



Indira Gandhi National Open University
School of Computer and Information Sciences

BCS-052
NETWORK
PROGRAMMING AND
ADMINISTRATION

Block

1

TCP/IP PROTOCOLS

UNIT 1

Introduction to TCP/IP	5
-------------------------------	----------

UNIT 2

Internet Protocol	45
--------------------------	-----------

UNIT 3

Transport Layer Protocols	61
----------------------------------	-----------

UNIT 4

Application Layer Protocols	76
------------------------------------	-----------

PROGRAMME DESIGN COMMITTEE

Prof. Manohar Lal, SOCIS, IGNOU, New Delhi	Prof. Arvind Kalia, Dept. of CS HP University, Shimla	Sh. Akshay Kumar Associate Prof. SOCIS, IGNOU, New Delhi
Prof. H.M Gupta Dept. of Elect. Engg. IIT, Delhi	Prof. Anju Sehgal Gupta SOCIS, IGNOU, New Delhi	Dr. P.K. Mishra, Associate Prof. Dept. of CS, BHU, Varanasi
Prof. M.N Doja, Dept. of CE Jamia Millia, New Delhi	Prof. Sujatha Verma SOS, IGNOU, New Delhi	Sh. P.V. Suresh, Asst. Prof. SOCIS, IGNOU, New Delhi
Prof. C. Pandurangan Dept. of CSE, IIT, Chennai	Prof. V. Sundarapandian IITMK, Trivendrum	Sh. V.V. Subrahmanyam, Asst. Professor SOCIS, IGNOU, New Delhi
Prof. I. Ramesh Babu Dept. of CSE Acharya Nagarjuna University, Nagarjuna Nagar (AP)	Prof. Dharamendra Kumar Dept. of CS, GJU, Hissar	Sh. M.P. Mishra, Asst. Prof. SOCIS, IGNOU, New Delhi
Prof. N.S. Gill Dept. of CS, MDU, Rohtak	Prof. Vikram Singh Dept. of CS, CDLU, Sirsa	Dr. Naveen Kumar, Asst. Prof. SOCIS, IGNOU, New Delhi
	Sh. Shashi Bhushan Associate. Prof. SOCIS, IGNOU, New Delhi	Dr. Subodh Kesharwani, Asst. Prof. SOMS, IGNOU, New Delhi

COURSE CURRICULUM DESIGN COMMITTEE

Prof. Naveen Garg, Professor IIT-Delhi	Dr. D.P Vidyarthi Associate Professor SC&SS, JNU, New Delhi	Sh. M.P. Mishra, Asst. Prof. SOCIS, IGNOU, New Delhi
Sh.Shashi Bhushan Associate Professor SOCIS, IGNOU, New Delhi	Sh. Akshay Kumar Associate Professor SOCIS, IGNOU, New Delhi	Dr. Naveen kumar, Asst. Prof. SOCIS, IGNOU, New Delhi
Ms Seema Gupta Asst. Professor, IP University		

SOCIS FACULTY

Sh. Shashi Bhushan, Director	Sh. Akshay Kumar Associate Professor	Dr. P.V. Suresh Associate Professor
Sh. V.V. Subrahmanyam, Associate Professor	Dr. Naveen Kumar Reader	Sh. M.P. Mishra Asst. Professor
Dr. Sudhansh Sharma Asst. Professor		

PREPARATION TEAM

Dr. D.K. Lobiyal (<i>Content Editor</i>) School of Computer and System Sciences, JNU	Md. Sohrabuddin Dept. of Computer Engineering Jamia Millia Islamia, New Delhi	Language Editors Dr. Malathi Nair & Dr. Nandini Sahu School of Humanities, IGNOU
Shri Amit Goel Dept. of Computer Engineering YMCA Institute of Engineering Faridabad	Shri V.P. Vishwakarma Amity School of Engineering and Technology, New Delhi	

Course Coordinator: Dr. Naveen Kumar, Reader, SOCIS, IGNOU, New Delhi

PRINT PRODUCTION: Sh. Tilak Raj, S.O.(P), MPDD, IGNOU, New Delhi

July, 2013

© Indira Gandhi National Open University, 2013

ISBN:

All rights reserved. No part of this work may be reproduced in any form, by mimeograph or any form, by mimeograph or any other means, without permission in writing form the Indira Gandhi National Open University.

Further information on the Indira Gandhi National Open University courses may be obtained from the University's office at Maidan Garhi, New Delhi-110068.

Printed and published on behalf of the Indira Gandhi National Open University by the Director, SOCIS.

Laser Composer : Tessa Media & Computers, C-206, A.F.E-II, Jamia Nagar, New Delhi-25

Print:

COURSE INTRODUCTION

TCP and IP were developed by a Department of Defence (USA) research project to connect a number of different networks designed by different vendors into a network of networks (the “Internet”). It was initially successful because it delivered a few basic services that everyone needs like file transfer, electronic mail, remote logon, etc.

TCP/IP has been deployed on virtually every type of hardware and software platform available today and forms the backbone of the Internet. As a result, this popular protocol suite has become the standard for building multi-vendor Intranets and Internet. This course provides a comprehensive technical introduction to TCP/IP. Also, it is a hands-on guide to TCP/IP technologies, and shows how the protocols operate in practice.

As you start exploring this course you will find the details of different layers and protocols of TCP/IP and their characteristics in *Block 1: TCP/IP Protocols*. Further, the next *Block 2: Fundamentals of TCP/IP Programming* provides a general introduction to the Unix sockets, APIs and their use for developing network applications.

Block 3: Network Administration with Linux covers TCP/IP networking, network administration and system configuration basics. Linux network configuration, management, monitoring and system tools are also covered in this block.

In your fifth semester, you will also study BCSL-056, a Lab course related to this course which contains the practical details about Unix , C language and Linux related to TCP/IP programming and Linux network administration. It contains, a series of laboratory exercises that span the various elements of network administration and TCP/IP programming.

BLOCK INTRODUCTION

The *Block: TCP/IP Protocols* will show how the protocols in the TCP/IP suite combine to provide services to users working in a distributed processing environment on an enterprise network (internet). This block provides an overview of the use of TCP/IP with the relationship between the OSI reference model. This leads into a detailed explanation of the TCP/IP layers and how data is processed in these layers. The objective of this block is to provide delegates with a practical understanding of TCP/IP and the skills required to develop network applications using socket programming.

UNIT 1 INTRODUCTION TO TCP/IP

Structure	Page Nos
1.0 Introduction	5
1.1 Objectives	6
1.2 Origin of TCP/IP and Internet	6
1.2.1 Communication	6
1.2.2 Why do we Need the Internet	8
1.2.3 Need of Protocol on Communication	8
1.2.4 Problems in Computer Communication	9
1.2.5 Dealing with Incompatibility	9
1.2.6 A Brief History of the Internet	10
1.2.7 Architecture of the Internet	11
1.3 TCP/IP Layer and Protocols	11
1.4 Network Access Layer	13
1.5 Internet Layer	14
1.5.1 Need for IP Address	14
1.5.2 Classes of IP Address	14
1.5.3 Special Meanings	15
1.5.4 Who Decides the IP Addresses	16
1.5.5 Internet Protocol	16
1.5.6 Address Resolution Protocol (ARP)	18
1.5.7 Reverse Address Resolution Protocol (RARP)	19
1.5.8 Internet Control Message Protocol (ICMP)	19
1.6 Transport Layer	20
1.6.1 Transmission Control Protocol	21
1.6.2 User Datagram Protocol (UDP)	25
1.7 Application Layer	26
1.7.1 Electronic Mail	26
1.7.2 Domain Name System (DNS)	30
1.7.3 How does the DNS Server Works?	32
1.7.4 Simple Network Management Protocol (SNMP)	34
1.7.5 Remote Login: TELNET	36
1.7.6 World Wide Web: HTTP	38
1.8 Networking Example	40
1.9 Summary	42
1.10 Answers to Check Your Progress	43
1.11 Further Readings	44

1.0 INTRODUCTION

Transmission Control Protocol (TCP)/Internet Protocol (IP) is a set of protocols developed to allow computers of all sizes from different vendors, running different operating systems, to communicate or to share resources across a network. A packet switching network research project was started by the USA Government in the late 1960s in 1990s, became the most widely used form of computer networking. This project centered around ARPANET. ARPANET is best-known TCP/IP network.

TCP/IP is the principal UNIX networking protocol and was designed to provide a reliable end-to-end byte stream over an unreliable internetwork. TCP is a connection-oriented protocol while IP is a connectionless protocol. TCP supplies logic to give a reliable connection-oriented protocol above IP. It provides a virtual-circuit that two processes can use to communicate. IP provides a connectionless and unreliable delivery system and transfer each datagram independently in the network.

UDP is a connectionless and unreliable protocol running over IP. It adds a checksum to IP for the contents of the datagram and pass members. In this unit we are going to discuss all the protocols of TCP/IP in brief.

1.1 OBJECTIVES

After going through this unit, you should be able to know:

- the need of Communication and Internetworking;
- how the Internet and TCP/IP came into existence;
- the meaning of the terms protocol and standard;
- the architecture of the TCP/IP Protocol Suite;
- difference between the OSI model and the TCP/IP Suite;
- the need and functionalities of Interface layer;
- the need and functionalities of Internet layer;
- the need and functionalities of Transport layer;
- the need and functionalities of Application layer, and
- how communication takes place on the Internet.

1.2 ORIGIN OF TCP/IP AND INTERNET

Before going through the origin of the Internet, let us examine what is communication.

1.2.1 Communication

Communication the process of sharing ideas, information, and messages with others at a particular time and place. Communication is a vital part of personal life and is also important in business, education, and any other situation where people encounter each other. Communication between two people is an outgrowth of methods developed over centuries of expression. Gestures, the development of language, and the necessity to engage in joint action all play a part. Communication, as we see it today, has evolved a long way. We will discuss the primitive modes of communication briefly.

i) Early Methods

Early societies developed systems for sending simple messages or signals that could be seen or heard over a short distance, such as drumbeats, fire and smoke signals, or lantern beacons. Messages were attached to the legs of carrier pigeons that were released to fly home (this system was used until World War I, which started in 1914). Semaphore systems (visual codes) of flags or flashing lights were employed to send messages over relatively short but difficult-to-cross distances, such as from hilltop to hilltop, or between ships at sea.

ii) Postal Services

The postal system is a system by which written documents normally enclosed in envelopes, and also small packages containing other matter, are delivered to

destinations around the world. Anything sent through the postal system is called post. In India the East India Company in Mumbai, Chennai and Calcutta introduced the postal system in 1766, further these postal service became available to the general public. Even after implementing different electronic communication mediums, postal system is still one of the popular communication systems available.

iii) Telegraph

The first truly electronic medium for communication was the telegraph, which sent and received electrical signals over long-distance wires. The first practical commercial systems were developed by the physicist, Sir Charles Wheatstone and the inventor Sir William F. Cooke in Great Britain, and by the artist and inventor Samuel F. B. Morse in the United States. Morse demonstrated the first telegraph system in New York in 1837. But regular telegraph service, relaying Morse code (system of code using on and off signals), was not established until 1844. Telegraphers would translate the letters of the alphabet into Morse code, tapping on an electrical switch, or key. The telegrapher at the other end of the line would decode the tapping as it came in, write down the message, and send it to the recipient by messenger. The telegraph made it possible for many companies to conduct their business globally for the first time.

iv) Telephone

Early devices capable of transmitting sound vibrations and even human speech appeared in the 1850s and 1860s. The first person to patent and effectively commercialise an electric telephone was Scottish-born American inventor Alexander Graham Bell. Originally, Bell thought that the telephone would be used to transmit musical concerts, lectures, or sermons.

The telephone network has also provided the electronic network for new computer-based systems like the Internet facsimile transmissions, and the World Wide Web. The memory and data-processing power of individual computers can be linked together to exchange the data transmitted over telephone line, by connecting computers to the telephone network through devices called modems (modulator-demodulators).

v) Computers and Internet

The earliest computers were machines built to make repetitive numerical calculations that had previously been done by hand. While computers continued to improve, they were used primarily for mathematical and scientific calculations, and for encoding and decoding messages. Computer technology was finally applied to printed communication in the 1970s when the first word processors were created.

At the same time computers were becoming faster, more-powerful and smaller, and networks were developed for interconnecting computers. In the 1960's the Advanced Research Projects Agency (ARPA) of the U.S. Department of Defence, along with researchers working on military projects at research centres and universities across the country, developed a network called the ARPANET, for sharing data and processing time of uniform computer connection over specially equipped telephone lines and satellite links. The network was designed to survive the attack or destruction of some of its parts and continue to work.

Soon, however, scientists using the ARPANET realised that they could send and receive messages as well as data and programs over the network. The ARPANET became the first major electronic-mail network; soon thousands of researchers all over the world used it. Later on the National Science Foundation (NSF) helped connect more universities and non-military research sites to the ARPANET, and renamed it the Internet because it was a network of networks among many different organisations.

Today, the Internet is the widely known computer network. It uses interconnection of computer system by both wired and wireless. Smaller networks of computers, called Local Area Networks (LANs), can be installed, in a single building or for a whole organisation. Wide Area Networks (WANs) can be used to span a large geographical area. LANs and WANs use telephone lines, computer cables, and microwave and laser beams to carry digital information around a smaller area, such as a single college campus. Internet can carry any digital signals, including video images, sounds, graphics, animations, and text, therefore it has become very popular communication tool.

1.2.2 Why do we Need the Internet?

The main reason is that each computer network is designed with a specific purpose. For example, LAN is used to connect computers in a smaller area, and it provides fast communication. As a result, networks become specialised identify. In many cases, these networks do not use the same hardware and software technology. It means that, a computer can communicate with the computers attached to the same network, because they are intercompatible. As more and more organisations had multiple computer networks, this became a major issue. As a result, the concept of internetworking (internet) came into being. This means that there should be a network of all physically separate networks.

