**Computer Networks (CS425)**

**Instructor: Dr. Dheeraj Sanghi**

Prev | Next | Index

---

## ISO-OSI 7-Layer Network Architecture

This lecture introduces the ISO-OSI layered architecture of Networks. According to the ISO standards, networks have been divided into 7 layers depending on the complexity of the fucntionality each of these layers provide. The detailed description of each of these layers is given in the notes below. We will first list the layers as defined by the standard in the increasing order of function complexity:

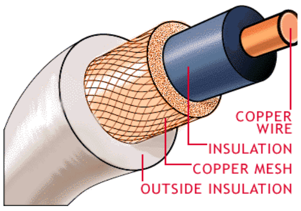
1. Physical Layer
2. Data Link Layer
3. Network Layer
4. Transport Layer
5. Session Layer
6. Presentation Layer
7. Application Layer

### Physical Layer

This layer is the lowest layer in the OSI model. It helps in the transmission of data between two machines that are communicating through a physical medium, which can be optical fibres,copper wire or wireless etc. The following are the main functions of the physical layer:

1. **Hardware Specification:** The details of the physical cables, network interface cards, wireless radios, etc are a part of this layer.

**Coaxial Cable**

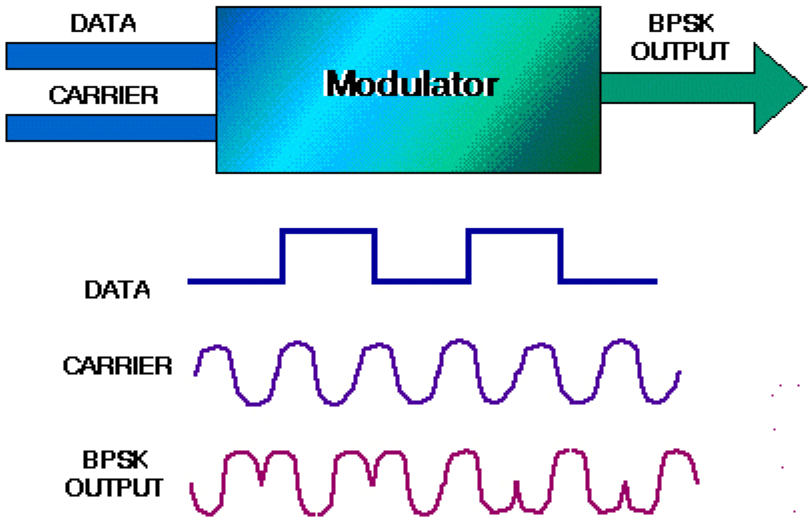


**Hybrid Cable**

**Wireless Card**

**Network Card**

2. **Encoding and Signalling:** How are the bits encoded in the medium is also decided by this layer. For example, on the coppar wire medium, we can use differnet voltage levels for a certain time interval to represent '0' and '1'. We may use +5mV for 1nsec to represent '1' and -5mV for 1nsec to represent '0'. All the issues of modulation is dealt with in this layer. eg, we may use Binary phase shift keying for the representation of '1' and '0' rather than using different volatage levels if we have to transfer in RF waves.



**Binary Phase Shift Keying**

3. **Data Transmission and Reception:** The transfer of each bit of data is the responsibility of this layer. This layer assures the transmissoin of each bit with a *high probability*. The transmission of the bits is not completely reliable as their is no error correction in this layer.

4. **Topology and Network Design:** The network design is the integral part of the physical layer. Which part of the network is the router going to be placed, where the switches will be used, where we will put the hubs, how many machines is each switch going to handle, what server is going to be placed where, and many such concerns are to be taken care of by the physical layer. The variosu kinds of netopologies that we decide to use may be ring, bus, star or a hybrid of these topologies depending on our requirements.

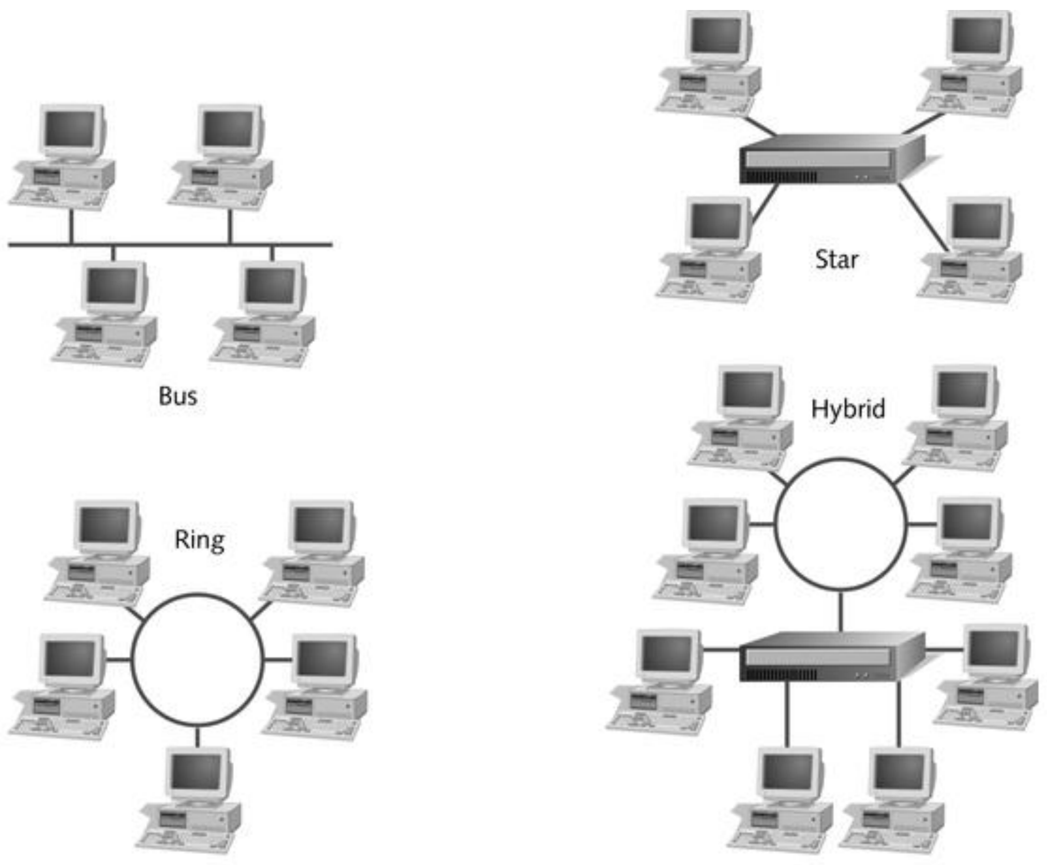
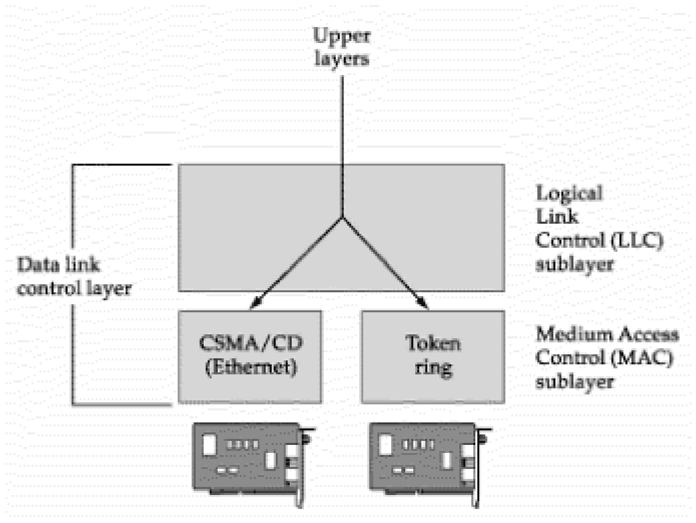


**Figure 1-7** Commonly used network topologies

**Data Link Layer**

This layer provides reliable transmission of a packet by using the services of the physical layer which transmits bits over the medium in an unreliable fashion. This layer is concerned with :

1. Framing : Breaking input data into frames (typically a few hundred bytes) and caring about the frame boundaries and the size of each frame.
2. Acknowledgment : Sent by the receiving end to inform the source that the frame was received without any error.
3. Sequence Numbering : To acknowledge which frame was received.
4. Error Detection : The frames may be damaged, lost or duplicated leading to errors.The error control is on **link to link** basis.
5. Retransmission : The packet is retransmitted if the source fails to receive acknowledgment.
6. Flow Control : Necessary for a fast transmitter to keep pace with a slow receiver.



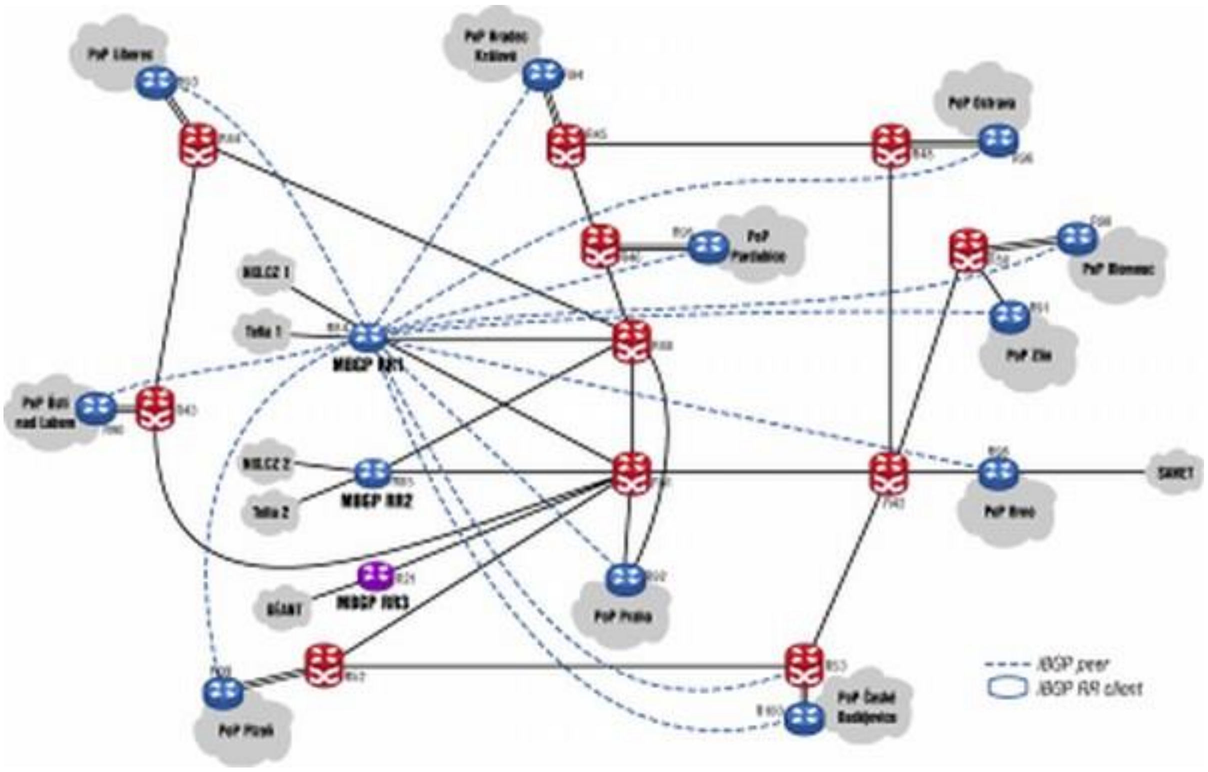
Upper layers
Data link control layer
Logical Link Control (LLC) sublayer
CSMA/CD (Ethernet)
Token ring
Medium Access Control (MAC) sublayer

**Data Link Layer**

**Network Layer**

Its basic functions are routing and congestion control.
**Routing:** This deals with determining how packets will be routed (transferred) from source to destination. It can be of three types :

- Static : Routes are based on static tables that are "wired into" the network and are rarely changed.
- Dynamic : All packets of one application can follow different routes depending upon the topology of the network, the shortest path and the current network load.
- Semi-Dynamic : A route is chosen at the start of each conversation and then all the packets of the application follow the same route.



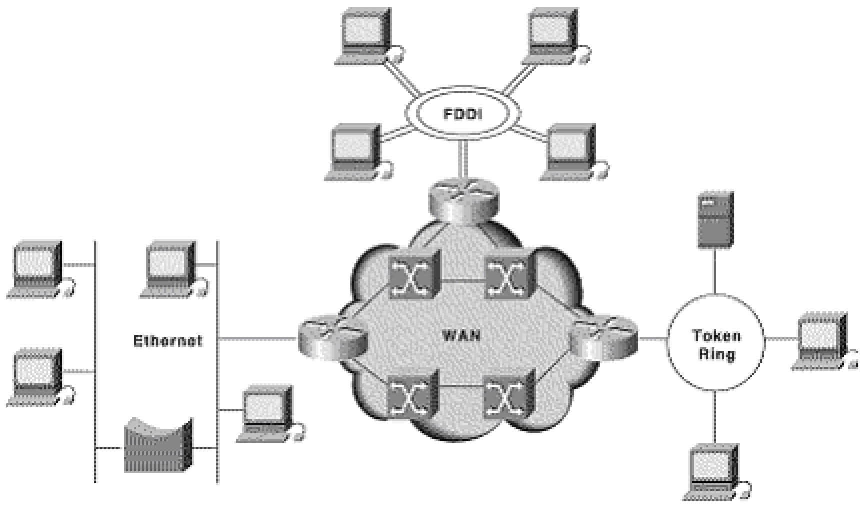
**Routing**

The services provided by the network can be of two types :

- **Connection less service:** Each packet of an application is treated as an independent entity. On each packet of the application the destination address is provided and the packet is routed.
- **Connection oriented service:** Here, first a connection is established and then all packets of the application follow the same route. To understand the above concept, we can also draw an analogy from the real life. Connection oriented service is modeled after the telephone system. All voice packets go on the same path after the connection is established till the connection is hung up. It acts like a tube ; the sender pushes the objects in at one end and the receiver takes them out in the same order at the other end. Connection less service is modeled after the postal system. Each letter carries the destination address and is routed independent of all the others. Here, it is possible that the letter sent first is delayed so that the second letter reaches the destination before the first letter.

**Congestion Control:** A router can be connected to 4-5 networks. If all the networks send packet at the same time with maximum rate possible then the router may not be able to handle all the packets and may drop some/all packets. In this context the dropping of the packets should be minimized and the source whose packet was dropped should be informed. The control of such congestion is also a function of the network layer. Other issues related with this layer are transmitting time, delays, jittering.

**Internetworking:** Internetworks are multiple networks that are connected in such a way that they act as one large network, connecting multiple office or department networks. Internetworks are connected by networking hardware such as routers, switches, and bridges.Internetworking is a solution born of three networking problems: isolated LANs, duplication of resources, and the lack of a centralized network management system. With connected LANs, companies no longer have to duplicate programs or resources on each network. This in turn gives way to managing the network from one central location instead of trying to manage each separate LAN. We should be able to transmit any packet from one network to any other network even if they follow different protocols or use different addressing modes.



**Inter-Networking**

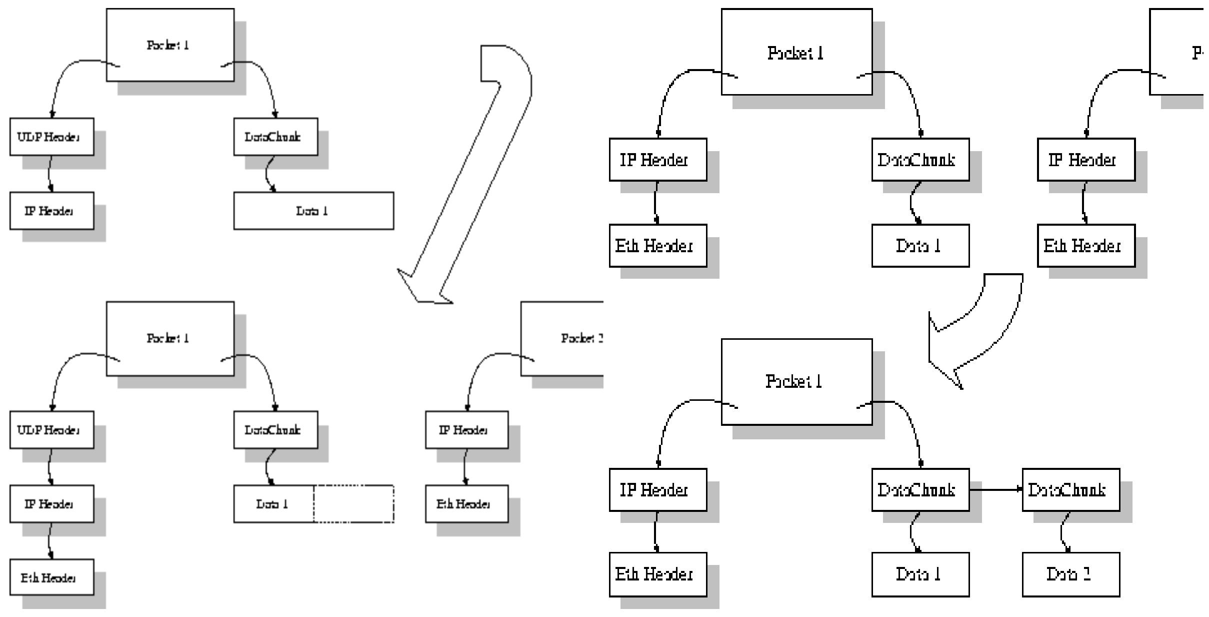
Network Layer **does not** guarantee that the packet will reach its intended destination. There are no reliability guarantees.

**Transport Layer**

Its functions are :

- **Multiplexing / Demultiplexing :** Normally the transport layer will create distinct network connection for each transport connection required by the session layer. The transport layer may either create multiple network connections (to improve throughput) or it may multiplex several transport connections onto the same network connection (because creating and maintaining networks may be expensive). In the latter case, demultiplexing will be required at the receiving end. A point to note here is that communication is always carried out between two processes and not between two machines. This is also known as process-to-process communication.
- **Fragmentation and Re-assembly :** The data accepted by the transport layer from the session layer is split up into smaller units (fragmentation) if needed and then passed to the network layer. Correspondingly, the data provided by the network layer to the transport layer on the receiving side is re-assembled.

**Fragmentation** **Reassembly**



- **Types of service :** The transport layer also decides the type of service that should be provided to the session layer. The service may be perfectly reliable, or may be reliable within certain tolerances or may not be reliable at all. The message may or may not be received in the order in which it was sent. The decision regarding the type of service to be provided is taken at the time when the connection is established.
- **Error Control :** If reliable service is provided then error detection and error recovery operations are also performed. It provides error control mechanism on **end to end** basis.
- **Flow Control :** A fast host cannot keep pace with a slow one. Hence, this is a mechanism to regulate the flow of information.
- **Connection Establishment / Release :** The transport layer also establishes and releases the connection across the network. This requires some sort of naming mechanism so that a process on one machine can indicate with whom it wants to communicate.

---

**References of Images**

- http://www.putergeek.com/.../ pci_combo_card_sm.jpg
- http://blue.utb.edu/libertad/clipart/pi_wireless_pc_card_b.jpg
- http://www.commscope.com/ images/hybrids.jpg hybrid cable
- http://www.cba.nau.edu/facstaff/maris-j/SavedStuff/Images/net_topo.gif
- http://www.ces.net/doc/2003/research/xl-unicast-routing.gif
- http://www.infinitygroup.com/images/internetworking.gif
- http://www.microway.com/.../ data_link_layer.gif
- http://searchnetworking.techtarget.com/WhatIs/images/coaxla.gif

- http://www.df.lth.se/~pkj/thesis_report/img13.gif
- http://www.df.lth.se/~pkj/ thesis_report/img12.gif
- http://www.ifla.org/VI/5/reports/rep3/rep3-2.gif
- http://msp.gsfc.nasa.gov/tdrss/bpsk.gif

---

**Computer Networks (CS425)**

**Instructor: Dr. Dheeraj Sanghi**

Prev | Next | Index

---

**Session Layer**

It deals with the concept of **Sessions** i.e. when a user logins to a remote server he should be **authenticated** before getting access to the files and application programs. Another job of session layer is to establish and maintain sessions. If during the transfer of data between two machines the session breaks down, it is the session layer which re-establishes the connection. It also ensures that the data transfer starts from where it breaks keeping it transparent to the end user. e.g. In case of a session with a database server, this layer introduces **check points** at various places so that in case the connectoin is broken and reestablished, the transition running on the database is not lost even if the user has not committed. This activity is called **Synchronization**. Another function of this layer is **Dialogue Control** which determines whose turn is it to speak in a session. It is useful in video conferencing.

**Presentation Layer**

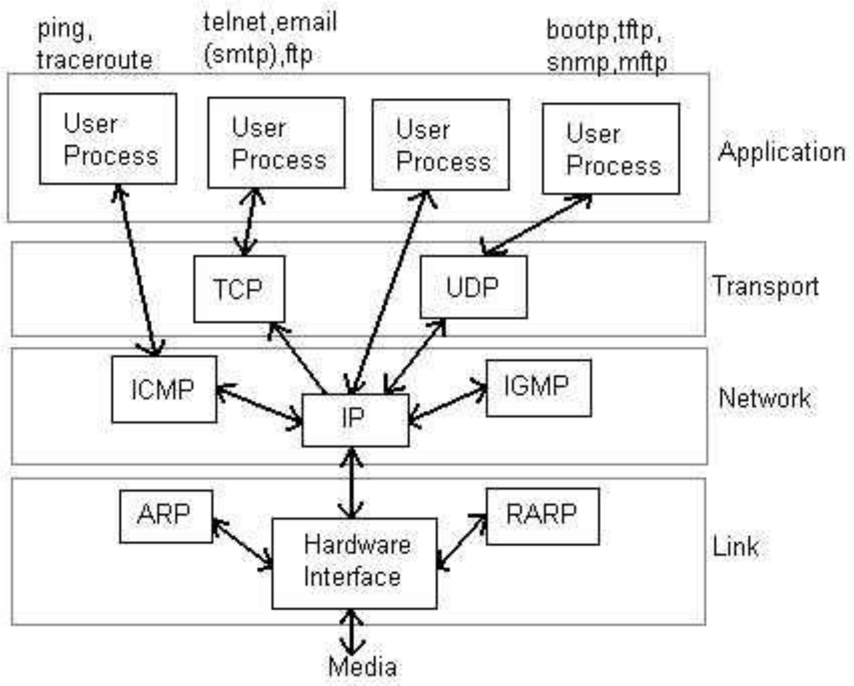
This layer is concerned with the syntax and semantics of the information transmitted. In order to make it possible for computers with different data representations to communicate data structures to be exchanged can be defined in abstract way alongwith standard encoding. It also manages these abstract data structres and allows higher level of data structres to be defined an exchange. It encodes the data in standard agreed way(network format). Suppose there are two machines A and B one follows 'Big Endian' and other 'Little Endian' for data representation. This layer ensures that the data transmitted by one gets converted in the form compatibale to othe machine. This layer is concerned with the syntax and semantics of the information transmitted.In order to make it possible for computers with different data representations to communicate data structures to be exchanged canbe defined in abstract way alongwith standard encoding. It also manages these abstract data structres and allows higher level of data structres to be defined an exchange. Other functions include compression, encryption etc.

**Application Layer**

The seventh layer contains the application protocols with which the user gains access to the network. The choice of which specific protocols and their associated functions are to be used at the application level is up to the individual user. Thus the boundary between the presentation layer and the application layer represents a separation of the protocols imposed by the network designers from those being selected and implemented by the network users.For example commonly used protocols are HTTP(for web browsing), FTP(for file transfer) etc.

**Network Layers as in Practice**

In most of the networks today, we do not follow the OSI model of seven layers. What is actually implemented is as follows. The functionality of Application layer and Presentation layer is merged into one and is called as the Application Layer. Functionalities of Session Layer is not implemented in most networks today. Also, the Data Link layer is split theoretically into **MAC (Medium Access Control) Layer** and **LLC (Link Layer Control)**. But again in practice, the LLC layer is not implemented by most networks. So as of today, the network architecture is of 5 layers only.



**Network Layers in Internet Today**

**Some Related Links on OSI Model and TCP Model**

- http://en.wikipedia.org/wiki/OSI_model
- http://www.tcpipguide.com/free/t_OSIReferenceModelLayers.htm
- http://www.geocities.com/SiliconValley/Monitor/3131/ne/osimodel.html
- http://www.tech-faq.com/osi-model.shtml
- http://www.networkdictionary.com/protocols/osimodel.php

---

## Physical Layer

Physical layer is concerned with transmitting raw bits over a communication channel. The design issues have to do with making sure that when one side sends a 1 bit, it is recieved by the other side as 1 bit and not as 0 bit. In physical layer we deal with the communication medium used for transmission.

**Types of Medium**

Medium can be classified into 2 categories.

1. **Guided Media :** Guided media means that signals is guided by the prescence of physical media i.e. signals are under control and remains in the physical wire. For eg. copper wire.
2. **Unguided Media :** Unguided Media means that there is no physical path for the signal to propogate. Unguided media are essentially electro-magnetic waves. There is no control on flow of signal. For eg. radio waves.

**Communication Links**

In a nework nodes are connected through links. The communication through links can be classified as

1. **Simplex :** Communication can take place only in one direction. eg. T.V broadcasting.
2. **Half-duplex :** Communication can take place in one direction at a time. Suppose node A and B are connected then half-duplex communication means that at a time data can flow from A to B or from B to A but not simultaneously. eg. two persons talking to each other such that when speaks the other listens and vice versa.
3. **Full-duplex :** Communication can take place simultaneously in both directions. eg. A discussion in a group without discipline.

Links can be further classified as

1. **Point to Point :** In this communication only two nodes are connected to each other. When a node sends a packet then it can be recieved only by the node on the other side and none else.
2. **Multipoint** : It is a kind of sharing communication, in which signal can be recieved by all nodes. This is also called broadcast.

Generally two kind of problems are associated in transmission of signals.

1. **Attenuation :** When a signal transmitts in a network then the quality of signal degrades as the signal travels longer distances in the wire. This is called attenuation. To improve quality of signal amplifiers are used at regular distances.
2. **Noise :** In a communication channel many signals transmits simultaneously, certain random signals are also present in the medium. Due to interference of these signals our signal gets disrupted a bit.

**Bandwidth**

Bandwidth simply means how many bits can be transmitted per second in the communication channel. In technical terms it indicates the width of frequency spectrum.

**Transmission Media**

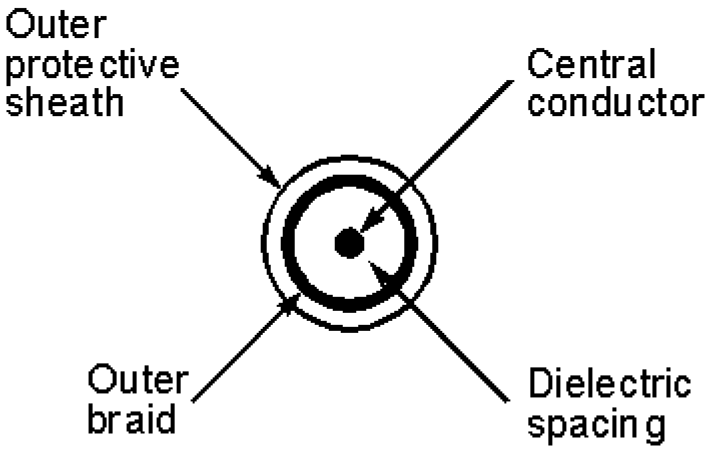
**Guided Transmission Media**
In Guided transmission media generally two kind of materials are used.

1. Copper
    - Coaxial Cable

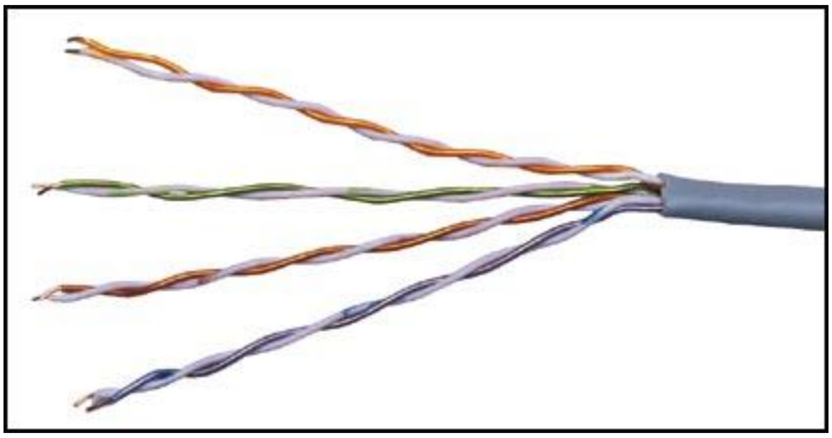
   - Twisted Pair

2. Optical Fiber

1. **Coaxial Cable:** Coaxial cable consists of an inner conductor and an outer conductor which are seperated by an insulator. The inner conductor is usually copper. The outer conductor is covered by a plastic jacket. It is named coaxial because the two conductors are coaxial. Typical diameter of coaxial cable lies between 0.4 inch to 1 inch. The most application of coaxial cable is cable T.V. The coaxial cable has high bandwidth, attenuation is less.



2. **Twisted Pair:** A Twisted pair consists of two insulated copper wires, typically 1mm thick. The wires are twisted togather in a helical form the purpose of twisting is to reduce cross talk interference between several pairs. Twisted Pair is much cheaper then coaxial cable but it is susceptible to noise and electromagnetic interference and attenuation is large.



Twisted Pair can be further classified in two categories:
**Unshielded twisted pair:** In this no insulation is provided, hence they are susceptible to interference.
**Shielded twisted pair:** In this a protective thick insulation is provided but shielded twisted pair is expensive and not commonly used.

The most common application of twisted pair is the telephone system. Nearly all telephones are connected to the telephone company office by a twisted pair. Twisted pair can run several kilometers without amplification, but for longer distances repeaters are needed. Twisted pairs can be used for both analog and digital transmission. The bandwidth depends on the thickness of wire and the distance travelled. Twisted pairs are generally limited in distance, bandwidth and data rate.

3. **Optical Fiber:** In optical fiber light is used to send data. In general terms prescence of light is taken as bit 1 and its absence as bit 0. Optical fiber consists of inner core of either glass or plastic. Core is surrounded by cladding of the same material but of different refrective index. This cladding is surrounded by a plastic jacket which prevents optical fiber from electromagnetic interferrence and harshy environments. It uses the principle of total internal reflection to transfer data over optical fibers. Optical fiber is much better in bandwidth as compared to copper wire, since there is hardly any attenuation or electromagnetic interference in optical wires. Hence there is less requirement to improve quality of signal, in long distance transmission. Disadvantage of optical fiber is that end points are fairly expensive. (eg. switches)

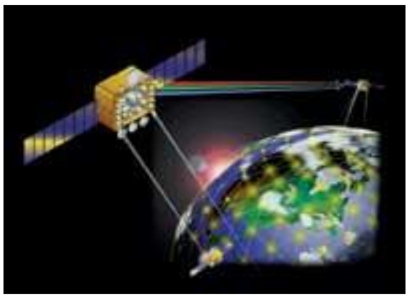
Differences between different kinds of optical fibers:

1. Depending on material
   - Made of glass
   - Made of plastic.
2. Depending on radius
   - Thin optical fiber
   - Thick optical fiber
3. Depending on light source
   - LED (for low bandwidth)
   - Injection lased diode (for high bandwidth)

**Wireless Transmission**

1. **Radio:** Radio is a general term that is used for any kind of frequency. But higher frequencies are usually termed as microwave and the lower frequency band comes under radio frequency. There are many application of radio. For eg. cordless keyboard, wireless LAN, wireless ethernet. but it is limited in range to only a few hundred meters. Depending on frequency radio offers different bandwidths.
2. **Terrestrial microwave:** In terrestrial microwave two antennas are used for communication. A focused beam emerges from an antenna and is recieved by the other antenna, provided that antennas should be facing each other with no obstacle in between. For this reason antennas are situated on high towers. Due to curvature of earth terristial microwave can be used for long distance communication with high bandwidth. Telecom department is also using this for long distance communication. An advantage of wireless communication is that it is not required to lay down wires in the city hence no permissions are required.
3. **Satellite communication:** Satellite acts as a switch in sky. On earth VSAT(Very Small Aperture Terminal) are used to transmit and recieve data from satellite. Generally one station on earth transmitts signal to satellite and it is recieved by many stations on earth. Satellite communication is generally used in those places where it is very difficult to obtain line of sight i.e. in highly irregular terristial regions. In terms of noise wireless media is not as good as the wired media. There are frequency band in wireless communication and two stations should not be allowed to transmit simultaneously in a frequency band. The most promising

advantage of satellite is broadcasting. If satellites are used for point to point communication then they are expensive as compared to wired media.



---

**References of Images**

- http://ai3.asti.dost.gov.ph/sat/levels.jpg
- http://www.ll.mit.edu/Image-Lib/photos/color-Satellite.jpg
- http://oldsite.vislab.usyd.edu.au/photonics/revolution/technology/images/twisted_pair.jpg
- http://www.radio-electronics.com/info/antennas/coax/cross_section_thro_coax.gif

---

**Computer Networks (CS425)**

**Instructor: Dr. Dheeraj Sanghi**

Prev| Next| Index

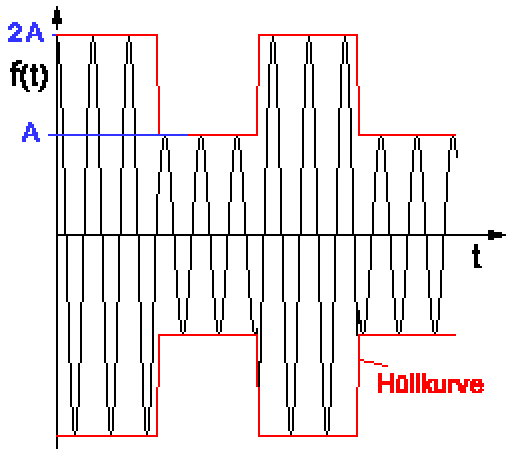
---

## Data Encoding

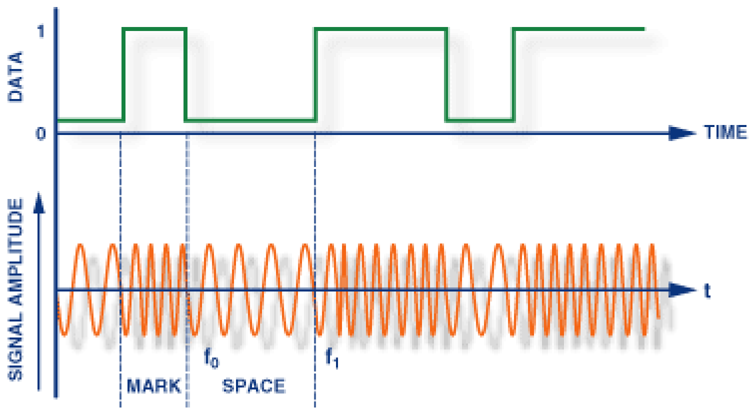
**Digital data to analog signals**

A modem (modulator-demodulator) converts digital data to analog signal. There are 3 ways to modulate a digital signal on an analog carrier signal.

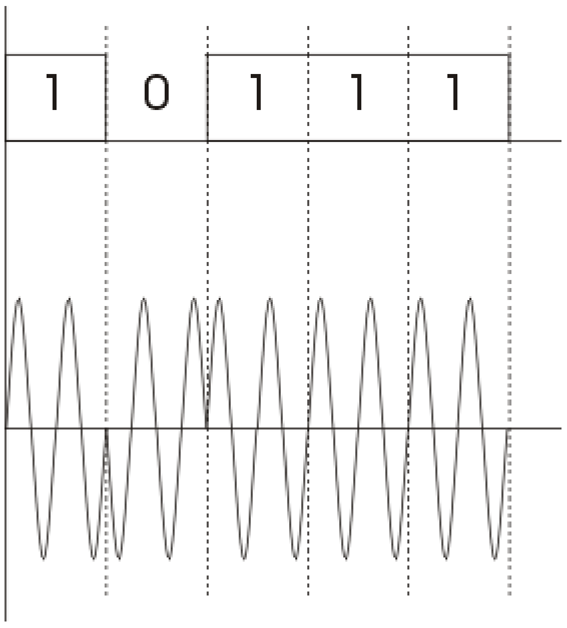
1. **Amplitude shift keying (ASK):** is a form of modulation which represents digital data as variations in the amplitude of a carrier wave. Two different amplitudes of carrier frequency represent '0' , '1'.



2. **Frequency shift keying (FSK):** In Frequency Shift Keying, the change in frequency define different digits. Two different frequencies near carrier frequency represent '0' ,''1'.



3. **Phase shift keying (PSK):** The phase of the carrier is discretely varied in relation either to a reference phase or to the phase of the immediately preceding signal element, in accordance with data being transmitted. Phase of carrier signal is shifted to represent '0' , '1'.



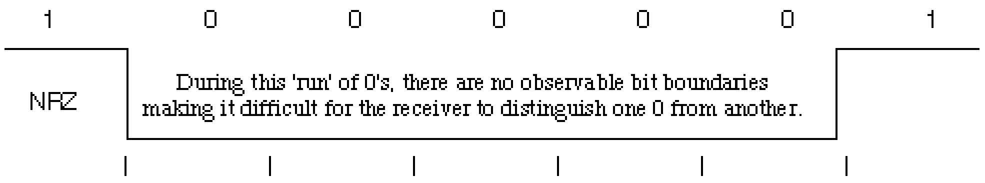
**Digital data to digital signals**

A digital signal is sequence of discrete , discontinuous voltage pulses. Each pulses a signal element. Encoding scheme is an important factor in how successfully the receiver interprets the incoming signal.

**Encoding Techniques**

Following are several ways to map data bits to signal elements.

- **Non return to zero(NRZ)** NRZ codes share the property that voltage level is constant during a bit interval. High level voltage = bit 1 and Low level voltage = bit 0. A problem arises when there is a long sequence of 0s or 1s and the volatage level is maintained at the same value for a long time. This creates a problem on the recieving end because now, the clock synchronization is lost due to lack of any transitions and hence, it is difficult to determine the exact number of 0s or 1s in this sequence.


1
0
0
0
0
0
1
NRZ
During this 'run' of 0's, there are no observable bit boundaries making it difficult for the receiver to distinguish one 0 from another.

The two variations are as follows:

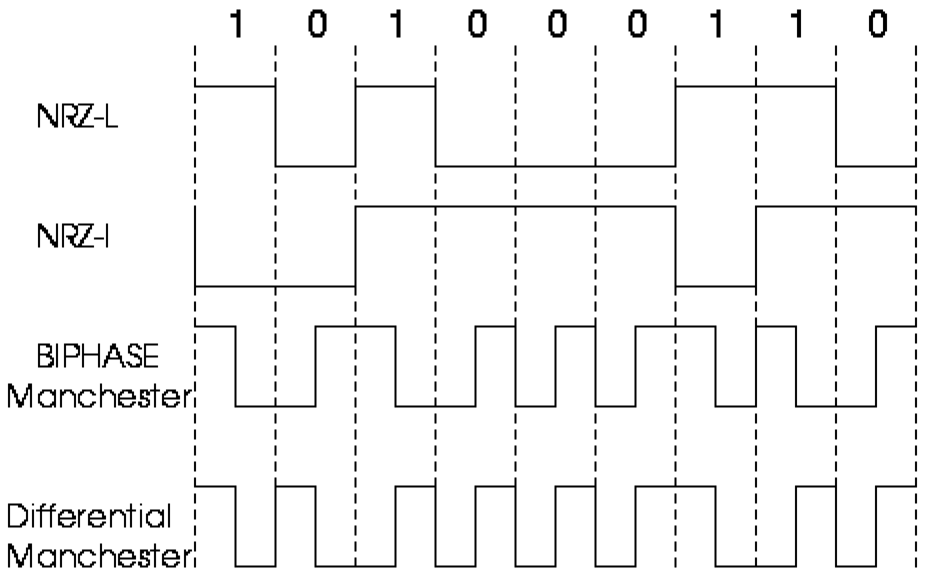
1. **NRZ-Level:** In NRZ-L encoding, the polarity of the signal changes only when the incoming signal changes from a 1 to a 0 or from a 0 to a 1. NRZ-L method looks just like the NRZ method, except for the first input one data bit. This is because NRZ does not consider the first data bit to be a polarity change, where NRZ-L does.
2. **NRZ-Inverted:** Transition at the beginning of bit interval = bit 1 and No Transition at beginning of bit interval = bit 0 or vicecersa. This technique is known as differential encoding.

NRZ-I has an advantage over NRZ-L. Consider the situation when two data wires are wrongly connected in each other's place.In NRZ-L all bit sequences will get reversed (B'coz voltage levels get swapped).Whereas in NAZ-I since bits are recognized by transition the bits will be correctly interpreted. A disadvantage in NRZ codes is that a string of 0's or 1's will prevent synchronization of transmitter clock with receiver clock and a separate clock line need to be provided.

- **Biphase encoding:** It has following characteristics:
    1. Modulation rate twice that of NRZ and bandwidth correspondingly greater. (Modulation is the rate at which signal level is changed).
    2. Because there is predictable transition during each bit time,the receiver can synchronize on that transition i.e. clock is extracted from the signal itself.
    3. Since there can be transition at the beginning as well as in the middle of the bit interval the clock operates at twice the data transfer rate.

Types of Encoding -->

- o **Biphase-manchester:** Transition from high to low in middle of interval = 1 and Transition from low to high in middle of interval = 0
- o **Differential-manchester:** Always a transition in middle of interval. No transition at beginning of interval=1 and Transition at beginning of interval = 0



- o **4B/5B Encoding:** In Manchester encoding scheme , there is a transition after every bit. It means that we must have clocks with double the speed to send same amount of data as in NRZ encodings. In other words, we may say that only 50% of the data is sent. This performance factor can be significantly improved if we use a better encoding scheme. This scheme may have a transition after fixed number of bits instead of every other bit. Like if we have a transition after every four bits, then we will be sending 80% data of actual capacity. This is a significant improvement in the performance.

This scheme is known as **4B/5B**. So here we convert 4-bits to 5-bits, ensuring at least one transition in them. The basic idea here is that 5-bit code selected must have :

- one leading 0
- no more than two trailing 0s

Thus it is ensured that we can never have more than three consecutive 0s. Now these 5-bit codes are transmitted using NRZI coding thus problem of consecutive 1s is solved.

The exact transformation is as follows :

| 4-bit Data | 5-bit code | 4-bit Data | 5-bit code |
|---|---|---|---|
| 0000 | 11110 | 1000 | 10010 |
| 0001 | 01001 | 1001 | 10011 |
| 0010 | 10100 | 1010 | 10110 |
| 0011 | 10101 | 1011 | 10111 |
| 0100 | 01010 | 1100 | 11010 |
| 0101 | 01011 | 1101 | 11011 |
| 0110 | 01110 | 1110 | 11100 |
| 0111 | 01111 | 1111 | 11101 |

Of the remaining 16 codes, 7 are invalid and others are used to send some control information like line idle(11111), line dead(00000), Halt(00100) etc.

There are other variants for this scheme viz. 5B/6B, 8B/10B etc. These have self suggesting names.

- **8B/6T Encoding:** In the above schemes, we have used two/three voltage levels for a signal. But we may altogether use more than three voltage levels so that more than one-bit could be send over a single signal. Like if we use six voltage levels and we use 8-bits then the scheme is called **8B/6T**. Clearly here we have 729(3^6) combinations for signal and 256(2^8) combinations for bits.

- **Bipolar AIM:** Here we have 3 voltage levels: middle,upper,lower
  - Representation 1: Middle level =0 Upper,Lower level =1 such that successive 1's will be represented alternately on upper and lower levels.
  - Representation 2 (pseudoternary): Middle level =1 Upper,Lower level=0

**Analog data to digital signal:**

The process is called digitization. Sampling frequency must be at least twice that of highest frequency present in the the signal so that it may be fairly regenerated. Quantization - Max. and Min values of amplitude in the sample are noted. Depending on number of bits (say n) we use we divide the interval (min,max) into 2(^n) number of levels. The amplitude is then approximated to the nearest level by a 'n' bit integer. The digital signal thus consists of blocks of n bits.On reception the process is reversed to produce analog signal. But a lot of data can be lost if fewer bits are used or sampling frequency not so high.

- **Pulse code modulation(PCM):** Here intervals are equally spaced. 8 bit PCB uses 256 different levels of amplitude. In non-linear encoding levels may be unequally spaced.
- **Delta Modulation(DM):** Since successive samples do not differ very much we send the differences between previous and present sample. It requires fewer bits than in PCM.

**Digital Data Communication Techniques:**

For two devices linked by a transmission medium to exchange data ,a high degree of co-operation is required. Typically data is transmitted one bit at a time. The timing (rate, duration,spacing) of these bits must be same for transmitter and receiver. There are two options for transmission of bits.

1. **Parallel** All bits of a byte are transferred simultaneously on separate parallel wires. Synchronization between multiple bits is required which becomes difficult over large distance. Gives large band width but expensive. Practical only for devices close to each other.
2. **Serial** Bits transferred serially one after other.Gives less bandwidth but cheaper. Suitable for transmission over long distances.

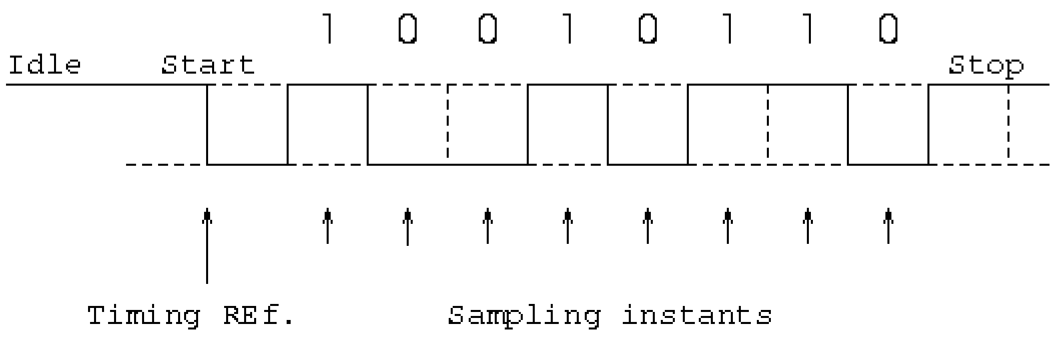
**Transmission Techniques:**

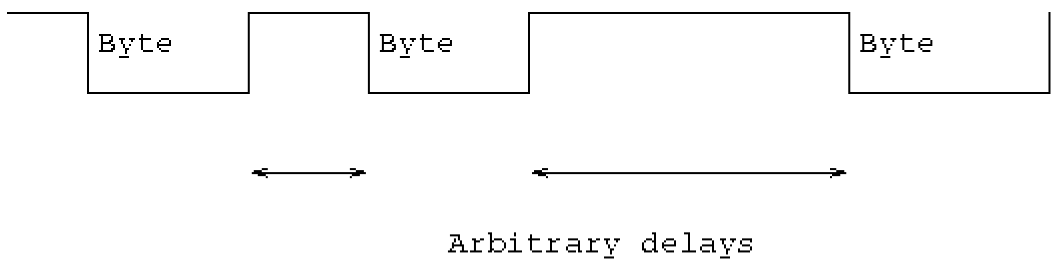
1. **Asynchronous:** Small blocks of bits(generally bytes) are sent at a time without any time relation between consecutive bytes .when no transmission occurs a default state is maintained corresponding to bit 1. Due to arbitrary delay between consecutive bytes,the time occurrences of the clock pulses at the receiving end need to be synchronized for each byte. This is achieved by providing 2 extra bits start and stop.

**Start bit:** It is prefixed to each byte and equals 0. Thus it ensures a transition from 1 to 0 at onset of transmission of byte.The leading edge of start bit is used as a reference for generating clock pulses at required sampling instants. Thus each onset of a byte results in resynchronization of receiver clock.

**Stop bit:** To ensure that transition from 1 to 0 is always present at beginning of a byte it is necessary that default state be 1. But there may be two bytes one immediately following the other and if last bit of first byte is 0, transition from 1 to 0 will not occur . Therefore a stop bit is suffixed to each byte equaling 1. It's duration is usually 1,1.5,2 bits.

Asynchronous transmission is simple and cheap but requires an overhead of 3 bits i.e. for 7 bit code 2 (start ,stop bits)+1 parity bit implying 30% overhead.However % can be reduced by sending larger blocks of data but then timing errors between receiver and sender can not be tolerated beyond [50/no. of bits in block] % (assuming sampling is done at middle of bit interval). It will not only result in incorrect sampling but also misaligned bit count i.e. a data bit can be mistaken for stop bit if receiver's clock is faster.

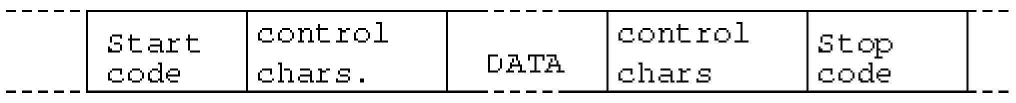




2. **Synchronous** - Larger blocks of bits are successfully transmitted.Blocks of data are either treated as sequence of bits or bytes. To prevent timing drift clocks at two ends need to be synchronized.This can done in two ways:
    1. Provide a separate clock line between receiver and transmitter. OR
    2. Clocking information is embedded in data signal i.e. biphase coding for digital signals.

Still another level of synchronization is required so that receiver determines beginning or end of block of data. Hence each block begins with a start code and ends with a stop code.These are in general same known as flag that is unique sequence of fixed no. of bits.In addition some control characters encompass data within these flags. **Data+control information** is called a frame. Since any arbitrary bit pattern can be transmitted there is no assurance that bit pattern for flag will not appear inside the frame thus destroying frame level synchronization. So to avoid this we use bit stuffing

**Bit Stuffing:** Suppose our flag bits are 01111110 (six 1's). So the transmitter will always insert an extra 0 bit after each occurrence of five 1's (except for flags). After detecting a starting flag the receiver monitors the bit stream . If pattern of five 1's appear, the sixth is examined and if it is 0 it isdeleted else if it is 1 and next is 0 the combination is accepted as a flag. Similarly byte stuffing is used for byte oriented transmission.Here we use an escape sequence to prefix a byte similar to flag and 2 escape sequences if byte is itself a escape sequence.



---

**Image References:**

- http://www.analog.com/library/analogDialogue/archives/38-08/DDS_06.gif

- http://racon.net/archive/FHTW/projects/Richtfunk/2psk.png
- http://fara.cs.uni-potsdam.de/~rnitschk/wireless/antenna/pics/009001.gif
- http://www.erg.abdn.ac.uk/users/gorry/course/images/nrz-run.gif

---

back to top
Prev | Next | Index

**Computer Networks (CS425)**

**Instructor: Dr. Dheeraj Sanghi**

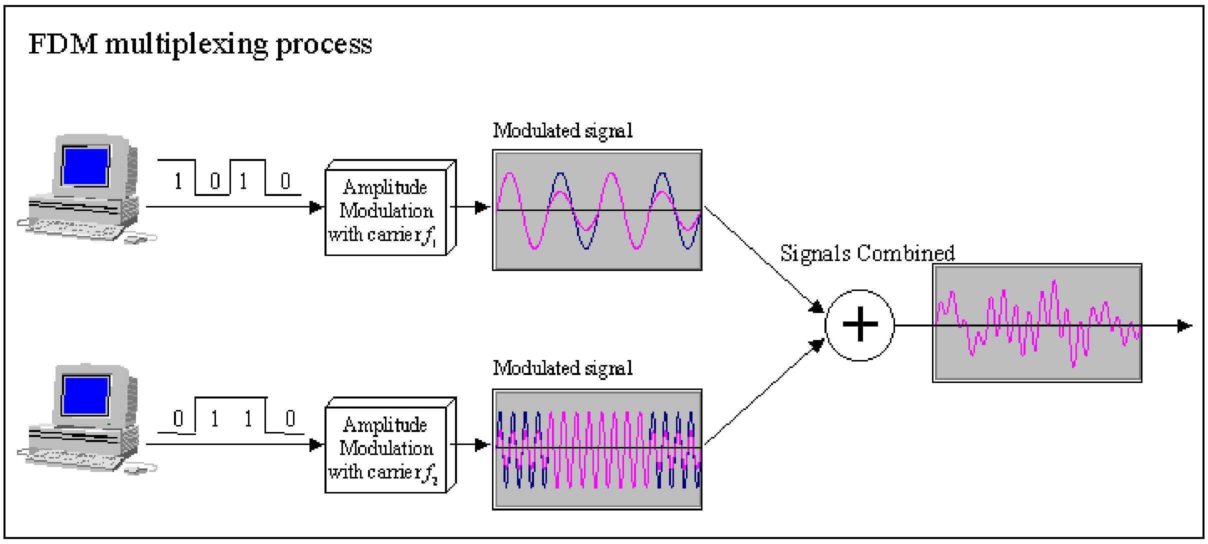
Prev | Next | Index

---

## Multiplexing

When two communicating nodes are connected through a media, it generally happens that bandwidth of media is several times greater than that of the communicating nodes. Transfer of a single signal at a time is both slow and expensive. The whole capacity of the link is not being utilized in this case. This link can be further exploited by sending several signals combined into one. This combining of signals into one is called multiplexing.

