operations defined by the service interface and the form and meaning of messages exchanged between peers
  - o *module* - that actually implements these two interfaces.
  - ✓ To distinguish between the interfaces and the module that implements these interfaces, we generally refer to the former as a *protocol specification*. Specifications are generally expressed using a combination of prose, pseudocode, state transition diagrams, pictures of packet formats, and other abstract notations.

❖ There are two additional things we need to explain about the mechanics of protocol layering.

1. *Encapsulation*
   - ✓ From Figure 1.10,
     - o RRP must communicate control information to its peer, instructing it how to handle the message when it is received. RRP does this by attaching a *header* to the message.

- *header :* A header is a small data structure from a few bytes to a few dozen bytesthat is used among peers to communicate with each other. Headers are usually attached to the front of a message.
- *trailer* : Peer-to-peer control information is sent at the end of the message, in which case it is called a *trailer*. The exact format for the header attached by RRP is defined by its protocol specification.
- *Data* : The data being transmitted on behalf of the application—is called the message's *body* or *payload*.
- The application data is *encapsulated* in the new message created by RRP. This process of encapsulation is then repeated at each level of the protocol graph.
- For example, HHP encapsulates RRP's message by attaching a header of its own. If we now assume that HHP sends the message to its peer over some network, then when the message arrives at the destination host, it is processed in the opposite order: HHP first interprets the HHP header at the front of the message (i.e., takes whatever action is appropriate given the contents of the header) and passes the body of the message (but not theHHPheader) up to RRP, which takes whatever action is indicated by the RRP header that its peer attached and passes the body of the message (but not the RRP header) up to the application program. The message passed up from RRP to the application on host 2 is exactly the same message as the application passed down to RRP on host 1; the application does not see any of the headers that have been attached to it to implement the lower-level communication services. This whole process is illustrated in Figure 1.11.

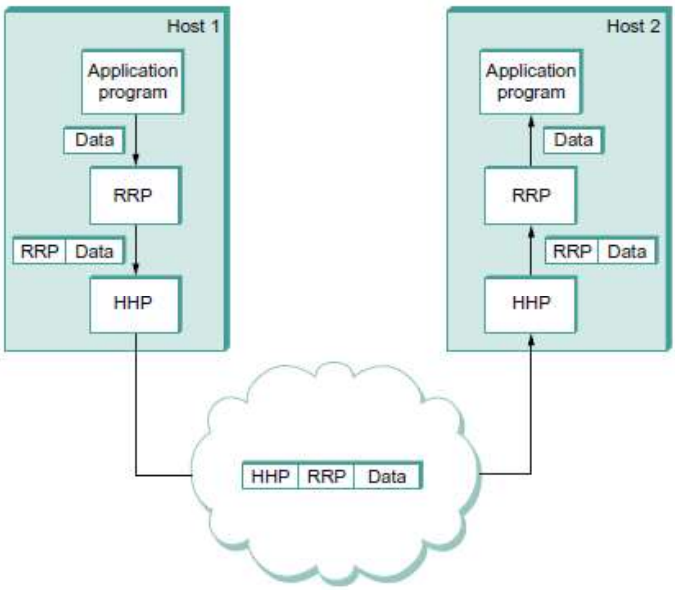


Figure 1.11 High-level messages are encapsulated inside of low-level messages

2. *Multiplexing and Demultiplexing*
   - ✓ Multiplexing : To multiplex multiple flows of data over a single physical link.
   - ✓ In Figure 1.10, for example, RRP as implementing a logical communication channel, with messages from two different applications multiplexed over this channel at the source host and then demultiplexed back to the appropriate application at the destination host.

- ✓ At the source host, RRP includes the appropriate demux key in its header. When the message is delivered to RRP on the destination host, it strips its header, examines the demux key, and demultiplexes the message to the correct application.
- ✓ For example, HHP has its own demux key to determine which messages to pass up to RRP and which to pass up to MSP. Some protocols use an 8-bit field (meaning they can support only 256 high-level protocols), and others use 16- or 32-bit fields. Also, some protocols have a single demultiplexing field in their header, while others have a pair of demultiplexing fields. In the past case, the same demux key is used on both sides of the communication, while in the latter case each side uses a different key to identify the high-level protocol (or application program) to which the message is to be delivered.

1.1.1.7.1. The 7-Layer Model

- ✓ The *Open Systems Interconnection* (OSI) architecture defines a partitioning of network functionality into seven layers, where one or more protocols implement the functionality assigned to a given layer.
- ✓ Fig 1.12 Illustrate OSI 7 layers.

1. ***Physical Layer :*** The physical layer coordinates the functions required to carry a bit stream over a physical medium. The physical layer is also concerned with the following:
   - *i.* ***Physical characteristics of interfaces and medium*** : The physical layer defines the characteristics of the interface between the devices and the transmission medium. It also defines the type of transmission medium.
   - *ii.* ***Representation of bits*** *:* The physical layer data consists of a stream of bits (sequence of Os or 1s) with no interpretation. To be transmitted, bits must been coded into signals--electrical or optical. The physical layer defines the type of encoding (how Os and I s are changed to signals).
   - *iii.* ***Data rate*** *:* The transmission rate-the number of bits sent each second-is also defined by the physical layer. In other words, the physical layer defines the duration of a bit, which is how long it lasts.
   - *iv.* ***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.
   - *v.* ***Line configuration*** *:* The physical layer is concerned with the connection of devices to the media. 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.
   - *vi.* ***Physical topology*** *:* The physical topology defines how devices are connected to make a network. Devices can be connected by using a mesh topology (every device is connected to every other device), a star topology (devices are connected through a central device), a ring topology (each device is connected to the next, forming a ring), a bus topology (every device is on a common link), or a hybrid topology (this is a combination of two or more topologies).
   - *vii.* ***Transmission mode*** *:* The physical layer also defines the direction of transmission between two devices: simplex, half-duplex, or full-duplex. In simplex mode, only one device can send; the other can only receive. The simplex mode is a one-way communication. In the half-duplex mode, two devices can send and receive, but not at the same time. In a full-duplex (or simply duplex) mode, two devices can send and receive at the same time.

2. ***Data Link Layer :*** The data link layer transforms the physical layer, a raw transmission facility, to a reliable link. 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

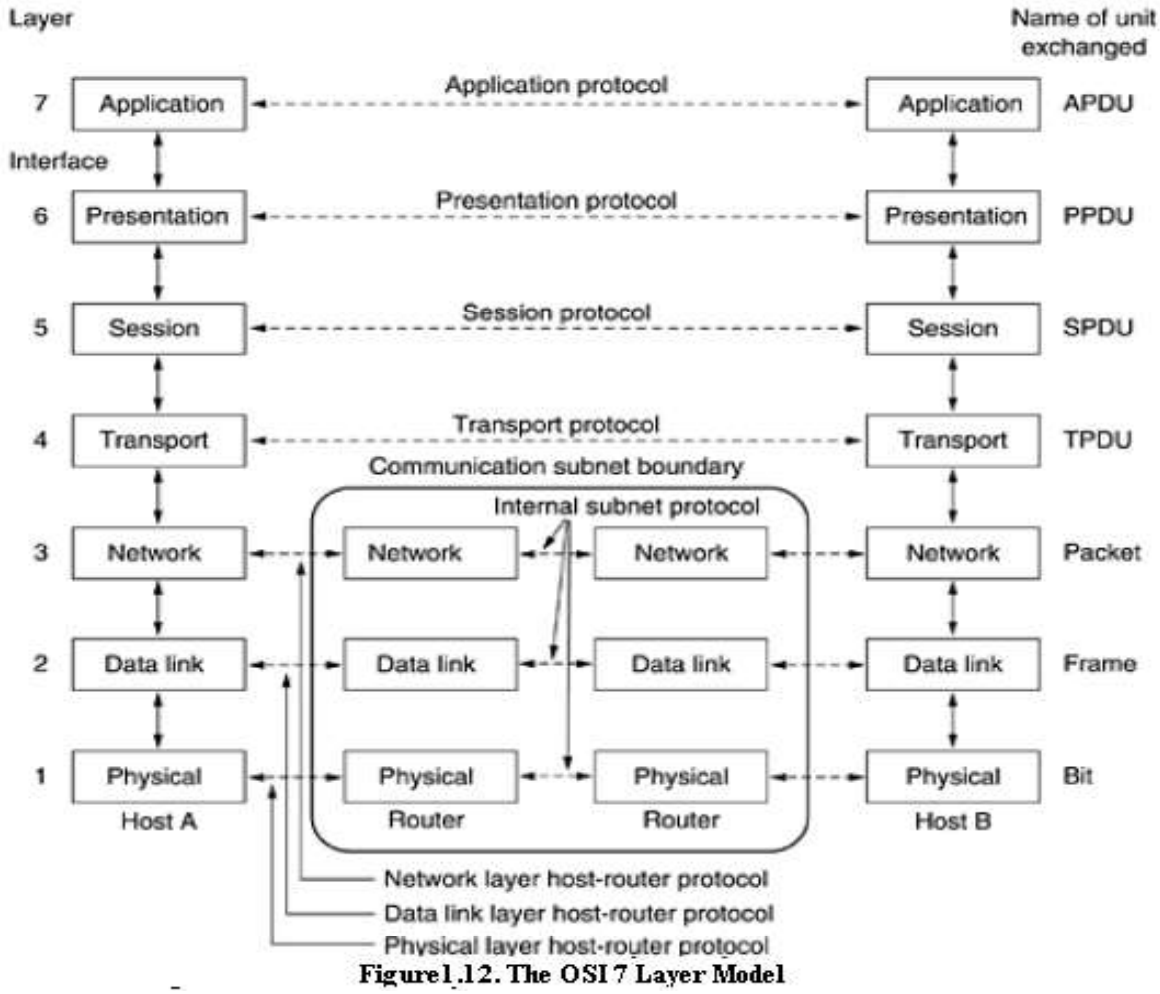


Figure1.12. The OSI 7 Layer Model

necessary to determine which device has control over the link at any given time.

3. ***Network Layer:*** The network layer is responsible for the source-to-destination delivery of a packet, possibly across multiple networks (links). Other responsibilities of the network layer include the following:
   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 that, among other things, includes the logical addresses of the sender and receiver.
   ii. ***Routing :*** When independent networks or links are connected to create *internetworks* (network f networks) or a large network, the connecting devices (called *routers* or *switches)* route or switch the packets to their final destination. One of the functions of the network layer is to provide this mechanism.
4. ***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. Other responsibilities of the transport layer include the following:
   i. ***Service-point addressing*** *:* Computers often run several programs at the same time. For this reason, source-to-destination delivery means delivery not only from one computer to the next but also from a specific process (running program) on one computer to a specific process (running program) on the other. The transport layer header must therefore include a type of address called a *service-point address* (or port address). The network layer gets each packet to the correct computer; the transport layer gets the entire message to the correct process on that computer.
   ii. ***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.
   iii. ***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.
   iv. ***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.
   v. ***Error control*** *:* Like the data link layer, the transport layer is responsible for error control. However, error control at this layer is performed process-to-process rather than across a single link. 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.
5. ***Session Layer :*** The services provided by the first three layers (physical, data link, and network) are not sufficient for some processes. The session layer is the network *dialog controller.* It establishes, maintains, and synchronizes the interaction among communicating systems. The session layer is responsible for dialog control and synchronization.

i. ***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.

ii. ***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 to ensure that each 100-page unit is received and acknowledged independently. In this case, if a crash happens during the transmission of page 523, the only pages that need to be resent after system recovery are pages 501 to 523. Pages previous to 501 need not be resent.

6. ***Presentation Layer :*** The presentation layer is concerned with the syntax and semantics of the information exchanged between two systems. Figure 2.13 shows the relationship between the presentation layer and the application and session layers. The presentation layer is responsible for translation, compression, and encryption.

i. ***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. Because different computers use different encoding systems, the presentation layer is responsible for interoperability between these different encoding methods. The presentation layer at the sender changes the information from its sender-dependent format into a common format. The presentation layer at the receiving machine changes the common format into its receiver-dependent format.

ii. ***Encryption*** *:* To carry sensitive information, a system must be able to ensure privacy. Encryption means that the sender transforms the original information to another form and sends the resulting message out over the network. Decryption reverses the original process to transform the message back to its original form.

iii. ***Compression*** *:* Data compression reduces the number of bits contained in the information. Data compression becomes particularly important in the transmission of multimedia such as text, audio, and video.

