UNIT 1

Introduction to Computer Networking

Contents:

1.1 Introduction

The computer network has revolutionized many aspects of our daily lives. It has affected the way we communicate, the way we gather information, the way we do business and even the way we spend our leisure time. The computer network is a communication system that has brought a wealth of information to our fingertips and organized it for our use.

Whenever studying about a particular subject, one would be naturally curious to know how actually things started. So we begin with a brief section about how computer networks started, and then go into the research activities that went into the development of computer networks. The computer networks or the *Internet* (which is the most popular, widely used, global computer network) is a structured, organized system. To understand how it works first we need to define the concepts of protocol and protocol architecture. A section is devoted for the same purpose.

There are two protocol architecture models; one is the OSI model, which is generally used as a reference model. Study of this model gives very good understanding about the computer network architecture. Another protocol architecture model is the TCP/IP model, which is the most widely used protocol model. It is based on this model the Internet is built. A brief introduction to the TCP/IP model is given with the intention of giving an overall picture of it. Later chapters would discuss in detail the various protocol modules of the TCP/IP. A comparison of TCP/IP with the OSI model is done at the end and the organizations, which sets the standards in the computer network is also explained.

1.2 Objectives:

In this Unit you would learn about

- How computer networks evolved
- The research activities that went into the computer networks in its earlier days
- What is meant by protocol and protocol architecture
- A brief study about the two protocol models; OSI and TCP/IP

- About organizations who set standards for protocols.

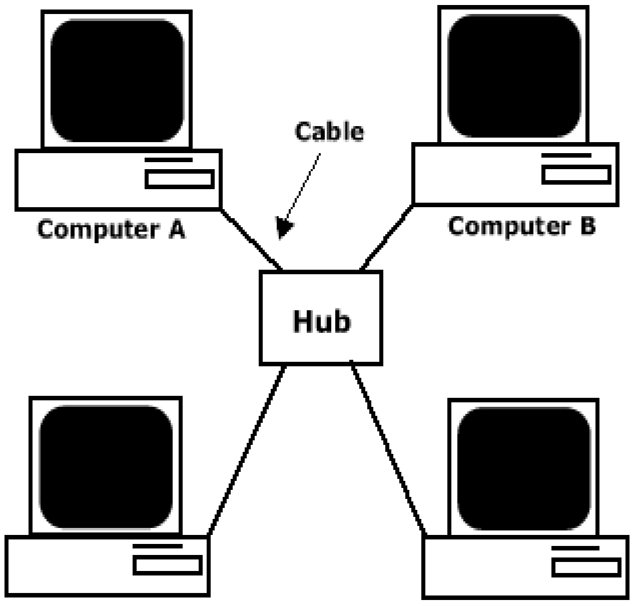
1.3 How networking of computers started:

Once, computer systems used to be stand-alone units. They were supposed to have all the hardware peripherals and software packages required to do a particular job. If the computer system needed printing capability, then a printer was connected to it, and corresponding software to drive the printer was loaded. Only after that one was able to print whatever document stored in the system.

As the usage of computer grew, a new requirement arose. *Need to share / exchange the information and resources amongst the computers and their users.* Information exchange can be electronic mail or file transfer. Resource sharing can be usage of peripheral device, such as printer, connected to other computer. It made no sense to have printer connected to each of the computers in Universities and the research laboratories (that's where the usage of computer was high in the beginning).

Initially the information exchange was in the form of exchange of magnetic tapes. When one wanted to exchange a document with others, then they were required to store the document file in a removable magnetic tape and carry it to other's computer and access it there. Obviously when the two computers involved are not very near, the process of transferring file from one computer to another becomes very time consuming.

When computers can be connected to the external peripheral devices like printer through a cable and information (in the format understood by them) can be exchanged between them, why not two individual computer systems be connected in the same way and exchange information? That would make the job of file / document sharing among users of different computers much simpler. Next step was very obvious – when two computers can be connected why not the rest of the computers in the University. This led to the implementation of a local campus wide network in which the computers in the campus were connected, enabling the exchange of information and sharing of resources within the campus to happen in an efficient manner.



If computers within a campus can be interconnected through the cables, why not use the existing telephone network to interconnect the computers situated in distant locations. This should not be a difficult job as the digital information stored in the computer is transferred to another computer through the cable in the form of electromagnetic field. The same principle (i.e. electromagnetic signal) is applied even in telephone network to transfer the audio signal from one end to another. So why not use the telephone network, in which case one can interconnect the computers located in far-off places. These ideas didn't remain as pipe dream, instead they became reality.

One followed the other and simultaneous advancement in the *data communication* field and advancement in the field of microelectronics which is responsible for the faster, smaller, cheaper, powerful computer processors and other devices which are basic building blocks of computer led to a situation wherein today we have a world wide *network of computers* widely known as *Internet* which connects several crores of computer of all capacities across the globe (probably the astronauts in MIR Space station can also access Internet, if they wish and are permitted).

A network can be as simple as two personal computers connected together using a modem of 1200 baud, or a complex as the TCP/IP Internet which was designed to handle heterogeneous systems running on unreliable communication link. There are number of ways to connect the computer to network. And also the things, which one can do once, connected to the network are many.

1.4 Research activities in the early days of Computer networks:

In the early days when the computer networking was taking shape in the Universities and other Research laboratories the things that were commonly done on network were

- Exchanging of mails with others,
- Exchange of files and other documents,
- Execution of computer programme on other remote computer. It was possible to sit at the ones department computer and from there connect to mainframe computer (which was faster apart from other things) of the university (or if permission was granted on some other research laboratory in far off place) and run the computer programme.
- Remote login, i.e. login to remote computer system (requirement of the permission goes without saying), and once one is logged on to the system, he becomes a regular user of the computer just like others who have logged in directly and not through network.
- Use the printer and other peripheral devices connected to the remote computer.

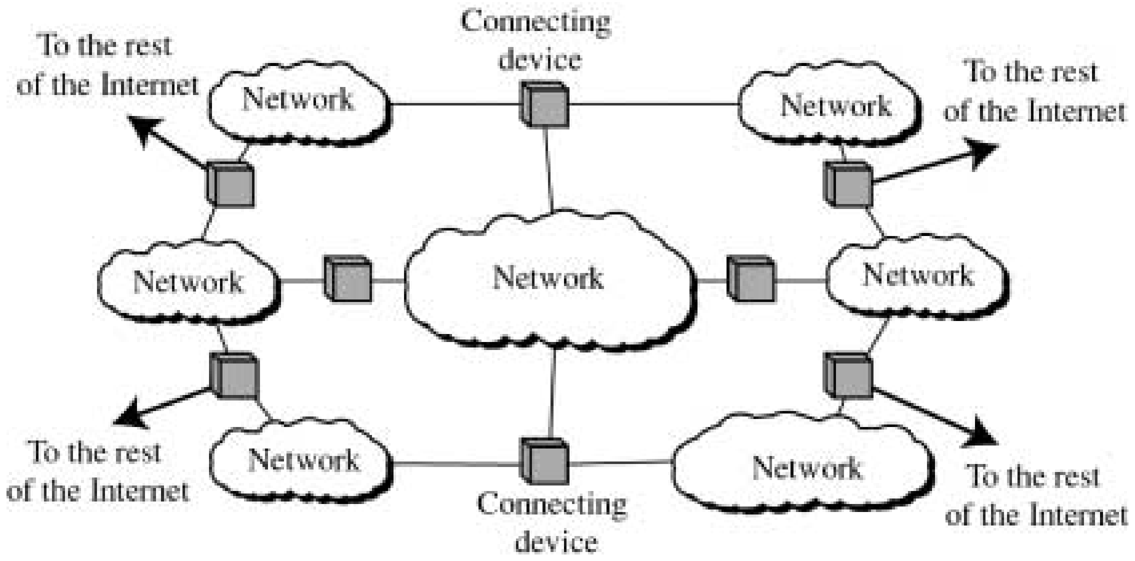
In fact it is the combination of the above things, which had greater impact. With the above possibilities, it was possible to have a group of scientists, professors, their students, administrators, from geographically distant places to work on a particular project overcoming the barrier of distance.

Possibilities like the above and simultaneous advancement in the field of computers and data communication fuelled further research into computer networks.

In fact it was the U.S government (for the military purpose), which took the initiative and funded the project for the development of networks known as ARPANET (Advanced Research Projects Agency Network) in the 70’s and early 80’s of twentieth century. Original idea of the government was to have all the computers of the military establishment to be connected, and the network should be so robust and reliable even in the event of war breaking out with the enemy state and some of the network infrastructure is damaged, the remaining computers should be able to work as a network without any problem. There should be no single point of control for the network, reliability on the data communication channels should be minimum and no assumptions should be made on the kind of computer and communication channel. In other words it was supposed to support heterogonous computer system and communication channel both of which are unreliable.

The researchers came out with a protocol known as TCP/IP (Transmission Control Protocol / Internet Protocol). This protocol gave the guidelines that should be followed by the participating computer systems in order to communicate as intended. Originally this was implemented on a network known as ARPANET. Most of the computers connected to this were belonging to the military establishment apart from the participating universities.

Later funding was provided to the University of California, Berkeley, to implement the protocol in their popular version of UNIX operating system, which was known as Berkeley Unix. This flavor of UNIX operating system was very popular amongst the academic community as the source code of the Operating System was not secret but was in the public domain. It was the implementation of the TCP/IP in the Berkeley UNIX, which led to the widespread implementation of networks based on TCP/IP as the graduates coming out of the Universities and getting employed in the industry were at ease with it. Since many networks were implemented using TCP/IP protocol, it became very easy to interconnect those *networks*, which were talking the same language of networking (TCP/IP). Thus born the *network of computer networks*, which is today known is Internet.


To the rest of the Internet
Connecting device
Network
Network
To the rest of the Internet
Network
Network
Network
To the rest of the Internet
Network
Network
Connecting device
To the rest of the Internet

TCP/IP is not the only protocol available for the computer networks. In fact during the same period of TCP/IP development there were many other protocols implemented by the industry. But none of them could succeed like the TCP/IP, which is the core of today’s Internet.

Now the computer network has become dominating media, sociologists are discussing about the impact of it on the society. New laws are getting enacted to bring some order to it in many countries.

In the earlier days one was able to exchange mails, document, execute programmes and do other such things. Now it is probably easy to list what one cannot do with the Internet instead of what one can do. In fact one can read tomorrows newspaper today (night) itself! This is possible as many newspaper companies have presence on the Internet and most of them update the content on previous night itself as soon as it is ready for the print.

The Internet is a structured, organized system. To understand how it works and its relationship to TCP/IP, first we need to define the concepts of protocols and standards. Also, we need to be aware of the various organizations that are involved in the development of Internet Standards.

1.5 Protocols and Protocol Architecture

Diplomats follow *rules* when they conduct business between nations, which is referred to in the media as protocol. Diplomatic protocol requires that one shouldn't insult his hosts and do respect local customs. Most embassies and commissions have specialists in protocol, whose function is to ensure that everything proceeds smoothly when communications are taking place. The protocol is a set of rules that must be followed in order to make sure that there will be no misunderstanding between the nations because of lack of proper communication.

Similarly, *computer protocols* define the manner in which communications take place. If one computer is sending information to another and they both follow the protocol properly, the message gets through; regardless of what types of systems they are and what operating systems they run (the basis for open systems). As long as the machines have software that can manage the protocol, communications are possible. Essentially, *a computer protocol is a set of rules that coordinates the exchange of information.*

Protocols have developed from very simple processes ("I'll send you one character, you acknowledge its receipt, and then I send the next character") to elaborate, complex mechanisms that cover all possible problems and transfer conditions.

A task such as sending a message from one side of the globe to another side can be very complex when you consider the manner in which it moves. A single protocol to cover all aspects of the transfer would be too large, unwieldy, and overly specialized. Therefore, several protocols have been developed, each handling a specific task.

Combining several protocols, each with their own dedicated purposes, would be a nightmare if the interactions between the protocols were not clearly defined. The concept of a layered structure was developed to help keep each protocol in its place and to define the manner of interaction between each protocol (essentially, a protocol for communications between protocol modules!).

Consider, for example, a particular user browsing a web site for information. There must be a data path between the two computers involved, one computer which is used by the user where browser application (like *Internet Explorer)* is running and another computer where actually the information requested by the user is stored (which is known as a WWW server). This data path can either be a direct one or via a communication network. But more is needed. Typical tasks to be performed are

1. The source system must either activate the direct data communication path or inform the communication network of the identity of the desired destination system.
2. The *source system* must ascertain that the *destination system* (WWW server) is prepared to receive query and pass on the information requested by in the query.
3. The *browser application* on the source system must ascertain that the required WWW *server programme* on the destination system is prepared to accept query for information for the user.
4. If the file formats used on the two systems are incompatible, one or the other system must perform a format translation function.

It is clear that there must be a high degree of cooperation between the two computer systems.

In discussing computer networks, two concepts are paramount:

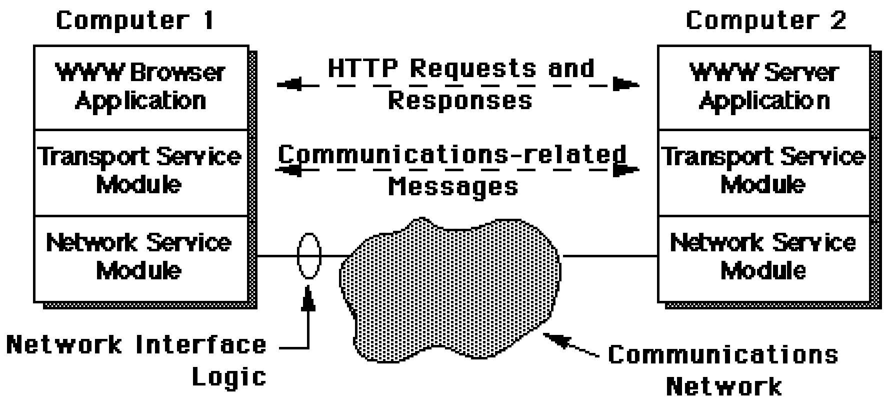
- Protocols
- Protocol architecture

A network ***protocol*** *is a set of rules for communication between computers. Protocols govern format, timing, sequencing, and error control.* Without these rules, the computer cannot make sense of the stream of incoming data bits.

A protocol is used for communication *between entities* in different systems. For two entities to communicate successfully, they must *speak the same language*. What is communicated, how it is communicated, and when it is communicated must conform to some mutually acceptable convention or protocol between the entities involved. The key elements of a protocol are

- **Syntax** Includes such things as data format and signal levels.
- **Semantics** Includes control information for coordination and error handling.
- **Timing** Includes speed matching and sequencing.

What is a protocol, really? It is software that resides either in a computer's memory or in the memory of a transmission device, like a network interface card. When data is ready for transmission, this software is executed. The software prepares data for transmission and sets the transmission in motion. At the receiving end, the software takes the data off the wire and prepares it for the computer by taking off all the information added by the transmitting end. Having introduced the concept of a protocol, we can now introduce the concept of protocol architecture.



A simplified architecture for WWW service

It is clear from the example mentioned earlier that, there must be a high degree of cooperation between the two computers. Instead of implementing the logic for this in a single module, the task is broken up into subtasks, each of which is implemented separately.

As an example, figure suggests the way in which WWW browsing could be implemented. Three modules are used. Tasks 3 and 4 in the preceding list could be performed by a WWW application module. The two modules (WWW application module and WWW server module) on the two systems exchange queries and the information requested by the query. However, rather than requiring the WWW application module to handle the details of actually transferring information and queries, the WWW application modules each rely on a Transport service module. This module is responsible for making sure that the queries and data information are reliably exchanged between systems. Among other things, this module would perform task 2. Now, the nature of the exchange between systems should be independent of the nature of the network that interconnects them. Therefore, rather than building details of the network interface logic into the Transport service module, it makes sense to have a third module, a Network Service module that performs task 1 by interacting with the network.

The WWW application module contains all of the logic that is unique to the WWW application, such as transmitting website address, relative path to the document page and any other necessary information required to uniquely identify a web page which user wants to see. Also WWW application module should have logic to transfer back to the user the requested web page. There is a need to transmit these queries and web pages reliably. However, the same sorts of reliability requirements are relevant to a variety of applications (e.g., electronic mail, document transfer). Therefore, a separate transport service module that can be used by a variety of applications meets these requirements. This module is concerned with assuring that the two computer systems are active and ready for data transfer and for keeping track of the data that are being exchanged to assure delivery. However, these tasks are independent of the type of network that is being used. Therefore, the logic for actually dealing with the network is separated out into a separate Network service module. That way, if the network to be used is changed, only the network service module is affected.

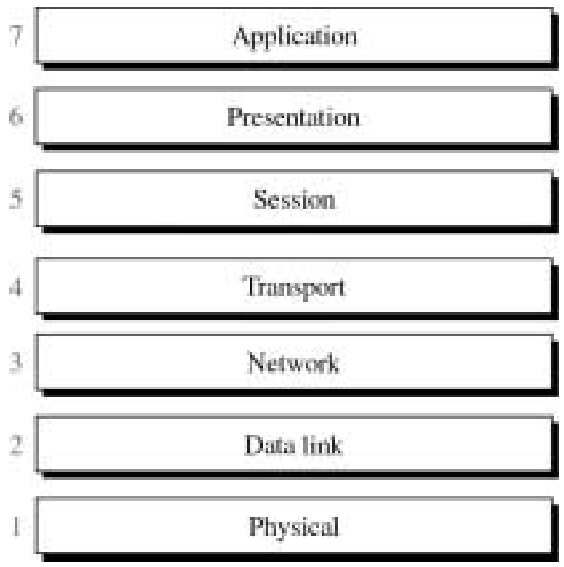
Thus, instead of a single module for performing communications, there is a structured set of modules that implements the communications function. That structure is referred to as *protocol architecture*. Each module in a layer is defined by a protocol and a set of protocols which work together are termed as ***protocol stack***. The terms protocol architecture and protocol stack are used interchangeably.

Two protocol architectures have served as the basis for the development of interoperable communications standards: the TCP/IP protocol suite and the OSI reference model. *TCP/IP is the most widely used interoperable architecture and OSI has become the standard model for classifying communication functions.* Hence a brief introduction to both of them is given below.

1.6 The OSI Protocol Architecture

The *Open System Interconnection (OSI)* model includes a set of protocols that attempt to define and standardize the data communications process. The OSI protocols were defined by the *International Standards Organization (ISO)*, which is a multinational body, dedicated to worldwide agreement on international standards.

The OSI model is not a single definition of how data communications actually takes place in the real world. Numerous protocols may exist at each layer. The OSI model states how the process should be divided and what protocols should be used at each layer. If a network vendor implements one of the protocols at each layer, its network components should work with other vendors' offerings.

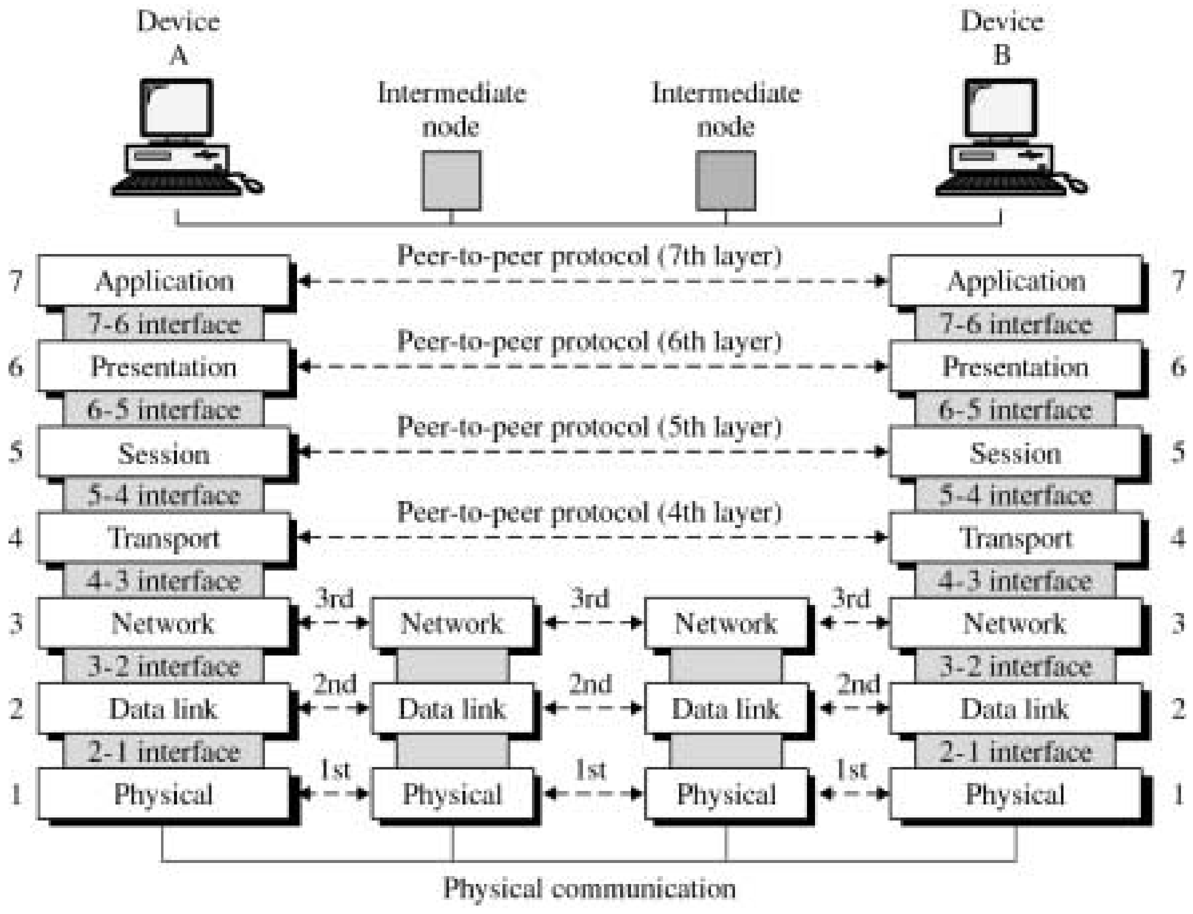


OSI protocols stack.

The OSI model is layered framework for the design of network systems that allows communication between all types of computer systems. It consists of seven separate but related layers, each of which defines a segment of the process of moving information across a network. Understanding the fundamentals of the OSI model provides a solid basis for exploring data communication.

1.6.1 Layered Architecture

The OSI model is composed of seven ordered layers: Physical (layer 1), Data ink (layer 2), Network (layer 3), Transport (layer 4), Session (layer5), Presentation (layer6), and Application (layer 7). The figure shows the layers involved when a message is sent from Device A to Device B. As the message travels from A to B, it may pass through many intermediate nodes. These intermediate nodes usually involve only the first three layers.



Within a single computer system, each layer calls upon the services of the layer just below it. For example, Network layer (layer 3) uses the services provided by the layer 2, which is Data Link, and it provides services to Layer 4 – Transport layer.

However *between computer systems*, layer ***x*** on one system communicates only with layer ***x*** on other system. The processes on each system that communicate at a given layer are called *peer-to-peer processes*. Communication between machines is therefore a peer-to-peer processes communicating to each other using protocols appropriate to the given layer.

1.6.2 Peer-to-Peer Processes

At the physical layer, communication is direct: Device A send a stream of bits to Device B. At the higher layers, however, communication must move down through the layers on Device A, over to the device B, and then back up through the layers. Each layer in the sending device adds its own information to the message it receives from the layer just above it and passes the whole package to the layer just below it.

Headers are added to the data at layers 6,5,4,3 and 2. Trailers are usually added only at layers 2.

At layer 1 the entire package is converted to a form that can be transferred to the receiving device. At the receiving system, the message is unwrapped layer by layer, with each process receiving and removing the data meant for it. For example, layer 2 removes data meant for it, and then passes the rest to the layer 3. Layer 3 then removes the data meant for it and passes the rest to layer 4, and so on.

1.6.3 Interfaces between Layers

The passing of data and network information down through the layers of the sending device and back up through the layers of the receiving device is made possible by an *interface* between each pair of adjacent layers. Each interface defines what information and services a layer must provide for the layer above it, so that the specific implementation of its functions can be modified or replaced without requiring changes to the surrounding layers.

1.6.4 Layer Organisation

The seven layers can be thought of as belonging to three subgroups. Layers 1, 2 and 3 – Physical, Data Link, and Network - are the network support layers; they deal with the physical aspects of moving data from one device to another device. Layers 5, 6 and 7 - Session, Presentation, Application - can be thought of user support layers; they allow interoperability among unrelated software systems. Layer 4, the Transport layer, links the two subgroups and ensures that what the lower layers have transmitted is in a form what the upper layers can use. The upper OSI layers are almost always implemented in software; lower layers are a combination of hardware and software; except for the physical layer, which is mostly hardware.

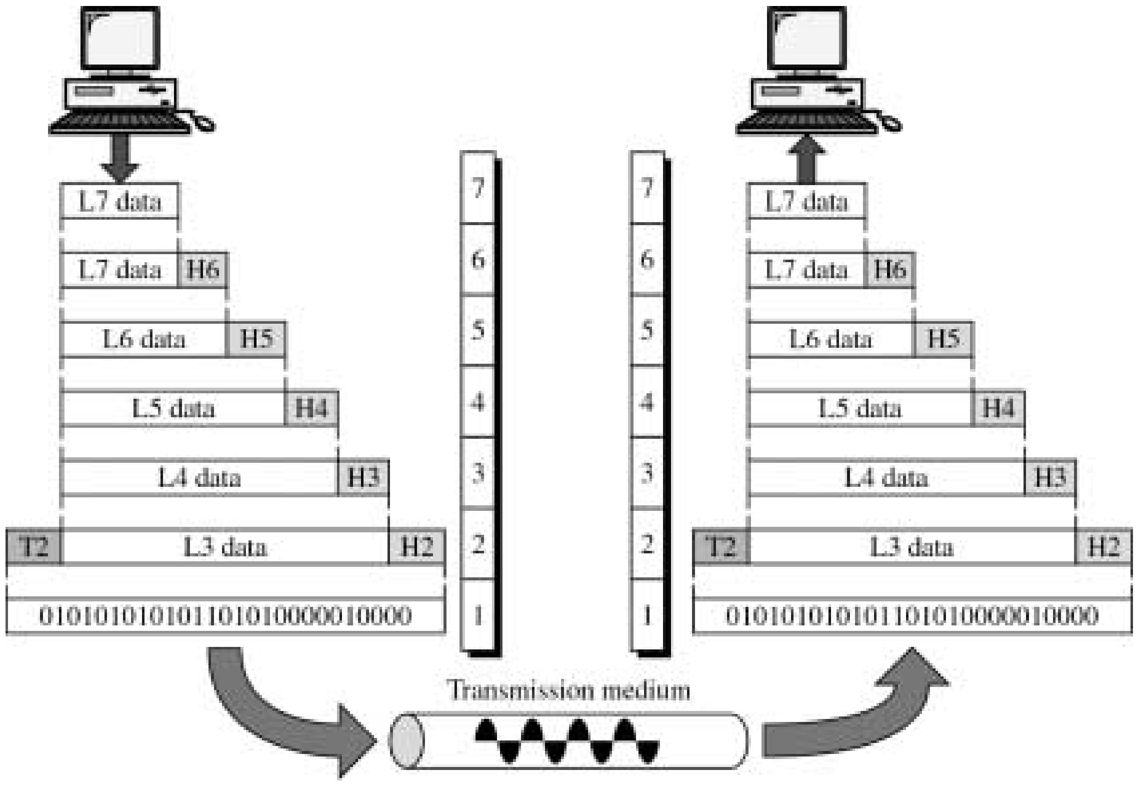


Figure gives an overall view of the OSI layers, L7 data means the data unit at layer 7, L6 data means the data unit at layer 6, and so on. The process starts at layer 7 (Application layer), then moves from layer to layer in descending order till it reaches layer 1. At each layer (except layer 7 and 1), a header is added to the data unit received from the upper layer. At the layer 2, a trailer is added as well. When the formatted data unit passes through the physical layer, it is changed into electromagnetic signal and transported along a physical link.

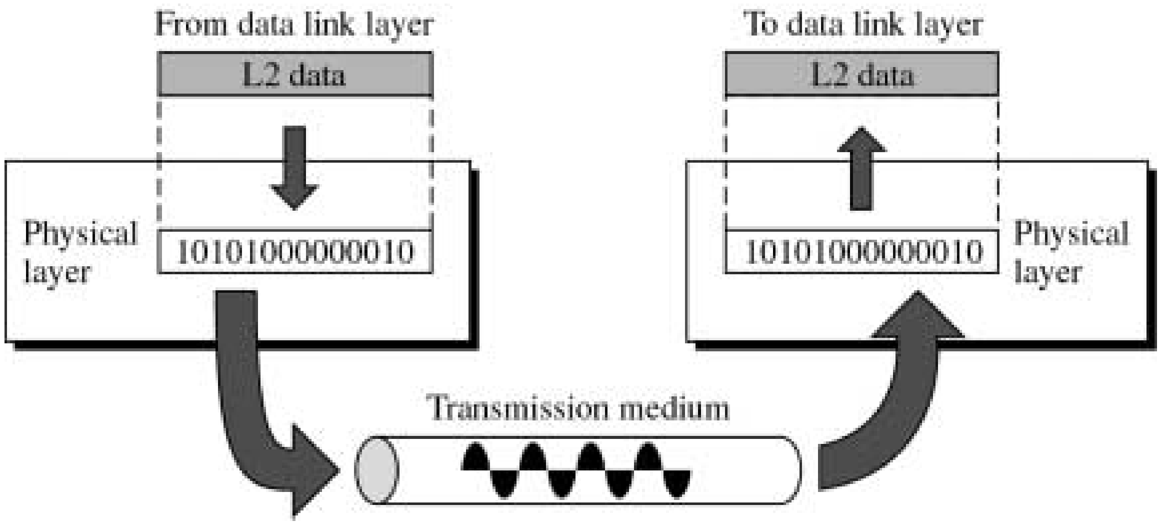
Upon reaching its destination, the signal passes into layer 1 and is transformed back into digital form. The data units then move back up through the OSI layers. As each block of data reaches the next higher layer, the header and trailer attached to it at the corresponding layer at the sending device are removed, and actions appropriate to that layer are taken. By the time it reaches layer 7, the message is again in a form appropriate to the application and is made available to the recipient programme.

1.6.5 Layers in the OSI Model

The Physical Layer

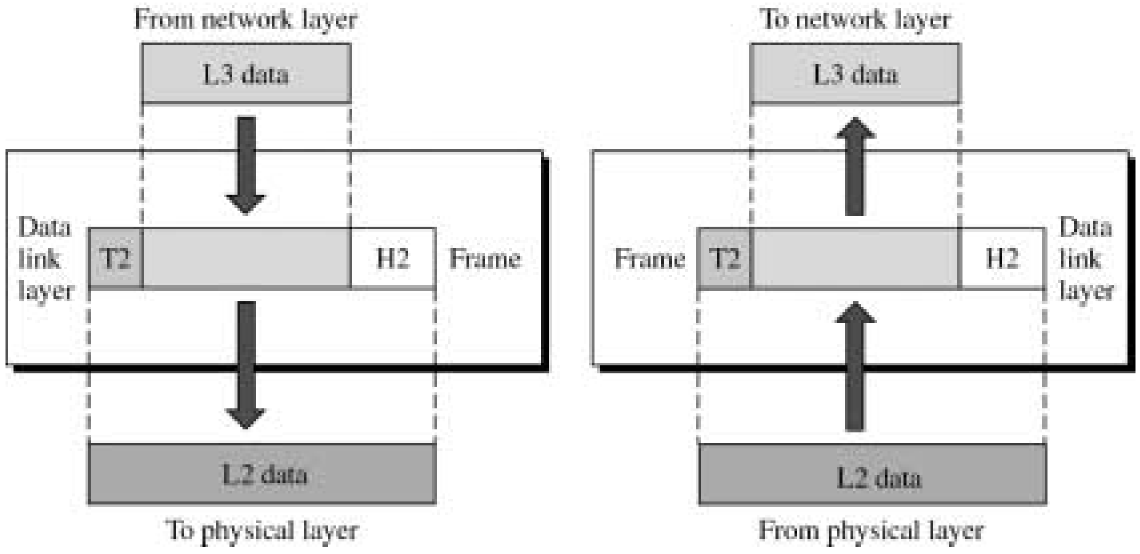
The physical layer is the lowest layer of the OSI model and deals with the "mechanical, electrical, functional, and procedural means" required for transmission of data, according to the OSI definition. This is really the wiring or other transmission form. When the OSI model was being developed, a lot of concern dealt with the lower two layers, because they are, in most cases, inseparable. The real world treats the data link layer and the physical layer as one combined layer, but the formal OSI definition stipulates different purposes for each.