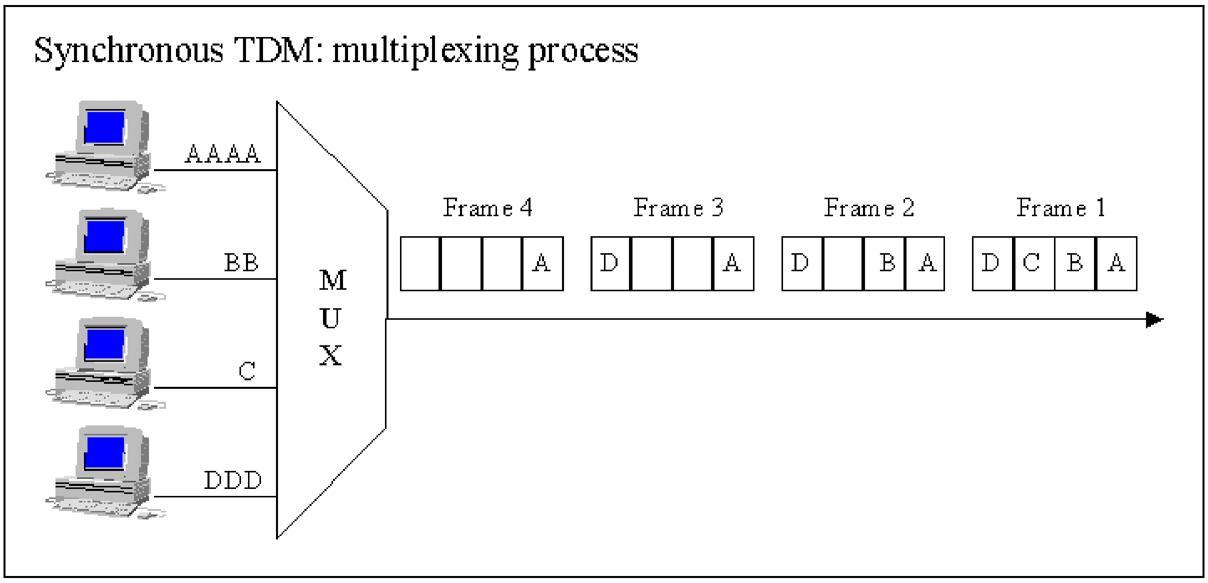
1. **Frequency Division Multiplexing (FDM):** This is possible in the case where transmission media has a bandwidth than the required bandwidth of signals to be transmitted. A number of signals can be transmitted at the same time. Each source is allotted a frequency range in which it can transfer it's signals, and a suitable frequency gap is given between two adjescent signals to avoid overlapping. This is type of multiplexing is commonly seen in the cable TV networks.



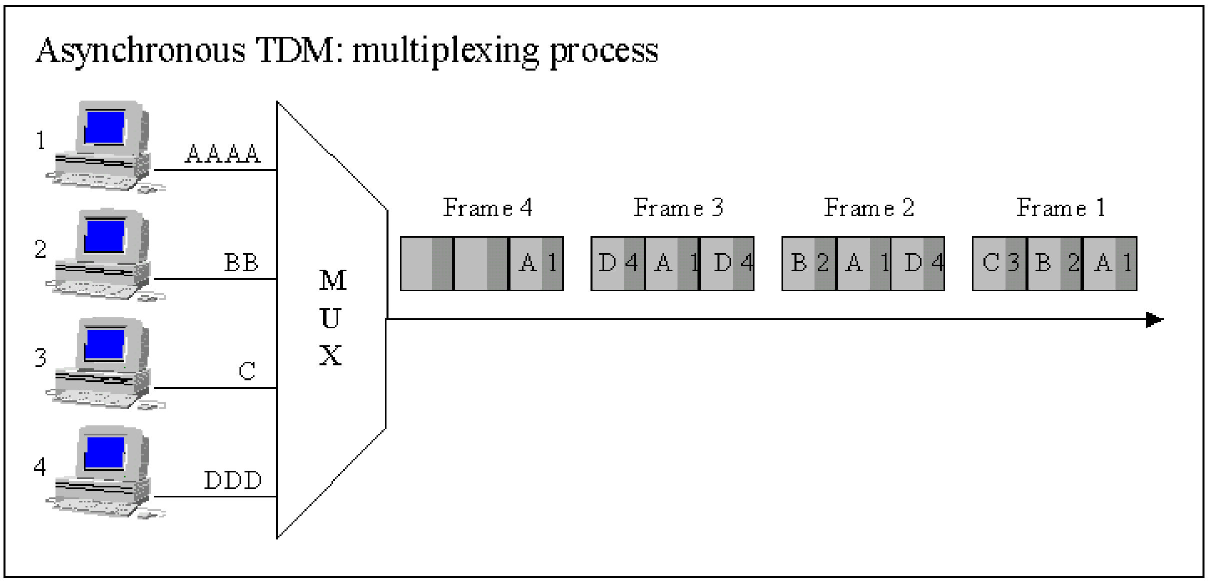
2. **Time Division Multiplexing (TDM):** This is possible when data transmission rate of the media is much higher than that of the data rate of the source. Multiple signals can be transmitted if

each signal is allowed to be transmitted for a definite amount of time. These time slots are so small that all transmissions appear to be in parallel.

1. **Synchronous TDM:** Time slots are preassigned and are fixed. Each source is given it's time slot at every turn due to it. This turn may be once per cycle, or several turns per cycle ,if it has a high data transfer rate, or may be once in a no. of cycles if it is slow. This slot is given even if the source is not ready with data. So this slot is transmitted empty.



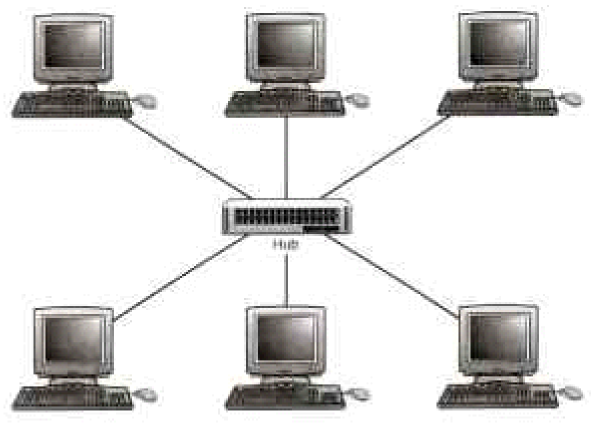
2. **Asynchronous TDM:** In this method, slots are not fixed. They are allotted dynamically depending on speed of sources, and whether they are ready for transmission.



**Network Topologies**

A network topology is the basic design of a computer network. It is very much like a map of a road. It details how key network components such as nodes and links are interconnected. A network's topology is comparable to the blueprints of a new home in which components such as the electrical system, heating and air conditioning system, and plumbing are integrated into the overall design. Taken from the Greek work "Topos" meaning "Place," Topology, in relation to networking, describes the configuration of the network; including the location of the workstations and wiring connections. Basically it provides a definition of the components of a Local Area Network (LAN). A topology, which is a pattern of interconnections among nodes, influences a network's cost and performance. There are three primary types of network topologies which refer to the physical and logical layout of the Network cabling. They are:

1. **Star Topology:** All devices connected with a Star setup communicate through a central Hub by cable segments. Signals are transmitted and received through the Hub. It is the simplest and the oldest and all the telephone switches are based on this. In a star topology, each network device has a home run of cabling back to a network hub, giving each device a separate connection to the network. So, there can be multiple connections in parallel.



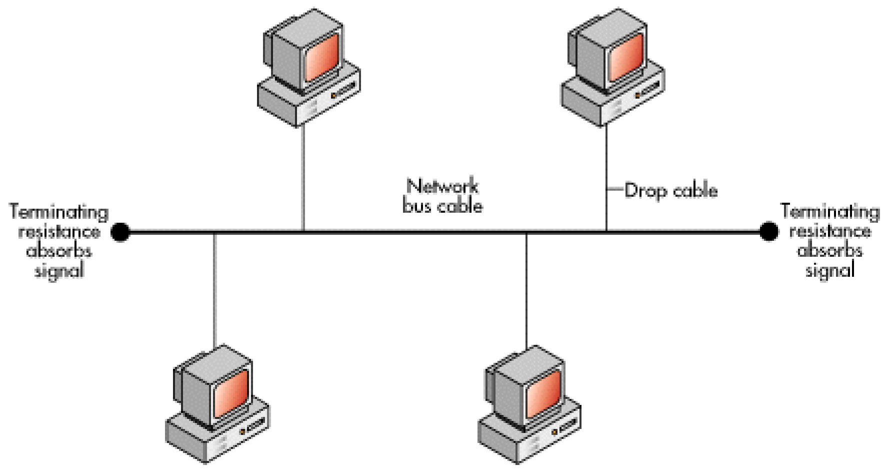
**Advantages**

- Network administration and error detection is easier because problem is isolated to central node
- Networks runs even if one host fails
- Expansion becomes easier and scalability of the network increases
- More suited for larger networks

**Disadvantages**

- Broadcasting and multicasting is not easy because some extra functionality needs to be provided to the central hub
- If the central node fails, the whole network goes down; thus making the switch some kind of a bottleneck
- Installation costs are high because each node needs to be connected to the central switch

2. **Bus Topology:** The simplest and one of the most common of all topologies, Bus consists of a single cable, called a Backbone, that connects all workstations on the network using a single line. All transmissions must pass through each of the connected devices to complete the desired request. Each workstation has its own individual signal that identifies it and allows for the requested data to be returned to the correct originator. In the Bus Network, messages are sent in both directions from a single point and are read by the node (computer or peripheral on the network) identified by the code with the message. Most Local Area Networks (LANs) are Bus Networks because the network will continue to function even if one computer is down. This topology works equally well for either peer to peer or client server.


Network bus cable
Drop cable
Terminating resistance absorbs signal
Terminating resistance absorbs signal

The purpose of the terminators at either end of the network is to stop the signal being reflected back.

**Advantages**

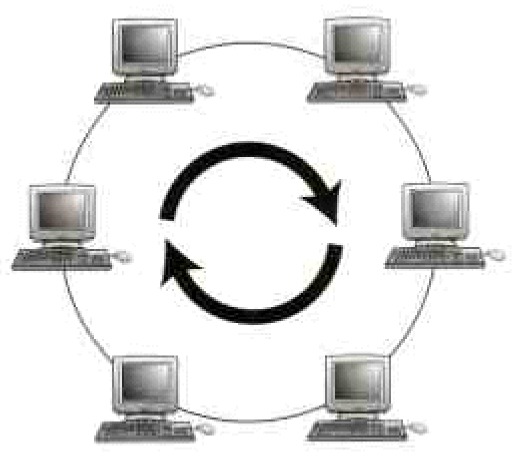
- Broadcasting and multicasting is much simpler
- Network is redundant in the sense that failure of one node doesn't effect the network. The other part may still function properly
- Least expensive since less amount of cabling is required and no network switches are required
- Good for smaller networks not requiring higher speeds

**Disadvantages**

- Trouble shooting and error detection becomes a problem because, logically, all nodes are equal
- Less secure because sniffing is easier
- Limited in size and speed

3. **Ring Topology:** All the nodes in a Ring Network are connected in a closed circle of cable. Messages that are transmitted travel around the ring until they reach the computer that they are addressed to, the signal being refreshed by each node. In a ring topology, the network signal is passed through each network card of each device and passed on to the next device.

Each device processes and retransmits the signal, so it is capable of supporting many devices in a somewhat slow but very orderly fashion. There is a very nice feature that everybody gets a chance to send a packet and it is guaranteed that every node gets to send a packet in a finite amount of time.



**Advantages**

- Broadcasting and multicasting is simple since you just need to send out one message
- Less expensive since less cable footage is required
- It is guaranteed that each host will be able to transmit within a finite time interval
- Very orderly network where every device has access to the token and the opportunity to transmit
- Performs better than a star network under heavy network load

**Disadvantages**

- Failure of one node brings the whole network down
- Error detection and network administration becomes difficult
- Moves, adds and changes of devices can effect the network
- It is slower than star topology under normal load

Generally, a BUS architecture is preferred over the other topologies - ofcourse, this is a very subjective opinion and the final design depends on the requirements of the network more than anything else. Lately, most networks are shifting towards the STAR topology. Ideally we would like to design networks, which physically resemble the STAR topology, but behave like BUS or RING topology.

---

**Data Link Layer**

Data link layer can be characterized by two types of layers:

1. Medium Access Layer (MAL)

2. Logical Link Layer

**Aloha Protocols**

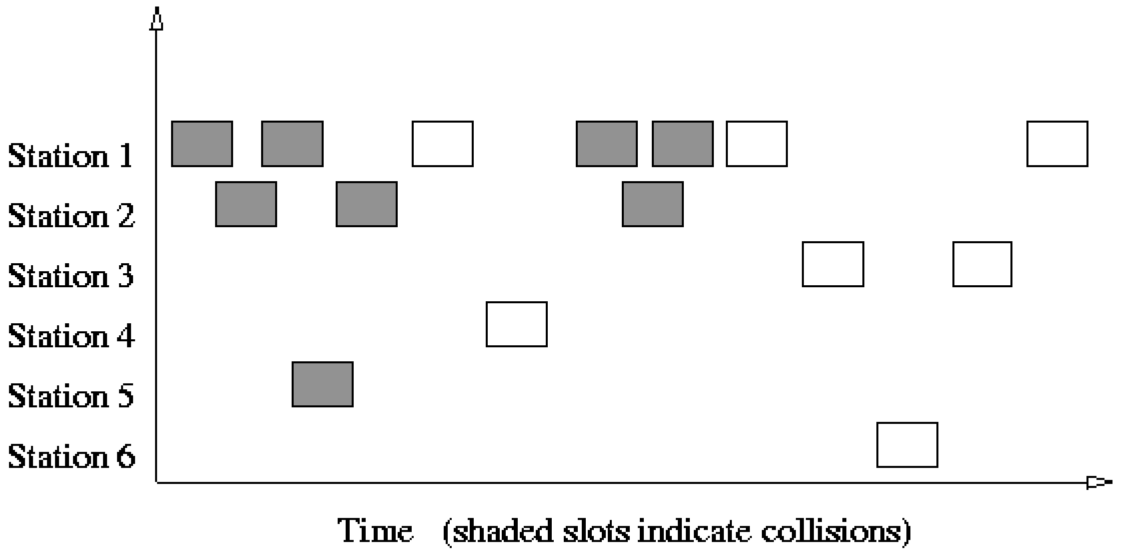
**History**

The Aloha protocol was designed as part of a project at the University of Hawaii. It provided data transmission between computers on several of the Hawaiian Islands using radio transmissions.

- Communications was typically between remote stations and a central sited named Menehune or vice versa.
- All message to the Menehune were sent using the same frequency.
- When it received a message intact, the Menehune would broadcast an ack on a distinct outgoing frequency.
- The outgoing frequency was also used for messages from the central site to remote computers.
- All stations listened for message on this second frequency.

**Pure Aloha**

Pure Aloha is an unslotted, fully-decentralized protocol. It is extremely simple and trivial to implement. The ground rule is - "when you want to talk, just talk!". So, a node which wants to transmits, will go ahead and send the packet on its broadcast channel, with no consideration whatsoever as to anybody else is transmitting or not.

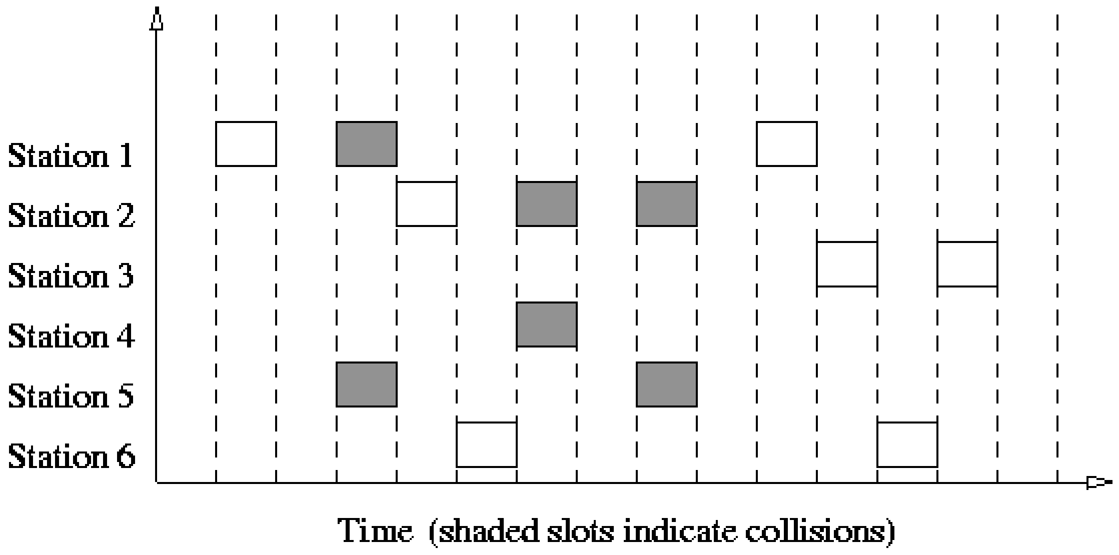


One serious drawback here is that, you dont know whether what you are sending has been received properly or not (so as to say, "whether you've been heard and understood?"). To resolve this, in Pure Aloha, when one node finishes speaking, it expects an acknowledgement in a finite amount of time - otherwise it simply retransmits the data. This scheme works well in small networks where the load is not high. But in large, load intensive networks where many nodes may want to transmit at the same time, this scheme fails miserably. This led to the development of Slotted Aloha.

**Slotted Aloha**

This is quite similar to Pure Aloha, differing only in the way transmissions take place. Instead of transmitting right at demand time, the sender waits for some time. This delay is specified as follows - the timeline is divided into equal slots and then it is required that transmission should take place only at slot boundaries. To be more precise, the slotted-Aloha makes the following assumptions:

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



In this way, the number of collisions that can possibly take place is reduced by a huge margin. And hence, the performance become much better compared to Pure Aloha. collisions may only take place with nodes that are ready to speak at the same time. But nevertheless, this is a substantial reduction.

**Carrier Sense Mutiple Access Protocols**

In both slotted and pure ALOHA, a node's decision to transmit is made independently of the activity of the other nodes attached to the broadcast channel. In particular, a node neither pays attention to whether another node happens to be transmitting when it begins to transmit, nor stops transmitting if another node begins to interfere with its transmission. As humans, we have human protocols that allow allows us to not only behave with more civility, but also to decrease the amount of time spent "colliding" with each other in conversation and consequently increasing the amount of data we exchange in our conversations. Specifically, there are two important rules for polite human conversation:

1. **Listen before speaking:** If someone else is speaking, wait until they are done. In the networking world, this is termed carrier sensing - a node listens to the channel before transmitting. If a frame from another node is currently being transmitted into the channel, a node then waits ("backs off") a random amount of time and then again senses the channel. If the channel is sensed to be idle, the node then begins frame transmission. Otherwise, the node waits another random amount of time and repeats this process.

2. **If someone else begins talking at the same time, stop talking.** In the networking world, this is termed collision detection - a transmitting node listens to the channel while it is transmitting. If it detects that another node is transmitting an interfering frame, it stops transmitting and uses some protocol to determine when it should next attempt to transmit.

It is evident that the end-to-end channel propagation delay of a broadcast channel - the time it takes for a signal to propagate from one of the the channel to another - will play a crucial role in determining its performance. The longer this propagation delay, the larger the chance that a carrier-sensing node is not yet able to sense a transmission that has already begun at another node in the network.

**CSMA- Carrier Sense Multiple Access**

This is the simplest version CSMA protocol as described above. It does not specify any collision detection or handling. So collisions might and WILL occur and clearly then, this is not a very good protocol for large, load intensive networks.

So, we need an improvement over CSMA - this led to the development of CSMA/CD.

**CSMA/CD- CSMA with Collision Detection**

In this protocol, while transmitting the data, the sender simultaneously tries to receive it. So, as soon as it detects a collission (it doesn't receive its own data) it stops transmitting. Thereafter, the node waits for some time interval before attempting to transmit again. Simply put, **"listen while you talk"**. But, how long should one wait for the carrier to be freed? There are three schemes to handle this:

1. **1-Persistent:** In this scheme, transmission proceeds immediately if the carrier is idle. However, if the carrier is busy, then sender continues to sense the carrier until it becomes idle. The main problem here is that, if more than one transmitters are ready to send, a collision is GUARANTEED!!

2. **Non-Persistent:** In this scheme, the broadcast channel is not monitored continuously. The sender polls it at random time intervals and transmits whenever the carrier is idle. This decreases the probability of collisions. But, it is not efficient in a low load situation, where number of collisions are anyway small. The problems it entails are:

    - If back-off time is too long, the idle time of carrier is wasted in some sense
    - It may result in long access delays

3. **p-Persistent:** Even if a sender finds the carrier to be idle, it uses a probabilistic distribution to determine whether to transmit or not. Put simply, "toss a coin to decide". If the carrier is idle, then transmission takes place with a probability p and the sender waits with a probability 1-p. This scheme is a good trade off between the Non-persistent and 1-persistent schemes. So, for low load situations, p is high (example: 1-persistent); and for high load situations, p may be lower. Clearly, the value of p plays an important role in determining the

performance of this protocol. Also the same p is likely to provide different performance at different loads.

CSMA/CD doesn't work in some wireless scenarios called **"hidden node"** problems. Consider a situation, where there are 3 nodes - A, B and C communicating with each other using a wireless protocol. Morover, B can communicate with both A and C, but A and C lie outside each other's range and hence can't communicate directly with each other. Now, suppose both A and C want to communicate with B simultaneously. They both will sense the carrier to be idle and hence will begin transmission, and even if there is a collision, neither A nor C will ever detect it. B on the other hand will receive 2 packets at the same time and might not be able to understand either of them. To get around this problem, a better version called CSMA/CA was developed, specially for wireless applications.

---

**Image References:**

- http://williams.comp.ncat.edu/Networks/multip10.gif
- http://williams.comp.ncat.edu/Networks/multip5.gif
- http://williams.comp.ncat.edu/Networks/multip13.gif
- http://www.e-networks.org/images/startopology.gif
- http://www.delmar.edu/Courses/CIS306/Primer/primf05.gif
- http://www.e-networks.org/images/ringtopology.gif
- http://www.laynetworks.com/images/aloha14.gif
- http://www.laynetworks.com/images/aloha15.gif

---

back to top
Prev | Next | Index

**Computer Networks (CS425)**

**Instructor: Dr. Dheeraj Sanghi**

Prev | Next | Index

---

## CSMA with Collision Avoidance

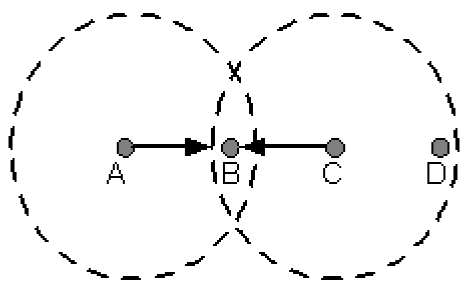
We have observed that CSMA/CD would break down in wireless networks because of hidden node and exposed nodes problems. We will have a quick recap of these two problems through examples.

### Hidden Node Problem

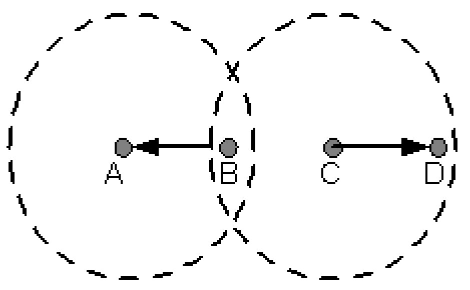
In the case of wireless network it is possible that A is sending a message to B, but C is out of its range and hence while "listening" on the network it will find the network to be free and might try to send packets to B at the same time as A. So, there will be a collision at B. The problem can be looked upon as if A and C are hidden from each other. Hence it is called the "hidden node problem".

**Exposed Node Problem**

If C is transmitting a message to D and B wants to transmit a message to A, B will find the network to be busy as B hears C trnasmitting. Even if B would have transmitted to A, it would not have been a problem at A or D. CSMA/CD would not allow it to transmit message to A, while the two transmissions could have gone in parallel.



Hidden node problem



Exposed node problem

**Addressing hidden node problem (CSMA/CA)**

Consider the figure above.Suppose A wants to send a packet to B. Then it will first send a small packet to B called **"Request to Send" (RTS)**. In response, B sends a small packet to A called **"Clear to Send" (CTS)**. Only after A receives a CTS, it transmits the actual data. Now, any of the nodes which can hear either CTS or RTS assume the network to be busy. Hence even if some other node which is out of range of both A and B sends an RTS to C (which can hear at least one of the RTS or CTS between A and B), C would not send a CTS to it and hence the communication would not be established between C and D.

One issue that needs to be addressed is how long the rest of the nodes should wait before they can transmit data over the network. The answer is that the RTS and CTS would carry some information about the size of the data that B intends to transfer. So, they can calculate time that would be required for the transmission to be over and assume the network to be free after that.Another interesting issue is what a node should do if it hears RTS but not a corresponding CTS. One possibility is that it assumes the recipient node has not responded and hence no transmission is going on, but there is a catch in this. It is possible that the node hearing RTS is just on the boundary of the node sending CTS. Hence, it does hear CTS but the signal is so deteriorated that it fails to recognize it as a CTS. Hence to be on the safer side, a node will not start transmission if it hears either of an RTS or a CTS.

The assumption made in this whole discussion is that if a node X can send packets to a node Y, it can also receive a packet from Y, which is a fair enough assumption given the fact that we are talking of a local network where standard instruments would be used. If that is not the case additional complexities would get introduced in the system.

**Does CSMA/CD work universally in the wired networks ?**

The problem of range is there in wired networks as well in the form of deterioration of signals. Normally to counter this, we use repeaters, which can regenerate the original signal from a deteriorated one. But does that mean that we can build as long networks as we want with repeaters. The answer, unfortunately, is NO! The reason is the beyond a certain length CSMA/CD will break down.

The mechanism of collision detection which CSMA/CD follows is through listening while talking. What this means is so long as a node is transmitting the packet, it is listening on the cable. If the data it listens to is different from the data it is transmitting it assumes a collision. Once it has stopped transmitting the packet, and has not detected collision while transmission was going on, it assumes that the transmission was successful. The problem arises when the distance between the two nodes is too large. Suppose A wants to transmit some packet to B which is at a very large distance from B. Data can travel on cable only at a finite speed (usually 2/3c, c being the speed of light). So, it is possible that the packet has been transmitted by A onto the cable but the first bit of the packet has not yet reached B. In that case, if a collision occurs, A would be unaware of it occurring. Therefore there is problem in too long a network.

Let us try to parametrize the above problem. Suppose "t" is the time taken for the node A to transmit the packet on the cable and "T" is the time , the packet takes to reach from A to B. Suppose transmission at A starts at time t0. In the worst case the collision takes place just when the first packet is to reach B. Say it is at t0+T-e (e being very small). Then the collision information will take T-e time to propagate back to A. So, at t0+2(T-e) A should still be transmitting. Hence, for the correct detection of collision (ignoring e)

**t > 2T**

t increases with the number of bits to be transferred and decreases with the rate of transfer (bits per second). T increases with the distance between the nodes and decreases with the speed of the signal (usually 2/3c). We need to either keep t large enough or T as small. We do not want to live with lower rate of bit transfer and hence slow networks. We can not do anything about the speed of the signal. So what we can rely on is the minimum size of the packet and the distance between the two nodes. Therefore, we fix some minimum size of the packet and if the size is smaller than that, we put in some extra bits to make it reach the minimum size. Accordingly we fix the maximum distance between the nodes. Here too, there is a tradeoff to be made. We do not want the minimum size of the packets to be too large since that wastes lots of resources on cable. At the same time we do not want the distance between the nodes to be too small. Typical minimum packet size is 64 bytes and the corresponding distance is 2-5 kilometers.

**Collision Free Protocols**

Although collisions do not occur with CSMA/CD once a station has unambigously seized the channel, they can still occur during the contention period. These collisions adversely affect the efficiency of transmission. Hence some protocols have been developed which are contention free.

**Bit-Map Method**

In this method, there N slots. If node 0 has a frame to send, it transmit a 1 bit during the first slot. No other node is allowed to transmit during this period. Next node 1 gets a chance to transmit 1 bit if it has something to send, regardless of what node 0 had transmitted. This is done for all the nodes. In general node j may declare the fact that it has a frsme to send by inserting a 1 into slot j. Hence after all nodes have passed, each node has complete knowledge of who wants to send a frame. Now they begin transmitting in numerical order. Since everyone knows who is transmitting and when, there could never be any collision.

The basic problem with this protocol is its inefficiency during low load. If a node has to transmit and no other node needs to do so, even then it has to wait for the bitmap to finish. Hence the bitmap will be repeated over and over again if very few nodes want to send wasting valuable bandwidth.

**Binary Countdown**

In this protocol, a node which wants to signal that it has a frame to send does so by writing its address into the header as a binary number. The arbitration is such that as soon as a node sees that a higher bit position that is 0 in its address has been overwritten with a 1, it gives up. The final result is the address of the node which is allowed to send. After the node has transmitted the whole process is repeated all over again. Given below is an example situation.

| **Nodes** | **Addresses** |
|---|---|
| A | 0010 |
| B | 0101 |
| C | 1010 |
| D | 1001 |
| | ---- |
| | 1010 |

Node C having higher priority gets to transmit. The problem with this protocol is that the nodes with higher address always wins. Hence this creates a priority which is highly unfair and hence undesirable.

**Limited Contention Protocols**

Both the type of protocols described above - Contention based and Contention - free has their own problems. Under conditions of light load, contention is preferable due to its low delay. As the load increases, contention becomes increasingly less attractive, because the overload associated with channel arbitration becomes greater. Just the reverse is true for contention - free protocols. At low load, they have high delay, but as the load increases , the channel efficiency improves rather than getting worse as it does for contention protocols.

Obviously it would be better if one could combine the best properties of the contention and contention - free protocols, that is, protocol which used contention at low loads to provide low delay, but used a cotention-free technique at high load to provide good channel efficiency. Such protocols do exist and are called Limited contention protocols.

It is obvious that the probablity of some station aquiring the channel could only be increased by decreasing the amount of competition. The limited contention protocols do exactly that. They first divide the stations up into ( not necessarily disjoint ) groups. Only the members of group 0 are permitted to compete for slot 0. The competition for aquiring the slot within a group is contention based. If one of the members of that group succeeds, it aquires the channel and transmits a frame. If there is collision or no node of a particular group wants to send then the members of the next group compete for the next slot. The probablity of a particular node is set to a particular value ( optimum ).

**Adaptive Tree Walk Protocol**

The following is the method of adaptive tree protocol. Initially all the nodes are allowed to try to aquire the channel. If it is able to aquire the channel, it sends its frame. If there is collision then the nodes are divided into two equal groups and only one of these groups compete for slot 1. If one of its member aquires the channel then the next slot is reserved for the other group. On the other hand, if there is a collision then that group is again subdivided and the same process is followed. This can be better understood if the nodes are thought of as being organised in a binary tree as shown in the following figure.

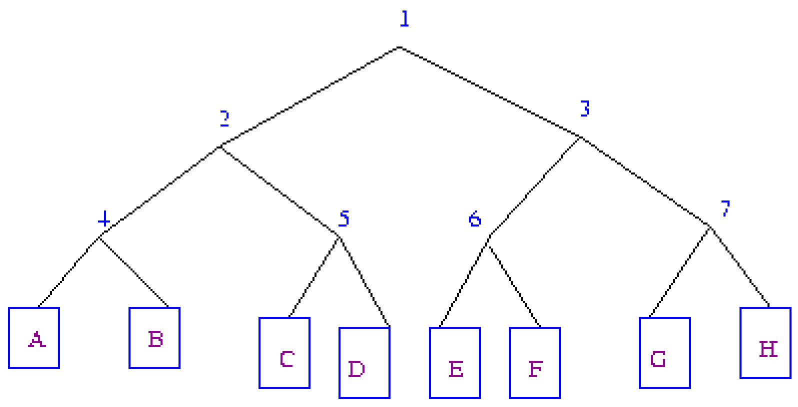


Fig. Adaptive Tree Walk abstraction of nodes in binary tree.

Many improvements could be made to the algorithm. For example, consider the case of nodes G and H being the only ones wanting to transmit. At slot 1 a collision will be detected and so 2 will be tried and it will be found to be idle. Hence it is pointless to probe 3 and one should directly go to 6,7.

---

**Image References:**

- http://www.ccs.neu.edu/course/csg150/Solutions-III_files/image002.gif

---

back to top
Prev| Next | Index

**Computer Networks (CS425)**

**Instructor: Dr. Dheeraj Sanghi**

Prev| Next| Index

**IEEE 802.3 and Ethernet**

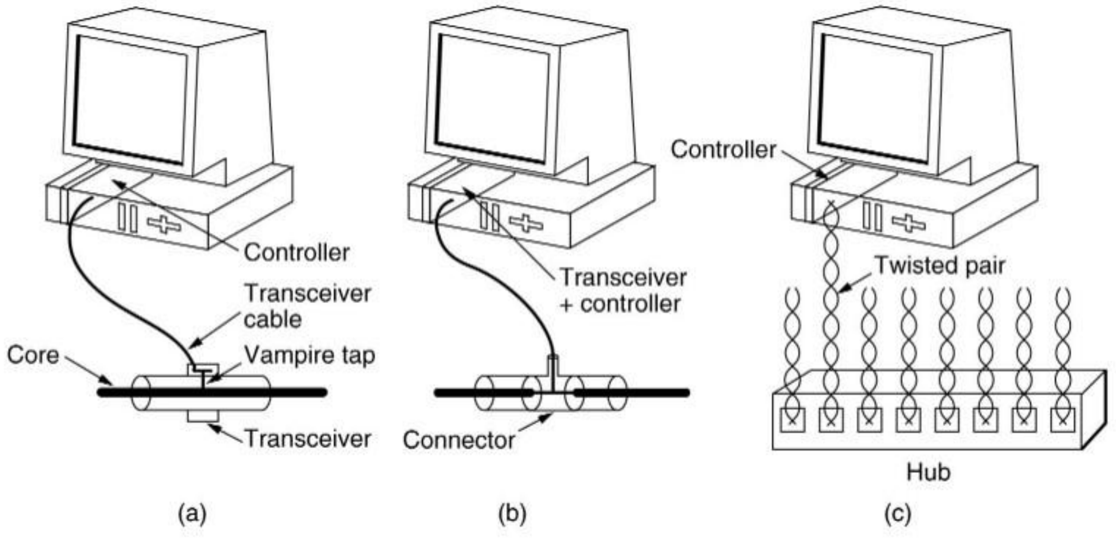
- Very popular LAN standard.
- Ethernet and IEEE 802.3 are distinct standards but as they are very similar to one another these words are used interchangeably.
- A standard for a 1-persistent CSMA/CD LAN.
- It covers the physical layer and MAC sublayer protocol.

**Ethernet Physical Layer**

A Comparison of Various Ethernet and IEEE 802.3 Physical-Layer Specifications

| **Characteristic** | **Ethernet Value** | **IEEE 802.3 Values** | | |
|---|---|---|---|---|
| | | **10Base5** | **10Base2** | **10BaseT** |
| Data rate (Mbps) | 10 | 10 | 10 | 10 |
| Signaling method | Baseband | Baseband | Baseband | Baseband |
| Maximum segment length (m) | 500 | 500 | 185 | 100 |
| Media | 50-ohm coax (thick) | 50-ohm coax (thick) | 50-ohm coax (thin) | Unshielded twisted-pai cable |
| Nodes/segment | 100 | 100 | 30 | 1024 |
| Topology | Bus | Bus | Bus | Star |

10Base5 means it operates at 10 Mbps, uses baseband signaling and can support segments of up to 500 meters. The 10Base5 cabling is popularly called the Thick Ethernet. Vampire taps are used for their connections where a pin is carefully forced halfway into the co-axial cable's core as shown in the figure below. The 10Base2 or Thin Ethernet bends easily and is connected using standard BNC connectors to form T junctions (shown in the figure below). In the 10Base-T scheme a different kind of wiring pattern is followed in which all stations have a twisted-pair cable running to a central hub (see below). The difference between the different physical connections is shown below:



**(a) 10Base5 (b)10Base2 (c)10Base-T**

All 802.3 baseband systems use Manchester encoding , which is a way for receivers to unambiguously determine the start, end or middle of each bit without reference to an external clock. There is a restriction on the minimum node spacing (segment length between two nodes) in 10Base5 and 10Base2 and that is 2.5 meter and 0.5 meter respectively. The reason is that if two nodes are closer than the specified limit then there will be very high current which may cause trouble in detection of signal at the receiver end. Connections from station to cable of 10Base5 (i.e. Thick Ethernet) are generally made using vampire taps and to 10Base2 (i.e. Thin Ethernet) are made using industry standard BNC connectors to form T junctions. To allow larger networks, multiple segments can be connected by repeaters as shown. A repeater is a physical layer device. It receives, amplifies and retransmits signals in either direction.

**Note:** To connect multiple segments, amplifier is not used because amplifier also amplifies the noise in the signal, whereas repeater regenerates signal after removing the noise.

**IEEE 802.3 Frame Structure**

| | | | | | | |
|---|---|---|---|---|---|---|
| **Preamble (7 bytes)** | **Start of Frame Delimiter (1 byte)** | **Dest. Address (2/6 bytes)** | **Source Address (2/6 bytes)** | **Length (2 bytes)** | **802.2 Header+Data (46-1500 bytes)** | **Frame Checksum (4 bytes)** |

***A brief description of each of the fields***

- **Preamble :**Each frame starts with a preamble of 7 bytes, each byte containing the bit pattern 10101010. Manchester encoding is employed here and this enables the receiver's clock to synchronize with the sender's and initialise itself.
- **Start of Frame Delimiter :**This field containing a byte sequence 10101011 denotes the start of the frame itself.
- **Dest. Address :**The standard allows 2-byte and 6-byte addresses. Note that the 2-byte addresses are always local addresses while the 6-byte ones can be local or global.

*2-Byte Address - Manually assigned address*

| Individual(0)/Group(1) (1 bit) | Address of the machine (15 bits) |
|---|---|

- *6-Byte Address - Every Ethernet card with globally unique address*

| Individual(0)/Group(1) (1 bit) | Universal(0)/Local(1) (1 bit) | Address of the machine (46 bits) |
|---|---|---|

- **Multicast :** Sending to group of stations. This is ensured by setting the first bit in either 2-byte/6-byte addresses to 1.
**Broadcast :** Sending to all stations. This can be done by setting all bits in the address field to 1.All Ethernet cards(Nodes) are a member of this group.

- **Source Address :**Refer to Dest. Address. Same holds true over here.
- **Length :** The Length field tells how many bytes are present in the data field, from a minimum of 0 to a maximum of 1500. The Data and padding together can be from 46bytes to 1500 bytes as the valid frames must be at least 64 bytes long, thus if data is less than 46 bytes the amount of padding can be found out by length field.
- **Data :**Actually this field can be split up into two parts - Data(0-1500 bytes) and Padding(0-46 bytes).
*Reasons for having a minimum length frame :*

  1. To prevent a station from completing the transmission of a short frame before the first bit has even reached the far end of the cable, where it may collide with another frame. Note that the transmission time ought to be greater than twice the propagation time between two farthest nodes.

**transmission time for frame > 2*propagation time between two farthest nodes**

  2. When a transceiver detects a collision, it truncates the current frame, which implies that stray bits and pieces of frames appear on the cable all the time. Hence to distinguish between valid frames from garbage, 802.3 states that the minimum length of valid frames ought to be 64 bytes (from Dest. Address to Frame Checksum).

- **Frame Checksum :** It is a 32-bit hash code of the data. If some bits are erroneously received by the destination (due to noise on the cable), the checksum computed by the destination wouldn't match with the checksum sent and therefore the error will be detected. The checksum algorithm is a cyclic redundancy checksum (CRC) kind. The checksum includes the packet from Dest. Address to Data field.

**Ethernet Frame Structure**

| Preamble (8 bytes) | Dest. Address (2/6 bytes) | Source Address (2/6 bytes) | Type (2 bytes) | Data (46-1500 bytes) | Frame Checksum (4 bytes) |
|---|---|---|---|---|---|

***A brief description of the fields which differ from IEEE 802.3***

- **Preamble :**The *Preamble* and *Start of Frame Delimiter* are merged into one in Ethernet standard. However, the contents of the first 8 bytes remains the same in both.
- **Type :**The length field of IEEE 802.3 is replaced by Type field, which denotes the type of packet being sent viz. IP, ARP, RARP, etc. If the field indicates a value less than 1500 bytes then it is length field of 802.3 else it is the type field of Ethernet packet.

**Truncated Binary Exponential Back off**

In case of collision the node transmitting backs off by a random number of slots , each slot time being equal to transmission time of 512 bits (64 Byte- minimum size of a packet) in the following fashion:

| No of Collision | Random No of slots |
|---|---|
| 1st | 0-1 |
| 2nd | 0-3 |
| 3rd | 0-7 |
| \| | \| |
| \| | \| |
| 10th | 0-1023 |
| ------------------------------------------- | |
| 11th | 0-1023 |
| 12th | 0-1023 |
| \| | \| |
| 16th | 0-1023 |

In general after i collisions a random number between 0-2^i-1 is chosen , and that number of slots is skipped. However, after 10 collisions have been reached the randomization interval is frozen at maximum of 1023 slots. After 16 collisions the controller reports failure back to the computer.

**5-4-3 Rule**

Each version of 802.3 has a maximum cable length per segment because long propagation time leads to difficulty in collision detection. To compensate for this the transmission time has to be increased which can be achieved by slowing down the transmission rate or increasing the packet size, neither of which is desirable. Hence to allow for large networks, multiple cables are connected via **repeaters**. Between any two nodes on an Ethernet network, there can be at most five segments, four repeaters and three populated segments (non-populated segments are those which do not have any machine connected between the two repeaters). This is known as the **5-4-3 Rule**.

---

**Image References:**

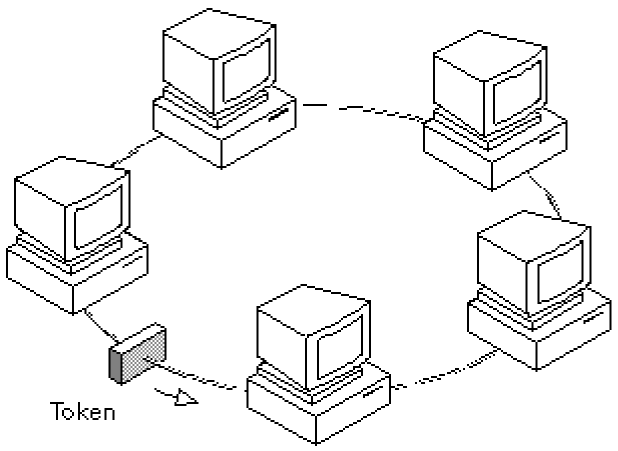
- http://homepages.ius.edu/rwisman/b438/Html/4-14.jpg

**Computer Networks (CS425)**

**Instructor: Dr. Dheeraj Sanghi**

**IEEE 802.5: Token Ring Network**

- Token Ring is formed by the nodes connected in ring format as shown in the diagram below. The principle used in the token ring network is that a token is circulating in the ring and whichever node grabs that token will have right to transmit the data.
- Whenever a station wants to transmit a frame it inverts a single bit of the 3-byte token which instantaneously changes it into a normal data packet. Because there is only one token, there can atmost be one transmission at a time.
- Since the token rotates in the ring it is guarenteed that every node gets the token with in some specified time. So there is an upper bound on the time of waiting to grab the token so that starvation is avoided.
- There is also an upper limit of 250 on the number of nodes in the network.
- To distinguish the normal data packets from token (control packet) a special sequence is assigned to the token packet. When any node gets the token it first sends the data it wants to send, then recirculates the token.



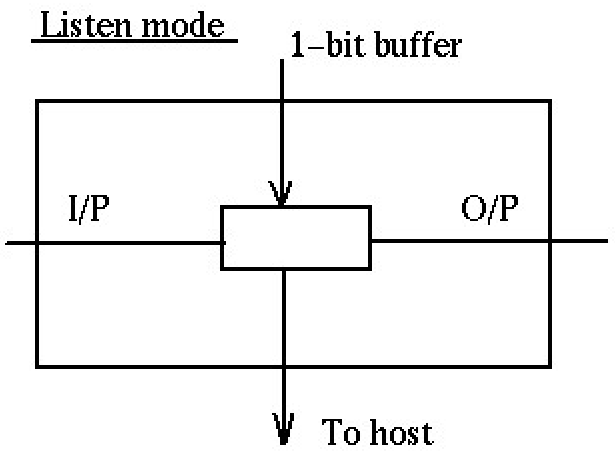
If a node transmits the token and nobody wants to send the data the token comes back to the sender. If the first bit of the token reaches the sender before the transmission of the last bit, then error situation araises. So to avoid this we should have:

**propogation delay + transmission of n-bits (1-bit delay in each node ) > transmission of the token time**

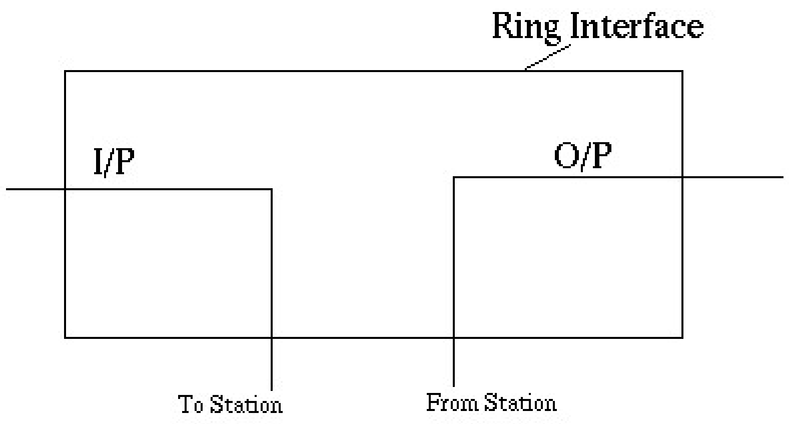
A station may hold the token for the token-holding time. which is 10 ms unless the installation sets a different value. If there is enough time left after the first frame has been transmitted to send more frames, then these frames may be sent as well. After all pending frames have been transmitted or the transmission frame would exceed the token-holding time, the station regenerates the 3-byte token frame and puts it back on the ring.

**Modes of Operation**

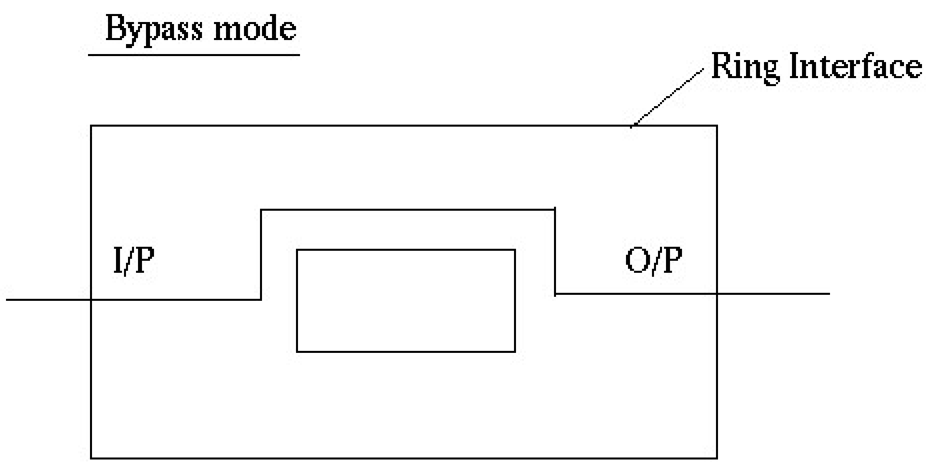
1. **Listen Mode:** In this mode the node listens to the data and transmits the data to the next node. In this mode there is a one-bit delay associated with the transmission.



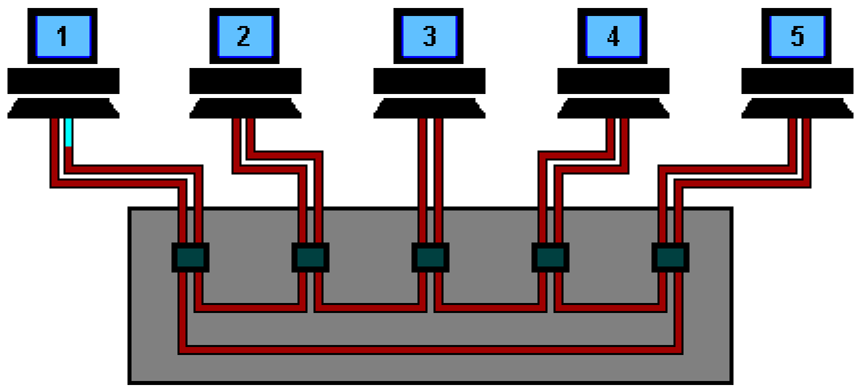
2. **Transmit Mode:** In this mode the node just discards the any data and puts the data onto the network.



3. **By-pass Mode:** In this mode reached when the node is down. Any data is just bypassed. There is no one-bit delay in this mode.



**Token Ring Using Ring Concentrator**



One problem with a ring network is that if the cable breaks somewhere, the ring dies. This problem is elegantly addressed by using a ring concentrator. A Token Ring concentrator simply changes the topology from a physical ring to a star wired ring. But the network still remains a ring logically. Physically, each station is connected to the ring concentrator (wire center) by a cable containing at least two twisted pairs, one for data to the station and one for data from the station. The Token still circulates around the network and is still controlled in the same manner, however, using a hub or a switch greatly improves reliability because the hub can automatically bypass any ports that are disconnected or have a cabling fault. This is done by having bypass relays inside the concentrator that are energized by current from the stations. If the ring breaks or station goes down, loss of the drive current will release the relay and bypass the station. The ring can then continue operation with the bad segment bypassed.

**Who should remove the packet from the ring ?**

There are 3 possibilities-

1. **The source itself removes the packet after one full round in the ring**.

2. **The destination removes it after accepting it**: This has two potential problems. Firstly, the solution won't work for broadcast or multicast, and secondly, there would be no way to acknowledge the sender about the receipt of the packet.

3. **Have a specialized node only to discard packets**: This is a bad solution as the specialized node would know that the packet has been received by the destination only when it receives the packet the second time and by that time the packet may have actually made about one and half (or almost two in the worst case) rounds in the ring.

Thus the first solution is adopted with the source itself removing the packet from the ring after a full one round. With this scheme, broadcasting and multicasting can be handled as well as the destination can acknowledge the source about the receipt of the packet (or can tell the source about some error).

**Token Format**

The token is the shortest frame transmitted (24 bit)
MSB (Most Significant Bit) is always transmitted first - as opposed to Ethernet

| | | |
|---|---|---|
| SD | AC | ED |

**SD = Starting Delimiter (1 Octet)**
**AC = Access Control (1 Octet)**
**ED = Ending Delimiter (1 Octet)**

**Starting Delimiter Format:**

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| J | K | O | J | K | O | O | O |

**J = Code Violation**
**K = Code Violation**

**Access Control Format:**

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| P | P | P | T | M | R | R | R |

**T=Token**
T = 0  for Token
T = 1  for Frame
When a station with a Frame to transmit detects a token which has a priority equal to or less than the Frame to be transmitted, it may change the token to a start-of-frame sequence and transmit the Frame

**P = Priority**
Priority Bits indicate tokens priority, and therefore, which stations are allowed to use it. Station can transmit if its priority as at least as high as that of the token.

**M = Monitor**
The monitor bit is used to prevent a token whose priority is greater than 0 or any frame from continuously circulating on the ring. If an active monitor detects a frame or a high priority token with the monitor bit equal to 1, the frame or token is aborted. This bit shall be transmitted as 0 in all

frame and tokens. The active monitor inspects and modifies this bit. All other stations shall repeat this bit as received.

**R = Reserved bits**
The reserved bits allow station with high priority Frames to request that the next token be issued at the requested priority.

**Ending Delimiter Format:**

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| J | K | 1 | J | K | 1 | 1 | E |

**J = Code Violation**
**K = Code Violation**
**I = Intermediate Frame Bit**
**E = Error Detected Bit**

**Frame Format:**

MSB (Most Significant Bit) is always transmitted first - as opposed to Ethernet

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| SD | AC | FC | DA | SA | DATA | CRC | ED | FS |

**SD=Starting Delimiter(1 octet)**
**AC=Access Control(1 octet)**
**FC = Frame Control (1 Octet)**
**DA = Destination Address (2 or 6 Octets)**
**SA = Source Address (2 or 6 Octets)**
**DATA = Information 0 or more octets up to 4027**
**CRC = Checksum(4 Octets)**
**ED = Ending Delimiter (1 Octet)**
**FS=Frame Status**

**Starting Delimiter Format:**

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| J | K | 0 | J | K | 0 | 0 | 0 |

**J = Code Violation**
**K = Code Violation**

**Access Control Format:**

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| P | P | P | T | M | R | R | R |

**T=Token**

T = “0” for Token,
T = “1” for Frame.