7. ***Application Layer :*** The application layer enables the user, whether human or software, to access the network. It provides user interfaces and support for services such as electronic mail, remote file access and transfer, shared database management, and other types of distributed information services. The application layer is responsible for providing services to the user. Some Application services are, *XAOO* (message-handling services), X.500 (directory services), and file transfer, access, and management (FTAM). The user in this example employs *XAOO* to send an e-mail message. Specific services provided by the application layer include the following:

i. ***Network virtual terminal*** : A network virtual terminal is a software version of a physical terminal, and it allows a user to log on to a remote host. To do so, the application creates a software emulation of a terminal at the remote host. The user's computer talks to the software terminal which, in turn, talks to the host, and vice versa. The remote host believes it is communicating with one of its own terminals and allows the user to log on.

ii. ***File transfer, access, and management :*** This application allows a user to access files in a remote host (to make changes or read data), to retrieve files from a remote computer for use in the local computer, and to manage or control files in a remote computer locally.

iii. ***Mail services*** *:* This application provides the basis for e-mail forwarding and storage.

iv. ***Directory services*** *:* This application provides distributed database sources and access for global information about various objects and services.

1.1.7.2. Internet Architecture (TCP/IP Architecture)

✓ The Internet architecture, which is also sometimes called the TCP/IP Architecture

✓ TCPIIP protocol suite is made *of five layers*:

1. *Physical Layer*
2. *data link Layer*
3. *Network Layer*
4. *Transport Layer*
5. *Application Layer*

✓ The *first four layers* provide *physical standards, network interfaces, internetworking, and transport functions* that correspond to the first four layers of the OSI model. The *three topmost layers in the OSI model*, however, are represented in TCP/IP by a single layer called the *application layer* (see Fig 1-33).

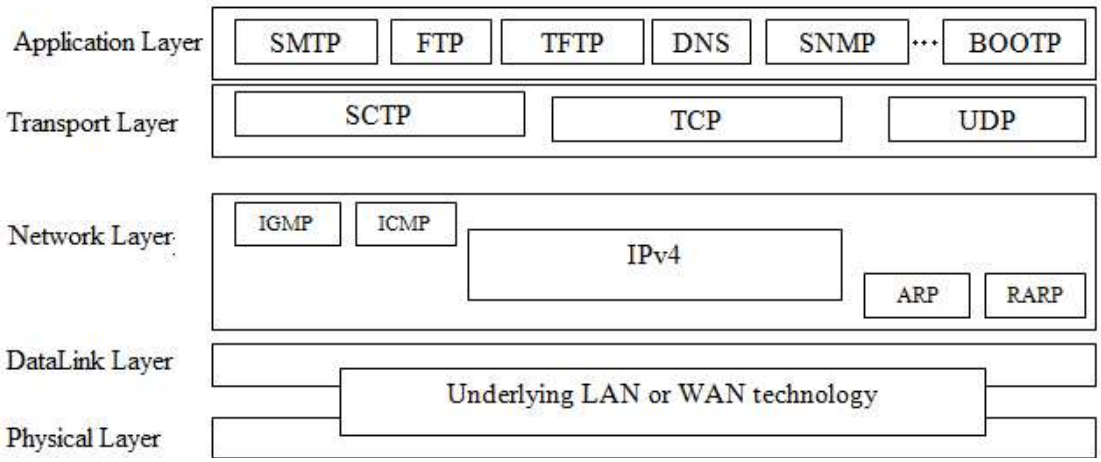


Figure 1.13 Position of IPv4 in TCP/IP protocol suite

Physical and Data Link Layers : At the physical and data link layers, TCP/IP does not define any specific protocol. It supports all the standard and proprietary protocols. A network in a TCP/IP internetwork can be a local-area network or a wide-area network.

Network Layer : The main protocol defined by TCP/IP is the Internetworking Protocol (IP); there are also some other protocols that support data movement in this layer.

✓ ***Internetworking Protocol (IP) :*** TCP/IP supports the Internetworking Protocol. It is an unreliable and connectionless protocol-a best-effort delivery service.

- The term *best effort* means that IP provides no error checking or tracking. IP assumes the unreliability of the underlying layers and does its best to get a transmission through to its destination, but with no guarantees.
- IP transports data in packets called *datagrams,* each of which is transported separately. Datagrams can travel along different routes and can arrive out of sequence or be duplicated. IP does not keep track of the routes and has no facility for reordering datagrams once they arrive at their destination.
- IP uses four supporting protocols:

i. *Address Resolution Protocol (ARP)* : The Address Resolution Protocol (ARP) is used to associate a logical address with a physical address. ARP is used to find the physical address of the node when its Internet address is known. On a typical physical network, such as a LAN, each device on a link is identified by a physical or station address, usually imprinted on the *network interface card (NIC)*.

ii. *Reverse Address Resolution Protocol(RARP):*The Reverse Address Resolution Protocol (RARP) allows a host to discover its Internet address when it knows only its physical address. It is used when a computer is connected to a network for the first time or when a diskless computer is booted.

iii. *Internet Control Message Protocol(ICMP):*The Internet Control Message Protocol (ICMP) is a mechanism used by hosts and gateways to send notification of datagram problems back to the sender. ICMP sends query and error reporting messages.

iv. *Internet Group Message Protocol(IGMP):*The Internet Group Message Protocol (IGMP) is used to facilitate the simultaneous transmission of a message to a group of recipients.

Transport Layer: The transport layer was represented in *TCP/IP* by three protocols:

i. *Transmission Control Protocol(TCP):* The Transmission Control Protocol (TCP) provides full transport-layer services to applications. TCP is a reliable stream transport protocol. The term *stream,* in this context, means connection-oriented: A connection must be established between both ends of a transmission before either can transmit data.

At the sending end of each transmission, TCP divides a stream of data into smaller units called *segments.* Each segment includes a sequence number for reordering after receipt, together with an acknowledgment number for the segments received. Segments are carried across the internet inside of IP datagrams. At the receiving end, TCP collects each datagram as it comes in and reorders the transmission based on sequence numbers.

ii. *User Datagram Protocol* (UDP) : It is a *process-to-process protocol* that adds only port addresses, checksum error control, and length information to the data from the upper layer.

iii. *Stream Control Transmission Protocol (STCP) : It* provides support for newer applications such as voice over the Internet. It is a transport layer protocol that combines the best features of UDP and TCP.

Application Layer : The *application layer* in TCPIIP is equivalent to the combined session, presentation, and application layers in the OSI modeL Many protocols are defined at this layer.

1.1.8. Network software

Application Programming Interface (Sockets)

❖ Network protocols are implemented in software, and nearly all computer systems implement their network protocols as part of the operating system, when we refer to the interface "exported by the network," we are

generally referring to the interface that the OS provides to its networking subsystem. This interface is often called the network *application programming interface* (API).

- The *socket interface* originally provided by the Berkeley distribution of Unix, which is now supported in virtually all popular operating systems, and is the foundation of language-specific interfaces, such as the Java socket library.
- Each protocol provides a certain set of *services*, and the API provides a *syntax* by which those services can be invoked on a particular computer system.
- A socket is as the point where a local application process attaches to the network. The interface defines operations for creating a socket, attaching the socket to the network, sending/receiving messages through the socket, and closing the socket.
- Steps are,
  - ✓ The first step is to create a socket, which is done with the following operation:

```
int socket(int domain, int type, int protocol)
```

    - The *domain* argument specifies the protocol *family* that is going to be used: PF INET denotes the Internet family, PF UNIX denotes the Unix pipe facility, and PF PACKET denotes direct access to the network interface (i.e., it bypasses the TCP/IP protocol stack).
    - The *type* argument indicates the semantics of the communication. SOCK STREAM is used to denote a byte stream. SOCK DGRAM is an alternative that denotes a message-oriented service, such as that provided by UDP.
    - The *protocol* argument identifies the specific protocol that is going to be used. In our case, this argument is UNSPEC because the combination of PF INET and SOCK STREAM implies TCP.
    - Finally, the return value from socket is a *handle* for the newly created socket—that is, an identifier by which we can refer to the socket in the future.
  - ✓ The next step depends on whether you are a client or a server. On a server machine, the application process performs a *passive* open—the server says that it is prepared to accept connections, but it does not actually establish a connection. The server does this by invoking the following three operations:

```
int bind(int socket, struct sockaddr *address, int addr len)
int listen(int socket, int backlog)
int accept(int socket, struct sockaddr *address, int *addr len)
```

    - The bind operation, as its name suggests, binds the newly created socket to the specified address. This is the network address of the *local* participant—the server. The port number is usually some well-known number specific to the service being offered; for example, web servers commonly accept connections on port 80.
    - The listen operation then defines how many connections can be pending on the specified socket. Finally, the accept operation carries out the passive open. It is a blocking operation that does not return until a remote participant has established a connection, and when it does complete it returns a *new* socket that corresponds to this just-established connection, and the address argument contains the *remote* participant's address.
    - When accept returns, the original socket that was given as an argument still exists and still corresponds to the passive open; it is used in future invocations of accept.

- ✓ On the client machine, the application process performs an *active* open; that is, it says who it wants to communicate with by invoking the following single operation:

```
int connect(int socket, struct sockaddr *address, int addr len)
```

  - This operation does not return until TCP has successfully established a connection, at which time the application is free to begin sending data. \
- ✓ Once a connection is established, the application processes invoke the following two operations to send and receive data:

```
int send(int socket, char *message, int msg len, int flags)
int recv(int socket, char *buffer, int buf len, int flags)
```

  - The first operation sends the given message over the specified socket, while the second operation receives a message from the specified socket into the given buffer. Both operations take a set of flags that control certain details of the operation.

Client

- Start with the client side, which takes the name of the remote machine as an argument. It calls the Unix utility gethostbyname to translate this name into the remote host's IP address.
- The next step is to construct the address data structure (sin) expected by the socket interface.
- The final step in setting up the connectionis to call socket and connect. Once the connect operation returns, the connection is established and the client program enters its main loop, which reads text from standard input and sends it over the socket.

```
#include <stdio.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>
#define SERVER_PORT 5432
#define MAX_LINE 256
int main(int argc, char * argv[])
{
   FILE *fp;
   struct hostent *hp;
   struct sockaddr_in sin;
   char *host;
   char buf[MAX_LINE];
   int s;
   int len;
   if (argc==2)
   {
         host = argv[1];
   }
   else
```

```
        {
              fprintf(stderr, "usage: simplex-talk host\n");
              exit(1);
        }/* translate host name into peer's IP address */
        hp = gethostbyname(host);
        if (!hp)
        {
              fprintf(stderr, "simplex-talk: unknown host: %s\n", host);
              exit(1);
        }/* build address data structure */
        bzero((char *)&sin, sizeof(sin));
        sin.sin_family = AF_INET;
        bcopy(hp->h_addr, (char *)&sin.sin_addr, hp->h_length);
        sin.sin_port = htons(SERVER_PORT);/* active open */
        if ((s = socket(PF_INET, SOCK_STREAM, 0)) < 0)
        {
              perror("simplex-talk: socket");
              exit(1);
        }
        if (connect(s, (struct sockaddr *)&sin, sizeof(sin)) < 0)
        {
              perror("simplex-talk: connect");
              close(s);
              exit(1);
        }/* main loop: get and send lines of text */
        while (fgets(buf, sizeof(buf), stdin))
        {
              buf[MAX_LINE-1] = '\0';
              len = strlen(buf) + 1;
              send(s, buf, len, 0);
        }
  }
```

Server

- The server is equally simple. It first constructs the address data structure by filling in its own port number (SERVER PORT).
- Next, the server performs the preliminary steps involved in a passive open; it creates the socket, binds it to the local address, and sets the maximum number of pending connections to be allowed.
- Finally, the main loop waits for a remote host to try to connect, and when one does, it receives and prints out the characters that arrive on the connection.

```
#include <stdio.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>
#define SERVER_PORT 5432
#define MAX_PENDING 5
#define MAX_LINE 256
int main()
{
	struct sockaddr_in sin;
	char buf[MAX_LINE];
	int len;
	int s, new_s;
	/* build address data structure */
	bzero((char *)&sin, sizeof(sin));
	sin.sin_family = AF_INET;
	sin.sin_addr.s_addr = INADDR_ANY;
	sin.sin_port = htons(SERVER_PORT);
	/* setup passive open */
	if ((s = socket(PF_INET, SOCK_STREAM, 0)) < 0)
	{
		perror("simplex-talk: socket");
		exit(1);
	}
	if ((bind(s, (struct sockaddr *)&sin, sizeof(sin))) < 0)
	{
		perror("simplex-talk: bind");
		exit(1);
	}
	listen(s, MAX_PENDING);
	/* wait for connection, then receive and print text */
	while(1) {
	if ((new_s = accept(s, (struct sockaddr *)&sin, &len)) < 0)
	{
		perror("simplex-talk: accept");
		exit(1);
	}
	while (len = recv(new_s, buf, sizeof(buf), 0))
		fputs(buf, stdout);
```

```
        close(new_s);
        }
}
```

1.1.9. Performance

1.1.9.1. Bandwidth and Latency

Network performance is measured in two fundamental ways: *bandwidth*(also called *throughput*) and *latency* (also called *delay*).