1.2.3 Need of Protocol on Communication

All methods of communication described above follow a protocol. Protocol is nothing but a convention. To signify that “Everything is ok and the train can start by a green flag is also a protocol. When we write a letter we follow a protocol. If we look at them carefully, we will find that protocols normally have hidden layers. A good example is human conversation over the phone which can be used as an analogy for communication using computers.

Assume that X and Y, want to have conversation over the telephone about a cricket match. We call this an *idea*. Assume that each person is taking down what other has to say. Thus, the conversation takes place in the form of several messages. A message is a block of sentence. It could also consist of one word such as OK, yes denoting a positive **acknowledgement** (ACK). It could also mean a **negative acknowledgement** (NAK) or a request to repeat such as come again, pardon me. All this happens both ways.

At the level of idea, X and Y feel that they are discussing a cricket match. However, in reality, the conversation consists of a number of messages.

A message could be too long. It may not be wise for X to speak for half an hour, only to receive a request to repeat. It is therefore necessary to send/receive acknowledgements after each sentence like ‘ok’, ‘come again’ etc. A sentence is analogous to a packet in computer world. The sender X will not speak until s/he hears some form of acknowledgement, or will repeat the sentence if s/he receives a negative acknowledgement. An alternative is *timeout* strategy. The speaker speaks a sentence and waits for some time for any acknowledgement. If s/he does not hear anything, s/he repeats the sentence.

Apart from this **error control**, we take care of **flow control**. Flow control refers to the speed mismatch between the listener and speaker. If the speaker speaks too fast, the listener will say go-slow. In computer world, if the receiving computer is not fast enough, and cannot hold any more data, it requests the sender to wait or control the transfer by slowdown.

Therefore, in computer communication, both speaker and listener should agree on the communication language/syntax, scheme of acknowledgement, during flow control, machine error control mechanism, etc. Thus, we can say that the conversation is

governed by some set of rules known to both the parties. This set of rules is called protocol and it necessary for disciplined manner of conversation/communication.

1.2.4 Problems in Computer Communication

The same concept of protocols described above applies to computer communication. The main difference is that earlier, the devices (telephones) on both sides are compatible. However, in an Internet work, it may or may not be so. The concept of Internetworking though, highly desirable, is not easily achievable.

Thus, any two networks, cannot directly communicate by connecting a wire between the networks. For example, one network could represent a binary 0 by -5 volts, another by +5 volts. Similarly, one could use a packet size of 128 bytes, whereas other could use 256 byte packets. The method of acknowledgement or error detection could be different. There could be many such differences.

1.2.5 Dealing with Incompatibility

The incompatibility issues are handled at two levels:

i) Hardware Issues

At the hardware level, an additional component called router is used to connect physically distinct networks as shown in *Figure 1*. A router connects to the network in the same way as any other computer. Any computer connected to the network has a Network Interface Card (NIC), which has the address (network id+host id), hard coded into it. A router is a device with more than one NICs. Router can connect incompatible networks as it has the necessary hardware (NIC) and protocols (TCP/IP).

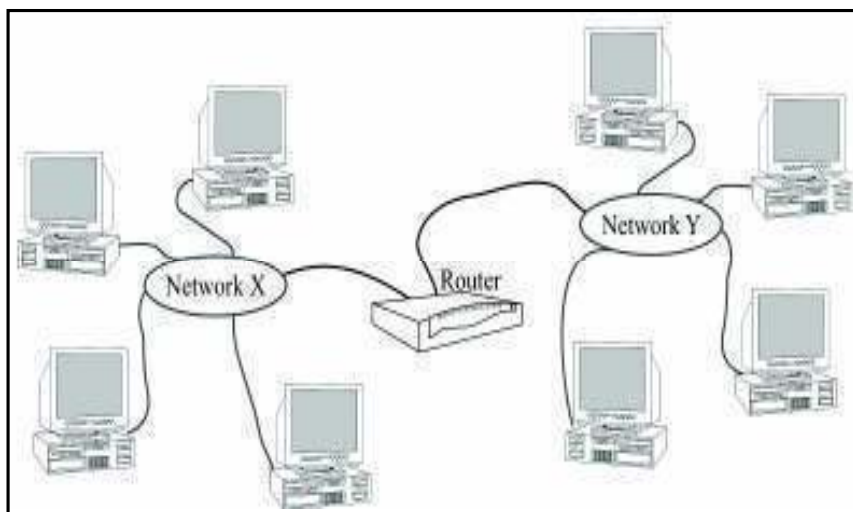


Figure 1: Router connects the Networks

ii) Software Issues

The routers must agree about the way information would be transmitted to the destination computer on a different network, since the information is likely to travel through different routers, there must be a predefined standard to which routers must confirm. Packet formats and addressing mechanism used by the networks may differ. One approach could be to perform conversion and reversion corresponding to different networks. But this approach is difficult and cumbersome. Therefore, the Internet communication follows one protocol, the TCP/IP protocol suite. The basic idea is that it defines a packet size, routing algorithms, error control, flow control methods universally.

It would be unwise to club all these features in a single piece of software — it would make it very bulky. Therefore, all these features are logically sub-grouped and then the sub-groups are further grouped into groups called layers. Each layer has an interface with the adjacent layers, and performs specific functions.

Need for Layering

Since it is difficult to deal with complex set of rules, and functions required for computer networking, these rules and functions are divided with logical groups called layers. Each layer can be implemented interdependently with an interface to other layers providing with services to it or taking its services like flow control and error control functions are grouped together and the layer is called data link layer. Speech in telephone conversation is translated, with electrical segments and vice-versa. Similarly in computer system the data or pattern are converted into signals before transmitting and receiving. These function and rules are grouped together in a layer called physical layer.

1.2.6 A brief history of the Internet

Internet is made up of thousand and thousands of interconnected networks. Although it has become extremely popular in the last decade, it came into being only in 1969.

ARPANET

In the mid 1960s, the Advanced Research Projects Agency (ARPA) in the US Department of Defence (DoD) wanted to find a way to connect computers so that their funded researchers could share their findings. In 1967, ARPA proposed its idea for ARPANET, a small network for connecting computers.

Birth of Internet and TCP/IP

In 1972, Vint Cerf and Bob Kahn, who were part of core ARPANET group, started Internetting Project with the aim to connect different networks together. Diverse interfaces, different packet sizes, diverse transmission rates presented many difficulties. Kahn Cerf proposed the idea of gateway as an intermediate hardware to transfer data from one network to another.

In 1973, they presented a landmark paper outlining the protocols for end-to-end communication. This paper on TCP/IP included concepts like datagram, gateway, and encapsulation. At this time, responsibility of ARPANET was handed to Defence Communication Agency (DCA).

Came 1977 and communication between networks was made possible. Internet consisting of three different networks, namely, ARPANET, packet radio, and packet satellite was now possible.

In 1978 noticing the popularity of BSD (Berkley Software Distribution) UNIX, which included network capabilities, ARPA signed a contract with Berkley under which TCP/IP software was incorporated in the operating system itself.

In 1983 original ARPANET protocols were abolished and TCP/IP was made de facto standard for Internet.

Time Line

- 1969 Four-node ARPA established
- 1972 Internetting Project begins
- 1973 Development of TCP/IP suite begins

- 1977 An Internet tested using TCP/IP
- 1978 UNIX distributed to universities
- 1983 TCP/IP became the official protocol for ARPANET.

1.2.7 Architecture of the Internet

Internet is organised to form a hierarchy. At the top, there is a very high-speed backbone and at the other end, there are users. There are Network Access Providers (NAP) and Internet Service Providers (ISP) at the intermediate level as shown in the *Figure 2*.

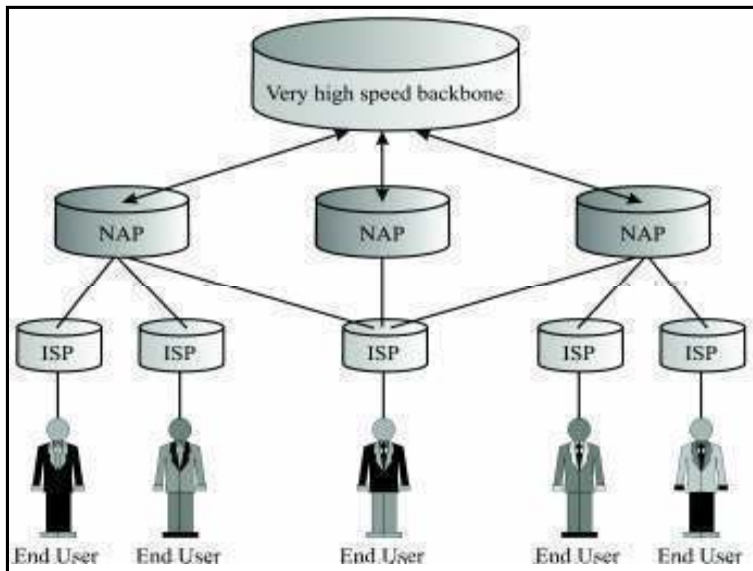


Figure 2: Architecture of Internet

A home user dials into the ISP, may be using a twisted pair telephone connection using a modem. The ISP connects to one of the Network Access Providers, which in turn, connects to the high-speed backbone at a Network Access Point. Network Access Point serves the purpose of connecting backbone networks to provide connectivity between end users.

1.3 TCP/IP LAYERS AND PROTOCOLS

The TCP/IP model is made up of four layers: interface layer, network, transport, and application. The first layer of TCP/IP (Application layer) is similar to the first three layers (Application, presentation and Session layer) of the OSI model. The services of transport layers of both the models are similar. Further, the services of network layers in both models are also similar, while some time network layer is also known as Internet layer. The last layer of TCP/IP is interface layer, which includes the services of data link layer and physical layer of OSI model. In OSI model, each layer takes the services of the lower layer. Whereas the layers of TCP/IP protocol suite contain relatively independent protocols.

The mapping of OSI & TCP/IP model is shown in the *Figure 3*, it also shows different protocols included in the TCP/IP model.

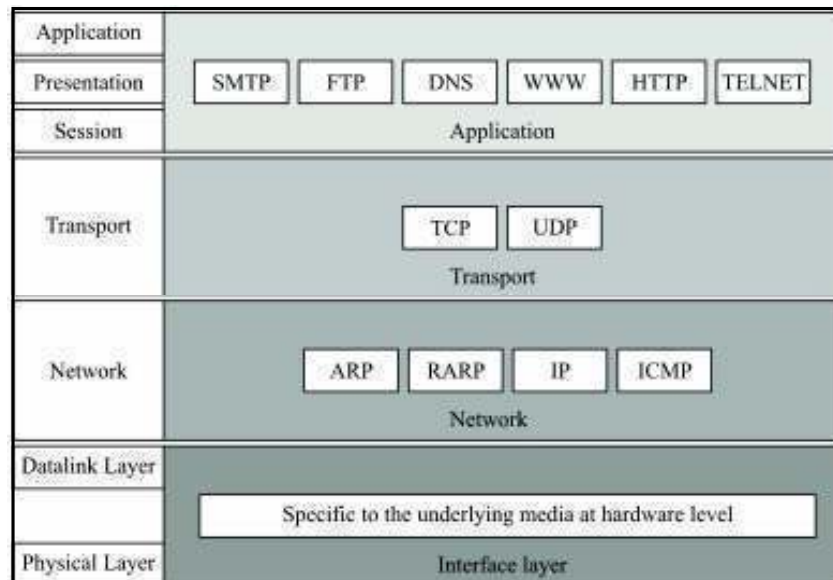


Figure 3: Mapping of OSI and TCP/IP model

Layers of TCP/IP Protocol Suite

As we know TCP/IP contains four layers and each layer has its specific functions, in the following section let's find out the functions of each layer of TCP/IP.

Interface layer or (Physical + Data Link Layer)

The physical layer deals with the hardware level like, transmission media, connections and the voltage for digital signals. The data link layer deals with media access and control strategies, frame format etc.

Internet Layer or Network Layer

The Internet layer is an important layer in the protocol suite. At this layer, TCP/IP supports Internetworking Protocol (IP). IP is a host-to-host protocol. This layer is responsible for the format of datagrams as defined by IP, and routing a datagram or packet to the next hop, but is not responsible for the accurate and timely delivery of datagrams to the destination in proper sequence. IP allows raw transmission functions allowing user to add functionalities necessary for given application. Ensuring maximum efficiency, TCP/IP supports four other protocols: ARP, RARP, ICMP and IGMP in this layer.

- **Address Resolution Protocol (ARP)**

On a LAN, each machine is identified with a unique physical address imprinted on the network interface card. ARP is used to find the physical address of a machine when its IP address is known.

- **Reverse Address Resolution Protocol (RARP)**

It is used to find the IP address of a machine when its physical address is known. It is used when a diskless computer is booted or a computer is connected to the network for the first time.

- **Internet Control Message Protocol (ICMP)**

IP is unreliable and best effort delivery. In case of failures ICMP is used to send notifications to the sender about packet problems. It sends error and query messages.

- **Internet Group Message Protocol (IGMP)**

It is used for multicasting, which is transmission of a single message to a group of recipients.

Transport Layer

At this layer, TCP/IP supports two protocols: TCP, UDP, IP is host-to-host protocol, which can deliver the packet from one physical device to another physical device. TCP, UDP, are transport level protocols, responsible for delivering a packet from one process on a device to another process on the other device.

User Datagram Protocol (UDP)

It is simpler of the two protocols. It does not provide reliability. It is, therefore faster, and using for applications in which delay is intolerable (in case of audio and video).

Transmission Control Protocol (TCP)

TCP is reliable, connection oriented protocol. By connection oriented, we mean that a connection must be established between both ends before either can transmit data. It ensures that communication is error-free and in sequence.

Application Layer

As said earlier, it is closer to combined session, presentation, and application layer of OSI model. It allows the user to run various applications on Internet. These applications are File Transfer Protocol (FTP), remote login (TELNET), email (SMTP), WWW (HTTP). The session layer of OSI model is almost dropped in TCP/IP.



Check Your Progress 1

- 1) How is the TCP/IP model different from OSI model?