The figure shows the position of the physical layer with respect to the transmission media and the data link layer.



The Data Link Layer

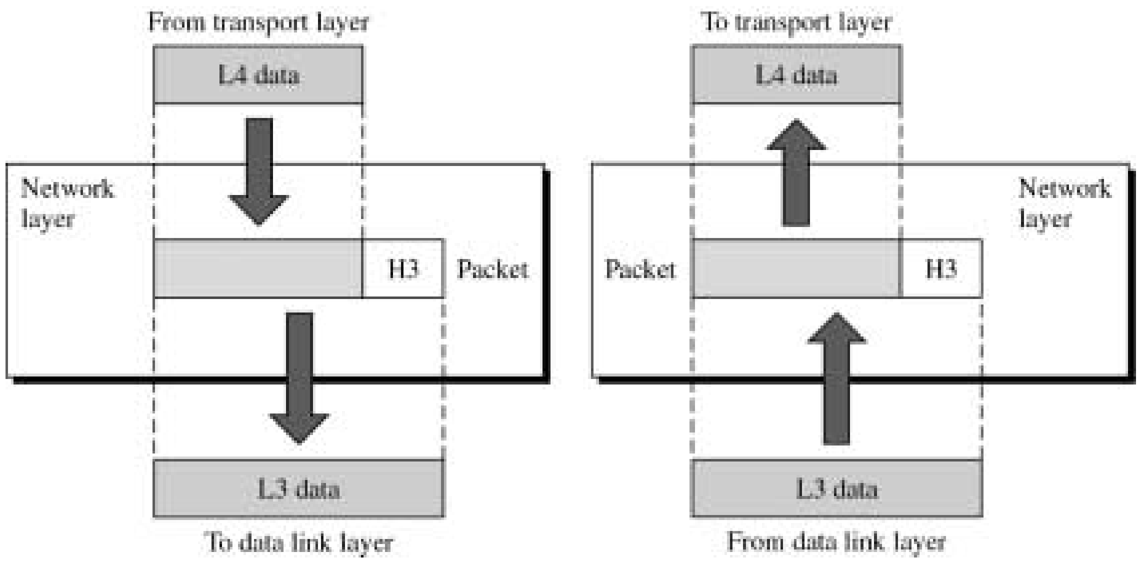
The data link layer, according to the OSI reference paper, *provides for the control of the physical layer, and detects and possibly corrects errors that can occur*. In practicality, the data link layer is responsible for correcting transmission errors induced during transmission (as opposed to errors in the application data itself, which are handled in the transport layer). This is achieved by the data link layer by performing the tasks like framing, flow control, error control, access control, and physical addressing. Framing is basically dividing the stream of bits received from the network layer into manageable data units called frames. Each frame to reach proper destination data link layer adds the physical address of the source and destination system into the header of the frame. If the sending system is transmitting the data at the rate at which it can not be absorbed by the receiving system, then there would be loss of data. For this reason data link layer imposes the restriction on the rate at which the data can be transmitted by the sending system through the flow control mechanism. Error checking is required as the bits transmitted over the physical medium may be corrupted. Lastly the access control is required in cases where more than one system is trying to send the data over the common physical medium.



The figure shows the relationship between the data link layer and its upper layer (network) and bottom layer (physical). It can be noted that the data link layer adds header and trailer at the transmitting end and removes the same at the receiving end. Header is used for the purpose of storing control information, where as trailer is used to signify the end of the frame, so that it becomes easy for the receiving system to build the frames from the raw bits received from the physical layer.

The Network Layer

The **network layer** is responsible for the source-to-destination delivery of a *packet* possibly across multiple networks (links). Whereas data link layer oversees the delivery of the packet between two systems *on the same network (link),* the network layer ensures that each packet gets from point of origin to its final destination. If the two systems are connected to the same link then there would be no requirement for the network layer. However if the two systems are attached to *different networks* with connecting devices (known as routers) between the networks, then this layer plays a crucial role in getting the packet from source to destination.



The figure shows the relationship between the network layer with its upper and lower layers. The header H3 has vital information for the routing of packets from the source to destination.

The *Network layer* provides for the transfer of data in the form of packets *across the communication networks*. It establishes, maintains, and terminates logical and physical connections across multiple interconnected networks. A key aspect of this transfer is the *routing of packets* from the source to the destination machine typically traversing a number of transmission links and network nodes where routing is carried out. Routing is the process by which a path is selected out of many available paths to the destination so that data packet reaches the destination fast, efficiently, reliably as required. This function makes the network most complex layer in the reference model.

The Transport Layer

The transport layer is designed to provide the “transparent transfer of data from a source end open system to a destination end open system," according to the OSI Reference Model. The transport layer establishes, maintains, and terminates communications links between two machines.

The *Transport layer* ensures data is successfully sent and received between two end systems. If data is sent incorrectly, this layer has the responsibility to ask for retransmission of the data. Also it ensures data are passed onto the upper layers in the same order in which they were sent. Specifically, it provides a reliable, network-independent message-interchange service to the top three application-oriented layers. This layer acts as an interface between the bottom and top three layers. By providing the *session layer* with a reliable message transfer service, it hides the detailed operation of the underlying network from the session layer.

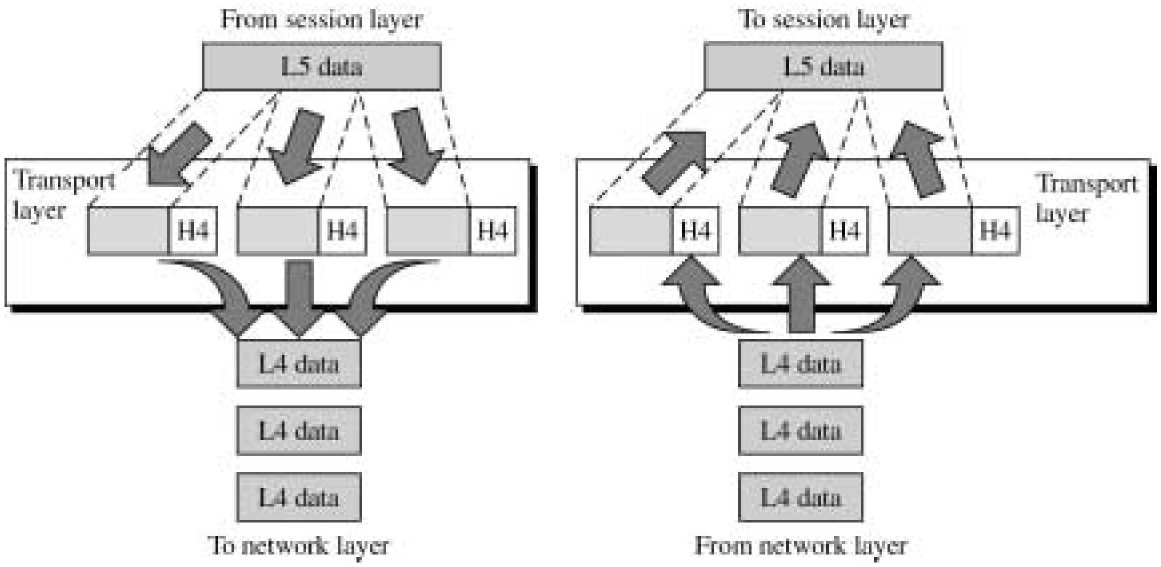
Some of the important task that are performed by the transport layer in order to meet the above requirement are

Service-point addressing: Computers often run several programs at the same time. For this reason, source-to-destination delivery means delivery not only from one computer to the next but also from a specific process (a running programme) to a specific process at the receiving end. The transport layer header must therefore include a type of address called *service-point address* (or port address). The network layer gets *each packet* to the correct computer; the transport layer gets the *entire message* to the correct *process* on that computer.

Segmentation and Reassembly: If the message sent by the application programme at the transmitting end is huge, there would be problem in moving that from the transmitting system to the receiving system as many of the physical network underneath would impose restriction on the maximum size of the data that can be transferred at one time. In order to over-come this restriction, transport layer splits the incoming message into segments (each of transferable size) at the transmitting system and reassembles them at the receiving end.

Apart from the above other tasks performed by the transport layer are C*onnection control, Flow control, Error control.*

The Figure shows the relationship between the transport layer and the session layer at the top and network layer at the bottom.



It can be observed that at the transmitting system, the incoming message data L5 is split into segments and header is added to each segment and passed on to the network layer as L4 data. At the receiving system, the incoming L4 data from the network layer are reassembled after removing the header H4 and passed on to the session layer as L5 data.

The Session Layer

The session layer organizes and synchronizes the exchange of data between application processes. It works with the application layer to provide simple data sets called *synchronization points* that let an application know how the transmission and reception of data are progressing. In simplified terms, the session layer can be thought of as a timing and flow control layer.

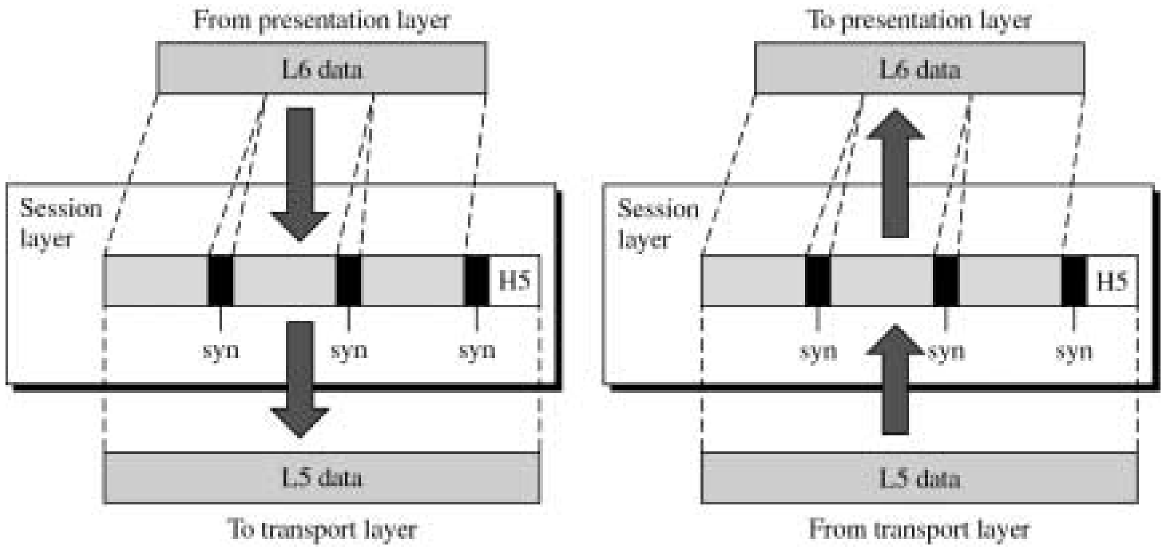
The session layer is the network *dialog controller*. It establishes, maintains, and synchronises the interaction between communicating systems. Specifically its responsibilities include

Dialog control: This allows two systems to enter into a dialog. It allows the communication between two processes to take place either in half-duplex (one way at a time) or full duplex (both ways simultaneously).

Synchronisation: The session layer allows a process to add checkpoints (synchronisation points) into a stream of data. For example, if a system is sending a file of 2,000 pages, it is advisable to insert checkpoints after every 100 pages to ensure that each 100 page is received and acknowledged independently. The reception of acknowledgement for these checkpoints ensures to the sending system that data up to the corresponding checkpoint is received properly by the receiving end system.

An example where session layer plays a crucial role is the file transfer application when used to transfer huge files (downloading of huge files). When huge files are transferred across the network, if the network connection speed is not high, it would take very long time for transfer. There is every chance that network connection would break during this transfer and the user is required to start afresh to do the file transfer, and there is no guarantee that the same would not repeat, making it sometime practically impossible to download very huge files. If the checkpoints are used by the file transfer application then, this problem can be sorted out easily. Suppose the connection fails during the 1034th page, then after reconnecting we can proceed from the last checkpoint, i.e. from page 1001. During retransmission time the only pages that are *resent for the second time* are 1001 to 1034 as pages till 1000 are received and *acknowledged.* If checkpoints were not there, we have to start from page 1 every time we try to reconnect.

The Figure illustrates the relationship between session layer and the presentation layer at the top and transport layer at the bottom.



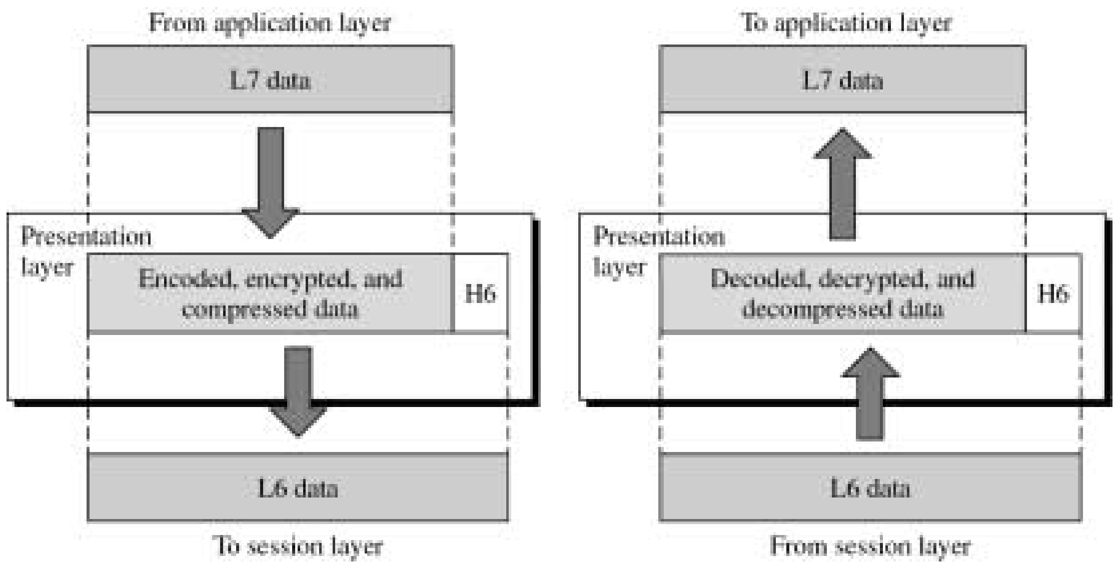
The Presentation Layer

The presentation layer's task is to isolate the lower layers from the application's data format. At the transmitting end, it converts the data from the application into a common format, often called the *canonical representation.* The presentation layer processes machine-dependent data from the application layer into a machine-independent format for the lower layers.

The presentation layer is where file formats and even character formats (ASCII and EBCDIC, for example) are lost. The conversion from the application data format takes place through a "common network programming language" (as it is called in the OSI Reference Model documents) that has a structured format.

Also the presentation layer is responsible for the encryption and compression of the data, so that transaction over the network is secure and efficient.

At the receiving end the presentation layer does the reverse for incoming data. It is converted from the common format into application-specific formats, based on the type of application the machine has instructions for. If the data comes in without reformatting instructions, the information might not be assembled in the correct manner for the user's application.

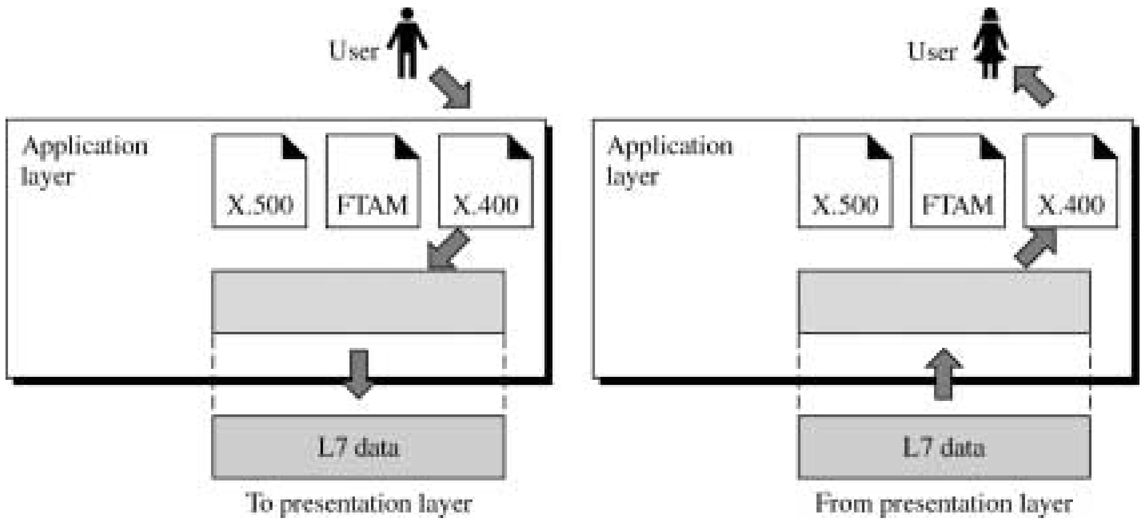


The Figure shows the relationship between the presentation layer and application layer at the top and session layer at the bottom. It could be observed that at the sending system the presentation layer takes the data from the application layer L7, does the necessary processing such encoding, encryption, compression and adds and header H6 and then passes it on to the session layer. At the receiving system, the reverse process happens.

The presentation layer gets the data from the session layer, removes the header H6 from it, and based on the instruction present in the header H6, does the decoding, decryption or decompression of the data and then passes the processed data to the appropriate application programme at the application layer. Note that the header H6 contains vital information to the application layer at the receiving end about how it should process the received data. This is true for every header at each layer.

The Application Layer

The application layer is the end-user interface to the OSI system. It is where the applications, such as electronic mail, web browsers like Internet Explorer or Netscape Navigator, USENET newsreaders, or database display modules, reside. The application layer's task is to display received information and send the user's new data to the lower layers. In distributed applications, such as client/server systems, the application layer is where the client application resides. It communicates through the lower layers to the server.



The Figure shows the relationship of the application layer to the user and the presentation layer. Of the many applications available the figure shows only three: X.400 (OSI messaging services), X.500 (directory services), and FTAM (File Transfer and Access Management). The user in the example uses X.400 to send an e-mail message. Note that no header or trailer is added at this layer.

Unfortunately, protocols in the real world do not conform precisely to these neat definitions. Some network products and architectures combine layers. Others leave layers out. Still others break the layers apart. But no matter how they do it, all working network products achieve the same result - getting data from here to there. This OSI model serves as the best reference model, using which one can study the other.

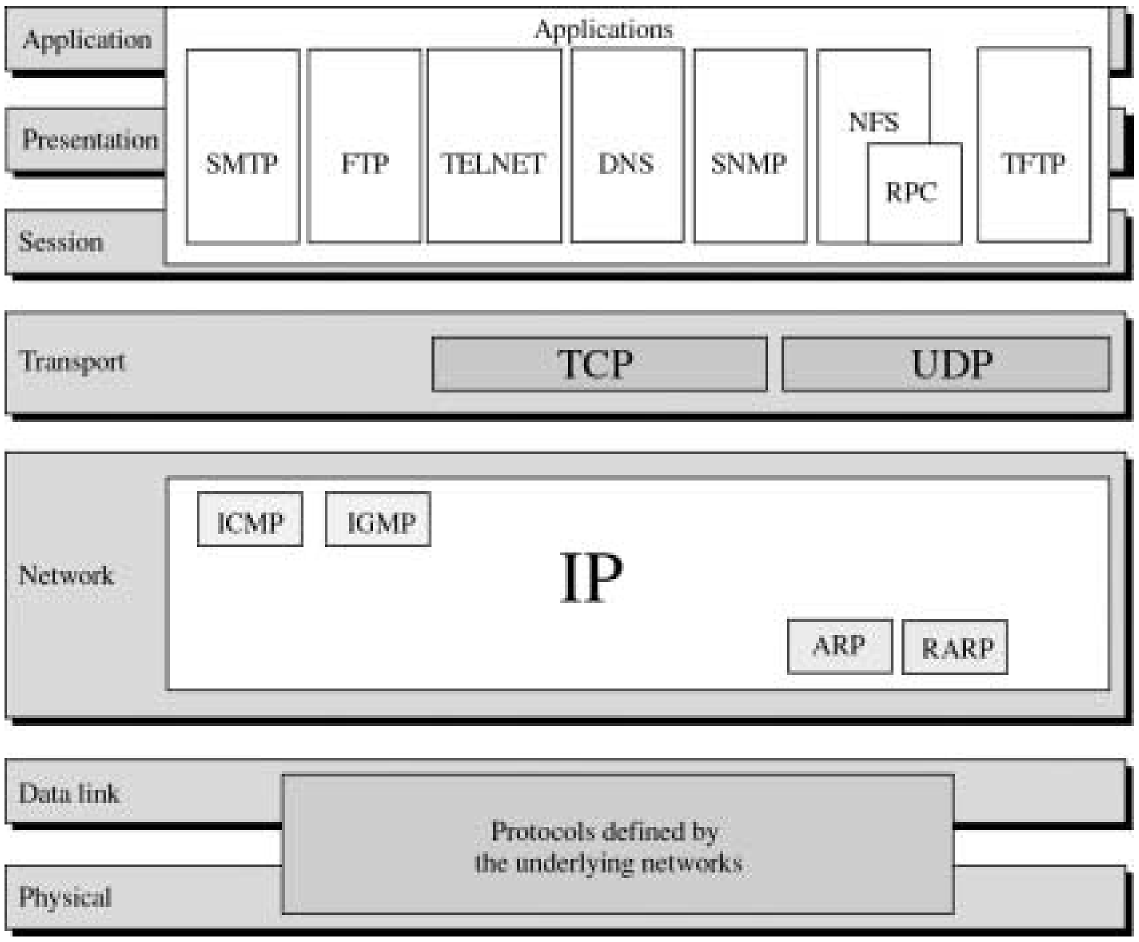
1.7 The TCP/IP Protocol Architecture

As noted earlier TCP/IP is a result of protocol research and development work conducted on the experimental network, ARPANET, funded by the Defence Advanced Research Projects Agency (DARPA) in the U.S, and is generally referred to as the TC/IP protocol suite. This protocol suite consists of a large collection of protocols that have been issued as Internet standards by the Internet Activities Board (IAB).

TCP/IP was developed prior to the OSI model. Therefore layers in the TCP/IP protocol stack do not match exactly with those in the OSI model. The TCP/IP model is made up of five layers.

- Physical layer
- Data Link layer
- Network layer (IP)
- Transport layer (TCP)
- Application layer

The first four layers provide physical standards, network interface, internetworking and transport functions that correspond to the first four layers of the OSI model. The three top layers of the OSI model, however, are represented in the TCP/IP by a single layer called *application layer*. The figure illustrates how TCP/IP protocol stack fits itself in the OSI reference model.



Physical and Data Link layer:

At the physical and data link layers, TCP/IP does not define any specific protocol. It supports all of the standard and proprietary protocols. A network in a TCP/IP inter-network can be a Local Area Network (LAN), or a Metropolitan Area Network (MAN) or a Wide Area Network (WAN).

Network layer:

The data link layer is concerned with access to and routing data across a network for *two end systems attached to the same network*. In those cases where two devices are attached to different networks, procedures are needed to allow data to traverse multiple interconnected networks. This is the function of the network layer. The ***Internet Protocol***

(IP) is used at this layer to provide the routing function across multiple networks. This protocol is implemented not only in the end systems but also in routers. A router is a processor that connects two or more *networks* and whose primary function is to relay data from one network to the other on its route from the source to the destination end system.

The Internet Protocol (IP) is the transmission mechanism used by the TCP/IP protocols. It is an unreliable and connectionless datagram protocol – i.e. it provides a best-effort delivery service. The term best-effort means that IP provides no error checking or tracking. IP assumes the unreliability of the underlying layers and does its best to get the get a transmission through to its destination, but with no guarantees. IP transports data in packets called *datagrams* each of which is transported separately. Datagram can travel through different routes and can arrive out of sequence or be duplicated. IP does not keep track of routes and has no facility to reorganise datagrams once they arrive at their destination. *These limitations should not be considered as the weakness of the protocol stack. It is intentional, to get the maximum efficiency*. The purpose at this layer is to provide the bare-bone transmission functions that free the user to add only those facilities necessary for a given application. It is assumed that the facilities that are missing for a particular application is taken care of by the either upper layer (transport layer) or the application itself.

At this network layer TCP/IP supports the *Internetworking Protocol (IP)*. IP, in turn, contains four supporting protocols; ARP, RARP, ICMP and IGMP. Each of these is described in detail in later chapters.

Transport layer:

The IP is a *host-to-host* protocol. It delivers the packet from source device to destination device. However it does not provide any mechanism to identify the source process (programme) and the destination process to which the packet is meant. One of the essential features provided by the Transport layer protocols are exactly this - supporting delivery of messages from a *process to another process*. At this layer TCP/IP suite defines two protocols – **UDP** (*User Datagram Protocol*) and **TCP** (*Transmission Control Protocol*).

Transmission Control Protocol (TCP): Most of the applications that are exchanging data, there is usually a requirement that the data be exchanged reliably. That is, we would like to be assured that all of the data arrive at the destination application and that the data arrive in the same order in which they were sent. And also the mechanisms for providing reliability should be essentially independent of the nature of the applications. The ***TCP*** is the most commonly used protocol to provide this functionality. Apart from these this also provides the mechanism by which the data can be delivered to the destination process within the destination system.

User Datagram Protocol (UDP): This is the simpler of the two standard TCP/IP transport protocols. This is a process-to-process protocols that adds only port addresses and checksum error control. It provides the mechanism (like TCP) by which the datagram can be delivered to the destination process within the destination system. It is assumed that any other feature required regarding transmission would be by the application programme itself.

Application layer:

Finally, the **application layer** contains the logic needed to support the various user applications. For each different type of application, such as file transfer, separate module is needed that is particular to that application. The application layer in TCP/IP is equivalent to the combined session, presentation, and application layers in the OSI model.

1.8 General Comparison between OSI and TCP/IP

In practice, OSI is a **de-jure** (according to law) standard and, TCP/IP is a **de-facto** (in reality) standard. The focus in the TCP/IP world is on agreeing on a protocol standard which can be made to work in diverse heterogeneous networks. The focus in the OSI world has always been more on the standard than the implementation of the standard.

The OSI reference model was devised *before* the protocols were implemented. This ordering means that the model was not biased toward one particular set of protocols, which made it quite general. The down side of this generalization was standard became quite complicated to implement and handle. Because of the complications OSI never gained the popularity though it was implemented in several organizations. Nevertheless, since as a standard it was a much-generalized one, it has been used as a reference model against which one can make comparison.

With the TCP/IP the reverse was true: the protocols came first, and the model was just a description of the existing protocols. There was no problem with the protocols fitting the model, but it is hardly possible to be use to describe other models.

The TCP/IP protocol suite has always had an applied, *get the job done* orientation. Over the years it has handled most challenges by growing to meet the needs, and it is now the de-facto standard for internetworking for several reasons, including:

1) It s relatively simple and robust compared to alternatives such as OSI;
2) It is available on virtually every hardware and operating system platform, often free;
3) It is the protocol suite on which the Internet depends.

1.9 Who’s who in Standards?

OSI protocol is developed by ISO - International Standards Organisations. ISO is a voluntary nontreaty organisation founded in 1946. Its members are the national standards organisations of the 89 member countries.

For Internet (TCP/IP), there is a central committee named *Internet Architecture Board – IAB.* Each of the members of IAB headed a task force on some issue of importance. When a standard was needed (for e.g., a new routing algorithm), the IAB members would come out with one, announce the change and communicate the same through the series of technical reports called R*equest For Comments* – popularly known as **RFC**s. There are nearly 2000 RFCs exist as of now.

The growth of the Internet was such that, a single governing body couldn’t handle it. Two new subsidiaries of IAB were created – ***IRTF*** *(Internet Research Task Force) and* ***IETF*** *(Internet Engineering Task Force).* IRTF mainly concentrated on the long term research while IETF focused on short term Engineering issues.

Later, the *Internet Society* was created, populated by people interested in the Internet. It is governed by the elected trustees who appoint IAB members.

There is a recognised process by which standards are done. To become a *Proposed Standard,* the basic idea must be completely explained in an RFC and have sufficient interest in the community to warrant consideration. To advance to the *Draft Standard* stage, there must be a working implementation that has been thoroughly tested by at least two independent sites for 4 months. If the IAB is convinced that the idea is sound and the software works, it can declare the RFC to be an *Internet Standard*.

1.10 Summary:

There are two protocol architecture models; OSI standardised by the ISO committee. This model is a very good reference model though it is not the widely used one. Another model is the TCP/IP model which is the outcome of longstanding research activities at the universities and research laboratories. TCP/IP model is the most widely used architecture and is the technology behind the today's Internet. OSI has seven layers where as TCP/IP defines only five. The seven layers of the OSI are the physical, data link, network, transport, session, presentation and application layers. TCP/IP does not have session and presentation layers but if their functionalities are required by any application then it is implemented in that particular application. There are various protocols specified at each layers and their interface is well defined so that if required one can just change one implementation of a particular protocol with the other implementation.

Outline of the rest of the book

Focus in the rest of the book is towards how in practice computer network works. Since TCP/IP is the de-facto protocol stack in the Internet domain, it is used to explain the functioning of various protocol layers. TCP/IP does not define protocols for physical and data-link layer, as it supports all of the standard and proprietary protocols at this layer. Hence we begin with network layer, then cover transport layer and end with application layer.

UNIT 2

Network Layer

Contents:

2.1 Introduction

The network layer is the most important layer in the TCP/IP model, as it is here where the responsibilities of moving the data from the source to the destination host across several physical networks are handled. The transfer of data, as for as the upper layers are concerned, should be transparent, whether the destination host resides in the same physical network or across several physical networks, whether the intermediary physical networks uses the same technology and protocols or not.

Another important service this layer should provide is the virtual network concept to the upper layers. As for as the, upper layers are concerned, there is only one network, and each host is recognized by a unique ID (which is implemented as IP address).

The following protocols at the network layer are covered in this Unit.

- IP - Internet Protocol
- ARP - Address Resolution Protocol
- RARP - Reverse Address Resolution Protocol
- ICMP - Internet Control Message Protocol

Before going to understand these protocols, one needs to know various types of addressing involved in the TCP/IP model. A brief section about the same is given in the beginning.

IP is not only the core of this layer but also most important protocol in the TCP/IP model. IP is covered in detail in the next section. Each system on the Internet is recognized by a unique IP address. Though it is just a 4 byte number, it stores many important information about the network, about the type of service. Hence one needs to understand it in detail. One of the important decisions an IP module has to take is where to forward the incoming datagram so that it reaches its destination host, in other words – *routing*. A good routing mechanism is vital for the Internet to run smoothly. It has to be efficient as the datagram traffic is huge, and it has to be designed such that, datagrams wont wander in the Internet without reaching their destination. A brief section about the same is given.

One would be curious to know what all information is stored and passed in an IP datagram. There is a predefined format in which a datagram has to filled, so that all the systems which receive it, understand its content without any ambiguity. This format is covered in the section, IP datagram. Each datagram has a header followed by the data sent by the upper layer. The header is covered in detail. At the end of the section 2.4 one would be having a good understanding about the IP and going through the rest of the protocols should not be problem.

Though IP is core of the network layer, for the network layer services as mentioned earlier to be implemented, there would be a need to send and receive the IP datagram itself! Definitely it is not a good design to have these modules, which sends and receives the IP datagram within the IP itself. Hence these requirements are implemented as separate protocols. They are ARP, RARP and ICMP. Each is covered in later section without going into the minute details of the same, as the focus should be on IP.