When a station with a Frame to transmit detects a token which has a priority equal to or less than the Frame to be transmitted, it may change the token to a start-of-frame sequence and transmit the Frame.

**P = Priority**
Bits Priority Bits indicate tokens priority, and therefore, which stations are allowed to use it. Station can transmit if its priority as at least as high as that of the token.

**M = Monitor**
The monitor bit is used to prevent a token whose priority is greater than 0 or any frame from continuously circulating on the ring. if an active monitor detects a frame or a high priority token with the monitor bit equal to 1, the frame or token is aborted. This bit shall be transmitted as 0 in all frame and tokens. The active monitor inspects and modifies this bit. All other stations shall repeat this bit as received.

**R = Reserved bits** the reserved bits allow station with high priority Frames to request that the next token be issued at the requested priority

**Frame Control Format:**

| | | |
|---|---|---|
| F | F | CONTROL BITS (6 BITS) |

**FF= Type of Packet-Regular data packet or MAC layer packet**
**Control Bits= Used if the packet is for MAC layer protocol itself**

**Source and Destination Address Format:**

The addresses can be of 2 bytes (local address) or 6 bytes (global address).

**local address format:**

| | |
|---|---|
| I/G (1 BIT) | NODE ADDRESS (15 BITS) |

alternatively

| | | |
|---|---|---|
| I/G (1 BIT) | RING ADDRESS (7 BITS) | NODE ADDRESS (8 BITS) |

The first bit specifies individual or group address.

**universal (global) address format:**

| | | | |
|---|---|---|---|
| I/G (1 BIT) | L/U (1 BIT) | RING ADDRESS (14 BITS) | NODE ADDRESS (32 BITS) |

The first bit specifies individual or group address.
The second bit specifies local or global (universal) address.

**local group addresses (16 bits):**

| | | |
|---|---|---|
| I/G (1 BIT) | T/B(1 BIT) | GROUP ADDRESS (14 BITS) |

The first bit specifies an individual or group address.
The second bit specifies traditional or bit signature group address.

**Traditional Group Address:** 2Exp14 groups can be defined.
**Bit Signature Group Address:** 14 grtoups are defined. A host can be a member of none or any number of them. For multicasting, those group bits are set to which the packet should go. For

broadcasting, all 14 bits are set. A host receives a packet only if it is a member of a group whose corresponding bit is set to 1.

**universal group addresses (16 bits):**

| I/G (1 BIT) | RING NUMBER | T/B (1 BIT) | GROUP ADDRESS (14 BITS) |
|---|---|---|---|

The description is similar to as above.

**Data Format:**

No upper limit on amount of data as such, but it is limited by the token holding time.

**Checksum:**

The source computes and sets this value. Destination too calculates this value. If the two are different, it indicates an error, otherwise the data may be correct.

**Frame Status:**

It contains the A and C bits.

**A bit set to 1: destination recognized the packet.**
**C bit set to 1: destination accepted the packet.**

This arrangement provides an automatic acknowledgement for each frame. The A and C bits are present twice in the Frame Status to increase reliability in as much as they are not covered by the checksum.

**Ending Delimiter Format:**

| J | K | 1 | J | K | 1 | I | E |
|---|---|---|---|---|---|---|---|

**J = Code Violation**
**K = Code Violation**
**I = Intermediate Frame Bit**
If this bit is set to 1, it indicates that this packet is an intermediate part of a bigger packet, the last packet would have this bit set to 0.
**E = Error Detected Bit**
This bit is set if any interface detects an error.

This concludes our description of the token ring frame format.

---

**Image References:**

- http://www.webopedia.com/FIG/RING.gif
- http://www.datacottage.com/nch/anigifs/trhubani.gif

---

back to top
Prev | Next | Index

---

**Token Ring Network (Contd...)**

**Phase Jitter Compensation :**

In a token ring the source starts discarding all it's previously transmitted bits as soon as they circumnavigate the ring and reach the source. Hence, it's not desirable that while a token is being sent some bits of the token which have already been sent become available at the incoming end of the source. This behavior though is desirable in case of data packets which ought to be drained from the ring once they have gone around the ring. To achieve the aforesaid behavior with respect to tokens, we would like the ring to hold at least 24 bits at a time. How do we ensure this?

Each node in a ring introduces a 1 bit delay. So, one approach might be to set the minimum limit on the number of nodes in a ring as 24. But, this is not a viable option. The actual solution is as follows. We have one node in the ring designated as **"monitor"**. The monitor maintains a 24 bits buffer with help of which it introduces a 24 bit delay. The catch here is what if the clocks of nodes following the source are faster than the source? In this case the 24 bit delay of the monitor would be less than the 24 bit delay desired by the host. To avoid this situation the monitor maintains 3 extra bits to compensate for the faster bits. The 3 extra bits suffice even if bits are 10 % faster. This compensation is called Phase Jitter Compensation.

**Handling multiple priority frames**

Each node or packet has a priority level. We don't concern ourselves with how this priority is decided. The first 3 bits of the Access Control byte in the token are for priority and the last 3 are for reservation.

P P P T M R R R

Initially the reservation bits are set to 000. When a node wants to transmit a priority n frame, it must wait until it can capture a token whose priority is less than or equal to n. Furthermore, when a data frame goes by, a station can try to reserve the next token by writing the priority of the frame it wants to send into the frame's Reservation bits. However, if a higher priority has already been reserved there, the station cannot make a reservation. When the current frame is finished, the next token is generated at the priority that has been reserved.

A slight problem with the above reservation procedure is that the reservation priority keeps on increasing. To solve this problem, the station raising the priority remembers the reservation priority that it replaces and when it is done it reduces the priority to the previous priority.

Note that in a token ring, low priority frames may starve.

**Ring Maintenance**

Each token ring has a monitor that oversees the ring. Among the monitor's responsibilities are seeing that the token is not lost, taking action when the ring breaks, cleaning the ring when garbled frames appear and watching out for orphan frames. An orphan frame occurs when a station transmits a short frame in it's entirety  onto a long ring and then crashes or is powered down before the frame can be removed. If nothing is done, the frame circulates indefinitely.

- **Detection of orphan frames:** The monitor detects orphan frames by setting the monitor bit in the Access Control byte whenever it passes through. If an incoming frame has this bit set, something is wrong since the same frame has passed the monitor twice. Evidently it was not removed by the source, so the monitor drains it.
- **Lost Tokens:** The monitor has a timer that is set to the longest possible tokenless interval : when each node transmits for the full token holding time. If this timer goes off, the monitor drains the ring and issues a fresh token.
- **Garbled frames:** The monitor can detect such frames by their invalid format or checksum, drain the ring and issue a fresh token.

The token ring control frames for maintenance are:

| *Control field* | *Name* | *Meaning* |
|---|---|---|
| 00000000 | Duplicate address test | Test if two stations have the same address |
| 00000010 | Beacon | Used to locate breaks in the ring |
| 00000011 | Claim token | Attempt to become monitor |
| 00000100 | Purge | Reinitialize the ring |
| 00000101 | Active monitor present | Issued periodically by the monitor |
| 00000110 | Standby monitor present | Announces the presence of potential monitors |

The monitor periodically issues a message "Active Monitor Present" informing all nodes of its presence. When this message is not received for a specific time interval, the nodes detect a monitor failure. Each node that believes it can function as a monitor broadcasts a "Standby Monitor Present" message at regular intervals, indicating that it is ready to take on the monitor's job. Any node that detects failure of a monitor issues a "Claim" token. There are 3 possible outcomes :

1. If the issuing node gets back its own claim token, then it becomes the monitor.
2. If a packet different from a claim token is received, apparently a wrong guess of monitor failure was made. In this case on receipt of our own claim token, we discard it. Note that our claim token may have been removed by some other node which has detected this error.
3. If some other node has also issued a claim token, then the node with the larger address becomes the monitor.

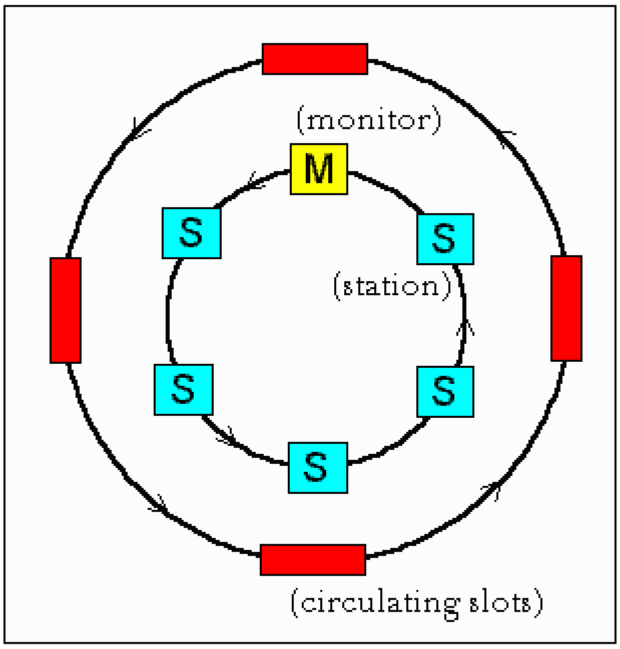
In order to resolve errors of duplicate addresses, whenever a node comes up it sends a **"Duplicate Address Detection"** message (with the destination = source) across the network. If the address recognize bit has been set on receipt of the message, the issuing node realizes a duplicate address and goes to standby mode. A node informs other nodes of removal of a packet from the ring through a **"Purge"** message. One maintenance function that the monitor cannot handle is locating breaks in the ring. If there is no activity detected in the ring (e.g. Failure of monitor to issue the **Active Monitor Present** token...) , the usual procedures of sending a claim token are followed. If the claim token itself is not received besides packets of any other kind, the node then sends **"Beacons"** at regular intervals until a message is received indicating that the broken ring has been repaired.

**Other Ring Networks**

The problem with the token ring system is that large rings cause large delays. It must be made possible for multiple packets to be in the ring simultaneously. The following ring networks resolve this problem to some extent :-

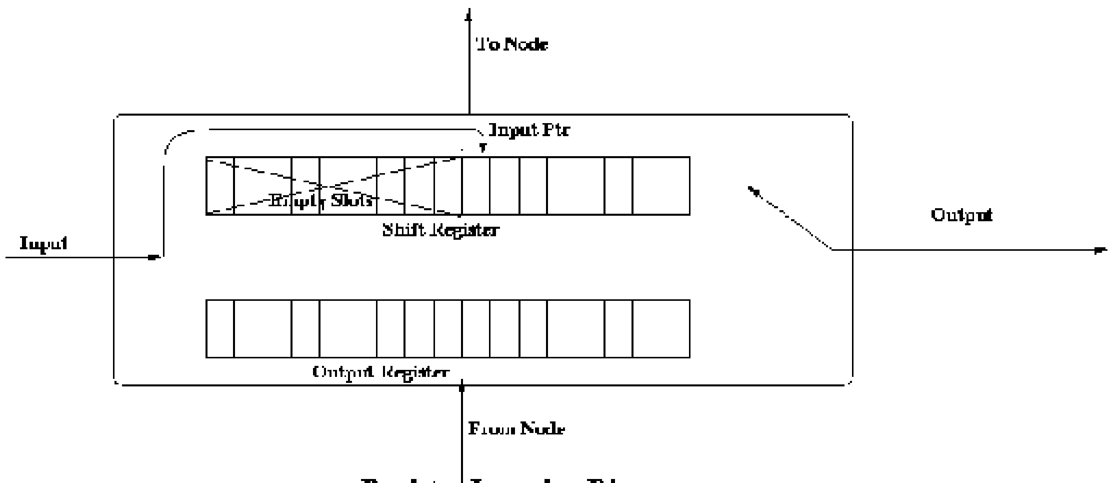
**Slotted Ring :**

In this system, the ring is slotted into a number of fixed size frames which are continuously moving around the ring. This makes it necessary that there be enough number of nodes (large ring size) to ensure that all the bits can stay on the ring at the same time. The frame header contains information as to whether the slots are empty or full. The usual disadvantages of overhead/wastage associated with fixed size frames are present.



**Register Insertion Rings :**

This is an improvement over slotted ring architecture. The network interface consists of two registers : a shift register and an output buffer. At startup, the input pointer points to the rightmost bit position in the input shift register .When a bit arrives it is in the rightmost empty position (the one indicated by the input pointer). After the node has detected that the frame is not addressed to it, the bits are transmitted one at time (by shifting). As new bits come in, they are inserted at  the position indicated by the pointer and then the contents are shifted. Thus the pointer is not moved. Once the shift register has pushed out the last bit of a frame, it checks to see if it has an output frame waiting. In case yes, then it checks that if the number of empty slots in the shift register is at least equal to the number of bits in the output frame. After this the output connection is switched to this second register and after the register has emptied its contents, the output line is switched back to the shift register. Thus, no single node can hog the bandwidth. In a loaded system, a node can transmit a k-bit frame only if it has saved up a k-bits of inter frame gaps.



Register Insertion Ring

Two major disadvantages of this topology are complicated hardware and difficulty in the detection of start/end of packets.

**Contention Ring**

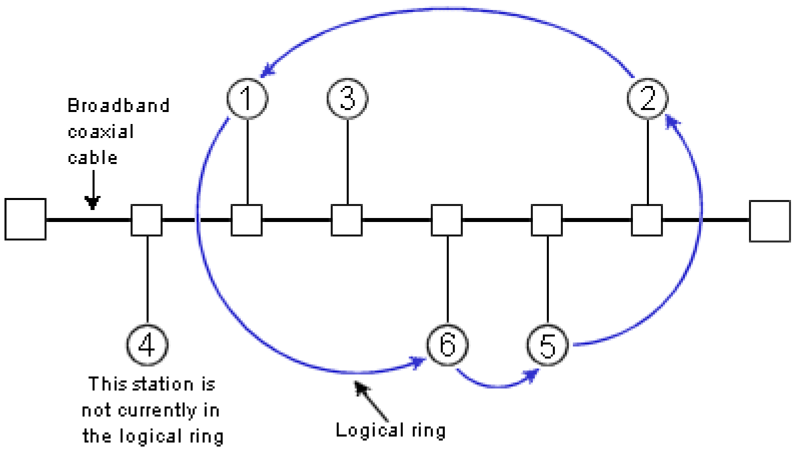
The token ring has primarily two problems:

- On light loads, huge overhead is incurred for token passing.
- Nodes with low priority data may starve if there is always a node with high priority data.

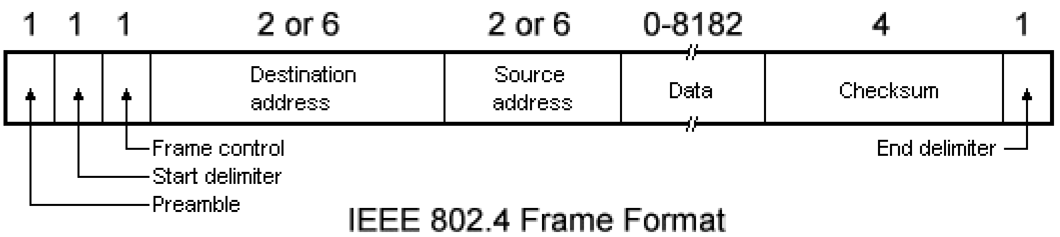
A contention ring attempts to address these problems. In a contention ring, if there is no communication in the ring for a while, a sender node will send its data immediately, followed by a token. If the token comes back to the sender without any data packet in between, the sender removes it from the ring. However under heavy load the behavior is that of a normal token ring. In case a collision, each of the sending nodes will remove the others' data packet from the ring, back off for a random period of time and then resend their data.

**IEEE 802.4: Token Bus Network**

In this system, the nodes are physically connected as a bus, but logically form a ring with tokens passed around to determine the turns for sending. It has the robustness of the 802.3 broadcast cable and the known worst case behavior of a ring. The structure of a token bus network is as follows:



**Frame Structure**



IEEE 802.4 Frame Format

A 802.4 frame has the following fields:

- Preamble: The Preamble is used to synchronize the receiver's clock.
- Starting Delimiter (SD) and End Delimiter (ED): The Starting Delimiter and Ending Delimiter fields are used to mark frame boundaries. Both of them contain analog encoding of symbols other than 1 or 0 so that they cannot occur accidentally in the user data. Hence no length field is needed.
- Frame Control (FC): This field is used to distinguish data frames from control frames. For data frames, it carries the frame's priority as well as a bit which the destination can set as an acknowledgement. For control frames, the Frame Control field is used to specify the frame type. The allowed types include token passing and various ring maintenance frames.
- Destination and Source Address: The Destination and Source address fields may be 2 bytes (for a local address) or 6 bytes (for a global address).
- Data: The Data field carries the actual data and it may be 8182 bytes when 2 byte addresses are used and 8174 bytes for 6 byte addresses.
- Checksum: A 4-byte checksum calculated for the data. Used in error detection.

**Ring Maintenance:**

**Mechanism:**

When the first node on the token bus comes up, it sends a **Claim_token** packet to initialize the ring. If more than one station send this packet at the same time, there is a collision. Collision is resolved by a contention mechanism, in which the contending nodes send random data for 1, 2, 3 and 4 units of time depending on the first two bits of their address. The node sending data for the longest time wins. If two nodes have the same first two bits in their addresses, then contention is done again based on the next two bits of their address and so on.

After the ring is set up, new nodes which are powered up may wish to join the ring. For this a node sends **Solicit_successor_1** packets from time to time, inviting bids from new nodes to join the ring. This packet contains the address of the current node and its current successor, and asks for nodes in between these two addresses to reply. If more than one nodes respond, there will be collision. The node then sends a **Resolve_contention** packet, and the contention is resolved using a similar mechanism as described previously. Thus at a time only one node gets to enter the ring. The last node in the ring will send a **Solicit_successor_2** packet containing the addresses of it and its successor. This packet asks nodes not having addresses in between these two addresses to respond.

A question arises that how frequently should a node send a Solicit_successor packet? If it is sent too frequently, then overhead will be too high. Again if it is sent too rarely, nodes will have to wait for a long time before joining the ring. If the channel is not busy, a node will send a Solicit_successor packet after a fixed number of token rotations. This number can be configured by the network administrator. However if there is heavy traffic in the network, then a node would defer the sending of bids for successors to join in.

There may be problems in the logical ring due to sudden failure of a node. What happens when a node goes down along with the token? After passing the token, a node, say node A, listens to the channel to see if its successor either transmits the token or passes a frame. If neither happens, it resends a token. Still if nothing happens, A sends a **Who_follows** packet, containing the address of the down node. The successor of the down node, say node C, will now respond with a **Set_successor** packet, containing its own address. This causes A to set its successor node to C, and the logical ring is restored. However, if two successive nodes go down suddenly, the ring will be dead and will have to be built afresh, starting from a **Claim_token** packet.

When a node wants to shutdown normally, it sends a **Set_successor** packet to its predecessor, naming its own successor. The ring then continues unbroken, and the node goes out of the ring.

The various control frames used for ring maintenance are shown below:

| **Frame Control Field** | **Name** | **Meaning** |
|---|---|---|
| 00000000 | Claim_token | Claim token during ring maintenance |
| 00000001 | Solicit_successor_1 | Allow stations to enter the ring |
| 00000010 | Solicit_successor_2 | Allow stations to enter the ring |
| 00000011 | Who_follows | Recover from lost token |
| 00000100 | Resolve_contention | Used when multiple stations want to enter |
| 00001000 | Token | Pass the token |
| 00001100 | Set_successor | Allow the stations leave the ring |

**Priority Scheme:**

Token bus supports four distinct priority levels: 0, 2, 4 and 6.

0 is the lowest priority level and 6 the highest. The following times are defined by the token bus:

- THT: Token Holding Time. A node holding the token can send priority 6 data for a maximum of this amount of time.
- TRT_4: Token Rotation Time for class 4 data. This is the maximum time a token can take to circulate and still allow transmission of class 4 data.
- TRT_2 and TRT_0: Similar to TRT_4.

When a station receives data, it proceeds in the following manner:

- It transmits priority 6 data for at most THT time, or as long as it has data.
- Now if the time for the token to come back to it is less than TRT_4, it will transmit priority 4 data, and for the amount of time allowed by TRT_4. Therefore the maximum time for which it can send priority 4 data is= Actual TRT - THT - TRT_4
- Similarly for priority 2 and priority 0 data.

This mechanism ensures that priority 6 data is always sent, making the system suitable for real time data transmission. In fact this was one of the primary aims in the design of token bus.

---

**Image References:**

- https://www.cis.strath.ac.uk/teaching/ug/classes/52.354/Images/SlottedRing.gif
- http://www.lkn.ei.tum.de/lehre/mmprog/mac/protocols/token-bus/pics/bus1.gif
- http://www.acerimmeronline.com/networks/images/token_bus_frame.gif

---

back to top
Prev| Next | Index

**Computer Networks (CS425)**

**Instructor: Dr. Dheeraj Sanghi**

Prev| Next| Index

---

**Data Link Layer**

**What is DLL(Data Link Layer)**

The Data Link Layer is the second layer in the OSI model, above the Physical Layer, which ensures that the error free data is transferred between the adjacent nodes in the network. It breaks the datagrams passed down by above layers and convert them into frames ready for transfer. This is called Framing. It provides two main functionalities

- Reliable data transfer service between two peer network layers
- Flow Control mechanism which regulates the flow of frames such that data congestion is not there at slow receivers due to fast senders.

**What is Framing?**

Since the physical layer merely accepts and transmits a stream of bits without any regard to meaning or structure, it is upto the data link layer to create and recognize frame boundaries. This can be accomplished by attaching special bit patterns to the beginning and end of the frame. If these bit patterns can accidentally occur in data, special care must be taken to make sure these patterns are not incorrectly interpreted as frame delimiters. The four framing methods that are widely used are

- Character count
- Starting and ending characters, with character stuffing
- Starting and ending flags, with bit stuffing
- Physical layer coding violations

**Character Count**

This method uses a field in the header to specify the number of characters in the frame. When the data link layer at the destination sees the character count,it knows how many characters follow, and hence where the end of the frame is. The disadvantage is that if the count is garbled by a transmission error, the destination will lose synchronization and will be unable to locate the start of the next frame. So, this method is rarely used.

**Character stuffing**

In the second method, each frame starts with the ASCII character sequence DLE STX and ends with the sequence DLE ETX.(where DLE is Data Link Escape, STX is Start of TeXt and ETX is End of TeXt.) This method overcomes the drawbacks of the character count method. If the destination ever loses synchronization, it only has to look for DLE STX and DLE ETX characters. If however, binary data is being transmitted then there exists a possibility of the characters DLE STX and DLE ETX occurring in the data. Since this can interfere with the framing, a technique called character stuffing is used. The sender's data link layer inserts an ASCII DLE character just before the DLE character in the data. The receiver's data link layer removes this DLE before this data is given to the network layer. However character stuffing is closely associated with 8-bit characters and this is a major hurdle in transmitting arbitrary sized characters.

**Bit stuffing**

The third method allows data frames to contain an arbitrary number of bits and allows character codes with an arbitrary number of bits per character. At the start and end of each frame is a flag byte consisting of the special bit pattern 01111110 . Whenever the sender's data link layer encounters five consecutive 1s in the data, it automatically stuffs a zero bit into the outgoing bit stream. This technique is called bit stuffing. When the receiver sees five consecutive 1s in the incoming data stream, followed by a zero bit, it automatically destuffs the 0 bit. The boundary between two frames can be determined by locating the flag pattern.

**Physical layer coding violations**

The final framing method is physical layer coding violations and is applicable to networks in which the encoding on the physical medium contains some redundancy. In such cases normally, a 1 bit is a high-low pair and a 0 bit is a low-high pair. The combinations of low-low and high-high which are not used for data may be used for marking frame boundaries.

**Error Control**

The bit stream transmitted by the physical layer is not guaranteed to be error free. The data link layer is responsible for error detection and correction. The most common error control method is to compute and append some form of a checksum to each outgoing frame at the sender's data link layer and to recompute the checksum and verify it with the received checksum at the receiver's side. If both of them match, then the frame is correctly received; else it is erroneous. The checksums may be of two types:
# Error detecting : Receiver can only detect the error in the frame and inform the sender about it. # Error detecting and correcting : The receiver can not only detect the error but also correct it.
Examples of Error Detecting methods:

- **Parity bit:**
  Simple example of error detection technique is parity bit. The parity bit is chosen that the number of 1 bits in the code word is either even( for even parity) or odd (for odd parity). For example when 10110101 is transmitted then for even parity an 1 will be appended to the data and for odd parity a 0 will be appended. This scheme can detect only single bits. So if two or more bits are changed then that can not be detected.

- **Longitudinal Redundancy Checksum:**
  Longitudinal Redundancy Checksum is an error detecting scheme which overcomes the problem of two erroneous bits. In this conceptof parity bit is used but with slightly more intelligence. With each byte we send one parity bit then send one additional byte which have the parity corresponding to the each bit position of the sent bytes. So the parity bit is set in both horizontal and vertical direction. If one bit get flipped we can tell which row and column have error then we find the intersection of the two and determine the erroneous bit. If 2 bits are in error and they are in the different column and row then they can be detected. If the error are in the same column then the row will differentiate and vice versa. Parity can detect the only odd number of errors. If they are even and distributed in a fashion that in all direction then LRC may not be able to find the error.

- **Cyclic Redundancy Checksum (CRC):**
  We have an n-bit message. The sender adds a k-bit Frame Check Sequence (FCS) to this message before sending. The resulting (n+k) bit message is divisible by some (k+1) bit number. The receiver divides the message ((n+k)-bit) by the same (k+1)-bit number and if there is no remainder, assumes that there was no error. How do we choose this number? For example, if k=12 then 1000000000000 (13-bit number) can be chosen, but this is a pretty crappy choice. Because it will result in a zero remainder for all (n+k) bit messages with the last 12 bits zero. Thus, any bits flipping beyond the last 12 go undetected. If k=12, and we take 1110001000110 as the 13-bit number (incidentally, in decimal representation this turns out to be 7238). This will be unable to detect errors only if the corrupt message and original message have a difference of a multiple of 7238. The probablilty of this is low, much lower than the probability that anything beyond the last 12-bits flips. In practice, this number is

chosen after analyzing common network transmission errors and then selecting a number which is likely to detect these common errors.

**How to detect source errors?**

In order ensure that the frames are delivered correctly, the receiver should inform the sender about incoming frames using positive or negative acknowledgements. On the sender's side the receipt of a positive acknowledgement implies that the frame has arrived at the destination safely while the receipt of a negative acknowledgement means that an error has occurred in the frame and it needs to be retransmitted. However, this scheme is too simplistic because if a noise burst causes the frame to vanish completely, the receiver will not respond at all and the sender would hang forever waiting for an acknowledgement. To overcome this drawback, timers are introduced into the data link layer. When the sender transmits a frame it also simultaneously starts a timer. The timer is set to go off after a interval long enough for the frame to reach the destination, be processed there, and have the acknowledgement propogate back to the sender. If the frame is received correctly the positive acknowledgment arrives before the timer runs out and so the timer is canceled. If however either the frame or the acknowledgement is lost the timer will go off and the sender may retransmit the frame. Since multiple transmission of frames can cause the receiver to accept the same frame and pass it to the network layer more than once, sequence numbers are generally assigned to the outgoing frames.
The types of acknowledgements that are sent can be classified as follows:

- Cumulative acknowledgements: A single acknowledgement informing the sender that all the frames upto a certain number have been received.
- Selective acknowledgements: Acknowledgement for a particular frame.

They may be also classified as:

- Individual acknowledgements: Individual acknowledgement for each frame.
- Group acknowledgements: A bit-map that specifies the acknowledgements of a range of frame numbers.

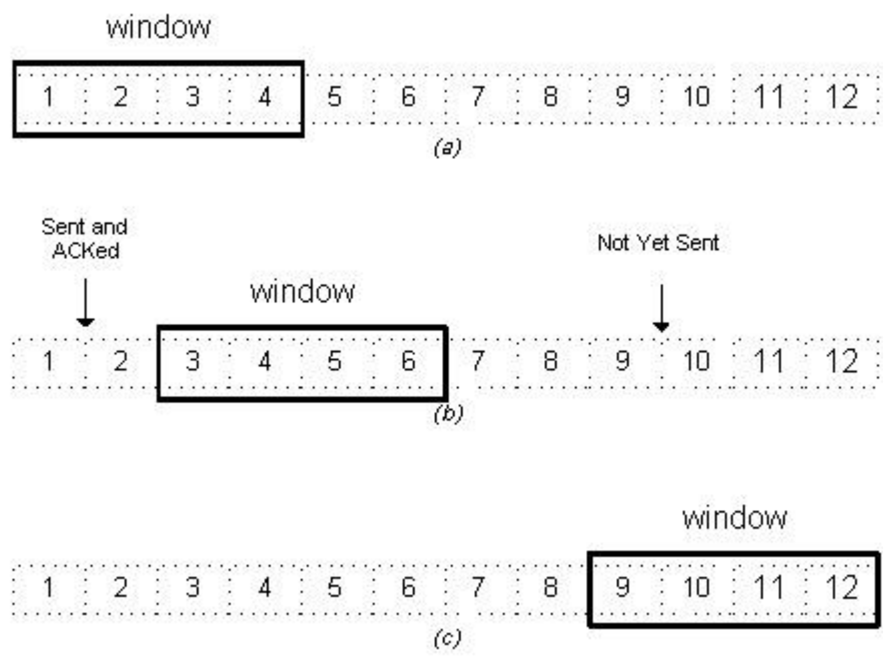
**Flow Control**

Consider a situation in which the sender transmits frames faster than the receiver can accept them. If the sender keeps pumping out frames at high rate, at some point the receiver will be completely swamped and will start losing some frames. This problem may be solved by introducing flow control. Most flow control protocols contain a feedback mechanism to inform the sender when it should transmit the next frame.

**Mechanisms For Flow Control:**

- **Stop and Wait Protocol:** This is the simplest file control protocol in which the sender transmits a frame and then waits for an acknowledgement, either positive or negative, from the receiver before proceeding. If a positive acknowledgement is received, the sender transmits the next packet; else it retransmits the same frame. However, this protocol has one major flaw in it. If a packet or an acknowledgement is completely destroyed in transit due to a noise burst, a deadlock will occur because the sender cannot proceed until it receives an acknowledgement. This problem may be solved using timers on the sender's side. When the frame is transmitted, the timer is set. If there is no response from the receiver within a certain time interval, the timer goes off and the frame may be retransmitted.

- **Sliding Window Protocols:** Inspite of the use of timers, the stop and wait protocol still suffers from a few drawbacks. Firstly, if the receiver had the capacity to accept more than one frame, its resources are being underutilized. Secondly, if the receiver was busy and did not wish to receive any more packets, it may delay the acknowledgement. However, the timer on the sender's side may go off and cause an unnecessary retransmission. These drawbacks are overcome by the sliding window protocols.
  In sliding window protocols the sender's data link layer maintains a 'sending window' which consists of a set of sequence numbers corresponding to the frames it is permitted to send. Similarly, the receiver maintains a 'receiving window' corresponding to the set of frames it is permitted to accept. The window size is dependent on the retransmission policy and it may differ in values for the receiver's and the sender's window. The sequence numbers within the sender's window represent the frames sent but as yet not acknowledged. Whenever a new packet arrives from the network layer, the upper edge of the window is advanced by one. When an acknowledgement arrives from the receiver the lower edge is advanced by one. The receiver's window corresponds to the frames that the receiver's data link layer may accept. When a frame with sequence number equal to the lower edge of the window is received, it is passed to the network layer, an acknowledgement is generated and the window is rotated by one. If however, a frame falling outside the window is received, the receiver's data link layer has two options. It may either discard this frame and all subsequent frames until the desired frame is received or it may accept these frames and buffer them until the appropriate frame is received and then pass the frames to the network layer in sequence.



In this simple example, there is a 4-byte sliding window. Moving from left to right, the window "slides" as bytes in the stream are sent and acknowledged.
Most sliding window protocols also employ ARQ ( Automatic Repeat reQuest ) mechanism. In ARQ, the sender waits for a positive acknowledgement before proceeding to the next frame. If no

acknowledgement is received within a certain time interval it retransmits the frame. ARQ is of two types :

1. **Go Back 'n':** If a frame is lost or received in error, the receiver may simply discard all subsequent frames, sending no acknowledgments for the discarded frames. In this case the receive window is of size 1. Since no acknowledgements are being received the sender's window will fill up, the sender will eventually time out and retransmit all the unacknowledged frames in order starting from the damaged or lost frame. The maximum window size for this protocol can be obtained as follows. Assume that the window size of the sender is n. So the window will initially contain the frames with sequence numbers from 0 to (w-1). Consider that the sender transmits all these frames and the receiver's data link layer receives all of them correctly. However, the sender's data link layer does not receive any acknowledgements as all of them are lost. So the sender will retransmit all the frames after its timer goes off. However the receiver window has already advanced to w. Hence to avoid overlap , the sum of the two windows should be less than the sequence number space.

**w-1 + 1 < Sequence Number Space**
**i.e., w < Sequence Number Space**
**Maximum Window Size = Sequence Number Space - 1**

2. **Selective Repeat:**In this protocol rather than discard all the subsequent frames following a damaged or lost frame, the receiver's data link layer simply stores them in buffers. When the sender does not receive an acknowledgement for the first frame it's timer goes off after a certain time interval and it retransmits only the lost frame. Assuming error - free transmission this time, the sender's data link layer will have a sequence of a many correct frames which it can hand over to the network layer. Thus there is less overhead in retransmission than in the case of Go Back n protocol.
   In case of selective repeat protocol the window size may be calculated as follows. Assume that the size of both the sender's and the receiver's window is w. So initially both of them contain the values 0 to (w-1). Consider that sender's data link layer transmits all the w frames, the receiver's data link layer receives them correctly and sends acknowledgements for each of them. However, all the acknowledgemnets are lost and the sender does not advance it's window. The receiver window at this point contains the values w to (2w-1). To avoid overlap when the sender's data link layer retransmits, we must have the sum of these two windows less than sequence number space. Hence, we get the condition

**Maximum Window Size = Sequence Number Space / 2**

---

**Image References:**

- http://condor.depaul.edu/~jkristof/technotes/sliding-window.jpg

- http://www.eas.asu.edu/trace/eee459_sp02/applet/archana/gupta5463.html

---

back to top
Prev| Next | Index

**Computer Networks (CS425)**

**Instructor: Dr. Dheeraj Sanghi**

Prev| Next| Index

---

## Network Layer (Continued...)

The network layer is concerned with getting packets from the source all the way to the destnation. The packets may require to make many hops at the intermediate routers while reaching the destination. This is the lowest layer that deals with end to end transmission. In order to achieve its goals, the network later must know about the topology of the communication network. It must also take care to choose routes to avoid overloading of some of the communication lines while leaving others idle. The main functions performed by the network layer are as follows:

- Routing
- Congestion Control
- Internetwokring

### Routing

Routing is the process of forwarding of a packet in a network so that it reaches its intended destination. The main goals of routing are:

1. **Correctness:** The routing should be done properly and correctly so that the packets may reach their proper destination.
2. **Simplicity:** The routing should be done in a simple manner so that the overhead is as low as possible. With increasing complexity of the routing algorithms the overhead also increases.
3. **Robustness:** Once a major network becomes operative, it may be expected to run continuously for years without any failures. The algorithms designed for routing should be robust enough to handle hardware and software failures and should be able to cope with changes in the topology and traffic without requiring all jobs in all hosts to be aborted and the network rebooted every time some router goes down.
4. **Stability:** The routing algorithms should be stable under all possible circumstances.
5. **Fairness:** Every node connected to the network should get a fair chance of transmitting their packets. This is generally done on a first come first serve basis.
6. **Optimality:** The routing algorithms should be optimal in terms of throughput and minimizing mean packet delays. Here there is a trade-off and one has to choose depending on his suitability.

**Classification of Routing Algorithms**

The routing algorithms may be classified as follows:

1. **Adaptive Routing Algorithm:** These algorithms change their routing decisions to reflect changes in the topology and in traffic as well. These get their routing information from adjacent routers or from all routers. The optimization parameters are the distance, number of hops and estimated transit time. This can be further classified as follows:

    1. **Centralized:** In this type some central node in the network gets entire information about the network topology, about the traffic and about other nodes. This then transmits this information to the respective routers. The advantage of this is that only one node is required to keep the information. The disadvantage is that if the central node goes down the entire network is down, i.e. single point of failure.

    2. **Isolated:** In this method the node decides the routing without seeking information from other nodes. The sending node does not know about the status of a particular link. The disadvantage is that the packet may be send through a congested route resulting in a delay. Some examples of this type of algorithm for routing are:

        - **Hot Potato:** When a packet comes to a node, it tries to get rid of it as fast as it can, by putting it on the shortest output queue without regard to where that link leads. A variation of this algorithm is to combine static routing with the hot potato algorithm. When a packet arrives, the routing algorithm takes into account both the static weights of the links and the queue lengths.

        - **Backward Learning:** In this method the routing tables at each node gets modified by information from the incoming packets. One way to implement backward learning is to include the identity of the source node in each packet, together with a hop counter that is incremented on each hop. When a node receives a packet in a particular line, it notes down the number of hops it has taken to reach it from the source node. If the previous value of hop count stored in the node is better than the current one then nothing is done but if the current value is better then the value is updated for future use. The problem with this is that when the best route goes down then it cannot recall the second best route to a particular node. Hence all the nodes have to forget the stored informations periodically and start all over again.

    3. **Distributed:** In this the node receives information from its neighbouring nodes and then takes the decision about which way to send the packet. The disadvantage is that if in between the the interval it receives information and sends the paket something changes then the packet may be delayed.

2. **Non-Adaptive Routing Algorithm:** These algorithms do not base their routing decisions on measurements and estimates of the current traffic and topology. Instead the route to be taken in going from one node to the other is computed in advance, off-line, and downloaded to the routers when the network is booted. This is also known as static routing. This can be further classified as:

    1. **Flooding:** Flooding adapts the technique in which every incoming packet is sent on every outgoing line except the one on which it arrived. One problem with this method is that packets may go in a loop. As a result of this a node may receive

several copies of a particular packet which is undesirable. Some techniques adapted to overcome these problems are as follows:

- **Sequence Numbers:** Every packet is given a sequence number. When a node receives the packet it sees its source address and sequence number. If the node finds that it has sent the same packet earlier then it will not transmit the packet and will just discard it.
- **Hop Count:** Every packet has a hop count associated with it. This is decremented(or incremented) by one by each node which sees it. When the hop count becomes zero(or a maximum possible value) the packet is dropped.
- **Spanning Tree:** The packet is sent only on those links that lead to the destination by constructing a spanning tree routed at the source. This avoids loops in transmission but is possible only when all the intermediate nodes have knowledge of the network topology.

Flooding is not practical for general kinds of applications. But in cases where high degree of robustness is desired such as in military applications, flooding is of great help.

2. **Random Walk:** In this method a packet is sent by the node to one of its neighbours randomly. This algorithm is highly robust. When the network is highly interconnected, this algorithm has the property of making excellent use of alternative routes. It is usually implemented by sending the packet onto the least queued link.

**Delta Routing**

Delta routing is a hybrid of the centralized and isolated routing algorithms. Here each node computes the cost of each line (i.e some functions of the delay, queue length, utilization, bandwidth etc) and periodically sends a packet to the central node giving it these values which then computes the **k** best paths from node **i** to node **j**. Let **Cij1** be the cost of the best **i-j** path, **Cij2** the cost of the next best path and so on.If **Cijn - Cij1 < delta**, (**Cijn** - cost of **n'th** best **i-j** path, **delta** is some constant) then path **n** is regarded equivalent to the best **i-j** path since their cost differ by so little. When **delta -> 0** this algorithm becomes centralized routing and when **delta -> infinity** all the paths become equivalent.

**Multipath Routing**

In the above algorithms it has been assumed that there is a single best path between any pair of nodes and that all traffic between them should use it. In many networks however there are several paths between pairs of nodes that are almost equally good. Sometimes in order to improve the performance multiple paths between single pair of nodes are used. This technique is called multipath routing or bifurcated routing. In this each node maintains a table with one row for each possible destination node. A row gives the best, second best, third best, etc outgoing line for that destination, together with a relative weight. Before forwarding a packet, the node generates a random number and then chooses among the alternatives, using the weights as probabilities. The tables are worked out manually and loaded into the nodes before the network is brought up and not changed thereafter.

**Hierarchical Routing**

In this method of routing the nodes are divided into regions based on hierarchy. A particular node can communicate with nodes at the same hierarchial level or the nodes at a lower level and directly under it. Here, the path from any source to a destination is fixed and is exactly one if the heirarchy is a tree.

---

back to top
Prev | Next | Index

**Computer Networks (CS425)**

**Instructor: Dr. Dheeraj Sanghi**

Prev | Next | Index

---

**Routing Algorithms**

**Non-Hierarchical Routing**

In this type of routing, interconnected networks are viewed as a single network, where bridges, routers and gateways are just additional nodes.

- Every node keeps information about every other node in the network
- In case of adaptive routing, the routing calculations are done and updated for all the nodes.

The above two are also the disadvantages of non-hierarchical routing, since the table sizes and the routing calculations become too large as the networks get bigger. So this type of routing is feasible only for small networks.

**Hierarchical Routing**

This is essentially a 'Divide and Conquer' strategy. The network is divided into different regions and a router for a particular region knows only about its own domain and other routers. Thus, the network is viewed at two levels:

1. The Sub-network level, where each node in a region has information about its peers in the same region and about the region's interface with other regions. Different regions may have different 'local' routing algorithms. Each local algorithm handles the traffic between nodes of the same region and also directs the outgoing packets to the appropriate interface.
2. The Network Level, where each region is considered as a single node connected to its interface nodes. The routing algorithms at this level handle the routing of packets between two interface nodes, and is isolated from intra-regional transfer.

Networks can be organized in hierarchies of many levels; e.g. local networks of a city at one level, the cities of a country at a level above it, and finally the network of all nations.

In Hierarchical routing, the interfaces need to store information about:

- All nodes in its region which are at one level below it.
- Its peer interfaces.

- At least one interface at a level above it, for outgoing packages.

Advantages of Hierarchical Routing :

- Smaller sizes of routing tables.
- Substantially lesser calculations and updates of routing tables.

Disadvantage :

- Once the hierarchy is imposed on the network, it is followed and possibility of direct paths is ignored. This may lead to sub optimal routing.

**Source Routing**

Source routing is similar in concept to virtual circuit routing. It is implemented as under:

- Initially, a path between nodes wishing to communicate is found out, either by flooding or by any other suitable method.
- This route is then specified in the header of each packet routed between these two nodes. A route may also be specified partially, or in terms of some intermediate hops.

Advantages:

- Bridges do not need to lookup their routing tables since the path is already specified in the packet itself.
- The throughput of the bridges is higher, and this may lead to better utilization of bandwidth, once a route is established.

Disadvantages:

- Establishing the route at first needs an expensive search method like flooding.
- To cope up with dynamic relocation of nodes in a network, frequent updates of tables are required, else all packets would be sent in wrong direction. This too is expensive.

**Policy Based Routing**

In this type of routing, certain restrictions are put on the type of packets accepted and sent. e.g.. The IIT- K router may decide to handle traffic pertaining to its departments only, and reject packets from other routes. This kind of routing is used for links with very low capacity or for security purposes.

**Shortest Path Routing**

Here, the central question dealt with is 'How to determine the optimal path for routing ?' Various algorithms are used to determine the optimal routes with respect to some predetermined criteria. A network is represented as a graph, with its terminals as nodes and the links as edges. A 'length' is associated with each edge, which represents the cost of using the link for transmission. Lower the cost, more suitable is the link. The cost is determined depending upon the criteria to be optimized. Some of the important ways of determining the cost are:

- **Minimum number of hops:** If each link is given a unit cost, the shortest path is the one with minimum number of hops. Such a route is easily obtained by a breadth first search method. This is easy to implement but ignores load, link capacity etc.

- **Transmission and Propagation Delays:** If the cost is fixed as a function of transmission and propagation delays, it will reflect the link capacities and the geographical distances. However these costs are essentially static and do not consider the varying load conditions.
- **Queuing Delays:** If the cost of a link is determined through its queuing delays, it takes care of the varying load conditions, but not of the propagation delays.

Ideally, the cost parameter should consider all the above mentioned factors, and it should be updated periodically to reflect the changes in the loading conditions. However, if the routes are changed according to the load, the load changes again. This feedback effect between routing and load can lead to undesirable oscillations and sudden swings.

**Routing Algorithms**

As mentioned above, the shortest paths are calculated using suitable algorithms on the graph representations of the networks. Let the network be represented by graph G ( V, E ) and let the number of nodes be 'N'. For all the algorithms discussed below, the costs associated with the links are assumed to be positive. A node has zero cost w.r.t itself. Further, all the links are assumed to be symmetric, i.e. if $d_{i,j}$ = cost of link from node i to node j, then $d_{i,j} = d_{j,i}$ . The graph is assumed to be complete. If there exists no edge between two nodes, then a link of infinite cost is assumed. The algorithms given below find costs of the paths from all nodes to a particular node; the problem is equivalent to finding the cost of paths from a source to all destinations.

**Bellman-Ford Algorithm**

This algorithm iterates on the number of edges in a path to obtain the shortest path. Since the number of hops possible is limited (cycles are implicitly not allowed), the algorithm terminates giving the shortest path.

**Notation:**

| | | |
|---|---|---|
| $d_{i,j}$ | = | Length of path between nodes i and j, indicating the cost of the link. |
| h | = | Number of hops. |
| D[ i,h] | = | Shortest path length from node i to node 1, with upto 'h' hops. |
| D[ 1,h] | = | 0 for all h . |

**Algorithm :**

```
Initial condition :    D[ i, 0] = infinity, for all i ( i != 1 )

Iteration         :    D[i, h+1] = min { $d_{i,j}$ + D[j,h] }   over all values of j .

Termination       :    The algorithm terminates when

                       D[i, h] = D [ i, h+1]   for all i .
```

**Principle:**
For zero hops, the minimum length path has length of infinity, for every node. For one hop the shortest-path length associated with a node is equal to the length of the edge between that node and node 1. Hereafter, we increment the number of hops allowed, (from h to h+1 ) and find out whether a shorter path exists through each of the other nodes. If it exists, say through node 'j', then its length must be the sum of the lengths between these two nodes (i.e. $d_{i,j}$ ) and the shortest path between j and 1 obtainable in upto h paths. If such a path doesn't exist, then the path

length remains the same. The algorithm is guaranteed to terminate, since there are utmost N nodes, and so N-1 paths. It has time complexity of O ( $N^3$ ) .

**Dijkstra's Algorithm**

**Notation:**
$D_i$ = Length of shortest path from node 'i' to node 1.
$d_{i,j}$ = Length of path between nodes i and j .

**Algorithm**
Each node j is labeled with Dj, which is an estimate of cost of path from node j to node 1. Initially, let the estimates be infinity, indicating that nothing is known about the paths. We now iterate on the length of paths, each time revising our estimate to lower values, as we obtain them. Actually, we divide the nodes into two groups ; the first one, called set P contains the nodes whose shortest distances have been found, and the other Q containing all the remaining nodes. Initially P contains only the node 1. At each step, we select the node that has minimum cost path to node 1. This node is transferred to set P. At the first step, this corresponds to shifting the node closest to 1 in P. Its minimum cost to node 1 is now known. At the next step, select the next closest node from set Q and update the labels corresponding to each node using :

$$\mathbf{D_j = \min [ D_j , D_i + d_{j,i} ]}$$

Finally, after N-1 iterations, the shortest paths for all nodes are known, and the algorithm terminates.

**Principle**
Let the closest node to 1 at some step be i. Then i is shifted to P. Now, for each node j , the closest path to 1 either passes through i or it doesn't. In the first case Dj remains the same. In the second case, the revised estimate of $D_j$ is the sum $D_i + d_{i,j}$ . So we take the minimum of these two cases and update $D_j$ accordingly. As each of the nodes get transferred to set P, the estimates get closer to the lowest possible value. When a node is transferred, its shortest path length is known. So finally all the nodes are in P and the $D_j$ 's represent the minimum costs. The algorithm is guaranteed to terminate in N-1 iterations and its complexity is O( $N^2$ ).

**The Floyd Warshall Algorithm**

This algorithm iterates on the set of nodes that can be used as intermediate nodes on paths. This set grows from a single node ( say node 1 ) at start to finally all the nodes of the graph. At each iteration, we find the shortest path using given set of nodes as intermediate nodes, so that finally all the shortest paths are obtained.

**Notation**
$D_{i,j}$ [n] = Length of shortest path between the nodes i and j using only the nodes 1,2,....n as intermediate nodes.

**Initial Condition**
Di,j[0] = $d_{i,j}$ for all nodes i,j .

**Algorithm**
Initially, n = 0. At each iteration, add next node to n. i.e. For n = 1,2, .....N-1 ,

$$Di,j[n + 1] = \min \{ D_{i,j}[n] , D_{i,n+1}[n] + D_{n+1,j}[n] \}$$

**Principle**
Suppose the shortest path between i and j using nodes 1,2,...n is known. Now, if node n+1 is allowed to be an intermediate node, then the shortest path under new conditions either passes through node n+1 or it doesn't. If it does not pass through the node n+1, then $D_{i,j}[n+1]$ is same as $D_{i,j}[n]$ . Else, we find the cost of the new route, which is obtained from the sum, $D_{i,n+1}[n] + D_{n+1,j}[n]$. So we take the minimum of these two cases at each step. After adding all the nodes to the set of intermediate nodes, we obtain the shortest paths between all pairs of nodes together. The complexity of Floyd-Warshall algorithm is O ( $N^3$ ).

It is observed that all the three algorithms mentioned above give comparable performance, depending upon the exact topology of the network.

---

back to top
Prev | Next | Index

**Computer Networks (CS425)**

**Instructor: Dr. Dheeraj Sanghi**

Prev | Next | Index

---

**ARP,RARP,ICMP Protocols**

**Address Resolution Protocol**

If a machine talks to another machine in the same network, it requires its physical or MAC address. But ,since the application has given the destination's IP address it requires some mechanism to bind the IP address with its MAC address.This is done through Address Resolution protocol (ARP).IP address of the destination node is broadcast and the destination node informs the source of its MAC address.

1. Assume broadcast nature of LAN
2. Broadcast IP address of the destination
3. Destination replies it with its MAC address.
4. Source maintains a cache of IP and MAC address bindings

But this means that every time machine A wants to send packets to machine B, A has to send an ARP packet to resolve the MAC address of B and hence this will increase the traffic load too much, so to reduce the communication cost computers that use ARP maintains a cache of recently acquired IP_to_MAC address bindings, i.e. they dont have to use ARP repeatedly. ARP Refinements Several refinements of ARP are possible: When machine A wants to send packets to macine B, it is possible that machine B is going to send packets to machine A in the near future.So to avoid ARP for machine B, A should put its IP_to_MAC address binding in the special packet while requesting for the MAC address of B. Since A broadcasts its initial request for the MAC address of B, every machine on the network should extract and store in its cache the IP_to_MAC address binding of A When a new machine appears on the network (e.g. when an operating system reboots) it can broadcast its

IP_to_MAC address binding so that all other machines can store it in their caches. This will eliminate a lot of ARP packets by all other machines, when they want to communicate with this new machine.

Example displaying the use of Address Resolution Protocol:

Consider a scenario where a computer tries to contact some remote machine using ping program, assuming that there has been no exchange of IP datagrams previously between the two machines and therefore arp packet must be sent to identify the MAC address of the remote machine.

The arp request message (who is A.A.A.A tell B.B.B.B where the two are IP addresses) is broadcast on the local area network with an Ethernet protocol type 0x806. The packet is discarded by all the machines except the target machine which responds with an arp response message (A.A.A.A is hh:hh:hh:hh:hh:hh where hh:hh:hh:hh:hh:hh is the Ethernet source address). This packet is unicast to the machine with IP address B.B.B.B. Since the arp request message included the hardware address (Ethernet source address) of the requesting computer, target machine doesn't require another arp message to figure it out.

---

**Reverse Address Resolution Protocol**

RARP is a protocol by which a physical machine in a local area network can request to learn its IP address from a gateway server's Address Resolution Protocol table or cache. This is needed since the machine may not have permanently attacded disk where it can store its IP address permanently. A network administrator creates a table in a local area network's gateway router that maps the physical machine (or Medium Access Control - MAC) addresses to corresponding Internet Protocol addresses. When a new machine is set up, its RARP client program requests from the RARP server on the router to be sent its IP address. Assuming that an entry has been set up in the router table, the RARP server will return the IP address to the machine which can store it for future use.

Detailed Mechanism

Both the machine that issues the request and the server that responds use physical network addresses during their brief communication. Usually, the requester does not know the physical address. So, the request is broadcasted to all the machines on the network. Now, the requester must identify istelf uniquely to the server. For this either CPU serial number or the machine's physical network address can be used. But using the physical address as a unique id has two advantages.