.....

.....

.....

.....

- 2) What are the different layers of TCP/IP protocol suite? Explain.

.....

.....

.....

1.4 NETWORK ACCESS LAYER

The design of TCP/IP hides the function of this layer from users—it is concerned with getting data across a specific type of physical network (such as Ethernet, Token Ring, etc.). This design reduces the need to rewrite higher levels of a TCP/IP stack when new physical network technologies are introduced (such as ATM and Frame Relay).

The functions performed at this level include encapsulating the IP datagrams into *frames* that are transmitted by the network. It also maps the IP addresses to the physical addresses used by the network. One of the strengths of TCP/IP is its addressing scheme, which uniquely identifies every computer on the network. This IP address must be converted into whatever address is appropriate for the physical network over which the datagram is transmitted.

1.5 INTERNET LAYER

The Internet layer is an important layer in the protocol suite. At this layer, TCP/IP supports Internetworking Protocol (IP). IP is host-to-host protocol. This layer is responsible for the format of datagrams as defined by IP, and routing and forwarding a datagram or packet to the next hop, but is not responsible for the accurate and timely delivery of datagrams to the destination in proper sequence. IP allows raw transmission functions allowing user to add functionalities necessary for given application, ensuring maximum efficiency.

1.5.1 Need for IP Address

The primary goal of the Internet is to provide an abstract view of the complexities involved in it. Internet must appear as single network of computers. At the same time network administrators or users must be free to choose hardware or various internetworking technologies like Ethernet, Token ring etc. Different networking technologies have different physical addressing mechanisms. Therefore, identifying a computer on Internet is a challenge. To have uniform addressing for computers over the Internet, IP software defines an IP address which is a logical address. Now, when a computer wants to communicate to another computer on the Internet, it can use logical address and is not bothered with the physical address of the destination and hence the format and size of data packet.

1.5.2 Classes of IP Address

Internet addresses are 32 bits long, written as four bytes separated by periods (full stops). They can range from 0.0. 0.0 to 223. 255. 255. 255. It's worth noting that IP addresses are stored in big-endian format, with the most significant byte first, read left to right. This contrasts with the little-endian format used on Intel- based systems for storing 32- bit numbers. This minor point can cause a lot of trouble for PC programmers and others working with raw IP data if they forget.

IP addresses comprise two parts, the network ID and the host ID. An IP address can identify a network (if the host part is all zero) or an individual host. The dividing line between the network ID and the host ID is not constant. Instead, IP addresses are split into five classes, which allow for a small number of very large networks, a medium number of medium- sized networks and a large number of small networks. The classes of IP address are briefly explained below, the structure of these classes are also shown in *Figure 4*.

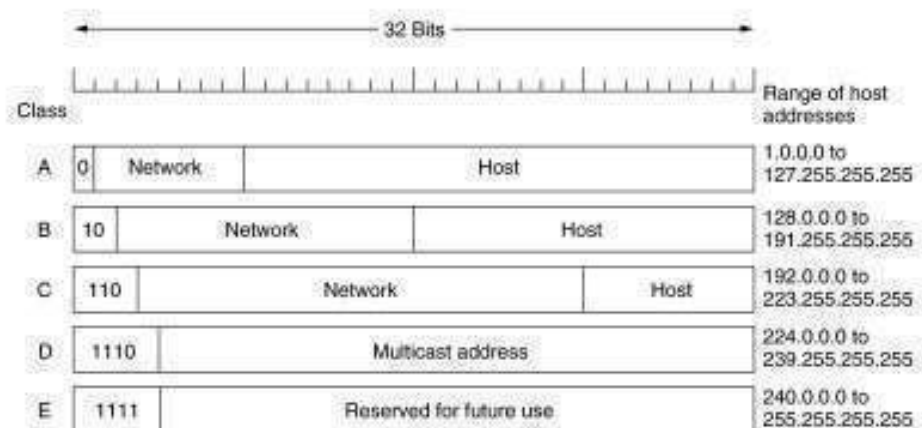


Figure 4: Classes of IP address

Class A addresses have a first byte in the range 0 to 127. The remaining three bytes can be used for unique host addresses. This allows for 126 networks each up to 16 million hosts.

Class B addresses can be distinguished by first byte values in the range 128 to 191 in these addresses, the first two bytes are used for the net ID, and the last two for the host ID, giving addresses for 16,000 networks, each with up to 65536 hosts.

Class C addresses have a first byte in the range 192 to 223. Here, the first three bytes identify the network, leaving just one byte for the individual hosts. This provides for 2 million networks of up to 256 hosts each.

Class D addresses have a first byte in the range 224 to 239. It is designed for multicasting.

Class E addresses have a first byte in the range 240 to 255. It is reserved for future purposes. The concept is shown here.

Although these addresses make it possible to uniquely identify quite a lot of networks and hosts, the number is not that large in relation to the current rate of expansion of the Internet. Consequently, a new addressing system has been devised which is a part of Internet Protocol version 6 (IPv6). IPv6 won't come into use for a couple of years, and understanding it isn't essential to understanding how IP works in general.

IP addresses can be further divided to obtain a subnet ID. The main net ID identifies a network of networks. The subnet ID lets you address a specific network within that network. This system of addressing more accurately reflects how real-world large networks re-connected together.

You decide how the subnet ID is arrived at by defining a 32-bit value called the subnet mask. This is logically ANDed with the IP address to obtain the subnet address. For example, if a subnet mask was 255.255.255.0 and an IP address was 128.124.14.5, 128.124 would identify the Class B network, 128.124.14 would identify the sub network, and 5 would identify the host on that sub network.

1.5.3 Special Meanings

A few IP addresses have special meanings. A network ID of 0 in an address means "this network", so for local communication only the host ID need be specified. A host ID of 0 means "this host".

A network ID of 127 denotes the loopback interface, which is another way of specifying "this host". The host ID part of the address can be anything in this case, though the address 127.0.0.1 is normally used. Packets sent to the loopback address will never appear on the network. It can be used by TCP/IP applications that run on the same machine and want to communicate with one another.

Addresses in the range 224.x.x.x to 239.x.x.x are Class D addresses, which are used for multi-casting. Addresses 240.x.x.x to 247.x.x.x are reserved for experimental purposes.

Net, subnet and host ID's of all binary ones (byte value 255) are used when an IP packet is to be broadcast. Mercifully, an address of 255.255.255.255 does not result in a broadcast to the entire Internet.

Three sets of addresses are reserved for private address space - networks of computers that do not need to be addressed from the Internet. There is one class A address (10.x.x.x), sixteen class B addresses (172.16.x.x to 172.31.x.x), and 256 class C addresses (192.168.0.x to 192.168.255.x). If you have equipment which uses IP addresses that have not been allocated by InterNIC then the addresses used should be within one of these ranges, as an extra precaution in case router misconfiguration allows packets to "leak" onto the Internet.

1.5.4 Who Decides the IP Addresses?

IP address of two Computers over the Internet is never same. To ensure this, there is a central authority that issues the IP address. An organisation or individual wanting to connect to the Internet needs to contact local ISP for obtaining a unique IP address at the Metadata the global level, Internet Assigned Number Authority (IANA) allocates a net id to the ISP.

1.5.5 Internet Protocol

IP is the transmission mechanism used by TCP/IP protocols for host-to-host communication Packets in IP layer are called datagrams. *Figure 5* shows the IP datagram format:

Version 4bits	HLEN 4 bits	Service type 4 bits	Total Length 16 bits	
Identification 16 bits			Flags 3 bits	Fragment Offset 13 bits
Time to Live 8 bits	Protocol 8 bits		Header Checksum 16 bits	
Source IP Address 32 bits				
Destination IP Address 32 bits				
Data				
Options				

Figure 5: IP datagram

A brief description of header fields in order is given below:

- **Version (4 bits):** It defines the version of IP protocol. Currently, the version is 4(IPv4), indicated by value 4. In future it would contain 6, for IPv6.
- **HLEN (4 bits):** It is needed because length of header is variable. When the header size is 20 bytes, its value is 5($5 \times 4 = 20$). With options, the maximum size is 60 bytes, when the value is 15 ($15 \times 4 = 60$). Each value represents number of 32-bit words.
- **Service Type (8 bits):** It is used to define type of service in terms of reliability, precedence, delay and cost.
- **Total length (16 bits):** it defines the total length of IP datagram. The maximum value can be $2^{16} = 65536$ bytes.
- **Identification (16 bits):** This field is used to unusually identify a datagram. It is useful to know the fragments belonging to same datagram fragments that are part of a datagram which contain same value in identification fields, so that they can be put together in the order to **reassemble** the datagram at receiver.
- **Flags (3 bits):** This field is used to uniquely identify a datagram. It is useful to know the fragments belonging to same datagrams.
- **Fragmentation Offset (13 bits):** It is a pointer that indicates the offset of the fragment in the original datagram before fragmentation.
- **Time to Live (8 bits):** It is used to control the maximum number of hops visited by the datagram. It is needed to restrict a datagram from continuing to travel in infinite loop without reaching the destination. This infinite looping may cause network congestion. This field limits the lifetime of datagram, after which the packet is discarded, so that datagram does not travel in infinite loop.

- **Protocol (8 bits):** An IP datagram may encapsulate data from various higher-level protocols like TCP, UDP, ICMP, and IGMP. This field specifies the final destination protocol to which the IP datagram should be delivered. Each protocol TCP, UDP etc. identified with a unique number.
- **Source Address (32 bits):** It stores the IP address of the source.
- **Destination Address (32 bits):** It stores the IP address of the final destination.
- **Options:** This field contains optional information such as routing details, timestamp etc. For instance, it can store route of a datagram, in the form of IP addresses of intermediate routers, optionally the time when it pass through that router.

The functions performed at this layer are as follows:

- **Define the datagram, which is the basic unit of transmission in the Internet:** The TCP/IP protocols were built to transmit data over the ARPANET, which was a *packet switching network*. A *packet* is a block of data that carries with it the information necessary to deliver it in a manner similar to a postal letter that has an address written on its envelope. A packet switching network uses the addressing information in the packets to switch packets from one physical network to another, moving them towards their final destination. Each packet travels the network independently of any other packet. The *datagram* is the packet format defined by IP.
- **Define the Internet addressing scheme:** IP delivers the datagram by checking the destination address in the header. If the destination address is the address of a host on the local network, the packet is delivered directly to the destination. If the destination address is not on the local network, the packet is passed to a router for delivery. Router is a devices that switch packets between the different physical networks. Deciding which router to use is called *routing*. IP makes the routing decision for each individual packet.
- **Move data between the Network Access Layer and the Host-to-Host Transport Layer:** When IP receives a datagram that is addressed to the local host, it must pass the data portion of the datagram to the correct host-to-host transport layer protocol. This selection is done by using the *protocol filed* in the datagram header. Each host-to-host transport layer protocol has a unique number that identifies it to IP.
- **Route datagrams to remote hosts:** Internet gateways are commonly (and perhaps more accurately) referred to as IP routers because they use IP to route packets between networks. In traditional TCP/IP jargon, there are only two types of network devices: gateways and hosts. Gateways forward packets between networks and hosts do not. However, if a host is connected to more than one network (called a *multi-homed host*), it can forward packets between the networks. When a multi-homed host forwards packets, it acts like any other gateway and is considered to be a gateway.
- **Fragment and reassemble datagrams:** As a datagram is routed through different networks, it may be necessary for the IP module in a gateway to divide the datagram into smaller pieces. A datagram received from one network may be too large to be transmitted in a single packet on a different network. This condition only occurs when a gateway interconnects dissimilar physical networks. For example, a physical network may be using Ethernet which specify packet (Frame) size of main 1500 bytes. Other physical network may be using FDDI that specify packet size **mark** of 4464 bytes. Therefore, the MTV of Ethernet to 1500 & MTU of FDDI is 4464. An IP packet-being transmitters over Ethernet is fragments limit 1500 bytes each.

Each type of network has a *maximum transmission unit (MTU)*, which is the largest packet it can transfer. If the datagram received from one network is longer than the

other network's MTU, it is necessary to divide the datagram into smaller fragments for transmission. This division process is called *fragmentation*.

1.5.6 Address Resolution Protocol (ARP)

We have seen that IP address makes the addressing uniform on the Internet. Routing of packets is done using the IP addresses of the packet. However, communication in a local network is broadcast, which is done using physical address. Therefore, when the packet reaches the destined network, there must be a process of obtaining the physical address corresponding to its IP address, of a computer in order to finally deliver the datagram to the destined computer. The physical address corresponding to an IP address is resolved by using address resolution protocol (ARP). ARP maps given IP address to a physical address as shown in the *Figure 6*. It takes host's IP address as input and gives its physical address as output.

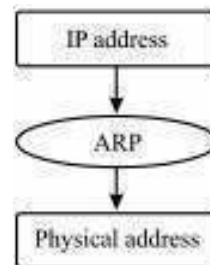


Figure 6: ARP maps the IP address to the physical address

ARP assumes that every host knows its IP address and physical address. Any time a host needs to know the physical address of another host on the network, it creates an ARP packet that includes the IP address X of the destination host asking—Are you the one whose IP address is X? If yes, please send back your physical address. This packet is then broadcasted over the local network. The computer, whose IP address matches X, sends a ARP reply packet, with its physical address. All the other hosts ignore the broadcast. Next time the host needs to send a datagram to the same destination, it need not broadcast an ARP query datagram; instead it can look up in its ARP cache. If the mapping is not found in the cache, then only the broadcast message is sent.