2.2 Objectives:

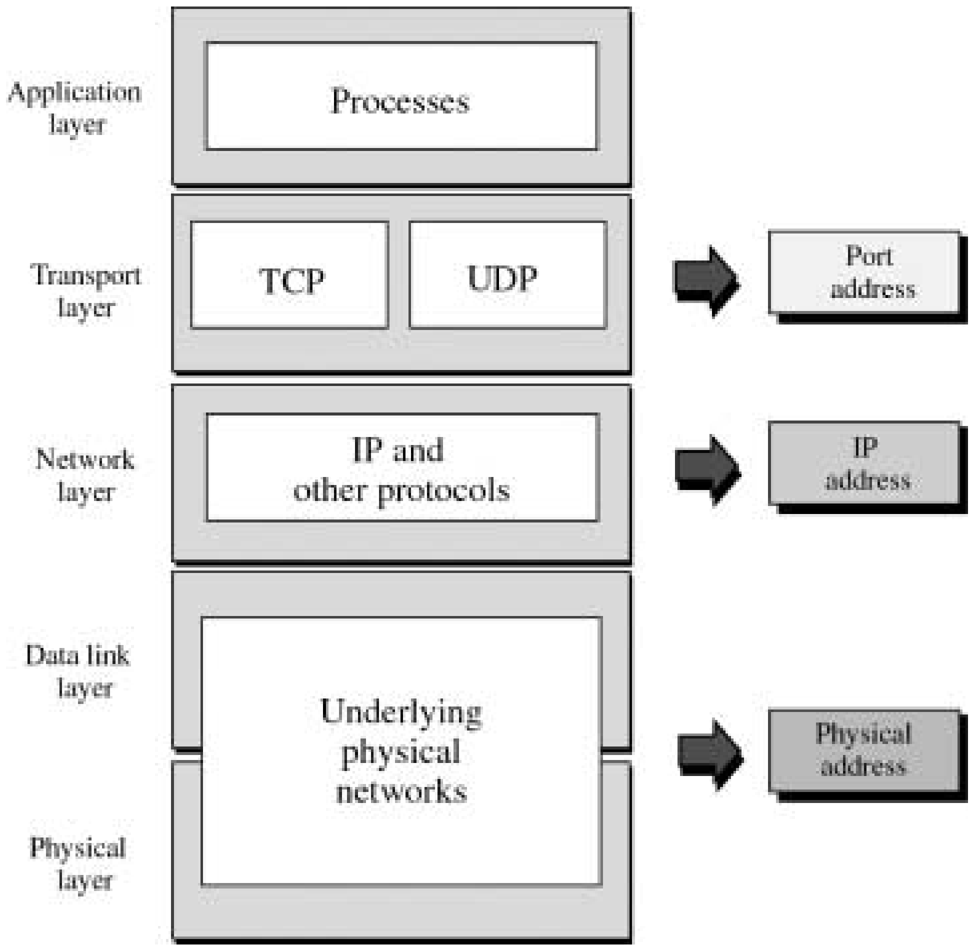
In this Unit you would

- Learn about how the addressing is designed in the TCP/IP model.
- Learn about the protocols at the Network Layer.
- Learn IP (Internet Protocol), which *is the core of the TCP/IP* in detail.
- Learn ARP (Address Resolution Protocol), which is used for address resolution.
- Learn RARP (Reverse Address Resolution Protocol) which is used for finding ones own IP address
- Learn ICMP (Internet Control Message Protocol), which is used for passing the error reports messages and making queries.

2.3 Addressing in TCP/IP:

TCP/IP model has three different levels of addressing:

- Physical (Link) address
- Internet work (IP) address
- Transport or Port address



Physical address:

This is the address of a node at the datalink layer, as defined by the LAN or WAN. It is included in the frame sent by the datalink layer. This address determines the host system on a particular network. The size and format of the physical address is not defined by the TCP/IP and it depends on the kind of the network. For example, Ethernet

LAN uses 6 byte (48 bit) physical address, which is imprinted into the Network Interface Card.

Physical addresses can be either **unicast (**Single recipient), **multicast** (a group of recipients) or **broadcast** (received by all in the network). However not all networks supports these. Ethernet – one of the popular LAN – supports all of these.

Internet address:

Internet addresses are necessary for universal communication services that are independent of underlying physical network. Physical networks have different addressing format depending upon the network technology used. Also the addressing doest not have any component using which one can identify the network to which it is connected; which is essential for the routing purpose.

The Internet addresses are designed for this purpose. An Internet address is currently a 32 bit (4 byte) address which can uniquely identify a host connected to the Internet. No two hosts *on the Internet* can have the same IP address. Also Internet addresses are defined such a way that given an IP address one can easily identify the network to which it is connected so that routing becomes easy.

The Internet addresses support unicast, multicast and broadcast addressing.

Port address:

The IP address and the physical address identify the source and the destination systems. They don't identify the process (a running programme on the computer) on these systems to which the data actually corresponds to. The final objective of the Internet communication is providing a communication link between two processes running on two different systems. For example, data sent by the FTP (File Transfer Protocol) client process from system A should reach the FTP server process at the system B. It should not reach the MAIL server process running on the system B. So it is not only crucial to identify the end systems to which the data meant, but the end processes are also to be identified. To achieve this, different processes are labeled uniquely. In TCP/IP this labeling is called as port address. A port address is 16 bits long (2 byte).

2.4 Internet Protocol

The IP module is central to the success of Internet technology. The Internet Protocol or *IP builds a single logical network from multiple physical networks*. In other words this layer shields the higher levels from the typical network architecture below it. Thus Internet Protocol (IP) is the most important protocol. The higher layers see only a one *virtual* or *logical network* and they need not be aware of where exactly the physical network resides, its architecture and so on. As for as, the higher layers are concerned, each host on the Internet is a system with a unique IP address. This interconnection of physical networks into a single virtual or logical network is the source of the name: *Internet*. A set of interconnected physical networks that limit the range of an IP packet is called an *Internet*. The term *Internet* (with capital I) is used to describe the global network of interconnected networks that runs on TCP/IP suite of protocol to which anyone can connect his network or host as long as he follows the TCP/IP standard and follows the guidelines of Internet authority.

IP hides the underlying network hardware from the network applications. If some one invents a new physical network, all that they have to do so that the physical network can become part of the Internet is provide a proper interface to the IP as defined by the Internet Protocol standard. The rest of the layers above IP (including IP) remain same. Thus, the network applications remain intact and are not vulnerable to changes in hardware technology.

Apart from providing a *virtual network* service to the upper layers, it is the IP layer which transports the datagram packet from source to destination system across several physical networks. It is the lowest level protocol, which provides the host-to-host datagram delivery mechanism. The layers below this only deliver the datagram within a single physical network.

It is an *unreliable and connectionless datagram protocol* – a best-effort delivery service. The term best-effort means that IP provides no error checking or tracking. IP assumes the unreliability of the underlying layers and does its best to get a transmission through to its destination, but with no guarantees. IP assumes that upper layer protocols will take care of the reliability issues if they need. In fact TCP protocol at the transport layer provides the reliability and connection oriented service to the application layer. The issue of reliability is not covered in the IP layer as it was designed with the efficiency in mind and some of the applications do not need the reliability and for them efficiency is crucial.

IP is also a *connectionless protocol* designed for packet switched network which uses the datagram approach. This means that each datagram is handled independently and each datagram can take an independent route to reach the destination. This means that when a message is split into many datagrams (because of constraint on the size imposed at the datalink layer) and sent by a source each datagram would take independent route to reach destination and eventually they may arrive out of order, some of them could be corrupted or lost. Again it is the responsibility of the higher layer protocols to take of these.

The important concepts of the IP such as addressing, routing and IP datagram are discussed in the following subsections.

2.4.1 Internet (IP) Addressing:

There is a need to identify the host systems on the Internet uniquely irrespective of the underlying network technology used. Also the addressing mechanism used should be such that very minimum logic is required to identify the network to which the host is connected in the Internet so that routing becomes easy and efficient. Routing of the data packets is an important task in the Internet so that the data packets reaches the destination system across multiple networks and since data traffic in the Internet is huge in today's Internet, routing mechanism should be as efficient as possible.

The identifier that is used in the Internet layer (IP) of the TCP/IP is called *Internet Address* or *IP Address*. It is a 32 bit (4 bytes) long implemented in software. This uniquely identifies the host *that is connected to the Internet.* These addresses are granted by the Internet Authority, to ensure each host on the Internet has a unique IP address. If the local network is not connected to the Internet then it is possible to have IP address of ones choice. However there is a subset of addresses available for such purposes and it is recommended to use those addresses only.

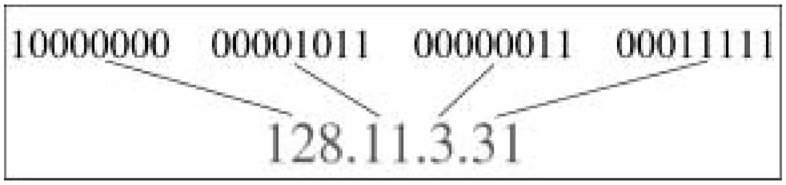
It should be also noted that, though IP addresses uniquely identifies the host on the Internet, it is possible for a host to have multiple IP addresses. In fact a router, which interconnects physical networks, will have many IP addresses depending on the number of physical networks it interconnects.

Each IP address is of 32 bits or 4 bytes length. It has two parts, one, which identifies the network, *netid*, another which identifies the host on that network, *hostid*. These parts vary depending on the class of the address. The first few bits determine the class and the rest are netid and hostid.

The *netid* uniquely identifies the physical network on the Internet and no two physical networks on the Internet can have the same *netid.* The *hostid* uniquely identifies the host on a physical network and no two-host systems can have the same *hostid*. Thus the combination of these two – *netid* and *hostid* – uniquely identifies the host the Internet.

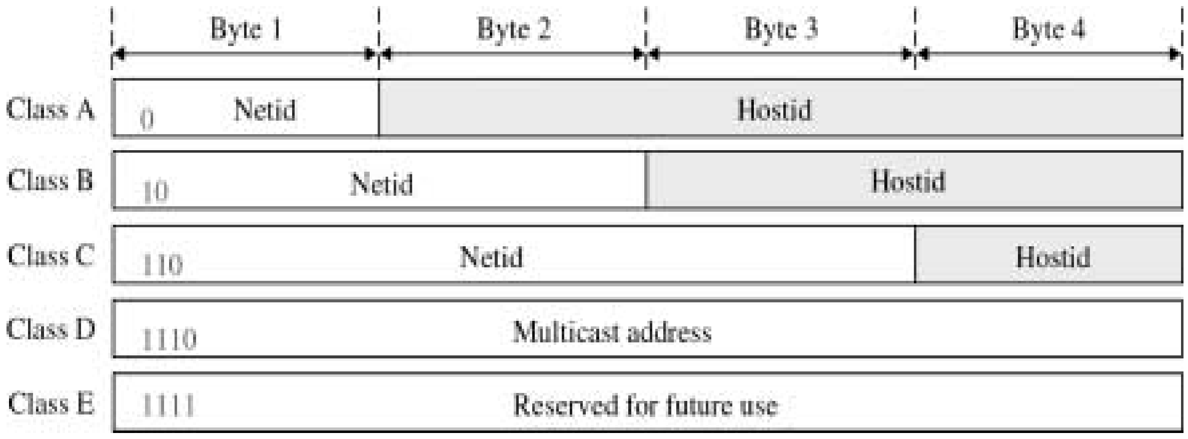
2.4.1.1 Decimal Notation

To make the 32 bit far more compact and easier to read and remember, Internet addresses are usually written in decimal form with decimal points separating the bytes. Figure shows the IP address in the decimal notation.



2.4.1.2 IP address Classes

There are five different IP address classes: A, B, C, D, and E. These are designed to cover the needs of different types of organization. The first few bits of the 32-bit IP address determines the type of address class to which the IP address belongs.



Class A

If the first bit is 0 then the given address is a class A address. Remaining portion of the first byte defines the *netid*. Byte 2 to 4 determines the *hostid* of the system on that physical network. The *netid* has 7 bits which means there can be totally $2^7 = 128$ networks with class A address. Since hostid has remaining 3 bytes (24 bits), each such network can accommodate $2^{24} = 16{,}777{,}216$ hosts. However not all of the addresses can be used as some of them (hostid all 0s and all 1s) are used for special purposes. Class A addresses are designed for organizations that are having huge number of hosts connected to their network. However it is highly improbable that an organization having so many hosts and many of those addresses are wasted in this class.

Class B

If the first two bits of the IP address are 10, then the given address is a Class B address. Here *hostid* are 16 bits (2 bytes) long and rest 14 bits are for the *netid*. That means there are $2^{14} = 16{,}384$ Class B networks and $2^{16} = 65{,}536$ hosts each on such network. As mentioned earlier for Class A, some of them are reserved for special purpose and would be explained later.

Class B address are designed for midsize organization that may have large number of host systems attached to the network.

Class C

If the first three bits of the IP address are 110, then the given address is a Class C address. The next 21 bits define the *netid* and the remaining 8 bits determine the hostid on that network. Hence there are 2^{21} = 2,097,152 Class C networks and 2^8 = 256 hosts each on such network. As mentioned earlier, some of them are reserved for special purpose.

Class C addresses are designed for small organization that will have few number s of hosts computer system connected to their networks. It should be noted that IP address can support large number of such organization as the *netid* has 21 bits.

Class D

Class D is a special kind of address meant for multicasting. This does not have the notion of *netid* or *hostid*. The first 4 bits (1110) define the class here. The remaining 28 bits uniquely identifies the multicast address on the Internet. Multicasting is discussed later.

Class E

This is a reserved class for special purposes by the Internet authority. There is no netid or hostid in this class. The first four bits of this class are 1111.

2.4.1.3 Determining the Class

There are two ways of determining the class of an address depending on the format it is represented.

If the address is given in the form of binary then depending upon the first few bits one can identify the class.

- If the first bit is 0 then it is Class A
- If the first 2 bits are 10 – Class B
- If the first 3 bits are 110 – Class C
- If the first 4 bits are 1110 – Class D
- If the first 4 bits are 1111 – Class E.

However mostly the addresses are given in the decimal notation, which requires following method for determining the Class.

- Class A - first number is between 0 and 127
- Class B - first number is between 128 and 191
- Class C - first number is between 192 and 223
- Class D - first number is between 224 and 239
- Class E - first number is between 240 and 255

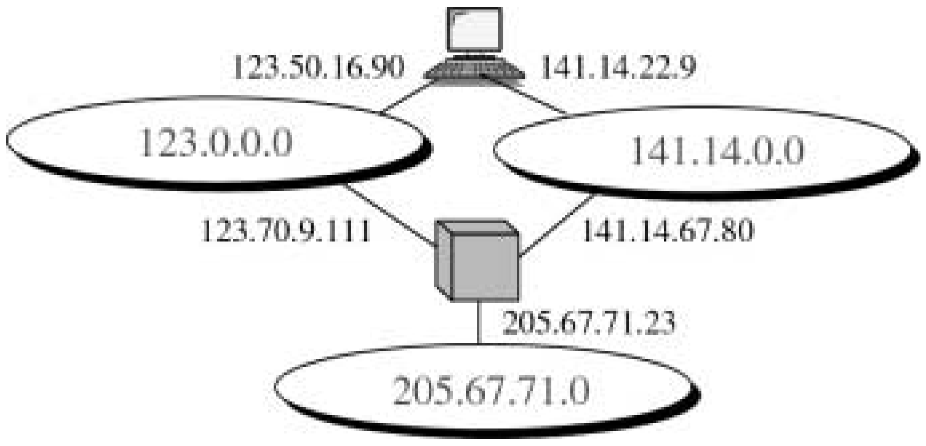
Getting the *netid* and the *hostid*

Given an IP address it is possible to get the *netid* and *hostid.*

- Class A - first byte is *netid* and remaining 3 bytes are *hostid.*
- Class B - first 2 bytes are *netid* and last 2 bytes are *hostid.*
- Class C - first 3 bytes are *netid* and remaining 1 byte is *hostid.*
- Class D – entire address is used for the multicasting.
- Class E – this is a reserved address class.

2.4.1.4 Multihomed Devices

An Internet address identifies a system within a physical network. Hence if the system is connected to more than one physical network it must have more than one Internet (IP) address. In fact a system has a different IP address for each network connected to it. Any system, which is connected to more than one physical network, is called *multihomed* system. A router which routes the IP datagram packet over the Internet should be connected to more than one network and hence it will have more than one IP address, one for each interface. The figure illustrates this.



The computer is connected to two networks and hence has two Internet addresses (123.50.16.90 and 141.14.22.9) and the router device is connected to three different networks and has 3 IP addresses (123.70.9.111, 141.14.67.80 and 205.67.71.23).

The Internet address defines the network location of a system. Since the Internet address is made up of *netid* and *hostid*, it can only define the connection of a system to a specific network. One of the ----implications of this is that if a computer is shifted from one network to another network its IP address must be as the *netid* will be different for the new network.

2.4.1.5 Unicast, Multicast and Broadcast addresses

Sometime a data packet may have to be sent to more than one system and some time data has to be sent to all the systems in the network. The Internet addressing mechanism has provision for these.

Unicast Addresses

Unicast communication is *one-to-one*. When a data packet is sent from a source system to an *individual destination system*, a unicast communication takes place. All system on the Internet should have one unique unicast address. Unicast addresses belong to class A, B or C.

Multicast Addresses

Multicast communication is *one-to*-many. When a data packet is sent from an individual source to *group of destinations*, a multicast communication takes place. A multicast address is a Class D address. The whole address defines a multicast group id. A system on the Internet can have one or more multicast addresses (in addition to its unicast address or addresses). If a system intends to participate in a particular multicast group then it should enable corresponding multicast address in its software. Note that the multicast addresses (or Class D) can never be a source IP address; it can only be a destination address in an IP datagram packet. Some of the multicast addresses are listed below

- 224.0.0.1 - All systems on this SUBNET
- 224.0.0.7 – ST routers.
- 224.0.1.7 - Audio news
- 224.0.1.11 – IETF-1-Audio
- 224.0.1.12 – IETF-1 Video

Broadcast Addresses

Broadcast communication is *one-to*-all. The Internet allows broadcasting only at the local network level. There are two types of broadcasting allowed.

- Limited broadcast - all 1s in the IP address. Both *netid* and *hostid* in this case are all 1. (In decimal notation 255.255.255.255). This identifies all the hosts connected to the local network. A IP datagram packet with destination address as 255.255.255.255 should be received by all the systems connected to the local network.
- Direct broadcast – This identifies the entire host connected to a particular network (need not be local network as in the case of Limited broadcast). Here the *netid* part will identify the destination network (any valid netid) and *hostid* part will have all 1s. Example address is 63.255.255.255. (63 identifies the Class A network and rest implies all hosts on that network).

2.4.1.6 Getting IP addresses

Any organization, which wants its network to be part of the Internet, should get the *netid* for its network from the Internet authorities. One cannot have the *netid* of their choice for their network if they want to be part of the Internet. This is because if two networks have same *netid* and both of them are part of the Internet then it will not be possible to identify the hosts uniquely in the Internet. Hence the Internet authority allocates unique *netid* to each network in the Internet. It is the responsibility of the individual organization to get a unique *netid* for their network.

Depending upon the size of the network an organization has, it will be allotted Class A or B or C addresses. Also there is an additional restriction of *availability of addresses* in a class as the Internet has grown very rapidly. For example if one is allotted a Class A address the *netid* can be 124. For Class B it can be 130.54. and 201.67.121 for Class C. Note that the responsibility of allocating the *hostid* part of the Internet address lies with the concerned organization which has taken a particular *netid*.

2.4.1.7 Private Networks

It should be noted that an organization has to get a unique *netid* if it wants its network to be part of the Internet. If they decide their network *not to be part of the Internet* then there is no necessary to get a unique netid. They have three choices for the netid.

- They can get a unique *netid* and still not connect their network to the Internet. Advantage of this method is that if at some time later the organization decides to connect their network to the Internet then the overhead of conversion is least. All one has to do is just connect the network to the internet through the router. However the drawback of this method is that it is not always possible to get the netid of their choice class.
- Chose any netid one wants and not connect the network to the Internet. Drawback is that there is always a possibility of getting confused with the IP address as being a *valid Internet* address.

- To overcome the above problems the Internet authorities have reserved three block of address for such purposes.

Class A	-	(netid) 10.0.0
Class B	-	(netid) 172.16 to 172.31
Class C	-	(netid) 192.168.0 to 192.168.255

  An organization can choose the netid of their choice from the above list. They don't need to get the permission and it is universally known that addresses with these netid are private addresses.

2.4.2 Routing of IP packets:

An IP packet on the Internet may have to travel across many networks before it reaches the destination host. Two networks are interconnected by a device named *router* (or *gateway* – which is the generalized term). In the Internet a router or gateway would have connected to at least two but in practice many networks. When an IP datagram packet arrives at the router the router has to decide to where to forward the packet so that eventually it reaches its destination host. It uses the destination IP address that is stored in the IP datagram packet to make the correct decision. The process of finding the next path for the incoming IP datagram packets so that it reaches its destination host is *routing*. The device, which does this and forwards the IP packet, accordingly is called *router*. Note that a *router* for its operation has to be connected to at least two networks. More the number of networks it is connected more the complexity of routing process.

An important function of the IP layer in the TCP/IP protocol architecture is *IP routin*g. This provides the basic mechanism for routers to interconnect different physical networks. A device can simultaneously function as both a normal host and a router.

The router only has information about four kinds of destinations:

- Hosts that are directly attached to one of the physical networks to which the router is attached.
- Hosts or networks for which the router has the explicit knowledge about where to forward packets meant for them.

- A default for all other destinations.

Additional protocols are needed to implement a full-function router. These types of routers are essential in most networks, because they can exchange information with other routers in the environment.

There are two types of IP routing: direct and indirect.

2.4.2.1 Direct Routing

If the destination host is attached to the same physical network as the source host, IP datagrams can be directly exchanged. This is called direct delivery and is referred to as direct routing. Direct routing occurs when both source and destination hosts are connected to the same physical network.

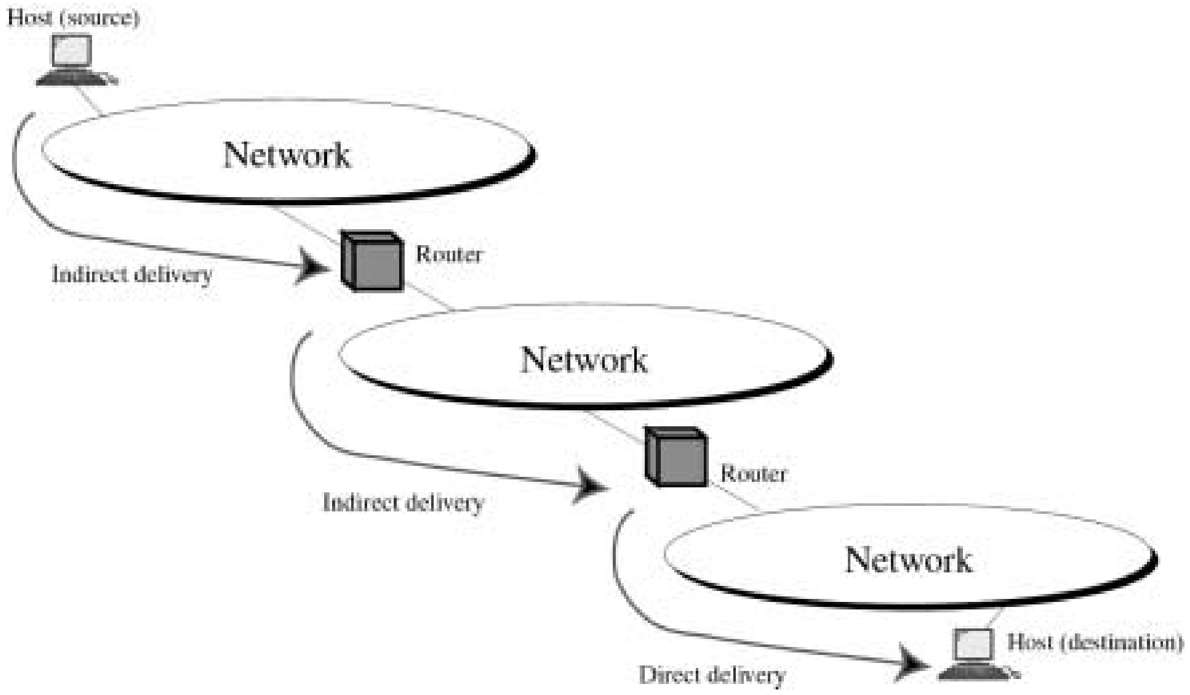
The sender can easily determine if the direct routing can be employed. It can extract the network address of the destination IP address i.e. *netid* (this can be done by setting the *hostid* bits to all 0s) and compare this with the *netid* of the networks to which it is connected. If any of the network to which it is connected has the same *netid* then the host can send the datagram packet over that network to reach the destination host.

Eventually every datagram packet has to be transmitted at the physical / data link layer. The Internet address or the IP address is valid only at the IP layer. At the data link layer the hosts are identified by the physical addresses. So the direct routing also involves resolving the *physical address* of the destination host from its *IP address* and forwarding at the datalink / physical layer. The process of finding the physical address from the IP address of the host system is implemented using a protocol known as ARP (Address Resolution Protocol). This protocol is discussed in later section.

2.4.2.2 Indirect Routing

Indirect routing occurs when the destination host is not connected to a network directly attached to the source host. The only way to reach the destination is via one or more IP router. The address of the first router (the first hop) is called an indirect route in the IP routing algorithm. The address of the first router is the only information needed by the source host to send a packet to the destination host.

The figure illustrates the indirect routing. The source and the destination systems are systems are on two different networks and the IP packets from the host system has to pass through two router devices as there is intermediary network also. However if both the host and the destination systems are on the same networks then there is no need of passing through any router device.



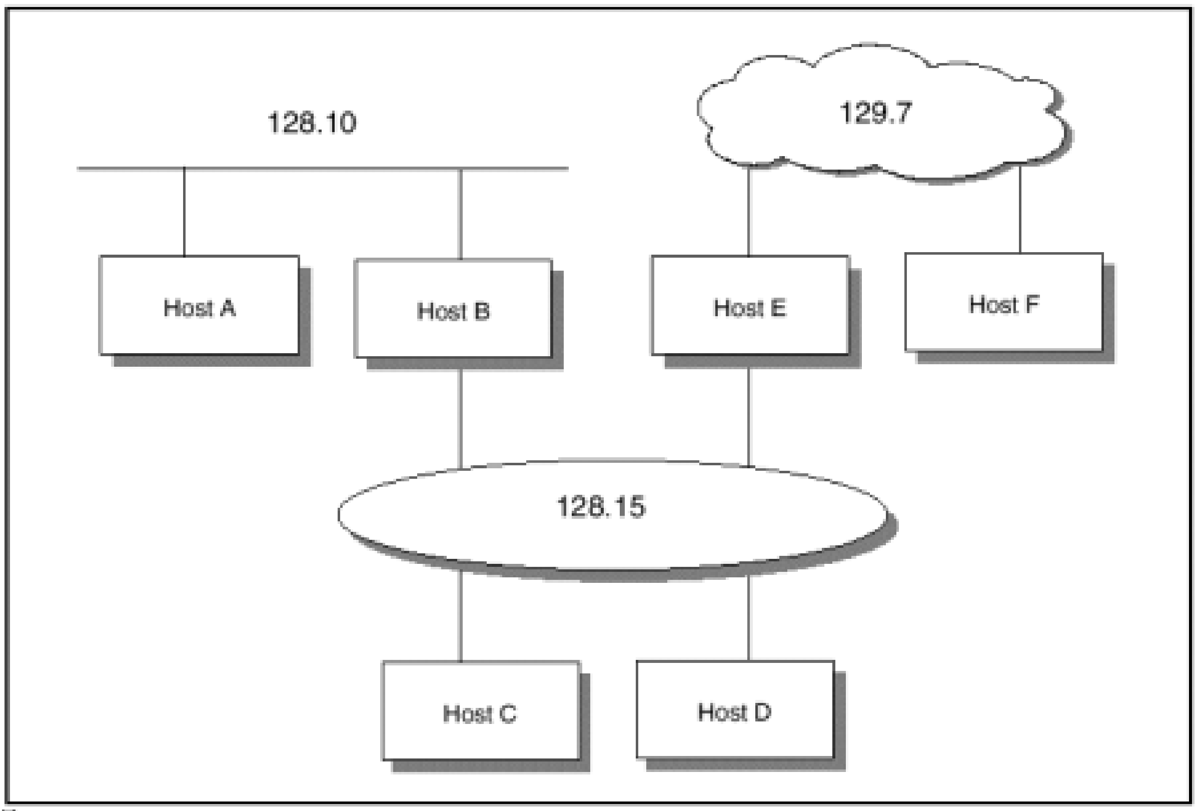
2.4.2.3 IP Routing table

The determination of routes is derived from the table known as *routing table.* This table is used to facilitate IP routing. Each host keeps the set of mappings between the following in this table:

- Destination IP network address (es)
- Route(s) to next gateway(s)

Three types of mappings are found in this table:

- The direct routes describing locally attached networks.
- The indirect routes describing networks reachable via one or more gateways.
- The default route which contains the (direct or indirect) route used when the destination IP network is not found in the mappings of above two types.



The figure represents a sample network. The routing table of Host D may contain the following (symbolic) entries.

Destination	**Router**
129.7.0.0.	Host E
128.15.0.0	Direct
128.10.0.0	Host B
Default	Host B

Since D is directly attached to the network 128.15.0.0, it maintains a direct route for this network. To reach networks 129.7.0.0 and 128.10.0.0, however, it must have an indirect route through E and B, respectively, since these networks are not directly attached to it.
The routing table of host F might contain the following (symbolic) entries:

Destination	**Router**
129.7.0.0.	Direct

Default	Host E

Because every host not on the 129.7.0.0 network must be reached via host E, host F simply maintains a default route through host E.

Static Routing Table

Routing tables can be a static one in which case the entries remain same unless someone changes it manually. If there is any change in the Internet topology, like some of the links going down temporarily, the entries will not be updated automatically resulting in the routing table which does not reflect the changed topology. However this simple mechanism is sufficient for many of the routers which interconnect small networks where changes in the topology is very unlikely or even if that happens the inconvenience caused is not much before the administrator updates the routing table manually.

Dynamic Routing Table

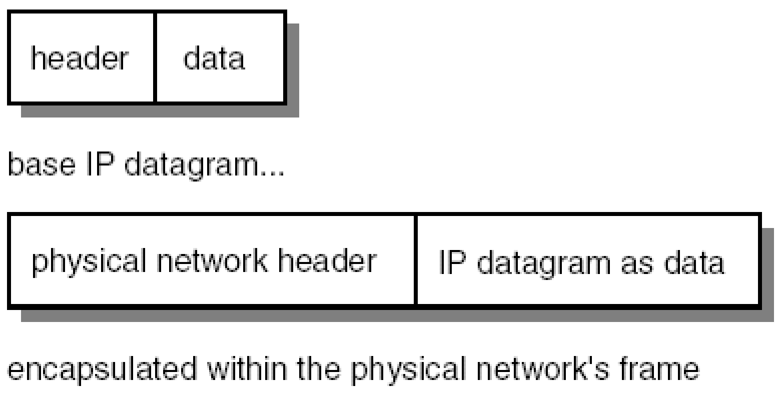
A dynamic routing table is updated periodically automatically. It does not need the manual intervention from the administrator for the updating process. However for this the routers must have implemented one of the dynamic routing protocols such as RIP, OSPF or BGP. Basically the routers participating in dynamic routing communicate with each other using one of the above-mentioned protocols informing about the status of the Internet by exchanging respective protocol packets. Through these communication whenever there is a change in the Internet topology, such as shutdown of a router or a link becoming inactive or even a link becoming active, the router which comes to know about such a change informs the rest of the router about the change using routing protocol, so that the remaining routers updates their routing tables appropriately. Note a change in the Internet may not result in the change in routing table at all the routers. It may affect only some of the routers. The routers in the Internet need to be updated dynamically for efficient delivery of the IP packets.