- These addresses are always available and do not have to be bound into bootstrap code.
- Because the identifying information depends on the network and not on the CPU vendor, all machines on a given network will supply unique identifiers.

**Request:**
Like an ARP message, a RARP message is sent from one machine to the another encapsulated in the data portion of a network frame. An ethernet frame carrying a RARP request has the usual preamle, Ethernet source and destination addresses, and packet type fields in front of the frame. The frame conatins the value 8035 (base 16) to identify the contents of the frame as a RARP message. The data

portion of the frame contains the 28-octet RARP message. The sender braodcasts a RARP request that specifies itself as both the sender and target machine, and supplies its physical network address in the target hardware address field. All machines on the network receive the request, but only those authorised to supply the RARP services process the request and send a reply, such machines are known informally as RARP servers. For RARP to succeed, the network must contain at least one RARP server.

**Reply:**

Servers answers request by filling in the target protocol address field, changing the message type from request to reply, and sending the reply back directly to the machine making the request.

**Timing RARP Transactions**

Since RARP uses the physical network directly, no other protocol software will time the response or retransmit the request. RARP software must handle these tasks. Some workstations that rely on RARP to boot, choose to retry indefinitely until the receive a response. Other implementations announce failure after only a few tries to avoid flooding the network with unnecessary broadcast.

**Mulitple RARP Servers**

Advantage: More reliability. Diadvantage: Overloading may result when all servers respond. So, to get away with disadvantage we have primary and secondary servers. Each machine that makes RARP request is assigned a primary server. Normally, the primary server responds but if it fails, then requester may time out and rebroadcast the request.Whenever a secondary server receives a second copy of the request within a short time of the first, it responds. But, still there might be a problem that all secondary servers respond, thus overloading the network. So, the solution adopted is to avoid having all secondary servers transmit responses simultaneously. Each secondary server that receives the request computes a random delay and then sends a response.

**Drawbacks of RARP**

- Since it operates at low level, it requires direct addresss to the network which makes it difficult for an application programmer to build a server.
- It doesn't fully utilizes the capability of a network like ethernet which is enforced to send a minimum packet size since the reply from the server contains only one small piece of information, the 32-bit internet address.

RARP is formally described in RFC903.

---

**ICMP**

This protocol discusses a mechanism that gateways and hosts use to communicate control or error information.The Internet protocol provides unreliable,connectionless datagram service,and that a datagram travels from gateway to gateway until it reaches one that can deliver it directly to its final destination. If a gateway cannot route or deliver a datagram,or if the gateway detects an unusual condition, like network congestion, that affects its ability to forward the datagram, it needs to instruct the original source to take action to avoid or correct the problem. The Internet Control Message Protocol allows gateways to send error or control messages to other gateways or hosts;ICMP provides communication between the Internet Protocol software on one machine and the Internet Protocol software on another. This is a special purpose message mechanism added by the designers to the TCP/IP protocols. This is to allow gateways in an internet to report errors or provide information about unexpecter circumstances. The IP protocol itself contains nothing to help the sender test connectivity or learn about failures.

**Error Reporting vs Error Correction**
ICMP only reports error conditions to the original source; the source must relate errors to individual application programs and take action to correct problems. It provides a way for gateway to report the error It does not fully specify the action to be taken for each possible error. ICMP is restricted to communicate with the original source but not intermediate sources.
**ICMP Message Delivery**
ICMP messages travel across the internet in the data portion of an IP datagram,which itself travels across the internet in the data portion of an IP datagram,which itself travels across each physical network in the data portion of a frame.Datagrams carryin ICMP messages are routed exactly like datagrams carrying information for users;there is no additional reliability or priority.An exception is made to the error handling procedures if an IP datagram carrying an ICMP messages are not generated for errors that result from datagrams carrying ICMP error messages.
**ICMP Message Format**
It has three fields;an 8-bit integer message TYPE field that identifies the message,an 8-bit CODE field that provides further information about the message type,and a 16-bit CHECKSUM field(ICMP uses the same additive checksum algorithm as IP,but the ICMP checksum only covers the ICMP message).In addition , ICMP messages that report errors always include the header and first 64 data bits of the datagram causing the problem. The ICMP TYPE field defines the meaning of the message as well as its format.
**The Types include :**

| **TYPE FIELD** | **ICMP MESSAGE TYPE** |
|---|---|
| 0 | ECHO REPLY |
| 3 | DESTINATION UNREACHABLE |
| 4 | SOURCE QUENCH |
| 5 | REDIRECT(CHANGE A ROUTE) |
| 8 | ECHO REQUEST |
| 11 | TIME EXCEEDED FOR A DATAGRAM |
| 12 | PARAMETER PROBLEM ON A DATAGRAM |
| 13 | TIMESTAMP REQUEST |
| 14 | TIMESTAMP REPLY |
| 15 | INFORMATION REQUEST(OBSOLETE) |
| 16 | INFORMATION REPLY(OBSOLETE) |
| 17 | ADDRESS MASK REQUEST |
| 18 | ADDRESS MASK REPLY TESTING DESTINATION |

**Reachabilty and Status :**
TCP/IP protocols provide facilities to help network managers or users identify network problems.One of the most frequently used debugging tools invokes the ICMP echo request and echo reply messages.A host or gateway sends an ICMP echo request message to a specified destination.Any machine that receives an echo request formulates an echo reply and returns to the original sender.The request contains an optional data area; the reply contains a copy of the data sent in the request.The echo request and associated reply can be used to test whether a destination is reachable

and responding.Because both the request and reply travel in IP datagrams,successful receipt of a reply verifies that major pieces of the transport system work.
1.1 : IP software on the source must route the datagram
2.2 : Intermediate gateways between the source and destination must be operating and must route datagram correctly.
3.3 : The destination machine must be running , and both ICMP and IP software must be working.
4.4 : Routes in gateways along the return path must be correct.

**Echo Request and Reply**
The field listed OPTIONAL DATA is a variable length field that contains data to be returned to the sender.An echo reply always returns exactly the same data as was received in the request.Fields IDENTIFIER and SEQUENCE NUMBER are used by the sender to match replies to request.The value of the TYPE field specifies whether the message is a request(8) or a reply(0).

**Reports of Unreachable Destinations**
The Code field in a destination unreachable message contains an integer that further describes th problem.Possible values are :

| **CODE VALUE** | **MEANING** |
|---|---|
| 0 | NETWORK UNREACHABLE |
| 1 | HOST UNREACHABLE |
| 2 | PROTOCOL UNREACHABLE |
| 3 | PORT UNREACHABLE |
| 4 | FRAGMENTATION NEEDED AND DF SET |
| 5 | SOURCE ROOT FAILED |
| 6 | DESTINATION NETWORK UNKNOWN |
| 7 | DESTINATION HOST UNKNOWN |
| 8 | SOURCE HOST ISOLATED |
| 9 | COMMUNICATION WITH DESTINATION NETWORK ADMINISTRATIVELY PROHIBITED |
| 10 | COMMUNICATION WTTH DESTINATION HOST ADMINISTRATIVELY PROHIBITED |
| 11 | NETWORK UNREACHABLE FOR TYPE OF SERVICE |
| 12 | HOST UNREACHABLE FOR TYPE OF SERVICE |

Whenever an error prevents a gateway from routing or delivering a datagram, the gateway sends a destination unreachable message back to the source and then drops the datagram.Network unreachable errors usually imply roting failures ; host unreachable errors imply delivery failures.Because the message contains a short prefix of the datagram that caused the problem, the source will know exactly which address is unreachable. Destinations may be unreachable because hardware is temporarily out of service, because the sender specified a nonexistent destination address, or because the gateway does not have a route to the destination network. Although gateways send destination unreachable messages if they cannot route or deliver datagrams, not all such errors can be detected.If the datagram contains the source route option with an incorrect route, it may trigger a source route failure message.If a gateway needs to fragment adatagram but the "don't fragment" bit is set, the gateway sends a fragmentation needed message back to the source.

**Congestion and Datagram Flow Control :**
Gateways cannot reserve memory or communication resources in advance of receiving datagrams because IP is connectionless. The result is, gateways can overrun with traffic, a condition known as congestion.Congestion arises due to two reasons :

1. A high speed computer may be able to generate traffic faster than a network can transfer it .
2. If many computers sumultaneously need to send datagrams through a single gateway , the gateway can experience congestion, even though no single source causes the problem.

When datagrams arrive too quickly for a host or a gateway to process, it enqueues them in memory temporarily.If the traffic continues, the host or gateway eventually exhausts menory ans must discard additional datagrams that arrive. A machine uses ICMP source quench messages to releive congestion. A source quench message is a request for the source to reduce its current rate of datagram transmission.
There is no ICMP messages to reverse the effect of a source quench.
**Source Quench :**
Source quench messages have a field that contains a datagram prefix in addition to the usual ICMP TYPE,CODE,CHECKSUM fields.Congested gateways send one source quench message each time they discard a datagram; the datagram prefix identifies the datagram that was dropped.

**Route Change Requests From Gateways :**
Internet routing tables are initialized by hosts from a configuration file at system startup, and system administrators seldom make routing changes during normal operations.Gateways exchange routing information periodically to accomadate network changes and keep their routes up-to-date.The general rule is , Gateways are assumed to know correct routes; host begin wint minimal routing information and learn new routes from gateways. The GATEWAY INTERNET ADDRESS field contains the address of a gateway that the host is to use to reach the destination mentioned in the datagram header. The INTERNET HEADER field contains IP header plus the next 64 bits of the datagram that triggered the message.The CODE field of an ICMP redirect message further specifies how to interpret the destination address, based on values assigned as follows :

| Code Value | Meaning |
|---|---|
| 0 | REDIRECT DATAGRAMS FOR THE NET |
| 1 | REDIRECT DATAGRAMS FOR THE HOST |
| 2 | REDIRECT DATAGRAMS FOR THE TYPE OF SERVICE AND NET |
| 3 | REDIRECT DATAGRAMS FOR THE TYPE OF SERVICE AND HOST |

Gateways only send ICMP redirect requests to hosts and not to other gateways.

**Detecting Circular or Excessively Long Routes :**
Internet gateways compute a next hop using local tables, errors in routing tables can produce a routing cycle for some destination. A routing cycle can consist of two gateways that each route a datagram for a particular destination to other, or it can consist of several gateways.To prevent datagrams from circling forever in a TCP/IP internet, each IP datagram contains a time-to-live counter , sometimes called a hop count. A gateway decrements the time-to-live counter whenever it processes the datagram and discards the datagram when the count reaches zero. Whenever a gateway discards a datagram because its hop count has reached zero or because a timeout occured while waiting for fragments of a datagram ,it sends an ICMP time exceeded message back to the datagram's source, A gateway sends this message whenever a datagram is discarded because the

time-to-live field in the datagram header has reached zero or because its reassembly timer expired while waiting for fragments.
The code field explains the nature of the timeout :

| Code Value | Meaning |
|---|---|
| 0 | TIME-TO-LIVE COUNT EXCEEDED |
| 1 | FRAGMENT REASSEMBLY TIME EXCEEDED |

Fragment reassembly refers to the task of collecting all the fragments from a datagram.

**Reprting Other Problems :**
When a gateway or host finds problems with a datagram not covered by previous ICMP error messages it sends a parameter problem message to the original source.To make the message unambigous, the sender uses the POINTER field in the message header to identify the octet in the datagram that caused the problem. Code 1 is used to report that a required option is missing; the POINTER field is not used for code 1.

**Clock Synchronization nd Transmit the estimation :**
ICMP messages are used to obtain the time from another machine.A requesting machine sends an ICMP timestamp request message to another machine, asking that the second machine return its current value of the time of day. The receiving machine returns a timestamp reply back to the machine making the request. TCP/IP protocol suite includes several protocols that can be used to synchronize clocks. This is one of the simplest techniques used by TCP/IP. The TYPE field idintifies the message as a request (13 ) or a reply ( 14 ); the IDENTIFIER and SEQUENCE NUMBER fields are used by the source to associate replies with requests.The ORIGINATE TIMESTAMP filed is filled in by the original sendet just before the packet is transmitted, the RECEIVE TIMESTAMP field is filled immediately upon receipt of a request, and the TRANSMIT TIMESTAMP field is filled immediately before the reply is transmitted. Hosts use the three timestamp fields to compute estimates of the delay time between them and to synchronize their clock.A host can compute the total time required for a request to travel to a destination, be transformed into a reply, and return. In practice, accurate estimation of round-trip delay can be difficult and substantially restirct the utility of ICMP timestanp messages.To obtain an accurate estimate to round trip delay one must take many measurements and average them.

**Obtaining a Subnet Mask:**
Subnet addressing is used by the hosts to extract some bits in the hostid portion of their IP address to identify a physical network.To participate in subnet addressing, hosts need to know which bits of the 32-bit internet address correspond to the physical network and which correspond to host identifiers. The information needed to interpret the address is represented in a 32-bit quatity called the subnet mask. To learn the subnet mask used for the local network, a machine can send an address mask request message to a gateway and receive an address mask reply. The TYPE field in an address mask message specifies whether the message is a request ( 17 ) or a reply ( 18 ). A reply contains the nework's subnet address mask in the ADDRESS MASK field.The IDENTIFIER and SEQUENCE NUMBER fields allow a machine to associate replies with requests.

---

back to top
Prev| Next | Index

**Computer Networks (CS425)**

**Instructor: Dr. Dheeraj Sanghi**

---

## Transport Layer Protocol

### What is TCP?

TCP was specifically designed to provide a reliable end to end byte stream over an unreliable internetwork. Each machine supporting TCP has a TCP transport entity either a user process or part of the kernel that manages TCP streams and interface to IP layer. A TCP entity accepts user data streams from local processes, breaks them up into pieces not exceeding 64KB and sends each piece as a separate IP datagram. Client Server mechanism is not necessary for TCP to behave properly.

The IP layer gives no guarantee that datagram will be delivered properly, so it is up to TCP to timeout and retransmit, if needed. Duplicate, lost and out of sequence packets are handled using the sequence number, acknowledgements, retransmission, timers, etc to provide a reliable service. Connection is a must for this service.Bit errors are taken care of by the CRC checksum. One difference from usual sequence numbering is that each byte is given a number instead of each packet. This is done so that at the time of transmission in case of loss, data of many small packets can be combined together to get a larger packet, and hence smaller overhead.

TCP connection is a *duplex connection*. That means there is no difference between two sides once the connection is established.

### TCP Connection establishment

The "three-way handshake" is the procedure used to establish a connection. This procedure normally is initiated by one TCP and responded to by another TCP. The procedure also works if two TCP simultaneously initiate the procedure. When simultaneous attempt occurs, each TCP receives a "SYN" segment which carries no acknowledgment after it has sent a "SYN". Of course, the arrival of an old duplicate "SYN" segment can potentially make it appear, to the recipient, that a simultaneous connection initiation is in progress. Proper use of "reset" segments can disambiguate these cases.

The three-way handshake reduces the possibility of false connections. It is the implementation of a trade-off between memory and messages to provide information for this checking.

The simplest three-way handshake is shown in figure below. The figures should be interpreted in the following way. Each line is numbered for reference purposes. Right arrows (-->) indicate departure of a TCP segment from TCP A to TCP B, or arrival of a segment at B from A. Left arrows (<--), indicate the reverse. Ellipsis (...) indicates a segment which is still in the network (delayed). TCP states represent the state AFTER the departure or arrival of the segment (whose contents are shown in the center of each line). Segment contents are shown in abbreviated form, with sequence number, control flags, and ACK field. Other fields such as window, addresses, lengths, and text have been left out in the interest of clarity.

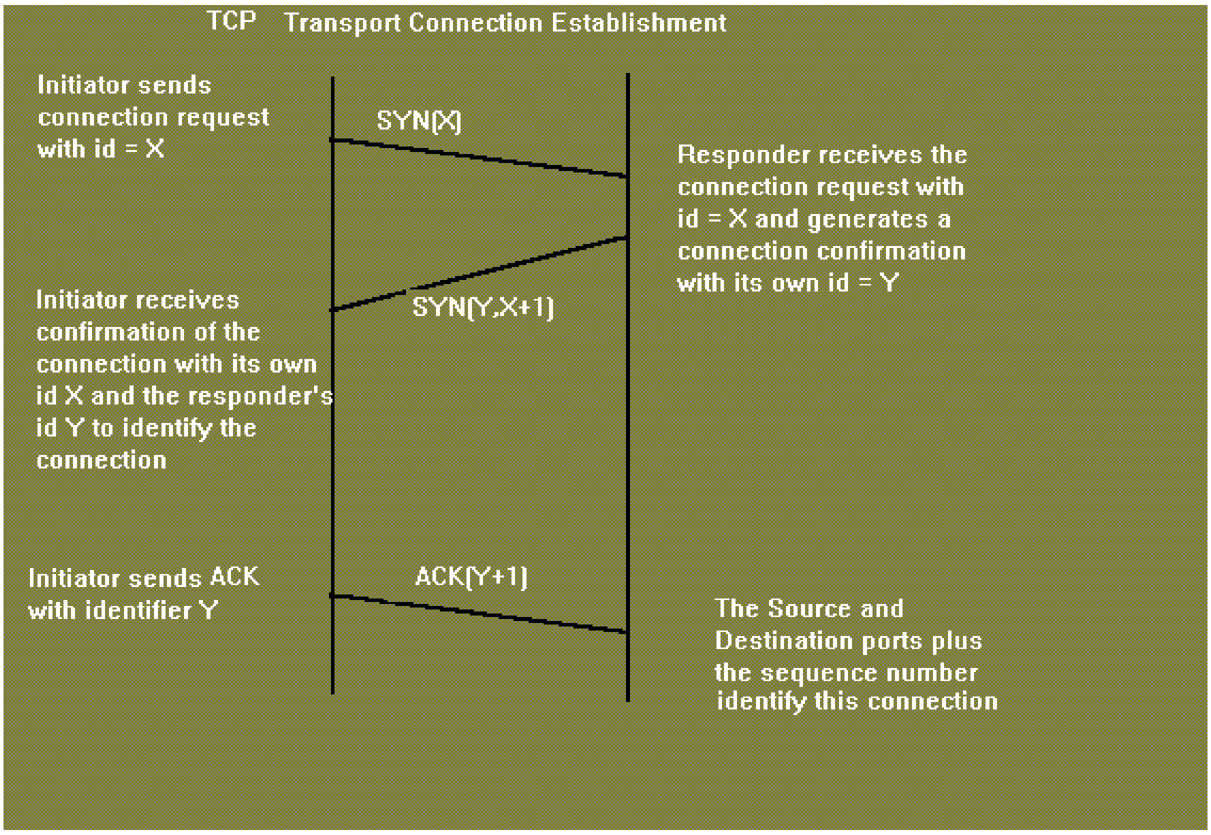
```
   TCP A                              TCP B

1. CLOSED                               LISTEN
```

```
2.  SYN-SENT    --> <SEQ=100><CTL=SYN>              --> SYN-RECEIVED

3.  ESTABLISHED <-- <SEQ=300><ACK=101><CTL=SYN,ACK>  <-- SYN-RECEIVED

4.  ESTABLISHED --> <SEQ=101><ACK=301><CTL=ACK>       --> ESTABLISHED

5.  ESTABLISHED --> <SEQ=101><ACK=301><CTL=ACK><DATA> --> ESTABLISHED
```

Basic 3-Way Handshake for Connection Synchronisation

In line 2 of above figure, TCP A begins by sending a SYN segment indicating that it will use sequence numbers starting with sequence number 100. In line 3, TCP B sends a SYN and acknowledges the SYN it received from TCP A. Note that the acknowledgment field indicates TCP B is now expecting to hear sequence 101, acknowledging the SYN which occupied sequence 100.

At line 4, TCP A responds with an empty segment containing an ACK for TCP B's SYN; and in line 5, TCP A sends some data. Note that the sequence number of the segment in line 5 is the same as in line 4 because the ACK does not occupy sequence number space (if it did, we would wind up ACKing ACK's!).


TCP Transport Connection Establishment
Initiator sends connection request with id = X
SYN(X)
Responder receives the connection request with id = X and generates a connection confirmation with its own id = Y
SYN(Y,X+1)
Initiator receives confirmation of the connection with its own id X and the responder's id Y to identify the connection
Initiator sends ACK with identifier Y
ACK(Y+1)
The Source and Destination ports plus the sequence number identify this connection

Simultaneous initiation is only slightly more complex, as is shown in figure below. Each TCP cycles from CLOSED to SYN-SENT to SYN-RECEIVED to ESTABLISHED.

```
    TCP A                              TCP B

1.  CLOSED                               CLOSED

2.  SYN-SENT     --> <SEQ=100><CTL=SYN>            ...
```

3. SYN-RECEIVED <-- <SEQ=300><CTL=SYN>            <-- SYN-SENT

4.            ... <SEQ=100><CTL=SYN>            --> SYN-RECEIVED

5. SYN-RECEIVED --> <SEQ=100><ACK=301><CTL=SYN,ACK> ...

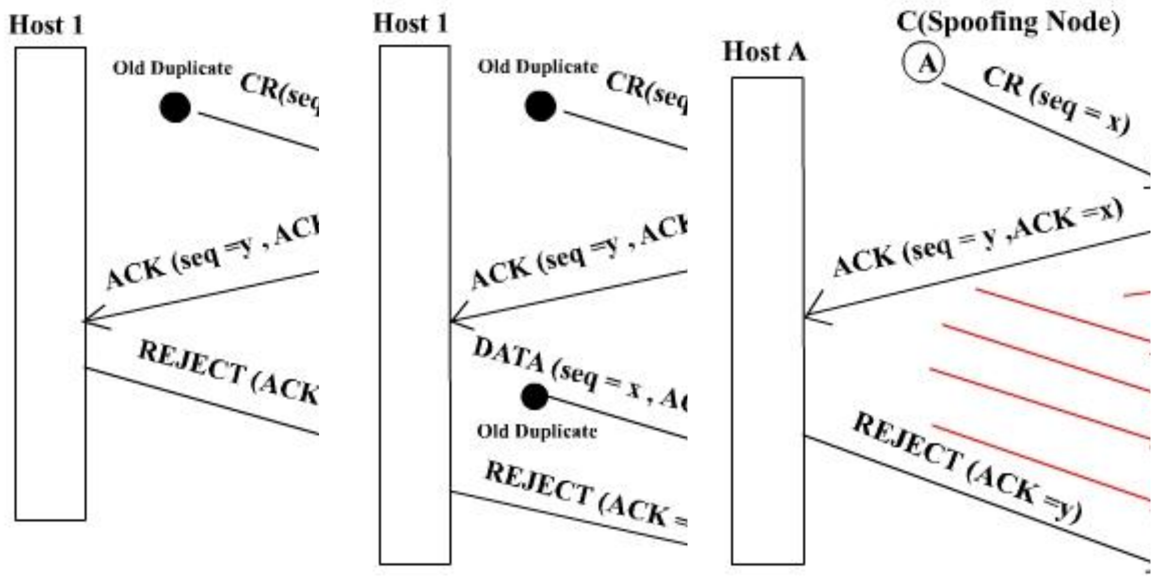
6. ESTABLISHED  <-- <SEQ=300><ACK=101><CTL=SYN,ACK> <-- SYN-RECEIVED

7.            ... <SEQ=101><ACK=301><CTL=ACK>     --> ESTABLISHED

Simultaneous Connection Synchronisation

**Question:** Why is three-way handshake needed? What is the problem if we send only two packets and consider the connection established? What will be the problem from application's point of view? Will the packets be delivered to the wrong application?

Problem regarding 2-way handshake
The only real problem with a 2-way handshake is that duplicate packets from a previous connection( which has been closed) between the two nodes might still be floating on the network. After a SYN has been sent to the responder, it might receive a duplicate packet of a previous connection and it would regard it as a packet from the current connection which would be undesirable.
Again spoofing is another issue of concern if a two way handshake is used.Suppose there is a node C which sends connection request to B saying that it is A.Now B sends an ACK to A which it rejects & asks B to close connection.Beteween these two events C can send a lot of packets which will be delievered to the application..

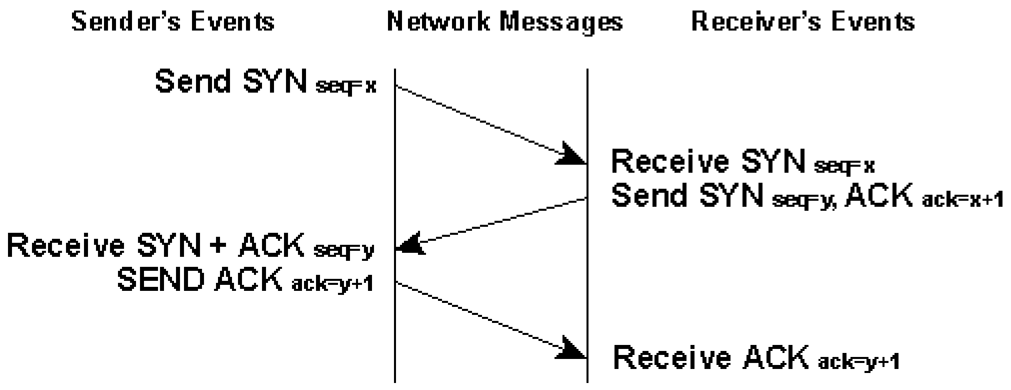


The first two figures show how a three way handshake deals with problems of duplicate/delayed connection requests and duplicate/delayed connection acknowledgements in the network.The third figure highlights the problem of spoofing associated with a two way handshake.

**Some Conventions**

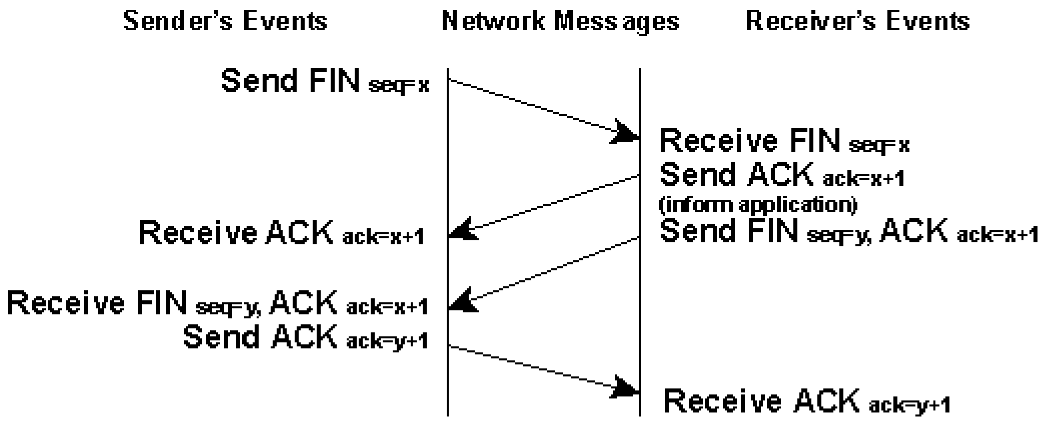
1. The ACK contains 'x+1' if the sequence number received is 'x'.
2. If 'ISN' is the sequence number of the connection packet then 1st data packet has the seq number 'ISN+1'
3. Seq numbers are 32 bit.They are byte seq number(every byte has a seq number).With a packet 1st seq number and length of the packet is sent.
4. Acknowlegements are cummulative.
5. Acknowledgements have a seq number of their own but with a length 0.So the next data packet have the seq number same as ACK.

**Connection Establish**



- The sender sends a SYN packet with serquence numvber say 'x'.
- The receiver on receiving SYN packet responds with SYN packet with sequence number 'y' and ACK with seq number 'x+1'
- On receiving both SYN and ACK packet, the sender responds with ACK packet with seq number 'y+1'
- The receiver when receives ACK packet, initiates the connection.

**Connection Release**



- The initiator sends a FIN with the current sequence and acknowledgement number.
- The responder on receiving this informs the application program that it will receive no more data and sends an acknowledgement of the packet. The connection is now closed from one side.

- Now the responder will follow similar steps to close the connection from its side. Once this is done the connection will be fully closed.

---

**Image References**

- http://www.renoir.vill.edu/~cassel/4900/transport.html
- www.uga.edu/~ucns/lans/tcpipsem/close.conn.gif
- www.uga.edu/~ucns/lans/tcpipsem/establish.conn.gif

---

back to top
Prev| Next | Index

**Computer Networks (CS425)**

**Instructor: Dr. Dheeraj Sanghi**

Prev| Next| Index

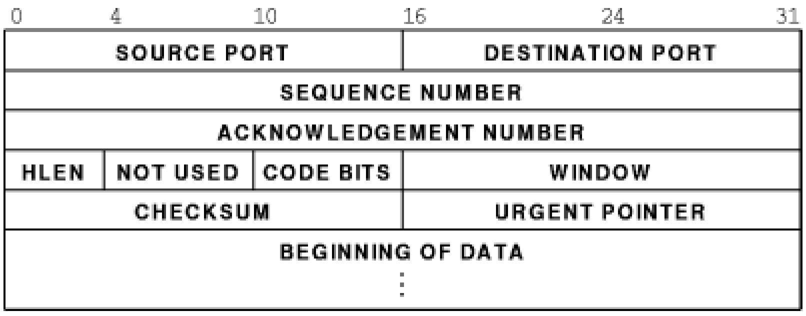
---

**Transport Layer Protocol (continued)**

TCP connection is a duplex connection. That means there is no difference between two sides once the connection is established.
**Salient Features of TCP**

- **Piggybacking of acknowledments:**The ACK for the last received packet need not be sent as a new packet, but gets a free ride on the next outgoing data frame(using the ACK field in the frame header). The technique is temporarily delaying outgoing ACKs so that they can be hooked on the next outgoing data frame is known as piggybacking. But ACK can't be delayed for a long time if receiver(of the packet to be acknowledged) does not have any data to send.
- **Flow and congestion control:**TCP takes care of flow control by ensuring that both ends have enough resources and both can handle the speed of data transfer of each other so that none of them gets overloaded with data. The term congestion control is used in almost the same context except that resources and speed of each router is also taken care of. The main concern is network resources in the latter case.
- **Multiplexing / Demultiplexing:** Many application can be sending/receiving data at the same time. Data from all of them has to be multiplexed together. On receiving some data from lower layer, TCP has to decide which application is the recipient. This is called demultiplexing. TCP uses the concept of port number to do this.

**TCP segment header:**



**Explanation of header fields:**

- **Source and destination port :**These fields identify the local endpoint of the connection. Each host may decide for itself how to allocate its own ports starting at 1024. The source and destination socket numbers together identify the connection.
- **Sequence and ACK number :** This field is used to give a sequence number to each and every byte transferred. This has an advantage over giving the sequence numbers to every packet because data of many small packets can be combined into one at the time of retransmission, if needed. The ACK signifies the next byte expected from the source and not the last byte received. The ACKs are cumulative instead of selective.Sequence number space is as large as 32-bit although 17 bits would have been enough if the packets were delivered in order. If packets reach in order, then according to the following formula:

(sender's window size) + (receiver's window size) < (sequence number space)

the sequence number space should be 17-bits. But packets may take different routes and reach out of order. So, we need a larger sequence number space. And for optimisation, this is 32-bits.

- **Header length :**This field tells how many 32-bit words are contained in the TCP header. This is needed because the options field is of variable length.
- **Flags :** There are six one-bit flags.
    1. **URG :** This bit indicates whether the urgent pointer field in this packet is being used.
    2. **ACK :**This bit is set to indicate the ACK number field in this packet is valid.
    3. **PSH :** This bit indicates PUSHed data. The receiver is requested to deliver the data to the application upon arrival and not buffer it until a full buffer has been received.
    4. **RST :** This flag is used to reset a connection that has become confused due to a host crash or some other reason.It is also used to reject an invalid segment or refuse an attempt to open a connection. This causes an abrupt end to the connection, if it existed.

5. **SYN :** This bit is used to establish connections. The connection request(1st packet in 3-way handshake) has SYN=1 and ACK=0. The connection reply (2nd packet in 3-way handshake) has SYN=1 and ACK=1.
6. **FIN :** This bit is used to release a connection. It specifies that the sender has no more fresh data to transmit. However, it will retransmit any lost or delayed packet. Also, it will continue to receive data from other side. Since SYN and FIN packets have to be acknowledged, they must have a sequence number even if they do not contain any data.

- **Window Size :** Flow control in TCP is handled using a variable-size sliding window. The Window Size field tells how many bytes may be sent starting at the byte acknowledged. Sender can send the bytes with sequence number between (ACK#) to (ACK# + window size - 1) A window size of zero is legal and says that the bytes up to and including ACK# -1 have been received, but the receiver would like no more data for the moment. Permission to send can be granted later by sending a segment with the same ACK number and a nonzero Window Size field.
- **Checksum :** This is provided for extreme reliability. It checksums the header, the data, and the conceptual pseudoheader. The pseudoheader contains the 32-bit IP address of the source and destination machines, the protocol number for TCP(6), and the byte count for the TCP segment (including the header).Including the pseudoheader in TCP checksum computation helps detect misdelivered packets, but doing so violates the protocol hierarchy since the IP addresses in it belong to the IP layer, not the TCP layer.
- **Urgent Pointer :** Indicates a byte offset from the current sequence number at which urgent data are to be found. Urgent data continues till the end of the segment. This is not used in practice. The same effect can be had by using two TCP connections, one for transferring urgent data.
- **Options :** Provides a way to add extra facilities not covered by the regular header. eg,
    - Maximum TCP payload that sender is willing to handle. The maximum size of segment is called MSS (Maximum Segment Size). At the time of handshake, both parties inform each other about their capacity. Minimum of the two is honoured. This information is sent in the options of the SYN packets of the three way handshake.
    - Window scale option can be used to increase the window size. It can be specified by telling the receiver that the window size should be interpreted by shifting it left by specified number of bits. This header option allows window size up to 230.
- **Data :** This can be of variable size. TCP knows its size by looking at the IP size header.

**Topics to be Discussed relating TCP**

1. **Maximum Segment Size :** It refers to the maximum size of segment ( MSS ) that is acceptable to both ends of the connection. TCP negotiates for MSS using OPTION field. In Internet environment MSS is to be selected optimally. An arbitrarily small segment size will result in poor bandwith utilization since Data to Overhead ratio remains low. On the other hand extremely large segment size will necessitate large IP Datagrams which require fragmentation. As there are finite chances of a fragment getting lost, segment size above

"fragmentation threshold " decrease the Throughput. Theoretically an optimum segment size is the size that results in largest IP Datagram, which do not require fragmentation anywhere enroute from source to destination. However it is very difficult to find such an optimum segmet size. In system V a simple technique is used to identify MSS. If H1 and H2 are on the same network use MSS=1024. If on different networks then MSS=5000.

2. **Flow Control :** TCP uses Sliding Window mechanism at octet level. The window size can be variable over time. This is achieved by utilizing the concept of "Window Advertisement" based on :
    1. **Buffer availabilty at the receiver**
    2. **Network conditions ( traffic load etc.)**

In the former case receiver varies its window size depending upon the space available in its buffers. The window is referred as RECEIVE WINDOW (Recv_Win). When receiver buffer begin to fill it advertises a small Recv_Win so that the sender does'nt send more data than it can accept. If all buffers are full receiver sends a "Zero" size advertisement. It stops all transmission. When buffers become available receiver advertises a Non Zero widow to resume retransmission. The sender also periodically probes the "Zero" window to avoid any deadlock if the Non Zero Window advertisement from receiver is lost. The Variable size Recv_Win provides efficient end to end flow control.
The second case arises when some intermediate node ( e.g. a router ) controls the source to reduce transmission rate. Here another window referred as COGESTION WINDOW (C_Win) is utilized. Advertisement of C_Win helps to check and avoid congestion.

3. **Congestion Control :** Congestion is a condition of severe delay caused by an overload of datagrams at any intermediate node on the Internet. If unchecked it may feed on itself and finally the node may start dropping arriving datagrams.This can further aggravate congestion in the network resulting in congestion collapse. TCP uses two techniques to check congestion.
    1. **Slow Start :** At the time of start of a connection no information about network conditios is available. A Recv_Win size can be agreed upon however C_Win size is not known. Any arbitrary C_Win size can not be used because it may lead to congestion. TCP acts as if the window size is equal to the minimum of ( Recv_Win & C_Win). So following algorithm is used.
        1. Recv_Win=X
        2. SET C_Win=1
        3. for every ACK received C_Win++
    2. **Multiplicative decrease :** This scheme is used when congestion is encountered ( ie. when a segment is lost ). It works as follows. Reduce the congestion window by half if a segment is lost and exponentially backoff the timer ( double it ) for the segments within the reduced window. If the next segment also gets lost continue the above process. For successive losses this scheme reduces traffic into the connection exponentially thus allowing the intermediate nodes to clear their queues. Once congestion ends SLOW START is used to scale up the transmission.

4. **Congestion Avoidance :** This procedure is used at the onset of congestion to minimize its effect on the network. When transmission is to be scaled up it should be done in such a way that it does'nt lead to congestion again. Following algorithm is used .
    1. At loss of a segment SET C_Win=1
    2. SET SLOW START THRESHOLD (SST) = Send_Win / 2
    3. Send segment
    4. If ACK Received, C_Win++ till C_Win <= SST
    5. else for each ACK C_Win += 1 / C_Win
5. **Time out and Retransmission :** Following two schemes are used :
    1. **Fast Retransmit**
    2. **Fast Recovery**

When a source sends a segment TCP sets a timer. If this value is set too low it will result in many unnecessary treransmissions. If set too high it results in wastage of banwidth and hence lower throughput. In Fast Retransmit scheme the timer value is set fairly higher than the RTT. The sender can therefore detect segment loss before the timer expires. This scheme presumes that the sender will get repeated ACK for a lost packet.

6. **Round Trip Time (RTT) :** In Internet environment the segments may travel across different intermediate networks and through multiple routers. The networks and routers may have different delays, which may vary over time. The RTT therefore is also variable. It makes difficult to set timers. TCP allows varying timers by using an adaptive retransmission algorithm. It works as follows.
    1. Note the time (t1) when a segment is sent and the time (t2) when its ACK is received.
    2. Compute RTT(sample) = (t 2 - t 1 )
    3. Again Compute RTT(new) for next segment.
    4. Compute Average RTT by weighted average of old and new values of RTT
    5. RTT(est) = a *RTT(old) + (1-a) * RTT (new) where 0 < a < 1
       A high value of 'a' makes the estimated RTT insensitive to changes that last for a short time and RTT relies on the history of the network. A low value makes it sensitive to current state of the network. A typical value of 'a' is 0.75
    6. Compute Time Out = b * RTT(est) where b> 1
       A low value of 'b' will ensure quick detection of a packet loss. Any small delay will however cause unnecessary retransmission. A typical value of 'b' is kept at .2

---

**Image References**

- http://plato.acadiau.ca/courses/comp/Eberbach/comp4343/lectures/transport/Com-TCP/f20_6.gif

back to top
Prev| Next | Index

**Computer Networks (CS425)**

**Instructor: Dr. Dheeraj Sanghi**

Prev| Next| Index

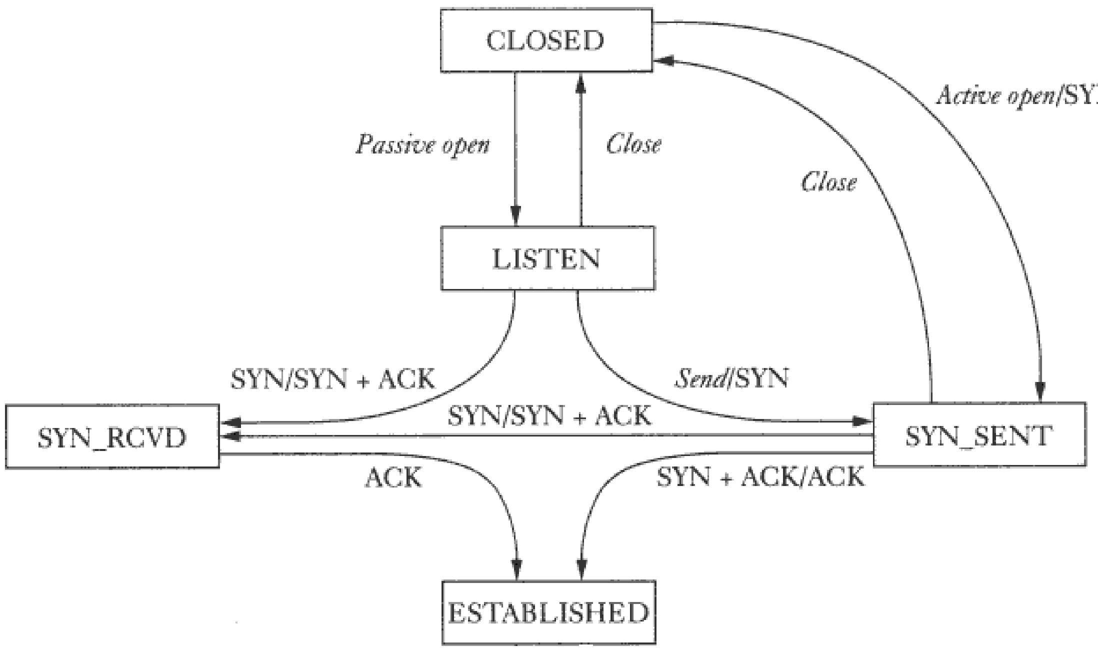
---

**Transport Layer Protocol- Implementation Issues**

In this class we discussed about the TCP from the implementation point of view and addressed various issues like state diagram and other details which TCP Standard does not define but supported by commercial implementations.

**State Diagram**

The state diagram approach to view the TCP connection establishment and closing simplifies the design of TCP implementation. The idea is to represent the TCP connection state, which progresses from one state to other as various messages are exchanged. To simplify the matter, we considered two state diagrams, viz., for TCP connection establishment and TCP connection closing.
Fig 1 shows the state diagram for the TCP connection establishment and associated table briefly explains each state.



TCP Connection establishment

The table gives brief description of each state of the above diagram.

State Description Table 1.

| | |
|---|---|
| Listen | Represents the state when waiting for connection request from any remote host and port. This specifically applies to a Server. From this state, the server can close the service or actively open a connection by sending SYN. |
| Syn-Sent | Represents waiting for a matching for a connection request after having sent a connection request. This applies to both server and client side. Even though server is considered as the one with passive open, it can also send a SYN packet actively. |
| Syn_Rcvd | Represents waiting for a confirmation connection request acknowledgment after having both received and sent connection request. |
| Estab | Represents an open connection. Data transfer can take place from this point onwards. |

After the connection has been established, two end-points will exchange useful information and terminate the connection. Fig. 2 shows the state diagram for terminating an active connection.

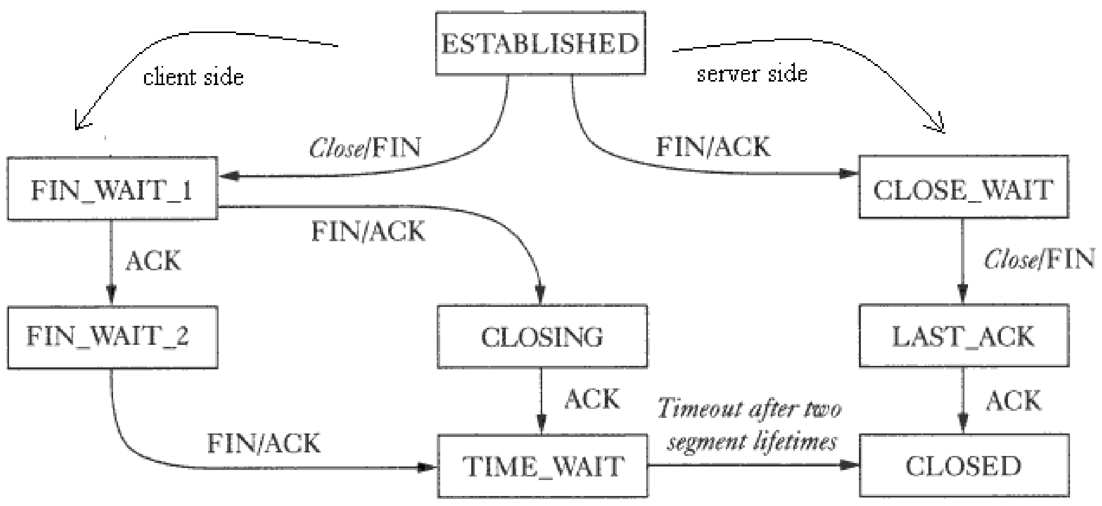


Fig 2.  TCP Connection termination

State Description Table 2

| | |
|---|---|
| FIN-WAIT-1 | Represents connection termination request from the remote TCP peer, or an acknowledgment of the connection termination request previously sent. This state is entered when server issues close call. |
| FIN-WAIT-2 | Represents waiting for a connection termination request from the remote TCP. |
| CLOSING | Represents connection termination request acknowledgment from the remote TCP. |
| TIME_WAIT | This represents waiting time enough for the packets to reach their destination. This waiting time is usually 4 min. |
| CLOSE_WAIT | Represents a state when the server receives a FIN from the remote TCP , sends ACK and issues close call sending FIN |
| LAST_ACK | Represents waiting for an ACK for the previously sent FIN-ACK to the remote TCP |
| CLOSE | Represents a closed TCP connection having received all the ACKs |

**Other implementation details**

**Quite Time**

It might happen that a host currently in communication crashes and reboots. At startup time, all the data structures and timers will be reset to an initial value. To make sure that earlier connection packets are gracefully rejected, the local host is not allowed to make any new connection for a small period at startup. This time will be set in accordance with reboot time of the operating system.

**Initial Sequence number :**

Initial sequence number used in the TCP communication will be initialized at boot time randomly, rather than to 0. This is to ensure that packets from old connection should not interfere with a new connection. So the recommended method is to

- Initialize the ISN at boot time by a random number
- For every 500 ms, increment ISN by 64K
- With every SYN received, increment ISN by 64K

**Maximum Request backlog at server**

As we have seen in Unix Networking programming, *listen(sd,n),* sets a maximum to the number of requests to be obliged by the server at any time. So if there are already n requests for connection, and n+1 request comes, two things can be done.

- Drop the packet silently
- Ask the peer to send the request later.

The first option is recommended here because, the assumption is that this queue for request is a coincident and some time later, the server should be free to process the new request. Hence if we drop the packet, the client will go through the time-out and retransmission and server will be free to process it.

Also, Standard TCP does not define any strategy/option of knowing who requested the connection. Only Solaris 2.2 supports this option.

**Delayed Acknowledgment**

TCP will piggyback the acknowledgment with its data. But if the peer does not have the any data to send at that moment, the acknowledgment should not be delayed too long. Hence a timer for 200 ms will be used. At every 200 ms, TCP will check for any acknowledgment to be sent and send them as individual packets.

**Small packets**

TCP implementation discourages small packets. Especially if a previous relatively large packet has been sent and no acknowledgment has been received so far, then this small packet will be stored in the buffer until the situation improves.

But there are some applications for which delayed data is worse than bad data. For example, in *telnet,* each key stroke will be processed by the server and hence no delay should be introduced. As we have seen in Unix Networking programming, options for the socket can be set as NO_DELAY, so that small packets are not discouraged.

**ICMP Source Quench**

We have seen in ICMP that ICMP Source Quench message will be send for the peer to slow down. Some implementations discard this message, but few set the **current window size** to 1.
But this is not a very good idea.

**Retransmission Timeout**

In some implementation (E.g.. Linux), RTO = RTT + 4 * delay variance is used to instead of constant 2.

Also instead of calculating RTT(est) from the scratch, cache will be used to store the history from which new values are calculated as discussed in the previous classes.

Standard values for Maximum Segment Life (MSL) will be between 0.5 to 2 minutes and Time wait state = f(MSL)

**Keep Alive Time**

Another important timer in TCP is keep alive timer. It is basically used by a TCP peer to check whether the other end is up or down. It periodically checks this connection. If the other end did not respond, then that connection will be closed.

**Persist Timer**

As we saw in TCP window management, when source sends one full window of packets, it will set its window size to 0 and expects an ACK from remote TCP to increase its window size. Suppose such an ACK has been sent and is lost. Hence source will have current window size = 0 and cannot send & destination is expecting next byte. To avoid such a deadlock, a Persist Timer will be used. When this timer goes off, the source will send the last one byte again. So we hope that situation has improved and an ACK to increase the current window size will be received.

---

**References**

- http://www.ssfnet.org/Exchange/tcp/tcpTutorialNotes.html

---

back to top
Prev| Next | Index

**Computer Networks (CS425)**

**Instructor: Dr. Dheeraj Sanghi**

Prev | Next | Index

---

**Unix Socket Programming**

**Client Server Architecture**

In the client server architecture, a machine(refered as client) makes a request to connect to another machine (called as server) for providing some service. The services running on the server run on known ports(application identifiers) and the client needs to know the address of the server machine and this port in order to connect to the server. On the other hand, the server does not need to know about the address or the port of the client at the time of connection initiation. The first packet which the client sends as a request to the server contains these informations about the client which are further used by the server to send any information. Client acts as the active device which makes the first move to establish the connection whereas the server passively waits for such requests from some client.

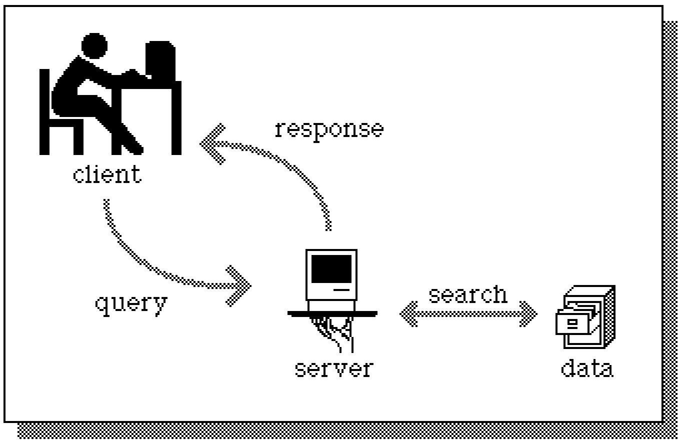


**Illustration of Client Server Model**

**What is a Socket ?**

In unix, whenever there is a need for inter process communication within the same machine, we use mechanism like signals or pipes(named or unnamed). Similarly, when we desire a communication between two applications possibly running on different machines, we need **sockets**. Sockets are treated as another entry in the unix open file table. So all the system calls which can be used for any IO in unix can be used on socket. The server and client applications use various system calls to conenct which use the basic construct called **socket**. A socket is one end of the communication channel between two applications running on different machines.

Steps followed by client to establish the connection:

1. Create a socket
2. Connect the socket to the address of the server
3. Send/Receive data
4. Close the socket

Steps followed by server to establish the connection:

1. Create a socket
2. Bind the socket to the port number known to all clients
3. Listen for the connection request
4. Accept connection request
5. Send/Receive data

**Basic data structures used in Socket programming**

**Socket Descriptor:** A simple file descriptor in Unix.

    int

**Socket Address:** This construct holds the information for socket address

```
struct sockaddrs {
   unsigned short    sa_family;    // address family, AF_xxx or PF_xxx
   char             sa_data[14];  // 14 bytes of protocol address
};
```

AF stands for Address Family and PF stands for Protocol Family. In most modern implementations only the AF is being used. The various kinds of AF are as follows:

| Name | Purpose |
|---|---|
| AF_UNIX, AF_LOCAL | Local communication |
| AF_INET | IPv4 Internet protocols |
| AF_INET6 | IPv6 Internet protocols |
| AF_IPX | IPX - Novell protocols |
| AF_NETLINK | Kernel user interface device |
| AF_X25 | ITU-T X.25 / ISO-8208 protocol |
| AF_AX25 | Amateur radio AX.25 protocol |
| AF_ATMPVC | Access to raw ATM PVCs |
| AF_APPLETALK | Appletalk |
| AF_PACKET | Low level packet interface |

In all the sample programs given below, we will be using AF_INET.
**struct sockaddr_in:** This construct holds the information about the address family, port number, Internet address,and the size of the struct sockaddr.

```
struct sockaddr_in {
   short int         sin_family;  // Address family
   unsigned short int sin_port;    // Port number
   struct in_addr     sin_addr;    // Internet address
   unsigned char      sin_zero[8]; // Same size as struct sockaddr
};
```

Some systems (like x8086) are Little Endian i-e. least signficant byte is stored in the higher address, whereas in Big endian systems most significant byte is stored in the higher address. Consider a situation where a Little Endian system wants to communicate with a Big Endian one, if there is no standard for data representation then the data sent by one machine is misinterpreted by the other. So standard has been defined for the data representation in the network (called Network Byte Order) which is the Big Endian. The system calls that help us to convert a short/long from Host Byte order to Network Byte Order and viceversa are

- htons() -- "Host to Network Short"
- htonl() -- "Host to Network Long"
- ntohs() -- "Network to Host Short"
- ntohl() -- "Network to Host Long"

**IP addresses**

Assuming that we are dealing with IPv4 addresses, the address is a 32bit integer. Remembering a 32 bit number is not convenient for humans. So, the address is written as a set of four integers seperated by dots, where each integer is a representation of 8 bits. The representation is like a.b.c.d, where a is the representation of the most significant byte. The system call which converts this representation into Network Byte Order is:

```
int inet_aton(const char *cp, struct in_addr *inp);
```

**inet_aton()** converts the Internet host address *cp* from the standard numbers-and-dots notation into binary data and stores it in the structure that *inp* points to. inet_aton returns nonzero if the address is valid, zero if not.
For example, if we want to initialize the sockaddr_in construct by the IP address and desired port number, it is done as follows:

```
struct sockaddr_in sockaddr;

sockaddr.sin_family = AF_INET;

sockaddr.sin_port = htons(21);

inet_aton("172.26.117.168", &(sockaddr.sin_addr));

memset(&(sockaddr.sin_zero), '\0', 8);
```

**Socket System Call**

A socket is created using the system call:

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

This system call returns a Socket Descriptor (like file descriptor) which is an integer value. Details about the Arguments:

1. **Domain:** It specifies the communication domain. It takes one of the predefined values described under the protocol family and address family above in this lecture.