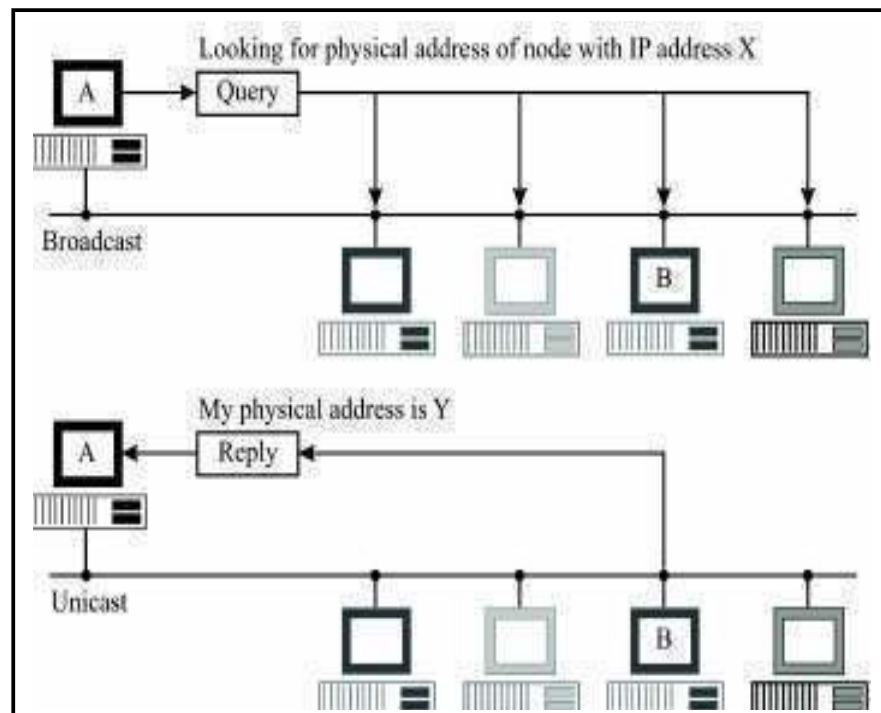


Figure 7: ARP query and reply

1.5.7 Reverse Address Resolution Protocol (RARP)

This protocol performs the job exactly opposite to ARP. It maps a physical address to its IP address as shown in *Figure 8*. Where is this needed? A node is supposed to have its IP address stored on its harddisk. However, there are situations when the host may not have hard disk at all, for example a diskless workstation. But also when a host is being connected to the network for the first time, at all such times, a host does not know its IP address. In that case RARP find out the IP address, this process is shown in *Figure 9*.

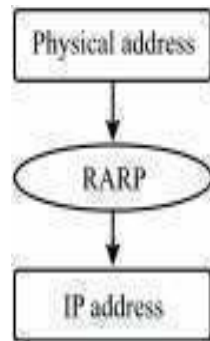


Figure 8: RARP maps the physical address to the IP address

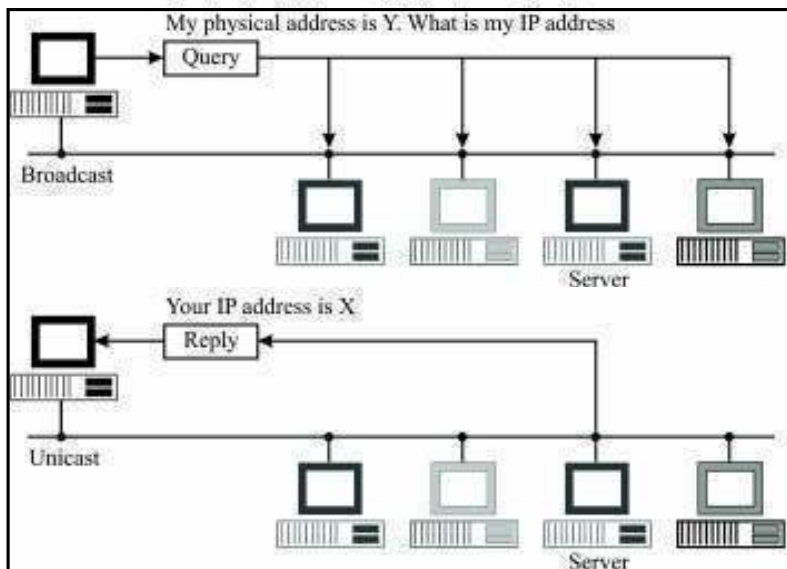


Figure 9: RARP query & reply

1.5.8 Internet Control Message Protocol (ICMP)

IP is best effort data delivery protocol. It means that IP makes best effort of delivering a datagram from source to destination. However, it does not guarantee that the datagram would be delivered correctly. IP has no error-reporting mechanism. A router or the final destination needs to inform the source if it must discard a datagram. IP also lacks a mechanism for host and management queries. A host sometimes needs to determine if a router or another host is alive to avoid sending datagrams to a router which is down. ICMP has been designed to report error occurred doing delivery of datagram.

The errors reported by ICMP are generally related to datagram processing. ICMP only reports errors involving fragment 0 of any fragmented datagrams. The IP, UDP or TCP protocols will usually take action based on ICMP messages. ICMP generally belongs to the IP layer of TCP/IP but relies on IP for support at the network layer. ICMP messages are encapsulated inside IP datagrams.

ICMP will report the following network information:

- Timeouts
- Network congestion
- Network errors such as an unreachable host or network.

The ping command is also supported by ICMP, and this can be used to debug network problems.

ICMP Messages

The ICMP message consists of an 8-bit type, an 8-bit code, an 8-bit checksum, and contents which vary depending on code and type.

ICMP is used for many different functions, the most important of which is error reporting. Some of these are “*port unreachable*”, “*host unreachable*”, “*network unreachable*”, “*destination network unknown*”, and “*destination host unknown*”. Some not related to errors are:

- **Timestamp request and reply** allows one system to ask another one for the current time.
- **Address mask and reply** is used by a diskless workstation to get its subnet mask at boot time.
- **Echo request and echo reply** is used by the ping program to see if another host is reachable. Thus two types of ICMP messages are defined: Error messages and Query messages.
- **Source Quench Message:** When datagrams arrive too quickly for processing, the destination host or an intermediate gateway sends an ICMP *source quench message* back to the sender. This message instructs the source to stop sending datagrams temporarily.
- **Redirect routes:** A gateway sends the ICMP *redirect message* to tell a host to use another gateway, presumably because the other gateway is a better choice. This message can only be used when the source host is on the same network as both gateways.

1.6 TRANSPORT LAYER

The transport layer runs on top of the Internet layer and is concerned with process-to-process delivery of data packets. Here, process is a running application program on a host. The main task of transport layer is to ensure correct delivery of packets. This introduces several responsibilities like flow control mechanism. By flow control we mean that a faster sender must not drown a slow receiver with data packets resulting in data loss. The transport layer also provides connection mechanism. It establishes connection with the receiver, transfers data, and terminates the connection. The transport layer includes acknowledgement service to check for packet loss in the network. In TCP/IP transport layer is represented by two protocols: TCP and UDP. UDP is simpler of the two protocols. It is unreliable, connectionless transport protocol. Unreliability means that it does not provide any flow control, error control, and acknowledgement services. Connectionless means that no connection is established between the sender and the receiver prior to sending data. If UDP detects an error, it silently drops the packet. What is the use of UDP if it is so unreliable? It is very useful when the speed of the delivery is critical and loss of a small number of packets is tolerable like images, voice or video. TCP is useful when it is necessary to ensure error free delivery of packets from source to the destination.

1.6.1 Transmission Control Protocol

We have seen that IP packets may travel through different routes and may arrive out of sequence. TCP puts the packets in sequence. An intermediate router may discard the IP packets and they may not arrive at the destination at all. TCP checks for missing packets and handles this issue retransmission request. Some packets may get duplicated due to hardware malfunction. TCP discards duplicate packets. Hence, TCP takes care of all these situations and makes the Internet reliable. Before discussing TCP and UDP in detail, let us discuss process to process communication.

Process to Process Communication

IP provides host-to-host communication. TCP provides process-to-process communication using the client server paradigm as shown in *Figure 10*, suppose out of several application programs running on the host, TCP delivers the data packet to the destined application program.

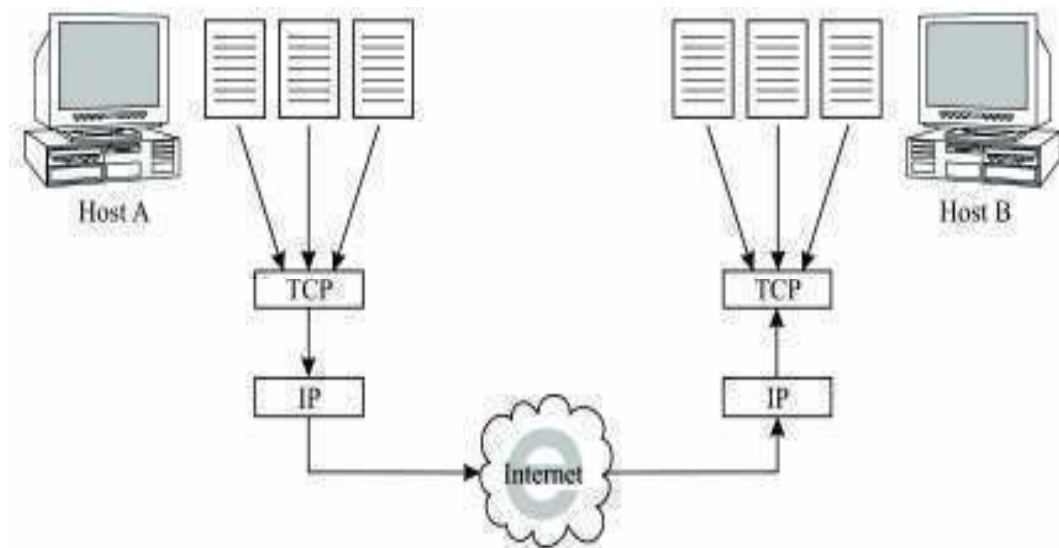


Figure 10: Domain of IP and TCP/UDP TCP like a multiplexer

Applications running on different hosts communicate with TCP with the help of concepts called ports. A port is a unique 16-bit number allocated to an application program. It is used when an application wants to open a TCP connection with another application on a remote computer. To understand how, let us take an analogy. A wants to call B staying in a hotel X, room no.Y. Now A dials the phone number of the hotel, after getting a response from the operator, A must tell the room number of B to be able to connect to B. This information is used to redirect the call to B.

Now, suppose an application X on host A wants to communicate to application Y on host B as given in *Figure 11* below. It must first reach B and then Y. Reaching B is not enough because there may be multiple applications running on B which might want to communicate with another application running on some other host. Hence, specifying the host is not enough.

A process on local host, called client, needs some services from a process on remote host called server. The client uses a random port number called ephemeral port number. The server also defines a port number with itself, which is known to all, and not random. This is because a client wanting to access a particular service will not know the port number if it is random. Therefore, TCP uses well-known port numbers. This also allows multiple TCP connections between two hosts are possible. Although the IP address would be same, the port number would be different each time.

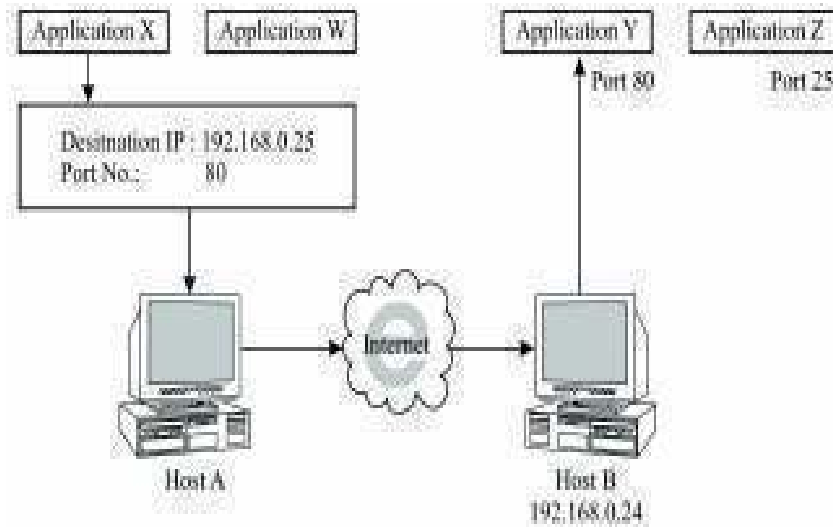


Figure 11: Use of port

The application X now provides the IP address of B, to identify the host B from thousands of hosts on the Internet. It provides port number, corresponding to application Y running on B to uniquely identify the application.

Sockets

The term Socket is used to identify the IP address and Port number concatenated together as shown in Figure 12.

Socket=IP address +port number

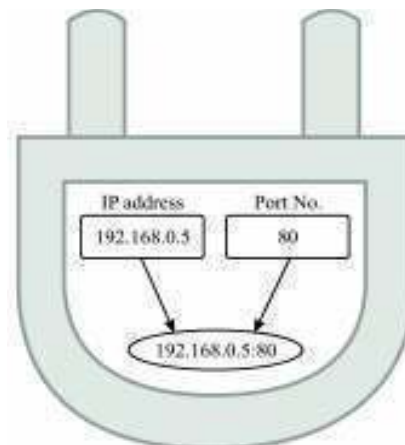


Figure 12: Socket

Socket Address

A client socket identifies a client process uniquely and a server socket identifies server process uniquely. Hence, a pair of sockets identifies a TCP connection between two applications on two different hosts.

Services Provided by TCP

- **Process to process communication:** TCP provides process-to-process communication as discussed above in section 1.6.1.
- **Stream Delivery Service:** TCP is stream oriented protocol. It allows the sending process to deliver data as a stream of bytes and allows receiving process to obtain data as a stream of bytes. For this, TCP needs two buffers, the sending buffer, and the receiving buffer, one for each direction. Figure 13 shows the use buffers in TCP stream delivery.