Bandwidth

- The bandwidth of a network is given by the number of bits that can be transmitted over the network in a certain period of time. For example, a network might have a bandwidth of 10 million bits/second (Mbps), meaning that it is able to deliver 10 million bits every second. It is sometimes useful to think of bandwidth in terms of how long it takes to transmit each bit of data.
- Ona10-Mbps network, for example, it takes 0.1 microsecond (µs) to transmit each bit.
- For example, on the bandwidth of a single physical link or of a logical process-to-process channel. At the physical level, bandwidth is constantly improving, with no end in sight.
- A second of time as a distance you could measure with a ruler and bandwidth as how many bits fit in that distance, then you can think of each bit as a pulse of some width. For example, each bit on a 1-Mbps link is 1 µs wide, while each bit on a 2-Mbps link is 0.5 µs wide, as illustrated in Figure 1.14.

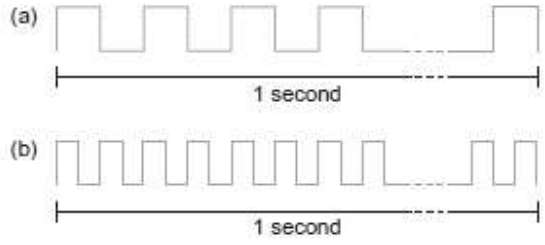


Figure 1.14 Bits transmitted at a particular bandwidth can be regarded as having some width: (a) bits transmitted at 1Mbps(each bits 1µs wide); (b)bitstransmitted at 2Mbps(each bits 0.5µs wide)

Latency

- *Latency,* corresponds to how long it takes a message to travel from one end of a network to the other. (As with bandwidth, we could be focused on the latency of a single link or an end-to-end channel.) Latency is measured strictly in terms of time.
- For example, a transcontinental network might have a latency of 24 milliseconds (ms); that is, it takes a message 24 ms to travel from one coast of North America to the other. There are many situations in which it is more important to know how long it takes to send a message from one end of a network to the other and back, rather than the one-way latency. This is called as *round-trip time* (RTT) of the network.
- latency having three components.
  - ✓ First, there is the speed-of-light propagation delay. This delay occurs because nothing,including a bit on a wire, can travel faster than the speed of light. If you know the distance between two points, you can calculate

the speed-of light latency, although you have to be careful because light travels across different media at different speeds: It travels at 3.0×108 m/s in a vacuum, 2.3×108 m/s in a copper cable, and 2.0×108 m/s in an optical fiber.

- ✓ Second, there is the amount of time it takes to transmit a unit of data. This is a function of the network bandwidth and the size of the packet in which the data is carried.
- ✓ Third, there may be queuing delays inside the network, since packet switches generally need to store packets for some time before forwarding them on an outbound link. So, to define the total latency as

```
Latency = Propagation+Transmit+Queue
Propagation = Distance/SpeedOfLight
Transmit = Size/Bandwidth
```

- where Distance is the length of the wire over which the data will travel,
- SpeedOfLight is the effective speed of light over that wire,
- Size is the size of the packet, and Bandwidth is the bandwidth at which the packet is transmitted.

- For example, a client that sends a 1-byte message to a server and receives a 1-byte message in return is latency bound. Assuming that no serious computation is involved in preparing the response, the application will perform much differently on a transcontinental channel with a 100-ms RTT than it will on an across-the-room channel with a 1-ms RTT. Whether the channel is 1 Mbps or 100 Mbps is relatively insignificant, however, since the former implies that the time to transmit a byte (Transmit) is 8μs and the latter implies Transmit = 0.08 μs.
- Figure 1.15 gives a sense of how latency or bandwidth can dominate performance in different circumstances. The graph shows how long it takes to move objects of various sizes (1 byte, 2 KB, 1 MB) across networks with RTTs ranging from 1 to 100 ms and link speeds of either 1.5 or 10 Mbps. use logarithmic scales to show relative performance. For a 1-byte object (say, a keystroke), latency remains almost exactly equal to the RTT, so that you cannot distinguish between a 1.5-Mbps network and a 10-Mbps network. For a 2-KB object (say, an email message), the link speed makes quite a difference on a 1-ms RTT network but a negligible difference on a 100-ms RTT network. And for a 1-MB object (say, a digital image), the RTT makes no difference—it is the link speed that dominates performance across the full range of RTT.
- When we are referring to the specific amount of time it takes a signal to propagate from one end of a link to another, we use the term *propagation delay*.
- As an aside, computers are becoming so fast that when we connect them to networks, it is sometimes useful to think, at least figuratively, in terms of *instructions per mile*. Consider what happens when a computer that is able to execute 1 billion instructions per second sends a message out on a channel with a 100-ms RTT. (To make the math easier, assume that the message covers a distance of 5000 miles.) If that computer sits idle the full 100 ms waiting for a reply message, then it has forfeited the ability to execute 100 million instructions, or 20,000 instructions per mile. It had better have been worth going over the network to justify this waste.

1.1.9.2. Delay×Bandwidth Product

- It is also useful to talk about the product of these two metrics, often called the *delay ×bandwidth product*.
- A channel between a pair of processes as a hollow pipe (see Figure 1.16), where the latency corresponds to the length of the pipe and the bandwidth gives the diameter of the pipe, then the delay × bandwidth product gives the volume of the pipe—the maximum number of bits that could be in transit through the pipe at any given instant.

Said another way, if latency (measured in time) corresponds to the length of the pipe, then given the width of each bit (also measured in time) you can calculate how many bits fit in the pipe.

- For example, a transcontinental channel with a one-way latency of 50 ms and a bandwidth of 45 Mbps is able to hold 50×10−3 s×45×106 bits/s = 2.25×106 bits or approximately 280 KB of data.
- The

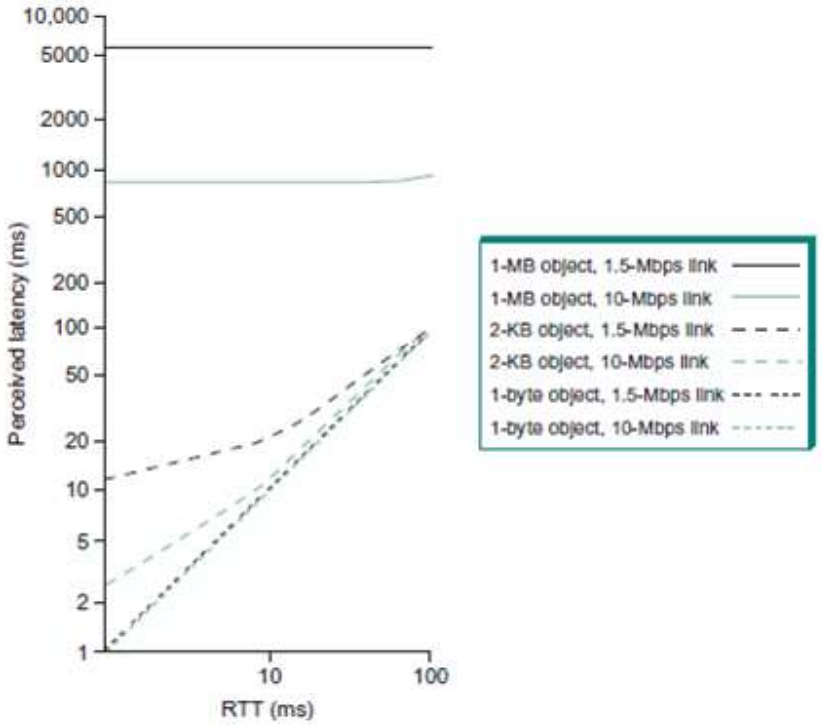


Figure 1.15. Perceived latency(response time) verses round-trip time for various object sizes and link speeds

delay×bandwidth product is important to know when constructing high-performance networks because it corresponds to how many bits the sender must transmit before the first bit arrives at the receiver.

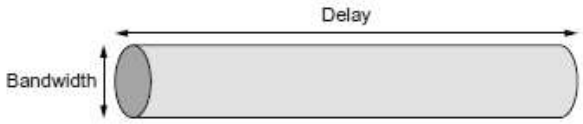


Figure 1.16. Network as a pipe

- If the sender is expecting the receiver to somehow signal that bits are starting to arrive, and it takes another channel latency for this signal to propagate back to the sender, then the sender can send up one RTT×bandwidth worth of data before hearing from the receiver that all is well.
- The bits in the pipe are said to be "in flight," which means that if the receiver tells the sender to stop transmitting it might receive up to one RTT×bandwidth's worth of data before the sender manages to respond.
- Table 1.1 shows some examples of RTT×bandwidth products for some typical network links.

1.1.9.3. High-Speed Networks

- High speed does not mean that latency improves at the same rate as bandwidth; the transcontinental RTT of a 1-Gbps link is the same 100 ms as it is for a 1-Mbps link.

- To appreciate the significance of ever-increasing bandwidth in the face of fixed latency, consider what is required to transmit a 1-MB file over a 1-Mbps network versus over a 1-Gbps network, both of which have an RTT of 100 ms. In the case of the 1-Mbps network, it takes 80 round-trip times to transmit the file; during each RTT, 1.25% of the file is sent.
- Figure 1.17 illustrates the difference between the two networks.
- In effect, the 1-MB file looks like a stream of data that needs to be transmitted across a 1-Mbps network, while it looks like a single packet on a 1-Gbps network.

Table 1.1 Sample Delay × Bandwidth Products

Link type	Bandwidth (typical)	One-way distance (typical)	Round-trip delay	RTT × Bandwidth
Dial-up	56 kbps	10 km	87 μs	5 bits
Wireless LAN	54 Mbps	50 m	0.33 μs	18 bits
Satellite	45 Mbps	35,000 km	230 ms	10 Mb
Cross-country fiber	10 Gbps	4,000 km	40 ms	400 Mb

- The effective end-to-end throughput that can be achieved over a network is given by the simple relationship

```
Throughput = Transfer Size/Transfer Time
```

  - where TransferTime includes not only the elements of one-way Latency identified earlier in this section, but also any additional time spent requesting or setting up the transfer.
- Generally,we represent this relationship as `TransferTime = RTT+1/Bandwidth×TransferSize`

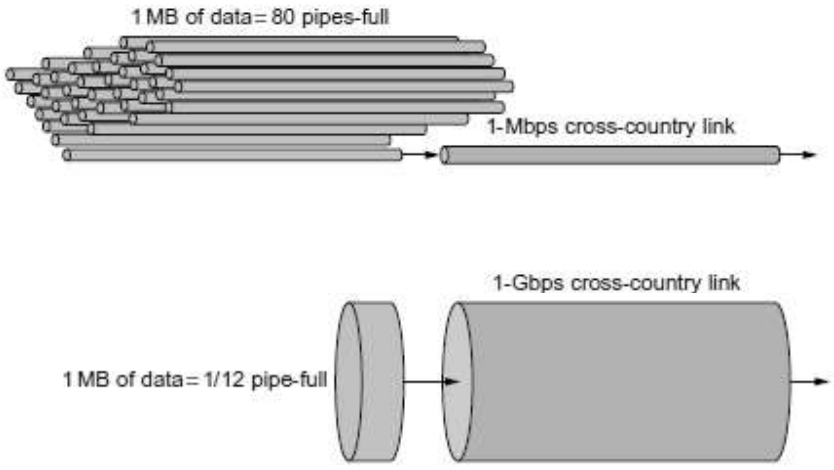


Figure 1.17. Relationship between bandwidth and latency. A 1-MB file would fill the 1-Mbps link 80 times but only fill the 1-Gbps 1/12 of one time

1.1.9.4. Application Performance Needs

- The variation in latency is called *jitter*.
- Consider the situation in which the source sends a packet once every 33 ms, as would be the case for a video application transmitting frames 30 times a second.

- If the packets arrive at the destination spaced out exactly 33 ms apart, then we can deduce that the delay experienced by each packet in the network was exactly the same.
- If the spacing between when packets arrive at the destination—sometimes called the *inter-packet gap*—is variable, however, then the delay experienced by the sequence of packets must have also been variable, and the network is said to have introduced jitter into the packet stream, as shown in Figure 1.18.