2. **Type:** It specifies the semantics of communication , or the type of service that is desired . It takes the following values:
    - SOCK_STREAM : Stream Socket
    - SOCK_DGRAM : Datagram Socket
    - SOCK_RAW : Raw Socket
    - SOCK_SEQPACKET : Sequenced Packet Socket
    - SOCK_RDM : Reliably Delivered Message Packet
3. **Protocol:** This parameter identifies the protocol the socket is supposed to use . Some values are as follows:
    - IPPROTO_TCP : For TCP (SOCK_STREAM)
    - IPPROTO_UDP : For UDP (SOCK_DRAM)

Since we have only one protocol for each kind of socket, it does not matter if we do not define any protocol at all. So for simplicity, we can put "0" (zero) in the protocol field.

**Bind System Call**

The system call bind associates an address to a socket descriptor created by socket.

```
int bind  ( int sockfd  ,  struct sockaddr *myaddr  ,  int addrlen );
```

The second parameter *myaddr* specifies a pointer to a predefined address of the socket.Its structure is a general address structure so that the bind system call can be used by both Unix domain and Internet domain sockets.

**Other System Calls and their Functions**

LISTEN : Annoumce willingness to accept connections ; give queue size.
ACCEPT : Block the caller until a commwction attempt arrives.
CONNECT : Actively attempt to establish a connection.
SEND : Send some data over the connection.
RECIEVE : Recieve sme data from the connection.
CLOSE : Release the connection.

---

back to top
Prev | Next | Index

---

**Unix Socket Programming (Contd..)**

**Client-Server Communication Overview**

The analogy given below is often very useful in understanding many such networking concepts. The analogy is of a number of people in a room communicating with each other by way of talking. In a typical scenario, if A has to talk to B, then he would call out the name of B and only if B was listening would he respond. In case B responds, then one can say that a connection has been established. Henceforth until both of them desire to communicate, they can carry out their conversation.

A Client-Server architecture generally employed in networks is also very similar in concept. Each machine can act as a client or a server.

**Server:** It is normally defined which provides some sevices to the client programs. However, we will have a deeper look at the concept of a "service" in this respect later. The most important feature of a server is that it is a passive entiry, one that listens for request from the clients.

**Client:** It is the active entity of the architecture, one that generated this request to connect to a particular port number on a particular server

Communication takes the form of the client process sending a message over the network to the server process. The client process then waits for a reply message. When the server process gets the request, it performs the requested work and sends back a reply.The server that the client will try to connect to should be up and running before the client can be executed. In most of the cases, the servers runs continuously as a daemon.

There is a general misconception that servers necessarily provide some service and is therefore called a server. For example an e-mail client provides as much service as an mail server does. Actually the term service is not very well defined. So it would be better not to refer to the term at all. In fact servers can be programmed to do practically anything that a normal application can do. In brief, a server is just an entity that listens/waits for requests.

To send a request, the client needs to know the address of the server as well as the port number which has to be supplied to establish a connection. One option is to make the server choose a random number as a port number, which will be somehow conveyed to the client. Subsequently the client will use this port number to send requests. This method has severe limitations as such information has to be communicated offline, the network connection not yet being established. A better option would be to ensure that the server runs on the same port number always and the client already has knowledge as to which port provides which service. Such a standardization already exists. The port numbers 0-1023 are reserved for the use of the superuser only. The list of the services and the ports can be found in the file /etc/services.

**Connection Oriented vs Connectionless Communication**

**Connection Oriented Communication**

Analogous to the telephone network.The sender requests for a communication (dial the number), the receiver gets an indication (the phone ring) the receiver accepts the connection (picks up the phone) and the sender receives the acknowledgment (the ring stops). The connection is established through a dedicated link provided for the communication. This type of communication is characterized by a high level of reliability in terms of the number and the sequence of bytes.

**Connectionless Communication**

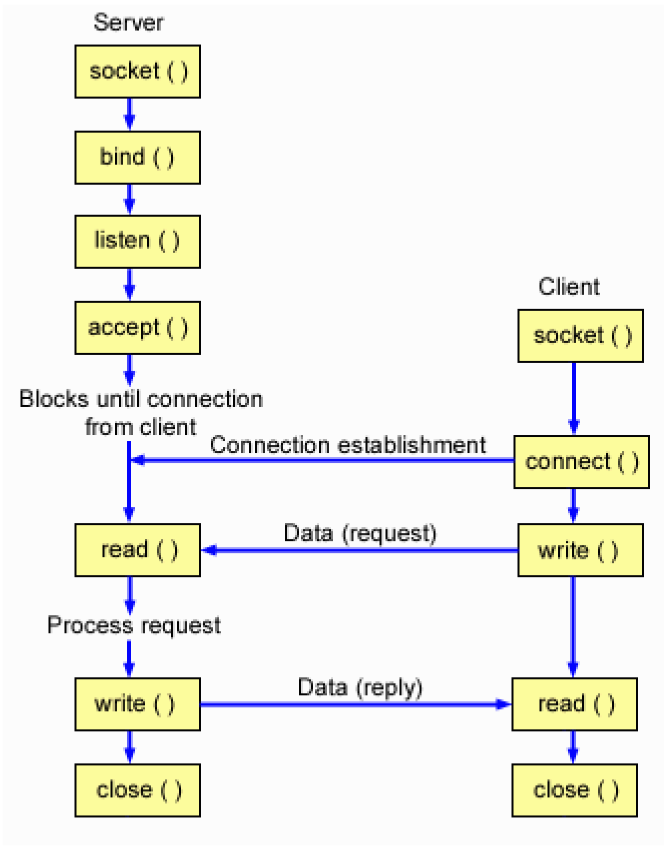
Analogous to the postal service. Packets(letters) are sent at a time to a particular destination. For greater reliability, the receiver may send an acknowledgement (a receipt for the registered letters).

Based on this two types of communication, two kinds of sockets are used:

- **stream sockets:** used for connection-oriented communication, when reliability in connection is desired.
- **datagram sockets:** used for connectionless communication, when reliability is not as much as an issue compared to the cost of providing that reliability. For eg. streaming audio/video is always send over such sockets so as to diminish network traffic.

**Sequence of System Calls for Connection Oriented communication**

The typical set of system calls on both the machines in a connection-oriented setup is shown in Figure below.



The sequence of system calls that have to be made in order to setup a connection is given below.

1. The *socket* system call is used to obtain a socket descriptor on both the client and the server. Both these calls need not be synchronous or related in the time at which they are called.The synopsis is given below:

```
#include<sys/types.h>

#include<sys/socket.h>

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

2.

3. Both the client and the server 'bind' to a particular port on their machines using the *bind* system call. This function has to be called only after a socket has been created and has to be passed the socket descriptor returned by the *socket* call. Again this binding on both the machines need not be in any particular order. Moreover the binding procedure on the client is entirely optional. The *bind* system call requires the address family, the port number and the IP address. The address family is known to be AF_INET, the IP address of the client is already known to the operating system. All that remains is the port number. Of course the programmer can specify which port to bind to, but this is not necessary. The binding can be done on a random port as well and still everything would work fine. The way to make this happen is not to call *bind* at all. Alternatively *bind* can be called with the port number set to 0. This tells the operating system to assign a random port number to this socket. This way whenever the program tries to connect to a remote machine through this socket, the operating system binds this socket to a random local port. This procedure as mentioned above is not applicable to a server, which has to listen at a standard predetermined port.

4. The next call has to be *listen* to be made on the server. The synopsis of the *listen* call is given below.

```
#include<sys/socket.h>

int listen(int skfd, int backlog);
```

5. *skfd* is the socket descriptor of the socket on which the machine should start listening. *backlog* is the maximum length of the queue for accepting requests.

6. The *connect* system call signifies that the server is willing to accept connections and thereby start communicating.

7. Actually what happens is that in the TCP suite, there are certain messages that are sent to and fro and certain initializations have to be performed. Some finite amount of time is required to setup the resources and allocate memory for whatever data structures that will be needed. In this time if another request arrives at the same port, it has to wait in a queue. Now this queue cannot be arbitrarily large. After the queue reaches a particular size limit  no more requests are accepted by the operating system. This size limit is precisely the *backlog* argument in the *listen* call and is something that the programmer can set. Today's processors are pretty speedy in their computations and memory allocations. So under normal circumstances the length of the queue never exceeds 2 or 3. Thus a *backlog* value of 2-3 would be fine, though the value typically used is around 5.Note that this call is different from the concept of "parallel" connections.The established connections are not counted in n. So, we may have 100 parallel connection running at a time when n=5.

8.

9. The *connect* function is then called on the client with three arguments, namely the socket descriptor, the remote server address and the length of the address data structure. The synopsis of the function is as follows:

```
#include<sys/socket.h>

#include<netinet/in.h> /* only for AF_INET , or the INET Domain */


int connect(int skfd, struct sockaddr* addr, int addrlen);
```

This function initiates a connection on a socket.
*skfd* is the same old socket descriptor.
*addr* is again the same kind of structure as used in the *bind* system call. More often than not, we will be creating a structure of the type *sockaddr_in* instead of *sockaddr* and filling it with appropriate data. Just while sending the pointer to that structure to the *connect* or even the *bind* system call, we cast it into a pointer to a *sockaddr* structure. The reason for doing all this is that the *sockaddr_in* is more convenient to use in case of INET domain applications. *addr* basically contains the port number and IP address of the server which the local machine wants to connect to. This call normally **blocks** until either the connection is established or is rejected.
*addrlen* is the length of the socket address structure, the pointer to which is the second argument.

10. The request generated by this *connect* call is processed by the remote server and is placed in an operating system buffer, waiting to be handed over to the application which will be calling the *accept* function. The *accept* call is the mechanism by which the networking program on the server receives that requests that have been accepted by the operating system. This synopsis of the *accept* system call is given below.

```
#include<sys/socket.h>


int accept(int skfd, struct sockaddr* addr, int addrlen);
```

*skfd* is the socket descriptor of the socket on which the machine had performed a *listen* call and now desires to accept a request on that socket.
*addr* is the address structure that will be filled in by the operating system by the port number and IP address of the client which has made this request. This *sockaddr* pointer can be type-casted to a *sockaddr_in* pointer for subsequent operations on it.
*addrlen* is again the length of the socket address structure, the pointer to which is the second argument.

This function *accept* extracts aconnection on the buffer of pending connections in the system, creates a new socket with the same properties as skfd, and returns a new file descriptor for the socket.

In fact, such an architecture has been criticized to the extent that the applications do not have a say on what connections the operating system should accept. The system accepts all requests irrespective of which IP, port number they are coming from and which application they are for. All such packets are processed and sent to the respective applications, and it is then that the application can decide what to do with that request.

The *accept* call is a blocking system call. In case there are requests present in the system buffer, they will be returned and in case there aren't any, the call simply blocks until one arrives.
This new socket is made on the same port that is listening to new connections. It might sound a bit weird, but it is perfectly valid and the new connection made is indeed a unique connection. Formally the definition of a *connection* is

**connection:** defined as a 4-tuple : (Local IP, Local port, Foreign IP, Foreign port)

For each connection at least one of these has to be unique. Therefore multiple connections on one port of the server, actually are different.

11. Finally when both *connect* and *accept* return the connection has been established.

12. The socket descriptors that are with the server and the client can now be used identically as a normal I/O descriptor. Both the *read* and the *write* calls can be performed on this socket descriptor. The *close* call can be performed on this descriptor to close the connection. Man pages on any UNIX type system will furnish further details about these generic I/O calls.

13. Variants of *read* and *write* also exist, which were specifically designed for networking applications. These are *recv* and *send*.

```
#include<sys/socket.h>


int recv(int skfd, void *buf, int buflen, int flags);

int send(int skfd, void *buf, int buflen, int flags);
```

Except for the *flags* argument the rest is identical to the arguments of the *read* and *write* calls. Possible values for the *flags* are:

| *used for* | *macro for the flag* | *comment* |
|---|---|---|
| *recv* | **MSG_PEEK** | look at the message in the buffer but do not consider it read |
| *send* | **MSG_DONT_ROUTE** | send message only if the destination is on the same network, i.e. directly connected to the local machine. |
| *recv* & *send* | **MSG_OOB** | used for transferring data out of sequence, when some bytes in a stream might be more important than others. |

14. To close a particular connection the *shutdown* call can also be used to achieve greater flexibility.

```
#include<sys/socket.h>


int shutdown(int skfd, int how);
```

*skfd* is the socket descriptor of the socket which needs to be closed.
*how* can be one of the following:

| | | |
|---|---|---|
| **SHUT_RD** | or0 | stop all read operations on this socket, but continue writing |
| **SHUT_WR** | or1 | stop all write operations on this socket, but keep receiving data |
| **SHUT_RDWR** | or2 | same as close |

A port can be reused only if it has been closed completely.

---

**Image References**

- http://publib.boulder.ibm.com/iseries/v5r1/ic2924/info/rzab6/rxab6502.gif

---

back to top
Prev| Next | Index

. **Computer Networks (CS425)**

**Instructor: Dr. Dheeraj Sanghi**

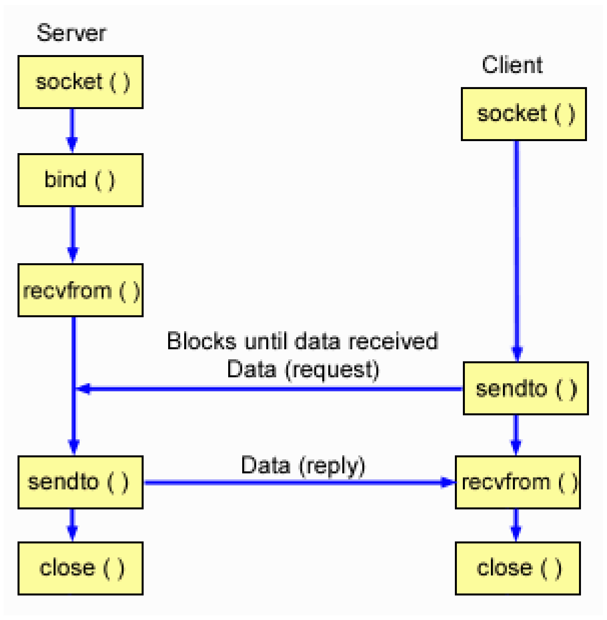
Prev | Next | Index

---

## Unix Socket Programming (Contd..)

### Sequence of System Calls for Connectionless Communication

The typical set of system calls on both the machines in a connectionless setup is shown in Figure below.



- The **socket** and **bind** system calls are called in the same way as in the connection-oriented case. Again the bind call is optional at the client side.
- The **connect** function is not called in a connectionless communication with the sane intention as above. Instead, if we call a connect() in this case, then we are simply specifying a particular server address to which we have to send, and from which we have to receive the Datagrams
- Every time a packet has to be sent over a socket, the remote address has to be mentioned. This is because there is no concept of a connection that can remember which remote machine to send that packet to.
- The calls **sendto** and **recvfrom** are used to send datagram packets. The synopses of both are given below.

int sendto(int skfd, void *buf, int buflen, int flags, struct sockaddr* to, int tolen); int recvfrom(int skfd, void *buf, int buflen, int flags, struct sockaddr* from, int fromlen);

**sendto** sends a datagram packet containing the data present in buf addressed to the address present in the **sockaddr** structure, to.

**recvfrom** fills in the buf structure with the data received from a datagram packet and the **sockaddr** structure, from with the address of the client from which the packet was received.

Both these calls block until a packet is sent in case of **sendto** and a packet is received in case of **recvfrom**. In the strict sense though **sendto** is not blocking as the packet is sent out in most cases and **sendto** returns immediately.

- Suppose if the program desires to communicate only to one particular machine and make the operating system discard packets from all other machines, it can use the connect call to specify the address of the machine with which it will exclusively communicate. All subsequent calls do not require the address field to be given. It will be understood that the remote address is the one specified in connect called earlier.

**Socket Options and Settings**

There are various options which can be set for a socket and there are multiple ways to set options that affect a socket.

Of these, **setsockopt()** system call is the one specifically designed for this purpose. Also, we can retrieve the option which are currently set for a socket by means of **getsockopt()** system call.

***int setsockopt(int socket, int level, int option_name, const void *option_value, socklen_t option_len);***

The socket argument must refer to an open socket descriptor. The level specifies who in the system is to interpret the option: the general socket code, the TCP/IP code, or the XNS code. This function sets the option specified by the option_name, at the protocol level specified by the level, to the value pointed to by the option_value for the socket associated with the file descriptor specified by the socket. The level argument specifies the protocol level at which the option resides. To set options at the socket level, we need to specify the level argument as **SOL_SOCKET**. To set options at other levels, we need to supply the appropriate protocol number for the protocol controlling the option. The option_name specifies a single option to set. The option_name and any specified options are passed uninterpreted to the appropriate protocol module for interpretations. The list of options available at the socket level **(SOL_SOCKET)** are:

- **SO_DEBUG**

Turns on recording of debugging information. This option enables or disables debugging in the underlying protocol modules. This option takes an int value. This is a boolean option.

- **SO_BROADCAST**

Permits sending of broadcast messages, if this is supported by the protocol. This option takes an int value. This is a boolean option.

- **SO_REUSEADDR**

Specifies that the rules used in validating addresses supplied to bind() should allow reuse of local addresses, if this is supported by the protocol. This option takes an int value. This is a boolean option.

- **SO_KEEPALIVE**

Keeps connections active by enabling the periodic transmission of messages, if this is supported by the protocol. This option takes an int value. If the connected socket fails to respond to these messages, the connection is broken and processes writing to that socket are notified with a SIGPIPE signal. This is a boolean option.

- **SO_LINGER**

Lingers on a close() if data is present. This option controls the action taken when unsent messages queue on a socket and close() is performed. If SO_LINGER is set, the system blocks the process during close() until it can transmit the data or until the time expires. If SO_LINGER is not specified, and close() is issued, the system handles the call in a way that allows the process to continue as quickly as possible. This option takes a linger structure, as defined in the <sys/socket.h> header, to specify the state of the option and linger interval.

- **SO_OOBINLINE**

Leaves received out-of-band data (data marked urgent) in line. This option takes an int value. This is a boolean option.

- **SO_SNDBUF**

Sets send buffer size. This option takes an int value.

- **SO_RCVBUF**

Sets receive buffer size. This option takes an int value.

- **SO_DONTROUTE**

Requests that outgoing messages bypass the standard routing facilities. The destination must be on a directly-connected network, and messages are directed to the appropriate network interface according to the destination address. The effect, if any, of this option depends on what protocol is in use. This option takes an int value. This is a boolean option.

- **SO_RCVLOWAT**

Sets the minimum number of bytes to process for socket input operations. The default value for SO_RCVLOWAT is 1. If SO_RCVLOWAT is set to a larger value, blocking receive calls normally wait until they have received the smaller of the low water mark value or the requested amount. (They may return less than the low water mark if an error occurs, a signal is caught, or the type of data next in the receive queue is different than that returned, e.g. out of band data). This option takes an int value. Note that not all implementations allow this option to be set.

- **SO_RCVTIMEO**

Sets the timeout value that specifies the maximum amount of time an input function waits until it completes. It accepts a timeval structure with the number of seconds and microseconds specifying the limit on how long to wait for an input operation to complete. If a receive operation has blocked for this much time without receiving additional data, it returns with a partial count or errno set to [EAGAIN] or [EWOULDBLOCK] if no data were received. The default for this option is zero, which indicates that a receive operation will not time out. This option takes a timeval structure. Note that not all implementations allow this option to be set.

- **SO_SNDLOWAT**

Sets the minimum number of bytes to process for socket output operations. Non-blocking output operations will process no data if flow control does not allow the smaller of the send low water mark value or the entire request to be processed. This option takes an int value. Note that not all implementations allow this option to be set.

- **SO_SNDTIMEO**

Sets the timeout value specifying the amount of time that an output function blocks because flow control prevents data from being sent. If a send operation has blocked for this time, it returns with a partial count or with errno set to [EAGAIN] ore [EWOULDBLOCK] if no data were sent. The default for this option is zero, which indicates that a send operation will not time out. This option stores a timeval structure. Note that not all implementations allow this option to be set.

For boolean options, 0 indicates that the option is disabled and 1 indicates that the option is enabled.Options at other protocol levels vary in format and name.

Some of the options available for the **IP_PROTO_TCP** socket are:

- **TCP_MAXSEG**

Returns the maximum segment size in use for the socket.The typical value for a 43.BSD socket using an Ethernet is 1024 bytes.

- **TCP_NODELAY**

When TCP is being used for a remote login,there will be many small data packets sent from the client's system to the server.Each packet can contain a single character that the user enters which is sent to the server for echoing and processing.It might be desirable to reduce the number of such small packets by combining a number of them into one big packet.But this causes a delay between the typing of a character by the user and its appearance on its monitor.This is certainly not something the user will appreciate. For such services it is desirable that the client's packets be sent as soon as they are ready.The **TCP_NODELAY** option is used for these clients to defeat the buffering algorithm, and allow the client's TCP to send small packets as soon as possible.

***int getsockopt(int socket, int level, int option_name, void *option_value, socklen_t *option_len);***

This function retrieves the value for the option specified by the option_name argument for the socket. If the size of the option value is greater than option_len, the value stored in the object pointed to by the option_value will be silently truncated. Otherwise, the object pointed to by the option_len will be modified to indicate the actual length of the value. The level specifies the protocol level at which the option resides. To retrieve options at the socket level, we need to specify the level argument as **SOL_SOCKET.** To retrieve options at other levels, we need to supply the appropriate protocol number for the protocol controlling the option. The socket in use may require the process to have appropriate privileges to use the **getsockopt()** function. The list of options for option_name is the same as those available for **setsockopt()** system call.

**Servers**

Normally a client process is started on the same system or on another system that is connected to the server's system with a network. Client processes are often initiated by the interactive user entering a command to a time-sharing system. The client process sends a request across the network to the server requesting service of some form. In this way, normally a server handles only one client at a time. If multiple client connections arrive at about the same time, the kernel queues them upto some limit, and returns them to accept function one at a time. But if the server takes more time to service each client (say a few seconds or minutes), we would need some way to overlap the service of one client with another client. Multiple requests can be handled in two ways. In fact servers are classified on the basis of the way they handle multiple requests.

1. **Iterative Servers:** When a client's request can be handled by the server in a known, finite amount of time, the server process handles the request itself. These servers handles one client at a time by iterating between them.

2. **Concurrent Servers:** These servers handle multiple clients at the same time. Among the various approaches that are available to handle these multiple clients, simplest approach is to call the UNIX fork system call , creating a child process for each client. When a connection is established, accept returns, the server calls fork, and then the child process services the client and the parent process waits for another connection. The parent closes the connected socket since the child handles this new client.

**Internet Superserver (inetd)**

Servers must keep running at all times. However, all the servers are not working all this time but merely waiting for a request from a client. To avoid this waste of resources, a single server is run which waits on all the port numbers. This Internet super server is called **inetd** in UNIX. It is referred to as the ``Internet Super-Server'' because it manages connections for several daemons. Programs that provide network service are commonly known as daemons. **inetd** serves as a managing server for other daemons. When a connection is received by **inetd,** it determines which daemon the connection is destined for, spawns the particular daemon and delegates the socket to it. Running one instance of **inetd** reduces the overall system load as compared to running each daemon individually in stand-alone mode. This daemon provides two features -

**1.** It allows a single process **(inetd)** to be waiting to service multiple connection requests, instead of one process for each potential service. This reduces the total number of processes in the system.

**2.** It simplifies the writing of the server processes to handle the requests, since many of the start-up details are handled by **inetd.**

The **inetd** process uses fork and exec system calls to invoke the actual server process.The only way the server can obtain the identity of the client is by calling the **getpeername** system call.

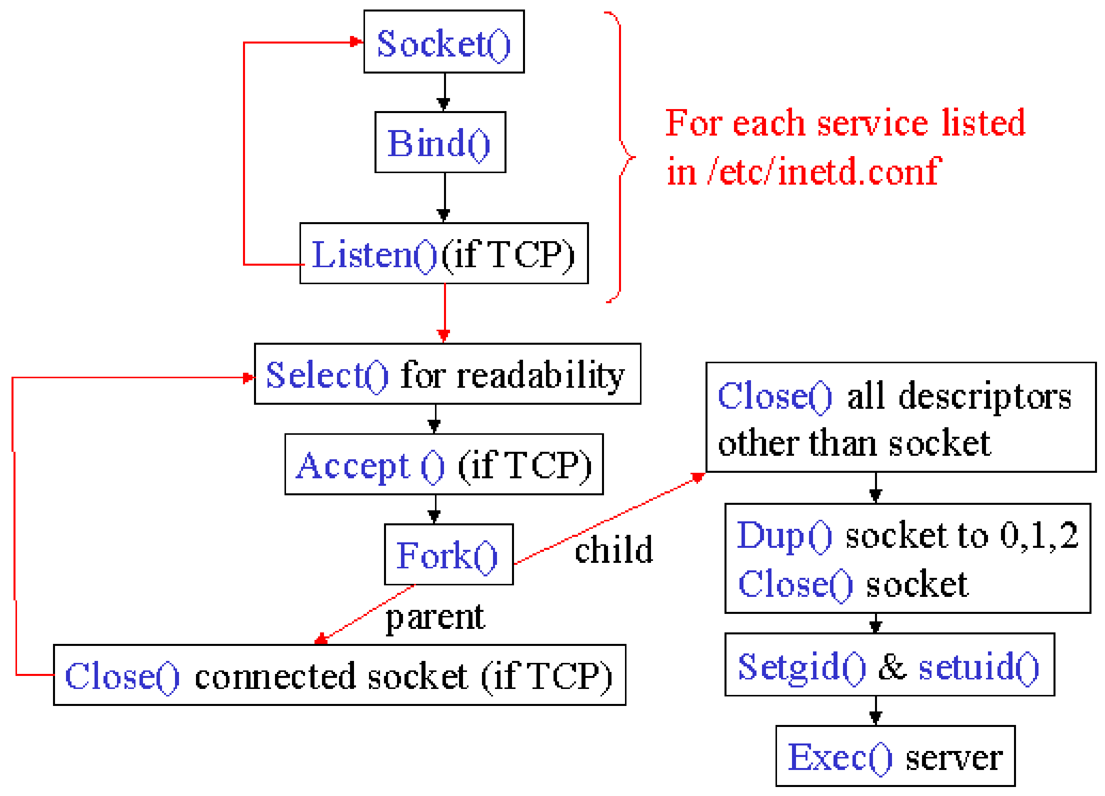
***int getpeername(int sockfd, struct sockaddr *peer, int *addrlen);***

**Getpeername** returns the name of the peer connected to socket *sockfd*. The *addrlen* parameter should be initialised to indicate the amount of space pointed to by peer . On return it contains the actual size of the name returned (in bytes). The name is truncated if the buffer provided is too small.

A similar system call is **getsockname. Getsockname** returns the current name for the specified socket. The *addrlen* parameter should be initialized to indicate the amount of space pointed to by name. On return it contains the actual size of the name returned (in bytes).

***int getsockname(int sockfd, struct sockaddr *name, int *addrlen);***

Following illustrates the steps performed by **inetd**



However, the **inetd** process has its own disadvantages. The code of a server is to be copied from the disk to memory each time it is execed, and this is expensive. So, if there is an application which is called frequently, like e-mail server, the above mentioned approach (using **inetd**) is not recommended.

---

**Image References**

- http://publib.boulder.ibm.com/iseries/v5r1/ic2924/info/rzab6/rxab6503.gif
- http://www.cs.odu.edu/~cs779/spring03/lectures/inetd.gif

---

**Computer Networks (CS425)**

**Instructor: Dr. Dheeraj Sanghi**

---

**Unix Socket Programming (Contd..)**

**Multiple Sockets**

Suppose we have a process which has to handle multiple sockets. We cannot simply read from one of them if a request comes, because that will block while waiting on the request on that particular socket. In the meantime a request may come on any other socket. To handle this input/output multiplexing we could use different techniques :

1. **Busy waiting:** In this methodology we make all the operations on sockets non-blocking and handle them simultaneously by doing polling. For example, we could use the read() system call this way and read from all the sockets together. The disadvantage in this is that we waste a lot of CPU cycles. To make the system calls non-blocking we use:

fcntl (s, f_setfl, fndelay);

2. **Asynchronous I/O:** Here we ask the Operating System to tell us whenever we are waiting for I/O on some sockets. The Operating System sends a signal whenever there is some I/O. When we receive a signal, we will have to check all sockets and then wait till the next signal comes. But there are two problems - first, the signals are expensive to catch and second, we would not be able to know if an input comes on a socket when we are doing I/O on another one. For Asynchronous I/O, we have a different set of commands (here we give the ones for UNIX with a VHD variant):

signal(sigio, io_handler); fcntl(s, f_setown, getpid()); fcntl(s, f_setfl, fasync);

3. **Separate process for each I/O:** We could as well fork out 10 different child processes for 10 different sockets. These child processes are very light weight and have some communication between them. Now these processes waiting on each socket can have blocking system calls. This wastes a lot of memory, data structures and other resources.

4. **Select() system call:** We can use the select system call to instruct the Operating System to wait for any one of multiple events to occur and to wake up the process only if one of these events occur. This way we would know that the I/O request has come from which socket.

int select(int nfds, fd_set *readfds, fd_set *writefds, fd_set *errorfds, struct timeval *timeout); void FD_CLR(int fd, fd_set *fdset); int FD_ISSET(int fd, fd_set *fdset); void FD_SET(int fd, fd_set *fdset); void FD_ZERO(fd_set *fdset);

The **select()** function indicates which of the specified file descriptors is ready for reading, ready for writing, or has an error condition pending. If the specified condition is false for all of the specified file descriptors, **select()** blocks up to the specified timeout interval, until the specified condition is true for at least one of the specified file descriptors. The nfds argument specifies the range of file descriptors to be tested. The **select()** function tests file descriptors in the range of 0 to nfds-1. **readfds, writefds and errorfds** arguments point to an object of type **fd_set**. **readfds** specifies the file descriptors to be checked for being ready to read. **writefds** specifies the file descriptors to be checked for being ready to write, **errorfds** specifies the file descriptors to be checked for error conditions pending.

On successful completion, the objects pointed to by the **readfds, writefds, and errorfds** arguments are modified to indicate which file descriptors are ready for reading, ready for writing, or have an error condition pending, respectively. For each file descriptor less than nfds, the corresponding bit will be set on successful completion if it was set on input and the associated condition is true for that file descriptor. The timeout is an upper bound on the amount of time elapsed before select returns.

It may be zero, causing select to return immediately. If the timeout is a null pointer, **select()** blocks until an event causes one of the masks to be returned with a valid (non-zero) value. If the time limit expires before any event occurs that would cause one of the masks to be set to a non-zero value, **select()** completes successfully and returns 0.

**Reserved Ports**

Port numbers from 1-1023 are reserved for the superuser and the rest of the ports starting from 1024 are for other users. But we have a finer division also which is as follows :

- 1 to 511 - these are assigned to the processes run by the superuser
- 512 to 1023 - they are used when we want to assign ports to some important user or process but want to show that this is a reserved superuser port
- 1024 to 5000 - they are system assigned random ports
- 5000 to FFFF - they are used to assign a port to user processes or sockets used by users

---

**Some Topics in TCP**

**Acknowledgement Ambiguity Problem**

There can be an ambiguous situation during the Retransmission time-out for a packet . Such a case is described below:
The events are :
1. A packet named 'X' is sent at time 't1' for the first time .
2. Timeout occours for 'X' and acknowledgement is not recieved .
3. So 'X' will be retransmitted at time 't2' .
4. The acknowledgment arrives at 't' .
In this case , we cannot be sure wether the acknowledgement recieved at 't' is for the packet 'X' sent at the time 't1' or 't2'. What should be our RTT sample value?

If we take ('t'-'t1' ) as the RTT sample :
t2-t1 (timeout period) is typically much greater than the average RTT . This implies that if we take t-t1 , then it will tend to increase the average RTT . If this happens for a large number of packets then the RTT will increase significantly. Now as the RTT increases , the timeout value increases and the damage caused by the above sequence of events increases . So this may cause the timeout to become very large unnecessarily .

If we take 't'-'t2' as the RTT sample :
Consider the case in which the acknowledgement for a packet takes a lot more time to arrive as compared to the timeout value . That is , the acknowledgement that arrives is for the packet which had been sent at the time 't1'.
That implies t-t2 is likely to be smaller and hence will tend to decrease the value of the RTT when included as a sample . A string of such events would cause the timeout to decrease . As the timeout decreases , the above sequence of events becomes more probable leading to even a further decrease in timeout . So if we consider t-t2, the timeout may become very small .

Since both the cases may lead to some problems , one possible solution is to discard the sample for

which timeout occours . But this can't be done!!If the packet gets lost , the network is most probaaly congested The previous estimate of RTT is now meaningless as it was for an uncongested network and the characteristics of the network have changed Also new samples cant be found due to the above ambiguity .
So we simply adopt the policy of doubling the value of RTO on packet loss . When the congestion in the network subisdes , then we can start sampling afresh or we can go back to the state before the congestion occurred .
Note that this is a temporary increase in the value of RTO, as we have not increased the value of RTT. So, the present RTT value will help us to go back to the previous situation when it becomes normal. This is called the **Acknowledgment Ambiguity Problem**.

**Fast Restransmit**

TCP may generate an immediate acknowledgment (a duplicate ACK) when an out- of-order segment is received. This duplicate ACK should not be delayed. The purpose of this duplicate ACK is to let the other end know that a segment was received out of order, and to tell it what sequence number is expected. Since TCP does not know whether a duplicate ACK is caused by a lost segment or just a reordering of segments, it waits for a small number of duplicate ACKs to be received. It is assumed that if there is just a reordering of the segments, there will be only one or two duplicate ACKs before the reordered segment is processed, which will then generate a new ACK. If three or more duplicate ACKs are received in a row, it is a strong indication that a segment has been lost. TCP then performs a retransmission of what appears to be the missing segment, without waiting for a retransmission timer to expire.

This algorithm is one of the algorithms used by the TCP for the purpose of Congestion/Flow Control. Let us consider ,a sender has to send packets with sequence numbers from 1 to 10 (Please note ,in TCP the bytes are given sequence numbers, but for the sake of explanation the example is given). Suppose packet 4 got lost.

How will the sender know that it is lost ?

The sender must have received the cumulative acknowledgment for packet 3. Now, time-out for packet 4 occurs. On the receiver side, the packet 5 is received. As it is an out of sequence packet, the duplicate acknowledgment (thanks to it's cumulative nature) is not delayed and sent immediately. The purpose of this duplicate ACK is to let the other end know that a segment was received out of order, and to tell it what sequence number is expected. So, now the sender sees a duplicate ACK. But can it be sure that the packet 4 was lost ?

Well, no as of now. Various situations like duplication of packet 3 or ACK itself, delay of packet 4 or receipt of an out of sequence packet etc. might have resulted in a duplicate ACK.

So, what does it do? Wait for more!!! Yeah, it waits for more duplicate packets. It is assumed that if there is just a reordering of the segments, there will be only one or two duplicate ACKs before the reordered segment is processed, which will then generate a new ACK. If three or more duplicate ACKs are received in a row, it is a strong indication that a segment has been lost. TCP then performs a retransmission of what appears to be the missing segment, without waiting for a retransmission timer to expire. This is called **Fast Retransmit**.

**Flow Control**

Both the sender and the receiver can specify the number of packets they are ready to send/receive. To implement this, the receiver advertises a Receive Window Size. Thus with every acknowledgment, the receiver sends the number of packets that it is willing to accept.

Note that the size of the window depends on the available space in the buffer on the receiver side. Thus, as the application keeps consuming the data, window size is incremented.

On the sender size, it can use the acknowledgment and the receiver's window size to calculate the sequence number up to which it is allowed to transmit. For ex. If the acknowledgment is for packet 3 and the window size is 7, the sender knows that the recipient has received data up to packet 3 and it can send packets of sequence number up to (7+3=10).

The problem with the above scheme is that it is too fast. Suppose, in the above example, the sender sends 7 packets together and the network is congested. So, some packets may be lost. The timer on the sender side goes off and now it again sends 7 packets together, thus increasing the congestion further more. It only escalates the magnitude of the problem.

---

back to top
Prev| Next | Index

lows.

**Computer Networks (CS425)**

**Instructor: Dr. Dheeraj Sanghi**

Prev | Next | Index

---

**Topics in TCP**

## TCP Congestion Control

If the receiver advertises a large window-size , larger than what the network en route can handle , then there will invariably be packet losses. So there will be re-transmissions as well . However , the sender cannot send all the packets for which ACK has not been received because this way it will be causing even more congestion in the network. Moreover , the sender at this point of time cannot be sure about how many packets have actually been lost . It might be that this is the only one that has been lost , and some following it have actually been received and buffered by the receiver. In that case , the sender will have unnecessarily sent a number of packets.

So the re-transmission of the packets also follows slow-start mechanism. However , we do indeed need to keep an upper bound on the size of the packets as it increases in slow start, to prevent it from increasing unbounded and causing congestion. This cap is put at half the value of the segment size at which packet loss started.

### Congestion Window

We have already seen one bound on the size of the segments sent by the receiver-namely , the receiver window that the receiver advertises . However there could be a bottleneck created by some intermediate network that is getting clogged up. The net effect is that just having the receiver window is not enough. There should be some bound relating to the congestion of the network path - congestion window captures exactly this bound. Similar to receiver window, we have another window , the Congestion Window , and the maximum size of the segments sent are bounded by the minimum of the sizes of the two windows. E.g. If the receiver says "send 8K" (size of the receiver window ) , but the sender knows that bursts of more than 4K (size of congestion window ) clog the network up, then it sends 4K. On the other hand , if the congestion window was of size 32K , then the sender would send segments of maximum size 8K.

**How do we calculate/manage the Congestion Window ?**

The size of the congestion window is initialized to 1.For every ACK received , it is incremented by 1. Another field that we maintain is threshold which is equal to half the size of the congestion window. Whenever a packet loss takes place, the congestion window is set to 1.Then we keep increasing the congestion window by 1 on every ACK received till we reach the threshold value. Thereafter, we increment the congestion window size by 1 after every round trip time.

Notice that TCP always tries to keep the flow rate slightly below the maximum value. So if the network traffic fluctuates slightly, then a lot of packets might be lost. Packet losses cause a terrible loss in throughput.

In all these schemes, we have been assuming that any packet loss occurs only due to network congestion. What happens if some packet loss occurs not due to some congestion but due to some random factors?

When a packet is lost, the congestion window size is set to 1. Then when we retransmit the packet, if we receive a cumulative ACK for a lot of subsequent packets, we can assume that the packet loss was not due to congestion, but because of some random factors. So we give up slow start and straightaway set the size of Congestion Window to the threshold value.

**Silly Window Syndrome**

This happens when the application supplying data to the sender does do in large chunks, but the application taking data from receiver (probably an interactive application) does it in very small chunks, say 1 byte at a time. The sender keeps advertising windows of size 1 byte each as the application consumes the bytes one at a time.

**Clark's Solution to this problem**

We try to prevent the sender from advertising very small windows. The sender should try to wait until it has accumulated enough space in the window to send a full segment or half the receiver's buffer size, which it can estimate from the pattern of window updates that it received in the past.

**Another problem:** What if the same behavior is shown by an interactive application at the sender's end ? That is , what if the sender keeps sending in segments of very small size?

**Nagle's algorithm**

when data comes to the sender one byte at a time , send the first byte and buffer all the remaining bytes till the outstanding byte is acknowledged. Then send all the buffered characters in one segment and start buffering again till they are acknowledged. It can help reduce the bandwidth usage for example when the user is typing quickly into a telnet connection and the network is slow .

**Persistent Timer**

Consider the following deadlock situation . The receiver sends an ACK with 0 sized window, telling the sender to wait. Later it send an ACK with non-zero window, but this ACK packet gets lost. Then both the receiver and the sender will be waiting for each other to do something. So we keep another timer. When this timer goes off, the sender transmits a probe packet to the sender with an ACK number that is old. The receiver responds with an ACK with updated window size and transmission resumes.

Now we look at the solution of the last two problems ,namely **Problem of Random Losses** and **Sequence Number Wrap Around**.

**Problem of Random Losses**

How do we know if a loss is a congestion related loss or random loss ?If our window size is very large then we cannot say that one packet loss is random loss.So we need to have some mechanism to find what packets are lost. Cumulative Acknowledgement is not a good idea for this.

**Solutions**

**Selective Acknowledgement**

We need a selective acknowledgement but that creates a problem in TCP because we use byte sequence numbers .So what we we do is that we send the sequence number and the length. We may have to send a large number of such Selective Acknowledgements which will increase the overhead So whenever we get out of sequence packets we send the information a few time not in all the packets anyway. So we cannot rely on Selective Acknowledgement anyway. If we have 32 bit sequence number and 32 bit length,then already we will have too much of overhead .One proposal is to use 16 bit length field. If we have very small gaps then we will think that random losses are there and we need to fill them .If large gaps are there we assume that congestion is there and we need to slow down.

**TCP Timestamps Option**

TCP is a symmetric protocol, allowing data to be sent at any time in either direction, and therefore timestamp echoing may occur in either direction. For simplicity and symmetry, we specify that timestamps always be sent and echoed in both directions. For efficiency, we combine the timestamp and timestamp reply fields into a single TCP Timestamps Option.