2.4.2.4 Routing process

Irrespective of the kind of routing table used, whenever an IP datagram arrives at the network IP layer, it determines the IP address of the destination host, and checks whether the datagram packet is meant for itself. If so, it processes the IP packets and takes action accordingly. If the IP packet is not meant for it, then it checks whether the destination host mentioned in the IP packet is in any of the network directly connected to it. This is direct routing and in most implementation of IP this determination can be done without looking into the routing table. If it is not direct routing, then it has to look into the routing table iteratively till it finds an entry corresponding to the destination IP address. If there are none then the default router is selected. Once the next router to which it has to forward the IP packet is found out, the IP layer forwards the packet to the datalink layer for forwarding process.

2.4.3 IP Datagram

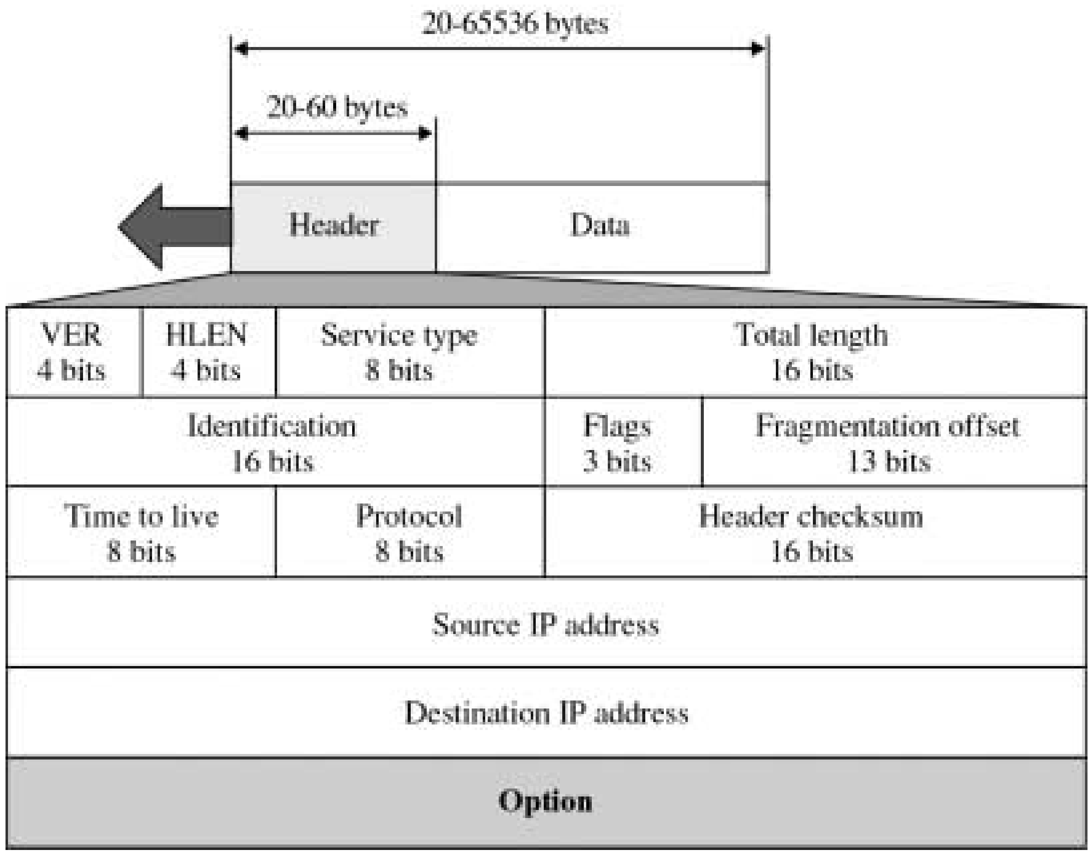
The unit of transfer in an IP network is called an IP datagram. It consists of an IP header and data relevant to higher-level protocols. A datagram is a variable length packet consisting of two parts: *header* and *data*. The maximum length of an IP datagram is 65,535 bytes (octets). The header can be from 20 to 60 bytes long and contains information essential for routing and delivery. The length of the data part varies from packet to packet but the total length of the IP datagram should be within 65,535 bytes.



IP can provide fragmentation and re-assembly of datagrams. All IP hosts must support 576 bytes datagrams without fragmentation. Fragments of a datagram each have a header. The header is copied from the original datagram. A fragment is treated as a normal IP datagrams while being transported to their destination. However, if one of the fragments gets lost, the complete datagram is considered lost. Since IP does not provide any acknowledgment mechanism, the remaining fragments are discarded by the destination host.

2.4.3.1 IP Datagram header:

The IP datagram header is shown below. It is customary in TP/IP to show the header in four-bytes sections. A brief description of each field given below.



VERS:

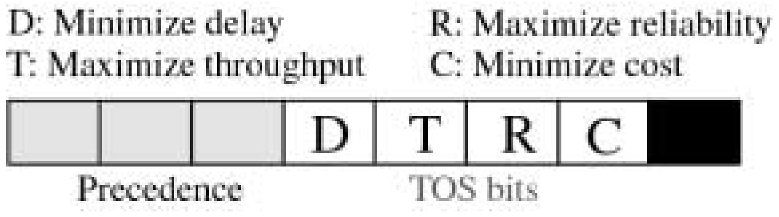
The field contains the IP protocol version. The current version is 4. 5 is an experimental version. 6 is the version for IPv6.

HLEN:

The length of the IP header counted in 32-bit quantities. This does not include the data field. It can vary from 20 bytes to 60 bytes. This field is required as the header has two parts: a fixed part of 20 bytes length and an optional part (when some optional settings are used) of variable length. However most of the datagrams will not have optional settings and hence headers will have a length of 20 bytes.

Service Type:

The service type is an indication of the quality of service requested for this IP datagram. This field contains the following information:



Where:

Precedence: This field specifies the nature and priority of the datagram used for the routing purpose. This field plays important role in conditions of congestion in the network. If a router is congested and needs to discard some of the pending datagram packets then it looks into the precedence fields and discards packets with lowest precedence first. Some datagram are important than others: for example, a datagram used network control is most urgent and important as it carries the information regarding the status of the network, which will be crucial for easing the congestion in the network.

000: Routine	001: Priority	010: Immediate	011: Flash
100: Flash override	101: Critical	110: Internet work control	

111: Network control

TOS: Specifies the type of service value. This field is used while determining the route for the datagram packet. However most of the existing routers does not use this field.

1000: Minimize delay	0100: Maximize throughput
0010: Maximize reliability	0001: Minimize monetary cost
0000: Normal service	

Total Length:

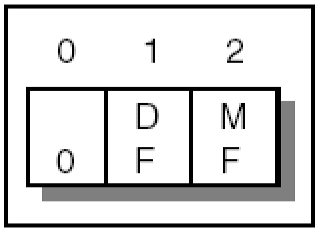
The total length of the datagram: header and data. Since 16 bits are used for this purpose the maximum total length is $2^{16} - 1 = 65,535$ bytes.

Identification:

A unique number assigned by the sender to aid in reassembling a fragmented datagram. Each fragment of a datagram has the same identification number.

Flags:

This field contains control flags. It has 3 bits.



0: Reserved, must be zero.

DF (Do not Fragment): 0 means allow fragmentation; 1 means do not allow fragmentation.

MF (More Fragments): 0 means that this is the last fragment of the datagram; 1 means that additional fragments will follow.

Fragment Offset:

This is used to aid the reassembly of the full datagram. The value in this field contains the number of 64-bit segments (header bytes are not counted) contained in earlier fragments. If this is the first (or only) fragment, this field contains a value of zero.

Time to Live:

This field specifies the time (in seconds) the datagram is allowed to travel. Theoretically, each router processing this datagram is supposed to subtract its processing time from this field. In practice, a router processes the datagram in less than 1 second. Thus the router subtracts one from the value in this field. The TTL becomes a hop-count metric rather than a time metric. When the value reaches zero, it is assumed that this datagram has been traveling in a closed loop and is discarded. The initial value should be set by the higher-level protocol that creates the datagram.

Protocol Number:

This field indicates the higher-level protocol to which IP should deliver the data in this datagram. Some of the protocol numbers are listed below.

0: Reserved

1: Internet Control Message Protocol (ICMP)

4: IP (IP encapsulation)

6: Transmission Control Protocol (TCP)

17: User Datagram Protocol (UDP)

41: IP Version 6 (IPv6)

51: Authentication Header for IPv6 (AH)

89: Open Shortest Path First (OSPF)

Header Checksum:

This field is a checksum for the information contained in the header. If the header checksum does not match the contents, it implies that the datagram is corrupted and is discarded.

Source IP Address:

This is the 32-bit IP address of the host, sending this datagram.

Destination IP Address:

This is the 32-bit IP address of the destination host, for this datagram.

Options:

This field is of variable length. Some of the options defined are

Source routing: which has the route this datagram should take in order to reach the destination

Record routing: which request the routers to record the route this datagram took while reaching the destination. If the router oblige then route should be recorded in the optional part of the header.

Internet Timestamp: This requests the routers to record the Internet timestamp in the optional part of the header.

2.4.3.2 Checksum calculation:

The error detection method used by most of the TCP/IP protocols is called *checksum*. The checksum protects against the corruption that may occur during the transmission of a packet. It is redundant information added to the packet.

The checksum is calculated at the sender and the value obtained is sent along with the packet. The receiver repeats the same calculation on the whole packet including the checksum. If the result is as expected then the packet is accepted; otherwise it is discarded. Note that IP does not inform the sender that it is discarding the packet as it will not be knowing for sure who the sender host system is as the header itself is corrupted. Also it is the duty of the transport layer to take of such situations.

Checksum calculation at the sender:

At the sender, the header (except the *checksum field)* of the datagram packet is divided into (16 bits) 2 bytes section. All these 2 bytes sections are added together using the one's complement arithmetic. (In one's complement arithmetic addition of *n-bit* numbers always result in another *n-bit* number.). The sum is then complemented (i.e. all 0 are converted into 1 and all original 1s are converted into 0s) to produce the *checksum*. This checksum is filled in the header of the IP datagram while sending. Note that the *checksum* field was not considered while calculating this.

Checksum calculation at the receiver:

At the receiver the header (including the *checksum* field) is again divided into 16 bit sections. Each of these sections is added using the same one's complement arithmetic. If the final result is 0 (i.e. all 16 bits are 0) then it is accepted. If the result is not 0 it indicates that there was a corruption of the header and hence the datagram will be discarded.

2.4.4 Fragmentation

When an IP datagram travels from one host to another, it may pass through different physical networks. Each physical network has a maximum frame size. This is called the *maximum transmission unit* (MTU). It limits the length of a datagram that can be placed in one physical frame.

A datagram can travel through different networks. Each router decapsulates the IP datagram from the frame, processes it, and then encapsulates it in another frame. The format and size of the received *frame* depends on the protocol used by the underlying physical network through which the frame passes through.

Since each protocol used at the physical layer has its own MTU, there is every possibility that the incoming frame exceeds the MTU of the outgoing physical network.

To enable forwarding the datagram in such cases IP implements a process to fragment datagrams exceeding the MTU. The process creates a set of datagrams within the maximum size. The receiving host reassembles the original datagram.

A datagram can be fragmented by the source or the any router in the path. The reassembly of the datagram, however, is done only by the destination host as each fragment becomes an independent datagram.

When a datagram is fragmented, required parts of the header must be copied by all the fragments. The host or the router, which fragments the datagram, must change the values of three fields: *flags, fragmentation offset, total length.* It goes without saying that *checksum field* for each of the fragments has to be recomputed and duly filled.

An unfragmented datagram has an all-zero fragmentation information field. That is, the *more fragments* flag bit is zero and the *fragment offset* is zero. The following steps are performed to fragment the datagram:

- The *DF flag* bit in the *flag* field is checked to see if fragmentation is allowed. If the bit is set which indicates not to fragment, the datagram will be discarded as it cannot be forwarded and an ICMP error returned to the originator.
- Based on the MTU value, the data field of the datagram is split into two or more parts. All newly created data portions must have a length that is a multiple of 8 octets, with the exception of the last data portion.
- Each data portion is placed in an IP datagram. The headers of these datagrams are minor modifications of the original:
  - The more fragments flag bit is set in all fragments except the last.
  - The fragment offset field in each is set to the location this data portion occupied in the original datagram, relative to the beginning of the original unfragmented datagram. The offset is measured in 8-octet units.
  - The header length field of the new datagram is set.
  - The total length field of the new datagram is set.
  - The header checksum field is re-calculated.
  - Each of these fragmented datagrams is now forwarded as a normal IP datagram. IP handles each fragment independently. The fragments can traverse different routers to the intended destination. They can be subject

to further fragmentation if they pass through networks specifying a smaller MTU.

At the destination host, the data is reassembled into the original datagram. The identification field set by the sending host is used together with the source and destination IP addresses in the datagram. Fragmentation does not alter this field.

In order to reassemble the fragments, the receiving host allocates a storage buffer when the first fragment arrives. The host also starts a timer. When subsequent fragments of the datagram arrive, the data is copied into the buffer storage at the location indicated by the fragment offset field. When all fragments have arrived, the complete original unfragmented datagram is restored. Processing continues as for unfragmented datagrams. If the timer is exceeded and fragments remain outstanding, the datagram is discarded.

2.5 ARP – Address Resolution Protocol.

The ARP protocol is a network-specific standard protocol. The address resolution protocol is responsible for converting the higher-level protocol addresses (IP addresses) to physical network addresses. It is described in RFC 826.

2.5.1 ARP Overview:

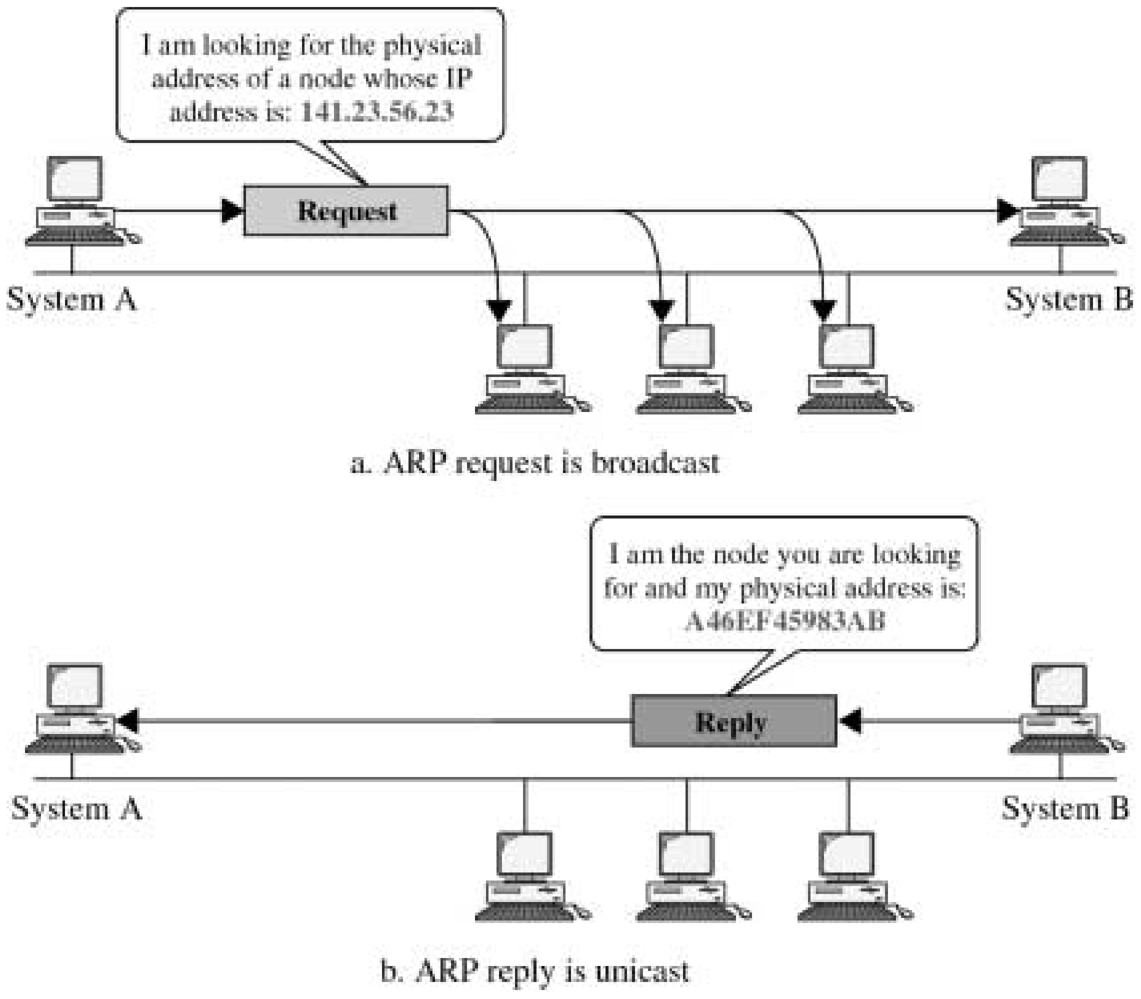
As mentioned earlier IP layer provides a *virtual or logical* network view to the higher layer protocols where in each host is identified by the unique address known as IP address. However when the datagram are sent on the physical network, this IP address cannot identify the destination system. At the physical or data link layer it is the *physical address, which* identifies the host system.

This implies that the when datagram from the IP layer is to be passed on the physical network to the another host on the physical network, the *physical address* of the host is to be determined from its IP address and this physical address will be filled in the data frame and sent over the link. In other words there is a need to do *address translation;* from IP address to physical address.

It is the responsibility of the ARP module, which resides at the *network layer* in the TCP/IP protocol stack to do this address translation.

For this purpose the ARP module will have a lookup table called *ARP cache*. This table will have entries for all the known IP addresses. For each IP address present in the cache corresponding *physical address* is stored against it. So when ARP module is requested to perform an address translation it first looks into this cache, if it finds an entry for the IP address, it gets the corresponding *physical address* and returns it.

However many times there will be no entry in the cache for requested IP address. In such cases ARP module broadcasts a message over the network requesting for the address translation and after getting the proper reply stores the physical address of the IP address requested in the cache table and passes the same to the upper layers which requested it.

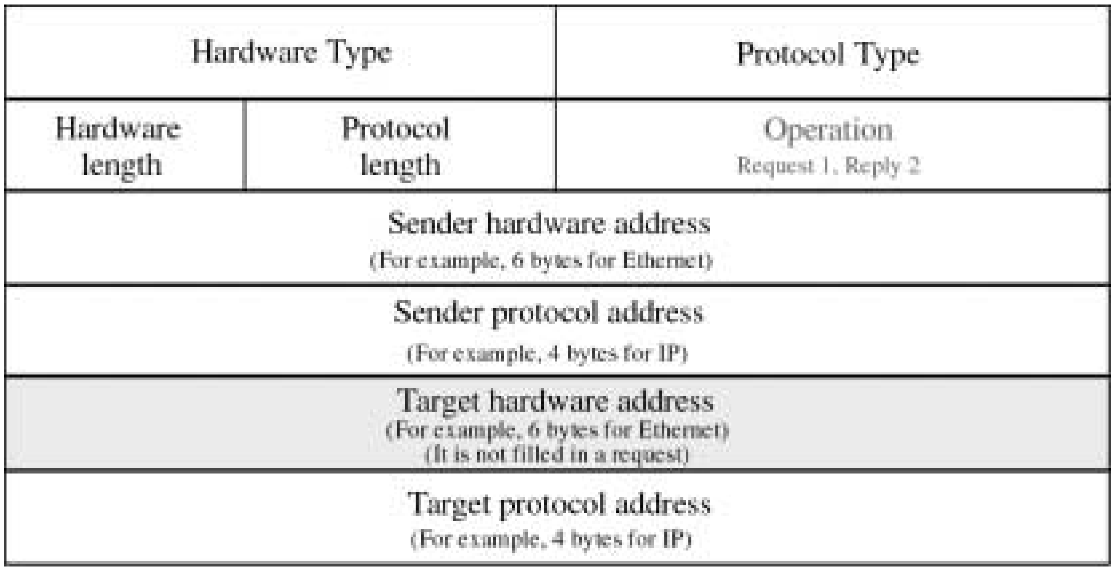

I am looking for the physical address of a node whose IP address is: 141.23.56.23
Request
System A
System B
a. ARP request is broadcast
I am the node you are looking for and my physical address is: A46EF45983AB
Reply
System A
System B
b. ARP reply is unicast

Every host or the router on the network receives and processes the ARP query packet, but only the intended recipient recognizes the IP address and sends back the an *ARP response* packet. The response packet contains the recipient's IP and physical addresses. The packet is unicast directly to the inquirer using the physical address received in the query packet.

In the figure, the system on the left has (A) has a packet that should be delivered to another (B) with an IP address 141.23.56.23. System A IP layer modules needs to pass this to data link layer, but it does not know the physical address of the recipient system B. The ARP module in the system A, looks for B's physical address in its cache, but does not see an entry there (for 141.23.56.23). So it sends a ARP query broadcast message over the network. Though this request message was received and processed by all the systems on the network, only the system with matching IP address i.e. system B, replies back with the ARP responses. (Note that the physical address *A46EF45983AB* is a hexadecimal number where in each number in fact represents 4bits. Physical address here is 48 bit long.)

2.5.2 ARP Packet format:

The figure shows the format of an ARP packet.



Hardware Type		Protocol Type
Hardware length	Protocol length	Operation Request 1, Reply 2
Sender hardware address (For example, 6 bytes for Ethernet)		
Sender protocol address (For example, 4 bytes for IP)		
Target hardware address (For example, 6 bytes for Ethernet) (It is not filled in a request)		
Target protocol address (For example, 4 bytes for IP)		

Hardware Type:

This is a 16-bit field defining the type of the network on which the ARP is running. Each LAN has been assigned an integer based on its type. For example, Ethernet has type1.

Protocol Type:

This is a 16-bit field defining the protocol, which is requesting the ARP service. The value of this for IPv4 is 0800_{16}. ARP can be used with any higher layer protocol.

Hardware Length:

This is an 8-bit field defining the length of the physical address in *bytes*. For Ethernet this value is 6 as it has 48 bits (=6 bytes) physical address.

Protocol Length:

This is an 8-bit field defining the length of the logical address in bytes. The IPv4 protocol has a value of 4 (as IP address are 4 bytes long).

Operation:

This is a 16-bit field defining the type of the packet. Two types are defined. ARP request (Type 1) and ARP reply / response (Type 2).

Sender Hardware Address:

This represents the physical hardware address of the sender machine. The length of this field is indicated by the Hardware Length field.

Sender Protocol Address:

This represent the logical address (or more commonly Internet address) of the sender. If the protocol is IP this represents the senders IP address.

Target Hardware Address:

This represents the physical hardware address of the target system. The length of this field is indicated by the Hardware Length field. Though this field is present in both the ARP request packet, it will be all 0s as the sender will not know the target systems hardware address.

Target Protocol Address:

This represent the logical address (or more commonly Internet address) of the target system. If the protocol is IP this represents the senders IP address. When ARP request packet is received at the system on the network whose IP address is same as this, then it replies with an ARP response packet filling the Target Hardware Address. Others systems, does not respond.

2.6 RARP – Reverse Address Resolution Protocol.

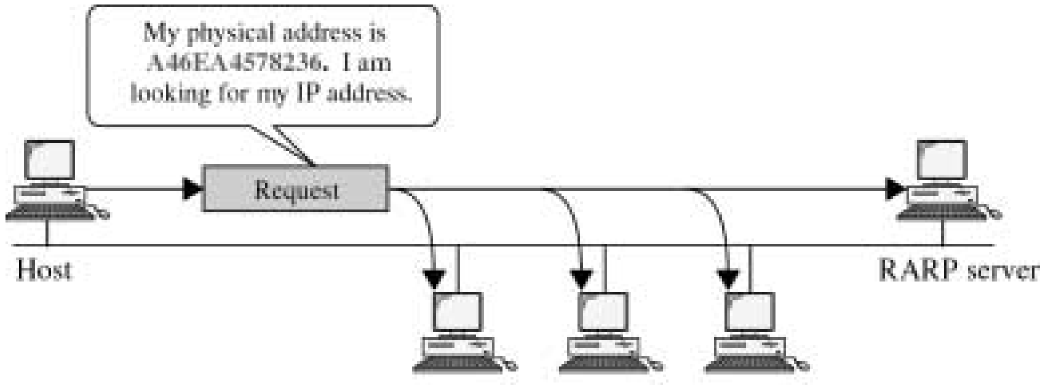
In some local area networks there would be a powerful computer, which acts as a server. Connected to that would be several hosts which will not have any disks (hard disk / floppy disk). People would use these hosts as the front-end system and would be connected to the server over the network. Eventually they would be working on the server but using these hosts as the front-end systems. Having disk-less hosts has an advantage in some setups, where there would be a central powerful server computer on which many people can work simultaneously by connecting through such hosts. As these hosts are disk-less system, they would not need any configuration by the administrator. Maintenance would be very minimal. These disk-less systems would be booted from the ROM (Read Only Memory chip), which is programmed by the manufacturer. It cannot include the IP address as the network administrator assigns them.

Each time these disk-less hosts are powered on, they will not be aware of their IP address, as they don't have any disk or storage device. However they will be aware of their Hardware address as encoded into the Network Interface Card, from which they can get it.

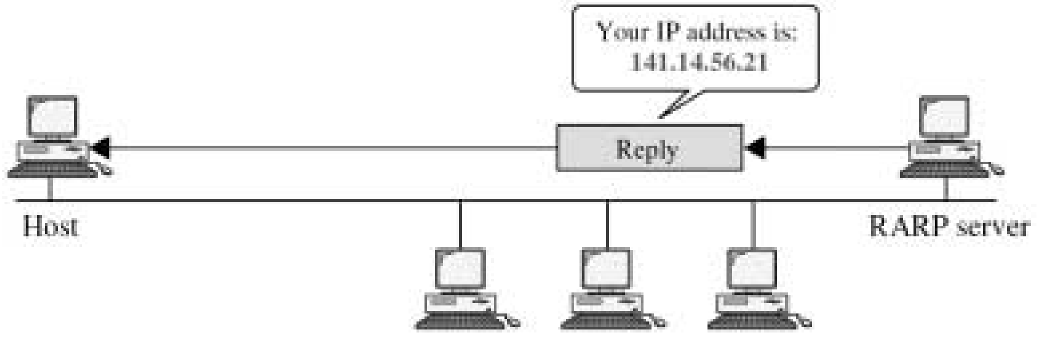
Since in a network each system is recognized by the logical or more commonly, IP address, it is vital for the system to be aware of its IP address. Any IP datagram to be sent, should have the senders IP address duly filled in. Otherwise the recipient of that packet would not know from whom the packet arriver and it would just discard.

To find out ones IP address these disk-less host systems uses a protocol known as *Reverse Address Resolution Protocol*. This is a protocol, which does the function opposite of that of ARP. i.e. given the physical address gets the IP address.

At boot up time, these disk-less hosts would broadcast a RARP request packet over the network after filling its physical address in it. The server which will have a reference table of physical address and their corresponding IP address would respond with the RARP response packet. These reference table-having entries for each of the physical address (present in the local network) and their IP address would be created and maintained by the administrator.


My physical address is A46EA4578236. I am looking for my IP address.
Request
Host
RARP server

a. RARP request is broadcast



b. RARP reply is unicast

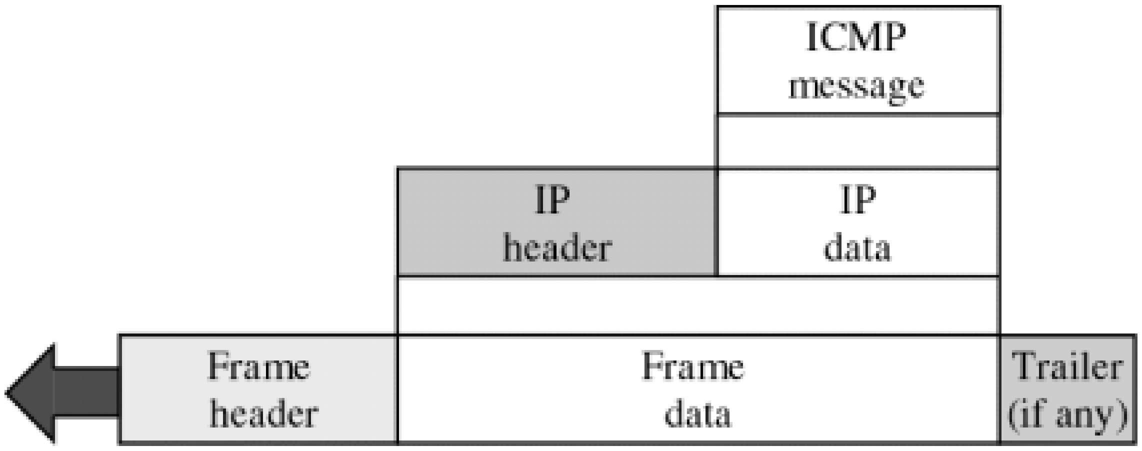
The format of the RARP is exactly same as the ARP packet except that the operation field is either RARP request or RARP reply.

2.7 ICMP –Internet Control Message Protocol.

As discussed earlier IP provides an unreliable and connectionless datagram delivery. It was designed this way to make the efficient use of the network. However IP protocol does not have any error-reporting or error-correcting mechanism. What happens if something goes wrong, what happens if the router does not find route for the IP packet and so discards it. The sender will be never aware of it, and he would keep resending the same. There are many situations like these where in a host IP layer should be made aware of what has happened to the packet it sent.

The ICMP or Internet Control Message Protocol was designed to compensate these deficiencies of IP. It is a companion to the IP protocol and like ARP and RARP is part of the network layer along with the IP.

Though ICMP is a network layer protocol its messages are never sent directly to the datalink layer. Instead the messages are first encapsulated inside the IP datagram and then sent to the lower datalink layer. This is shown in the figure.



2.7.1 Types of ICMP messages:

ICMP messages are divided into two broad categories: *error-reporting messages* and *query messages.*

The error reporting messages report the problems that a router or a host (destination) may encounter when it processes an IP packet. They would report back the problem back to the IP module at the sending system.

The query messages, which occur in pairs, help a host or a network manager get specific information from a router or another host. For example, this can be used by the hosts to discover the routers present in their network. The host would send a ICMP query asking for routers to respond. The routers present in the network will respond with an ICMP reply message. The host would get enough information about the router from this reply.

Under error-reporting type, the following messages are present

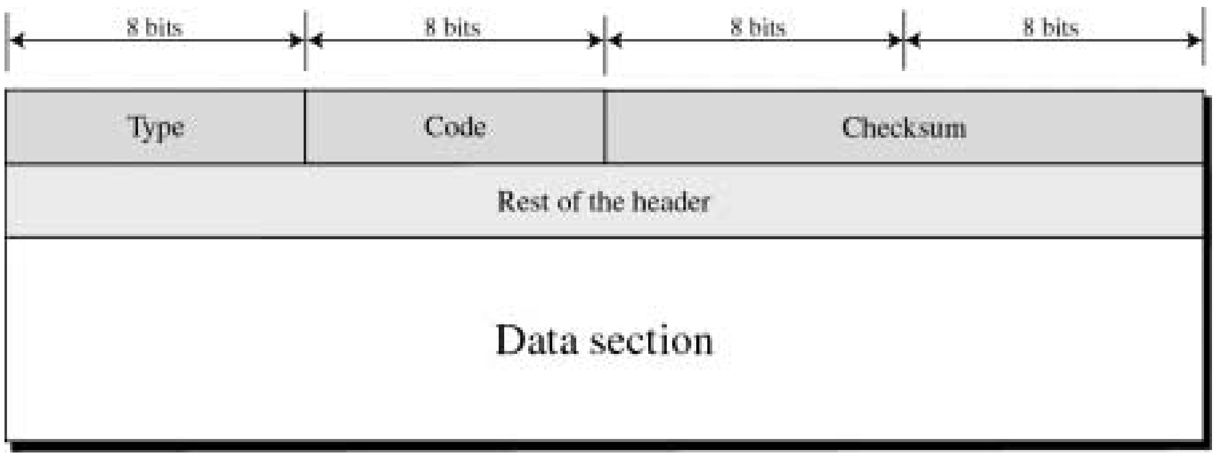
- *Destination unreachable*
- *Source quench*
- *Time exceeded*
- *Parameter problem*
- *Redirection*

The query messages type has following messages.