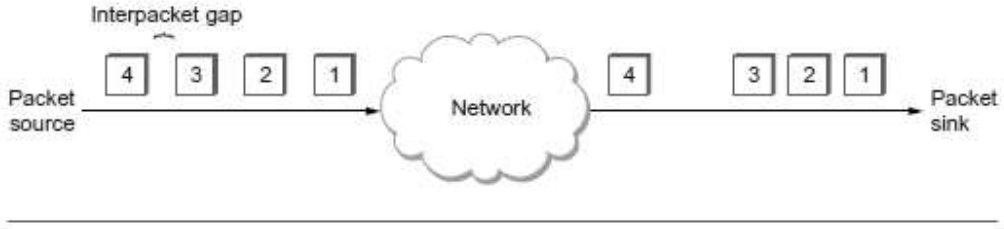


Figure 1.18. Network-induced Jitter

1.2. LINK LAYER

- The two main functions of the data link layer are *data link control* and *media access control*.
  - ✓ The first, data link control, deals with the design and procedures for communication between two adjacent nodes: node-to-node communication.
  - ✓ The second function of the data link layer is media access control, or how to share the link.

1.2.1. Link layer Services

Link layer Services are,

i. Framing
ii. Error Control
iii. Flow Control

1.2.1.1. Framing

- Figure 1.19. illustrate how to transmit a sequence of bits over a point-to-point link from adaptor to adaptor
- packet-switched networks, which means that blocks of data (called *frames* at this level), not bit streams, are exchanged between nodes.
- It is the network adaptor that enables the nodes to exchange frames.

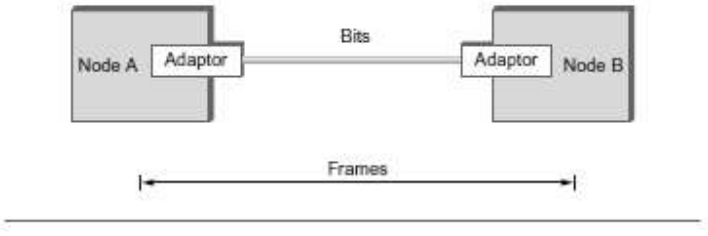


Figure 1.19 Bits flow between adaptors,frames between hosts

- When node A wishes to transmit a frame to node B, it tells its adaptor to transmit a frame from the node's memory. This results in a sequence of bits being sent over the link.
- The adaptor on node B then collects together the sequence of bits arriving on the link and deposits the corresponding frame in B's memory.
- There are several ways to address the framing problem.

b. Byte-Oriented Protocols (BISYNC, PPP, DDCMP)

➢ A *byte-oriented* approach is the Binary Synchronous Communication (BISYNC) protocol developed by IBM in the late 1960s, and the Digital Data Communication Message Protocol (DDCMP) used in Digital Equipment Corporation's DECNET.

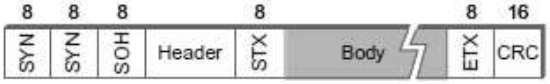


Figure 1.20. BISYNC frame format

Sentinel-Based Approaches

Binary Synchronous Communication(BISYNC)

- BISYNC uses special characters known as *sentinel characters* to indicate where frames start and end.
- Figure 1.20 illustrates the BISYNC protocol's frame format.
  - A packet as a sequence of labeled fields. The packets are transmitted beginning with the leftmost field.
  - The beginning of a frame is denoted by sending a special SYN (synchronization) character.
  - The data portion of the frame is then contained between two more special characters:

a. STX (start of text)

b. ETX (end of text).

  - The SOH (start of header) field serves start of text.
  - The *problem with the sentinel approach*, is that the ETX character might appear in the data portion of the frame.
  - BISYNC overcomes this problem by "escaping" the ETX character by preceding it with a DLE (data-link-escape) character whenever it appears in the body of a frame; the DLE character is alsoescaped (by preceding it with an extra DLE) in the frame body. This approach is often called *character stuffing* because extra characters are inserted in the data portion of the frame.
  - The frame format also includes a field labeled CRC (cyclic redundancy check), which is used to detect transmission errors

Point-to-Point Protocol (PPP)

- Point-to-Point Protocol (PPP), which is commonly used to carry Internet Protocol packets over various sorts of point-topoint links, is similar to BISYNC in that it also uses sentinels and character stuffing.
- The format for a PPP frame is given in Figure 1.21.

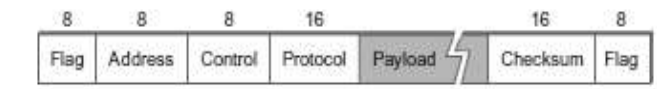


Figure 1.21 PPP frame format

  - The special start-of-text character, denoted as the Flag field in Figure 1.21, is 01111110.
  - The Address and Control fields usually contain default values.

- The Protocol field is used for demultiplexing; it identifies the high-level protocol such as IP or IPX (an IP-like protocol developed by Novell).
- The frame payload size is 1500 bytes by default.
- The Checksum field is either 2 (by default) or 4 bytes long.
- The PPP frame format is unusual in that several of the field sizes are negotiated rather than fixed. This negotiation is conducted by a protocol called the Link Control Protocol (LCP).
- PPP and LCP work in tandem: LCP sends control messages encapsulated in PPP frames such messages are denoted by an LCP identifier in the PPP Protocol field and then turns around and changes PPP's frame format based on the information contained in those control messages.
- LCP is also involved in establishing a link between two peers when both sides detect that communication over the link is possible (e.g., when each optical receiver detects an incoming signal from the fiber to which it connects).

c. Byte-Counting Approach

- In framing the number of bytes contained in a frame can be included as a field in the frame header.
- The DECNET's DDCMP uses this approach, as illustrated in Figure 1.22
  - The COUNT field specifies how many bytes are contained in the frame's body.
  - *One danger with Byte-Counting Approach* is that a transmission error could corrupt the count field, in which case the end of the frame would not be correctly detected.

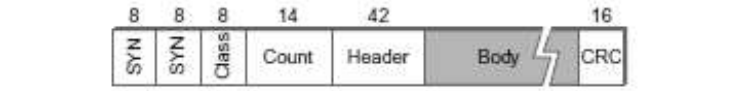


Figure 1.22. DDCMP Frame Format

  - The receiver will accumulate as many bytes as the bad COUNT field indicates and then use the error detection field to determine that the frame is bad. This is sometimes called a *framing error*. The receiver will then wait until it sees the next SYN character to start collecting the bytes that make up the next frame. It is therefore possible that a framing error will cause back-to-back frames to be incorrectly received.

1. Bit-Oriented Protocols (HDLC)

- The frame as a collection of bits. These bits might come from some character set, such as ASCII; they might be pixel values in an image; or they could be instructions and operands from an executable file.
- The Synchronous Data Link Control(SDLC) protocol developed by IBM is an example of a bit-oriented protocol; SDLC was later standardized by the ISO as the High-Level Data Link Control (HDLC) protocol.

Figure 1.23. HDLC Frame Format

- HDLC frame format is given in Figure 1.23.
- HDLC denotes both the beginning and the end of a frame with the distinguished bit sequence 01111110.
- This sequence is also transmitted during any times that the link is idle so that the sender and receiver can keep their clocks synchronized.

- The bits 01111110 might cross byte boundaries—bit-oriented protocols use the analog of the DLE character, a technique known as *bit stuffing*.
- Bit stuffing in the HDLC protocol works as follows.
  - On the sending side, any time five consecutive 1s have been transmitted from the body of the message (i.e., excluding when the sender is trying to transmit the distinguished 01111110 sequence), the sender inserts a 0 before transmitting the next bit.
  - On the receiving side, should five consecutive 1s arrive, the receiver makes its decision based on the next bit it sees (i.e., the bit following the five 1s). If the next bit is a 0, it must have been stuffed, and so the receiver removes it.
  - If the next bit is a 1, then one of two things is true:
    - Either this is the end-of-frame marker or an error has been introduced into the bit stream.
    - By looking at the *next* bit, the receiver can distinguish between these two cases.
      ii. If it sees a 0 (i.e., the last 8 bits it has looked at are 01111110), then it is the end-of-frame marker
      iii. if it sees a 1 (i.e., the last 8 bits it has looked at are 01111111), then there must have been an error and the whole frame is discarded.
  - In the latter case, the receiver has to wait for the next 01111110 before it can start receiving again, and, as a consequence, there is the potential that the receiver will fail to receive two consecutive frames.

d. ClockBased Framing (SONET)

- SONET was first proposed by Bell Communications Research (Bellcore), and then developed under the American National Standards Institute (ANSI) for digital transmission over optical fiber; it has since been adopted by the ITU-T.
- SONET has been for many years the dominant standard for long-distancetransmission of data over optical networks.
- SONET addresses both the *framing problem* and the *encoding problem*. It also addresses a problem that is very important for *phone companies* the multiplexing of several low-speed links onto one high-speed link. (In fact, much of SONET's design reflects the fact that phone companies have to be concerned with multiplexing large numbers of the 64-kbps channels that traditionally are used for telephone calls.)
- In SONET no bit stuffing is used, so that a frame's length does not depend on the data being sent.
- The lowest speed SONET link, which is known as STS-1 and runs at 51.84 Mbps.
- An STS-1 frame is shown in Figure 1.24. It is arranged as 9 rows of 90 bytes each, and the first 3 bytes of each row are overhead, with the rest being available for data that is being transmitted over the link. The first 2 bytes of the frame contain a special bit pattern, and it is these bytes that enable the receiver to determine where the frame starts.

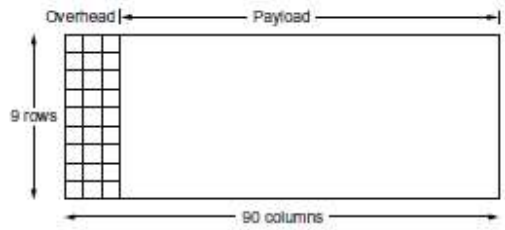


Figure 1.24. A SONET STS-1 frame

- The receiver looks for the special bit pattern consistently, hoping to see it appearing once every 810 bytes, since each frame is 9×90 = 810 bytes long.

- When the special pattern turns up in the right place enough times, the receiver concludes that it is in sync and can then interpret the frame correctly.
- ***complexity of SONET***
  i. The overhead bytes.
  ii. SONET runs across the carrier's optical network, not just over a single link.
  iii. The fact that SONET provides a considerably richer set of services than just data transfer. For example, 64 kbps of a SONET link's capacity is set aside for a voice channel that is used for maintenance.
- The overhead bytes of a SONET frame are encoded using NRZ, where 1s are high and 0s are low. To ensure that there are plenty of transitions to allow the receiver to recover the sender's clock, the payload bytes are *scrambled*. This is done by calculating the exclusive OR (XOR) of the data to be transmitted and by the use of a well-known bit pattern. The bit pattern, which is 127 bits long, has plenty of transitions from 1 to 0, so that XOR ing it with the transmitted data is likely to yield a signal with enough transitions to enable clock recovery.
- SONET supports the multiplexing of multiple low-speed links in the following way.
  - A given SONET link runs at one of a finite set of possible rates, ranging from 51.84 Mbps (STS-1) to 2488.32 Mbps (STS-48), and beyond. All of these rates are integer multiples of STS-1. The significance for framing is that a single SONET frame can contain sub frames for multiple lower-rate channels.
  - A second related feature is that each frame is 125 μs long. This means that at STS-1 rates, a SONET frame is 810 bytes long, while at STS-3 rates, each SONET frame is 2430 bytes long.
  - The synergy between these two features: 3×810 = 2430, meaning that three STS-1 frames fit exactly in a single STS-3 frame.
  - The STS-N frame can be thought of as consisting of N STS-1 frames, where the bytes from these frames are interleaved; that is, a byte from the first frame is transmitted, then a byte from the second frame is transmitted, and so on.
  - The reason for interleaving the bytes from each STS-N frame is to ensure that the bytes in each STS-1 frame are evenly paced; that is, bytes show up at the receiver at a smooth 51 Mbps, rather than all bunched up during one particular 1/Nth of the 125-μs interval.
  - Although it is accurate to view an STS-N signal as being used to multiplex N STS-1 frames, the payload from these STS-1 frames can be linked together to form a larger STS-N payload; such a link is denoted STS-Nc (for *concatenated*). One of the fields in the overhead is used for this purpose.
  - Figure 1.25 schematically depicts concatenation in the case of three STS-1 frames being concatenated into a single STS-3c frame. The significance of a SONET link being designated as STS-3c rather than STS-3 is that, in the former case, the user of the link can view it as a single 155.25- Mbps pipe, whereas an STS-3 should really be viewed as three 51.84-Mbps links that happen to share a fiber.

Figure 1.25. Three STS-1 frame multiplexed onto one STS-3C frame

  - Finally, the preceding description of SONET is overly simplistic in that it assumes that the payload for each frame is completely contained within the frame.

- View the STS-1 frame just described as simply a placeholder for the frame, where the actual payload may *float* across frame boundaries. This situation is illustrated in Figure 1.26. Here we see both the STS-1 payload floating across two STS-1 frames and the payload shifted some number of bytes to the right and, therefore, wrapped around.