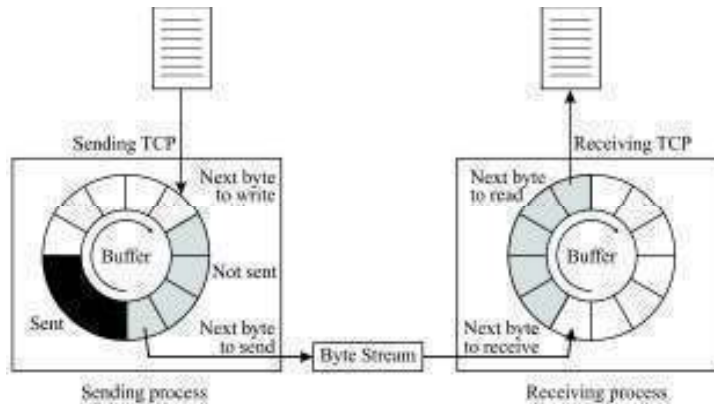


Figure 13: Stream Delivery Service

At the sending site, the buffer can have three types of chambers. The white section is the section of empty bytes which can be filled by the sender. The gray section contains bytes that have been sent but not acknowledged. They are kept in the buffer until the ACK is received. The black area contains the bytes to be sent by the sending TCP.

At the receiving site, the buffer is divided into two sections. The white section is the section of empty bytes which can be filled by the received bytes. The black section contains the bytes which can be read by the receiving process. It handles the speed mismatch between the sender and the receiver. There is one more step before the data can be sent.

IP, the service provider of TCP, sends data as packet, and not as stream of bytes. TCP groups a number of bytes into a segment, add a TCP header and delivers the packet to the IP layer for transmission. These segments can be of different sizes. The segment is now encapsulated inside a IP datagram and transmitted.

- Connection Oriented Service:** When a process host A wants to communicate to a process on host B, the two TCPs establish a connection, data is exchanged, and the connection is terminated. This connection is virtual and not physical. All the IP datagrams may follow different routes, may arrive out of sequence. TCP take the responsibility of delivering the bytes in order. Hence maintaining a virtual connection. TCP provides full duplex communication. Data can flow in both the directions.
- Reliability:** TCP is a reliable transport protocol. TCP incorporates error control, flow control, and acknowledgement services.

TCP Connections

We have said again and again that TCP is connection oriented. You may wonder how TCP connections orients as it uses the services of IP, a connectionless service. It is because TCP connection is virtual and not physical. It uses the services of IP for packet delivery but controls the connection itself. It takes care of reordering, retransmission, duplication, of which IP is unaware.

Let's see this connection establishment in detail.

TCP uses a technique called **three-way handshaking**. It means that three messages are exchanged between the sender and the receiver to establish the connection as shown in *Figure 14*. After this, TCP guarantees a reliable data transfer. It has been proved that this three way handshake is necessary and sufficient for establishing a successful connection.

The server waits passively to accept any active connection request. The client always initiates the connection request. The client sends the SYN segment in which only the SYN flag is set. This segment is used for the synchronisation of sequence numbers. The client chooses a random number and sends it as the first sequence number. It does not carry any relevant data, and consumes one sequence number.

The server sends the SYN ACK segment, with SYN and ACK bits set by sending this, server informs the client about the sequence number it will use in future and acknowledges the receipt of first SYN segment. It also consumes one sequence number.

The client sends the third segment, an ACK segment. It is an acknowledgement for the second segment. It bears the same sequence number as the first segment and hence, does not consume any sequence number. After this handshake, communication can start between the two ends.

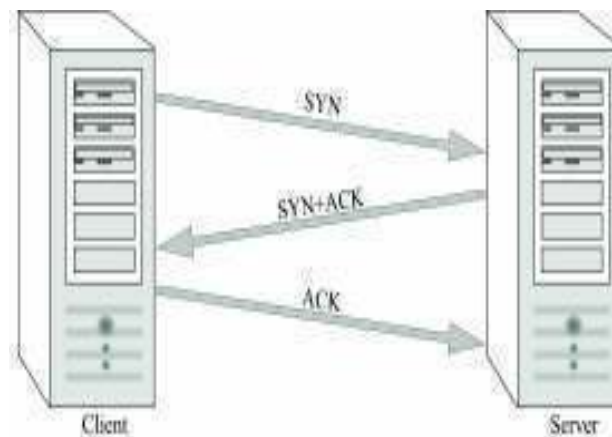


Figure 14: Three-way handshaking

The format of a TCP segment is shown in *Figure 15*. The header size is between 20 to 60 bytes, followed by data. It contains 20 bytes if the header does not contain any option.

Source Port 16 bits							Destination Port 16 bits						
Sequence Number 32 bits													
Acknowledgment Number 32 bits													
HLEN 4 bits		Reserved 6 bits		URG	ACK	PSH	RST	SYN	FIN	Window size 16 bits			
Checksum 16 bits							Urgent pointer 16 bits						
Data													
Options													

Figure 15: TCP segment format

Here is a brief description of header fields in order.

Source port number (16 bit): It specifies the source port number corresponding to the application which is sending the segment.

Destination Port Number (16 bit): It specifies the port number of the destination computer, corresponding to the receiving applications.

Sequence Number (4 bytes): It specifies the number assigned to the first byte of the data portion contained in this TCP packet. Each byte to be transmitted is numbered in

an increasing sequence. It tells the destination host which byte comprises the first byte of the TCP segment. During the connection establishment phase, the source and the destination generate a unique random number. If this random number is 3000, and the first segment contains 2000 bytes, then the sequence no. will be 3002. 3000 and 3001 are used in connection establishment. The second segment would have a sequence number of 5002(3002+2000), and so on.

Acknowledgement Number (4 bytes): On receiving a packet with sequence number X, the receiver sends back X+1 as the acknowledgement number. It defines the sequence number which the receiver is expecting next.

Header length (4 bits): The header length can be between 20 to 60 bytes. therefore, the value of this field can be between 5 ($5 \times 4 = 20$) and 15 ($15 \times 4 = 60$)

Reserved: These 6 bits are reserved for future use.

Flag (6 bits): This field signifies 6 control flags, each one of them occupying one bit. Out of these, two are most important. The SYN flag indicate that the source wants to establish a connection with the destination. The FIN flag means that the sender wants to close the TCP connection.

Window Size (2 bits): This field determines the size of the window the other party must maintain. It is useful for flow control.

Checksum (16 bits): It contains the checksum for error detection.

Urgent Pointer: This field is used in situations when the segment contains urgent data. It specifies the number that must be added to obtain the number of the first urgent byte in the data section of the segment.

Pseudoheader: It is the part of the header of the IP packet in which the segment is encapsulated with some fields set to zero. It is added to have better error control. This way we ensure that if the IP header is corrupted, the segment is not delivered to the wrong host.

The protocol field is added to ensure that the packet belongs to TCP and not UDP because a process can use either TCP or UDP, the destination port number can be same. The value of this field is 6. If this value is changed, the checksum calculation will detect it and discard the packet instead of delivering it to the wrong protocol.

1.6.2 User Datagram Protocol (UDP)

The other protocol in transport layer is UDP. UDP is connectionless protocol. It allows a computer to send data without needing to establish a virtual connection. There is no error checking except for checksum. It does not provide sequencing, flow control or acknowledgements. Thus, UDP packets may be lost, arrive out of sequence, or duplicated. It is left to the application program to take care of these issues.

Source Port 16 bits	Destination Port 16 bits
UDP length 16 bits	UDP checksum 16 bits
Data	

Figure 16: UDP packet format

Here is a brief description of header fields of DP packet is shown in *Figure 16*.

Source port number (16 bit): It specifies the source port number corresponding to the application which is sending the segment.

Destination Port Number (16 bit): It specifies the port number of the destination computer, corresponding to the receiving applications.

Total Packet Length (16 bits): It defines the total length of the UDP packet. However, this field is redundant, because there is a packet length field in IP, which encapsulates the UDP packet. Therefore UDP packet length=IP packet length-IP header length. This field is retained as an additional check.

Checksum: It is again used for error detection.

Pseudoheader: This field is added for the same purpose as in TCP.



Check Your Progress 2

- 1) What are the functionalities of Internet layer?

.....

.....

.....

.....

.....

.....

.....

- 2) What do you mean by ICMP source quench message

.....

.....

.....

.....

.....

.....

.....

1.7 APPLICATION LAYER

Till now, we have studied Internet layer and Network layer protocols which provide end-to-end delivery of packets. These protocols would have no meaning without the Application Layer Protocols like DNS, HTTP, TELNET, and FTP. We will discuss all the application layer protocols in brief.

1.7.1 Electronic Mail

Email is one of the most popular Internet services. At the beginning of the Internet era, emails were short and consisted of text only. Today it is much more complex allowing to send text, audio and video. It also allows to send a message to more than one recipient.

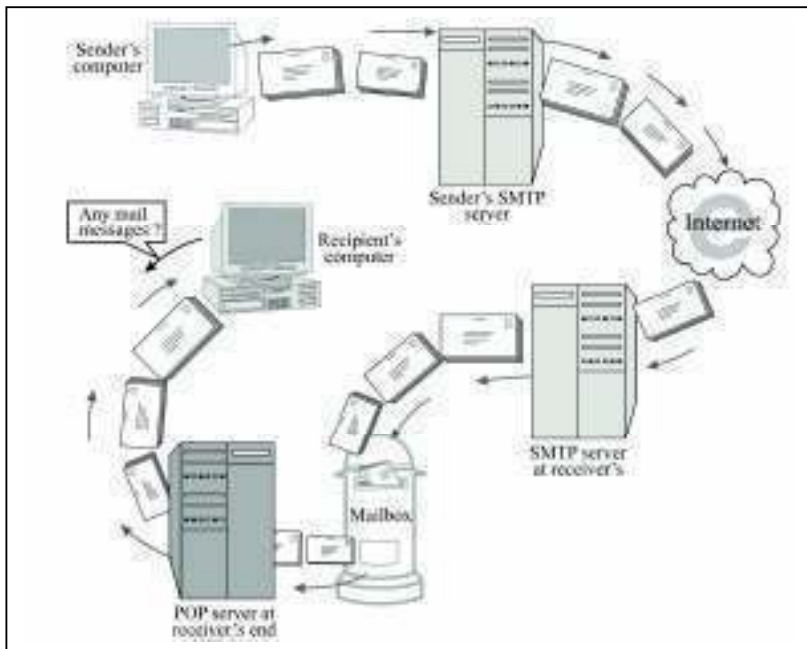


Figure 17: Journey of email message

In a typical scenario, both the sender and recipient are users on two different systems, and are connected to the system via a point to point WAN, or a LAN, which uses an email server for handling emails. Hence the message needs to be sent over the internet. How the sender computer sends the message through internet to the receipts computer is shown in *Figure 17*.

The sender uses an USER AGENT PROGRAM to prepare a message. It then sends the message through the LAN or WAN. This is done through a pair of message transfer agents (client and server). Whenever the sender has a message to send, he calls the user agent, which in turn, calls the Mail Transfer Agent (MTA) client. The MTA client establishes a connection with the MTA server, which is running all the time. The system at sender's site queues all the messages received. It then uses an MTA client to send message to the system at receiver's site. After the message arrives at Bob's mail server, another client server agents comes into picture which are called Message Access Agents (MAA). The receiver sends a request to the MAA server, which is running all the time, and requests the transfer of messages.

The email transfer protocols

For email messaging, every domain maintains an email server. The server runs protocols software that enable email communication. There are two main emails protocols: POP and SMTP. Because both the email protocol software programs run on server computers, the server computers are themselves called POP server and SMTP server. A single server can host both the POP and SMTP server programs.

SMTP server (Simple Mail Transfer Protocol)

SMTP is the Internet protocol used to transfer electronic mail between computers. The second generation of SMTP is called ESMTP (for Extended SMTP), but the differences are not important for this introduction.

It actually transfers the email message from the SMTP server of the sender to the SMTP server of the recipient. Its main job is to carry the message between the sender and the receiver. It uses TCP/IP underneath. That is, it runs on top of TCP/IP.

At the sender's site, an SMTP server takes the message sent by a user's computer.

The SMTP server at the sender's end then transfers the message to the SMTP server of the recipient.

The SMTP server at the recipient's end takes the message and stores it in the appropriate user's mailbox.

POP server

The Post Office Protocol provides a standard mechanism for retrieving emails from remote server for a mail recipient. Suppose that a home user X usually connects to the Internet using a dial-up connection to an ISP. Also, another person Y has sent an email to X, when X is not connected to the Internet. Now, the email message gets stored in the mailbox of the user provided by the ISP.

When X connects to the Internet next time and wants to see the new mails that have arrived for him since the last time he had connected to the Internet, he opens his email program. That email client program invokes POP client, which contacts the POP server. The email client opens the mailbox for the user X and sends the emails arrived for him to the POP client.

POP server is needed because SMTP server expects the destination host to be online all the time so that it can make a TCP connection to it, and forward the mail. But desktop computers are usually powered off after business hours. Hence the SMTP server forwards the mails to the mailbox, and the POP server retrieves the mails from the respective email boxes of the user when requested by POP clients.

POP3 (version 3). This is simple and limited in its functionality. It has two modes: the delete mode and the keep mode. In delete mode, mail is deleted from the mailbox after retrieval. It is used when the user is working at his personal computer and can save and organise the information after reading or replying. The keep mode is used when the user would check his mail from his primary computer. The mail is read but kept in the system for later retrieval.

IMAP Protocol

Another mail access protocol is the Internet Mail Access protocol, version 4 (IMAP4).

It is similar to POP3, but it has more features as given below. In this, a user can-

- check the email header before downloading.
- search the contents of the email for a specific keyword prior to downloading.
- partially load the email it is helpful if the bandwidth is limited.
- create, rename, and delete mailboxes on the mail server.
- create a hierarchy of mailboxes in a folder for email storage.

File Transfer Protocol

We have seen how email works in *Figure 17*. But there are situations when we want to receive or send a file from or to a remote computer. Emails are just short messages. In fact, the greatest volume of data exchange in the Internet today is due to file transfer. Special software and a set of rules called File Transfer Protocol (FTP) exists for this purpose. It is an application layer protocol that is aimed at providing a very simple interface for any user of the Internet to transfer files. Whether you know it or not, you most likely use FTP all the time.

Although transferring files from one system to another seems simple and straightforward task, some problems must be dealt with first. For example, two systems may use different file name conventions. Two systems may have different way to represent text or data. Two systems may have different directory structures. All these problems are elegantly solved by FTP.