```
Kind: 8
Length: 10 bytes
+-------+-------+---------------------+---------------------+
|Kind=8 |  10   |      TS Value       |    TS Echo Reply    |
+-------+-------+---------------------+---------------------+
    1       1            4                     4     (length in bytes)
```

The Timestamps option carries two four-byte timestamp fields. The Timestamp Value field (TSval) contains the current value of the timestamp clock of the TCP sending the option. The Timestamp Echo Reply field (TSecr) is only valid if the **ACK** bit is set in the TCP header; if it is valid, it echos a times- tamp value that was sent by the remote **TCP** in the TSval field of a Timestamps option. When TSecr is not valid, its value must be zero. The TSecr value will generally be the time stamp for the last in-sequence packet received.

Example:

| | | | | | | |
|---|---|---|---|---|---|---|
| Sequence of packet send : | 1 (t1) | 2 (t2) | 3 (t3) | 4 (t4) | 5 (t5) | 6 (t6) |
| sequence of packets received: | 1 | 2 | 4 | 3 | 5 | 6 |
| time stamp copied in **ACK:** | | t1 | t2 | t3 | | |

**PAWS: Protect Against Wrapped Sequence Numbers**

PAWS operates within a single TCP connection, using state that is saved in the connection control block. PAWS uses the same TCP Timestamps option as the RTTM mechanism described earlier, and assumes that every received TCP segment (including data and ACK segments) contains a timestamp **SEG.TSval** whose values are monotone non-decreasing in time. The basic idea is that a segment can be discarded as an old duplicate if it is received with a timestamp **SEG.TSval** less than some timestamp recently received on this connection.

In both the PAWS and the RTTM mechanism, the "timestamps" are 32- bit unsigned integers in a modular 32-bit space. Thus, "less than" is defined the same way it is for TCP sequence numbers, and the same implementation techniques apply. If s and t are timestamp values, s < t if 0 < (t - s) < 2**31, computed in unsigned 32-bit arithmetic.

The choice of incoming timestamps to be saved for this comparison must guarantee a value that is monotone increasing. For example, we might save the timestamp from the segment that last advanced the left edge of the receive window, i.e., the most recent in- sequence segment. Instead, we choose the value **TS.Recent** for the RTTM mechanism, since using a common value for both PAWS and RTTM simplifies the implementation of both. **TS.Recent** differs from the timestamp from the last in-sequence segment only in the case of delayed ACKs, and therefore by less than one window. Either choice will therefore protect against sequence number wrap-around.

RTTM was specified in a symmetrical manner, so that TSval timestamps are carried in both data and ACK segments and are echoed in TSecr fields carried in returning ACK or data segments. PAWS submits all incoming segments to the same test, and therefore protects against duplicate ACK segments as well as data segments. (An alternative un-symmetric algorithm would protect against old duplicate ACKs: the sender of data would reject incoming ACK segments whose TSecr values were less than the TSecr saved from the last segment whose ACK field advanced the left edge of the send window. This algorithm was deemed to lack economy of mechanism and symmetry.)

TSval timestamps sent on {SYN} and {SYN,ACK} segments are used to initialize PAWS. PAWS protects against old duplicate non-SYN segments, and duplicate SYN segments received while there is a synchronized connection. Duplicate {SYN} and {SYN,ACK} segments received when there is no connection will be discarded by the normal 3-way handshake and sequence number checks of TCP.

**Header Prediction**

As we want to know that from which TCP connection this packet belongs. So for each new packet we have to match the header of each packet to the database that will take a lot of time so what we do is we first compare this header with the header of last received packet and on an average this will reduce the work. Assuming that this packet is from the same TCP connection from where we have got the last one (locality principal).

---

back to top
Prev| Next | Index

acing="0" style="border-collapse: collapse" bordercolor="#111111" width="100%" id="AutoNumber11">

---

## UDP (User Datagram Protocol)

UDP -- like its cousin the Transmission Control Protocol (TCP) -- sits directly on top of the base Internet Protocol (IP). In general, UDP implements a fairly "lightweight" layer above the Internet Protocol. It seems at first site that similar service is provided by both UDP and IP, namely transfer of data.But we need UDP for multiplexing/demultiplexing of addresses.

UDP's main purpose is to abstract network traffic in the form of datagrams. A datagram comprises one single "unit" of binary data; the first eight (8) bytes of a datagram contain the header information and the remaining bytes contain the data itself.

UDP Headers
The UDP header consists of four (4) fields of two bytes each:

| | |
|---|---|
| Source Port | Destination Port |
| length | checksum |

- source port number
- destination port number
- datagram size
- checksum

UDP port numbers allow different applications to maintain their own "channels" for data; both UDP and TCP use this mechanism to support multiple applications sending and receiving data concurrently. The sending application (that could be a client or a server) sends UDP datagrams through the source port, and the recipient of the packet accepts this datagram through the destination port. Some applications use static port numbers that are reserved for or registered to the application. Other applications use dynamic (unregistered) port numbers. Because the UDP port headers are two bytes long, valid port numbers range from 0 to 65535; by convention, values above 49151 represent dynamic ports.

The datagram size is a simple count of the number of bytes contained in the header and data sections . Because the header length is a fixed size, this field essentially refers to the length of the variable-sized data portion (sometimes called the payload). The maximum size of a datagram varies depending on the operating environment. With a two-byte size field, the theoretical maximum size is

65535 bytes. However, some implementations of UDP restrict the datagram to a smaller number -- sometimes as low as 8192 bytes.

UDP checksums work as a safety feature. The checksum value represents an encoding of the datagram data that is calculated first by the sender and later by the receiver. Should an individual datagram be tampered with (due to a hacker) or get corrupted during transmission (due to line noise, for example), the calculations of the sender and receiver will not match, and the UDP protocol will detect this error. The algorithm is not fool-proof, but it is effective in many cases. In UDP, check summing is optional -- turning it off squeezes a little extra performance from the system -- as opposed to TCP where checksums are mandatory. It should be remembered  that check summing is optional only for the sender, not the receiver. If the sender has used checksum then it is mandatory for the receiver to do so.

Usage of the Checksum in UDP is optional. In case the sender does not use it, it sets the checksum field to all 0's. Now if the sender computes the checksum then the recipient must also compute the checksum an set the field accordingly. If the checksum is calculated and turns out to be all 1's then the sender sends all 1's instead of all 0's. This is since in the algorithm for checksum computation used by UDP, a checksum of all 1's if equivalent to a checksum of all 0's. Now the checksum field is unambiguous for the recipient, if it is all 0's then checksum has not been used, in any other case the checksum has to be computed.

---

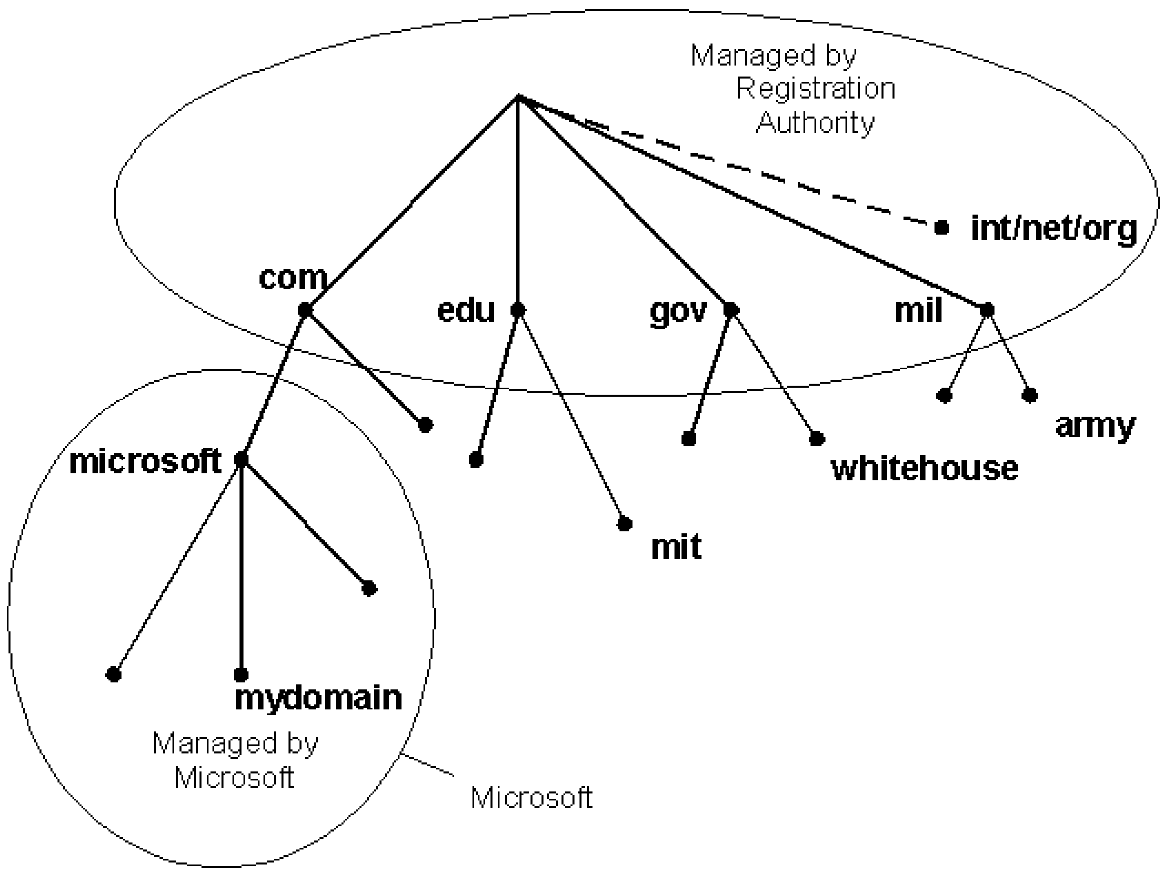
**DNS (Domain Name Service)**

The internet primarily uses IP addresses for locating nodes. However, its humanly not possible for us to keep track of the many important nodes as numbers. Alphabetical names as we see would be more convenient to remember than the numbers as we are more familiar with words. Hence, in the chaotic organization of numbers (IP addresses) we would be much relieved if we can use familiar sounding names for nodes on the network.

There is also another motivation for DNS. All the related information about a particular network (generally maintained by an organization, firm or university) should be available at one place. The organization should have complete control over what it includes in its network and how does it "organize" its network. Meanwhile, all this information should be available transparently to the outside world.

Conceptually, the internet is divide into several hundred top level domains where each domain covers many hosts. Each domain is partitioned in subdomains which may be further partitioned into subsubdomains and so on... So the domain space is partitioned in a tree like structure as shown below. It should be noted that this tree hierarchy has nothing in common with the IP address hierarchy or organization.

The internet uses a hierarchical tree structure of Domain Name Servers for IP address resolution of a host name.



The top level domains are either generic or names of countries. eg of generic top level domains are .edu .mil .gov .org .net .com .int etc. For countries we have one entry for each country as defined in ISO3166. eg. .in (India) .uk (United Kingdom).

The leaf nodes of this tree are target machines. Obviously we would have to ensure that the names in a row in a subdomain are unique. The max length of any name between two dots can be 63 characters. The absolute address should not be more than 255 characters. Domain names are case insensitive. Also in a name only letters, digits and hyphen are allowed. For eg. www.iitk.ac.in is a domain name corresponding to a machine named www under the subsubdomain iitk.ac.in.

**Resource Records:**
Every domain whether it is a single host or a top level domain can have a set of resource records associated with it. Whenever a resolver (this will be explained later) gives the domain name to DNS it gets the resource record associated with it. So DNS can be looked upon as a service which maps domain names to resource records. Each resource record has five fields and looks as below:

| Domain Name | Class | Type | Time to Live | Value |
|---|---|---|---|---|

- Domain name: the domain to which this record applies.
- Class: set to IN for internet information. For other information other codes may be specified.
- Type: tells what kind of record it is.
- Time to live: Upper Limit on the time to reach the destination

- Value: can be an IP address, a string or a number depending on the record type.

---

**Image Referecnes**

- http://www.microsoft.com/technet/images/prodtechnol/windows2000serv/plan/images/w2kdns201_BIG.gif

---

back to top
Prev| Next | Index

ave received the cumulative acknowledgment for

**Computer Networks (CS425)**

**Instructor: Dr. Dheeraj Sanghi**

Prev | Next | Index

---

**DNS (Contd...)**

**Resource Record**

A **Resource Record** (RR) has the following:

- **owner** which is the domain name where the RR is found.
- **type** which is an encoded 16 bit value that specifies the type of the resource in this resource record. It can be one of the following:
    - **A** a host address
    - **CNAME** identifies the canonical name of an alias
    - **HINFO** identifies the CPU and OS used by a host
    - **MX** identifies a mail exchange for the domain.
    - **NS** the authoritative name server for the domain
    - **PTR** a pointer to another part of the domain name space
    - **SOA** identifies the start of a zone of authority class which is an encoded 16 bit value which identifies a protocol family or instance of a protocol.
- **class** One of: **IN** the Internet system or **CH** the Chaos system
- **TTL** which is the time to live of the RR. This field is a 32 bit integer in units of seconds, an is primarily used by resolvers when they cache RRs. The TTL describes how long a RR can be cached before it should be discarded.
- **RDATA** Data in this field depends on the values of the type and class of the RR and a description for each is as follows:

- for A: For the IN class, a 32 bit IP address For the CH class, a domain name followed by a 16 bit octal Chaos address.
- for CNAME: a domain name.
- for MX: a 16 bit preference value (lower is better) followed by a host name willing to act as a mail exchange for the owner domain.
- for NS: a host name.
- for PTR: a domain name.
- for SOA: several fields.

**Note:** While short TTLs can be used to minimize caching, and a zero TTL prohibits caching, the realities of Internet performance suggest that these times should be on the order of days for the typical host. If a change can be anticipated, the TTL can be reduced prior to the change to minimize inconsistency during the change, and then increased back to its former value following the change. The data in the RDATA section of RRs is carried as a combination of binary strings and domain names. The domain names are frequently used as "pointers" to other data in the DNS.

**Aliases and Cannonical Names**

Some servers typically have multiple names for convenience. For example www.iitk.ac.in & yamuna.iitk.ernet.in identify the same server. In addition multiple mailboxes might be provided by some organizations. Most of these systems have a notion that one of the equivalent set of names is the canonical or primary name and all others are aliases.

When a name server fails to find a desired RR in the resource set associated with the domain name, it checks to see if the resource set consists of a CNAME record with a matching class. If so, the name server includes the CNAME record in the response and restarts the query at the domain name specified in the data field of the CNAME record.

**Name Servers**

Name servers are the repositories of information that make up the domain database. The database is divided up into sections called zones, which are distributed among the name servers. Name servers can answer queries in a simple manner; the response can always be generated using only local data, and either contains the answer to the question or a referral to other name servers "closer" to the desired information. The way that the name server answers the query depends upon whether it is operating in recursive mode or iterative mode:

- The simplest mode *for the server* is non-recursive, since it can answer queries using only local information: the response contains an error, the answer, or a referral to some other server "closer" to the answer. All name servers must implement non-recursive queries.
- The simplest mode *for the client* is recursive, since in this mode the name server acts in the role of a resolver and returns either an error or the answer, but never referrals. This service is optional in a name server, and the name server may also choose to restrict the clients which can use recursive mode.

**Recursive Query vs Iterative Query**

If the server is supposed to answer a recursive quesry then the response is either the reource record data or a error code. A server operating in this mode will never return the name of any forwarding name server but will contact the appropiate name server itself and try to get the information.

In iterative mode, on the other hand, if the server does not have the information requested locally then it return the address of some name server who might have the information about the query. It is then the responsibility of the contacting application to contact the next name server to resolve its query and do this iteratively until gets an answer or and error.

**Relative Names**

In place of giving full DNS names like cu2.cse.iitk.ac.in or bhaskar.cc.iitk.ac.in one can give just cu2 or bhaskar.This can be used by the server side as well as the client side.But for this one has to manually specify these extensions in the database of the servers holding the resource records.

---

**BOOTP**

The BOOTP uses UDP/IP. It is run when the machine boots. The protocol allows diskless machines to discover their IP address and the address of the server host. Additionally name of the file to be loaded from memory and executed is also supplied to the machine. This protocol is an improvement over RARP which has the follwing limitations:

1. Networks which do not have a broadcast method can't support RARP as it uses the broadcast method of the MAC layer underneath the IP layer.
2. RARP is heavily dependent on the MAC protocol.
3. RARP just supplies the IP address corresponding to a MAC address It doesn't support respond with any more data.
4. RARP uses the computer hardware's address to identify the machine and hence cannot be used in networks that dynamically assign hardware addresses.

**Events in BOOTP**

1. The Client broadcasts its MAC address (or other unique hardware identity number) asking for help in booting.
2. The BOOTP Server responds with the data that specifies how the Client should be configured (pre-configured for the specific client)

**Note:** BOOTP doesn't use the MAC layer broadcast but uses UDP/IP.

**Configuration Information**

The important informations provided are:

- IP address
- IP address of the default router for that particular subnet
- Subnet mask
- IP addresses of the primary and secondary nameservers

Additionaly it may also provide:

- Time offset from GMT
- The IP address of a time server
- The IP address of a boot server
- The name of a boot file (e.g. boot image for X terminals)
- The IP domain name for the client

But the problem with BOOTP is that it again can't be used for the dynamic IP's as in RARP servers.For getting dynamic IP's we use DHCP.

---

**DHCP (Dynamic Host Configuration Protocol)**

DHCP (Dynamic Host Configuration Protocol) is a protocol that lets network administrators manage centrally and automate the assignment of Internet Protocol (IP) addresses in an organization's network. If a machine uses Internet's set of protocol (TCP/IP), each machine that can connect to the Internet needs a unique IP address. When an organization sets up its computer users with a connection to the Internet, an IP address must be assigned to each machine. Without DHCP, the IP address must be entered manually at each computer and, if computers move to another location in another part of the network, a new IP address must be entered. DHCP lets a network administrator supervise and distribute IP addresses from a central point and automatically sends a new IP address when a computer is plugged into a different place in the network.

**IP Address Allocation Mechanism**

DHCP supports three mechanisms for IP address allocation.

- **Automatic allocation:** DHCP assigns a permanent IP address to a host.
- **Dynamic allocation:** DHCP assigns an IP address to a host for a limited period of time (or until the host explicitly relinquishes the address).
- **Manual allocation:** Host's IP address is assigned by the network administrator, and DHCP is used simply to convey the assigned address to the host. A particular network will use one or more of these mechanisms, depending on the policies of the network administrator.

**Messages Used by DHCP**

- **DHCP Discover** - Client broadcast to locate available servers. It is assumed atleast one of the servers will have resources to fulfill the request.( may include additional pointers to specific services required eg. particular subnet, minimum time limit etc ).
- **DHCP Offer -** Server to client in response to DHCP Discover with offer of configration parameters.
- **DHCP Request -** Client broadcast to servers requesting offered parameters from one server and implicitly declining offers from all others.( also important in case of lease renewal if the alloted time is about to expire ).
- **DHCP Decline -** Client to server indicating configration parameters invalid.

- **DHCP Release -** Client to server relinquishing network address and cancelling current lease.( in case of a graceful shut down DHCP server is sent a DHCP Release by the host machine).
- **DHCP Ack -** Server to client with configration parameters, including committed Network address.
- **DHCP Nack -** Server to client refusing request for configratin parameters (eg. requested network address already allocated).

**Timers Used**

Note that lease time is the time specified by the server for which the services have been provided to the client.

- **Lease Renewal Timer -** When this timer expires machine will ask the server for more time sending a DHCP Request.
- **Lease Rebinding Timer -** Whenever this timer expires, we have not been receiving any response from the server and so we can assume the server is down. Thus send a DHCP Request to all the servers using IP Broadcast facility. This is only point of difference between Lease renewal and rebinding.
- **Lease Expiry Timer -** Whenever this timer expires, the system will have to start crashing as the host does not have a valid IP address in the network.

**Timer Configuration Policy**

The timers have this usual setting which can be configured depending upon the usage pattern of the network. An example setting has been discussed below.

Lease Renewal = 50 % Lease time
Lease Rebinding = 87.5 % Lease time
Lease Expiry = 100 % Lease time

---

back to top
Prev| Next | Index

er>

**Computer Networks (CS425)**

**Instructor: Dr. Dheeraj Sanghi**

Prev | Next | Index

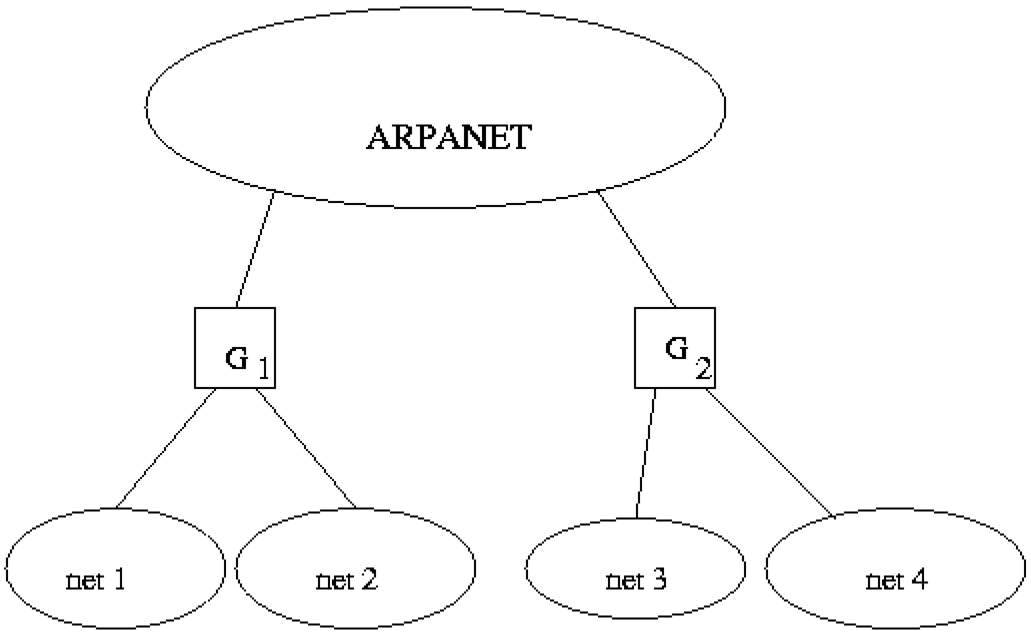
---

**Routing in Internet**

**The Origin of Internet**

The response of Internet to the issue of choosing routing tables with complete/par tail information is shown by the following architecture. There are a few nodes having complete routing information and a large number of nodes with partial information. The nodes with complete information, called core gateways, are well connected by a Backbone Network. These nodes talk to each other to keep themselves updated. The non-core gateways are connected to the core gateways. (Historically, this architecture comes from the ARPANET.)

The original internet was structured around a backbone of ARPANET with several core gateways connected to it .These core gateways connected some Local Area Networks (LANs) to the rest of the network. These core gateways talked to themselves and exchanged routing information's. Every core gateway contained complete information about all possible destinations.



**How do you do routing ?**

The usual IP routing algorithm employs an internet routing table (some times called an IP routing table) on each machine that Stores the information about the possible destinations, and how to reach them.

**Default Routes**

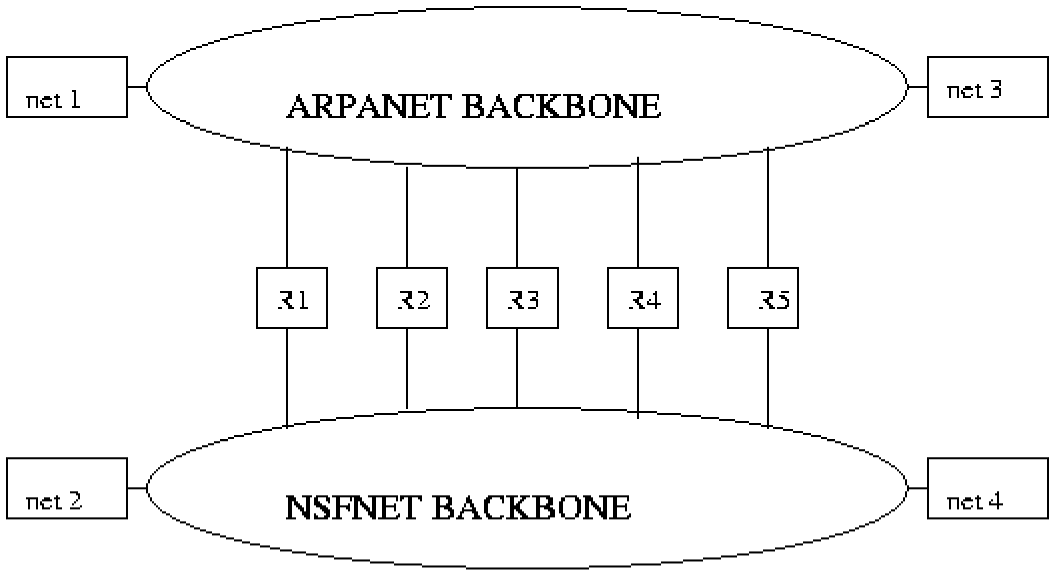
This technique used to hide information and keep routing table size small consolidates multiple entries into a default case. If no route appears in the routing table, the routing routine sends the data gram to the *default router*.

Default routing is especially useful when a site has a small set of local addresses and only one connection to the rest of the internet.

**Host-Specific Routes**

Most IP routing software allows per-host routes to be specified as a special case. Having per-host routes gives the local network administrator more control over network use, permits testing, and can also be used to control access for security purposes. when debugging network connections or routing tables, the ability to specify a special route to one individual machine turns out to be especially useful.

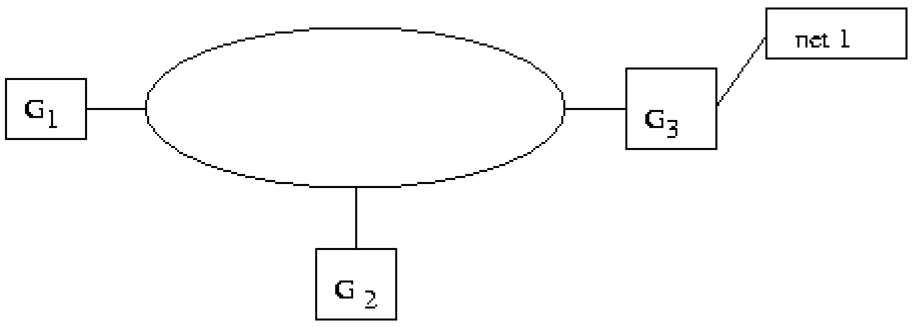
**Internet with Two Backbones**



As long as there was just one single router connecting ARPANET with NSFNET there was no problem. The core gateways of ARPANET had information about all destinations and the routers inside NSFNET contained information about local destinations and used a default route to send all non-NSFNET traffic to between NSFNET and ARPANET as both of them used different matrices to measure costs. the core gateways through the router between ARPANET and NSFNET. However as multiple connections were made between the two backbones, problems arise. Which route should a packet from net1 to net2 take? **Should it be R1 or R2 or R3 or R4 or R5?** For this some exchange of routing information between the two backbones was necessary. But, this was again a problem as how should we compare information.

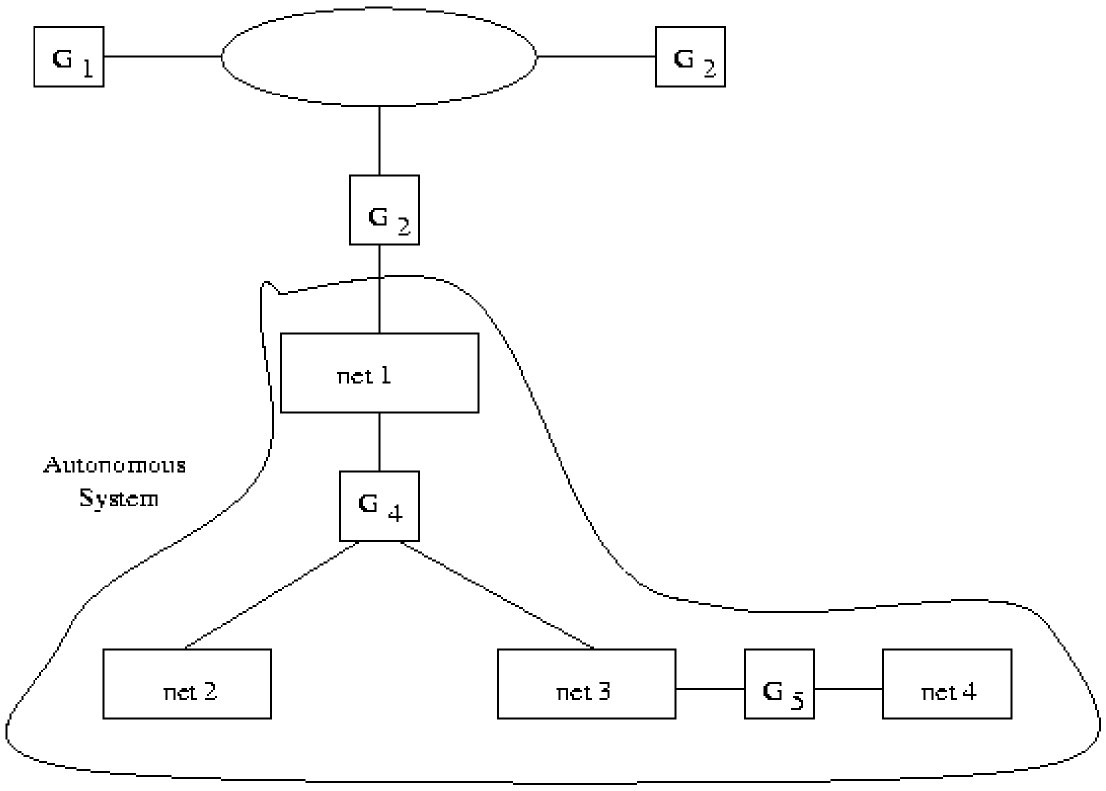
**Gateway-To-Gateway Protocol (GGP)**

This was the protocol used by the core-routers to exchange routing information among themselves. This is based on ***Distance Vector Algorithm*** and uses number of hops as the distance metric. This is a very poor metric as this does not take into account the load on the links and whether a link is slow or fast. A provision is made to manually increment the hop count in case a link is particularly slow.A protocol based on Shortest Path First Algorithm , known as **SPREAD** ,was also used for the same purpose.



**Added Complexity To The Architecture Model**

As the number of networks and routers increased, to reduce the load on the core gateways because of the enormous amount of calculations, routing was done with some core gateways keeping complete information and the non-core gateways keeping partial information.



In thisarchitecture, $G_1$ ,$G_2$ ,$G_3$ are all core gateways and $G_4$ and $G_5$ are non-core gateways. We must have a mechanism for someone to tell $G_2$ that it is connected to net2 , net3 and net4 , besides net1. Only $G_5$ can tell this to $G_2$ and so we must provide for a mechanism for $G_2$ to talk to $G_5$ . A concept of one backbone with core gateways connected to *Autonomous Systems* was developed.
An *Autonomous system* is a group of networks controlled by a single administrative authority. Routers within an autonomous system are free to choose their own mechanisms for discovering , propagating ,validating , and checking the consistency of routes. Each autonomous system must agree to advertise network reachability information to other autonomous systems. Each advertisement propagates through a core router. The assumption made is that most of the routers in

the autonomous system have complete information about the autonomous system. One such router will be assigned the task of talking to the core gateway.

**Interior Gateway Protocols (IGP)**

IGP is a type of protocols used by the routers in an autonomous system to exchange network reachability and routing information. Some of IGPs are given below.

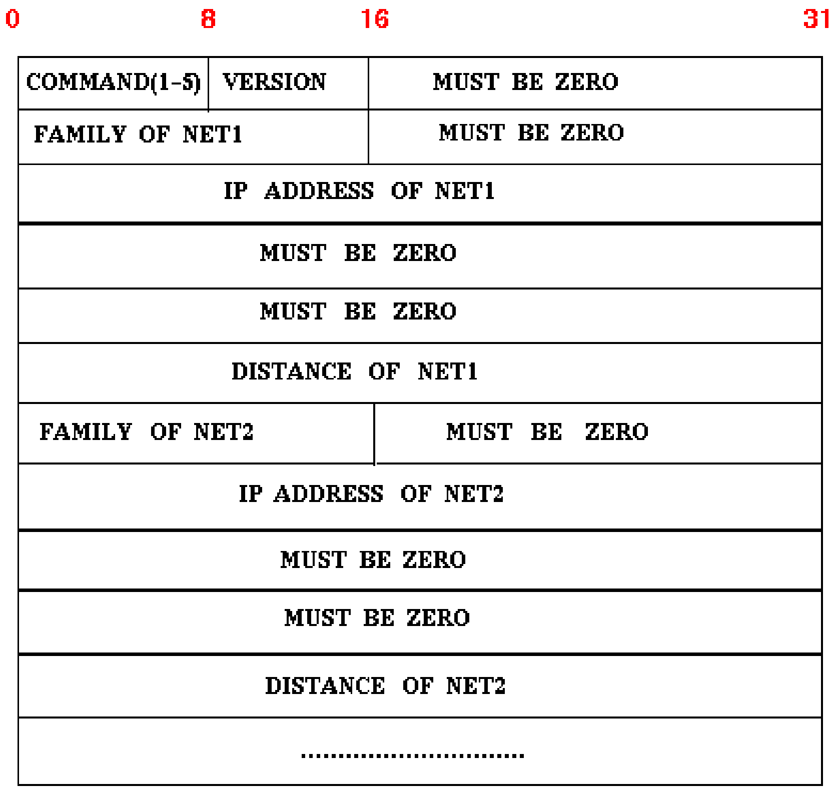
**Routing Information Protocol (RIP)**

This is one of the most widely used IGP. It was developed at Berkeley. This is also known by the name of the program that  implements it, routed .This implements Distance Vector algorithm.Features of RIP:

- RIP uses a hop count metric to measure the distance to a destination. To compensate for differences in technologies, many RIP implementations allow managers to configure artificially high hop counts when advertising connections to slow networks. All routinfg updates are broadcast. This allows all hosts on the network to know about the routes.
- To prevent routes from oscillating between two or more equal cost paths, RIP specifies that existing routes should be retained until a new route has strictly lower cost. Since RIP does not explicitly detect routing loops, RIP must either assume participants can be trusted (being part of one autonomous system) or take precautions to prevent such loops.
- To prevent instabilities, RIP must use a low value for the maximum possible distance.RIP uses 16 as the maximum hop count. This restricts the maximum network diameter of the system to 16.
- To solve the slow convergence problem arising due to slow propagation of routing information, RIP uses Hold Down. If a particular link is down , any new information about that link is not accepted till some time. This is because the router must wait till the information aboutthe link being down propagates to another router before accepting information from that router about that down link.
- RIP runs on top of TCP/IP. RIP allows addresses to be of a maximum size of 14 Bytes. The Distance varies from 1 to 16 (where 16 is used to signify infinity). RIP address 0.0.0.0 denotes a default route. There is no explicit size of the RIP message and any number of routes can be advertized.

The message format is as shown:

## FORMAT OF A RIP MESSAGE



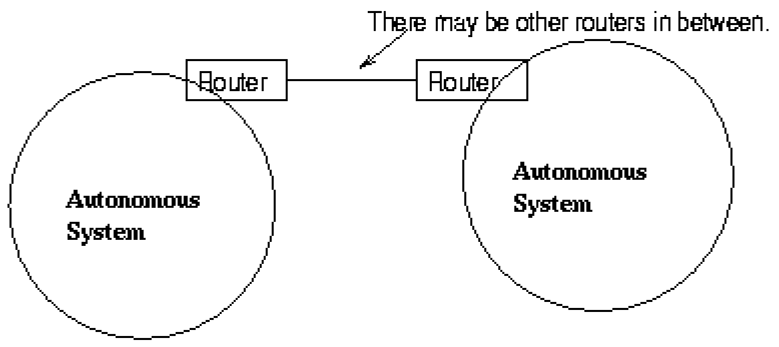
| 0 | 8 | 16 | 31 |
|---|---|---|---|
| COMMAND(1-5) | VERSION | MUST BE ZERO | |
| FAMILY OF NET1 | | MUST BE ZERO | |
| IP ADDRESS OF NET1 | | | |
| MUST BE ZERO | | | |
| MUST BE ZERO | | | |
| DISTANCE OF NET1 | | | |
| FAMILY OF NET2 | | MUST BE ZERO | |
| IP ADDRESS OF NET2 | | | |
| MUST BE ZERO | | | |
| MUST BE ZERO | | | |
| DISTANCE OF NET2 | | | |
| ............................... | | | |

**OSPF(Open Shortest Path First )**

This is an Interior Gateway Protocol designed by the Internet Engineering Task Force ( IETF ). This algorithm scales better than the vector distance algorithms. This Protocol tackles several goals:

- OSPF includes type of service(ToS) routing. So, you can installmultiple routers to a given destination, one for each type of service. When routing a datagram, a router running OSPF uses both the destination address and type of service fields in the IP Header to choose a route.
- OSPF provides load balancing. If there are multiple routes to a given destination at the same cost, OSPF distributes traffic over all the routes equally.
- OSPF allows for creation of AREA HIERARCHIES. This makes the growth of the network easier and makes the network at a site easier to manage. Each area is self contained, so, multiple groups within a site can cooperate in the use of OSPF for routing.
- OSPF protocol specifies that all exchanges between the routers be authenticated. OSPF allows variety of authentication schemes, and even allows one area to choose a different scheme from the other areas.
- To accomodate multi-access networks like ethernet, OSPF allows every multi-access network to have a designated router( designated gateway).
- To permit maximum flexibility, OSPF allows the description of a virtual network topology that abstracts away from details of physical connections.

- OSPF also allows for routers to exchange routing information learned from other sites. The message format distinguishes between information acquired from external sources and information acquired from routers interior to the site, so there is no ambiguity about the source or reliability of routes.
- It hastoo much overhead of sending LSPs but is gradually becoming popular.



Two routers may be configured to think they are neighbours

**Exterior Gateway Protocol (EGP)**

If two routers belonging to two different autonomous systems exchange routing information ,the protocol used is called EGP . EGP consists of:

- **Acquisition Request:** A router sends a request to another neighbour router saying 'I want to talk'.
- **Acquisition Confirm:** This is a positive reply to the Acquisition request.
- **Acquisition Refuse:** This is a negative response to the Acquisition request.
- **Cease Request:** This requests termination of neighbour relationship.
- **Cease Confirm:** This is a confirmation response to the Cease Request.
- **Hello :** This is used to find if the neighbour router is up or down.This requests router to respond if alive.
- **I Heard You:** This is a response to the Hello message confirming that the router is alive. Because it is possible for Hello or I Heard You messages to be lost in transit, EGP uses a k-out-of-n rule to determine whether a network is down.At least k of the last n messages must fail for the router to declare its neighbour down.
- **Poll Request:** This is a request for network routing update.
- **Routing Update:** This conveys routing information about reachable networks to its EGP neighbour. The routing information is the distance vector of the reachable networks.
- **Error:** This is a response to an incorrect message.

EGP is used only to find network reachability and not for differentiating between good and bad routes. We can only use distance metric to declare a route plausible and not for comparing it with some other route (unless the two route form part of a same autonomous system). Since there cannot be two different routes to the same network, EGP restricts the topology of any internet to a tree structure in which a core system forms the root. There are no loops among other autonomous systems connected to it. This leads to several problems:

- Univerasal connectivity fails if the core gateway system fails.
- EGP can advertise only one path to a given network.
- EGP does not support load sharing on routers between arbitrary autonomous systems.
- Multiple backbone networks with multiple connections between them cannot be handled by EGP.

**Border Gateway Protocol(BGP)**

BGP is a distance-vector protocol used to communicate between different ASes. Instead of maintaining just the cost to each destination,each BGP router keeps track of the exact path used.Similarly,instead of periodically giving each neighbour its estimated cost to each destination, each BGP router tells its neighbours the path it is using.Every BGP router contains a module that examines routes to a given destination and scores them returning a number for destination to each route. Any route violating a policy constraint automatically gets a score of infinity. The router adapts a route with shortest distance.The scoring function is not a part of the BGP protocol and can be any function that the system managers want.BGP easily solves the count to infinity problem that plagues other distance-vector algorithms as whole path is known.

---

back to top
Prev| Next | Index

**Computer Networks (CS425)**

**Instructor: Dr. Dheeraj Sanghi**

Prev | Next | Index

---

**Remote Procedure Call**

**Data Conversion**

There is a possibility of having computers of different architectures in the same network. For e.g. we may have DEC or Intel machines ( which use Little-endian representation) connected with IBM or Motorola PCs (which use Big-endian representation) . Now a message from an Intel machine, sent in Little-endian order, may be interpreted by an IBM machine in Big-endian format. This will obviously give erroneous results. So we must have some strategy to convert data from one machine's native format to the other one's or, to some standard network format.

**Available Methods :**

- ***Abstract Syntax Notation (ASN.1) :*** This is the notation developed by the ISO, to describe the data structures used in communication, in a flexible yet standard enough way. The basic idea is to define all the data structure types ( i.e., data types) needed by each application in ASN.1

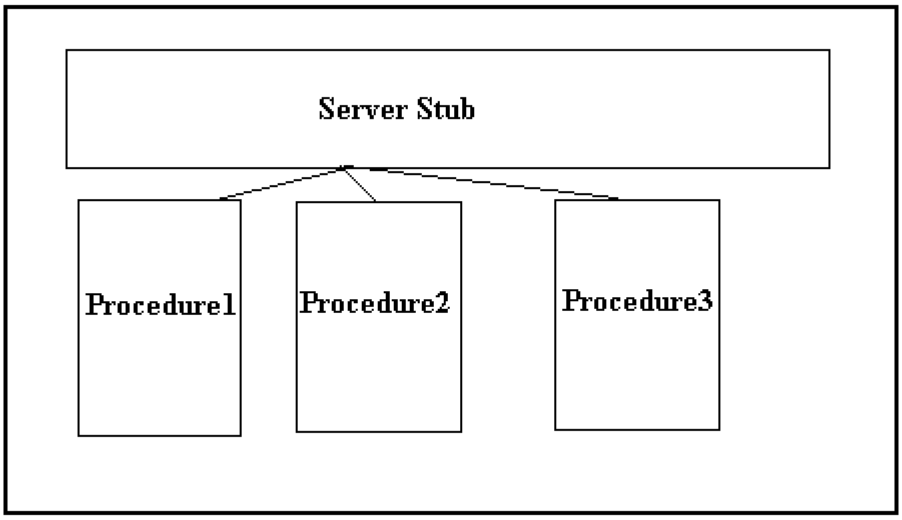
and package them together in a **module**. When an application wants to transmit a data structure, it can pass the data structure to the presentation layer, along with the ASN.1 name of the data structure. Using the ASN.1 definition as a guide, the presentation layer then knows what the types and sizes of the fields are, and thus know how to encode them for transmission ( *Explicit Typing*). It can be implemented with Asymmetric or Symmetric data conversion.

- ***External Data Representation (XDR) :*** This is implemented using Symmetric data representation. XDR is the standard representation used for data traversing the network. XDR provides representations for most of the structures that a C-program can specify. However the encodings contain only data items and not information about their types. Thus, client and servers using XDR must agree on the exact format of messages that they will exchange ( *Implicit Typing* ).
  The chief advantage lies in flexibility: neither the server nor the client need to understand the architecture of the other. But, computational overhead is the main disadvantage. Nevertheless, it simplifies programming, reduces errors, increases interoperability among programs and makes easier network management and debugging.
  *The Buffer Paradigm :* XDR uses a buffer paradigm which requires a program to allocate a buffer large enough to hold the external representation of a message and to add items (i.e., fields) one at a time. Thus a complete message is created in XDR format.

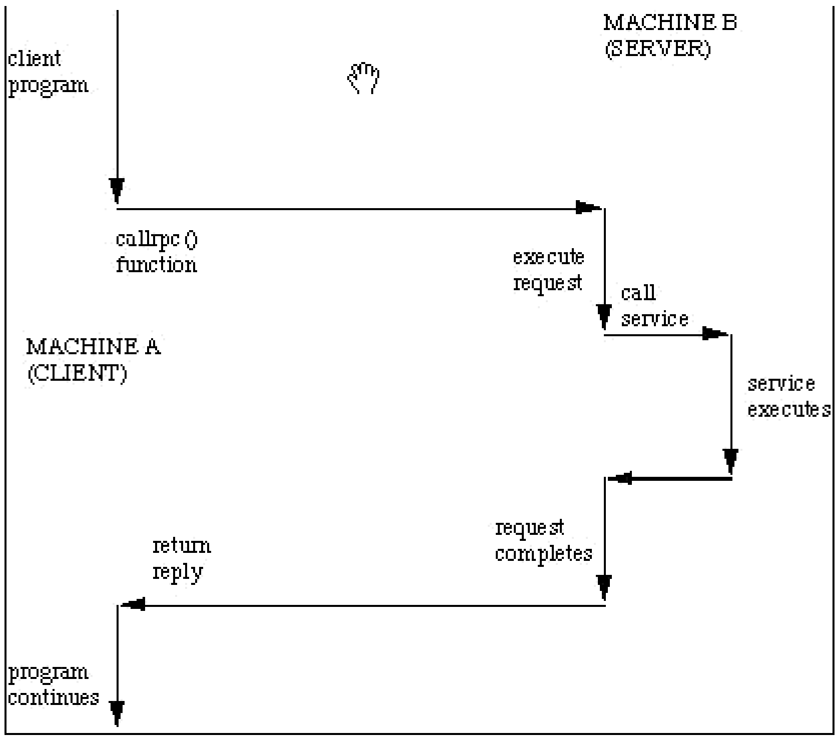
---

**Remote Procedure Call (RPC)**

RPC comes under the Application-Oriented Design, where the client-server communication is in the form of Procedure Calls. We call the machine making the procedure call as *client* and the machine executing the called procedure as *server*. For every procedure being called there must exist a piece of code which knows which machine to contact for that procedure. Such a piece of code is called a *Stub*. On the client side, for every procedure being called we need a unique stub. However, the stub on the server side can be more general; only one stub can be used to handle more than one procedures (see figure ). Also, two calls to the same procedure can be made using the same stub.



Now let us see how a typical remote procedure call gets executed :-

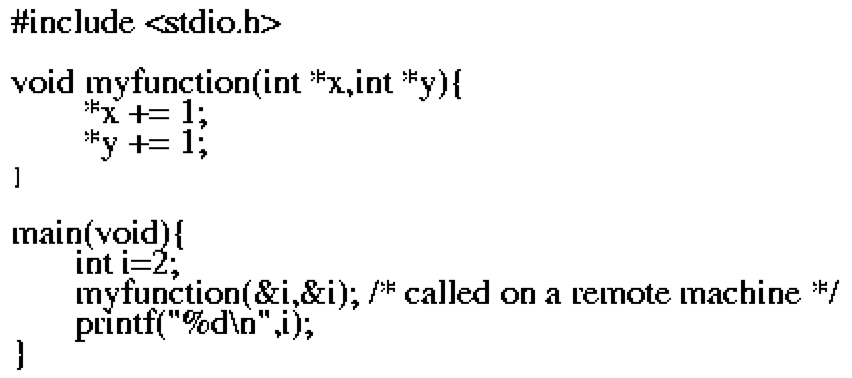


1. Client program calls the stub procedure linked within its own address space. It is a normal local call.
2. The client stub then collects the parameters and packs them into a message (*Parameter Marshalling*). The message is then given to the transport layer for transmission.

3. The transport entity just attaches a header to the message and puts it out on the network without further ado.
4. When the message arrives at the server the transport entity there passes it tot the server stub, which unmarshalls the parameters.
5. The server stub then calls the server procedure, passing the parameters in the standard way.
6. After it has completed its work, the server procedure returns, the same way as any other procedure returns when it is finished. A result may also be returned.
7. The server stub then marshalls the result into a message and hands it off at the transport interface.
8. The reply gets back to the client machine.
9. The transport entity hands the result to the client stub.
10. Finally, the client stub returns to its caller, the client procedure, along-with the value returned by the server in step 6.

This whole mechanism is used to give the client procedure the illusion that it is making a direct call to a distant server procedure. To the extent the illusion exceeds, the mechanism is said to be **transparent**. But the transparency fails in *parameter passing*. Passing any data ( or data structure) by value is OK, but passing parameter 'by reference' causes problems. This is because the pointer in question here, points to an address in the address space of the client process, and this address space is not shared by the server process. So the server will try to search the address pointed to by this passed pointer, in its own address space. This address may not have the value same as that on the client side, or it may not lie in the server process' address space, or such an address may not even exist in the server address space.

One solution to this can be **Copy-in Copy-out**. What we pass is the value of the pointer, instead of the pointer itself. A local pointer, pointing to this value is created on the server side (*Copy-in*). When the server procedure returns, the modified 'value' is returned, and is copied back to the address from where it was taken (*Copy-out*). But this is disadvantageous when the pointer involved point to huge data structures. Also this approach is not foolproof. Consider the following example ( C-code) :



```
#include <stdio.h>

void myfunction(int *x,int *y){
      *x += 1;
      *y += 1;
]

main(void){
      int i=2;
      myfunction(&i,&i); /* called on a remote machine */
      printf("%d\n",i);
}
```

The procedure 'myfunction()' resides on the server machine. If the program executes on a single machine then we must expect the output to be '4'. But when run in the client-server model we get '3'. Why ? Because 'x, and 'y' point to different memory locations with the same value. Each then increments its own copy and the incremented value is returned. Thus '3' is passed back and not '4'.

Many RPC systems finesse the whole problem by prohibiting the use of reference parameters, pointers, function or procedure parameters on remote calls (**Copy-in**). This makes the implementation easier, but breaks down the transparency.

**Protocol :** Another key implementation issue is the protocol to be used - TCP or UDP. If TCP is used then there may be problem in case of network breakdown. No problem occurs if the breakdown happens before client sends its request (client will be notified of this), or after the request is sent and the reply is not received ( time-out will occur). In case the breakdown occurs just after the server has sent the reply, then it won't be able to figure out whether its response has reached the client or not. This could be devastating for bank servers, which need to make sure that their reply has in fact reached to the client ( probably an ATM machine). So UDP is generally preferred over TCP, in making remote procedure calls.

**Idempotent Operations:**

If the server crashes, in the middle of the computation of a procedure on behalf of a client, then what must the client do? Suppose it again sends its request, when the server comes up. So some part of the procedure will be re-computed. It may have instructions whose repeated execution may give different results each time. If the side effect of multiple execution of the procedure is exactly the same as that of one execution, then we call such procedures as **Idempotent Procedures**. In general, such operations are called **Idempotent Operations**.

For e.g. consider ATM banking. If I send a request to withdraw Rs. 200 from my account and some how the request is executed twice, then in the two transactions of 'withdrawing Rs. 200' will be shown, whereas, I will get only Rs. 200. Thus 'withdrawing is a non-idempotent operation. Now consider the case when I send a request to 'check my balance'. No matter how many times is this request executed, there will arise no inconsistency. This is an idempotent operation.

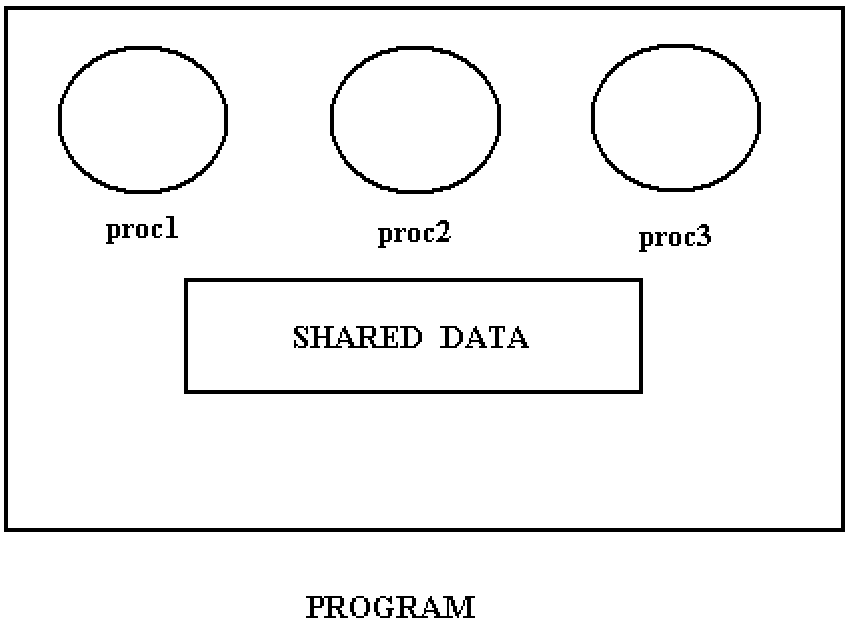
**Semantics of RPC :**

If all operations could be cast into an idempotent form, then time-out and retransmission will work. But unfortunately, some operations are inherently non-idempotent (e.g., transferring money from one bank account to another ). So the exact semantics of RPC systems were categorized as follows:

- *Exactly once :* Here every call is carried out 'exactly once', no more no less. But this goal is unachievable as after a server crash it is impossible to tell that a particular operation was carried out or not.
- *At most once :* when this form is used control always returns to the caller. If everything had gone right, then the operation will have been performed exactly once. But, if a server crash is detected, retransmission is not attempted, and further recovery is left up to the client.
- *At least once :* Here the client stub keeps trying over and over, until it gets a proper reply. When the caller gets control back it knows that the operation has been performed one or more times. This is ideal for idempotent operations, but fails for non-idempotent ones.
- *Last of many :* This a version of 'At least once', where the client stub uses a different transaction identifier in each retransmission. Now the result returned is guaranteed to be the result of the final operation, not the earlier ones. So it will be possible for the client stub to tell which reply belongs to which request and thus filter out all but the last one.

**SUN RPC Model**

The basic idea behind Sun RPC was to implement NFS (Network File System). Sun RPC extends the remote procedure call model by defining a remote execution enviroment. It defines a ***remote program*** at the server side as the basic unit of software that executes on a remote machine. Each remote program consists of one or more remote procedures and global data. The global data is static data and all the procedures inside a remote program share access to its global data. The figure below illustrates the conceptual organization of three remote procedures in a single remote program.



Sun RPC allows both TCP and UDP for communication between remote procedures and programs calling them. It uses the at least once semantic i.e., the remote procedure is executed at least once. It uses copy-in method of parameter passing but does not support copy-out style. It uses XDR for data representation. It does not handle orphans(which are servers whose corresponding clients have died). Thus if a client gives a request to a server for execution of a remote procedure and eventually dies before accepting the results, the server does not know whom to reply. It also uses a tool called ***rpcgen*** to generate stubs automatically.

Let us suppose that a client (say client1) wants to execute procedure P1(in the figure above). Another client (say client2) wants to execute procedure P2(in the figure above). Since both P1 and P2 access common global variables they must be executed in a mutually exclusive manner. Thus in view of this Sun RPC provides mutual exclusion by default i.e. no two procedures in a program can be active at the same time. This introduces some amount of delay in the execution of procedures, but mutual exclusion is a more fundamental and important thing to provide, without it the results may go wrong.

Thus we see that anything which can be a threat to application programmers, is provided by SUN RPC.

**How A Client Invokes A Procedure On Another Host**

The remote procedure is a part of a program executing in a remote host. Thus we would have to properly locate the host, the program in it, and the procedure in the program. Each host can be specified by a unique 32-bit integer. SUN RPC standard specifies that each remote program executing on a computer must be assigned a unique 32-bit integer that the caller uses to identify it. Furthermore, Sun RPC assigns a 32-bit integer identifier for each remote procedure inside a given remote program. The procedures are numbered sequentially: 1, 2, ...., N. To help ensure that program numbers defined by separate organizations do not conflict, Sun RPC has divided the set of program numbers into eight groups.
Thus it seems sufficient that if we are able to locate the host, the program in the host, and the procedure in the program, we would be able to uniquely locate the remote procedure which is to be executed.

**Accommodating Multiple Versions Of A Remote Program**

Suppose somebody wants to change the version of a remote procedure in a remote program. Then as per the identification method described above, he or she would have to make sure that the newer version is compatible with the older one. This is a bottleneck on the server side. Sun RPC provides a solution to this problem. In addition to a program number, Sun RPC includes a 32-bit integer ***version number*** for each remote program. Usually, the first version of a program is assigned version 1. Later versions each receive a unique version number.
Version numbers provide the ability to change the details of a remote procedure call without obtaining a new program number. Now, the newer client and the older client are disjoint, and no compatibility is required between the two. When no request comes for the older version for a pretty long time, it is deleted. Thus, in practice, each RPC message identifies the intended recipient on a given computer by a triple:

(program number, version number, procedure number)

Thus it is possible to migrate from one version of a remote procedure to another gracefully and to test a new version of the server while an old version of the server continues to operate.

**Mapping A Remote Program To A Protocol Port**

At the bottom of every communication in the RPC model there are transport protocols like UDP and TCP. Thus every communication takes place with the help of sockets. Now, how does the client know to which port to connect to the server? This is a real problem when we see that we cannot have a standard that a particular program on a particular host should communicate through a particular port. Because the program number is 32 bit and we can have $2^{32}$ programs whereas both TCP and UDP uses 16 bit port numbers to identify communication endpoints. Thus RPC programs can potentially outnumber protocol ports. Thus it is impossible to map RPC program numbers onto protocol ports directly. More important, because RPC programs cannot all be assigned a unique protocol port, programmers cannot use a scheme that depends on well-known protocol port assignments. Thus, at any given time, a single computer executes only a small number of remote programs. As long as the port assignments are temporary, each RPC program can obtain a protocol port number and use it for communication.

If an RPC program does not use a reserved, well-known protocol port, clients cannot contact it directly. Because, when the server (remote program) begins execution, it asks the operating system to allocate an unused protocol port number. The server uses the newly allocated protocol port for all communication. The system may choose a different protocol port number each time the server begins(i.e., the server may have a different port assigned each time the system boots).

The client (the program that issues the remote procedure call) knows the machine address and RPC program number for the remote program it wishes to contact. However, because the RPC program (server) only obtains a protocol port after it begins execution, the client cannot know which protocol port the server obtained. Thus, the client cannot contact the remote program directly.

**Dynamic Port Mapping**

To solve the port identification problem, a client must be able to map from an RPC program and a machine address to the protocol port that the server obtained on the destination machine when it started. The mapping must be dynamic because it can change if the machine reboots or if the RPC program starts execution again.

To allow clients to contact remote programs, the Sun RPC mechanism includes a dynamic mapping service. The RPC port mapping mechanism uses a server to maintain a small database of port mappings on each machine. This RPC server waits on a particular port number (111) and it receives the requests for all remote procedure calls.

Whenever a remote program (i.e., a server) begins execution, it allocates a local port that it will use for communication. The remote program then contacts the server on its local machine for registration and adds a pair of integers to the database:

(RPC program number, protocol port number)

Once an RPC program has registered itself, callers on other machines can find its protocol port by sending a request to the server. To contact a remote program, a caller must know the address of the machine on which the remote program executes as well as the RPC program number assigned to the program. The caller first contacts the server on the target machine, and sends an RPC program number. The server returns the protocol port number that the specified program is currently using. This server is called the ***RPC port mapper*** or simply the ***port mapper***. A caller can always reach the port mapper because it communicates using the well known protocol port, 111. Once a caller knows the protocol port number the target program is using, it can contact the remote program program directly.

**RPC Programming**

RPC Programming can be thought in multiple levels. At one extreme, the user writing the application program uses the RPC library. He/she need not have to worry about the communication through the network. At the other end there are the low level details about network communication. To execute a remote procedure the client would have to go through a lot of overhead e.g., calling XDR for formatting of data, putting it in output buffer, connecting to port mapper and subsequently connecting to the port through which the remote procedure would communicate etc. The ***RPC library*** contains procedures that provide almost everything required to make a remote procedure call. The library contains procedures for marshaling and unmarshaling of the arguments and the results respectively. Different XDR routines are available to change the format of data to XDR from native, and from XDR to native format. But still a lot of overhead remains to properly call the library routines. To minimize the overhead faced by the application programmer to call a remote procedure a tool named ***rpcgen*** is devised which generates client and server stubs. The stubs are generated automatically, thus they have loose flexibility e.g., the timeout time, the number of retransmissions are fixed. The program specification file is given as input and both the server and client stubs are automatically generated by rpcgen. The specification file should have a .x extension attatched to it. It contains the following information:-

- constant declarations ,
- global data (if any),
- information about all remote procedures ie.
- procedure argument type ,
- return type .

---

**References**

1. http://www.cs.cf.ac.uk/Dave/C/node33.html#fig:rpc

---

back to top
Prev| Next | Index

**Computer Networks (CS425)**

**Instructor: Dr. Dheeraj Sanghi**

Prev | Next | Index

---

**Remote Procedure Call (Contd...)**

We now look at the different ways of writing RPC programs. There are three levels at which RPC programs can be written:

1. On one extreme we can use some standard applications or programs provided by sun-RPC. For example, one can use the library function int rnusers (char *machinename ) for finding number of users logged onto a remote system.
2. On the other hand we can use RPC runtime library : This has the maximum flexibility and efficiency. It has various functions like opening a connection, connecting to a port-mapper and other low level functions. Using this we can write our own stubs. This is however relatively difficult to use.
3. The best approach is to use RPCgen : RPCgen stands for RPC generator. It generates client and server stubs. There are several details that cannot be easily controlled (for example, the number of retries in case of timeout). RPCgen takes as input a specification file which has a list of the procedures and arguments. It creates the client stub and server stub.

**Writing the Configuration File**

If we use *RPCgen*, then our work is essentially reduced to writing a specification file. This file has the procedure names, argument types, return types etc. Here we show a simple RPC specification file ( spec.x ) for printing a message on some other machine :

```
program MESSAGEPROG {

        version MESSAGEVERS {

            int PRINTMESSAGE ( string ) = 1;

        } = 1;

} = 99;
```

We will have to do some changes on the server as well as client side. The server program ( msg_proc.c ) will look like this :

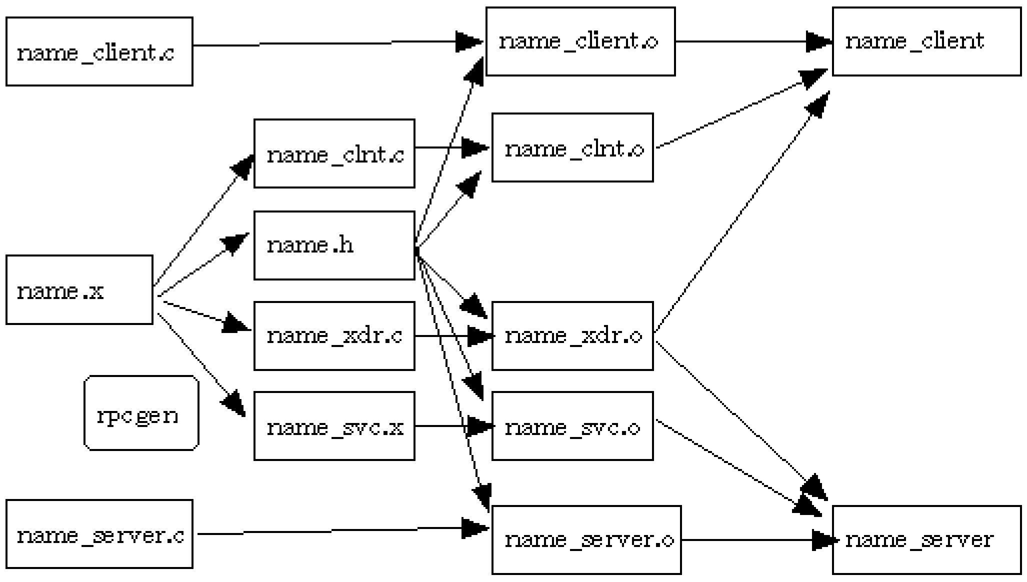
```
#include <stdio.h>

#inculde <rpc/rpc.h>

#include "msg.h"

int *printmessage_1( msg )

char **msg;

{

    . . . . .

    . . . . .