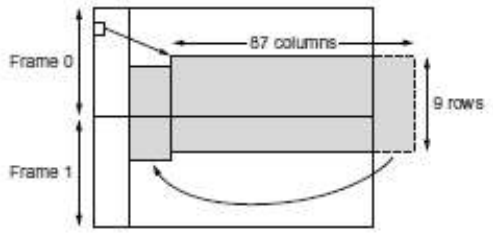


Figure 1.26. SONET frames out of phase

- One of the fields in the frame overhead points to the beginning of the payload. The value of this capability is that it simplifies the task of synchronizing the clocks used throughout the carriers' networks, which is something that carriers spend a lot of their time.

1.2.1.2. ERROR DETECTION

- ***Goal :*** Error detecting codes is to provide a high probability of detecting errors combined with a relatively low number of redundant bits.
- Two basic approaches can be taken when the recipient of a message detects an error.
  - One is to notify the sender that the message was corrupted so that the sender can retransmit a copy of the message. If bit errors are rare, then in all probability the retransmitted copy will be error free.
  - Second, some types of error detection algorithms allow the recipient to reconstruct the correct message even after it has been corrupted; such algorithms rely on *error-correcting codes*.
- One of the most common techniques for detecting transmission errors is a technique known as the *cyclic redundancy check* (CRC).
- Previous methods are,
  - BISYNC protocol when it is transmitting ASCII characters (CRC is used as the error-detecting code when BISYNC is used to transmit EBCDIC3),
  - The basic idea behind any error detection scheme is to add redundant information to a frame that can be used to determine if errors have been introduced.
  - Transmitting two complete copies of the data. If the two copies are identical at the receiver, then it is probably the case that both are correct. If they differ, then an error was introduced into one (or both) of them, and they must be discarded. This is a rather poor error detection scheme for two reasons.
    i. First, it sends n redundant bits for an n-bit message.
    ii. Second, many errors will go undetected any error that happens to corrupt the same bit positions in the first and second copies of the message.
- To provide strong error detection capability while sending only k redundant bits for an n-bit message, where k <<n.
- On an Ethernet, for example, a frame carrying up to 12,000 bits (1500 bytes) of data requires only a 32-bit CRC code, or as it is commonly expressed, uses CRC-32.
- The terminology for the extra bits. they are referred to as *error-detecting codes.*

- Both the sender and the receiver know exactly what that algorithm is. The sender applies the algorithm to the message to generate the redundant bits. It then transmits both the message and those few extra bits.
- When the receiver applies the same algorithm to the received message, it should (in the absence oferrors) come up with the same result as the sender. It compares the result with the one sent to it by the sender. If they match, it can conclude (with high likelihood) that no errors were introduced in the message during transmission.
- If they do not match, it can be sure that either the message or the redundant bits were corrupted, and it must take appropriate action that is, discarding the message or correcting it if that is possible.

- When the algorithm to create the code is based on addition, they may be called a *checksum*. It is an error check that uses a summing algorithm
- Two simpler schemes for error detection
  i. *Two dimensional parity*
  ii. *Checksums*.

a. Two-Dimensional Parity

- It is based on "simple" (one-dimensional) parity, which usually involves adding oneextra bit to a 7-bit code to balance the number of 1s in the byte. For example, odd parity sets the eighth bit to 1 if needed to give an odd number of 1s in the byte, and even parity sets the eighth bit to 1 if needed to givean even number of 1s in the byte.
- Two-dimensional parity does a similar calculation for each bit position across each of the bytes contained in the frame. This results in an extra parity byte for the entire frame, in addition to a parity bit for each byte.
- Figure 1.27 illustrates how two-dimensional even parity works for an example frame containing 6 bytes of data.
- The third bit of the parity byte is 1 since there is an odd number of 1s in the third bit across the 6 bytes in the frame.

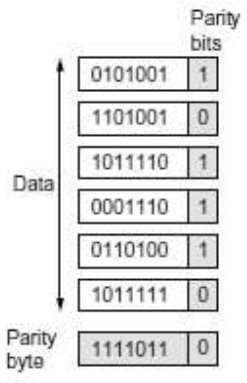


Figure 1.27. Two dimensional Parity

It can be shown that two-dimensional parity catches all 1-, 2-, and 3-bit errors, and most 4-bit errors. In this case, we have added 14 bits of redundant information to a 42-bit message

b. Internet Checksum Algorithm

- The Internet checksum is very simple—you add up all the words that are transmitted and then transmit the result of that sum. The result is the checksum. The receiver performs the same calculation on the received data and compares the result with the received checksum.
- If any transmitted data, including the checksum itself, is corrupted, then the results will not match, so the receiver knows that an error occurred.
- The exact scheme used by the Internet protocols works as follows.
  - Consider the data being check summed as a sequence of 16-bit integers.
  - Add them together using 16-bit ones complement arithmetic and then take the ones complement of the result. That 16-bit number is the checksum.

- In ones complement arithmetic, a negative integer (−x) is represented as the complement of x; that is, each bit of x is inverted.
- When adding numbers in ones complement arithmetic, a carryout from the most significant bit needs to be added to the result.
- Consider, for example, the addition of −5 and −3 in ones complement arithmetic on 4-bit integers: +5 is 0101, so −5 is 1010; +3 is 0011, so −3 is 1100. If we add 1010 and1100, ignoring the carry, we get 0110. In ones complement arithmetic, the fact that this operation caused a carry from the most significant bit causes us to increment the result, giving 0111, which is the ones complement representation of −8 (obtained by inverting the bits in 1000)

➢ The following routine gives a straightforward implementation of the Internet's checksum algorithm.

- The count argument gives the length of buf measured in 16-bit units.
- The routine assumes that buf has already been padded with 0s to a 16-bit boundary.

```
u short cksum(u short *buf, int count)
{
        register u long sum = 0;
        while (count--)
        {
                sum += *buf++;
                if (sum & 0xFFFF0000)
                {
                        /* carry occurred,
                        so wrap around */
                        sum &= 0xFFFF;
                        sum++;
                }
        }
        return (sum & 0xFFFF);
}
```

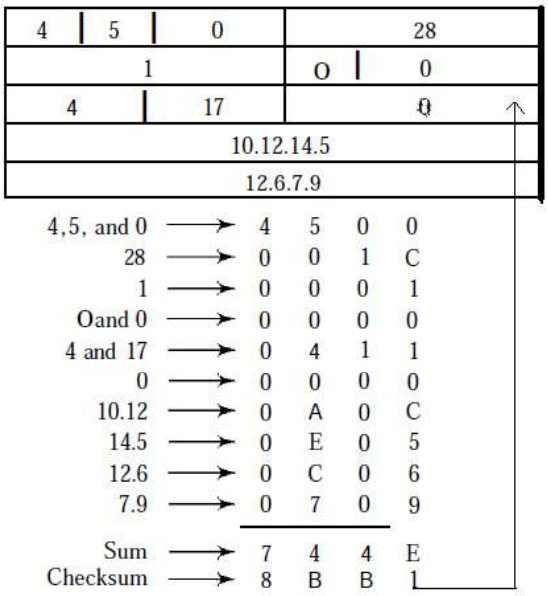


Figure 1.28 *Example of checksum calculation in IPv4*

c. Cyclic Redundancy Check

➢ **Goal:** Error detection algorithms is to maximize the probability of detecting errors using only a small number of redundant bits.

➢ For example, a 32-bit CRC gives strong protection against common bit errors in messages that are thousands of bytes long.

➢ The theoretical foundation of the cyclic redundancy check is rooted in a branch of mathematics called *finite fields*.

➢ To start, think of an (n+1)-bit message as being represented by an n degree polynomial, that is, a polynomial whose highest-order term is x^n. The message is represented by a polynomial by using the value of each bit in the message as the coefficient for each term in the polynomial, starting with the most significant bit to represent the highest-order term.

➢ For example, an 8-bit message consisting of the bits 10011010 corresponds to the polynomial

$$M(x) = 1\times x^7 + 0\times x^6 + 0\times x^5 + 1\times x^4 + 1\times x^3 + 0\times x^2 + 1\times x^1 + 0\times x^0$$
$$= x^7 + x^4 + x^3 + x^1$$

- Here, a sender and a receiver as exchanging polynomials with each other.
- For the purposes of calculating a CRC, a sender and receiver have to agree on a *divisor* polynomial, C(x). C(x) is a polynomial of degree k.
- For example, the Ethernet standard uses a well-known polynomial of degree 32. When a sender wishes to transmit a message M(x) that is n+1 bits long, what is actually sent is the (n+1)-bit message plus k bits.
- Call the complete transmitted message, including the redundant bits, P(x).
- P(x) exactly divisible by C(x) using following procedure;
  i. If P(x) is transmitted over a link and there are no errors introduced during transmission, then the receiver should be able to divide P(x) by C(x) exactly, leaving a remainder of zero.
  ii. Or , if some error is introduced into P(x) during transmission, then in all likelihood the received polynomial will no longer be exactly divisible by C(x), and thus the receiver will obtain a nonzero remainder implying that an error has occurred.
- In polynomial arithmetic, where coefficients may be only one or zero, and operations on the coefficients are performed using modulo 2 arithmetic. This is referred to as "polynomial arithmetic modulo 2."
- The key properties of polynomial arithmetic modulo are,
  - Any polynomial B(x) can be divided by a divisor polynomial C(x) if B(x) is of higher degree than C(x).
  - Any polynomial B(x) can be divided once by a divisor polynomial C(x) if B(x) is of the same degree as C(x).
  - The remainder obtained when B(x) is divided by C(x) is obtained by performing the exclusive OR (XOR) operation on each pair of matching coefficients.
- For example, the polynomial $x^3 + 1$ can be divided by $x^3 + x^2 + 1$ (because they are both of degree 3) and the remainder would be $0\times x^3 + 1\times x^2 + 0\times x^1 + 0\times x^0 = x^2$ (obtained by XORing the coefficients of each term). In terms of messages, we could say that 1001 can be divided by 1101 and leaves a remainder of 0100. You should be able to see that the remainder is just the bitwise exclusive OR of the two messages.
- An example appears below.
  - Recall that we wanted to create a polynomial for transmission that is derived from the original message M(x), is k bits longer than M(x), and is exactly divisible by C(x). The following ways are used.
    1. MultiplyM(x) by xk; that is, add k zeros at the end of the message. Call this zero-extended message T(x).
    2. Divide T(x) by C(x) and find the remainder.
    3. Subtract the remainder from T(x).

    It should be obvious that what is left at this point is a message that is exactly divisible by C(x). We may also note that the resulting message consists of M(x) followed by the remainder obtained in step 2, because when we subtracted the remainder (which can be

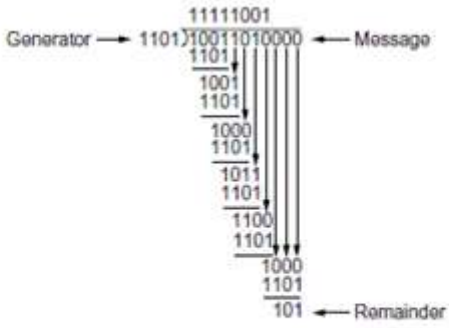


Figure 1.29. CRC calculation using polynomial long division

no more than k bits long), we were just XOR ing it with the k zeros added in step 1.