FTP differs from other application layer protocols in one respect. All the other application layer protocols use a single connection between client and server. However, FTP uses two TCP/IP connections. One connection is used for actual data transfer, and the other is used for control information as shown in *Figure 18*. This makes FTP more efficient.

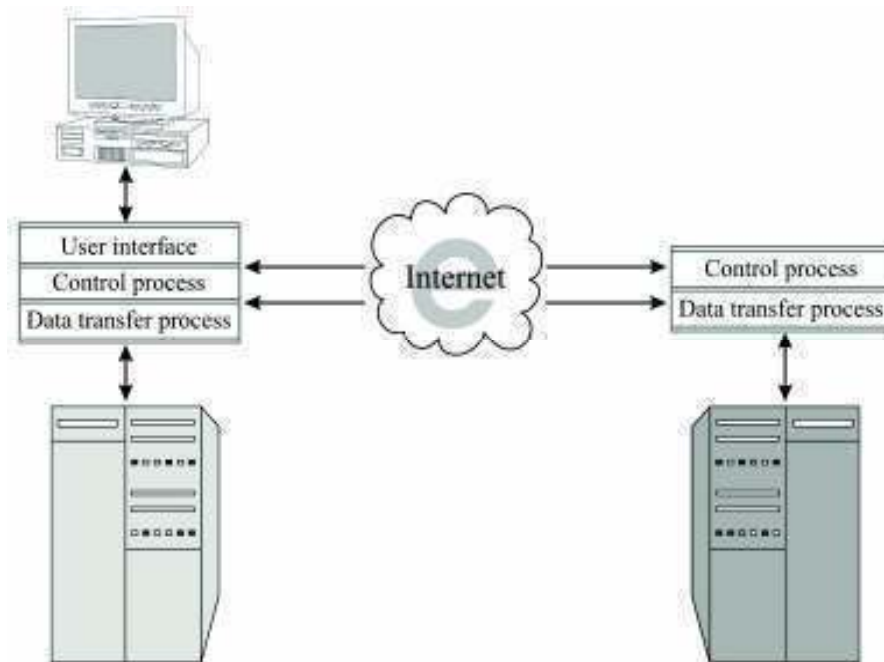


Figure 18: Data transfer through Internet

The components of the client and the server are shown in the *Figure 18*. The user interface component is not required at the server side, since there is no interaction exactly at the time of file transfer. The TCP control connection is made between the control processes of the client and the server. The TCP data transfer connection is made between the data transfer processes of the client and the server. While the file is being transferred, the server keeps track of how many bytes have been sent, and how much are remaining. It sends this information on the control connection, and this way reassures the user about the progress of data transfer. If multiple files are to be transferred in a single FTP session, then the control connection remains active throughout the entire FTP session. The data connection is opened and closed for each file transferred.

Prior to data transfer, the following attributes need to be specified:

Type of the file to be transferred: The file to be transferred can be an ASCII, EBCDIC, or an image file. Image file is actually a misnomer. It has nothing to do with the image file. Instead, it signifies a binary file that is not interpreted by FTP in any manner, and it is sent as it is. If the file has to be sent as ASCII or EBCDIC, then the destination must be ready to accept in that mode. If the file has to be transferred without any regard to its content, then the third mode is used.

The structure of the data: File can be transferred by FTP by interpreting its structure in the following ways:

Byte-oriented structure: The file can be transferred as a continuous stream of data, where no structure is assumed.

Record-oriented structure: The file is divided into records and the records are transferred one by one.

The transmission mode: FTP can transfer a file using any of the three modes as described.

Stream mode: This is the default mode; the data is delivered from FTP to TCP as a continuous stream of bytes.

Block mode: The data can be delivered from FTP to TCP as a block. Each data block follows a three byte header. The first byte is the descriptor, whereas the remaining two bytes defines the size of the block.

Compressed mode: The file can be compressed before sending. Normally Run Length Encoding is used for compressing a file.

This information is used by FTP to resolve the heterogeneity problem.

WU-FTP, CuteFTP is most commonly used FTP software applications.

What is an FTP Site?

An FTP site is like a large filing cabinet. With a traditional filing cabinet the person who does the filing has the option to label and organise the files, however they see fit. They also decide which files to keep locked and which remain public. It is the same with an FTP site.

The virtual 'key' to get into FTP site is the UserID and Password. If the creator of the FTP site is willing to give everyone access to the files, the UserID is 'anonymous' and the Password is your e-mail address (e.g. name@domain.com). If the FTP site is not public, there will be a unique UserID and Password for each person who is granted access.

When connecting to an FTP site that allows anonymous logins, you're frequently not prompted for a name and password. Hence, when downloading from the Internet, you most likely are using an anonymous FTP login and you don't even know it.

To make FTP connection you can use a standard Web browser (Internet Explorer, Netscape, etc.) or a dedicated FTP software program, referred to as an FTP 'Client'.

When using a Web browser for an FTP connection, FTP uploads are difficult, or sometimes impossible, and downloads are not protected (not recommended for uploading or downloading large files).

When connecting with an FTP Client, uploads and downloads couldn't be easier, and you have added security and additional features. For once, you're able to resume a download that did not successfully finish, which is a very nice feature for people using dial-up connections who frequently lose their Internet connection.

1.7.2 Domain Name System (DNS)

Machines on the Internet are identified by numerical "IP addresses" like 192.0.2.1. Domain names make it possible to refer to machines by a name rather than a number, which is definitely easier. If we had to remember the IP addresses of all of the Web sites we visit every day, we would all go nuts. Human beings just are not that good at remembering strings of numbers. In early days of Internet, all host names and associated IP addresses were recorded in a single file called hosts.txt, which was maintained by NIC (Network Information Center) in the US. Every night all the hosts attached to the Internet would obtain a copy of this file to refresh their domain name entries. As the internet grew, it became impossible to keep it up-to-date. To solve this problem, Domain Name System was developed. This DNS is consulted whenever any message is sent to any other computer on the Internet. It simply gives the mapping of domain names to IP addresses. Domain name servers translate domain names to IP addresses. That sounds like a simple task, and it would be able to handle four things:

- There are billions of IP addresses currently in use, and most machines have a human-readable name as well.
- There are many billions of DNS requests made every day. A single person can easily make a hundred or more DNS requests a day, and there are hundreds of millions of people and machines using the Internet daily.
- Domain names and IP addresses change daily.
- New domain names get created daily.

The DNS Name Space

The problem of mapping host names to IP addresses is very difficult given the numbers of machines on the Internet.

The postal system faces a similar challenge, which it has dealt by requiring the sender to specify the country, state, city, street name, house no. Internet uses the same principle. The Internet is divided into top-level domains, such having several hosts underneath. Again, each domain is divided into sub-domains, which can be further classified into sub-domains. This creates a tree-like structure which is shown in *Figure 19*.

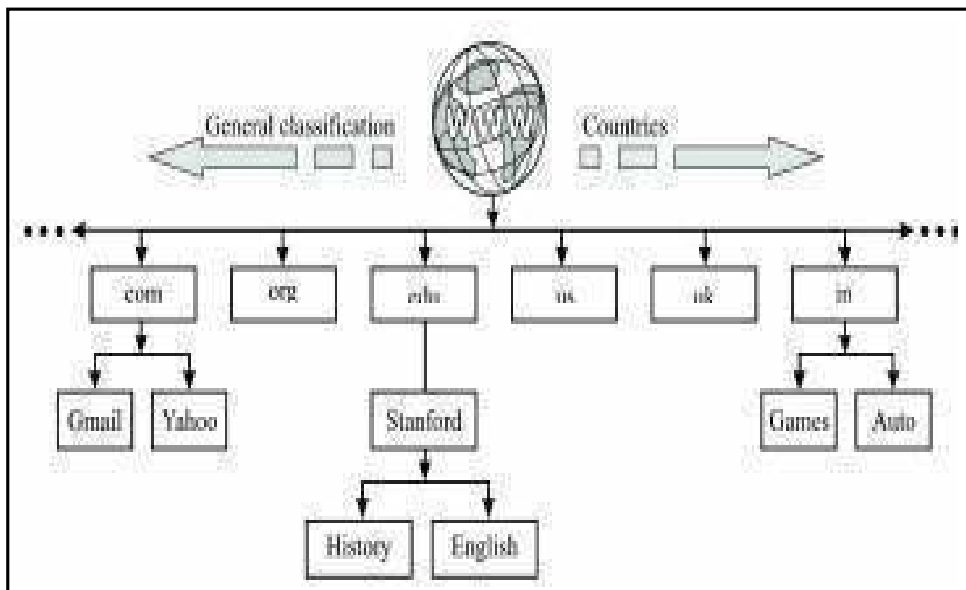


Figure 19: Domain Name space

The topmost domains are classified as: *generic*, and *countries*. The generic domains are classified into com (commercial), gov (the US federal government), edu (educational), org (non-profit organizations), net (network, etc. The country domains specify one entry for each country these clarifications are also shown in *Figure 20*. For example, uk (United Kingdom), in (India) and so on.

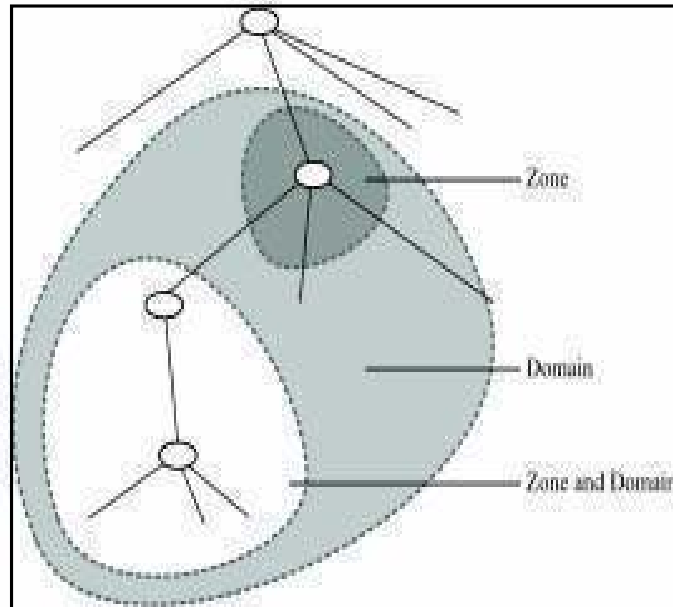


Figure 20: Zones and domains in DNS

For registering a name called *bajaj* under the category *auto*, which is within *India(in)*, the full path will be *bajaj.auto.in*

Each domain is fully qualified by the path upward from it to the top most root. The names within a full path are separated by dot. Remember domain names are case-insensitive.

DNS Server

The information is contained in the domain name space. It must be stored at some place, but it would be inefficient and unreliable to store all the information at one place. The solution is to distribute the information among many computers called DNS servers. Thus, every domain has a domain name server.

Primary and Secondary DNS servers

DNS defines two types of servers: Primary server and secondary server.

A *primary server* is responsible for creating, maintaining and updating the zone file. It stores this zone file on its local disk.

A *secondary server* neither creates nor updates the zone file but stores the latest zone file from the primary server. The idea is to have redundancy so that if the primary server fails, the secondary server can continue serving the clients. A server can be primary server for one zone and secondary server for another zone.

1.7.3 How does the DNS Server Works?

DNS is designed to be a client server application. A host that needs to map a name to an address is called a resolver.

The DNS works in a manner similar a telephone directory inquiry service. You dial up the inquiry service and ask for a person's telephone number by providing his name. If the person is local, the inquiry service immediately comes up with the answer. If s/he stays in another state, the directory service directs your call to that state's telephone directory service, or asks you to call them. This is very similar to the way. The DNS server works. In this case, you specify the domain name and ask for its corresponding address.

All day, a DNS server does two things:

- Accepting requests from programs for mapping domain names into IP addresses (revolvers).
- Accepting requests from other DNS servers to map domain names into IP addresses.

When such a request comes, a DNS server has the following options:

- It can supply the IP address because it knows it from its zone file.
- It can contact another DNS server and try to locate the IP address for the name requested. Every DNS server has an entry called alternate DNS server, which is the DNS server it should get in touch with for unresolved domains. If this server is in authority, it responds, otherwise sends the query to another server. When the query is finally resolved, it travels back until it finally reaches the resolver. This type of resolution is called recursive resolution which is shown in *Figure 21* given below.

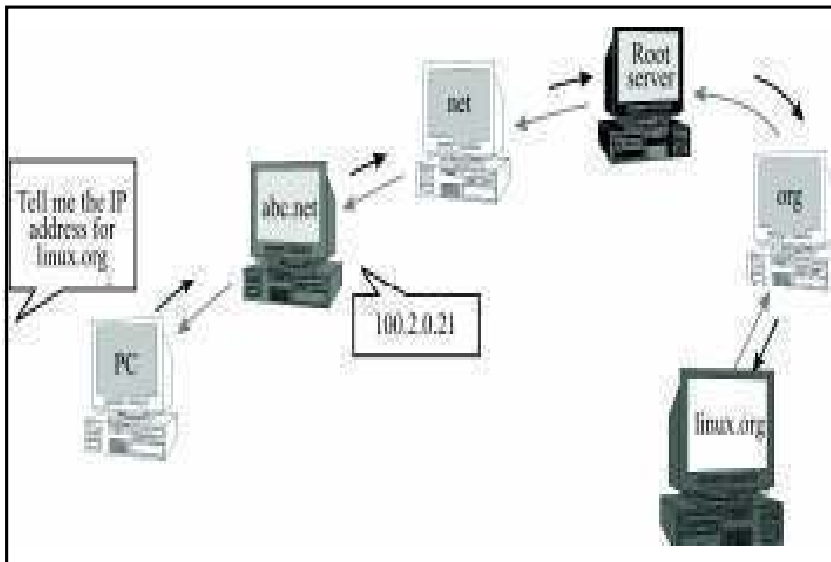


Figure 21: Recursive solution

It simply says "I do not know the IP address for the requested domain but here is the IP address for a name server which knows more than me". The client is responsible for repeating the query to this second server. If this newly addressed server can resolve the query, it answers the query, and otherwise, it returns address of the alternate server. This process is called iterative resolution which is shown in *Figure 22*.