- *Echo request and reply*
- *Timestamp request and reply*
- *Address mask request and reply*
- *Router solicitation and advertisement*

2.7.2 ICMP message format:

An ICMP message has an 8 byte header and a variable size data section. The first 4 bytes of the header is common to all the messages. The rest 4bytes would depend on the message type. The first field, ICMP type, defines the type of the message. The code field specifies the reason for the particular message type. The checksum field is used for detecting the error in the ICMP packet. Its usage is same as in IP datagram. The rest of the header is specific to each message type.



The data section in the ICMP packet has information for finding the original packet which caused the error. The IP layer, which receives error-report ICMP message, would use the information stored in this data section to find out about the packet, which was sent by it, resulted in this error-reporting message. It is important for the IP module at the receiving host to identify the original IP packet, which caused the error-report ICMP message generation, so that it can take appropriate action and also inform the appropriate upper layers about the same.

2.7.3 Error reporting messages:

One of the main responsibilities of the ICMP is to report error. As discussed earlier IP is a unreliable protocol and does not have error-reporting mechanism. So it is the responsibility of the ICMP that the IP module at the sending system becomes aware of the error, so that it can take remedial action. However ICMP does not correct the errors. It is left for the higher layer protocols, which uses the error reports, sent by the ICMP for the same purpose.

Error messages are always sent to the original source because the only information available in the datagram about the route is the original source and destination systems IP addresses. The ICMP uses the source IP address to send the error message to the source (originator) of the datagram.

All the error messages contain a data section that includes the IP header of the original datagram plus the first 8 bytes of data in that datagram. The original datagram header is added to give the original, which finally receives the ICMP message, the information about the datagram itself. The 8 bytes of data is also added, as that will have crucial information about the upper layer protocols which generated the original message at the sender.

Destination unreachable:

When a router cannot route the datagram or a host can not deliver the datagram to the intended process, the datagram is discarded and the router or host sends a destination unreachable message back to the host that initiated the datagram.

Source quench:

The IP protocol is a connectionless protocol. There is no communication between source host, which generates the datagram, the routers, and the destination host which process it. Hence IP cannot provide *flow-control* mechanism, by which the routers or the destination host can influence, the rate at which the datagram are generated at the source. Without flow control, if the source host is generating datagrams at higher rate than the router or the destination system can process them, they will lead to congestion and eventually those datagrams will be dropped.

ICMP source quench message provides a mechanism for this to be avoided. The source quench message informs the source that a datagram has been discarded at the router or the destination host. Upon receiving the source host is supposed to slow down the generation of datagram packets.

Time Exceeded:

Whenever a router receives a datagram whose *time-to-live* field has the value of zero, it discards the datagram and sends this *Time Exceeded* message back to the source host.

This is done in order to ensure that the datagrams do not move from one router to another indefinitely in a loop, in the Internet without reaching the destination host. This looping can happen if the routing tables doest not reflect the true topology of the network, as the topology changes dynamically and some time the routers may not be aware of it.

When a router cannot route the datagram or a host can not deliver the datagram to the intended process, the datagram is discarded and the router or host sends a destination unreachable message back to the host that initiated the datagram.

2.7.4 Query Messages:

Apart from error reporting, the ICMP also provides mechanism for diagnosing the network status. This is accomplished through the *query messages,* of the ICMP. In this type of message a host system or router sends a message querying about the network status to another system on the network. The receiving system replies with the answer in the query reply message.

There are four pairs of query message.

- Echo request and reply
- Timestamp request and reply
- Address mask request and reply
- Router solicitation and reply

Echo request and reply:

The echo-request and echo-reply message are designed for diagnostic purposes. Network managers and users use this pair of messages to identify the network problems. One frequently uses this to determine whether a particular host or the router on the network is reachable. They send an *echo-request* message to the destination host. If the Internet link between the source and the destination system is active and the destination system is running, then the ICMP module at the destination system responds with the *echo-reply* message. Receipt of this reply message at the source host confirms the active link between the two systems.

Any host or router can send the *echo-request* message to another host or router on the network. Upon host or the router after receiving the echo-request message, creates a *echo-reply* message and returns it to the original.

Timestamp Request and Reply:

Two hosts can use *time-stamp request* and *time-stamp reply* messages to determine the round-trip time needed for an IP datagram to travel between them. It can also be used to synchronize the clocks in two systems. The echo-request and echo-reply message are designed for diagnostic purposes. Network managers and users use this pair of messages to identify the network problems.

The source creates a timestamp-request message. The source fills the original timestamp field with the Universal Time shown by its clock at the departure time.

The destination host creates the time-stamp-reply message. The destination copies the original timestamp value from the request message into the same field in its reply message. It then fills the *receive timestamp* field with the Universal time shown by its clock at the time of message arrival. And fills the *transmit timestamp* field with the time at message transmission.

Upon receiving the *timestamp-reply* message the host can compute the round trip time using the following formulas.

Send time = value of receive timestamp – value of original timestamp

Receiving time = time the reply message received – value of transmit timestamp

Round-trip time = Send time + Receive time.

Address Mask Request and Reply:

The IP address of a host contains a *network address*, *subnet address* and the *host identifier.* A host may know its full IP address but it may not know which part of the address defines the network and subnet addresses and which part identifies the host id.

For example, a host may know its 32-bit IP address as

10011011. 10100010. 11100101. 10101001

But it may not that the left 20 bits represent network and subnet addresses. And the remaining 12 bits represent the host. In this case the host needs the following *mask*:

11111111. 11111111.11110000.00000000

By doing binary AND operation between the *mask* and the IP address bits, we can get the network and subnet addresses. The 0s identify the position of host id.

Applying the above mask to the IP address we get

Net ID and Subnet ID:		10011011. 10100010. 1110
Host ID	:	0101. 10101001

It should be noted that the netid and the subnet id are determined by the network administrator. So the host may not be aware of it. In order to get the *mask*, using which a host can find out its netid, sub netid and host id, a host sends an A*ddress Mask Request Message* over the LAN by either broadcasting or sending directly to the router. Upon receipt of this message the router (or any other server host) responds with the *Address Mask Reply Message*, providing necessary mask to the host.

Router Solicitation and Advertisement:

A host, which wants to send the data to a host on another network, needs to know address of the routers connected to its own network. Also it should know which routes are alive and active. If there is more than one router in the network, the host should find out which router to use to send the datagram to the destination host. For this purpose the host uses this message.

A host can broadcast or multicast a *router-solicitation* message. The router or routers that receive this message broadcast their own routing information using the *router-advertisement message.* A router can also periodically advertise *router-advertisement message* even if no host solicited.

2.8 Summary

The network layer is the most important layer in the TCP/IP model, as it is here where the responsibilities of moving the data from the source to the destination host across several physical networks are handled. This layer provides a transparent mechanism to the upper layer for data transmission to the upper layers. Because of the logical network concept provided by this layer, all that is required to identify uniquely a host on the Internet is the IP address. Core protocol of this layer is the IP. It provides all the above services. However because of the efficiency it does not provide the connection oriented data service and does not guarantee the error free delivery. These are to be taken care by the upper layer protocols. There are other protocols at this layer, which compliment the service of IP. They are ARP, RARP which are used to find out the physical address of destination system and ones own IP address respectively. Also there is a protocol, ICMP, which provides the mechanism for error control and other similar services.

<u>UNIT 3</u>

Transport Layer

Contents:

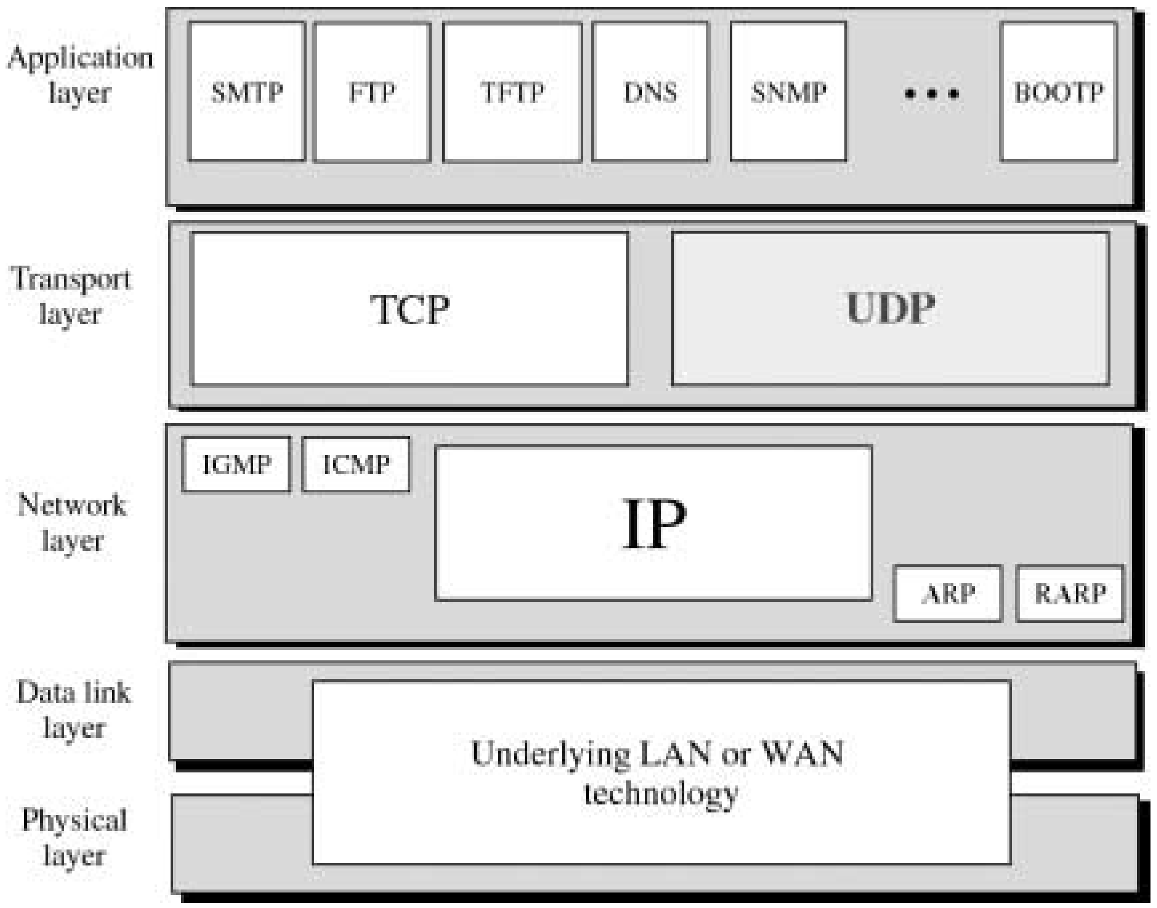
3.1 Introduction:

The Transport Layer is the heart of the TCP/IP protocol model. It is here where the tasks related to *process-to-process communication, reliability, stream data services* are provided. These are the services, which made the task of communicating across network simple from the application programme perspective.

There are two protocols defined at the transport layer.

- UDP – User Datagram Protocol
- TCP – Transmission Control Protocol

The Figure illustrates the relationship between the transport layer protocols and the rest of the layers.



UDP is the simpler among the two, and it provides the *process-to-process* communication service and limited error checking. Because of its simplicity and unreliable service, it is efficient and hence used in applications where complete reliability is not important. It is especially useful in multimedia application where in voice and video data are sent and which does not require high reliability but requires faster data transfer. Even if some of the data is lost or corrupted, the loss of information is minimal and it is manageable.

We begin this unit with a section on UDP. The concepts of *port number (address), socket address* are explained. This follows with a section about the UDP datagram format. Various fields and how they are used are explained. Complete understanding of UDP requires also the knowledge about its operation. Various operations that are involved in the data transmission from an application to another, while using UDP at the transport layer, are explained for the same purpose.

TCP is the protocol, which made the task of communicating across the network simpler from the application programme (process) point of view. The developers who write application programme, does not have to worry about many cumbersome tasks that are involved whenever there is data loss, or they arrive out of order, or issues related with the flow control. TCP handles all of them and provides a *reliable, connection-oriented, stream data service* to the end process. It also has the flow-control built into it. All these features are explained in the section related to TCP.

3.2 Objectives:

In this Unit you would learn about

- Port address and Socket addressing used for process-to-process communication
- User Datagram Protocol (UDP), services offered by it, its datagram format.
- Various steps involved in UDP operation.
- Services offered by the TCP.
- TCP segment format
- How flow control, error control mechanisms are implemented in TCP
- Various steps that are involved in the data transfer using TCP.

3.3 UDP – User Datagram Protocol:

It should be noted that the underlying physical network technology available are quite reliable. They have mechanisms built in, to check for error. Though this reliability is not fool-proof, it is sufficient for many end applications. Loss or corruption of data to some extent is tolerable. All that these applications requires is a mechanism by which they can send data from one application programme or process to another on some host across the network. They should not be worry about the underlying network technology used, where exactly host resides and such things.

If one observes carefully, the services required by these applications are mostly met by the IP layer itself, except for the *process-to-process* communication.

UDP is the protocol, which fills this gap. UDP provides *process-to-process* communication instead of *host-to-host* communication.

UDP is a connectionless, unreliable transport protocol. It does not add anything to the services of IP except for providing process-to-process communication. Also, it performs very limited error checking.

Because of its simplicity, it is more efficient. Establishing a connection with the destination process is fast, and *if reliability is not an important criteria* then the *transfer of data is faster for bulk transfer.* This is because reliability requires, implementing the acknowledgements, resending the lost or corrupted datagrams and many more overheads. This definitely decreases rate at which datagrams can be sent. As mentioned earlier some of the application as in the case of multimedia, which requires bulk transfer of data, needs no foolproof reliability. The reliability provided by the underlying networks is sufficient. Loss of data to some extent is tolerable, and in fact in some cases there are techniques available to recover the lost data with the source requiring to send it again.

It is no wonder, despite its very limited services; *UDP is the desired protocol* for these applications at the transport layer.

3.3.1 Process-to-Process Communication:

Before we examine UDP, we must first understand host-to-host communication and process-to-process communication and the difference between them.

The IP is responsible for communication between two hosts. As a network layer protocol, IP can deliver the message only to the destination host system. However, this is an incomplete delivery. The message still needs to be handed to the correct process or a running instance of a programme. This is where a transport layer protocol such as UDP takes over.

3.3.1.1 Port Numbers:

In TCP/IP model, each process requiring the TCP/IP communication service is assigned a 16bit (2 byte) number called *port number*. The TCP/IP standard has defined unique port numbers for some of the well-known network application process. Because of this it is possible to uniquely identify a network application process running on a host machine.

This approach of identifying the process can be best explained by using the *client-server model*. A process on the local host, called a *client*, needs services from a process usually on the remote host, called a *server*.

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

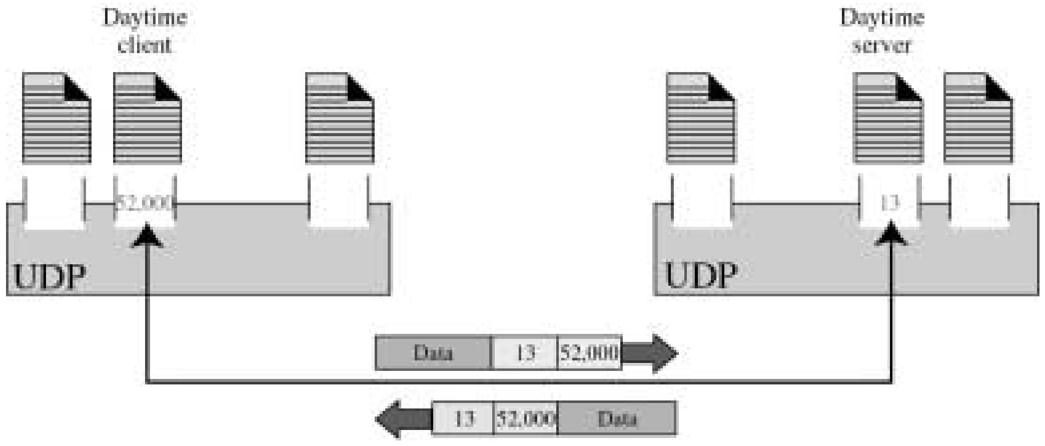
Computer systems today support both multi-user and multiprogramming environments. Local and remote computers can run several server programs at the same time. For communication, we must define the

- Local host
- Local process
- Remote host
- Remote process

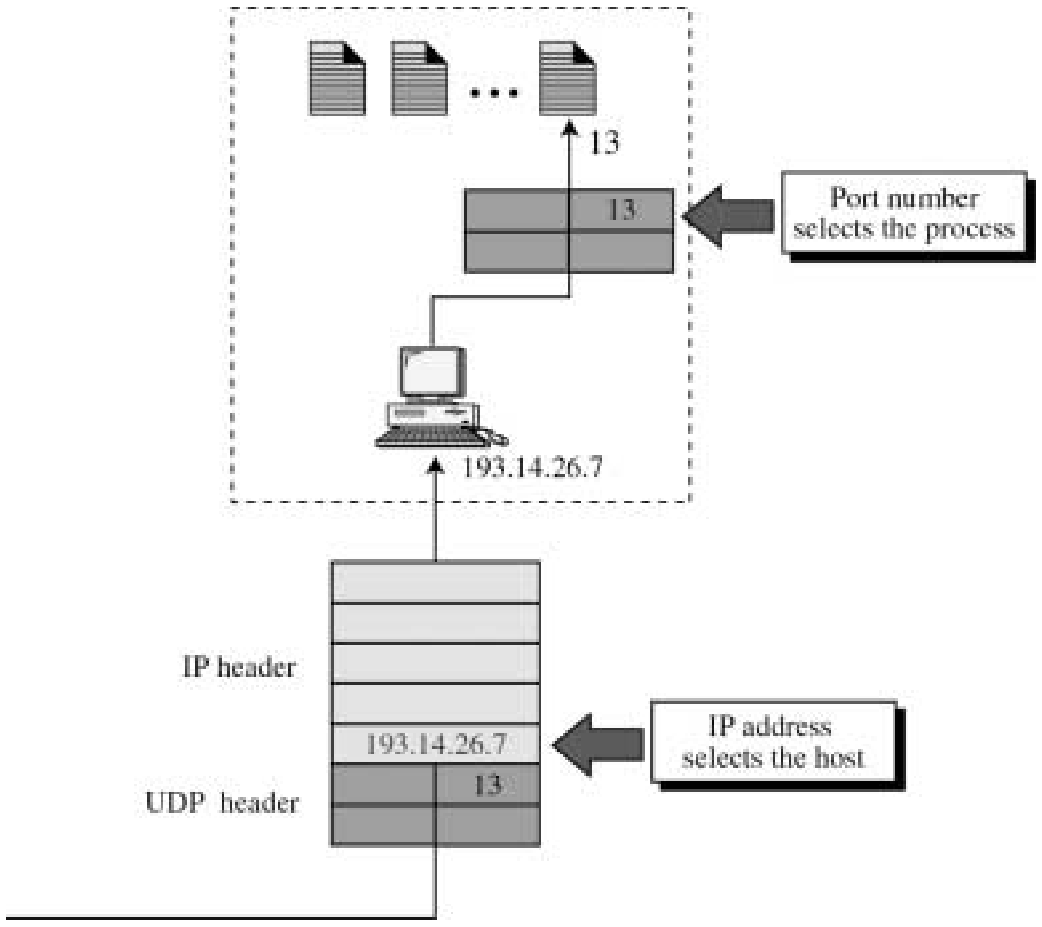
The local host and the remote are identified using IP addresses. To define the processes, we need second identifiers, which are called Port numbers. In the TCP/IP protocol suite, the port numbers are integers between 0 and 65,535.

The client program is assigned a port number, chosen randomly by the UDP software running on the client host. This is the *ephemeral port number*.

The server process must also be assigned a port number. This port number, however, cannot be chosen randomly. If the computer at the server site runs a server process and assigns a random number as the port number, the process at the client site that wants to access that server and use its services will not know the port number. Of course, one solution would be to send a special packet and request the port number of a specific server, but this requires more overheads. TCP/IP has decided to use universal port numbers for servers; these are called well-known port numbers. Every client process knows the well-known port number of the corresponding server process. For example, while the Daytime *client process*, discussed above, can use an ephemeral (temporary) port number 52,000 to identify itself, the Daytime *server process* must use the well-known (permanent) port number 13. Figure shows this concept.



It should be clear by now the roles played by the IP addresses and port numbers in selecting the final destination of data. The destination IP address defines the host among the different hosts in the Internet. After the host has been selected, the port number selects one of the intended processes on this particular. The following figure helps in understanding this concept.



Assignment of Port numbers:

The port numbers divided into three ranges:

- **Well-known ports:** The ports ranging from 0 to 1,023 are assigned and controlled by Internet Authority. These are the well-known ports.
- **Registered ports**: The ports ranging from 1,024 to 49,151 are not assigned. They can only be registered with Internet Authority to prevent duplication.
- **Dynamic ports:** The ports ranging from 49,152 to 65,535 are neither controlled nor registered. They can be used by any process. These are the *ephemeral ports*.

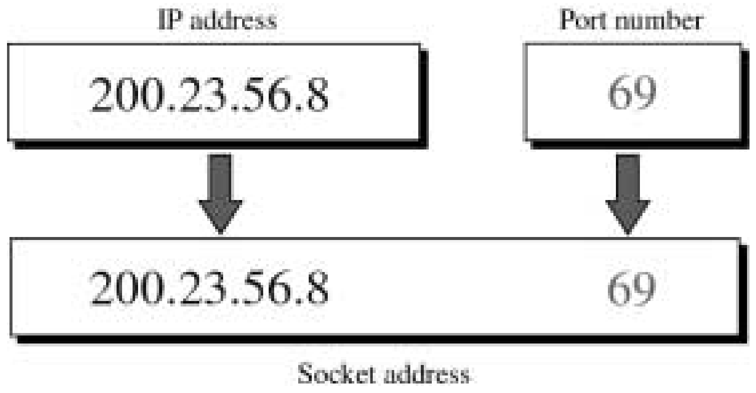
Well-known Ports for UDP

Some well-known port numbers used by UDP are listed below.

Port No.	Protocol	Description
7	Echo	Echoes a received datagram back to the sender
13	Daytime	Returns the date and the time
17	Quote	Returns
53	Name Server	Given domain name, returns IP address.
69	TFTP	Trivial File Transfer.
111	RPC	Remote Procedure Call
161	SNMP	Simple Network Management Protocol
162	SNMP	Simple Network Management Protocol – trap

3.3.1.2 Socket Addresses:

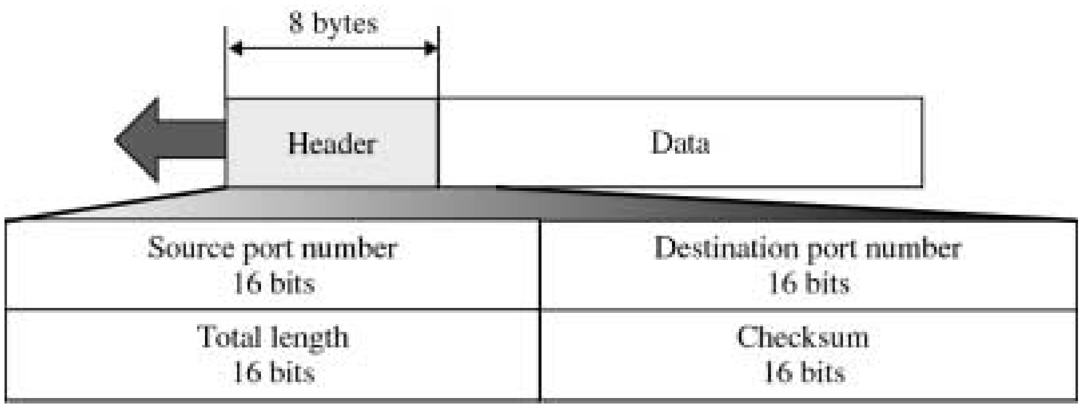
It is clear identifying the intended process has two components, the IP and address and port number. The combination of an IP address and a port number is called *socket address*. Socket address uniquely identifies a process on Internet, whereas IP address uniquely identifies a host on the Internet and the port number uniquely identifies process on a host system. The client socket address defines the client process uniquely and the server address defines the server uniquely.



To use services of UDP, one needs a pair of socket addresses: the client socket address and the server socket address. These four pieces of information are part of the IP header and the UDP header. The IP header contains the IP addresses; the UDP header contains the port numbers.

3.3.2 UDP Datagram:

UDP packets, known as *user datagrams*, have a fixed-size header of eight bytes. Figure shows the format of a user datagram.



Fields are explained below:

Source port number:

This is the port number of the *source process* sending the UDP datagram. It is 16 bits long (2 byte). If the source process is a client (a client sending a request), the port number, in most cases, is an *ephemeral port number*. If the source process is a server (a server sending a response), the port number, in most cases, is a *well-known port number*.

Destination port number:

This is the port number of the *destination process* to which this UDP datagram is meant. It is also 16 bits long. Its characteristics are similar to the Source Port Number

Length:

This is a 16-bit field that defines the total length of the UDP datagram, header plus data. The 16 bits can define a total length of 0 to 65,535 bytes. However, the minimum length is eight bytes, which indicates a user datagram with only header and no data. However it should be noted that IP has a limitation of 65,535 bytes for its datagram. Therefore, the length of the data can be between 0 and 65,507 (65,535 – 20 – 8) bytes (twenty bytes for IP header and 8 bytes for UDP header).

The UDP length can be computed using data length in the IP header. However, the designers of UDP protocol felt that it was more efficient for the destination UDP to calculate the length of the data from the information provided in the UDP datagram rather than asking to supply this information. We should remember that when the IP module delivers the UDP user datagram to the UDP layer, it has already dropped the IP header.

Checksum:

This field is used to detect errors over the entire user datagram (header plus data). Unlike lower layer protocols like IP, which provide checksum only for their header, this provides the checksum for entire datagram, which includes the data sent by the user process also.

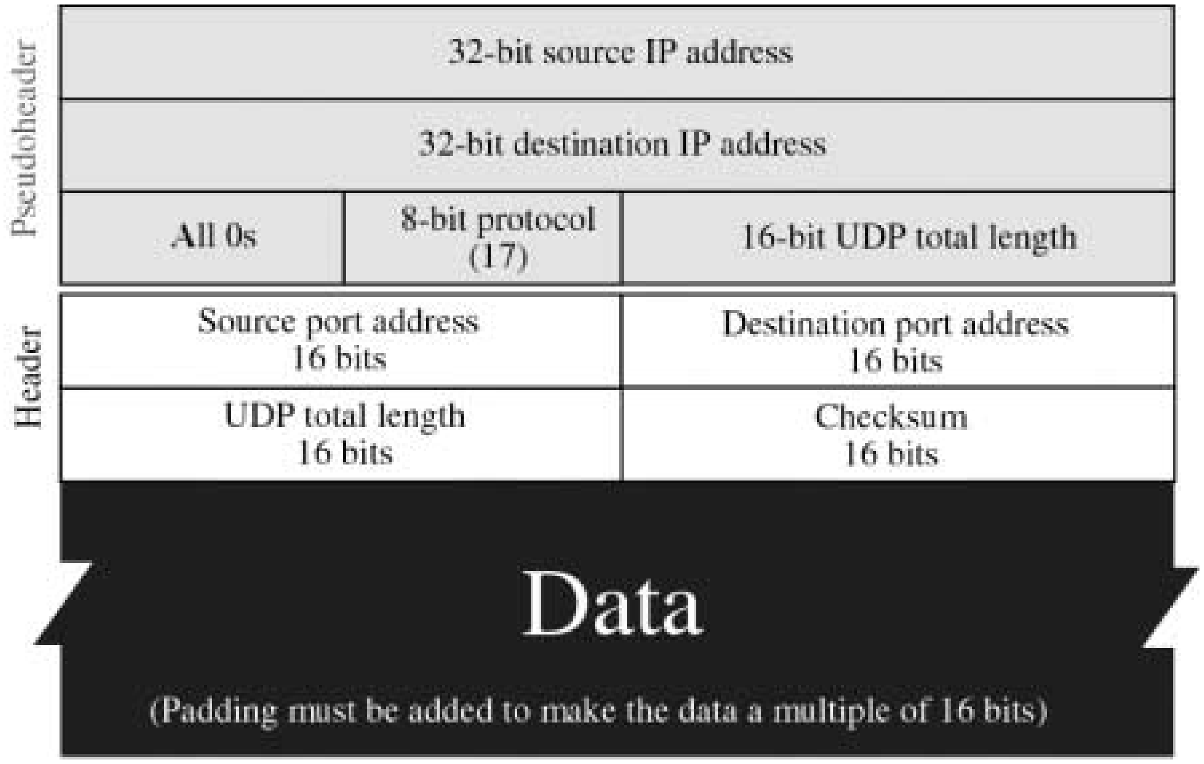
3.3.3 UDP Checksum Computation:

The arithmetic behind the Checksum computation is same as in the IP checksum computation. However UDP checksum calculation is different from the one for IP and ICMP. Here the checksum includes three sections:

- *Pseudo header:* The pseudo header is part of the header of the IP packet in which the user datagram is to be encapsulated for transmission with some fields with 0s
- *UDP header*: Header for this UDP datagram.
- *User Data:* Data sent by the upper layer.

The pseudoheader is added to ensure that the, user datagram reaches the intended process which uses the intended transport protocol on the intended host. In other words it should not only reach the correct process through the correct protocol on the correct destination host. Since in no other layer it is possible to check all these three, a pseudoheader derived from the IP header, which has destination and source IP address and protocol numbers, is used. The pseudoheader along with the UDP header uniquely identifies the destination process.

Figure below illustrates the fields used in checksum.


Pseudoheader
32-bit source IP address
32-bit destination IP address
All 0s
8-bit protocol (17)
16-bit UDP total length
Header
Source port address 16 bits
Destination port address 16 bits
UDP total length 16 bits
Checksum 16 bits
Data
(Padding must be added to make the data a multiple of 16 bits)

Checksum Calculation at the Source:

At the source host system, the sender follows these steps to calculate the checksum:

1. Add the pseudoheader to the UDP user datagram.
2. If the total number of bytes is not even, add one byte of padding (all 0s). The padding is only for the purpose of calculating the checksum and will be discarded afterwards.
3. Fill the checksum field with zeros.
4. Divide the total bits into 16-bit (two-byte) words sections.
5. Add all 16-bit sections using one's complement arithmetic.
6. Take ones complement of the result (change 0s to 1s and all original 1s to 0s), which is a 16-bit checksum number.

Once 16 bits *checksum* field is computed using the above method, it is filled into the *original UDP* header and the resulting UDP datagram is sent. It should be noted that the pseudoheader and the padding bits are not sent but only used during the computation of checksum.

Checksum Calculation at Destination Host:

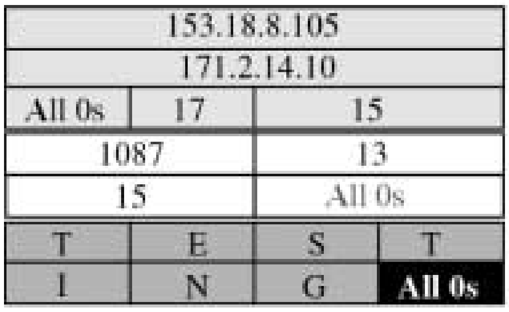
The receiver follows these steps to calculate the checksum:

1. Obtain the IP header, derive the pseudoheader from it, and add it to the UDP user datagram.
2. Add padding if needed.
3. Divide the total bits into 16-bit sections.
4. Add all 16-bit sections using one's complement arithmetic.
5. Take ones' complement the result to get the checksum.