- An example, Consider the message $x^7 + x^4 + x^3 + x^1$, *or 10011010*.
  - It begin by multiplying by x3, since our divisor polynomial is of degree 3. This gives 10011010000.
  - divide this by C(x), which corresponds to 1101 in this case. Figure 1.29 shows the polynomial long-division operation.
    - The first step of our example,we see that the divisor 1101 divides once into the first four bits of the message (1001), since they are of the same degree, and leaves a remainder of 100 (1101 XOR 1001).
    - The next step is to bring down a digit fromthe message polynomial untilwe get another polynomial with the same degree as C(x), in this case 1001. We calculate the remainder again (100) and continue until the calculation is complete.
    - the very bottom of Figure 1.29 that the remainder of the example calculation is 101. So we know that 10011010000 minus 101 would be exactly divisible by C(x), and this is what we send. The minus operation in polynomial arithmetic is the logical XOR operation, so we actually send 10011010101
  - The recipient divides the received polynomial by C(x) and, if the result is 0, concludes that there were no errors. If the result is nonzero, it may be necessary to discard the corrupted message; with some codes, it may be possible to *correct* a small error (e.g., if the error affected only one bit).
- A code that enables error correction is called an *error-correcting code* (ECC)
  - If the transmitted message is P(x), we may think of the introduction of errors as the addition of another polynomial E(x), so the recipient sees P(x)+E(x).
- One common type of error is a single-bit error, which can be expressedas E(x) = x*i* when it affects bit position i. If we select C(x) such that the first and the last term (that is, the xk and x0 terms) are nonzero, then we already have a two-term polynomial that cannot divide evenly into the one term E(x). Such a C(x) can, therefore, detect all single-bit errors.
  - In general, it is possible to prove that the following types of errors can be detected by a C(x) with the stated properties:
    - ✓ All single-bit errors, as long as the xk and x0 terms have nonzero coefficients
    - ✓ All double-bit errors, as long as C(x) has a factor with at least three terms
    - ✓ Any odd number of errors, as long as C(x) contains the factor (x+1)
- Any "burst" error (i.e., sequence of consecutive errored bits) for which the length of the burst is less than k bits
- Six versions of C(x) are widely used in link-level protocols (shown in Table 1.2).
- For example, Ethernet uses CRC-32, while HDLC uses CRCCCITT. ATM, uses CRC-8, CRC-10, and CRC-32.
- Finally, we note that the CRC algorithm, while seemingly complex, is easily implemented in hardware using a k-bit shift register and XOR gates. The number of bits in the shift register equals the degree of the generator polynomial (k).
- Figure 1.30 shows the hardware that would be used for

CRC	$C(x)$
CRC-8	$x^8+x^2+x^1+1$
CRC-10	$x^{10}+x^9+x^5+x^4+x^1+1$
CRC-12	$x^{12}+x^{11}+x^3+x^2+x+1$
CRC-16	$x^{16}+x^{15}+x^2+1$
CRC-CCITT	$x^{16}+x^{12}+x^5+1$
CRC-32	$x^{32}+x^{26}+x^{23}+x^{22}+x^{16}+x^{12}+x^{11}$ $+x^{10}+x^8+x^7+x^5+x^4+x^2+x+1$

Table 1.2 Common CRC Polynomials

the generator $x^3 + x^2 + 1$ from our previous example. The message is shifted in from the left, beginning with the most significant bit and ending with the string of k zeros that is attached to the message, just as in the long division example.

- When all the bits have been shifted in and appropriately XORed, the register contains the remainder—that is, the CRC (most significant bit on the right).
- The position of the XOR gates is determined as follows:
  - ✓ If the bits in the shift register are labeled 0 through k −1, left to right, then put an XOR gate in front of bit n
  - ✓ if there is a termxn in the generator polynomial. Thus, an XOR gate in front of positions 0 and 2 for the generator x3 +x2 +x0.

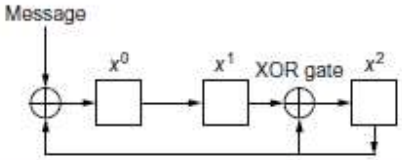


Figure 1.30. CRC calculation using shift register

1.2.1.3. FLOW CONTROL

- Flow control refers to a set of procedures used to restrict the amount of data that the sender can send before waiting for acknowledgment.
- Two fundamental mechanisms of flow control are, acknowledgments and timeouts.
- An acknowledgment (ACK for short) is a small control frame that a protocol sends back to its peer saying that it has received an earlier frame. The receipt of an acknowledgment indicates to the sender of the original frame that its frame was successfully delivered.
- If the sender does not receive an acknowledgment after a reasonable amount of time, then it retransmits the original frame. This action of waiting a reasonable amount of time is called a timeout.
- The general strategy of using acknowledgments and timeouts to implement reliable delivery is sometimes called automatic repeat request (normally abbreviated ARQ).
- The following topics describes the flow control mechanisms.

1. Stop-and-Wait Automatic Repeat Request

- 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. The ACK frame for this protocol has a sequence number field. In this protocol, the sender simply discards a corrupted ACK frame or ignores an out-of-order one.
- **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.
  - ✓ For example, if we decide that the field is m bits long, the sequence numbers start from 0, go to 2m - 1, and then are repeated.
  - ✓ Assume we have used x as a sequence number; we only need to use x + 1 after that. There is no need for x + 2.
  - ✓ To show this, assume that the sender has sent the frame numbered x. Three things can happen.
    1. 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.

2. 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.
3. The frame is corrupted or never arrives at the receiver site; the sender resends the frame (numbered x) after the time-out.

The sequence numbers of stop and wait is 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

Figure 1.32 shows the design of the Stop-and-Wait ARQ Protocol. The sending device keeps a copy of the last frame

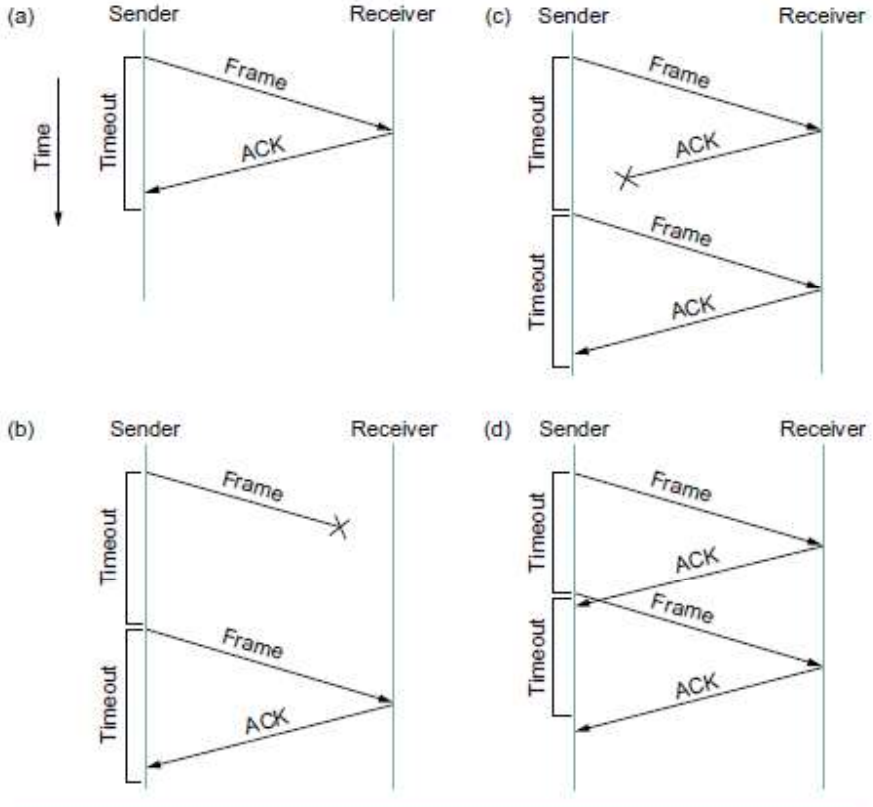


Figure 1.31. Timeline showing four different scenarios for the stop-and-wait algorithm. (a) The ACK is received before the timer expires; (b) the original frame is lost; (c) the ACK is lost; (d) the timeout fires too soon

transmitted until it receives an acknowledgment for that frame. A data frames uses a seqNo (sequence number); an

ACK frame uses an ackNo (acknowledgment number). The sender has a control variable, which we call S_n (sender, next frame to send), that holds the sequence number for the next frame to be sent (0 or 1).

Figure 1.32 *Design of the Stop-and-WaitARQ Protocol*

Design of the Stop-and-WaitARQ Protocol Example : Flow diagram for Stop-and-Wait ARQ

The receiver has a control variable, which we call R_n (receiver, next frame expected), that holds the number of the next frame

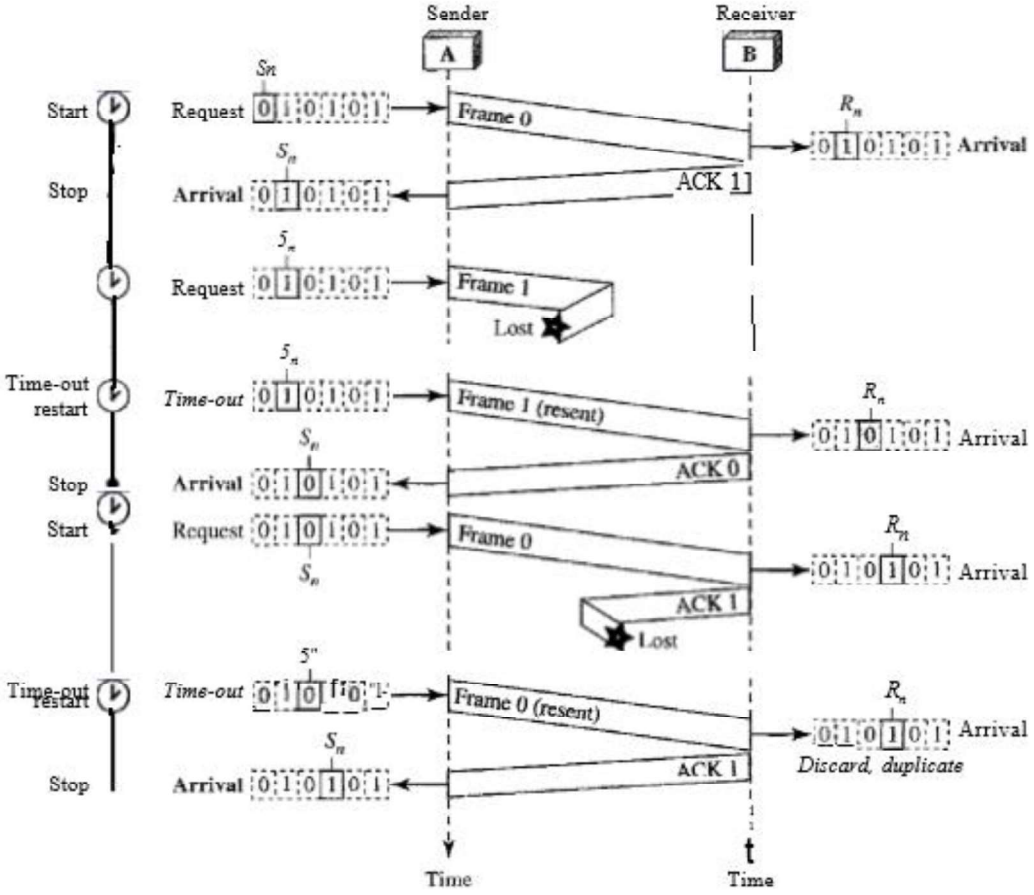


Figure 1.33 . *Flow diagram*

expected. When a frame is sent, the value of S_n is incremented (modulo-2), which means if it is 0, it becomes 1 and vice versa. When a frame is received, the value of R_n is incremented (modulo-2), which means if it is 0, it becomes 1 and vice versa. Three events can happen at the sender site; one event can happen at the receiver site. Variable Sn points to the slot that matches the sequence number of the frame that has been sent, but not acknowledged; Rn points to the slot that matches the sequence number of the expected frame.

Algorithm

Sender-site algorithm for Stop-and-Wait ARQ

```
Sn=0;                          //Frame 0 should be sent first
canSend = true                 //Allow the first frame to go
while(true)                    //Repeat forever
{
       WaitForEvent();         //Sleep until an event occurs
       if (Event (RequestToSend) AND canSend)
      {
            GetData();
            MakeFrame(Sn);     //The seqNo is Sn
            StoreFrame(Sn);    //Keep copy
            SendFrame(Sn);     //Send the data frame
            StartTimer();
            Sn= Sn + 1;
            canSend = false;   //cannot send until ACK arrives
      }
      WaitForEvent();          //Sleep until an event occurs
      if(Event(ArrivalNotification)//An ACK has arrived
      {
            ReceiveFrame(ackNo);           //Receive the ACK frame
            if(not corrupted AND ackNo == Sn)  //valid ACK
            {
                  StopTimer();
                  PurgeFrame(Sn-1);        //Copy is not needed
                  canSend=true;
            }
      }
      if(Event(TimeOut))                   //The timer Expired
      {
            StartTimer();
            ResendFrame(Sn-1);             //Resend a copy check
      }
}
```

Receiver-site algorithm for Stop-and-Wait Protocol

```
Rn=0;                    //Frame 0 expected to arrive first
while(true)                    //Repeat forever
{
      WaitForEvent();          //Sleep      until      an      event      occurs
      if(Event(ArrivalNotification))//Data frame arrived
      {
            ReceiveFrame();
            if(corrupted(frame))
                  sleep();
            if(seqNo== Rn)                  //valid data frame
            {
                  ExtractData();
            DeliverData();//Deliver data to network layer
                  Rn = Rn +1;
            }
            SendFrame();                    //Send an ACK
      }
}
```

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 is only 1000/20,000, or 5 percent. For this reason, for a link with a high bandwidth or long delay, the use of Stop-and-Wait ARQ wastes the capacity of the link.