}
```

On the client side the program ( client.c ) will look like

```
#include <stdio.h>

#inculde <rpc/rpc.h>

#include "msg.h"

main( int argc, char *argv[])

{

        client *c1;

        int *result;

        char *server = argv[1];

        char *message = argv[2];

        if (( c1 = clnt_create( server, MESSAGEPROG, MESSAGEVERS, "tcp" )) == NULL

        {
```

```
        // error
    }
    result = printmessage_1( &message, c1);
    . . . . .
}
```

After creating the specification file we give the command $rpcgen spec.x ( where spec.x is the name of the specification file ). The following files actions are taken and the files spec.h, spec_svc.c, spec_clnt.c get created :



Once we have these files we write

$cc msg_proc.c spec_svc.c

$cc client.c spec_clnt.c

1. When we start the server program it creates a socket and binds any local port to it. It then calls *svc_register,* to register the program number and version. This function contacts the port mapper to register itself.

2. When the client program is started it calls *clnt_create*. This call specifies the name of the remote system, the program number, version number, and the protocol. This functions contacts the port mapper and finds the port for the server ( Sun RPC supports both TCP and UDP).

3. The client now calls a remote procedure defined in the client stub. This stub sends the datagram/packet to the server, using the port number obtained in step two. The client waits for a response transmitting the requests a fixed number of times in case of a timeout. This datagram/packet is received by the server stub associated with the server program. The server stub executes the called procedure. When the function returns to the server stub it takes the return value, converts it to the XDR format and transmits it back to the client. The client stub receives the response, converts it as required and returns to the client program

**Authentication**

RPC defines several possible forms of authentication, including a simple authentication scheme that relies on UNIX and a more complex scheme that uses the ***Data Encryption Standard (DES)***.

Authentication protocols can be of the following types:

- **NULL Authentication -** In this case, no authentication is done. Neither the client cares about its identity nor the server cares who the client is. Example is a time server.
- **UNIX Style Authentication -** Unix authentication relies on the client machine to supply its hostname and the userid of the user making the request. The client also specifies its local time as a timestamp which can be used to sequence requests. The client also sends the main numeric group identifier of the group of which the user is a member and also the group identifiers of all the groups of which the user is a member. Based on this information, the server decides whether the client will be given permission to execute the procedure requested. This is a very weak form of security as the user and group identifiers are the same as UID and GID in the client's own machine, and anyone can send these information and see the data. This form of authentication is used in NFS.
- **Data Encryption Standard (DES) -** Here the client gives a password which is sent to the server in encrypted form. Encryption is done based on keys which are known only to the client and the server. This is indeed a powerful method of authentication.
- **SHORT -** This method is used for short form of authentication in messages after the first one. The client is authenticated only once during the initial handshake and a handle is given to the client. In future the client communicates with the server using the handle. It is difficult for another user to break in. This is not an entirely new style of authentication, and it can be used with any form of authentication.

---

**Image References**

- http://www.hlla.is.tsukuba.ac.jp/~yas/sie/pdsoft-2001/2002-01-10/images/rpcgen-files.gif

---

back to top
Prev| Next | Index

**Computer Networks (CS425)**

**Instructor: Dr. Dheeraj Sanghi**

Prev | Next | Index

---

**Distributed Applications**

We can use any of the following two approaches in designing a distributed application.

- **Communication-Oriented Design:** Begin with the communication protocol. Design a message format and syntax. Design the client and server components by specifying how each reacts to in-coming messages and how each generates outgoing meassages.
- **Application-Oriented Design:** Begin with the application. Design a conventional application program to solve the problem. Build and test a working version of the conventional program into two or more pieces, and add communication protocols that allow each piece to execute on a separate computer.

**Semantics of Applications**

- **Normal Application:** A main program which may call procedures defined within the program (proc A in this case). On return from this procedure the program continues. This procedure (proc A) may itself call other procedures (proc B in this case). Refer to the figure below:

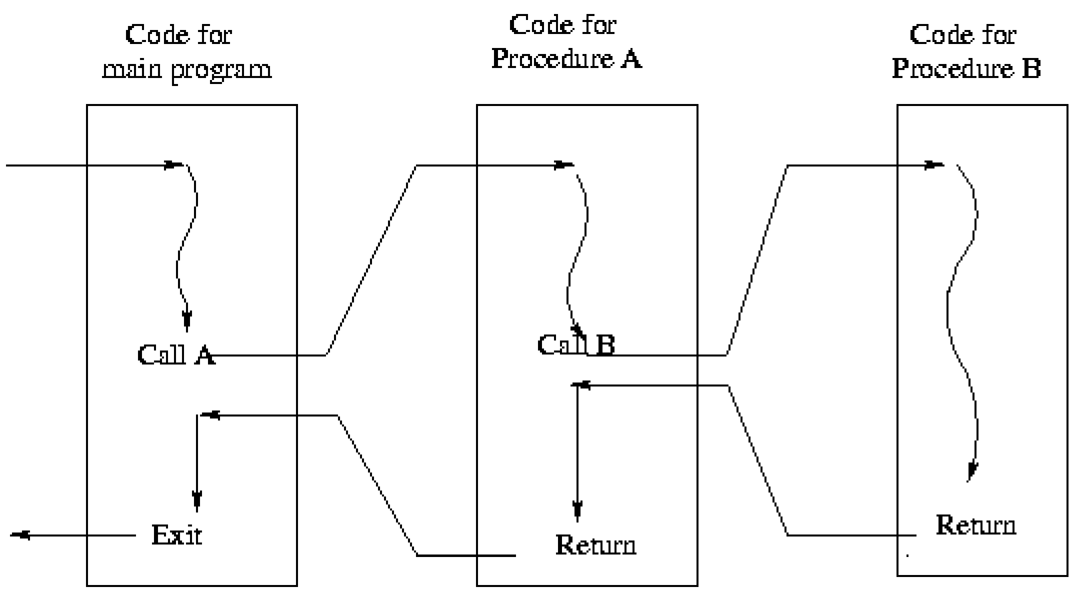


Fig1. Flow of control in a normal program

- **Distributed Application:** A client program executing on a machine 1 may call a procedure (proc A) which is defined and run on another machine ( we say server for machine 1 is machine 2) upon return from the call the program on machine 1 continues. The server program on machine 2 may in turn act as a client and call procedures on another machine3 (now machine 3 is a server for machine 2). Refer to the figure below:

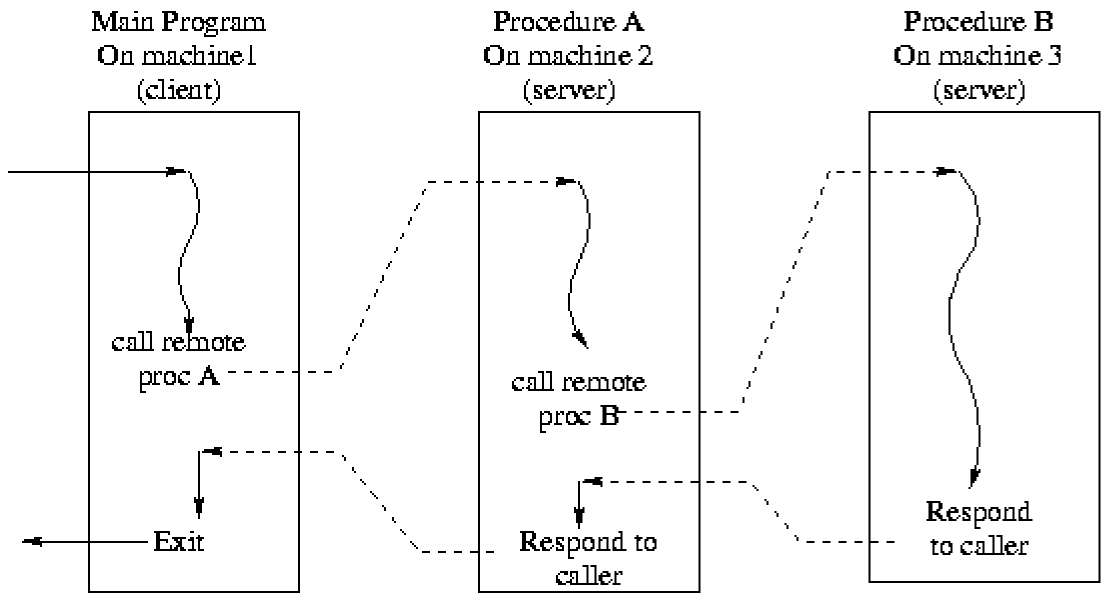


Fig2. Flow of control in a distributed program

**Passing Arguments in Distributed Programs**

**Problem: Incompatibility in argument storage**

For example, some machines may use 7 bit for storing characters while some others might use 8 bit, some machines may use Big-endian representation while others might use Small-endian representation.

**Possible Solutions**

- One solution may be to find out the architecture of receiving end, convert the data to be sent to that architectue and then send the data. However, this will lead to following problems:
    1. It is not easy to find out the architecture of a machine.
    2. If I change the architecture of my machine then this information has to be conveyed to the client.

- Another solution is to have a standard format for networks. This may lead to inefficiency in the case when the two communicating machines have the same architecture beacuse in this case the conversion is unnecessary.

**XDR (External Data Representation)**

XDR was the solution adopted by SUN RPC. RPC was mainly the outcome of the need for distributed filesystems(NFS).

**Buffer Paradigm**

The program allocates a buffer large enough to hold the external representation of a message and adds items one at a time. The library routine invoked to allocate space for the buffer is xdr_mem_create . After allocating space we may append data to this buffer using various conversion library routines like xdr_int (xdr_int coverts an integer to it's external representaion and appends it to the buffer) to convert native objects to external representaion and then append to the buffer. After all the data to be passed has been converted and appended we send the buffer.

**ASN.1**

First add the information related to the the data being sent to the buffer and then append the data to the buffer. For example, to send a character followed by an integer (if the sending machine uses one byte for char and two bytes for integers) we send the information as - one byte char, two byte integer ...

The routines for encoding and decoding are the same, depending on the type of the buffer which may be (specified at the time fo allocating space for the buffer) XDR_ENCODE or XDR_DECODE encoding or decoding are performed respectively.

For the routine xdr_int(xdrs, &i)

- If the allocation was done as xdr_mem_create(xdrs, buf, BUFSIZE, XDR_ENCODE) then the value obtained by converting i to its external representation would be appended to the buffer.
- If the allocation was done as xdr_mem_create(xdrs, buf, BUFSIZE, XDR_DECODE) then an integer will be extracted , decoded , and the value will be stored in the variable i.

There are routines (like xdr_stdin_create) to write/read from sockets and file descriptors.

---

back to top
Prev| Next | Index

~P

Mj⍰ĪB

**Computer Networks (CS425)**

**Instructor: Dr. Dheeraj Sanghi**

---

**Applications**

**FTP**

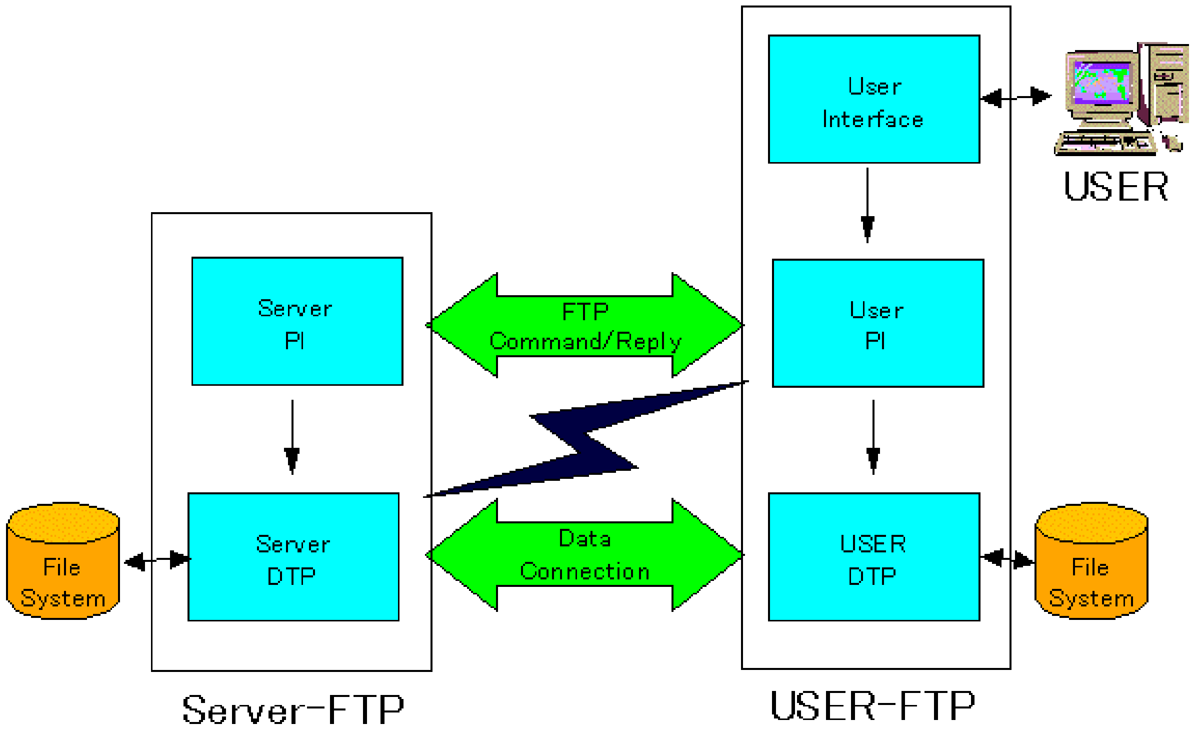
Given a reliable end-to-end trasport protocol like TCP, File Transfer might seem trivial. But, the details authorization, representation among heterogeneous machines make the protocol complex.

FTP offers many facilities :

- Interactive Access : Most implementations provide an interactive interface that allows humans to easily interact with remote servers.
- Format (representation) specification : FTP allows the client to specify the type and format of stored data.
- Authentication Control : FTP requires client to authorize themselves by sending a login name and password to the server before requesting file transfers.

FTP Process Model

FTP allows concurrent accesses by nultiple clients. Clients use TCP to connect to the server. A master server awaits connections and creates a slave process to handleeach connection. Unlike most servers, the slave process does not perform all the necessary computation. Instead the slave accepts and handles the control connection from the client, but uses an additinal process to handle a separate data transfer connection. The control connection carries the command that tells the server which file to transfer.



Data transfer connections and the data transfer processes that use them can be created dynamically when needed, but the control connection persists throughout a session. Once the control connection disappears, the session is terminated and the software at both ends terminates all data transfer processes.

In addition topassing user commands to the server, FTP uses the control connection to allow client and server processes to coordinate their use of dynamically assigned TCP protocol ports and the creation of data transfer processes that use those ports.

**Proxy commands** - allows one to copy files from any machine to any other arbitrary machine ie. the machine the files are being copied to need not be the client but any other machine.
Sometimes some **special processing** can be done which is not part of the protocol. eg. if a request for copying a file is made by issuing command 'get file_A.gz' and the zipped file does not exist but the file file_A does , then the file is automatically zipped and sent.
Consider what happens when the **connection breaks during a FTP session**. Two things may happen, certain FTP servers may again restart from the beginning and whatever portion of the file had been copied is overwritten. Other FTP servers may ask the client how much it has already read and it simply continues from that point.

---

**TFTP**

TFTP stands for Trivial File Transfer Protocol. Many applications do not need the full functionality of FTP nor can they afford the complexity. TFTP  provides an inexpensive mechanism that does not need complex interactions between the client and the server. TFTP restricts operations to simple file transfer and does not provide authentication. Diskless devices have TFTP encoded in read-only memory(ROM) and use it to obtain an initial memory image when the machine is powered on. The advantage of using TFTP is that it allows bootstrapping code to use the same underlying TCP/IP protocols. that the operating system uses once it begins execution. Thus it is possible for a computer to bootstrap from a server on another physical network. TFTP does not have a reliable stream transport service. It runs on top of UDP or any other unreliable packet delivery system using timeout and retransmission to ensure that data arrives. The sending side transmits a file in fixed size blocks and awaits acknowledgements for each block before sending the next.

**Rules for TFTP**

The first packet sent requests file transfer and establishes connection between server and client. Other specifications are file name and whether it is to be transferred to client or to the server. Blocks of the file are numbered starting from 1 and each data packet has a header that specifies the number of blocks it carries and each acknowledgement contains the number of the block being acknowledged. A block of less than 512 bytes signals end of file. There can be five types of TFTP packets . The initial packet must use operation codes 1 or 2 specifying either a read request or a write request and also the filename. Once the read request or write request has been made the server uses the IP address and UDP port number of the client to identify subsequent operations.Thus data or ack msgs do not contain filename. The final message type is used to report errors.
TFTP supports symmetric retransmission. Each side has a timeout and retransmission.If the side sending data times out, then it retransmits the last data block. If the receiving side times out it retransmits the last acknowledgement. This ensures that transfer will not fail after a single packet loss.
Problem caused by symmetric retransmission - **Sorcerer's Apprentice Bug**
When an ack for a data packet is delayed but not lost then the sender retransmits the same data packet which the receiver acknowledges. Thus both the acks eventually arrives at the sender and the sender now transmits the next data packet once corresponding to each ack. Therefore a retransmission of all the subsequent packets are triggered . Basically the receiver will acknowledge both copies of this packet and send two acks which causes the sender in turn to send two copies of the next  packet.. The cycle continues with each packet being transmitted twice.
TFTP supports multiple file types just like FTP ie. binary and ascii data. TFTP may also be integrated with email . When the file type is of type mail then the FILENAME field is to be considered as the name of the mailbox and instead of writing the mail to a new file it should be appended to it. However this implementation is not commonly used .

---

Now we look at another very common application EMAIL

**EMAIL (electronic mail - SMTP , MIME , ESMTP )**

Email is the most widely used application service which is used by computer users. It differs from other uses of the networks as network protocols send packets directly to destinations using timeout and retransmission for individual segments if no ack returns. However in the case of email the system  must  provide for instances when the remote machine or the network connection has failed and take some special action.Email applications involve two aspects -

- User-agent( pine, elm etc.)
- Transfer agent( sendmail daemon etc.)

When an email is sent it is the mail transfer agent (MTA) of the source that contacts the MTA of the destination. The protocol used by the MTA 's on the source and destination side is called SMTP. SMTP stands for **Simple Mail Transfer Protocol.**. There are some protocols that come between the user agent and the MTA eg. POP,IMAP which are discussed later.

**Mail Gateways -**

Mail gateways are also called mail relays, mail bridges and in such systems the senders machine does not contact the receiver's machine directly  but sends mail across one or more intermediate machines that forward it on. These **intermediate machines** are called mail gateways.Mail gateways are introduce unreliablity.Once the sender sends to first intermediate m/c then it discards its local copy. So failure at an intermediate machine may result in message loss without informing the sender or the receiver. Mail gateways also introduce delays. Neither the sender nor the receiver can determine  how long the delay will last or where it has been delayed.
However mail gateways have an advantage  providing interoperability ie. they provide connections among standard TCP/IP mail systems and other mail systems as well as between TCP/IP internets and networks that do not support Internet protocols. So when there is a change in protocol then the mail gateway helps in translating the mail message from one protocol to another since it will be designed to understand both. .

**SIMPLE MAIL TRANSFER PROTOCOL(SMTP)**

TCP/IP protocol suite specifies a standard for the exchange of mail between machines. It was derived from the (MTP ) Mail Transfer Protocol. it deals with how the nderlying mail delivery system passes messages across a link from one.machine to another. The mail is enclosed in what is called an **envelope** . The enveilope contains the To and From fields and these are followed by the mail . The mail consists of two parts namely the Header and the Data.
The Header has the To and From fields. If Headers are defined by us they should start with X. The standard headers do not start with X.
In SMTP data portion can contain only printable ASCII characters The old method of sending a binary file was to send it in uuencoded form but there was no way to distinguish between the many types of binary files possible eg. .tar , .gz , .dvi etc.

**MIME(Multipurpose Internet Mail Extension)**

This alllows the transmission of Non ASCII data througfh email, MIME allows arbitrary data to be encoded in ASCII and sent in a standard email message. Each MIME message includes information that tells the recipient the type of data and the type of encoding used.and this information alongwith the MIME version resides in the MIME header. Typical MIME header looks like -

***MIME-Version: 1.0***
***Content-Description:***
***Content-Id:***
***Content-Type: image/gif***
***Content-Transfer-Encoding: base64***

Content Descirption : contains the file name of the file that is being sent. Content -Type : is an important field that specifies the data format ie. tells what kind of data is being sent. It contains two identifiers a content type and a subtype separated by a slash. for e.g. image/gif
There are 7 Content Types -

1. text

2. image
3. video
4. audio
5. application
6. multipart
7. message

Content type - **Message**
It supports 3 subtypes namely

1. RFC822 - the old mail message format
2. Partial- means that ordinary message is just a part and the receiver should wait for all the parts before putting it in the mailbox.
3. external_body - destination MTA will fetch file from remote site.

Content Type - **Multipart**
Multiple messages which may have different content types can be sent together. It supports 4 subtypes namely

1. mixed -Look at each part independently
2. alternative - The same message is sent in multiple types and formats and the receiver may choose to read the message in any form he wishes.
3. parallel -The different parts of the message have to be read in parallel. ie.audio , video and text need to be read in a synchronised fashion
4. digest -There are multiple RFC messages in mail. The addresses of the receivers are in the form of a mailing list. Although file header is long it prevents cluttering of mail box.

**PROBLEMS WITH SMTP**

1. There is no convenient  way to send nonprintable characters
2. There is no way to know if one has received mail or not or has read it or not.
3. Someone else can send a mail on my behalf.

So a better protocol was proposed - **ESMTP** ESMTP stands for Extended Simple Mail Transfer Protocol. It is compatible with SMTP. Just as the first packet sent in SMTP is HELO similarly in ESMTP the first packet is called EHELO. If the receiver supports ESMTP then it will answer to this EHELO packet by sending what data type and what kind of encoding it supports. Even a SMTP based receiver can reply to it. Also if there is an error message or there is no answer then the sender uses SMTP.

**DELIVERY PROTOCOLS**

The delivery protocols determine how the mail is transferred by  the mail transfer agent    to the user agent which provides an interface for reading mails.
**There are 3 kinds**
**1. POP3 (Post Office Protocol)** Here the mail person accesses the mail box  from

say a PC and the mail gets accumulated on a server. So in POP3 the mail is downloaded to the PC at a time interval which can be specified by the user. POP3 is used when the mail is always read from the same machine, so it helps to download the mail to it in advance.

**2.IMAP(Intermediate Mail Access Protocol)** Here the user may access the mail box on the server from different machines so there is no point in downloading the mail before hand. Instead when the mail has to be read one has to log on to the server. (IMAP thus provides **authentication**) The mailbox on the server can be looked upon as a **relational database.**

**3.DMSP(Distributive Mail System Protocol)** There are multiple mailboxes on different servers. To read the mail I connect to them from time to time and whenever I do so the mail will be downloaded. When a reply is sent then it will put the message in a queue. Thus DMSP is like a **pseudo MTA.**

**Ensuring Network Security**

1. How to ensure that nobody else reads your mail?
2. How to be sure that the mail has not been seen by someone else in your name?
3. Integrity ie. mail has not been tampered with
4. Non-Repudiability- means once I send a mail I cannot deny it, and this fact can be proved to a third person
5. Authentication

**Mechanisms (PGP & PEM)**

PGP (Pretty Good Privacy) - It uses some crytography algorithm to crypt the messages.
**Symmetric PGP**- The key used for encryption and decryption is the same.
**Asymmetric PGP -** The key used for encryption and decryption is different.Keys come in pairs - public (known to all) and private. which everybody has. Usually encryption is done using public key so that the private key is used for decryption by the receiver only for whom the message is meant.
Eg. of Symmetric PGP is DES, IDEA
Eg. of Asymmetric PGP is RSA
Symmetric is usually faster In asymmetric PGP there is a problem of key distribution. A hash function is applied on every message so that no two messages hash to the same value. Now the hash function is encrypted . If the hash function of source and destination matches then No tampering. If the key for encryption is private then not everybody can generate the message although anyone can read it . So this scheme **lacks privacy** tackles the other security issues.

---

**References**

1. http://hp.vector.co.jp/authors/VA019876/sokrpg/doc/img/ftpmodel.gif

---

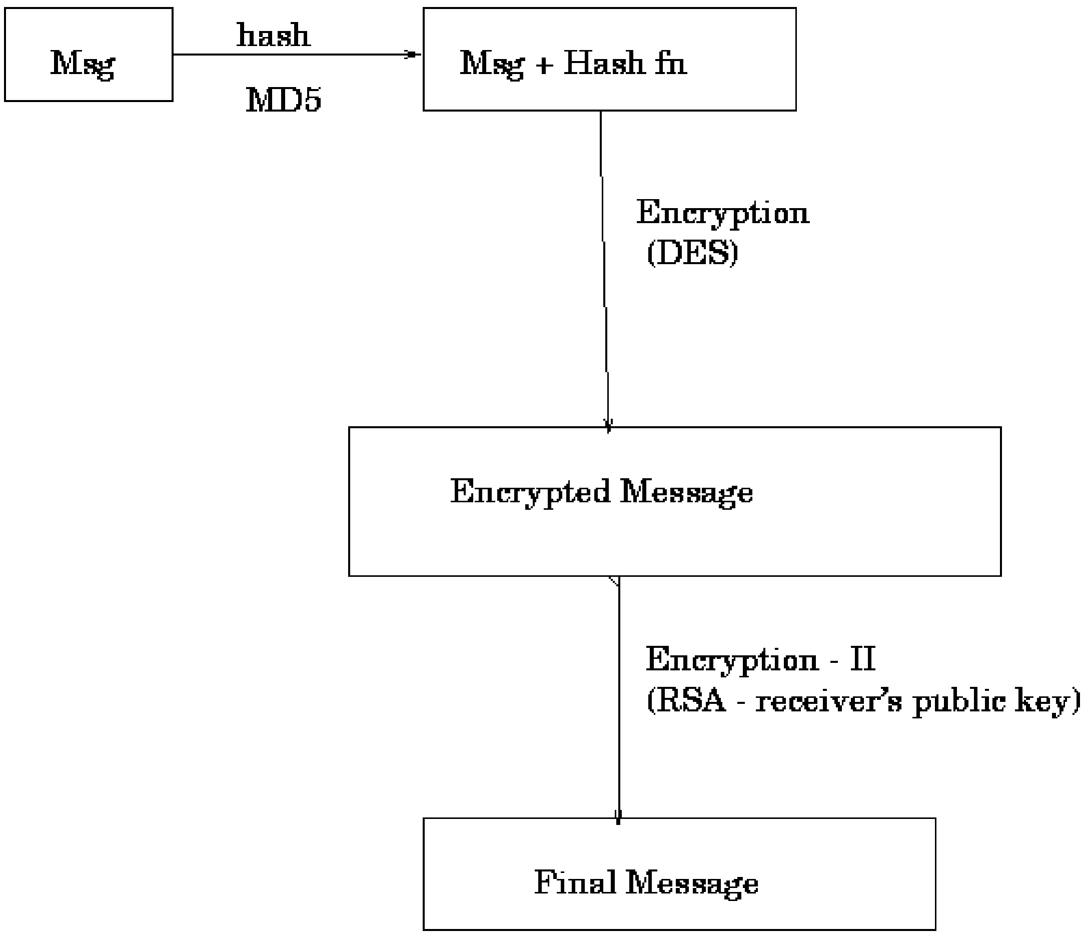
back to top
Prev | Next | Index

---

## PEM & SNMP

### PEM(Privacy Enhanced Mail)

This is a IETF standard , a result of a group working for a long time. The basic idea is have privacy by virtue of hierarchial authentication.  A receiver trusts the message of the sender when it i accompanied by a certificate from his trusted authority. These authoratative certificates are distributed from a group called Internet Policy Registration Authority (IPRA) and  Policy Certificate Authority (PCA). These trusted authority actually certifies the public key sent by senders. The mode of operation is as follows :



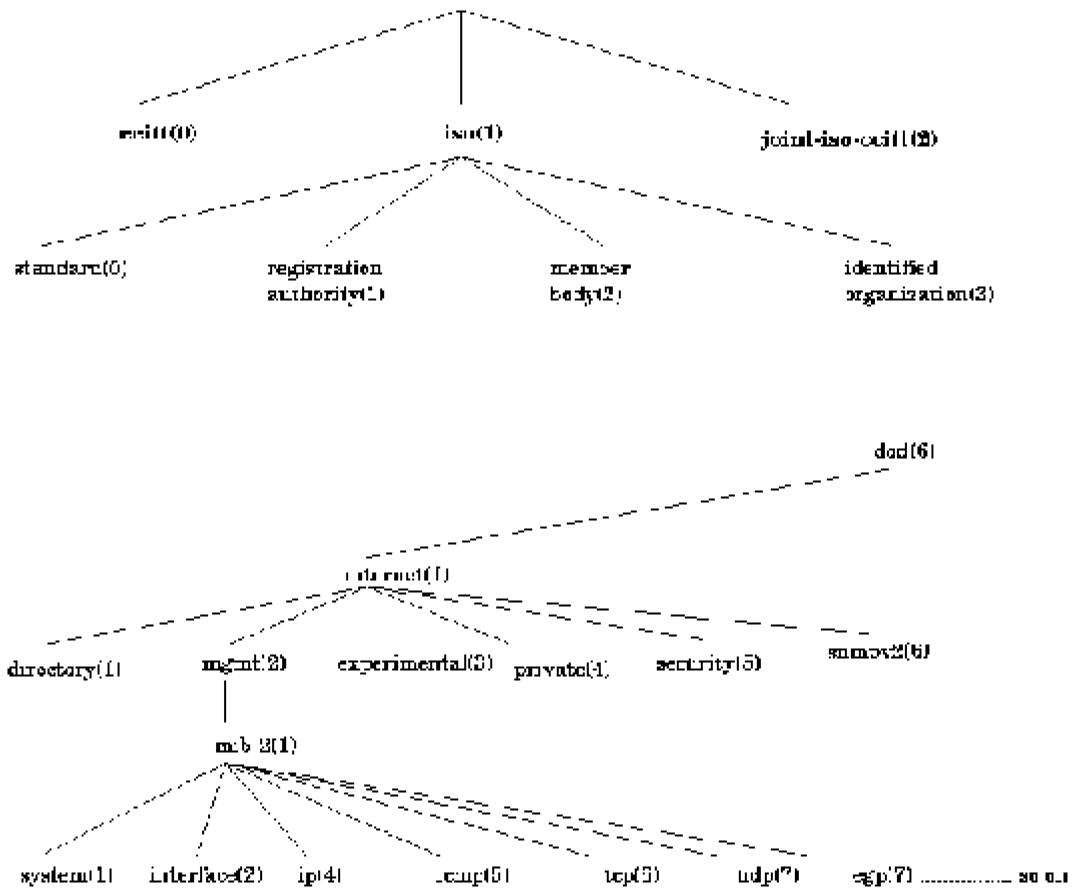
One difference with PGP is that it doesn't support compression.

**SNMP(Simple Network Management Protocol)**

A large network can often get into various kinds of trouble due to routers (dropping too many packets), hosts( going down) etc. One has to keep track of all these occurence and adapt to such situations. A protocol has been defined . Under this scheme all entities in the network belong to 4 class :

1. Managed Nodes
2. Management Stations
3. Management Information (called Object)
4. A management protocol

The managed nodes can be hosts,routers,bridges,printers or any other device capable of communicating status information to others. To be managed directly by SNMP , a node must be capable of running am SNMP management process, called SNMP agent. Network management is done by management stations by exchanging information with the nodes. These are basically general purpose computers running special management software.
The management stations polls the stations periodically . Since SNMP uses unreliable service of UDP the polling is essential to keep in touch with the nodes. Often the nodes sends a trap message indiacting that it is going to go down. The management stations then periodically checks (with an increased frequaency) . This type of polling is called trap directed polling. Often a group of nodes are represented by a single node which communicate with the managemenet stations. This type of node is called proxy agent. The proxy agent can also server as a security arrangement.
All the variables in these scheme are called Objects. Each variable can be referenced by a specific addressing scheme adopted by this system. The entire collection of all objects is called Management Information Base (MIB). The adrressing is hierarchial as seen in the picture.



**Part Of the ASN.1 object naming tree**

Internet is adressed as 1.3.61. All the objects under this domain has this string at the beginning. The informations are exchanged in a standard and vendor-neutral way . All the data are represented in Abstract Syntax Notation 1 (ASN.1). It is similar to XDR as in RPC but it have widely different representation scheme. A part of it actually adopted in SNMP and modified to form Structure Of Information Base. The Protocol specifies various kinds of messages that can be exchanged between the managed nodes and the management station.

| **Message** | **Description** |
|---|---|
| 1. Get_Request | Request the value for a variable |
| 2. Get_Response | Returns the value of the variable asked for |
| 3. Get_Next_Request | Request a variable next to the previous one |
| 4. Set_Request | Set the value of an Object. |
| 5. Trap | Agent to manager Trap report |
| 6. Get_bulk_request | Request a set of variable of same type |

7. Inform_Request Exchange of MIB among Management stations

The last two options has been actually added in the SNMPv2. The fourth option need some kind of authentication from the management station.

**Addressing Example :**

Following is an Example of the kind of address one can refer to when fetching a value in the table :-

```
(20) IP-Addr-Table = Sequence of IPAddr-Entry (1)
          IPAddrEntry = SEQUENCE {
                              IPADDENTRYADDR    : IPADDR (1)
                              Index                      :  integer (2)
                              Netmask                  :  IPAddr (3)            }
```

So when accessing the netmask of some IP-entity the variable name wld be :

*1.3.6.1.2.4.20 .1.3.key-value*

Here since Ip-address the unique key to index any member of the array the address can be like :-

*1.3.6.1.2.4.20.1.3.128.10.2.3*

---

back to top
Prev| Next | Index

**Computer Networks (CS425)**

**Instructor: Dr. Dheeraj Sanghi**

Prev | Next | Index

---

**Firewalls**

**Introduction**

This lecture discusses about security mechanisms in the Internet namely Firewall . In brief, It's a configuration of routers and networks placed between an organization's internal internet and a connection to an external internet to provide security. In other words, Firewall is a mechanism to provide limited access to machines either from the outside world to internal internet or from internal world to outside world. By, providing these security mechanisms, we are increasing the processing time before one can access a machine. So, there is a trade-off between security and ease of use. A firewall partitions an internet into two regions, referred to informally as the inside and outside.

```
                                         __
                                        |  | __________ Firewall
       _________________________          |  |         _______________________
       |                       |        |  |       |                     |
       |                          |            |  |        |                      |
       |    Rest of Internet      |________    |  |_____   |    Intranet          |
       |                          |            |  |        |                      |
       |__________________________ |           |  |        |______________________|
                                         |_|
          Outside                                          Inside
```

Security Lapses

- Vulnerable Services - NFS : A user should not be allowed to export certain files to the outside world and from the outside world also, someone should not be allowed to export our files.
- Routing based attacks : Some kind of ICMP message should not be allowed to enter my network. e.g.. Source routing and change route ICMP's.
- Controlled access to our systems : e.g.. Mail server and web pages should be accessible from outside but our individual PC's should not be accessible from the outside world.
- Authentication : Encryption can be used between hosts on different networks.
- Enhanced Privacy : Some applications should be blocked. e.g.. finger ...
- PING & SYN attack : Since these messages are send very frequently, therefore you won't be able to do anything except reply to these messages. So, I should not allow these messages to enter my network.

So. whatever I provide for my security is called Firewall. It is a mechanism and not just a hardware or software.

**Firewall Mechanisms**

1. Network Policy : Here, we take into consideration, what services are allowed for outside and inside users and the services which are allowed can have additional restrictions. e.g.. I might be allowed to download things from the net but not upload i.e.. some outside users cannot download the things from our net. Some exceptional cases might be there which have to be handled separately. And if some new application comes up then , we choose an appropriate network policy.

2. Authentication mechanism : An application can be designed which ask for a password for authentication.

3. Packet Filtering : Router have information about some particular packets which should not be allowed.

4. Application gateways : or proxy servers.

**Certain Problems with Firewall**

1. Complacency : There are lots of attacks on the firewall from internal users and therefore, it's limitations should be understood.

2. Encapsulated packets : An encapsulated packet is an IP packet within another IP packet. If we ask the router to drop encapsulated packets then, it will drop the multicast packets also.

3. Throughput :So, in order to check which packets are allowed and which are not, we are doing some processing which can be an overhead and thus affects throughput.

**Authentication:**

We can use the following mechanisms:

- One time passwords:  passwords are used only once and then it changes. But only the user and the machine knows the changing passwords.
- password aging : User are forced to change passwords after some time on regular intervals.
- smart cards :  swipe through the PC.
- biometrics : eyes or finger prints are used.

**Packet Filtering :**

Terms associated:

- Source IP address
- Destination IP address
- Source port #
- Destination port #
- protocol
- interface

Many commercial routers offer a mechanism that augments normal routing and permits a manager to further control packet processing. Informally called a *packet filter,* the mechanism requires the manager to specify how the router should dispose of each datagram. For example, the manager might choose to *filter* (i.e.. block) all datagrams that come from a particular source or those used by a particular application, while choosing to route other datagrams to their destination.

The term *packet filter arises* because the filtering mechanism does not keep a record of interaction or a history of previous datagrams. Instead, the filter considers each datagrams separately. When a datagram first arrives, the router passes the datagram through its packet filter before performing any other processing. If the filter rejects the datagram, the router drops it immediately.

For example, normally I won't allow TFTP, openwin, RPC, rlogin, rsh packets to pass through the router whether from inside or outside and router just discard these packets. But I might put some restrictions on telnet, ftp, http, and smtp packets in order to pass through the router and therefore some processing is to be done before discarding or allowing these packets.

Because TCP/IP does not dictate a standard for packet filters, each router vendor is free to choose the capabilities of their packet filter as well as the interface the manager uses to configure the filter.

Some routers permit a manager  to configure separate filter actions for each interface, while others have a single configuration for all interfaces. Usually, when specifying datagrams that the filter should block, a manager can list any combination of source IP address, destination IP address, protocol, source protocol port number, and destination protocol port number.
So, these filtering rules may become more tricky with complex network policies.

Since, Filtering rules are based on port numbers, there is a problem with RPC applications. First, the number of well-known ports is large and growing. Thus, a manager would need to update such a list continually because a simple error of  omission could leave the firewall vulnerable. Second, much of the traffic on an internet does not travel to or from a well-known port. In addition to programmers who can choose port numbers for their private client-server applications, services like *Remote Procedure Call (RPC)* assigns port dynamically. Third, listing ports of well-known services leaves the firewall vulnerable to *tunneling*, a technique in which one datagram is temporarily encapsulated in another for transfer across part of an internet.

**<u>Relay Software (proxies)</u> :**

I can run multiple proxy on same machine. They may detect misuse by keeping loops. For example, some machine give login to Ph.D.. students. So, in this case it's better to keep proxy servers than to give login on those machines. But the disadvantage with this is that there are two connections for each process.

```
 _________                 __________
|         |               |          |
|  User   |_______________| Proxy    |___________   Outside
|_________|       1.      |__________|      2.
```

**<u>Various Firewall Considerations</u>**

1. <u>Packet Filtering Firewall</u>
This is the simplest design and it is considered when the network is small and user don't run many Intranet applications.

```
                         __________
                        |          |
Intranet  ______________| Router   |___________  Internet
                        |__________|
                             |
                             |
                           Filter
```

2. <u>Dual home gateway</u>
This gives least amount of flexibility. Instead of router, we have application gateways.

```
                       ______________
                      | Application  |
Inside  ______________|    level     |___________  Outside
                      |   gateway    |
                      |______________|
                          proxy
```

3. <u>Sreened host Firewall</u>
It's the combination of the above two schemes. Some applications are allowed uninterrupted while some have to be screened. For any reasonable size network, Screened host firewall can get loaded.

```
                 _________                            ___________
                |         |                          |           |
   Inside  _________| Router 1  |________________________ | Router 2    |______  Outside
                |_________|          |                |__________ |
                                 ____|______
                                |           |
                                |   Proxy   |
                                |___________|
```

The problem with this is that there is only one proxy and thus, it may get overloaded. Therefore, to reduce load, we can use multiple screened host firewalls. And this is what normally used.

```
             _________                                     __________
            |         |                                   |          |
  Inside  _____ | Router 1  |________________________________  | Router 2    |_____Outside
            |_________|        |                           |__________ |
                           ____|____
                          |         |
                          | Proxy 1  |     Proxy2   .......
                          |________ |
```

**<u>Modem pool</u>**

User can dial and open only a terminal server but he has to give a password. But TELNET and FTP client does not understand proxy. Therefore, people come out with *Transparent proxy* which means that I have some memory which keeps track of whether this packet was allowed earlier or not and therefore, I need not check this time. Client does not know that there is somebody who is checking my authentication.
So, transparent proxy is used only for checking the IP packets whereas proxy is used when many IP addresses are not available.

**<u>Private IP</u>** (PIP address)
It is an extension of transparent proxy. Here we also change the IP address (source address) to one of the allocated IP address and send it. So, the client does not know that the IP address has been changed, only the proxy server knows it. The machine that changes the IP address is *Network address translator (*NAT) . NAT also changes other things like CRC, TCP header checksum  ( this is calculated using pseudo IP header). NAT can also change the port number.

   e.g..   Port address translation