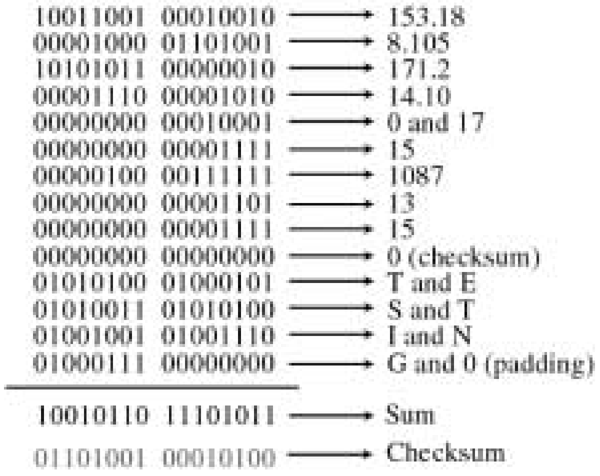
UDP datagram is accepted if the resulting checksum is all 0s. Otherwise it is discarded, as it indicates an error in it.

An Example

Figure shows the checksum calculation for a very small user datagram with only seven bytes of data. Because the number of bytes of data odd, padding is added for checksum calculation. This pseudoheader as well as the padding will be dropped when the user datagram is delivered to the IP.



153.18.8.105			
171.2.14.10			
All 0s	17	15	
1087		13	
15		All 0s	
T	E	S	T
I	N	G	All 0s



```
10011001 00010010 ——→ 153.18
00001000 01101001 ——→ 8.105
10101011 00000010 ——→ 171.2
00001110 00001010 ——→ 14.10
00000000 00010001 ——→ 0 and 17
00000000 00001111 ——→ 15
00000100 00111111 ——→ 1087
00000000 00001101 ——→ 13
00000000 00001111 ——→ 15
00000000 00000000 ——→ 0 (checksum)
01010100 01000101 ——→ T and E
01010011 01010100 ——→ S and T
01001001 01001110 ——→ I and N
01000111 00000000 ——→ G and 0 (padding)
_________________
10010110 11101011 ——→ Sum
01101001 00010100 ——→ Checksum
```

3.3.4 UDP Operation:

UDP uses concepts common to the transport layer. These concepts are discussed here briefly.

Connectionless Services

As mentioned earlier, UDP provides a connectionless service. This means that each user datagram sent by the UDP is an independent datagram. There is no relationship between the different user datagrams even if they *are coming from the same source process and going to the same destination program*. The user datagrams are not numbered. Also, there is no connection establishment and no connection termination at the beginning and end of a transaction. This means that each user datagram *can* travel a different path.

One of the ramifications of being connectionless is that the process that uses UDP cannot send a stream of data to UDP and expect to receive the same stream of data at the destination. Instead each request must be small enough to fit into one user datagram. Only those processes sending short messages should use UDP. Otherwise if needed, the end application should take care of section of data arriving out of order and reordering them to get the original stream.

Flow and Error Control

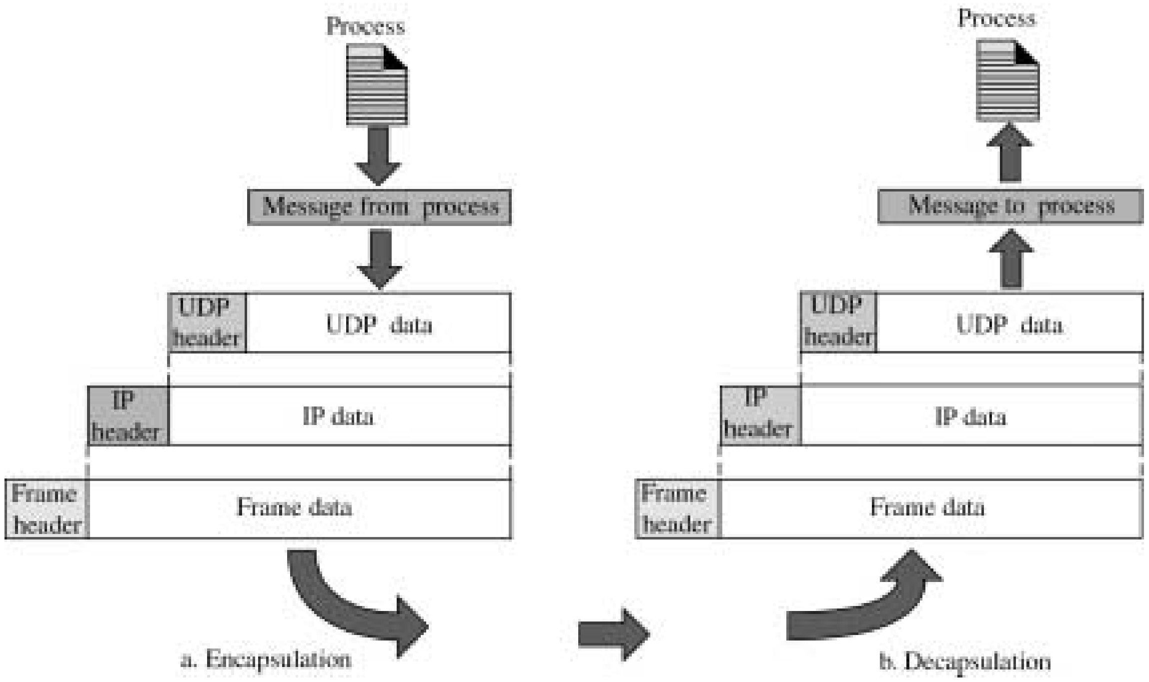
UDP is a very simple, unreliable transport protocol. There is no flow control, and hence no windowing mechanism. The receiver may overflow with incoming messages. Again it is left for the end application to take of it. The end applications which uses UDP either generates less data or they have inbuilt mechanism to take care of flow control.

There is no error control mechanism in UDP except for the checksum. This means that the sender does not know if a message has been lost or duplicated. When the receiver detects an error using the checksum, the user datagram is silently discarded.

The lack of flow control and error control means that the process using UDP should provide for these mechanisms.

Encapsulation ad Decapsulation:

To send a message from one process to another, the UDP protocol encapsulates and decapsulates the messages as shown in figure below.



Encapsulation

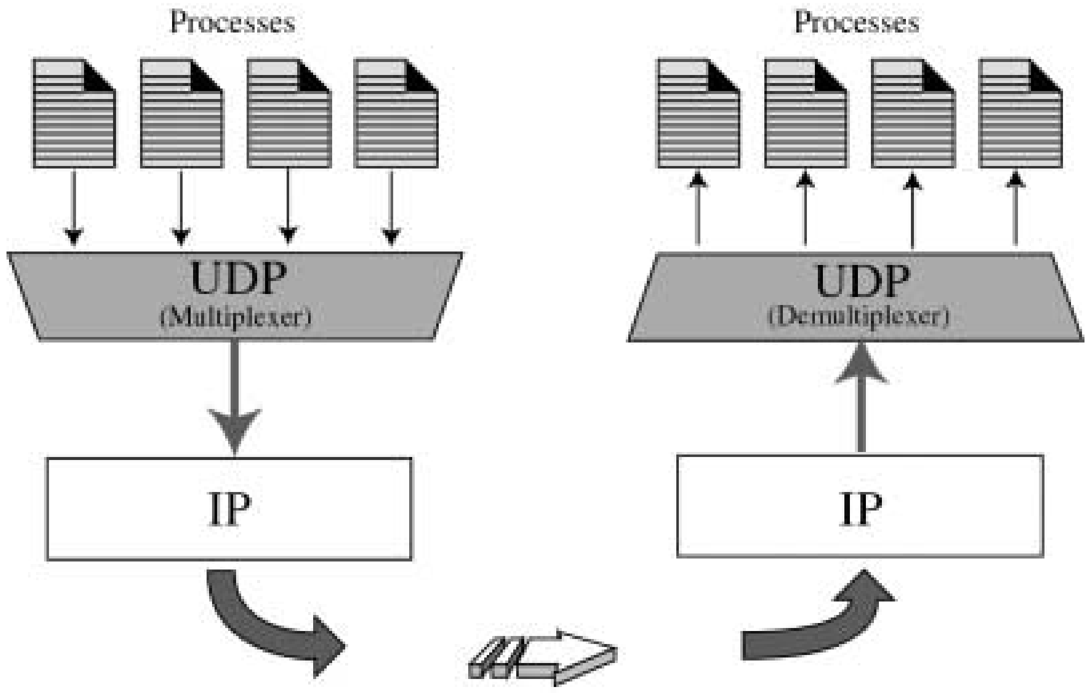
When a process has a message to send through UDP, it passes the message to UDP along with a pair of socket addresses and the length of data. UDP receives the data and adds the UDP header. UDP then passes the user datagram to the IP with the socket addresses. IP adds its own header, using the value 17 in the protocol field, indicating that the data has come from the UDP protocol. The IP datagram is then passed to the data link layer. The data link layer receives the IP datagram, adds its own header (and possibly a trailer), and passes it to the physical layer. The physical layer encodes the bits into electrical or optical signals and sends it to the remote machine.

Decapsulation

When the message arrives at the destination host, the physical layer decodes the signals into bits and passes it to the link layer. The data link uses the header (and the trailer) to check the data. If there is no error, the header and trailer are dropped and the datagram is passed to the IP. The IP software does its own checking. If there is no error, the header is dropped and the user datagram is passed to the UDP with the sender and receiver IP addresses. UDP uses the checksum to check the entire user datagram. If there is no error, the header is dropped and the application data along with the sender socket address is passed t the process. The sender socket address is passed to the process in case it needs to respond to the message received.

Multiplexing and Demultiplexing

In a host running a TCP/IP protocol, there is only one UDP but possibly several processes that may want to use the services of UDP. To handle this situation UDP uses the concept called *Multiplexing* and *Demultiplexing*.



Multiplexing: At the source host, there may be several processes that need to send user datagrams. However, there is only one UDP. This is a many-to-one relationship and requires multiplexing. UDP accepts messages from different processes, each message will have a port number assigned to it, which identifies the process which generated the message. After adding the header, UDP passes the user datagram to IP.

Demultiplexing: At the receiver site, there is only one UDP. However, we may have many processes that can receive user datagrams. This is a one-to-many relationship and requires Demultiplexing. UDP receives user datagrams from IP. After error checking and dropping of the header, UDP delivers each message to the appropriate process based on the port numbers.

3.3.5 Use of UDP:

The following lists some uses of the UDP protocol:

- UDP is suitable for a process that requires simple request – response communication and with little concern for flow and error control. It is not usually used for a process that needs to send bulk data needing error and flow control, such as FTP.
- UDP is suitable for a process with internal flow and error-control mechanisms. For example, the Trivial File Transfer Protocol (TFTP) process includes flow and error control. It can easily use UDP.
- UDP is suitable transport protocol for transport protocol for multicasting and broadcasting. Multicasting and broadcasting capabilities are embedded in the UDP software but not in the TCP software.
- UDP is used for management processes such as SNMP.
- UDP is used for some route updating protocols such as Routing Information protocol (RIP).

Note that these above processes can use the other transport protocol TCP also. However while designing the TCP/IP architecture the designers had the foresight one day the computers can be used for multimedia communication involving the voice and video data, which requires *transportation of huge data in real time* and error correction or reliability is not essential for these. This transporting huge data in real time means the overhead involved in the transport layer should be minimum, so that data transportation is fast and efficient. In fact with the advent of powerful desktop computer and high bandwidth communication networks, today there are several multimedia application.

Most importantly today's multimedia applications, such as VoIP (Voice over IP - using which one can even talk to another person, who has telephone, by making the telephone call through this protocol), and Video conferencing applications uses a protocol known as RTP (Real Time Protocol) for data transfer at the application layer. This RTP protocol uses the services of UDP. If it uses the services of TCP, which provides the reliability, connection oriented and stream service, the quality of the service suffers!

3.4 TCP – Transmission Control Protocol:

Apart from the IP, the protocol, which is responsible for widespread usage of TCP/IP model for computer-to-computer communication network, is the TCP. It is the combination of TCP at the transport layer and IP at the network layer, which is largely responsible for success of this model.

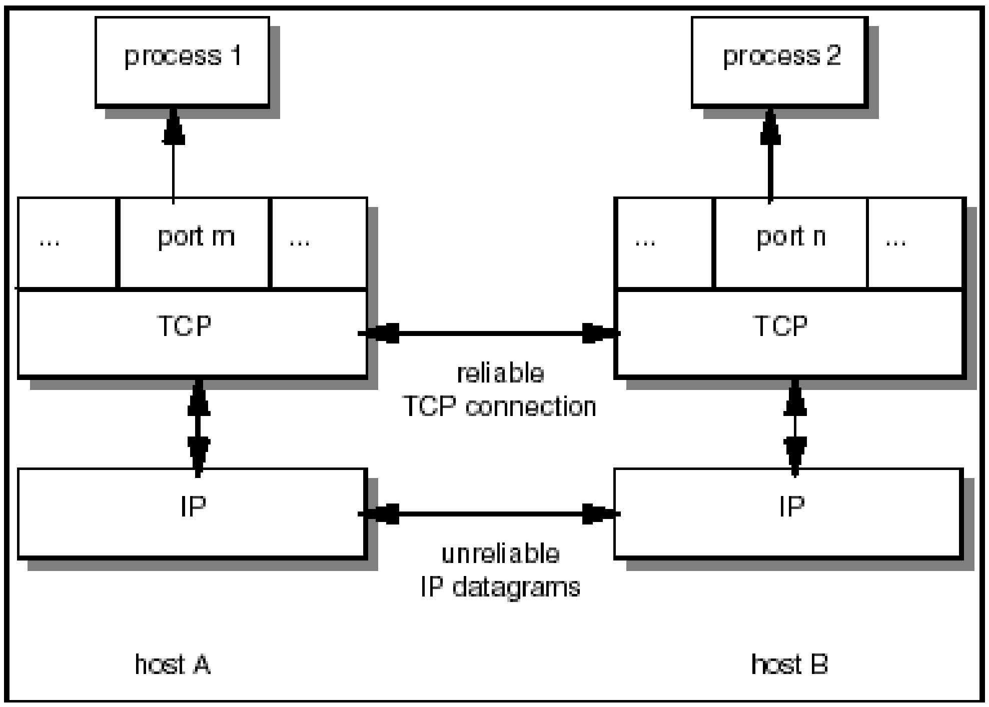
One of the reasons for a communication protocol suite to be successful is, need to be adopted by the end application developers. The network application programme, which needs to communicate over the network, expects a very simple, easy to implement communication channel. As for as sending or receiving data with another process is concerned, the details involved must be minimal. If one wants to write an application programme, which communicates with another over the network, he should not be expected to build logic into his programme to take care of so many issues that are involved in the communication. He should not worry, about things like, some part of the message getting corrupted, or lost. There are other issues which should not be handled by him, like rearranging the datagrams which arriver out of order, issues of congestion and resulting delay, issues of flow control by which the rate at which the source sends the data can be controlled.

In other words many of the application process needs easy to use communication channel, which provides the *reliable, stream oriented* communication service, which takes are of *error and flow control* also.

As we have seen earlier, IP provides an unreliable, best-effort, datagram service. And UDP adds just the process-to-process communication facility apart from error checking mechanism to some extent.

So there is a need for another protocol, which sits on top of the IP, and provides the services as mentioned above to the end application. So, the designer of TCP/IP model came out with TCP at the transport layer.

TCP provides *process-to-process* communication channel, which is *reliable* and *stream oriented* in nature. Also it takes care of the *error-control* and *flow control.*



The *process-to-process* mechanism is very similar to the one used by UDP. Port numbers are used to identify an individual process. Association of IP address and port number is known as the socket. Each communicating process is identified by a *socket.* And a pair of socket determines the communication channel. One important thing, which should be noted, is that port number can be shared by both UDP as well as TCP. i.e. there can be a port number which is assigned to two processes at the same host, *but which uses different protocols, one process uses UDP and another TCP*. Port number does not identify the protocol. That distinction is made by the *protocol* field in the IP header.

3.4.1 Services offered by TCP:

TCP can be characterized by the following facilities it provides for the applications using it:

Stream Data Transfer:

TCP provides stream data transfer service, which means the destination process receives the *stream of data* in exactly the same manner it is sent by the source process.

Unlike in datagram service, there is no concept of unit of data transfer. In datagram service, all the data have to be sent as a single unit. This imposes two restrictions. One is all the data should be available at the time of sending otherwise they will be sent in another datagram. Also each datagram unit is a separate entity and there is no relation between them. Another restriction is on the size of the data transfer.

Source TCP accepts a stream of characters from the sending application program *as and when they arrive*, creates packets, called segments, of appropriate size extracted from the stream, and sends them across the network. The receiving TCP receives segments, extracts data from them, orders them if they have arrived out of order, and delivers them as a stream of characters to the receiving application program.

For stream delivery, the sending and receiving TCPs use buffers. The sending TCP uses a sending buffer to store the data coming from the sending application program. The sending application program delivers data at the rate it is created. For example, if the user is typing the data on a keyboard, the data is delivered to the sending TCP character by character. If the data is coming from a file, data may be delivered to the sending TCP line-by-line, or block-by-block. The sending application program writes data to the buffer of the sending TCP. However, the sending TCP does not create a segment of data for each write operation issued from the sending application programme. TCP may choose to combine the result of several write operations into one segment to make transmission more efficient.

The receiving TCP receives the segments and stores them in a receiving buffer. The receiving application program uses the read operation to read the data from the receiving buffer, but it does not have to read all of the data contained in one segment in one operation. Since the rate of reading can be slower than the rate of receiving, the data is kept in the buffer until the receiving application reads if completely.

Full-Duplex Service:

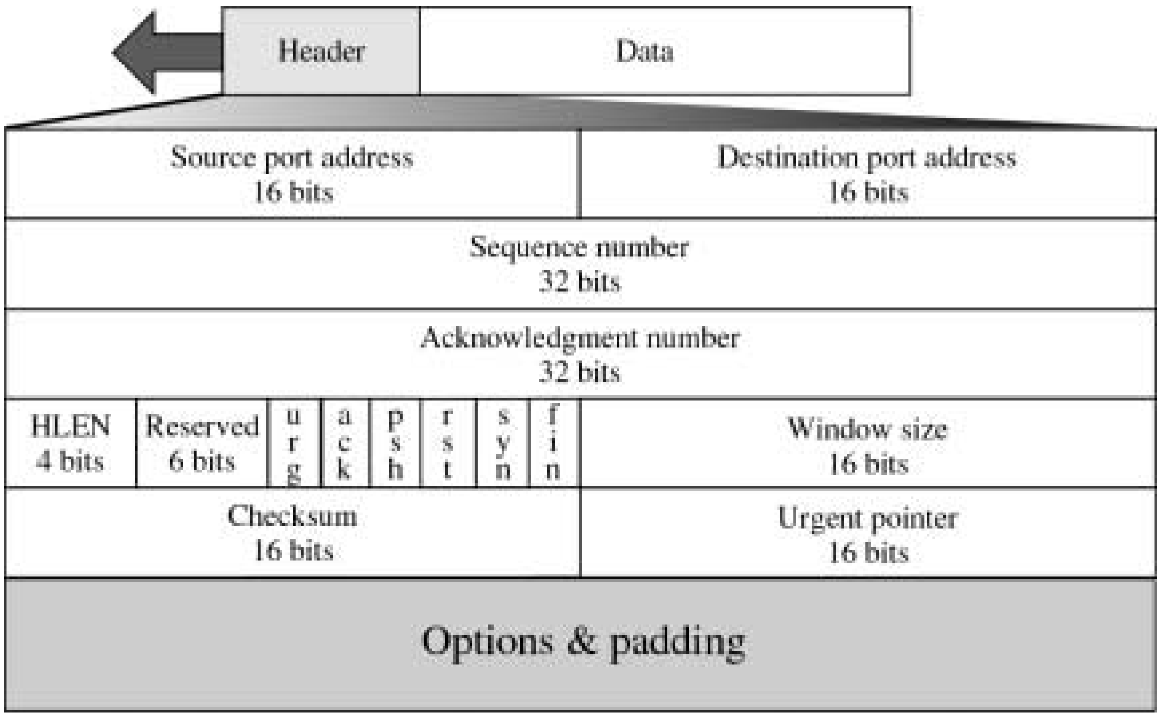
TCP offers *full-duplex service*, where data can flow in both directions at the same time. After two application programs are connected to each other, they can both send and receive data. One TCP connection can carry data from application A to B and, at the same time, data from B to A. When a packet is going from A to B, it can also carry an acknowledgement of the packets received from B. Likewise, when a packet is going from B to A, it can also carry an acknowledgement of the packets received from A. This is called piggybacking because acknowledgements can be sent with data.

Reliable Service:

TCP is reliable transport protocol. It uses the acknowledgement mechanism to ensure that no part of the data is lost or corrupted or duplicated. Details of how this achieved is discussed in later section.

3.4.2 TCP Segment:

In IP a unit of data that is transmitted at once is called *IP datagram.* In UDP, also it is referred to as datagram – *UDP Datagram*. But in TCP, a unit of data transfer is known as *TCP segment.* Since TCP provides the *stream-oriented data service* it uses the concept of *segment* for a unit of transfer.



The segment consists of a 20-to 60-byte header, followed by data from the application program. The header is 20 bytes if there are no options and up to 60 bytes if it contains some options. Some of the header fields are discuss in this section.

Source port address:

This is a 16-bit (2 bytes) field that defines the port number of the application program in the host that is sending the segment. This serves the same purpose as the source port address in the UDP header.

Destination port address:

This is a 16-bit (2 bytes) field that defines the port number of the application program in the host that is receiving the segment.

Sequence number:

This 32-bit (4 bytes) field defines the number assigned to the first byte of data contained in this segment. As we said before, TCP is a stream transport protocol. To ensure connectivity, each byte to be transmitted is numbered. The sequence number tells the destination the position of the first byte of this segment, in the original stream of data at the source.

Acknowledgement number:

This 32-bit (4 bytes) field defines the byte number that the source of the segment is expecting to receive from the other end process. If host has received successfully till byte number n from the other host, then it defines $n + 1$ as the acknowledgement number, which indicates it is expecting data starting from location $n+1$ at the other hosts' stream.

Header length:

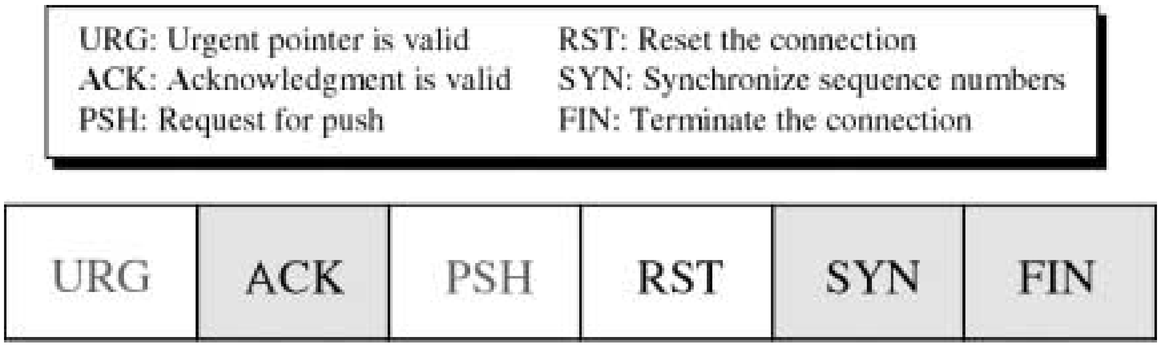
This four-bit field indicates the length of the TCP header. It is equal to number of four-byte words in the TCP header. The length of the header can be between 20 and 60 bytes. Therefore, the value of this field can be between 5 (5 x 4 = 20) and 15 (15 x 4 = 60).

Reserved:

This is a six-bit field reserved for future use.

Control:

This field defines six different control bits or flags as shown in figure. One or more of these bits can be set at a time.


URG: Urgent pointer is valid
ACK: Acknowledgment is valid
PSH: Request for push
RST: Reset the connection
SYN: Synchronize sequence numbers
FIN: Terminate the connection
URG
ACK
PSH
RST
SYN
FIN

These bits enable flow control, connection establishment and termination, and the mode of data transfer in TCP. A brief description of each bit is given below.

URG: The value of urgent pointer is valid.

ACK: The value of acknowledgement field is valid.

PSH: Request to push the data

RST: Request to reset the connection.

SYN: Request to synchronize the sequence number during connection

FIN: Request to terminate the connection.

Window size:

This field defines the size of the window, in bytes, that the other end must maintain. Note that the length of this field is 16 bits, which means that the maximum size of the window is 65,535 bytes. This is discussed in detail later.

Checksum:

This 16-bit (2 bytes) field contains the checksum, used for error checking purpose. The checksum method employed is same as the one used in UDP.

Urgent pointer:

This 16-bit field, which is valid only if the urgent flag is set, is used when the segment contains urgent data. It defines the number that must be added to the sequence number to obtain the number of the last urgent byte in the data section of the segment. This will be discussed later.

Options:

There can be up to 40 bytes of optional information in the TCP header.

3.4.3 Connection in TCP:

TCP is a connection-oriented protocol. A connection-oriented protocol establishes a virtual path between the source and the destination processes. All of the segments belonging to a message are then sent over this virtual path. Having a single virtual pathway for the entire message facilities, the acknowledgement process as well as the retransmission of corrupted or lost frames.

In TCP, connection-oriented transmission is achieved through two procedures: *connection establishment* and *connection termination.*

Connection Establishment:

TCP transmits data in full-duplex mode. When two TCP modules in two host systems are connected, they should be able to send segments to each other simultaneously. This implies that before any data transfer, each process must initialize communication and get approval from the other process. Four actions needs to be taken before the two processes, on host A and host B respectively, can send data:

- Host A sends a segment to announce its wish for connection and includes its initialization information about the traffic from A to B.
- Host B sends a segment to acknowledge (confirm) the request of A.
- Host B sends a segment that includes its initialization information about the traffic from B to A.

- Host A sends a segment to acknowledge (confirm) the request B.

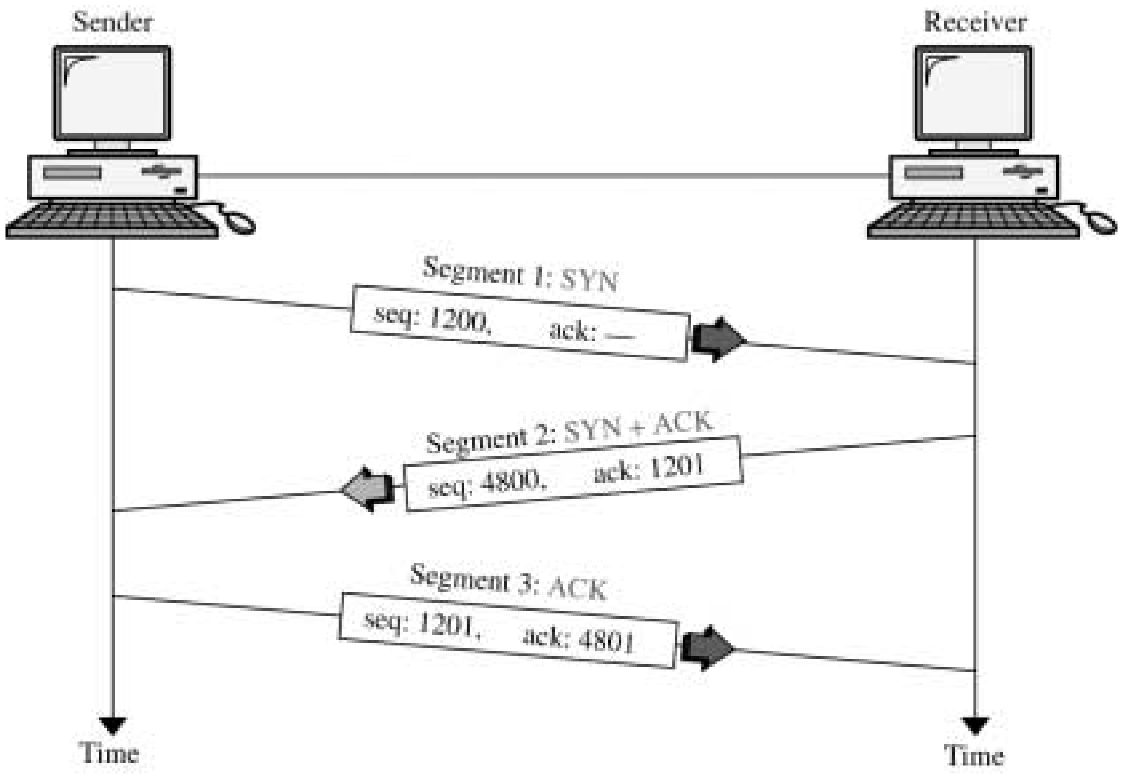
At the successful completion of last step a connection is established. However Host B can both confirm the request of host A and send its own request in a single segment.

Three-way Handshaking:

The connection establishment described above is called three-way handshaking. In this procedure, an application program, called the client, wants to make a connection with another application program, called the server, using TCP as the transport layer protocol.

The three-way handshaking procedure starts with the server. The server program tells its TCP that it is ready to accept a connection. This is called a request for a passive open.

The client program makes a request for an active open. A client that wishes to connect to a server tells its TCP that it needs to be connected to a particular server. The clients TCP can now start the three-way handshaking process as shown in Figure.



The steps of the process are as follows :

The client sends the first segment, with the SYN bit set, which is known as SYN segment. The segment includes the source and destination port numbers. The destination port number clearly defines the server to which the client wants to be connected. The segment also contains the client *Initialization Sequence Number* (ISN) used for numbering the bytes of data sent from the client to the server. If the client needs a large window, it defines the window scale factor here using the appropriate option. This segment defines the wish of the client to make a connection with certain parameters.

The server sends the second segment, a SYN and ACK segment. This segment has a dual purpose. First, it acknowledges the receipt of the first segment using the ACK flag and acknowledgement number field. The acknowledgement is the client initialization sequence number plus one. The server must also define the client window size. Second, the segment is used as the initialization segment for the server. It contains the initialization segment for the server and initialization sequence number used to number the bytes sent from the server. As mentioned before, this is two segments combined into one.

The client sends the third segment. This is just an ACK segment. It acknowledges the receipt of the second segment using the ACK flag and acknowledgement number field. The acknowledgement number is the server initialization sequence number plus one. The client must also define the server window size. Note that data can be sent with the third packet.

Connection Termination:

Any of the two end processes involved in exchanging data (client or server) can close the connection. When connection in one direction is terminated, the other process can continue sending data in the other direction. Therefore, four actions are needed to close the connections in both directions:

- Host A sends a segment announcing its wish for connection termination.
- Host B sends a segment acknowledging (confirming) the request of A. After this, the connection is closed in A to B direction, but not in the other. Host B can continue sending data to A.

- When host B has finished sending its own data, it sends a segment to indicate that it wants to close the connection.
- Host A acknowledges (confirms) the request of B.

This implies four steps. Steps 2 and 3 here cannot be combined as in connection establishment. Steps 2 and 3 may or may not happen at the same time. The connection may be closed in one direction, but left open in the other direction.

Four-Way Handshaking:

The connection termination described above is called four-way handshaking. In this procedure, an application program, usually the client, wants to terminate a connection.

- The procedure starts with the client. The client program tells its TCP that it has finished sending data and wishes to terminate the connection. This is a request for an active close.
- After receiving the request for an active close, the client TCP closes communication in the client to server direction. However, communication in the other direction is still open.
- When the server program has finished sending data in the server-client direction, it can request from its TCP to close the connection in the server-client direction. This is usually a passive close.

Connection Resetting:

TCP may request the resetting of a connection (by setting the RST flag bit). Resetting here means that the current connection to be destroyed. This happens in one of three cases:

The TCP on one side has requested a connection to a nonexistent port. The TCP on the other side may send a segment with its RST bit set, to annual the request.

One TCP may want to abort the connection due to an abnormal situation. It can send an RST segment to close the connection.

The TCP on one side may discover that the TCP on the other side is idle for a long time. It may send an RST segment to destroy the connection.