Example

What is the utilization percentage of the link in Example 11.4 if we have a protocol that can send up to 15 frames before stopping and worrying about the acknowledgments?

Solution The bandwidth-delay product is still 20,000 bits. The system can send up to 15 frames or 15,000 bits during a round trip. This means the utilization is 15,000/20,000, or 75 percent. Of course, if there are damaged frames, the utilization percentage is much less because frames have to be resent.

Pipelining

A task is often begun before the previous task has ended. This is known as pipelining. There is no pipelining in Stop-and-Wait ARQ because we need to wait for a frame to reach the destination and be acknowledged before the next frame can be sent.

Pipelining *does apply to our next two protocols because several frames can be sent before we receive news about the previous frames. Pipelining improves the efficiency of the transmission if the number of bits in transition is large with respect to the bandwidth-delay product.*

Sliding Window

- In the sliding window, the sender and receiver need to deal with only part of the possible sequence numbers.
- The range which is the concern of the sender is called the *send sliding window*; the range that is the concern of the receiver is called the *receive sliding window.*

send sliding window

- The *send window* is an imaginary box covering the sequence numbers of the data frames which can be in transit. In each window position, some of these sequence numbers define the frames that have been sent; others define those that can be sent.
- Figure 1.34 shows a sliding window of size 15 (m =4). The window at any time divides the possible sequence numbers into four regions.
  - ✓ 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.
  - ✓ The second region, colored in Figure 1.34, 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. These are outstanding frames.
  - ✓ The third range, white in the Figure 1.34, 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.
  - ✓ Finally, the fourth region defines sequence numbers that cannot be used until the window slides

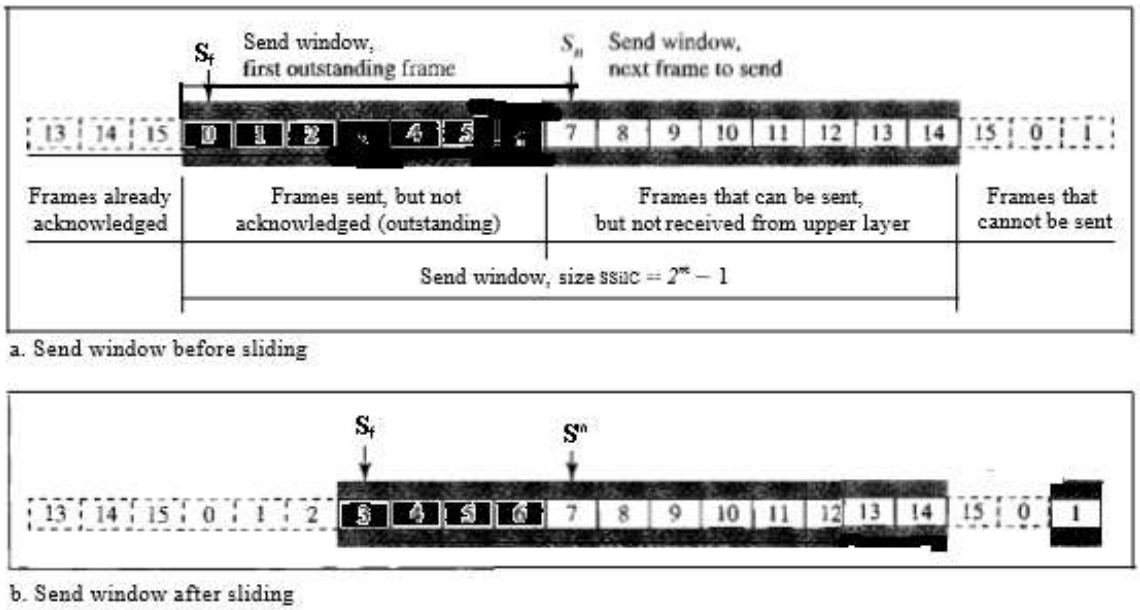


Figure 1.34 *Send window for Go-Back-NARQ*

- Figure 1.34a The variables S_f- send window, the first outstanding frame,

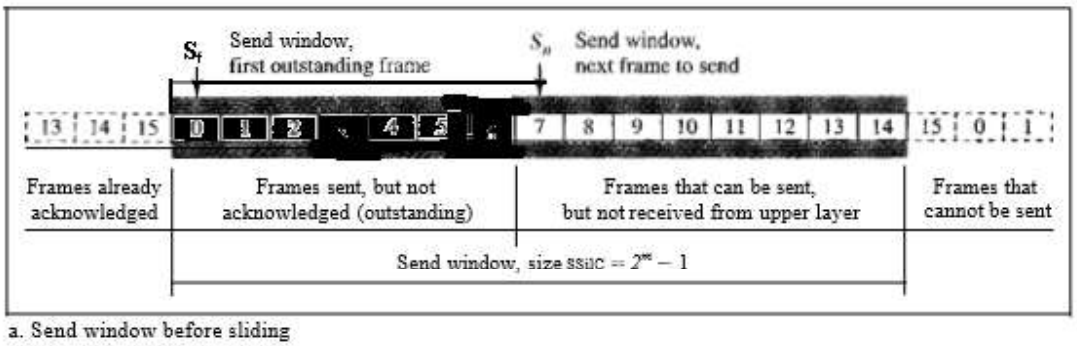
S_n (send window, the next frame to be sent)

S_{size} (send window, size).

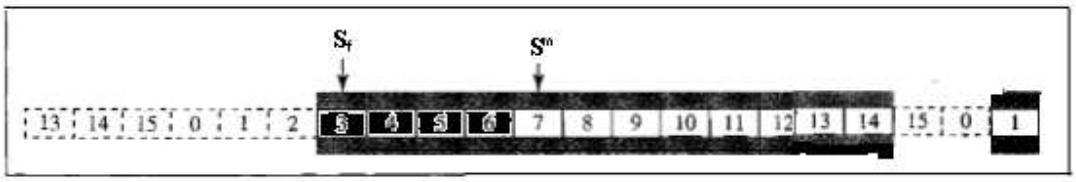
- Figure 1.34b shows how a send window can slide one or more slots to the right when an acknowledgment arrives from the other end. In Figure 1.34b, frames 0, 1, and 2 are acknowledged, so the window has slid to the right three slots. Note that the value of S_f is 3 because frame 3 is now the first outstanding frame.

Receive sliding window

- 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. Figure 1.35 shows the receive window.
- The receive window is an abstract concept defining an imaginary box of size 1 with one single variable R_n. The window slides when a correct frame has arrived; sliding occurs one slot at a time.



a. Send window before sliding



b. Send window after sliding

Figure 1.35. *Send window for Go-Back-NARQ*

- Variable R_n receive window, next frame expected

2. 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 this protocol we can send several frames before receiving acknowledgments; we keep a copy of these frames until the acknowledgments arrive.

Sequence Numbers

- Frames from a sending station are numbered sequentially. 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 modulo1, where m is the size of the sequence number field in bits.

Acknowledgment

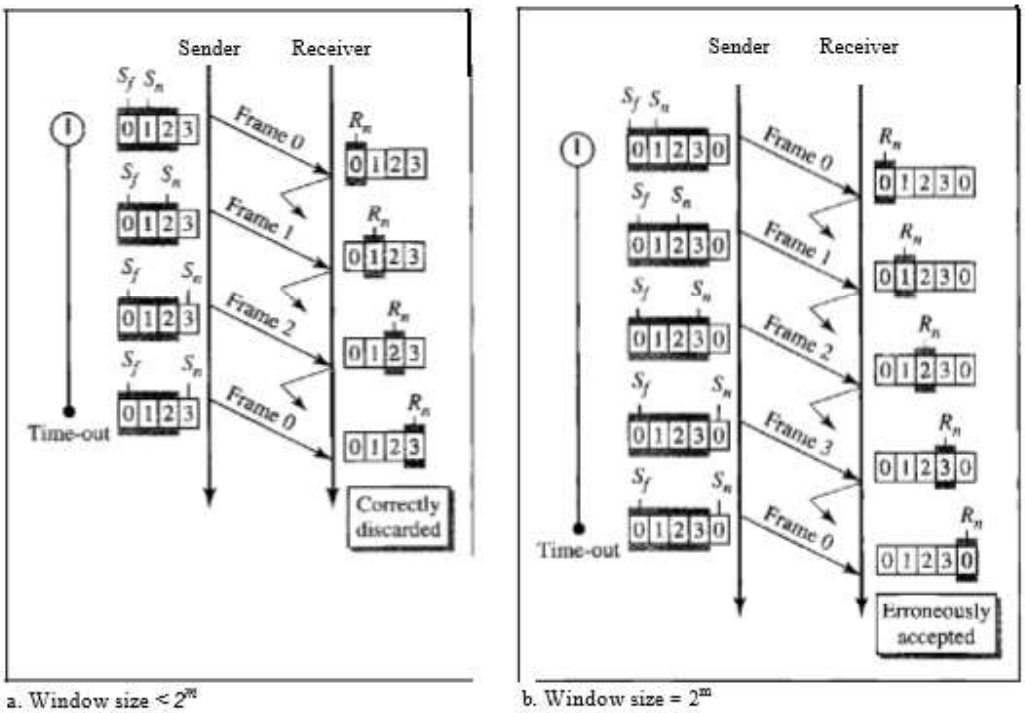
- If a frame is damaged or is received out of order, the receiver is silent and will discard all subsequent frames until it receives the one it is expecting.

Resending a Frame

- When the timer expires, the sender resends all outstanding frames. For example, suppose the sender has already sent frame 6, but the timer for frame 3 expires.
- This means that frame 3 has not been acknowledged; the sender goes back and sends frames 3, 4, 5, and 6 again. That is why the protocol is called *Go-Back-N-ARQ*.

Design

- The multiple frames can be in transit in the forward direction, and multiple acknowledgments in the reverse direction. The idea is similar to Stop-and-Wait ARQ; the difference is that the send window allows us to have as many frames in transition as there are slots in the send window.
- The size of the send window must be less than 2^m. As an example, we choose m=2, which means the size of the window can be $2^m - 1$, or 3.
- Figure 1.36 compares a window size of 3 against a window size of 4.
  - ✓ If the size of the window is 3 (less than 22) and all three acknowledgments are lost, the frame timer expires and all three frames are resent. The receiver is now expecting frame 3, not frame 0, so the duplicate frame is correctly discarded.
  - ✓ On the other hand, if the size of the window is 4 (equal to 22) and all acknowledgments are lost, the sender



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

Figure 1.36 *Window sizefor Go-Back-NARQ*

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.

Algorithm

Go-Back-N Sender algorithm

```
Sn=2^m-1;
Sf=0;
Sn=0;
while(true)                          //Repeat forever
{
       WaitForEvent();               //Sleep until an event occurs
       if(Event(RequestToSend))      //A packet to send
      {
            if(Sn - Sf >= Sw)        //If window is full
                  sleep();
            GetData();
            MakeFrame(Sn);           //The seqNo is Sn
            StoreFrame(Sn);          //Keep copy
            SendFrame(Sn);           //Send the data frame
            Sn= Sn + 1;
            if(timer is not running)
                  StartTimer();
      }
      if(Event(ArrivalNotification) //An ACK arrives
      {
            Receive(ACK);                 //Receive the ACK frame
            if(corrupted(ACK)
                  sleep();
            if(ackNo> Sf)&&( ackNo> Sn)) //if a valid ACK
                  while(Sf <= ackNo)
                  {
                        PurgeFrame(Sf);
                        Sf= Sf + 1;
                        StopTimer();
                  }
      }
      if(Event(TimeOut))                  //The timer Expired
      {
            StartTimer();
            Temp= Sf;
```

```
            While(Temp<Sn)
            {
                  SendFrame(Sf);
                  Sf = Sf+1
            }
      }
}
```

Receiver-site algorithm for Go-Back-N Protocol

```
Rn=0;                                  //Frame 0 expected to arrive first
while(true)                            //Repeat forever
{
      WaitForEvent();                  //Sleep     until     an     event     occurs
      if(Event(ArrivalNotification))//Data frame arrived
      {
            ReceiveFrame();
            if(corrupted(frame))
                  sleep();
            if(seqNo== Rn)                  //valid data frame
            {
                  DeliverData();            //Deliver data to network layer
                  Rn = Rn +1;               //Slide window
                  SendAck(Rn);
            }
      }
}
```

Example

Go-Back-NARQ versus Stop-and-Wait ARQ

- The Stop-and Wait ARQ Protocol is actually a Go-Back-NARQ in which there are only two sequence numbers and the send window size is 1. In other words, m = 1, 2^m - 1 = 1.
- In Go-Back-NARQ, we said that the addition is modulo-2m; in Stop-and-Wait ARQ it is 2, which is the same as 2^m when m = 1.