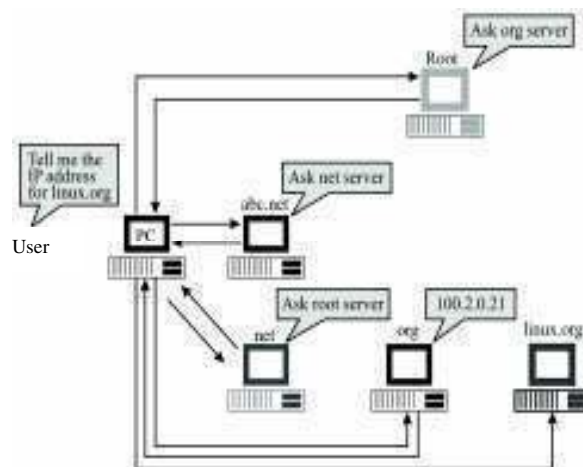


Figure 22: Iterative solution

It can return an error message because the requested domain name is invalid or does not exist.

Caching

Each time a server receives a query for the name that is not in its domain, it sends the query to the alternate server. It stores the information received from the response in its cache memory before sending it to the client. If any client asks for the same mapping, it can check its cache memory and resolve the problem. Thus, caching speeds up the resolution process to avoid sending outdated mapping to clients. The server adds an information called time-to live (TTL) to the cache entry. It defines this time in seconds that the receiving server can cache the information. After that the entry is invalid and any query must be sent to the alternate server. DNS requires that each server keeps a TTL counter for each mapping it caches.

1.7.4 Simple Network Management Protocol (SNMP)

Since it was developed in 1988, the Simple Network Management Protocol has become the de facto standard for internetwork management. Because it is a simple solution, requiring little code to implement, vendors can easily build SNMP agents to their products. SNMP is extensible, allowing vendors to easily add network management functions to their existing products. SNMP also separates the management architecture from the architecture of the hardware devices, which broadens the base of multivendor support. Perhaps most important, unlike other so-called standards, SNMP is not a mere paper specification, but an implementation that is widely available today.

SNMP is based on the manager/agent model consisting of a manager, an agent, a database of management information, managed objects and the network protocol. The manager provides the interface between the human network manager and the management system. The agent provides the interface between the manager and the physical device(s) being managed (see the illustration above).

A typical agent usually:

- Implements full SNMP protocol.
- Stores and retrieves management data as defined by the Management Information Base.
- Can asynchronously signal an event to the manager
- Can be a proxy for some non-SNMP manageable network node.

A typical manager usually:

- Implemented as a Network Management Station (the NMS)
- Implements full SNMP Protocol Able to
- Query agents
- Get responses from agents
- Set's variables in agents
- Acknowledge's asynchronous events from agents.

The manager and agent use a **Management Information Base (MIB)** and a relatively small set of commands to exchange information. The MIB is organised in a tree structure with individual variables, such as point status or description, being represented as leaves on the branches. A long numeric tag or **object identifier (OID)** is used to distinguish each variable uniquely in the MIB and in SNMP messages.

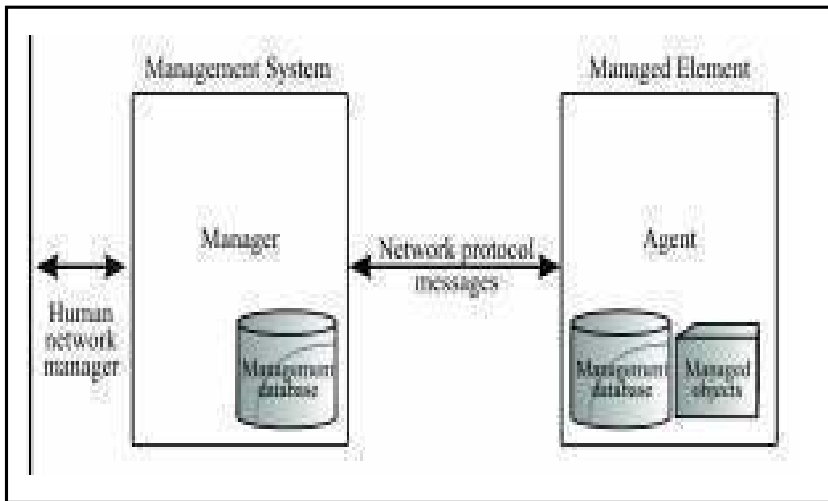


Figure 23: SNMP Protocol

SNMP uses five basic messages (GET, GET-NEXT, GET-RESPONSE, SET, and TRAP) to communicate between the manager and the agent. The GET and GET-NEXT messages allow the manager to request information for a specific variable. The agent, upon receiving a GET or GET-NEXT message, will issue a GET-RESPONSE message to the manager with either the information requested or an error indication as to why the request cannot be processed. A SET message allows the manager to request a change to be made to the value of a specific variable in the case of an alarm remote that will operate a relay. The agent will then respond with a GET-RESPONSE message indicating the change has been made or an error indication as to why the change cannot be made. The TRAP message allows the agent to spontaneously inform the manager of an 'important' event.

As you can see, most of the messages (GET, GET-NEXT, and SET) are only issued by the SNMP manager. Because the TRAP message is the only message capable of being initiated by an agent, it is the message used by DPS Remote Telemetry Units (RTUs) to report alarms. This notifies the SNMP manager as soon as an alarm condition occurs, instead of waiting for the SNMP manager to ask.

The small number of commands used is only one of the reasons SNMP is "simple." The other simplifying factor is its reliance on an unsupervised or connectionless communication link. This simplicity has led directly to its widespread use, specifically in the Internet Network Management Framework. Within this framework, it is considered 'robust' because of the independence of the managers from the agents, e.g., if an agent fails, the manager will continue to function, or *vice versa*. The unsupervised communication link does however create some interesting issues for network alarm monitoring.

Security levels with basic SNMP

Different Security levels (like authentication and authorization) are implemented in SNMP, let find out what we meant by authentication and authorization.

Authentication

Trivial authentication based on plain text community name is exchanged in SNMP messages. Authentication is based on the assumption that the message is not tampered with or interrogated.

Authorization

Once community name is validated then agent or manager checks to see if sending address is permitted or has the rights for the requested operation. "View" or "Cut" of

the objects together with permitted access rights is then derived for that pair (community name, sending address).

Underlying Communication Protocols

SNMP assumes that the communication path is a connectionless communication subnetwork. In other words, no prearranged communication path is established prior to the transmission of data. As a result, SNMP makes no guarantees about the reliable delivery of the data. Although in practice most messages get through, and those that don't can be retransmitted. The primary protocols that SNMP implements are the User Datagram Protocol (UDP) and the Internet Protocol (IP). SNMP also requires Data Link Layer protocols such as Ethernet or TokenRing to implement the communication channel from the management to the managed agent. The connectionless nature of SNMP leaves the recovery and error detection up to the NMS (Network Management Station) and even up to the agent. However, keep in mind that SNMP is actually transport independent (although original design was connectionless transport function, which corresponds to the UDP protocol) and can be implemented on other transports as well:

1.7.5 Remote Login: TELNET

The main task of the Internet and its TCP/IP protocols is to provide services for users. For example, a user may want to run an application at the remote site and create results that can be transferred to the local site. One way to satisfy this demand is to create client-server applications program for each such desired service. Though, file transfer programs (FTP), email (SMTP), etc. are available, but it would be impossible to write a client-server application for each demand. The better solution would be a general purpose client-server program that lets a user access any application program on a remote user, or allow a user to log on to a remote computer. TELNET (**TE**Rminal **NE**Twork) allows remote login services.

TELENT has two parts: client and server. Once a user using the services of a TELNET client connects to the remote TELNET server the keystrokes type by the user on the client are sent to the remote server to be interpreted upon to give the impression that the user is working on the remote computer.

Local login

When a user logs on to a local time sharing system, it is called local login. As user types at a terminal, the keystrokes are accepted by the terminal driver and passed to the operating system. The operating system interprets the combination of characters, and invokes the desired application. This mechanism is not simple as the operating system may associate special meanings to special characters. For example, Ctrl+z means suspend in UNIX. This does not create problem in local login because the terminal emulator and the driver know the exact meaning of characters.

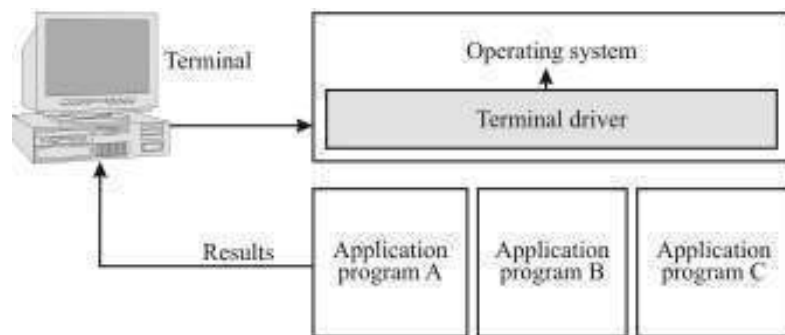


Figure 24: Local Login

When a user logs on to access an application on a remote computer, the users need to perform remote login.

Here the telnet client and server come into use. The user sends the keystrokes to the terminal driver where the local operating system accepts the characters but does not interpret them. The characters are sent to the TELNET client, which transform the characters to a universal character set called Network Virtual Terminal (NVT) and delivers them to the local TCP/IP stack. This is necessary because telnet is a general purpose application, and was designed to work between any terminal and any host. Thus, the client maps the terminal type to NVT. At the other end, server maps the NVT on to the actual terminal type the server is serving.

The commands or text, in NVT format travels through the Internet, and arrive at the TCP/IP stack of the remote machine. The characters are delivered to the local operating system, and are passed to the TELNET server, which performing mapping of NVT characters. However, they can be directly passed to the operating system, because it is designed to receive characters from terminal driver. The solution is to add a pseudo terminal driver, which pretends that the characters are coming from the terminal and not from the client. The operating system then passes the characters to the appropriate application.

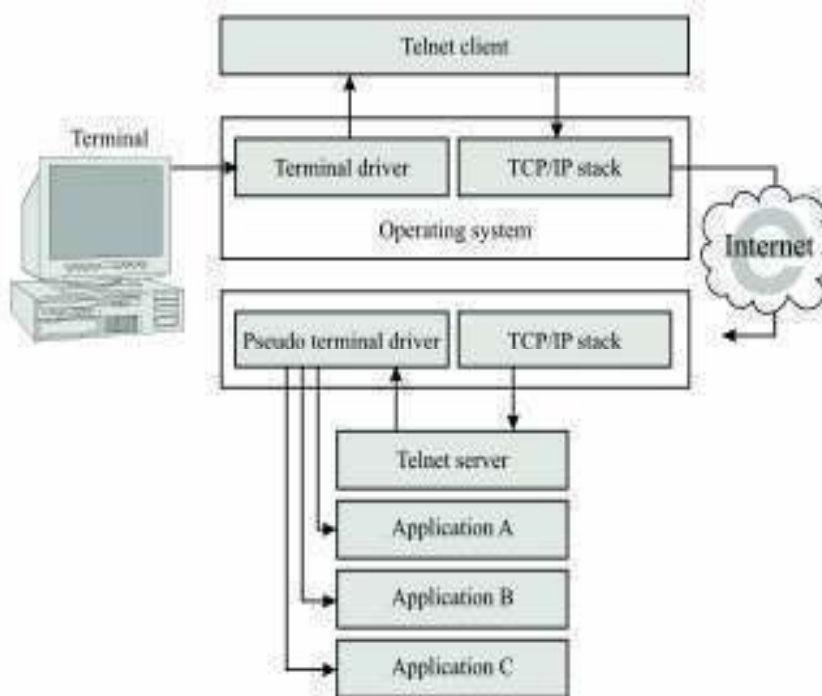


Figure 25: Remote Login

Communication in TELNET

Technically, TELNET server is quite complicated. It has to handle many clients at the same time, and respond in real time. To handle this issue, TELNET server uses the principle of delegation. Whenever there a new client request for a TELNET connection, it creates a child process and lets the child handle the particular client.

TCP uses only one TCP connection. The same TCP connection is used to transfer both the control and data characters. How does the TELNET then distinguish between control and data characters? For this, it mandates that each sequence of control characters must be preceded by a special control character called Interpret As Command (IAC).

1.7.6 World Wide Web: HTTP

The WWW project was initiated by CERN to create a system to handle distributed resources necessary for scientific research. Apart from email, the most popular applications running on Internet is the World Wide Web (WWW). It is so popular that people confuse it with Internet.

However, it is just an application such as email, FTP that uses TCP/IP. Many companies have internet websites, which is a collection of web pages stored on a web server. The server has web server software running on it. The function of web server is to store web pages and transmit them to a client computer when requested to do so.

In your company's website, you can display the information about the products, employees and policies. Sites can be dedicated to specialised tasks such as displaying news, share prices, sports etc. WWW consists of thousands of such websites of individuals and companies giving huge details about people, companies, events, news etc. WWW is an online repository of information that users can view using a program called web browser.

Architecture

The WWW today is a distributed client-server service. The client using a browser can access a service using a server. The service provider is distributed over many locations called sites.

Web server

A web server is a program running on a server computer, additionally, it consists of the web site, consisting of many web pages. It is simply a file written in HTML (Hypertext Markup Language). It can consist of text, graphics, sound, video, animation. Every web site has a server process that passively listens for TCP connection requests at port 80. After the connection is established, the client sends one request and the server sends one response. The request – response model is governed by Hyper Text Transfer Protocol (HTTP).