3.4.4 Flow Control mechanism in TCP:

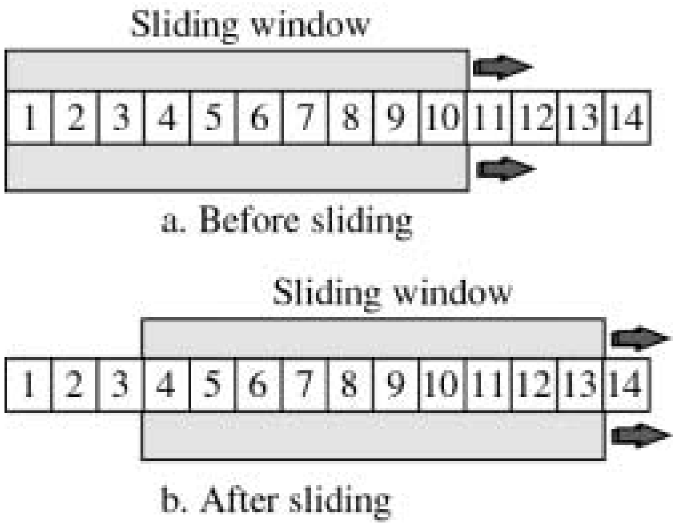
Flow control defines the amount of the data a source can send before receiving an acknowledgement from the destination. In an extreme case, a transport layer protocol could send one byte of data and wait for an acknowledgement before sending the next byte. But this is an extremely slow process. If the data is traveling a long distance, the source is idle while it waits for an acknowledgement.

At the other extreme, a transport layer protocol can send all of the data it has without worrying about acknowledgement. This speeds up the process, but it may over whelm the receiver. Besides, if some part of the data is lost, duplicated, received out of order, or corrupted, the source will not know until all has been checked by the destination.

TCP uses a solution that stands somewhere in between. It defines a window, which is imposed on the buffer of data delivered from the application program and is ready to be sent. TCP sends as much data as is defined by the window.

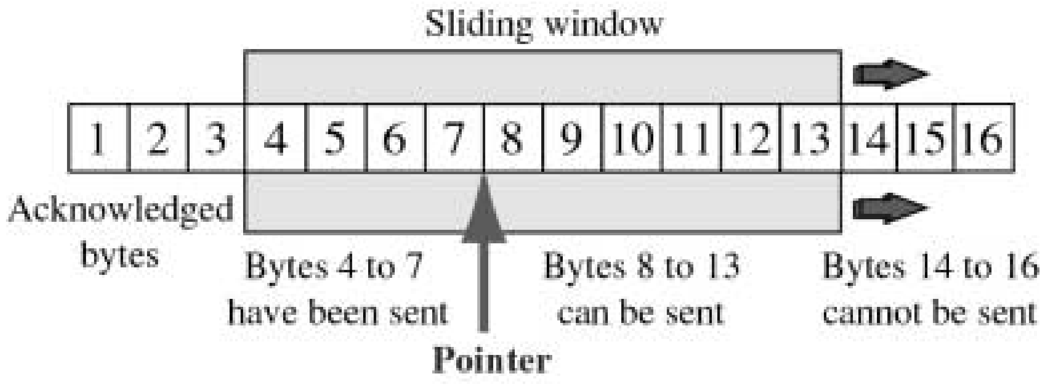
Sliding Window:

To accomplish flow control, TCP uses a sliding window protocol. With this method, both hosts use a window for each connection. The window covers a portion of the buffer that a host can send before worrying about an acknowledgement from the other host. The window is called a sliding window because it slides over the buffer as the receiver sends acknowledgement of the bytes received without any error. Figure shows a sliding window of size 10.



Before receiving any acknowledgement from the destination, the source can send up to 10 bytes. However, if it receives acknowledgement of the first three bytes, it can slide the window three bytes to the right. This means that now it cans end 10 more bytes before worrying about an acknowledgement.

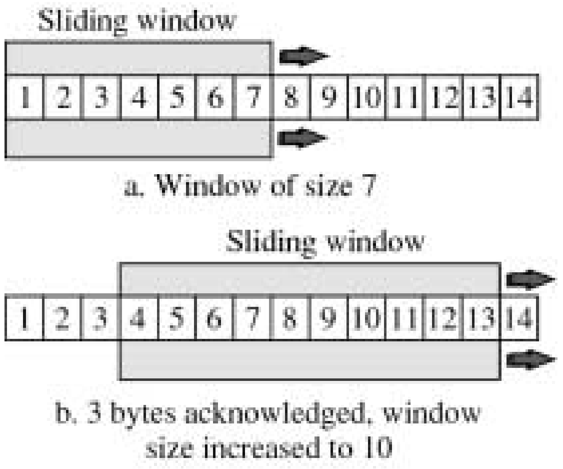
The following Figure shows the previous window, but a pointer is added which the source uses to know which bytes have already been sent and which ones can be sent.


Sliding window
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
Acknowledged bytes
Bytes 4 to 7 have been sent
Bytes 8 to 13 can be sent
Bytes 14 to 16 cannot be sent
Pointer

The previous example shown is a fixed size window. However *flow-control* needs the size of the window to vary to achieve the flow control. Hence in TCP window size is variable. The destination, in each acknowledgement segment, can define the size of the window. The advertised size is relative to the acknowledgement number. For example, if the receiver acknowledges the receipt of byte 3,000 and defines the size of the window to be 200, it means that the window now expands from byte 3001 to byte 3,200.

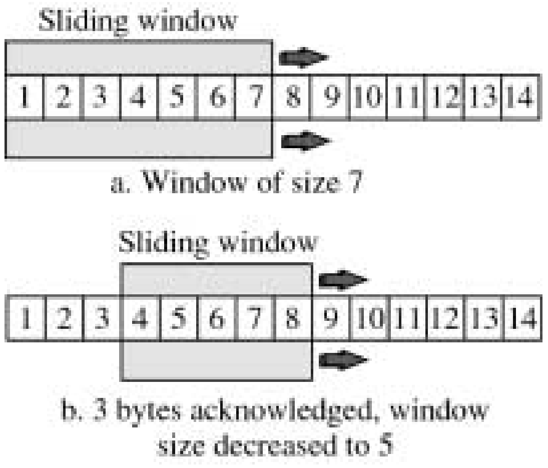
Increasing the Window Size:

The destination can increase the size of the window in an acknowledgement segment. The figure shows how window can slide with a simultaneous increase in size.



Decreasing the Window size:

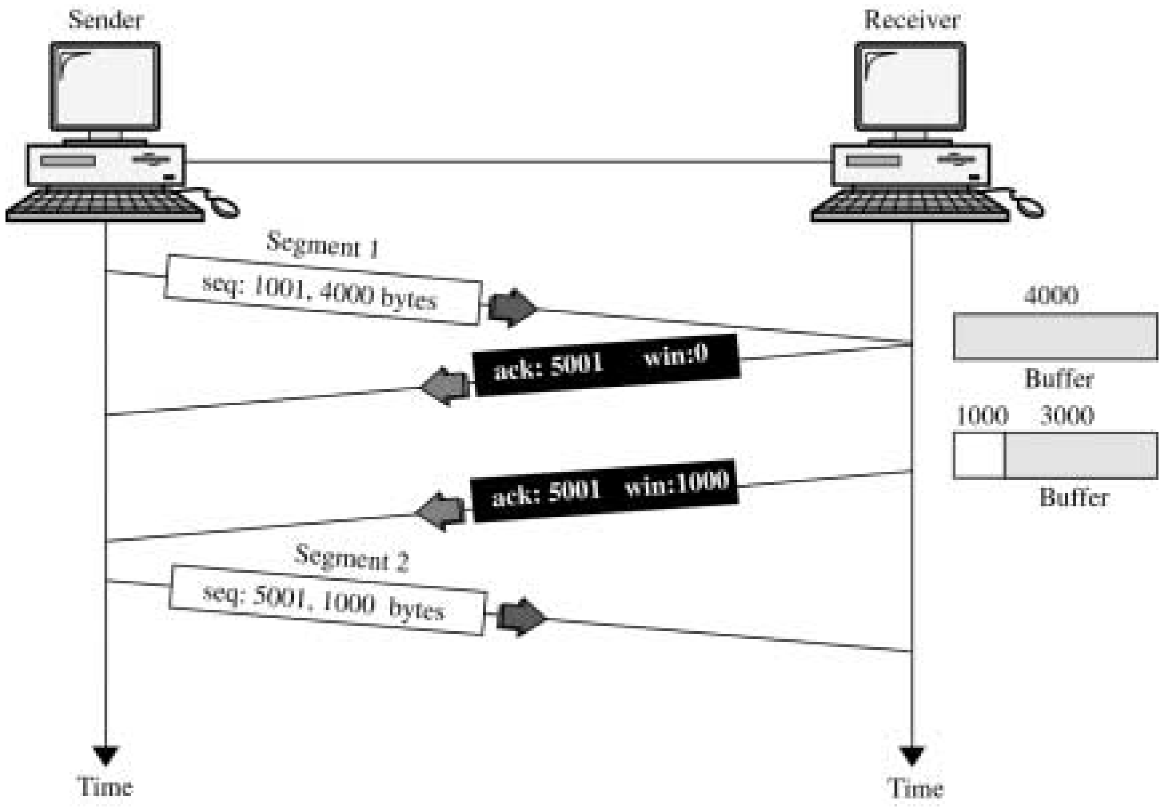
The destination can decrease the size of the window in an acknowledgement segment. Figure shows how the window can slide with a simultaneous decrease in size. However, there is a restriction: The window size cannot decrease in such a way that the leading edge of the window slides to the left.



Window Management:

TCP uses two buffers and one window to control the flow of data. The sending TCP has a buffer that stores data coming from the sending application program. The application program creates data and writes it to the buffer. The sender imposes a window on this buffer and sends segments as long as the size of the window is not zero. The TCP receiver has a buffer also. It receives data, checks them, and stores them in the buffer to be consumed by the receiving application program.

As mentioned earlier the size of the window in the sending TCP is determined by the receiver and is announced in the ACK segments. How does the receiver choose the size of the window? The size of the window, announced by the receiver, is usually the space left over in the receiving TCP buffer. Figure shows an example.



Consider that the sending TCP has originally defined a very large buffer. The receiving TCP has defined a buffer of size 4000 bytes. During the connection setup, the receiving window announces the size of window to be 4000, the same size as its buffer.

The sending TCP sends 4000 bytes of data in its first segment. The buffer of the receiving window becomes full. The receiving TCP acknowledges the receipt of the segment, but announces a window size of zero. The sending TCP cannot send any more data. It must wait for acknowledgement advertising a nonzero window size.

At the receiver, the application program consumes 1000 bytes of data, resulting in 1000 bytes of available buffer space. The receiving TCP sends a new acknowledgement with a window size of 1000. The sender can now send a segment of 1000, which fills up the buffer. And so on.

Note that by advertising an appropriate size of the window, receiving TCP can control the rate at which the sending TCP transmits the data, thus achieving the flow-control.

3.4.5 Error Control mechanism in TCP:

TCP is a reliable transport layer protocol. This means that an application program that delivers a stream of data to TCP relies on TCP to deliver the entire stream to the application program on the other end in order, without error, and without any part lost or duplicated.

TCP provides reliability using error control. Error control includes mechanisms for detecting corrupted segments, lost segments, out-of-order segments, and duplicated segments. Error control also includes a mechanism for correcting errors after they are detected.

Error Detection and correction:

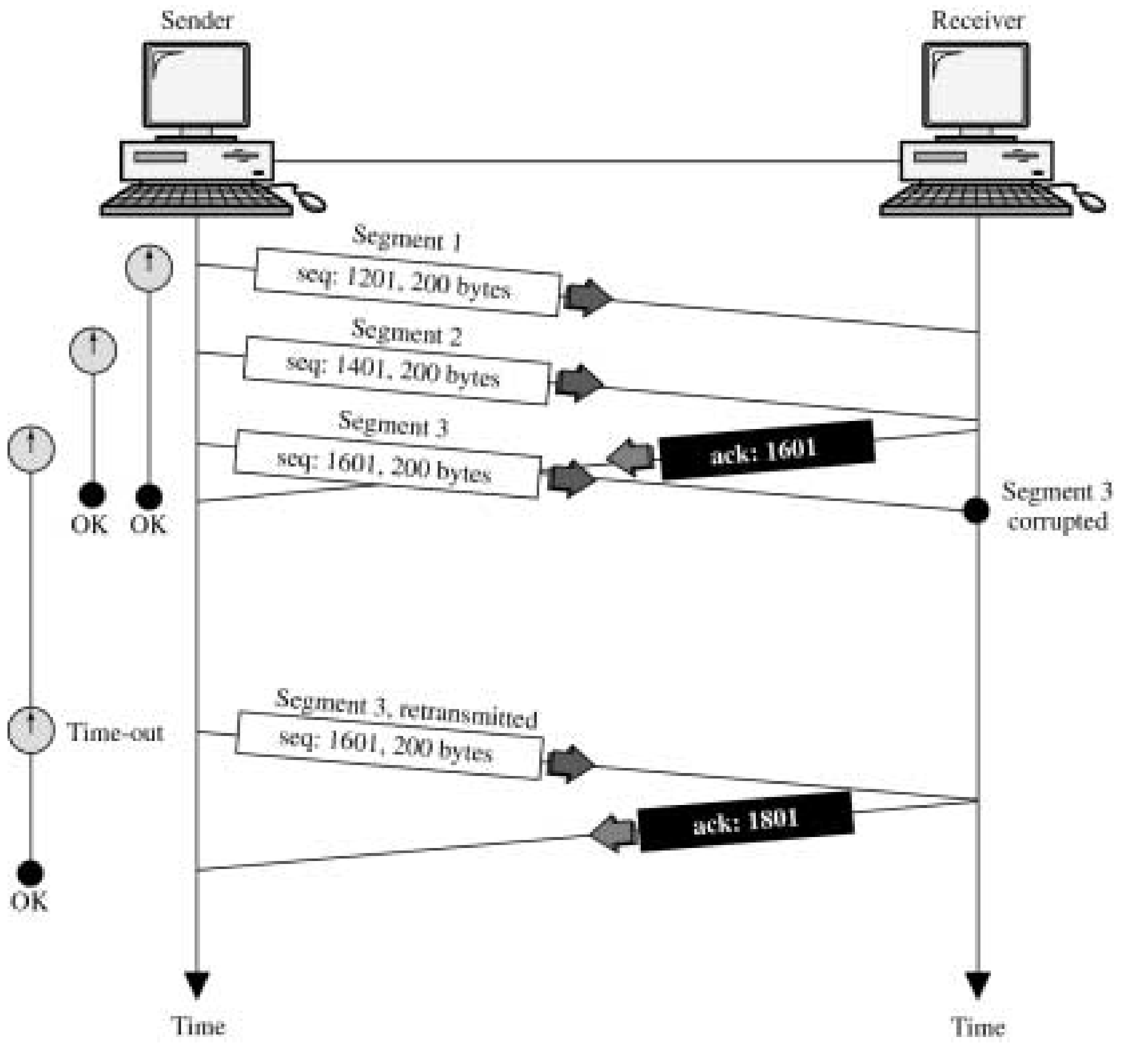
Error detection in TCP is achieved through the use of three simple tools: checksum, acknowledgement, and time-out. Time-out is the duration for which the sending TCP waits for the acknowledgement, after which it considers the data sent is either corrupted or lost and retransmits the same.

Each segment includes the checksum field, which is used to check for a corrupted segment. By computing checksum, one can find whether the segment is corrupted or not. If it is corrupted, it is discarded by the destination TCP. TCP uses the acknowledgement method to confirm the receipt of those segments that have reached the destination TCP uncorrupted. If a segment is not acknowledged before the time-out, it is considered to be either corrupted or lost.

The error-correction mechanism used by TCP is also very simple. The source TCP starts one time-out counter for each segment sent. Each counter is checked periodically. When a counter expires, the corresponding segment is considered to be either corrupted or lost, and the segment will be retransmitted.

Corrupted segment:

Figure shows a corrupted segment arriving at the destination.



In this example the source sends segments 1 through 3, each 200 bytes. The sequence number begins at 1,201 on segment 1. The receiving TCP receives segments 1 and 2, using the checksum, finds them error free. It acknowledges the receipt of segments 1 and 2 using acknowledgement number 1,601, which means that it has received bytes 1,201 to 1,600 safe and sound, and is expecting to receive byte 1,601. However, it finds that segment 3 to be corrupted and discards segment 3. Note that although it has received bytes 1,601 to 1,800 in segment 3, the destination does not consider this as a receipt because this segment was corrupted. After the time-out for segment 3 expires, the source TCP will resend segment 3. After receiving segment 3, the destination sends the acknowledgement for byte 1,801, which indicates that it has received bytes 1,201 to 1,800 error free.

Lost Segment:

The situation is exactly the same as the corrupted segment. In other words, from the point of the source and destination, a lost segment and a corrupted segment are the same. A corrupted segment is discarded by the final destination; a lost segment is discarded by some intermediate node and never reaches the destination.

Duplicate Segment:

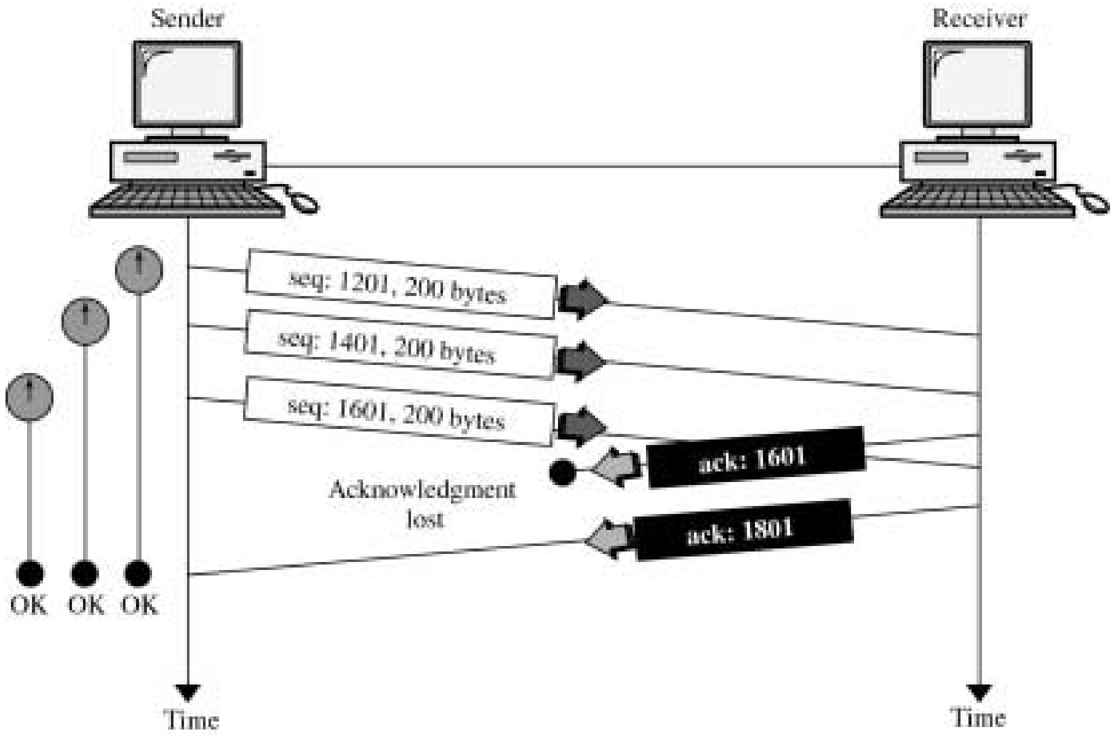
A duplicate segment can be created, for example, by a source TCP when the acknowledgement does not arrive before the time-out. Handling the duplicated segment is a simple process for the destination TCP. The destination TCP expects a continuous stream of bytes. When a packet arrives that contains the same sequence number as another received segment, the destination TCP simply discards the packet.

Out-of-Order Segment:

TCP uses the services of IP, an unreliable, connectionless network layer protocol. The TCP segment is encapsulated in an IP datagram. Each datagram is an independent entity. The routers are free to send each datagram through any route they find suitable. One datagram may follow a route with a short delay; another may follow another route with a longer delay. If datagrams arrive out of order, the TCP segments that are encapsulated in the datagrams will be out of order as well. The handling of out-of-order segments by the destination TCP is very simple; It does not acknowledge an out-of-order segment until it receives all of the segments that precede it. Of course, if the acknowledgement is delayed, the timer of the out-of-order segment may mature at the source TCP and the segment may be resent. The duplicates then will be discarded by the destination TCP.

Lost Acknowledgement:

Figure shows a lost acknowledgement sent by the destination.



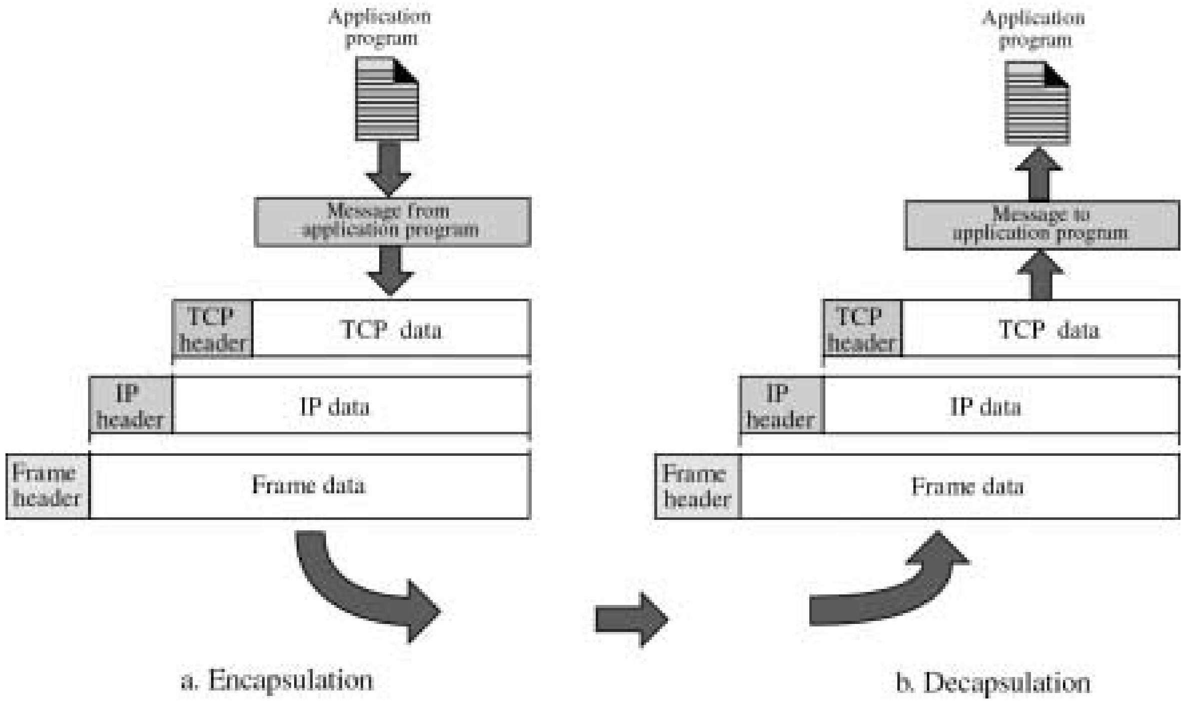
In the TCP acknowledgement mechanism, a lost acknowledgement may not even be noticed by the source TCP. TCP uses an accumulative acknowledgement system. Each acknowledgement is a confirmation that everything up to the byte specified by the acknowledgement number has been received. For example, if the destination sends an ACK segment with an acknowledgement number for byte 1,801, it is confirming that bytes 1,201 to 1,800 have been received. If the destination had previously sent an acknowledgement for byte 1,601, meaning it has received bytes 1,202 to 1,600, loss of the acknowledgement is totally irrelevant.

3.4.6 TCP operation:

Basic operations that are involved in the transport layer are common to both UDP and TCP. However because of the stream-oriented service it has few extra operations which are necessary for end application. All the operations are explained.

Encapsulation and Decapsulation:

To send a message from an application program to another, the TCP protocol encapsulates and decapsulates messages. The figure shows the Encapsulation and Decapsulation operation.



Encapsulation

When a process has a message to send through TCP, it passes the message to TCP and the length of data. TCP receives the data and adds the TCP header. TCP then passes the user datagram to the IP with the socket addresses. IP adds its own header. The IP datagram is then passed to the data link layer. The data link layer receives the IP datagram, adds its own header (and possibly a trailer), and passes it to the physical layer.

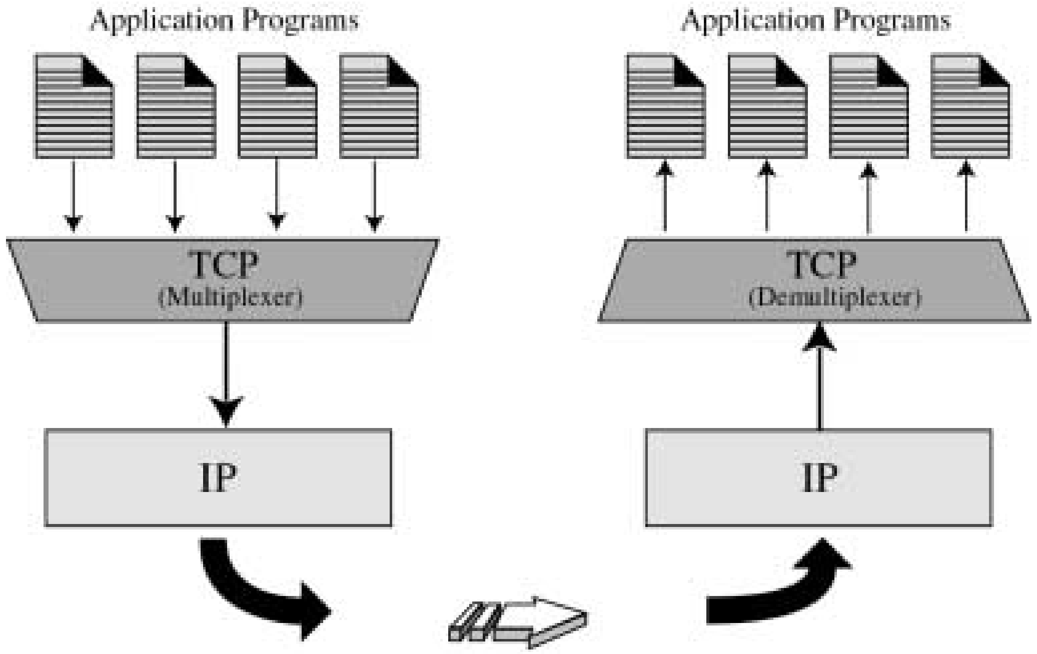
The physical layer encodes the bits into electrical or optical signals and sends it to the remote machine.

Decapsulation

When the message arrives at the destination host, the physical layer decodes the signals into bits and passes it to the link layer. The data link uses the header (and the trailer) to check the data. If there is no error, the header and trailer are dropped and the datagram is passed to the IP. The IP software does its own checking. If there is no error, the header is dropped and the user datagram is passed to the TCP with the sender and receiver IP addresses. TCP uses the checksum to check the entire user datagram. If there is no error, the header is dropped and the application data is passed to the process.

Multiplexing and Demultiplexing:

The multiplexing and demultiplexing operation is similar to the one for UDP.



Multiplexing: At the source host, there may be several processes that need to send user datagrams. However, there is only one TCP. This is a many-to-one relationship and requires multiplexing. TCP accepts messages from different processes, each message will have a port number assigned to it, which identifies the process which generated the message. After adding the header, TCP passes the user datagram to IP.

Demultiplexing: At the receiver site, there is only one TCP. However, we may have many processes that can receive user datagrams. This is a one-to-many relationship and requires demultiplexing. TCP receives user datagrams from IP. After error checking and dropping of the header, TCP delivers each message to the appropriate process based on the port numbers.

Pushing Data:

We saw that the sending TCP uses a buffer to store the stream of data coming from the sending application program. The sending TCP has the choice to create segments of any size from the stream. The receiving TCP also buffers the data when they arrive and delivers them to the application program when the application program is ready or when the receiving TCP feels that it is convenient. This type of flexibility increases the efficiency of TCP.

However, there are occasions in which the application program is not comfortable with this flexibility. For example, consider an application program that communicates interactively with another application program on the other end. The application program on one site wants to send a keystroke to the application at the other site and receive an immediate response. Delayed transmission and delayed delivery of data may not be acceptable by the application program.

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

Although the push operation can be dictated by the application program, today most implementations ignore such requests. TCP has the choice to use this operation or not.

Urgent Data:

TCP is a stream-oriented protocol. This means that the data is presented from the application program to the TCP as a stream of characters. Each byte of data has a position in the stream. However, there are occasions in which an application program needs to send urgent bytes. This means that the sending application program wants a piece of data to be read out of order by the receiving application program. Suppose that the sending application program is sending data to be processed by the receiving application program. When the result of processing comes back, the sending application program finds that everything is wrong. It wants to abort the process, but it has already sent a huge amount of data. If it issues an abort command (Control + C), these two characters will be stored at the end of the receiving TCP buffer. It will be delivered to the receiving application program after all the data has been processed.

The solution is to send a segment with the URG bit set. The sending application program tells the sending TCP that the piece of data is urgent. The sending TCP creates a segment and inserts the urgent data at the beginning of the segment. The rest of the segment can contain normal data from the buffer. The urgent pointer field in the header defines the end of the urgent data and the start of normal data.

When the receiving TCP receives a segment with the URG bit set, it extracts the urgent data from the segment, using the value of the urgent pointer, and delivers it, out of order, to the receiving application program.

3.5 Summary:

Transport Layer uses the services provided by the Network layer, for moving the data from source host to destination host and provides the mechanism for *process-to-process* communication. It has two protocols defined, namely UDP and TCP. Apart from process-to-process communication it has the responsibility of providing the reliability, stream-oriented data service to the end applications.

UDP, adds the process-to-process facility to IP service and provides the Datagram service to the application programme. Though it does not reliable data transfer, because of its simplicity it is efficient in data transfer. It is especially useful, in applications where the reliability is not very important criteria as loss of data to some extant is tolerable, but efficiency (i.e. the speed of data transfer) is most important. The multimedia application falls into this kind of category and they use UDP for their service.

The success of TCP/IP model, to some extant lies with the *reliable, stream-oriented* data transfer service offered by the transport layer through the TCP. This offers a simple to use, process-to-process data communication mechanism to the application layer programmes. This protocol takes care of the *error-control* and *flow-control* in the data stream.

Many processes can use the service of the UDP and TCP simultaneously. This is achieved by the use of unique numbers to identify the process which is called as **port numbers** and using the concept called multiplexing and demultiplexing.

UNIT 4

Application Layer

Contents:

4.1 Introduction:

The layers in TCP/IP do not correspond exactly to the OSI layers. One of the reasons could be the TCP/IP protocol suite was designed before the OSI model. The application layer in TCP/IP is equivalent to the combined session, presentation, and application layers of the OSI model. This means that all of the functionalities associated with those three layers are handled in one single layer, the application layer. But it should be noted that not many applications requires the services offered by the session and presentation layers. Applications, which require those services, would handle the same.

In other words, every application program must include all required tasks of the session, presentation, and application layers of the OSI model. This has some advantages and some disadvantages. One advantage is that each application program is independent. It requires only those functions needed for the job for which the application is designed. This saves needless calls through services that just pass parameters. One disadvantage is that the same tasks appear in different application programs, making them more complex. In addition, this kills the whole idea of modularity and layered architecture of the OSI model.

It is important to understand how exactly the end application processes communicate using the data transfer service offered by the lower layer protocols such as TCP. A common model, known as the *client-server* model is used to explain, how data transfer takes from one process to another to provide the service offered by the server programme. The concept of process is also explained.

To complete the study, an example network application is explained. The application explained is FTP (File Transfer Protocol), which is used to transfer *files* from one host computer to another over the network.

4.2 Objectives:

In this Unit you would learn about

- Issues involved in the application layer using the Client Server Model
- Different types of Clients and Servers
- Concept of Process

- An example Network application – FTP (File Transfer Protocol) in detail.

4.3 Client Server Model:

The purpose of a TCP/IP or Internet is to provide data communication services to users. The hurdles created because of the geographical distance while accessing information from another host should be minimized.

If a user at a local host computer wishes to receive a service from a computer at a remote site. Computer executes specific *programmes* to perform the specific job. There would be a programme to do the Word processing; a programme to browse the web site and so on.