```
                                    ____________
                    X  -------|              |
                         |     NAT       |
                Y -------|______________ |
```

$X_1$ , $P_1$ ----> $G_1$ , $P_a$ (IP address, port #)
X1 , P2 ----> G1 , Pb
Y , P3 ----> G1, Pc

I may not like to have global IP address because then, anybody can contact me inspite of these security measures. So, I work with Private IP. In that case, there has to be a one-to-one mapping between private IP and global IP.

---

back to top
Prev| Next | Index

**Computer Networks (CS425)**

**Instructor: Dr. Dheeraj Sanghi**

Prev | Next | Index

---

## Wireless Networks

### Introduction

As the need of communication became more and more demanding, new technologies in the field of networks developed. One of them is the use of wireless networks. It is the transmission of data from source to destination without the use of wires as the physical media.

### Why to use Wireless?

Three reasons may be stated for the over-growing use of wireless networks across the world:

1. They are ubiquitous networks. As the do not require messy wires as a medium of communication, they can be used to connect far-off places.
2. They are cheaper than wired networks specially in the case of long-distance communication.
3. They are pretty effective and fast, especially with the modern advancements in this field.

### Some Terms and Technologies:

**ATM-Asynchronous Transfer Mode:**
ATM is a connection-oriented switching technology. It was built to support ISDN (Integrated Services Digital Network). ISDN required high speed cables for both its narrow band (64 Kbps) and broad band (155 Mbps) transmission. There were two technologies available for transmitting data-

1. **Circuit Switching:** In this technology, when a user makes a call, the resources are reserved for him. The advantage of this technology is that it prevents collisions among various users. But the disadvantage is that it leads to inefficient utilization of bandwidth-- if the user fails to

send data or if the transmission speed is faster than the speed of sending data. then most of the bandwidth is wasted.

2. **Packet Switching:** In this technology, resources are never reserved for any particular user. The advantage of this technology is that it leads to efficient utilization of bandwidth i.e. the channel is never free until & unless there are no users, But the disadvantage is that it causes many collision.

ATM was built as a combination of the best features of these two. Also ATM provides QoS (Quality of Service) based on the following priority pattern:

1. **CBR-Constant Bit Rate:** Jobs that can tolerate no delay are assigned the CBR priority. These jobs are provided same number of bits every frame.. For example, viewing a video reel definitely requires some blocks in every frame.
2. **VBR-Variable Bit Rate:** Jobs that may produce different sized packets at different times are assigned VBR priority. They are provided with a variable number of bits varying between a maximum and a minimum in different frames. e.g.. a document may be compressed differently by different machines. Transmitting it will be a variable transmission.
3. **ABR-Available Bit Rate:** This is the same as VBR except that it has only the minimum fixed. If there are no CBR or VBR jobs left, it can use the entire frame,
4. **UBR-Unavailable Bit Rate:** These jobs are the least priority jobs. The network does not promise anything but simply tries its best to transmit it.

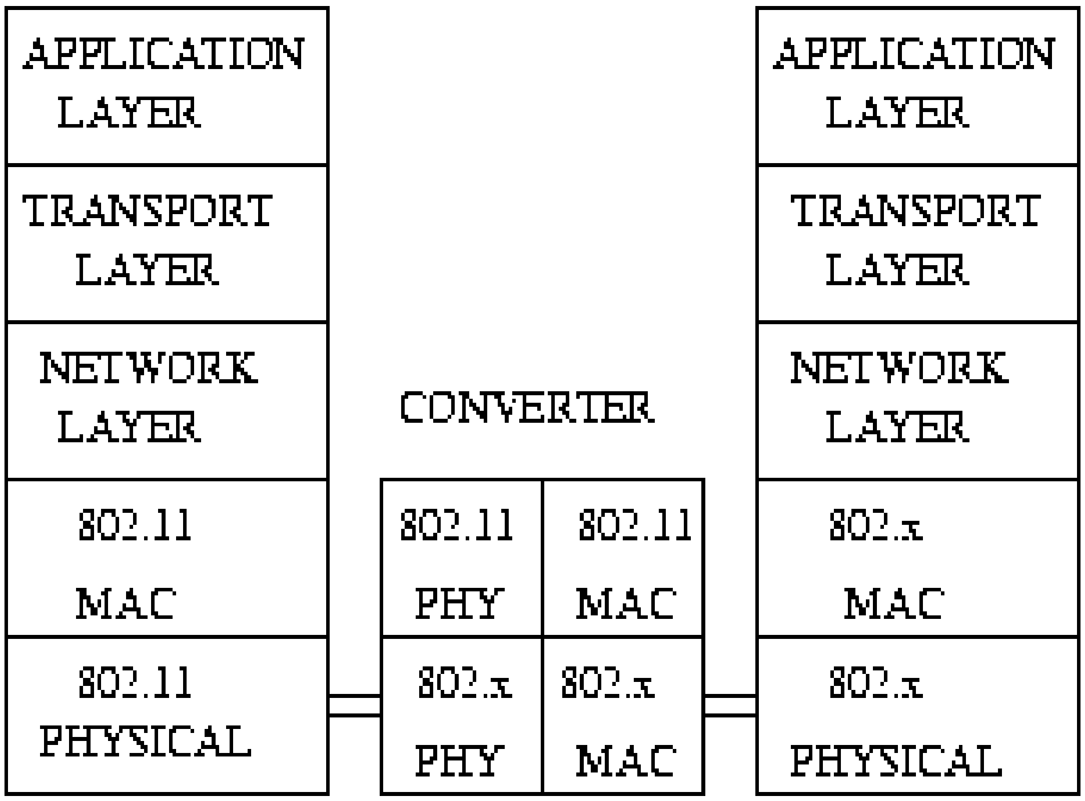
**WLAN-Wireless LAN**
This is currently being used as dictated by the standards of IEEE 802.11. It can be installed at the medium access layer and the data transmission can occur using a converter to reach the wired LAN network.( IEEE 802.x)

**WATM-Wireless ATM**
It is the wireless version of ATM. It provides QoS. It is not yet available in market. because installing it will require the simultaneous installation of ATM infrastructure. It is currently being tested thoroughly.

**Coupling of Networks:**
The alternatives are:

1. WLAN LAN
2. WATM LAN
3. WLAN ATM
4. WATM ATM

1. WLAN-LAN is the simplest of the above. According to the IEEE standards, the IEEE 802.11 (WLAN) can be used with IEEE 802.x (LAN) as follows:



2. WLAN-ATM- NOT FEASIBLE.
3. WATM-LAN- NOT FEASIBLE because WATM requires an infrastructure of the type ATM
4. WATM-ATM-this is also a simple scheme because WATM can run on ATM.

**Issues involved in Wireless Networks**

- **Cost and Speed:** As it is being considered as an alternative to wired networks, it should be faster and cheaper.
- **Quality of Transmission:** It gives a higher BER (Bit Error Rate). The BER is greater than $10^{-6.}$ This is caused because transmission quality depends highly on the physical media including landscape, weather etc.
- **RayLeigh Fading:** The data has to travel the distance through a medium like air. Several rays of the same stream cause Rayleigh fading due to interference. This causes poor transmission.
- **Multipath Propagation:** Similarly, due to multipath propagation, the signal received at the destination may be garbled.
- **Hand-Offs:** If hand-offs are used i.e., hexagonal cells each having a base station and many mobile terminals, two Mobile terminals that are far enough can use the same bandwidth. This reuse of bandwidth is helpful.

- **Dynamic Physical Characteristics:** The terminal may be mobile and constantly moving. Thus the distance between the base station and any active terminal may be constantly changing. This has to be taken into account while designing.
- **Practical Implementation**: The practical implementation of any wireless network requires CSMA/CD for proper transmission. The range of any terminal is fixed. So, there may be two terminals that are out of range of each other. These are called HIDDEN TERMINALS. Collisions may be caused due to simultaneous sending of data from any two hidden terminals. The HIDDEN TERMINAL PROBLEM should be overcome with the help of Base Station.
- .**Mobility and Network Topologies:** Wireless networks should be effective enough to overcome the problems caused by the topology of the area and the mobility of the terminals
- **Frequency Allocation: Licensed & Unlicensed:** For licensed networks, permission has to be taken from the authorities that grant you a fixed bandwidth which is not used by anybody else while unlicensed networking does not require any such permissions. It just provides with some unlicensed bands which can be used by anybody. Unlicensed bands may thus, cause collisions.
- **Capture Effect:** If there are more than one terminals requiring the attention of the Base Station, the one nearer to the base station may capture it. This unfair access to the base station should be prevented.
- **Power Requirements and Battery:** This problem arises for the Mobile Terminals that run battery or cells. Much dissipation of power is caused when switching from receiving mode to sending mode and vice versa.
- **Human Safety:** Not all bandwidths can be used . Also, the intensity should not be very high as it may lead to several complications in human body e.g.. cataract.

**Wireless Physical Media**

In the wireless physical media, three technologies are used:

1. **Transmission at Infrared frequency:** This is easier to build and set-up. It is mainly used for indoor purposes because the beam has to be focussed and can't cross opaque media like walls etc.
2. **Transmission through Microwave:** This is preferred as it requires low power consumption. (the bandwidth is fixed) But the basic problem is that it requires Line-of-Sight. Also, it requires license.
3. **Transmission at Radio Frequency:** This is the one that is most familiar to us. The bandwidth is pretty large.

**Integrity and Security of the signal**

**Spread Spectrum:** To reduce the effect of noise signals, the bandwidth of the signal is increased tremendously. This is costly but assures better transmission. This is called SPREAD-SPECTRUM. This is used in two ways:

- **FHSS (Frequency hopping spread spectrum)**: The entire packet is not sent at the same bandwidth. Say, it is sent at frequency range A for time T1, frequency range B for time T2, A for T1, B for T2 and so on. The receiver also knows this sequence and so, looks at A for time

T1, then at B for time T2 and so on. Thus this sort of understanding between the sender and receiver prevents the signal from being completely garbled .

- **DSSS (Direct Sequence Spread Spectrum):** This involves sending of coded data instead of the actual data. This code is known to the destination only which can decipher the data now.

The problem still left undealt is that of bursty errors. If there is lot of traffic, interference may hinder the Base Station from receiving data for a burst of time. This is called **"Bursty Errors".**
Such problem are looked at by **MAC**-Medium Access Control.

**MEDIUM ACCESS CONTROL**

To control the traffic, various techniques are used. MAC fulfills the following requirements:

1. **QoS Requirements:** It provides Quality of Service according to the priority of jobs.
2. **Error Control:** Error handling is done using some codes.
3. **Frame Size:** To transmit maximum data, we want the frame-size to be maximum but at the same time, large frame-size highly increases the probability of errors. So, MAC provides a tradeoff between the above two factors determining the size of the frame.
4. **Secure Transmission:** The data meant for a particular receiver is secured from others.
5. **Reasonable Transmission:** If the number of users increases, each should get reasonable service. MAC prevents unfair access to channel.
6. **Efficient utilization of Power:** If a transmitter is always on, it is continuously using power even if there is no data on the channel for it. This is reduced by sending the transmitter to "sleep mode" whenever the battery is going down. In this mode, the transmitter is unable to receive any data.

**Architecture for Wireless Network**

There are two types of architecture possible:

1. **AD-HOC NETWORK**
2. **INFRASTRUCTURE NETWORK**

The Ad-Hoc network can be set up anytime. It does not require a Base Station. It is generally used for indoor purposes.
The Infrastructure network involves Base Station and Mobile Terminals. It provides uplink facility ( link from MT to BS) and downlink facility (link from BS to MT).

**THE MAC PROTOCOL**

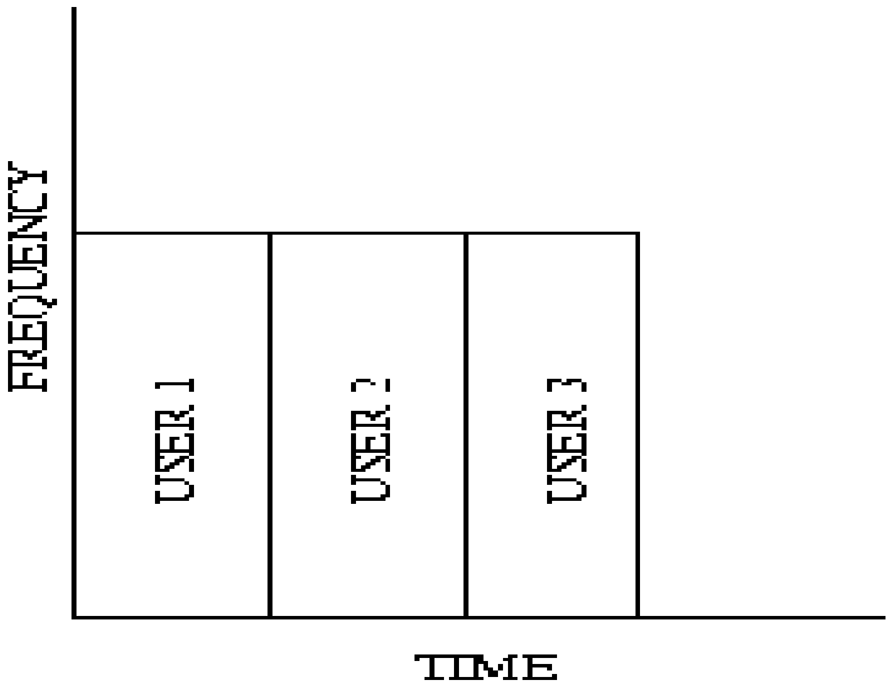
This protocol decides how to assign data slots to different users. The various policies it uses are:

1. **Fixed Assignment Policy**
2. **Random Assignment Policy**
3. **Centrally Controlled Policy**
4. **Distributed Controlled Policy**
5. **Hybrid Controlled Policy**

**Fixed Assignment Policy:**

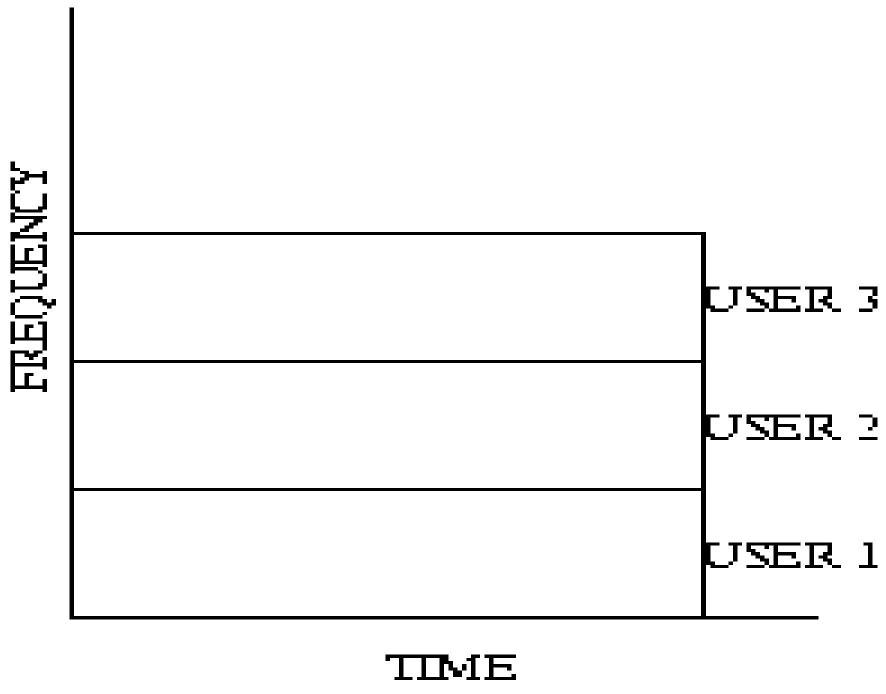
In this policy, each terminal is assigned some sort of data slot to speak. It causes a fixed delay. It is done in 3 ways:

1. **TDMA (TIME DIVISION MULTIPLE ACCESS) :** Each user is given a fixed time to speak., after which the chance goes to another user. This cycle continues indefinitely.

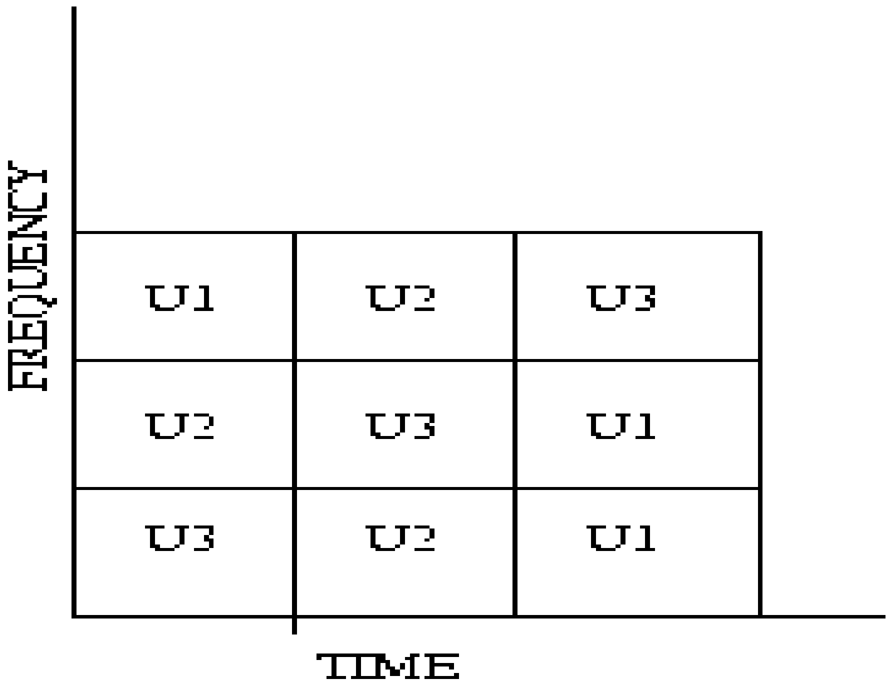


2. **FDMA (FREQUENCY DIVISION MULTIPLE ACCESS):** Each user is given a fixed bandwidth in which he can speak at all

times.



3. **CDMA (CODIVISION MULTIPLE ACCESS):** Each user is given different frequencies at different times. This ensures that each user gets a fair amount of channel each time.



Also, sometimes, statistical multiple access is used in which a slot is assigned to a user only if it has data to send.

**Random Assignment Policy**
In this  policy, contention slots are provided to all the users. Problem may arise if the number of users increase drastically. The number of contention slots should be variable. This may cause some limiting of data slots but is necessary to prevent the derailment of the service .

**Centrally Controlled Policy:**
This is used in an infrastructure architecture.  It involves the participation of a Base Station which may assign slots and priorities(CNBR,VBR etc.) to all the users.

**Distributed Controlled Policy:**
This is used in Ad-Hoc architecture. The control is among the terminals which decide among themselves about who is going to speak first.

**Hybrid Controlled Policy:**
This combines the best features of centrally controlled and distributed controlled policies.

**KINDS OF MAC PROTOCOLS:**

There are two kinds of Mac protocols:

1. **FDD (Frequency Division Duplex)** This provides two separate bandwidths for uplink and downlink transmission. This leads to inefficient utilization of bandwidth as there is  more  traffic on downlink than uplink
2. **TDD (Time Division Duplex)** This provides an adoptive boundary between the uplink and downlink frequency which depends on the what is being used at that particular time. It works as follows:

Any mobile terminal can be in 3 states : empty state, request state and ready-to-transmit state.

1. uplink-MT1 sends a random-access request to BS to communicate with MT2
2. downlink: BS sends a b-bit access id to MT2
3. uplink:  MT1 sends the packet
4. downlink: BS sends the packet to MT2

The TDD is more in use now-a-days.

---

back to top
Prev| Next | Index

**Computer Networks (CS425)**

**Instructor: Dr. Dheeraj Sanghi**

Prev | Next | Index

## Network Security

Data on the network is analogous to possessions of a person. It has to be kept secure from others with malicious intent. This intent ranges from bringing down servers on the network to using people's private information like credit card numbers to sabotage of major organizations with a presence on a network. To secure data, one has to ensure that it makes sense only to those for whom it is meant. This is the case for data transactions where we want to prevent eavesdroppers from listening to and stealing data. Other aspects of security involve protecting user data on a computer by providing password restricted access to the data and maybe some resources so that only authorized people get to use these, and identifying miscreants and thwarting their attempts to cause damage to the network among other things.

The various issues in Network security are as follows :

1. **Authentication:** We have to check that the person who has requested for something or has sent an e-mail is indeed allowed to do so. In this process we will also look at how the person authenticates his identity to a remote machine.
2. **Integrity:** We have to check that the message which we have received is indeed the message which was sent. Here CRC will not be enough because somebody may deliberately change the data. Nobody along the route should be able to change the data.
3. **Confidentiality:** Nobody should be able to read the data on the way so we need Encryption
4. **Non-repudiation:** Once we sent a message, there should be no way that we can deny sending it and we have to accept that we had sent it.
5. **Authorization:** This refers to the kind of service which is allowed for a particular client. Even though a user is authenticated we may decide not to authorize him to use a particular service.

For authentication, if two persons know a secret then we just need to prove that no third person could have generated the message. But for Non-repudiation we need to prove that even the sender could not have generated the message. So authentication is easier than Non-repudiation. To ensure all this, we take the help of cryptography. We can have two kinds of encryption :

1. **Symmetric Key Encryption:** There is a single key which is shared between the two users and the same key is used for encrypting and decrypting the message.
2. **Public Key Encryption:** There are two keys with each user : a public key and a private key. The public key of a user is known to all but the private key is not known to anyone except the owner of the key. If a user encrypts a message in his private key then it can be decrypted by anyone by using the sender's public key. To send a message securely, we encrypt the message in the public key of the receiver which can only be decrypted by the user with his private key.

Symmetric key encryption is much faster and efficient in terms of performance. But it does not give us Non-repudiation. And there is a problem of how do the two sides agree on the key to be used assuming that the channel is insecure ( others may snoop on our packet ). In symmetric key exchange, we need some amount of public key encryption for authentication. However, in public key encryption, we can send the public key in plain text and so key exchange is trivial. But this does not authenticate anybody. So along with the public key, there needs to be a certificate. Hence we would need a public key infrastructure to distribute such certificates in the world.

**Key Exchange in Symmetric Key Schemes**

We will first look at the case where we can use public key encryption for this key exchange. . The sender first encrypts the message using the symmetric key. Then the sender encrypts the symmetric key first using it's private key and then using the receiver's public key. So we are doing the encryption twice. If we send the certificate also along with this then we have authentication also. So what we finally send looks like this :

Z : $\text{Certificate}_{\text{sender}} + \text{Public}_{\text{reciever}}(\text{Private}_{\text{sender}}(E_k)) + E_k(M)$

Here $E_k$ stands for the symmetric key and $E_k(M)$ for the message which has been encrypted in this symmetric key.

However this still does not ensure integrity. The reason is that if there is some change in the middle element, then we will not get the correct key and hence the message which we decrypt will be junk. So we need something similar to CRC but slightly more complicated. This is because somebody might change the CRC and the message consistently. This function is called Digital Signature.

**Digital Signatures**

Suppose A has to send a message to B. A computes a hash function of the message and then sends this after encrypting it using its own private key. This constitutes the signature produced by A. B can now decrypt it, recompute the hash function of the message it has received and compare the two. Obviously, we would need the hash functions to be such that the probability of two messages hashing to the same value is extremely low. Also, it should be difficult to compute a message with the same hash function as another given message. Otherwise any intruder could replace the message with another that has the same hash value and leave the signatures intact leading to loss of integrity. So the message along with the digital signature looks like this :

$Z + \text{Private}_{\text{sender}}(\text{Hash}(M))$

**Digital Certificates**

In addition to using the public key we would like to have a guarantee of talking to a known person. We assume that there is an entity who is entrusted by everyone and whose public key is known to everybody. This entity gives a certificate to the sender having the sender's name, some other information and the sender's public key. This whole information is encrypted in the private key of this trusted entity. A person can decrypt this message using the public key of the trusted authority. But how can we be sure that the public key of the authority is correct ? In this respect Digital signatures are like I-Cards. Let us ask ourselves the question : How safe are we with I-Cards? Consider a situation where you go to the bank and need to prove your identity. I-Card is used as a proof of your identity. It contains your signature. How does the bank know you did not make the I-Card yourselves? It needs some proof of that and in the case of I-Cards they contain a counter signature by the director for the purpose. Now how does the bank know the signature I claim to be of the director indeed belongs to him? Probably the director will also have an I-Card with a counter signature of a higher authority. Thus we will get a chain of signing authorities. Thus in addition to signing we need to prove that the signatures are genuine and for that purpose we would probably use multiple I-Cards each carrying a higher level of signature-counter signature pair.

So in order to distribute the public key of this authority we use certificates of higher authority and so on. Thus we get a tree structure where the each node needs the certificates of all nodes above it on the path to the root in order to be trusted. But at some level in the tree the public key needs to be known to everybody and should be trusted by everybody too.

---

back to top
Prev| Next | Index

sses messages across a link from one.machine to another. The mail is enclosed in what is called an **envelope** . The enveilope contains the To and From fields and these are followed by the mail . The mail consists of two parts namely the Header and the Data.

**Computer Networks (CS425)**

**Instructor: Dr. Dheeraj Sanghi**

Prev | Next | Index

---

## Network Security(Contd...)

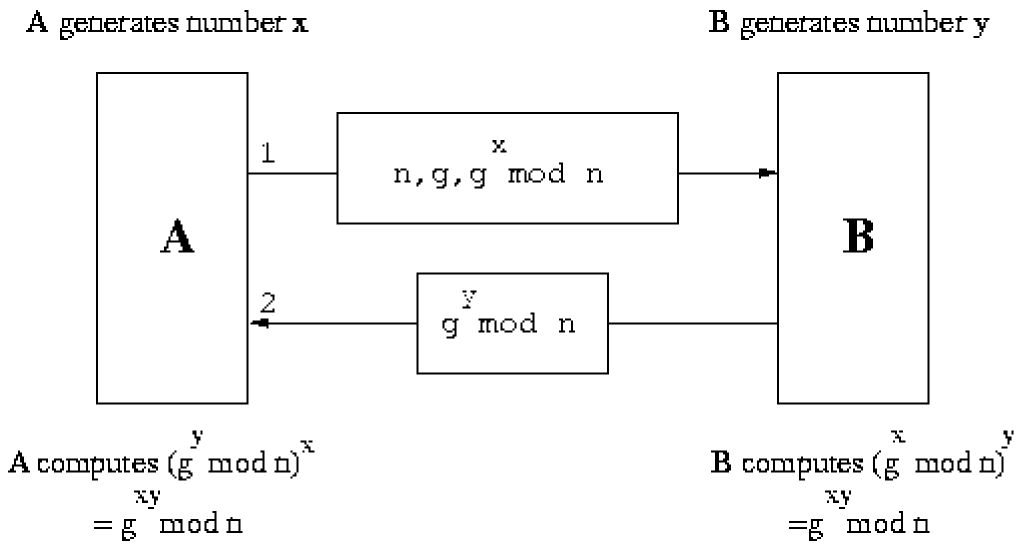
### Key Exchange in Symmetric Key Schemes (contd.)

In this lecture we will look at key exchange in symmetric key schemes where public key encryption cannot be used. So the encryption using public and private keys is not possible. We will see that in this scenario how do we exchange the symmetric key. The two people who are communicating do not want others to understand what they are talking about. So they would use a language which others possibly do not understand. But they have to decide upon a common language. For this the language has to be encrypted in some key which will be somehow known to the other person.

Key exchange in symmetric key schemes is a tricky business because anyone snooping on the exchange can get hold of the key if we are not careful and since there is no public-private key arrangement here, he can obtain full control over the communication. There are various approaches to the foolproof exchange of keys in these schemes. We look at one approach which is as follows:-

### Diffie - Hellman Key Exchange

A and B are two persons wishing to communicate. Both of them generate a random number each, say x and y respectively. There is a function f which has no inverse. Now A sends f(x) to B and B sends f(y) to A. So now A knows x and f(y) and B knows y and f(x). There is another function g such that g(x,  f(y)) = g(y, f(x)). The key used by A is g(x, f(y)) and that used by B is g(y, f(x)). Both are actually same. The implementation of this approach is described below :



A generates number x
B generates number y
A
B
1
n, g, g^x mod n
2
g^y mod n
A computes (g^y mod n)^x = g^xy mod n
B computes (g^x mod n)^y = g^xy mod n

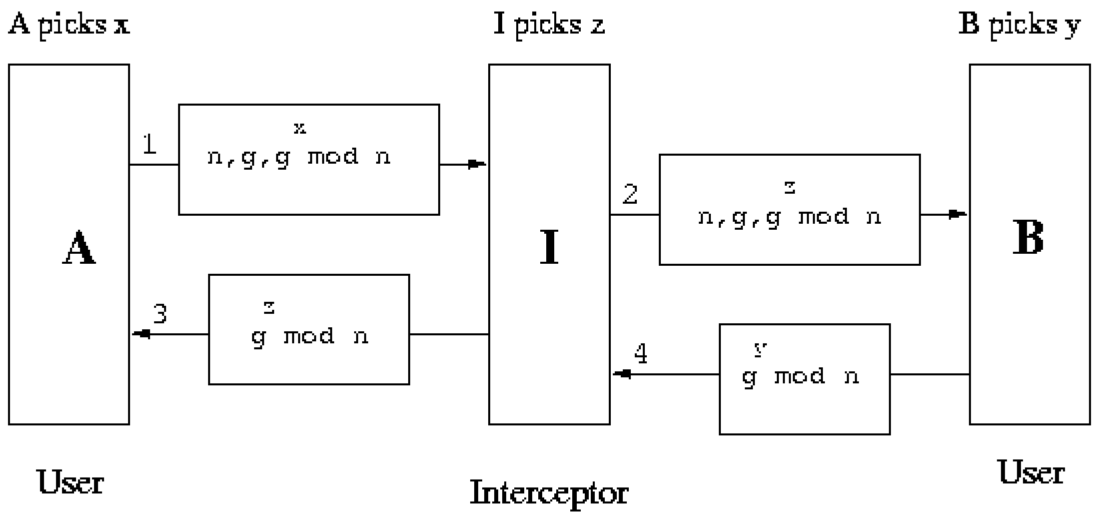
**The Diffie-Hellman Key Exchange**

1. A has two large prime numbers **n** and **g**. There are other conditions also that these numbers must satisfy.
2. A sends **n, g and $g^x$ mod n** to B in a message. B evaluates **($g^x$ mod n)$^y$** to be used as the key.
3. B sends **$g^y$ mod n** to A. A evaluates **($g^y$ mod n)$^x$** to be used as the key. So now both parties have the common number **$g^{xy}$ mod n**. This is the symmetric (secret communication) key used by both A and B now.

This works because though the other people know **n, g, $g^x$ mod n, $g^y$ mod n** but still they cannot evaluate the key because they do not know either x or y.

**Man in the Middle Attack**

However there is a security problem even then. Though this system cannot be broken but it can be bypassed. The situation which we are referring to is called the **man-in-the-middle attack**. We assume that there is a guy C in between A and B. C has the ability to capture packets and create new packets. When A sends **n, g and $g^x$ mod n,** C captures them and sends **n, g and $g^z$ mod n** to B. On receiving this B sends **n, g and $g^y$ mod n** but again C captures these and sends **n, g and $g^z$ mod n** to A. So A will use the key **($g^z$ mod n)$^x$** and B will use the key **($g^z$ mod n)$^y$** . Both these keys are known to C and so when a packet comes from A, C decrypts it using A's key and encrypts it in it's own key and then sends it to B. Again when a packet comes from B, it does a similar thing before sending the packet to A. So effectively there are two keys - one operating between A and C and the other between C and B.



**Security problem with the Diffie-Hellman Exchange**

There must be some solution to this problem. The solution can be such so that we may not be able to communicate further ( because our keys are different ) but atleast we can prevent C from looking at the data. We have to do something so that C cannot encrypt or decrypt the data. We use a policy that A only sends half a packet at a time. C cannot decrypt half a packet and so it is stuck. A sends the other half only when it receives a half-packet from B. C has two options when it receives half a packet :

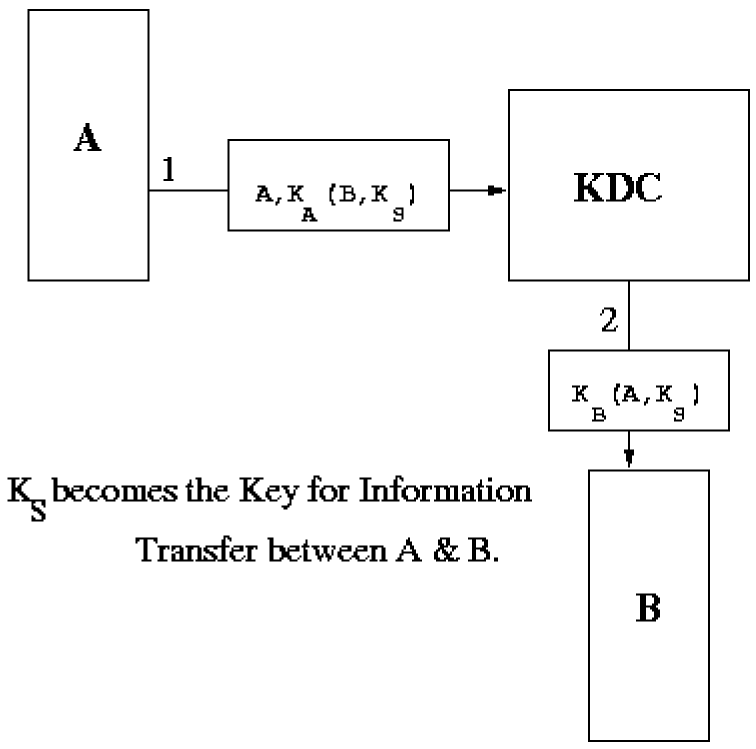
1. It does not send the packet to B at all and dumps it. In this case B will anyway come to know that there is some problem and so it will not send it's half-packet.
2. It forwards the half-packet as it is to B. Now when B sends it's half-packet, A sends the remaining half. When B decrypts this entire packet it sees that the data is junk and so it comes to know that there is some problem in communication.

Here we have assumed that there is some application level understanding between A and B like the port number. If A sends a packet at port number 25 and receives a packet at port number 35, then it will come to know that there is some problem. At the very least we have ensured that C cannot read the packets though it can block the communication.

There is another much simpler method of exchanging keys which we now discuss :

**Key Distribution Center**

There is a central trusted node called the Key Distribution Center ( KDC ). Every node has a key which is shared between it and the KDC. Since no one else knows A's secret key (**$K_A$**) KDC is sure that the message it received has come from A. We show the implementation through this diagram :



## The Concept of Key Distribution Center ( KDC )

When A wants to communicate with B, it sends a message encrypted in it's key to the KDC. The KDC then sends a common key to both A and B encrypted in their respective keys. A and B can communicate safely using this key. There is a problem with this implementation also. It is prone to **replay attack**. The messages are in encrypted form and hence would not make sense to an intruder but they may be replayed to the listener again and again with the listener believing that the messages are from the correct source. To prevent this, we can use:

- **Timestamps:** which however don't generally work because of the offset in time between machines. Synchronization over the network becomes a problem.
- **Nonce numbers:** which are like ticket numbers. B accepts a message only if it has not seen this nonce number before.

---

back to top
Prev| Next | Index

ed to prove your identity. I-Card is used as a proof of your

**Computer Networks (CS425)**

**Instructor: Dr. Dheeraj Sanghi**

Prev | Next | Index

**Network Security(Contd...)**

**Key Distribution Centre(Recap.)**

There is a central trusted node called the Key Distribution Center ( KDC ). Every node has a key which is shared between it and the KDC. Since no one else knows node A's secret key $K_A$, KDC is sure that the message it received has come from A. When A wants to communicate with B it could do two things:

1. A sends a message encrypted in it's key $K_A$ to the KDC. The KDC then sends a common key $K_S$ to both A and B encrypted in their respective keys $K_A$ and $K_B$. A and B can communicate safely using this key.
2. Otherwise A sends a key $K_S$ to KDC saying that it wants to talk to B encrypted in the key $K_{A.}$ KDC send a message to B saying that A wants to communicate with you using $K_{S.}$

There is a problem with this implementation. It is prone to **replay attack**. The messages are in encrypted form and hence would not make sense to an intruder but they may be replayed to the listener again and again with the listener believing that the messages are from the correct source. When A send a message $K_A(M)$, C can send the same message to B by using the IP address of A. A solution to be used is to use the key only once. If B sends the first message $K_A(A,K_S)$ also along with K(s,M), then again we may have trouble. In case this happens, B should accept packets only with higher sequence numbers.

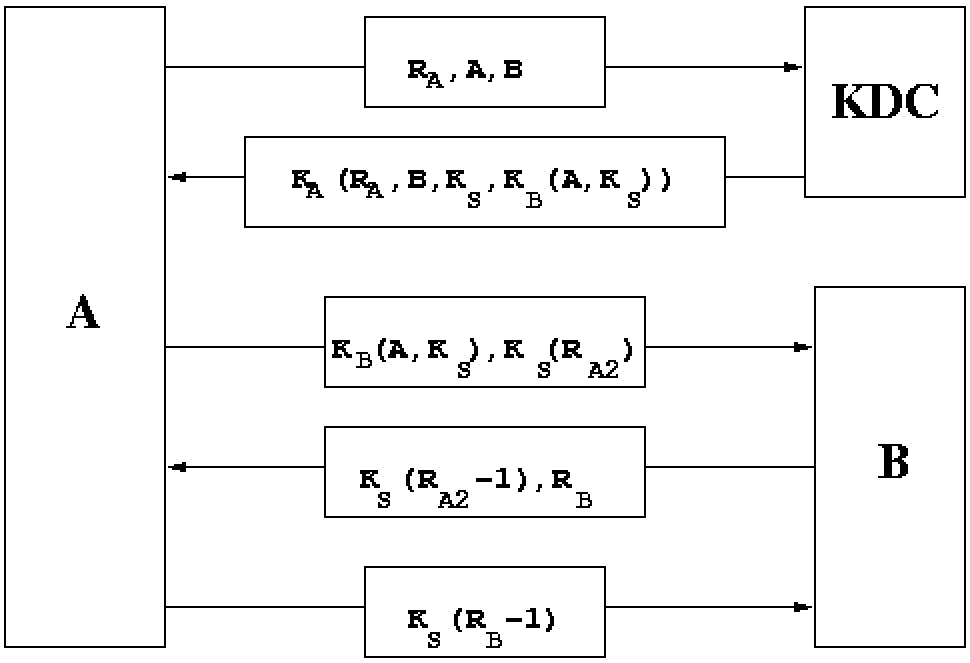
To prevent this, we can use:

- **Timestamps** which however don't generally work because of the offset in time between machines. Synchronization over the network becomes a problem.
- **Nonce numbers** which are like ticket numbers. B accepts a message only if it has not seen this nonce number before.

In general, 2-way handshakes are always prone to attacks. So we now look at an another protocol.

**Needham-Schroeder Authentication Protocol**

This is like a bug-fix to the KDC scheme to eliminate replay attacks. A 3-way handshake (using nonce numbers) very similar to the ubiquitous TCP 3-way handshake is used between communicating parties. A sends a random number $R_A$ to KDC. KDC send back a ticket to A which has the common key to be used.



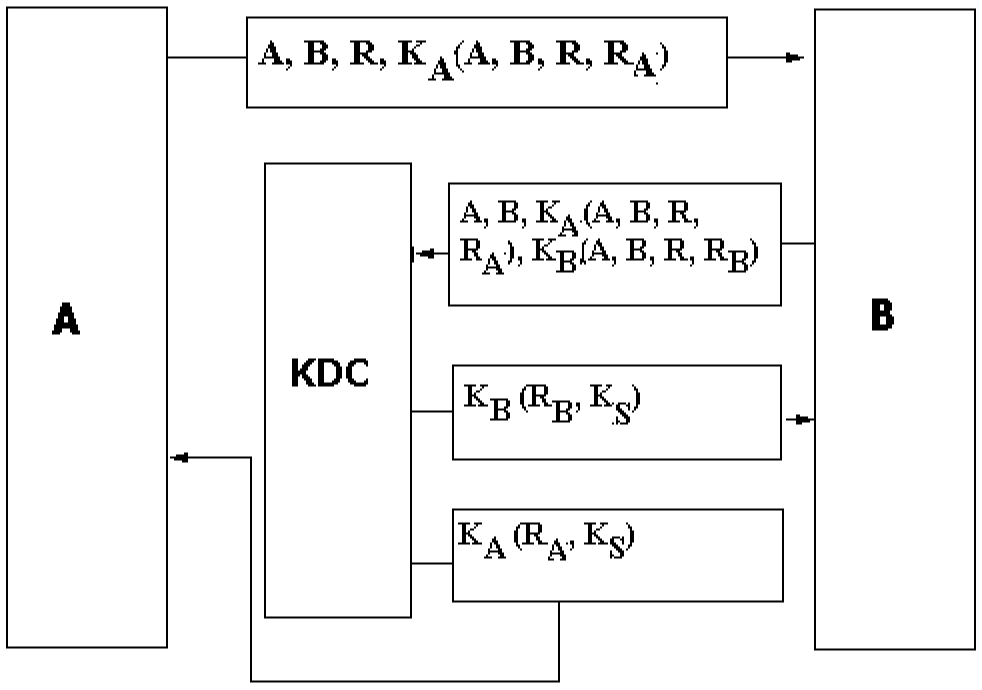
The Needham-Schroeder Authentication Protocol

$R_A$, $R_B$ and $R_{A2}$ are nonce numbers. $R_A$ is used by A to communicate with the KDC. On getting the appropriate reply from the KDC, A starts communicating with B, whence another nonce number $R_{A2}$ is used. The first three messages tell B that the message has come from KDC and it has authenticated A. The second last message authenticates B. The reply from B contains $R_B$, which is a nonce number generated by B. The last message authenticates A. The last two messages also remove the possibility of replay attack.

However, the problem with this scheme is that if somehow an intruder gets to know the key $K_S$ ( maybe a year later ), then he can replay the entire thing ( provided he had stored the packets ). One possible solution can be that the ticket contains a time stamp. We could also put a condition that A and B should change the key every month or so. To improve upon the protocol, B should also involve KDC for authentication. We look at one possible improvement here. which is a different protocol.

**Otway-Rees Key Exchange Protocol**

Here a connection is initiated first. This is followed by key generation. This ensures greater security. B sends the message sent by A to the KDC and the KDC verifies that A, B, R in the two messages are same and $R_A$ and $R_B$ have not been used for some time now. It then sends a common key to both A and B.


A, B, R, K_A(A, B, R, R_A)
A
KDC
A, B, K_A(A, B, R, R_A), K_B(A, B, R, R_B)
B
K_B (R_B, K_S)
K_A (R_A, K_S)

THE OTWAY-REES KEY EXCHANGE PROTOCOL

In real life all protocols will have time-stamps. This is because we cannot remember all random numbers generated in the past. We ignore packets with higher time stamps than some limit. So we only need to remember nonces for this limit. Looking at these protocols, we can say that designing of protocols is more of an art than science. If there is so much problem in agreeing on a key then should we not use the same key for a long time. The key can be manually typed using a telephone or sent through some other media.

**Challenge - Response Protocol**

Suppose nodes A and B have a shared key $K_{AB}$ which was somehow pre-decided between them. Can we have a secure communication between A and B ? We must have some kind of a three way handshake to avoid replay attack So, we need to have some interaction before we start sending the data. A *challenges* B by sending it a random number $R_A$ and expects an encrypted reply using the pre-decided key $K_{AB}$. B then *challenge*s A by sending it a random number $R_B$ and expects an encrypted reply using the pre-decided key $K_{AB.}$

```
     A                                   B

1. A, RA------------->

2.                       <--------KAB(RA), RB

3. KAB(RB)---------->
```

Unfortunately this scheme is so simple that this will not work. This protocol works on the assumption that there is a unique connection between A and B. If multiple connections are possible, then this protocol fails. In replay attack, we could repeat the message $K_{AB}(M)$ if we can somehow

convince B that I am A. Here, a node C need not know the shared key to communicate with B. To identify itself as A, C just needs to send $K_{AB}(R_{B1})$ as the response to the challenge-value $R_{B1}$ given by B in the first connection. C can remarkably get this value through the second connection by asking B itself to provide the response to its own challenge. Thus, C can verify itself and start communicating freely with B.
Thus, replay of messages becomes possible using the second connection. Any encryption desired, can be obtained by sending the value as $R_{B2}$ in the second connection, and obtaining its encrypted value from B itself.

A B

1st Connection: A, $R_A$------------->

<----------$K_{AB}(R_A)$, $R_{B1}$

2nd Connection: A, $R_{B1}$------------>

<--------- $K_{AB}(R_{B1})$, $R_{B2}$

1st Connection: $K_{AB}(R_{B1})$--------->

Can we have a simple solution apart from time-stamp ? We could send $K_{AB}(R_{A,}R_B)$ in the second message instead of $K_{AB}(R_A)$ and $R_{A.}$ It may help if we keep two different keys for different directions. So we share two keys one from A to B and the other from B to A. If we use only one key, then we could use different number spaces ( like even and odd) for the two directions. Then A would not be able to send $R_B$. So basically we are trying to look at the traffic in two directions as two different traffics. This particular type of attack is called **reflection attack**.

**5 - way handshake**

We should tell the sender that the person who initiates the connection should authenticate himself first. So we look at another protocol. Here we are using a 5-way handshake but it is secure. When we combine the messages, then we are changing the order of authentication which is leading to problems. Basically $K_{AB}(R_B)$ should be sent before $K_{AB}(R_A)$. If we have a node C in the middle, then C can pose as B and talk to A. So C can do replay attack by sending messages which it had started some time ago.

A B

1. A------------------>
2. <-----------------$R_B$
3. $K_{AB}(R_B)$---------->
4. $R_A$---------------->
5. <----------$K_{AB}(R_A)$

Fig: 5-way handshake in Challenge-Response Protocol

On initiating a connection B challenges A by sending it a random number $R_B$ and expects an encrypted reply using the pre-decided key $K_{AB.}$ When A sends back $K_{AB}(R_B)$, B becomes sure that it is talking to the correct A, since only A knows the shared key. Now A challenges B by sending it a random number $R_A$, and expects an encrypted reply using the pre-decided key $K_{AB.}$ When B sends back $K_{AB}(R_A)$, A becomes sure that it is talking to the correct B, since only B knows the shared key.

However in this case also, if we have a node C in the middle, then C can pose as B and talk to A. So C can do replay attack by sending messages which it had stored some time ago !!

---

back to top
Prev| Next | Index

addressing. We could say that the network number is 21 bits ( for 8 class C networks ) or say that it is 24 bits and 7 numbers following that. For example : a.b.c.d / 21

**Computer Networks (CS425)**

**Instructor: Dr. Dheeraj Sanghi**

Prev | Next | Index

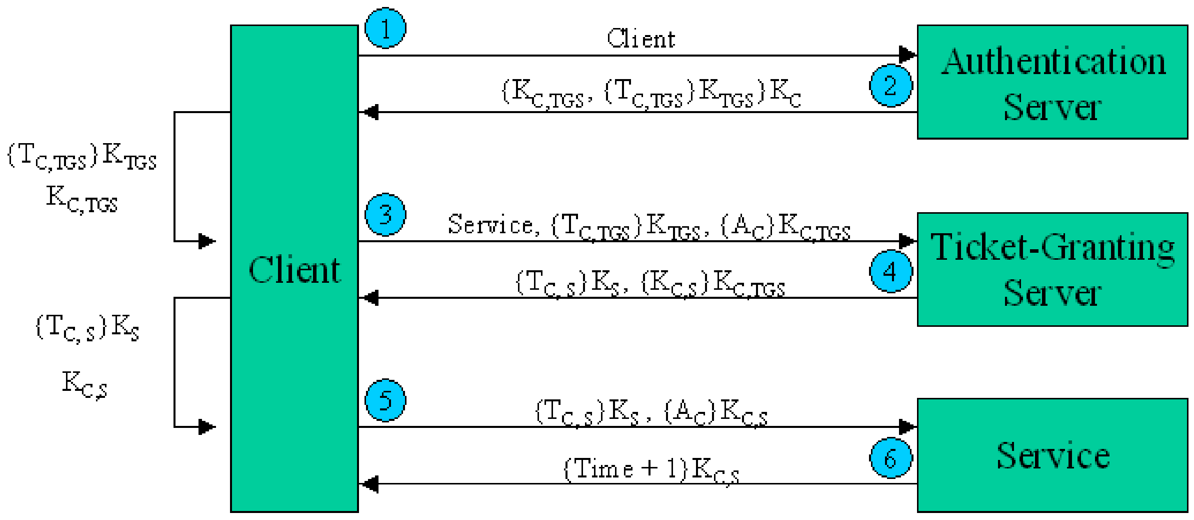
---

## Kerberos

Kerberos was created by Massachusetts Institute of Technology as a solution to many network security problems. It is being used in the MIT campus for reliability. The basic features of Kerberos may be put as:

- It uses symmetric keys.
- Every user has a password ( key from it to the Authentication Server )
- Every application server has a password.
- The passwords are kept only in the Kerberos Database.
- The Servers are all physically secure.(No unauthorized user has access to them.)
- The user gives the password only once.
- The password is not sent over the network in plain text or encrypted form.
- The user requires a ticket for each access.

A diagrammatic representation of the interfaces involved in Kerberos may be put as:


1
Client
Authentication Server
2
$\{K_{C,TGS}, \{T_{C,TGS}\}K_{TGS}\}K_C$
$\{T_{C,TGS}\}K_{TGS}$
$K_{C,TGS}$
3
Service, $\{T_{C,TGS}\}K_{TGS}$, $\{A_C\}K_{C,TGS}$
Client
Ticket-Granting Server
4
$\{T_{C,S}\}K_S$, $\{K_{C,S}\}K_{C,TGS}$
$\{T_{C,S}\}K_S$
$K_{C,S}$
5
$\{T_{C,S}\}K_S$, $\{A_C\}K_{C,S}$
Service
6
$\{Time + 1\}K_{C,S}$

$$T_{C,S} = \{\text{client, service, IP address, timestamp, lifetime, } K_{C,S}\}$$
$$= \text{Ticket for Client to use Service}$$

$$A_C = \{\text{Client, IP Address, Timestamp}\}$$
$$= \text{Client Authenticator}$$

The exchanges of information between the want of transaction by a User with the application server and the time that they actually start exchanging data may be put as:

1. **Client to the Authentication Server(AS):** The following data in plain text form are sent:
   - Username.
   - Ticket Granting Server(TGS) name.
   - A nonce id 'n'.
2. Response from the **Authentication Server(AS) to the Client:** The following data in encrypted form with the key shared between the AS and the Client is sent:
   - The TGS session key.
   - The Ticket Granting Ticket. This contains the following data encrypted with the TGS password and can be decrypted by the TGS only.
     - Username.
     - The TGS name.
     - The Work Station address.
     - The TGS session key.
   - The nonce id 'n'.
3. **Client to the Ticket Granting Server:** This contains the following data
   - The Ticket Granting ticket.

   - Authenticator.
   - The Application Server.
   - The nonce id 'n'
4. **Ticket Granting Server to the Client:** The following data encrypted by the TGS session key is sent:
   - The new session key.
   - Nonce id 'n'
   - Ticket for the application server- The ticket contains the following data encrypted by the application servers' key:
     - Username
     - Server name
     - The Workstation address
     - The new session key.

After these exchanges the identity of the user is confirmed and the normal exchange of data in encrypted form using the new session key can take place. The current version of Kerberos being developed is Kerberos V5.

**Types of Tickets**

1. **Renewable Tickets:** Each ticket has a timer bound , beyond that no authentication exchange can take place . Applications may desire to hold tickets which can be valid for long periods of time. However, this can expose their secret session key to potential theft for equally long periods, and those stolen keys would be valid until the expiration time of the ticket(s). Simply using short-lived tickets and obtaining new ones periodically would require the client to have long-term access to its secret key, an even greater risk. Renewable tickets can be used to mitigate the consequences of theft.
2. **Post Dated Tickets:** Applications may occasionally need to obtain tickets for use much later, e.g., a batch submission system would need tickets to be valid at the time the batch job is serviced. However, it is dangerous to hold valid tickets in a batch queue, since they will be on-line longer and more prone to theft. Postdated tickets provide a way to obtain these tickets from the AS at job submission time, but to leave them "dormant" until they are activated and validated by a further request of the AS. Again this is for additional security.
3. **Proxiable Tickets:** At times it may be necessary for a principal to allow a service to perform an operation on its behalf. The service must be able to take on the identity of the client, but only for a particular purpose. A principal can allow a service to take on the principal's identity for a particular purpose by granting it a proxy. This ticket allows a client to pass a proxy to a server to perform a remote request on its behalf, e.g., a print service client can give the print server a proxy to access the client's files on a particular file server in order to satisfy a print request.
4. **Forwardable Tickets:** Authentication forwarding is an instance of the proxy case where the service is granted complete use of the client's identity. An example where it might be used is

when a user logs in to a remote system and wants authentication to work rom that system as if the login were local.

**Time Stamps:**

- **Authentication:** This is the time when i first authenticated myself .
- **Start:** This is the time when valid period starts.
- **End:** This is the time when valid period ends.
- **Renewal time:** This is the time when ticket is renewed.
- **Current time:** This time is for additional security. This stops using old packets. Here we need to synchronize all clocks.

**Cross Realm Authentication**

The Kerberos protocol is designed to operate across organizational boundaries. A client in one organization can be authenticated to a server in another. Each organization wishing to run a Kerberos server establishes its own "realm". The name of the realm in which a client is registered is part of the client's name, and can be used by the end-service to decide whether to honor a request.

By establishing "inter-realm" keys, the administrators of two realms can allow a client authenticated in the local realm to use its authentication remotely (Of course, with appropriate permission the client could arrange registration of a separately-named principal in a remote realm, and engage in normal exchanges with that realm's services. However, for even small numbers of clients this becomes cumbersome, and more automatic methods as described here are necessary). The exchange of inter-realm keys (a separate key may be used for each direction) registers the ticket-granting service of each realm as a principal in the other realm. A client is then able to obtain a ticket-granting ticket for the remote realm's ticket- granting service from its local realm. When that ticket-granting ticket is used, the remote ticket-granting service uses the inter- realm key (which usually differs from its own normal TGS key) to decrypt the ticket-granting ticket, and is thus certain that it was issued by the client's own TGS. Tickets issued by the remote ticket- granting service will indicate to the end-service that the client was authenticated from another realm.

**Limitations of Kerberos**

- **Password Guessing:** Anyone can get all privileges by cracking password.
- **Denial-of-Service Attack:** This may arise due to keep sending request to invalid ticket.
- **Synchronization of Clock:** This is the most significant limitation to the kerberos.

---

**Public Key Authentication Protocol**

Mutual authentication can be done using public key authentication. To start with let us assume A and B want to establish a session and then use secret key cryptography on that session. The purpose of this initial exchange is authenticate each other and agree on a secret shared session key.

**Setup**

A sends a request to AS for getting B's public key. Similarly B is trying to get the A's public key. AS sends public key of B and name of B in encrypted form using AS's private key.

**Handshake**

Whether it came from A or from someone else., but he plays along and sends A back a message containing A's n1, his own random number n2 and a proposed session key, Ks. When A gets this message, he decrypts it using his private key. He sees n1 in it, and hence gets sure that B actually got the message. The message must have come from B, since none else can determine n1. A agrees to the session by sending back message. When B sees n2 encrypted with the session key he just generated, he knows A got message and verified n1.

**Digital Signatures**

The authenticity of many legal, financial and other documents is determined by the presence or absence of an authorized handwritten signature. The problem of devising a replacement for handwritten signatures is a difficult one. Basically, what is needed is a system bu which one party can send a assigned message to other party in such a way that:

1. The receiver can verify the claimed identity of sender
2. The sender cannot later repudiate the contents of the message.
3. The receiver cannot possibly have concocted the message himself

**Message Digest**

One criticism of signature methods is that they often couple two distinct functions : authentication and secrecy. Often, authentication is needed but secrecy is not. Since cryptography is slow, it is frequently desirable to be able to send signed plaintext documents.One scheme, known as MESSAGE DIGEST, is based on the idea of a one-way hash function that takes an arbitrarily long piece of plaintext and from it computes a fixed length bit string. This hash function has three important properties:

1. Given p, it is easy to compute MD(P).
2. Given MD(P), it is effectively impossible to find P.
3. No one can generate two messages that have the same message digest.

**Main Steps in Authentication**

- Sender computes checksum of message and sends it to AS.
- AS returns signature block. Signature block consists of name and checksum of message in encrypted form using AS's symmetric key.
- Recipient sends signature block to AS.
- AS decrypt signature.
    - verifies sender's name.
    - sends checksum back to recipient.
- Recipient verifies checksum.

---

**Image References**

- http://guir.berkeley.edu/projects/osprelims/summaries/img/kerberos.gif

---

back to top
Prev| Next | Index

ge

**Computer Networks (CS425)**

**Instructor: Dr. Dheeraj Sanghi**

Prev | Next | Index

---

**Network File System (NFS)**

Network File System ( NFS ) is a distributed file system ( DFS ) developed by Sun Microsystems. This allows directory structures to be spread over the net- worked computing systems.

A DFS is a file system whose clients, servers and storage devices are dis- persed among the machines of distributed system. A file system provides a set of file operations like read, write, open, close, delete etc. which forms the file services. The clients are provided with these file services. The basic features of DFS are multiplicity and autonomy of clients and servers.

NFS follows the directory structure almost same as that in non-NFS system but there are some differences between them with respect to:

- Naming
- Path Names
- Semantics

**Naming**

Naming is a mapping between logical and physical objects. For example, users refers to a file by a textual name, but it is mapped to disk blocks. There are two notions regarding name mapping used in DFS.

- **Location Transparency:** The name of a file does not give any hint of file's physical storage location.
- **Location Independence:** The name of a file does not need to be changed when file's physical storage location changes.

A location independent naming scheme is basically a dynamic mapping. NFS does not support location independency.

There are three major naming schemes used in DFS. In the simplest approach, files are named by some combination of machine or host name and the path name. This naming scheme is neither location independent nor location transparent. This may be used in server side. Second approach is to attach or mount the remote directories to the local directories. This gives an appearance of a coherent directory. This scheme is used by NFS. Early NFS allowed only previously mounted remote directories. But with the advent of automount , remote directories are mounted on demand based on the table of mount points and file structure names. This has other advantages like the file-mount table size is much smaller and for each mount point, we can specify many servers. The third approach of naming is to use name space which is identical to all machines. In practice, there are many special files that make this approach difficult to implement.

**Mounting**

The mount protocol is used to establish the initial logical connection between a server and a client. A mount operation includes the name of the remote directory to be mounted and the name of the server machine storing it. The server maintains an export list which specifies local file system that it exports for mounting along with the permitted machine names. Unix uses /etc/exports for this purpose. Since, the list has a maximum length, NFS is limited in scalabilty. Any directory within an exported file system can be mounted remotely on a machine. When the server receives a mount request, it returns a file handle to the client. File handle is basically a data-structure of length 32 bytes. It serves as the key for further access to files within the mounted system. In Unix term, the file handle consists of a file system identifier that is stored in super block and an inode number to identify the exact mounted directory within the exported file system. In NFS, one new field is added in inode that is called the generic number.

Mount can be is of three types -

1. **Soft mount:** A time bound is there.
2. **Hard mount:** No time bound.
3. **Automount:** Mount operation done on demand.

**NFS Protocol and Remote Operations**

The NFS protocol provides a set of RPCs for remote operations like **lookup, create, rename, getattr, setattr, read, write, remove, mkdir** etc. The procedures can be invoked only after a file handle for the remotely mounted directory has been esta- blished. NFS servers are stateless servers. A stateless file server avoids to keep state informations by making each request self-contained. That is, each request iden- tifies the file and the position of the file in full. So, the server needs not to store file pointer. Moreover, it needs not to establish or terminate a connection by opening a file or closing a file, repectively. For reading a directory, NFS does not use any file pointer, it uses a **magic cookie**.

Except the opening and closing a file, there is almost one-to-one mapping between Unix system calls for file operations and the NFS protocol RPCs. A remote file operation can be translated directly to the corresponding RPC. Though conceptu- ally, NFS adheres to the remote service paradigm, in practice, it uses buffering and caching. File blocks and attributes are fetched by RPCs and cached locally. Future remote operations use the cached data, subject to consistency constraints.

Since, NFS runs on RPC and RPC runs on UDP/IP which is unreliable, operations should be idempotent.

**Cache Update Policy**

The policy used to write modified data blocks to the server's master copy has critical effect on the system performance and reliability. The simplest policy is to **write through** the disk as soon as they are placed on any cache. It's advantageous because it ensures the reliability but it gives poor performance. In server site this policy is often followed. Another policy is **delayed write**. It does not ensure reliability. Client sites can use this policy. Another policy is **write-on-close**. It is a variation of delayed write. This is used by Andrews File System (AFS).

In NFS, clients use delayed write. But they don't free delayed written block until the server confirms that the data have been written on disk. So, here, Unix semantics are not preserved. NFS does not handle client crash recovery like Unix. Since, servers in NFS are stateless, there is no need to handle server crash recovery also.

**Time Skew**

Because of differences of time at server and client, this problem occures. This may lead to problems in performing some operations like " make ".

**Performance Issues**

To increase the reliability and system performance, the following things are generally done.

1. Cache, file blocks and directory informations are maintained.
2. All attributes of file / directory are cached. These stay 3 sec. for files and 30 sec. for directory.
3. For large caches, bigger block size ( 8K ) is benificial.

This is a brief description of NFS version 2. NFS version 3 has already been come out and this new version is an enhancement of the previous version. It removes many of the difficulties and drawbacks of NFS 2.

---

**Andrews File System (AFS)**

AFS is a distributed file system, with scalability as a major goal. Its efficiency can be attributed to the following practical assumptions (as also seen in UNIX file system):

- Files are small (i.e. entire file can be cached)
- Frequency of reads much more than those of writes
- Sequential access common
- Files are not shared (i.e. read and written by only one user)
- Shared files are usually not written
- Disk space is plentiful

AFS distinguishes between client machines (workstations) and dedicated server machines. Caching files in the client side cache reduces computation at the server side, thus enhancing performance. However, the problem of sharing files arises. To solve this, all clients with copies of a file being modified by another client are not informed the moment the client makes changes. That client thus updates its copy, and the changes are reflected in the distributed file system only after the client closes the file. Various terms related to this concept in AFS are:

- **Whole File Serving**: The entire file is transferred in one go, limited only by the maximum size UDP/IP supports

- **Whole File Caching**: The entire file is cached in the local machine cache, reducing file-open latency, and frequent read/write requests to the server
- **Write On Close**: Writes are propagated to the server side copy only when the client closes the local copy of the file

In AFS, the server keeps track of which files are opened by which clients (as was not in the case of NFS). In other words, AFS has **stateful servers**, whereas NFS has **stateless servers**. Another difference between the two file systems is that AFS provides **location independence** (the physical storage location of the file can be changed, without having to change the path of the file, etc.) as well as **location transparency** (the file name does not hint at its physical storage location). But as was seen in the last lecture, NFS provides only location transparency. Stateful servers in AFS allow the server to inform all clients with open files about any updates made to that file by another client, through what is known as a **callback**. Callbacks to all clients with a copy of that file is ensured as a **callback promise** is issued by the server to a client when it requests for a copy of a file.

The key software components in AFS are:

- **Vice**: The server side process that resides on  top of the unix kernel, providing shared file services to each client
- **Venus**: The client side cache manager which acts as an interface between the application program and the Vice

All the files in AFS are distributed among the servers. The set of files in one server is referred to as a **volume**. In case a request can not be satisfied from this set of files, the vice server informs the client where it can find the required file.

The basic file operations can be described more completely as:

- Open a file: Venus traps application generated file open system calls, and checks whether it can be serviced locally (i.e. a copy of the file already exists in the cache) before requesting Vice for it. It then returns a file descriptor to the calling application. Vice, along with a copy of the file, transfers a callback promise, when Venus requests for a file.
- Read and Write: Reads/Writes are done from/to the cached copy.
- Close a file: Venus traps file close system calls and closes the cached copy of the file. If the file had been updated, it informs the Vice server which then replaces its copy with the updated one, as well as issues callbacks to all clients holding callback promises on this file. On receiving a callback, the client discards its copy, and works on this fresh copy.

The server wishes to maintain its states at all times, so that no information is lost due to crashes. This is ensured by the Vice which writes the states to the disk. When the server comes up again, it also informs all the servers about its crash, so that information about updates may be passed to it.

A client may issue an open immediately after it issued a close (this may happen if it has recovered from a crash very quickly). It will wish to work on the same copy. For this reason, Venus waits a while (depending on the cache capacity) before discarding copies of closed files. In case the application had not updated the copy before it closed it, it may continue to work on the same copy. However, if the copy had been updated, and the client issued a file open after a certain time interval (say 30 seconds),  it will have to ask the server the last modification time, and accordingly, request for a new copy. For this, the clocks will have to be synchronized.

---

back to top
Prev| Next | Index