3. Selective Repeat Automatic Repeat Request

- In a noisy link a frame has a higher probability of damage, which means the resending of multiple frames. This resending uses up the bandwidth and slows down the transmission. For noisy links, there is another mechanism that does not resend N frames when just one frame is damaged; 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 Selective Repeat Protocol also uses two windows: a *send window* and a *receive window.*

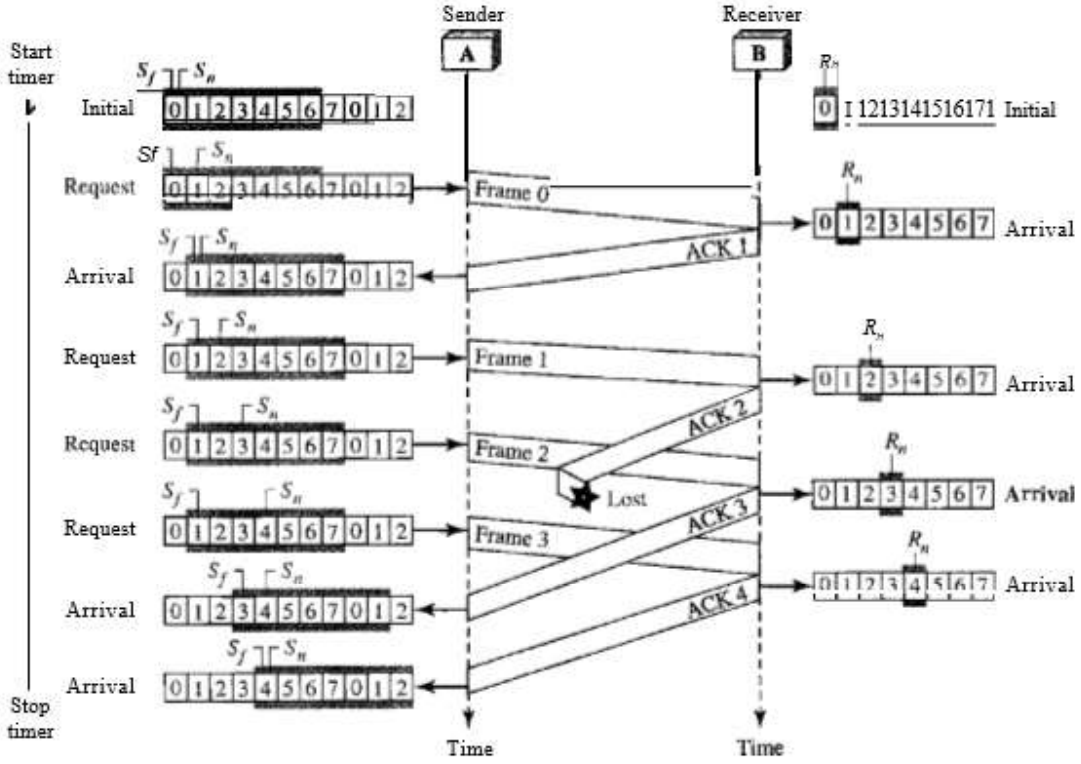


Figure 1.37 *Flow diagram*

Selective Repeat Protocol Vs Go-Back-N

- First, the size of the send window is much smaller; it is 2^m-1.
- Second, the receive window is the same size as the send window. The send window maximum size can be 2^m- 1. For example, if m = 4, the sequence numbers go from 0 to 15, but the size of the window is just 8 (it is 15 in the Go-Back-N Protocol). The smaller window size means less efficiency in filling the pipe, but the fact that there are fewer duplicate frames can compensate for this.
- The receive window in Selective Repeat is totally different from the one in Go- Back-N.
  - ✓ First, the size of the receive window is the same as the size of the send window (2^m-1).
  - ✓ The Selective Repeat Protocol allows as many frames as the size of the receive window to arrive out of order and be kept until there is a set of in-order frames to be delivered to the network layer.

Window Sizes

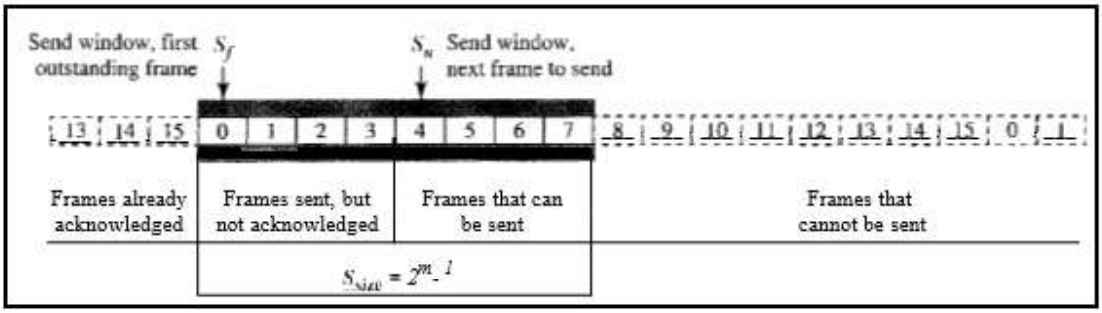


Figure 1.38 *Send window for Selective Repeat ARQ*

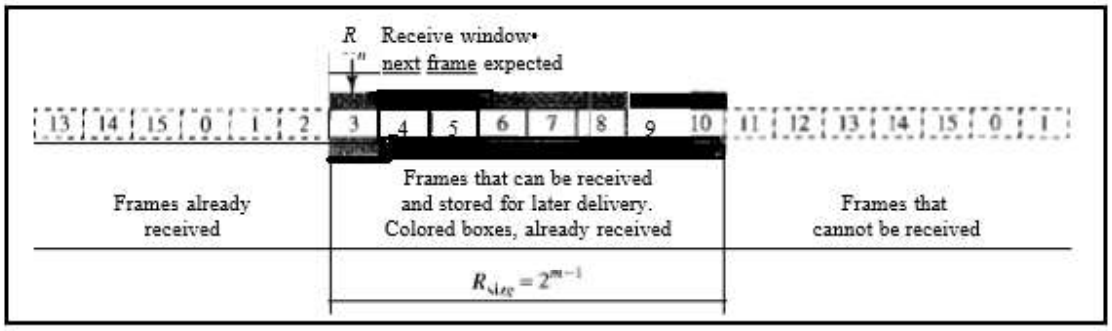


Figure 1.39 *Receive window for Selective Repeat ARQ*

- The size of the sender and receiver windows must be at most one- half of 2^m.

Selective Repeat Automatic Repeat Request Sender algorithm

```
Sn=2^m-1;
Sf=0;
Sn=0;
while(true)                          //Repeat forever
{
       WaitForEvent();          //Sleep until an event occurs
       if(Event(RequestToSend))      //A packet to send
      {
            if(Sn - Sf >= Sw)          //If window is full
                  sleep();
            GetData();
            MakeFrame(Sn);             //The seqNo is Sn
            StoreFrame(Sn);            //Keep copy
            SendFrame(Sn);             //Send the data frame
            Sn= Sn + 1;
            StartTimer(Sn)
      }
      if(Event(ArrivalNotification) //An ACK arrives
      {
            Receive(frame);            //Receive the ACK or NAK
            if(corrupted(frame)
                  sleep();
            if(FrameType==NAK)
                  if(nakNo between Sf and Sn)
                  {
                        Resend(nakNo);
                        StartTimer(nakNo);
                  }
```

```
            if(FrameType==ACK)
                  if(ackNo between Sf and Sn)
                  {
                        while(Sf <= ackNo)
                        {
                              Purge(Sf);
                              StopTimer();
                              Sf= Sf + 1;
                        }
                  }
      }
      if(Event(TimeOut(t))                //The timer Expires
      {
            StartTimer();
            SendFrame(Sf);

      }
}
```

Receiver-site algorithm for Go-Back-N Protocol

```
Rn=0;
NakSent=false;
AckNeeded=false;
Repeat(for all slots)
while(true)                           //Repeat forever
{
      WaitForEvent();                 //Sleep    until    an    event    occurs
      if(Event(ArrivalNotification))//Data frame arrived
      {
            Receive(Frame);
            if(corrupted(frame))&&(NOT NakSent)
            {
                  SendAck(Rn);
                  NakSent=true;
                  sleep();
            if(seqNo <> Rn)&&(NOT NakSent)
            {
                  SendAck(Rn);
                  NakSent=true;
                  if(seqNo in window)&&(!Marked(seqNo))
                  {
```

```
                StoreFrame(seqNo);
                Marked(seqNo);
                while(Marked(Rn))
                {
                        DeliverData(Rn);
                        Purge(Rn);
                        Rn = Rn +1;
                        AckNeeded=true;
                }
if(AckNeeded)
                {
                SendAck(Rn);
```

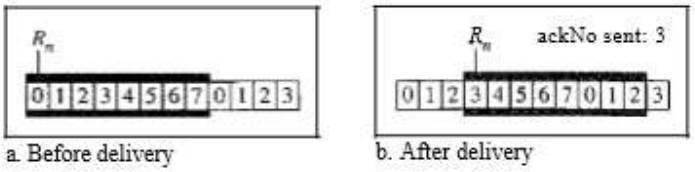


Figure 1.40 *Delivery of data in Selective Repeat ARQ*

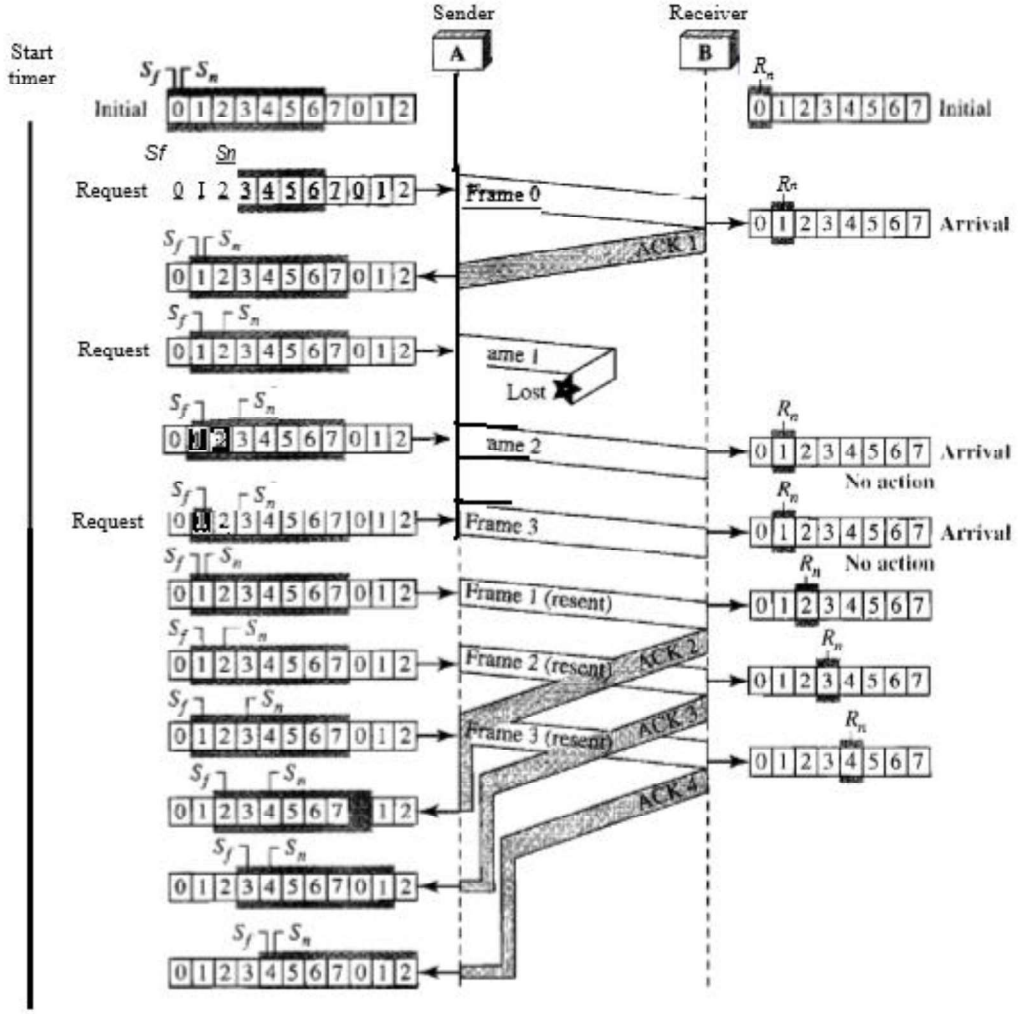


Figure 1.41 *Flow diagram*

```
                AckNeeded=false;
```

```
                                NakSent=False;
                            }
                        }
                    }
                }
            }
```

Example

Piggybacking

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.