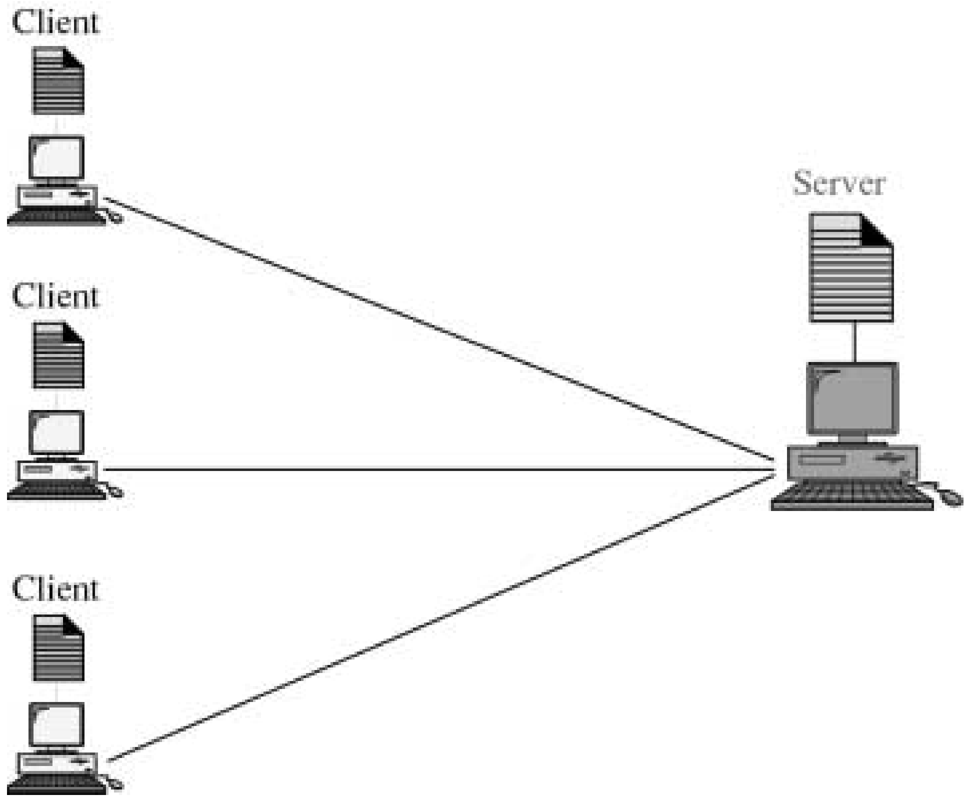
In other words for data communication to take place, a computer runs a program to request a service from another program residing on the destination computer. This means that two computers, connected by an Internet, should each run a program, one to provide a service and one to request a service.

It should be clear now that if we want to use the services available on an Internet, application programs, running at two end computers and communicating with each other, are needed. In other words, in an Internet, the application programs are the entities that communicate with each other, not the computers or users.

At first glance, it looks simple to enable communication between two application programs, one running at the local host, and the other running at the remote host system. But there are finer details that have to be taken care in order to have a proper communication.

It is important to determine beforehand, who makes the request for service and who offers it? Issue of whether both the application programs be able to request services and provide services or should the application programs just do one or the other, must be resolved. One solution is to have an application program, called the *client*, running on the local host computer, request a service from another application program, called the *server,* running on the remote computer. In other words, the task of requesting program is either a requester (a client), or a provider (a server). If a machine needs to request a service and provide a service, two application programs must be installed. In other words, application programs come in pairs. Client and server, both having the same name.

Next issue that arises is should an application program provide services only to one specific application program installed somewhere in an Internet or should it provide services for any application program that requests this service? The most common solution is a server providing a service for any client, not a particular client. In other words, the client-server relationship is many-to-one. Many clients can use the services of one server.



When should an application program be running? All of the time or just when there is a need for the service? Generally, a client program, which requests a service, should run only when it is needed. The server program, which provides a service, should run all the time because it does not know when its service is needed.

4.3.1 Client:

A client is a program running on the local host computer requesting the service from a server. A client program is *finite*, which means it is started by the user (or another application program) and terminates when the service is complete. A client opens the communication channel using the IP address of the remote host and the well-known port address of the specific server program running on that machine. This is called an *active open*. After a channel of communication is opened, the client sends its request and receives a response. Although the request-response part may be repeated several times, the whole process is finite and eventually comes to an end. At this moment, the client closes the communication channel with an active close.

4.3.2 Server:

A server is a program on the remote host computer providing service to the clients. When it starts, it opens the channel for incoming requests from clients, but it never initiates a service until it is requested to do so. This is called a passive open.

A server program is an infinite program. When it starts it runs infinitely unless a problem arises. It waits for incoming requests from clients. When a request arrives, it responds to the request, either iteratively or concurrently as explained in the next section.

4.3.3 Concurrency:

Here the term concurrency refers to the ability to run simultaneously. Both clients and serves can run in concurrent mode. The concurrency in clients and server differ and they are explained below.

Concurrency in Clients:

Clients can be run on a computer either iteratively or concurrently. Running clients iteratively means running them one by one; one client must start, run, and terminate before the computer can start another client. Most computers today, however, allow concurrent clients, that is, two or more clients can run at the same time.

Concurrency in Servers

Because an iterative server can process only one request at a time, it receives a request, processes it, and sends the response to the requestor before it handles another request. If there is a request for from another client, then it has to either rejected or kept in waiting till the server finishes the first one.

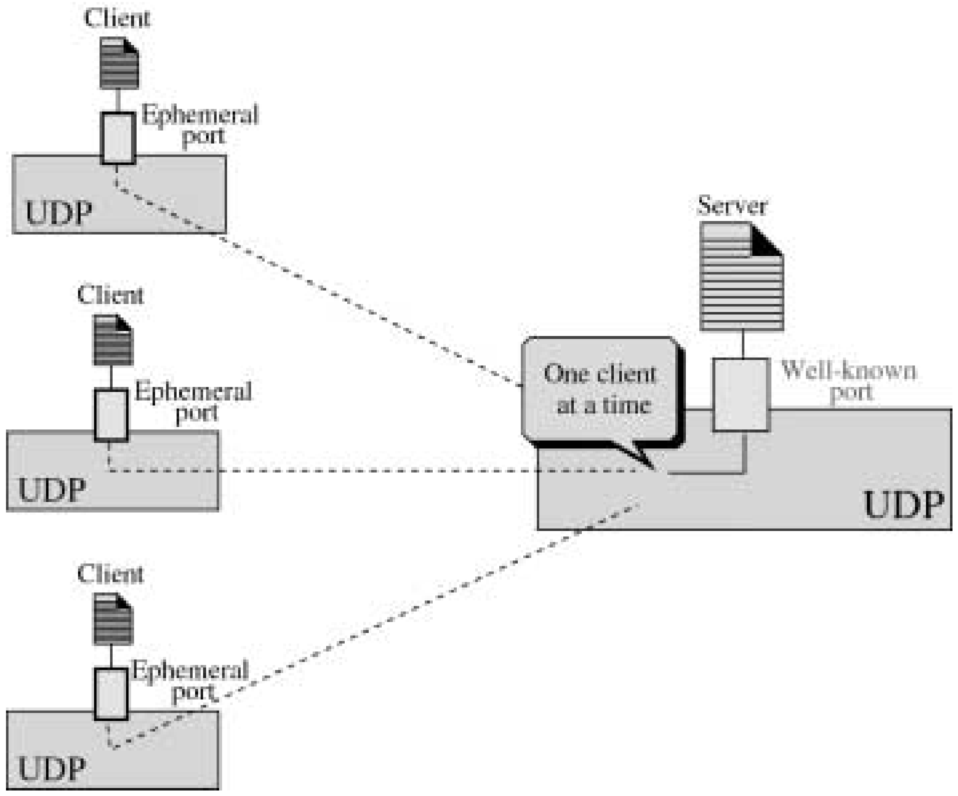
A concurrent server, on the other hand, can process many requests at the same time and thus can share its time between many requests.

The servers use UDP, a connectionless transport layer protocol, or TCP, a connection-oriented transport layer protocol. Server operation, therefore, depends on two factors: the transport layer protocol and the service method. Theoretically we can have four types of servers: connectionless iterative, connectionless concurrent, connection-oriented iterative, and connection-oriented concurrent. However, only the first and the last are commonly used. Hence only they are discussed.

4.3.4 Connectionless Iterative Server:

The servers that use UDP are normally iterative, which means that the server processes one request at a time. A server gets the request received in a datagram from UDP, processes the request, and gives the response to UDP to send to the client. The server pays no attention to the other datagrams. These datagrams are stored in a queue, waiting for service. They could all be from one client or from many clients. In either case they are processed one by one in order of arrival.

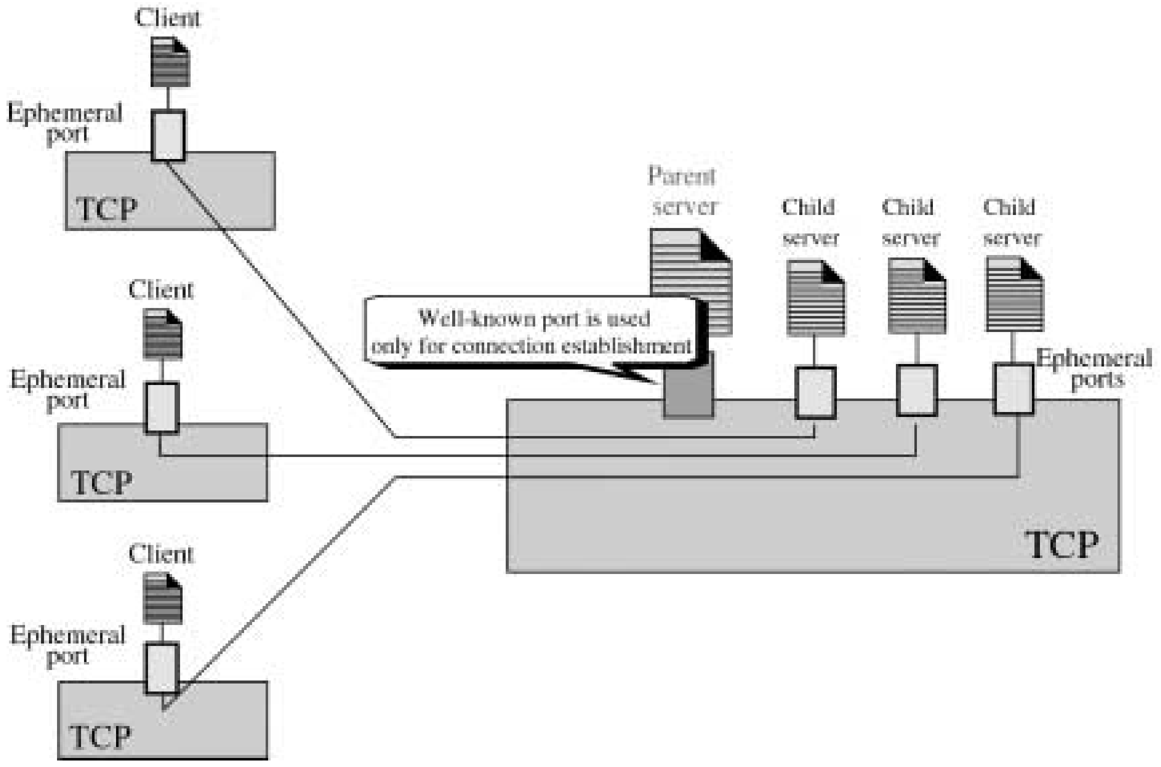
The server uses one single port for this purpose, the well-known port. All the datagrams arriving at this port wait in line to be served. The Figure illustrates the same.



Generally the services provided by these iterative servers are one which takes very short time execute. Example service is the Internet Daytime service, which returns the time and the date.

4.3.5 Connection-Oriented Concurrent Server:

The servers that use TCP are normally concurrent. This means that the server can serve many clients at the same time. Communication is connection-oriented, which means that a request is a stream of bytes that can arrive in several segments and the response can occupy several segments. A connection is established between the server and each client, and the connection remains open until the entire stream is processed and the connection is terminated.



This type of server cannot use only one well-known port because each connection needs a port and many connections may be open at the same time. Many ports are needed, but a server can use only one well-known port. The solution is to have one well-known port and many ephemeral ports. The server makes a passive open at the well-known port. A client can make its initial approach to this port to make the connection. After the connection is made, the server assigns a temporary port to this connection to free the well-known port. Data transfer can now take place between these two temporary ports, one at the client site and the other at the server site. The well-known port is now free for another client to make the connection. The idea is to push demultiplexing to TCP instead of the server.

The server must also have one buffer for each connection. The segments come from the client, are stored in the appropriate buffer, and will be served concurrently by the server.

To provide this service, most implementations use the concept of parent and child servers. A server running infinitely and accepting connections from the clients is called a *parent server*. The parent uses the well-known port. After it makes a connection, the parent server creates a child server and an ephemeral port and lets the child server handle he service. It thereby frees itself so that it can wait for another connection.

4.4 Processes:

Understanding the concept of a process is necessary to comprehend the client-server model. In this section, this concept and its relationship to the client-server model, particularly concurrent processing is explained.

Most operating systems, distinguish a program from a process. Whereas a program and a process are related to each other, they are not the same thing. The relationship between a program and a process is similar to the relationship between a class and an object in object-oriented programming.

In object-oriented programming, a class is just a definition. One can define one single class, but many instances of that class, called objects, can be instantiated. The class is just the declarations and definitions and definitions of members. Memory allocation and the storing of data in the data members occur only when an instance of the class (an object) is created. Although all objects have the same types of data elements, the values stored in those data elements may be totally different for each object.

A program is code. The code defines all the variables and actions to be performed on those variables. A process, on the other hand, is an instance of a program. When the operating system executes a program, an instance of the program, a process, is created. The operating system can create several processes from one program, which means several instances of the same program are running at the same time (concurrently). Although all processes have the same data types, memory is allocated for each process separately. Also, the values stored in variables may be totally different from one process to another. In fact, the functions executed by each process of the same programme may differ, as each may take different inputs.

4.5 FTP (File Transfer Protocol)- An example network application:

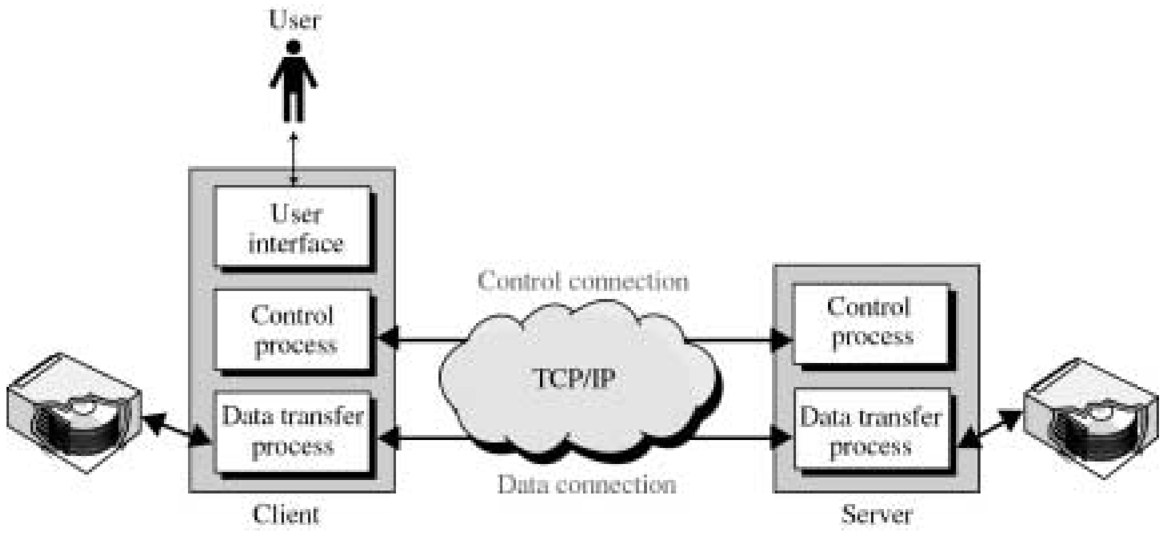
File transfer protocol (FTP) is the standard mechanism provided by TCP/IP for copying a file from one host to another. Transferring files from one computer to another is one of the most common tasks expected from the networking environment.

Although transferring files from one system to another seems simple and straightforward, some problems must be dealt with first. For example, two systems may use different file name conventions. Two systems may have different ways to represent text and data. Two systems may have different directory structures. All of these problems have been solved by FTP in a very simple and elegant approach.

FTP differs from other client-server applications in that it establishes two connections between the hosts. One connection is used for data transfer, the other for control information (commands and responses). Separation of commands and data transfer makes FTP more efficient. The control connection uses very simple rules of communication. We need to transfer only a line of command or a line of response at a time. The data connection, on the other hand, needs more complex rules due to the variety of data type transferred.

FTP uses two well-known TCP ports: Port 21 is used for the control connection, and port 20 is used for the data connection.

Figure shows the basic model of FTP. The client has three components: user interface, client control process, and the client data transfer process. The server has two components: the server control process and the server data transfer process. The control connection is made between the control processes. The data connection is made between the data processes.



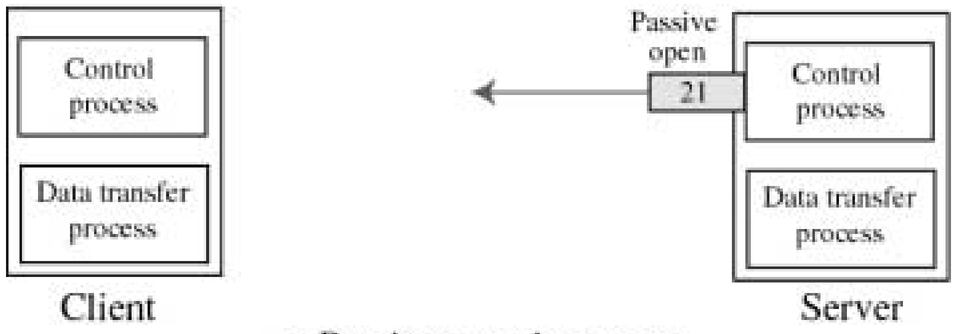
The control connection remains connected during the entire interactive FTP session. The data connection is opened and then closed for each file transferred. It opens each time commands that involve transferring files are used, and it closes when the file is transferred. In other words, when a user starts an FTP session, the control connection is opened. While the control connection is open, the data connection can be opened and closed multiple times if several files are transferred.

The two FTP connections control and data use different strategies and different port numbers.

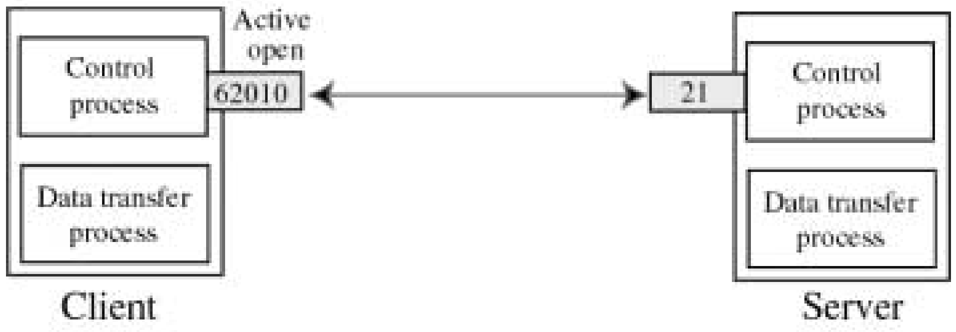
4.5.1 Control Connection:

The control connection is created in the same way as other application programs described so far. There are two steps:

1. The server issues a passive open on the well-known port 21 and waits for a client.
2. The client uses an ephemeral port and issues an active open connection to the port 21 of server process.



a. Passive open by server



b. Active open by client

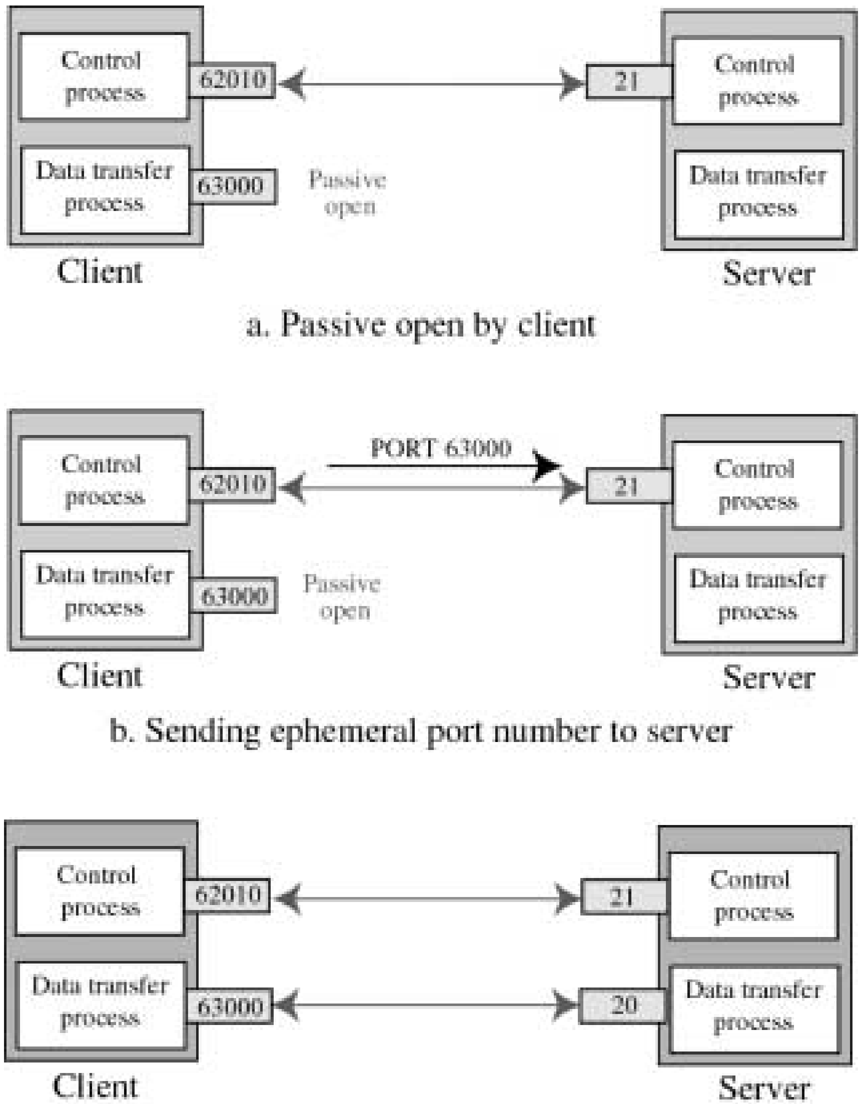
The connection remains open during the entire process. The service type, used by the IP protocol, is *minimize delay* because this is an interactive connection between a user (human) and a server. The user types commands and expects to receive responses without significant delay. After the initial connection, the server process creates a child process and assigns the duty of serving the client to the child process using an ephemeral port.

4.5.2 Data Connection:

The data connection uses the well-known port 20 at the server site. However, the creation of a data connection is different from what we have seen so far. The following shows how FTP creates a data connection:

1. The client, not the server, issues a passive open using an ephemeral port. This must be done by the client because it is the client that issues the commands for transferring files.
2. The client sends this port number to the server using the PORT command.

3. The server receives the port number and issues an active open using the well-known port 20 and the received ephemeral port number.
4. After the initial connection, the server process creates a child process and assigns the duty of serving the client to the child process using an ephemeral port.



4.5.3 Communication:

The FTP client and server, which run on different computers, must communicate with each other. These two computers may use different operating systems, different character sets, different file structures, and different file formats. FTP must make this heterogeneity compatible.

Communication over Control Connection

FTP uses ASCII character set to communicate across the control channel. Communication is achieved through commands and responses. This simple method is adequate for the control connection because we send one command (response) at a time. Each command or response is only one short line so we need not worry about file format or file structure. Each line is terminated with a two-character (carriage return and line feed) end-of-line token.

Communication over Data Connection

The purpose and implementation of the data connection are different from that of the control connection. We want to transfer files through the data connection. The client must define the types of file to be transferred, the structure of the data, and the transmission mode. Before sending the file through the data connection, we prepare for transmission through the control connection. The heterogeneity problem is resolved by defining three attributes of communications: file type, data structure, and transmission mode. A common format for the above three attributes would be agreed upon before actually transferring the data.

4.5.4 Command Processing:

FTP uses the control connection to establish a communication between the client control process and the server control process. During this communication, the commands are sent from the client to the server and the responses are sent back from the server to the client.

Commands

We can roughly divide the commands into six groups: access commands, file management commands, data formatting commands, port defining commands, file transferring commands, and miscellaneous commands.

- **Access Commands.** These commands let the user access the remote system.
    - Commands are - USER, PASS, ACCT, REIN, QUIT, ABOR
- **File Management commands.** These commands let the user access the file system on the remote computer. They allow the user to navigate through the directory structure, create new directories, delete files, and so on.
    - Commands are – CWD, CDUP, DELE, LIST, ….
- **Data formatting commands.** These commands let the user define the data structure, file type and transmission mode. The defined format is then used by the file transfer commands.
    - Commands are – TYPE, STRU, MODE.
- **Port defining commands.** These commands define the port number for the data connection on the client site.
    - Commands are – PORT, PASV
- **File transfer commands.** These commands actually let the user transfer files.
    - RETR, STOR, APPE, STOU, ….
- **Miscellaneous commands.** These commands deliver information to the FTP user at the client site.
    - Commands are - HELP, NOOP, …

Responses

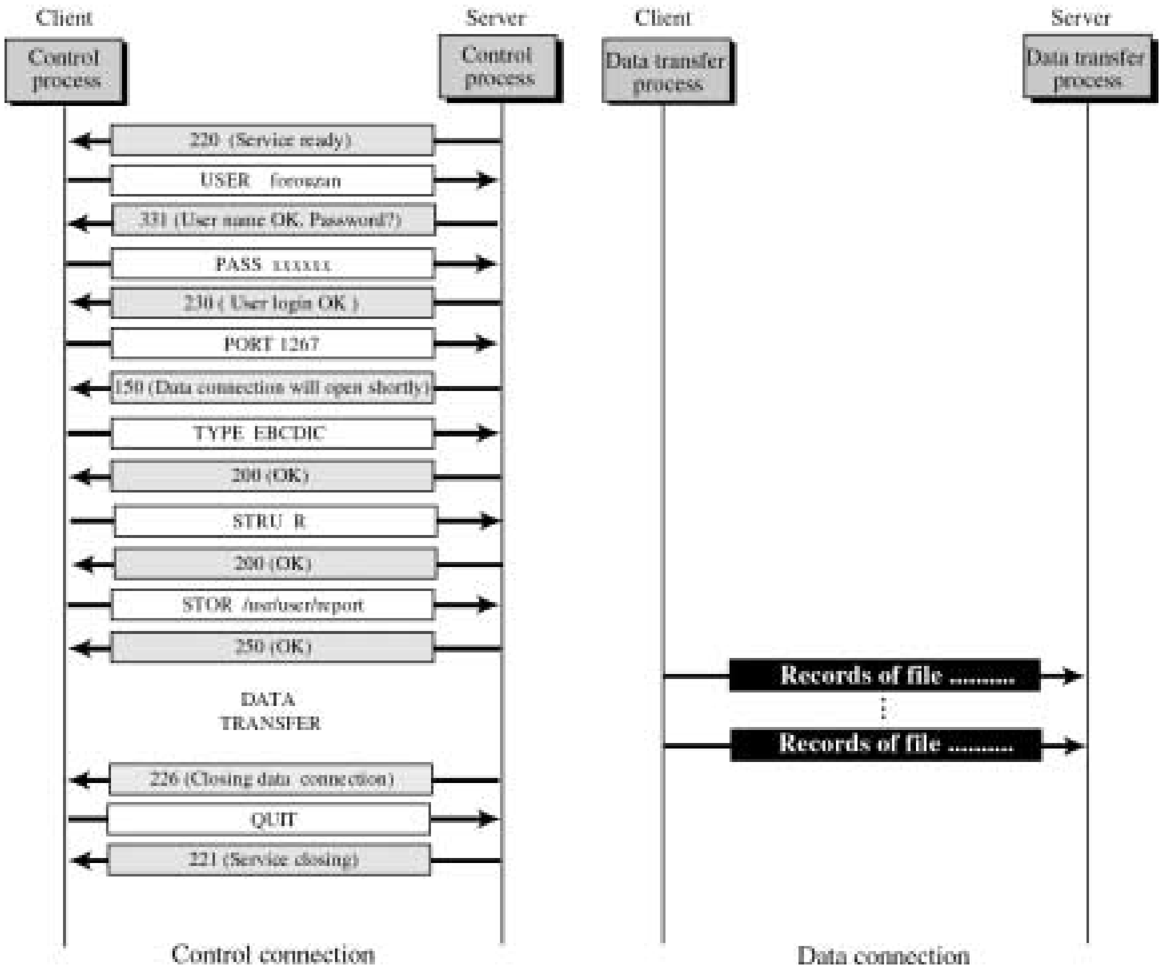
Every FTP command generates at least one response. A response has two parts: a three-digit number followed by text. The numeric part defines the code; the text part defines needed parameters or extra explanations. The numeric part is used by the client programme to ascertain the response, while the text part is meant for the user.

4.5.5 File Transfer:

File transfer occurs over the data connection under the control of the commands sent over the control connection. However, we should remember that file transfer in FTP means one of three things.

- A file is to be copied from the server to the client. This is called retrieving a file.
- A file is to be copied from the client to the server. This is called storing a file.
- A list of directory or file names is to be sent from the server to the client. Note that FTP treats a list of directory or file names as a file. It is sent over the data connection.

An example, showing the steps involved in storing an image file into the server from client side.



1. After the control connection to port 21 is created, the FTP server sends the 220 (service ready) response on the control connection.
2. The client sends the USE command (this USER command passes the user information to server).
3. The server responds with 331 (user name is OK, a password is required).
4. The client sends the PASS command (sends the password for the user).
5. The server responds with 230 (user login OK, if user id and password match).
6. The client issues a passive open on an ephemeral port for data connection and sends the PORT command (over the control connection) to give this port number to the server.
7. The server does not open the connection at this time, but prepares itself for issuing an active open on the data connection between port 20 (server side) and the ephemeral port received from the client. It sends the response 150 (data connection will be open shortly).
8. The client sends the TYPE command (to indicate the type of data as binary)
9. The server responds with response 200 (command OK)
10. The client sends the STRU command (defines the data following as of file type.)
11. The server responds with response 200 (command OK)
12. The client sends the STOR command (to request the server to store the data)
13. The server opens the data connection and sends the message 250.
14. The client sends the file on the data connection. After the entire file is sent the data connection is closed. Closing the data connection means end-of-file.
15. The server sends the response 226 on the control connection.
16. The client sends the QUIT command (or it can send the other commands to open another data connection for transferring another file).
17. The server responds with 221 (service closing) and it closes the control connection.

The above example is given to indicate the steps involved in an FTP operation, over the control and data connection. Note that before sending the actual data in step 14, the client is indicating to the server, the type of the data (as binary in step 8), its format (as File, in step 10) and action to be taken on the data received (as store, in step 12).

4.6 Summary:

In TCP/IP, the application layer corresponds to combine session, presentation and application layers of OSI model. Any application, which needs the particular service of the session and presentation layers, will implement the same themselves. The two processes communicating over the TCP/IP network can be best described using the client-server model. Client programmes are generally run by the users whenever they need its service. On the contrary the server programmes are always running, waiting on a well-known port for requests from the clients. A connection-less iterative server uses UDP as its transport layer protocol, and serves one client at a time. A connection-oriented server concurrent server uses TCP as its transport protocol, and can serve many clients at the same time.

File Transfer Protocol (FTP) is a TCP/IP client-server application for copying files, from one host to another. FTP requires two connections for data transfer: a control connection and a data connection. Prior to actual transfer of the files, its file type, format, and transmission mode are defined by the client through the control connection. Actual transfer of data happens over the data connection.