Web browser

A web browser acts as a client in WWW interaction. Using this program, user can request for a web page on a web server. The browser then interprets and displays the document. A variety of vendors offer commercial browsers. A browser usually consists of three parts: a controller, client protocol and interpreters. The controller receives the input from the keyboard or the mouse and uses client programs to access the document. After the document has been accessed, the controller uses one of the interpreters to display the document on screen. The client protocol can be one of the protocols described previously like FTP, TELNET, or HTTP.

Uniform Resource Locator (URL)

A client that wants to access a web page needs to specify the address. The Uniform Resource locator is the standard for specifying any kind of information on the Internet. It has four things: protocol, host computer, port, and path.

Anatomy of URL

<http://www.ignou.ac.in/80/index>

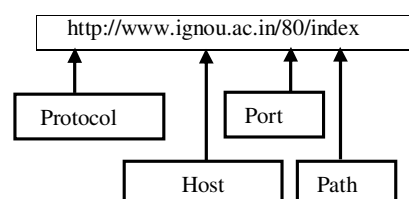


Figure 26: Components of URL

- *The protocol* is the client-server program used to retrieve the document. Most common is HTTP.
- *Host* is the name of the computer on which the information resides.
- Port number is transport address of the client or server program on a website.
- *Path* is the pathname of the file on the website, and can consists of slashes.

HTTP

Hyper Text Transfer Protocol (HTTP) is a used mainly to transfer data on World Wide Web. The commands from the client are embedded in a request message .The contents of the request message are embedded in a response message. HTTP uses the services of TCP at port 80.

HTTP is a stateless protocol since each transaction is independent of the previous transaction. The TCP connection between the client and the server is established for every page. It does not remember anything about the previous request. Keeping HTTP stateless was aimed at making the Web simple.

Sample HTTP request and response transaction is shown in figure 27.

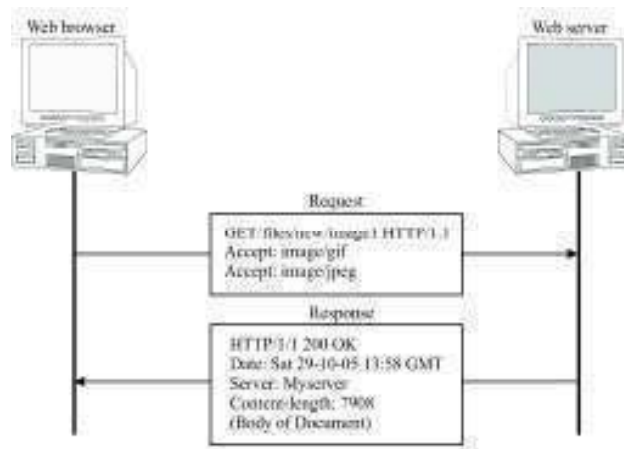


Figure 27: HTTP request and response

The GET command requests the web server at www.myserver.com for a image file image1. The HTTP/1.0 indicates that the browser uses the 1.0 version of the HTTP protocol.

The first line of response indicates that the server is also using HTTP version 1.0. The return code of 200 indicates that the server processed the request successfully. Request message can be of following types.

Method	Action
GET	Requests a document from the server
HEAD	Request information about the document but not the document itself; i.e. head of the HTML page
POST	Sends some information from client to server. It appends the data to the existing document.
PUT	Sends a document from to server. It replaces the existing document.
TRACE	Echoes the incoming request
CONNECT	Reserved
OPTION	Enquire about available options

Locating Information on the Internet

The amount of information available on the Internet is mind-boggling. However, finding the right information is usually very cumbersome. For enabling people to search information efficiently, search engines are used.

The search engines may continuously crawl through the Web Pages on the Internet and gather information about them to facilitate search for users in those Web pages.



Check Your Progress 3

- 1) Explain the concept of recursive and iterative resolution in of DNS?

.....

.....

.....

.....

.....

- 2) How does FTP differ from other application layer protocols?

.....

.....

.....

.....

.....

1.8 NETWORKING EXAMPLE: PUTTING IT ALTOGETHER

After seeing how communication takes place using computers, we describe four scenarios to give a clear picture of it

Example 1: In the first example, there are just two computers, which are not part of any network as shown in *Figure 28*.

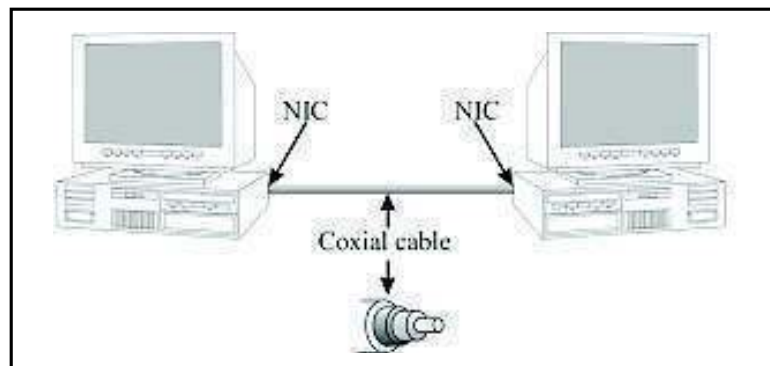
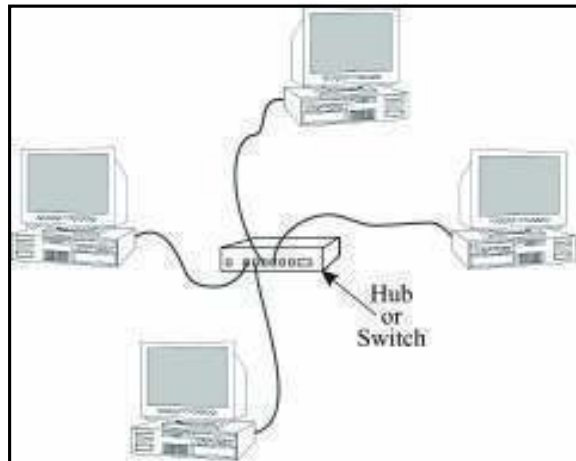


Figure 28: Two computers connected with Coaxer Cable

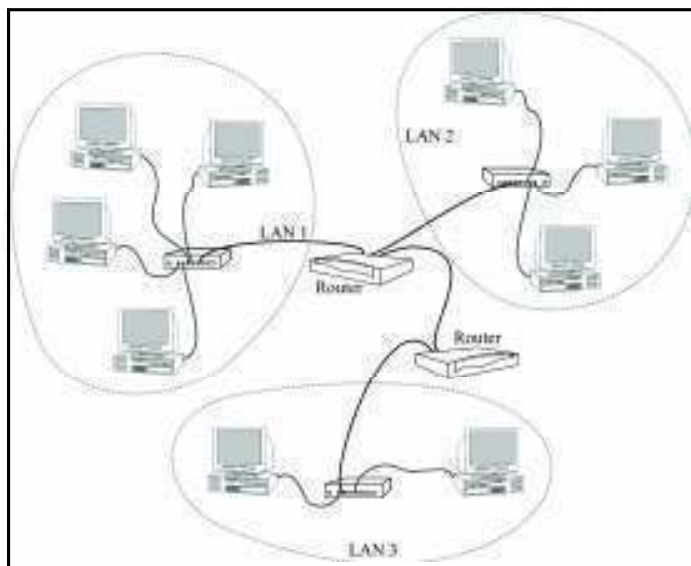
In order to have communication, both computers should have Network Interface card and TCP/IP software running over it. They can connect using coaxial cable, and can choose any IP address, as they are not part of any network. The link is very fast as there is no sharing of bandwidth.

Example 2:**Figure 29: Computers connected with Hubs**

A number of computers are connected to each other using a hub or a switch device as shown in *Figure 29*. Now the computers require uniform addressing, a network interface card and TCP/IP software they can connect to switch or hub using any physical medium. At a time only two computers can communicate. The packets from a computer are first transferred to the switch, which sends it to the destined computer.

Example 3:

A number of small networks like the ones in a computer Lab connect to each other, using routers as shown in *Figure 30*. A computer wanting to communicate to another computer first checks if the destination computer is on the same network. If so, it uses ARP to know the IP address of the destination and forwards the packet to it. However, if it is not on the same network, the packet is forwarded to the router connected to the network. A router is a multihomed device, which is a part of more than one network. If the destination computer is connected to the router, it is delivered, otherwise it is forwarded to another router which is expected to know the way to destination. The path followed in order to route the packets is determined by the routers which employ criteria like shortest path, minimum delay, minimum cost to decide the next router to which the packet should be forwarded. The links and thus the bandwidth is shared among the computers.

**Figure 30: LANS connected with Routers**

Example 4:

The fourth example is of the Internet given below in *Figure 31*, where a number of networks are connected to each other. These networks differ from each other in a number of ways like packet format, underlying hardware, and the protocols used. The scenario differs from the third one in that a gateway is required. A gateway can forward packets across different networks that may also use different protocols. Thus a gateway not only has the ability of translating between different packets formats, but also different protocols. Communication in the Internet is made possible by use of routers which forward packets between networks using the same protocol, and gateways connect large, incompatible networks.

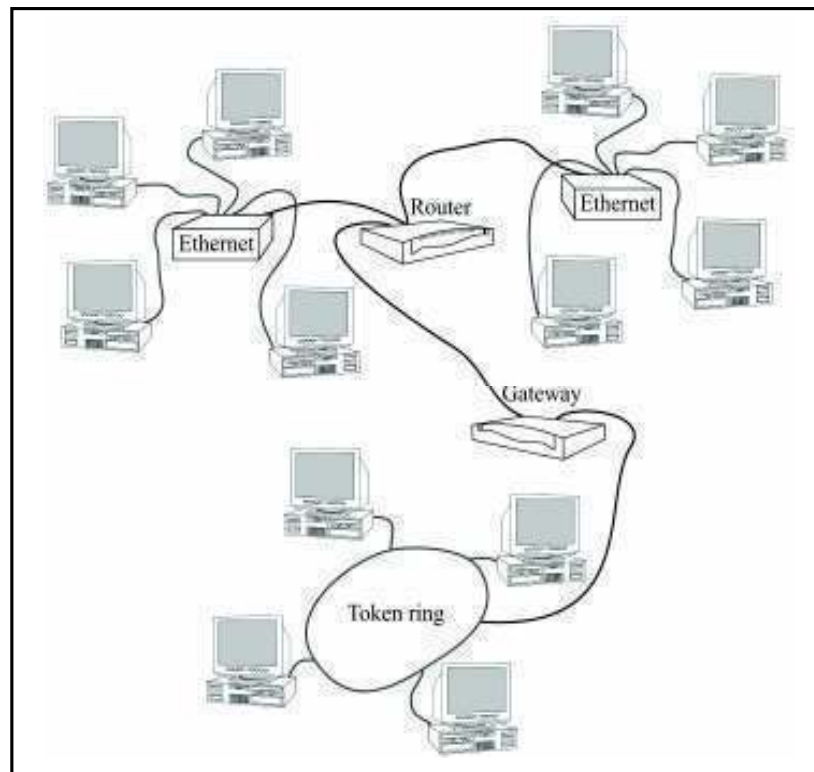


Figure 31: Different network connected with gateway and routers

1.9 SUMMARY

This unit is complete overview of TCP/IP protocols. Till now we have studied that the Internet and TCP/IP came into being at around the same time. Some standards need to be flowed at the hardware and software level in order to connect incompatible networks. TCP is the software standard for the Internet. It is organised as a stack of layers. Each layer performs has interface with adjacent layers and certain functionalities. IP is connectionless, unreliable, and best effort packet delivery mechanism. It provides node-to-node communication ARP, RARP, ICMP are other Internet layer protocols. TCP is reliable, connection-oriented protocol, and uses the services of IP. UDP is another transport layer protocol, which is unreliable, connectionless. It is much simpler than TCP and used in applications, which do not require the reliability but needs faster delivery. Together they provide end-to-end delivery of packets. Application layer protocols provide users with mechanisms to use internet for file transfer, (FTP) connection to remote hosts (TELNET), viewing web pages (WWW), manage network resources (SNMP) and lot of other functionalities. In the end, scenarios for typical communication between two computers are discussed. The next unit of this course covers about the Internet Protocol and its role in TCP/IP.

Check Your Progress 1

- 1) The TCP/IP model is made up of four layers: interface layer, network, transport, and application. The first layer of TCP/IP (Application layer) is similar to the first three layers (Application, Presentation and Session layer) of the OSI model. The services of transport layers of both the models are similar. Further, the services of network layers in both models are also similar, while some time network layer is also known as Internet layer. The last layer of TCP/IP is Interface layer, which includes the services of data link layer and physical layer of OSI model. In OSI model, each layer takes the services of the lower layer. Whereas the layers of TCP/IP protocol suite contain relatively independent protocols.
- 2) The TCP/IP model is made up of four layers: interface, network, transport, and application. Layers of TCP/IP Protocol Suite are explained below in detail:

i) **Interface Layers**

The physical layer deals with the hardware level, voltages. The data link layer deals with media access and control strategies, frame format etc. At this level, TCP/IP does not define any protocol. It supports all standards and protocols.

ii) **Internet Layer or Network Layer**

The Internet layer is an important layer in the protocol suite. At this layer, TCP/IP supports Internetworking Protocol (IP). IP is a host-to host protocol. This layer is responsible for the format of data-grams as defined by IP, and routing a datagram or packet to the next hop, but is not responsible for the accurate and timely delivery of datagrams to the destination in proper sequence. IP allows raw transmission functions allowing user to add functionalities necessary for given application. An ensuring maximum efficiency, TCP/IP supports four other protocols: ARP, RARP, ICMP, IGMP.

iii) **Transport Layer**

At this layer, TCP/IP supports two protocols: TCP, UDP, IP is host-to-host protocol, which can deliver the packet from one physical device to another physical device. TCP, are UDP, are transport level protocols, responsible for delivering a packet from one process on a device to another process on the other device. User Datagram Protocol (UDP)

It is simpler of the two protocols. It does not provide reliability. It is, therefore faster, and using for applications in which delay is intolerable (in case of audio and video)

Transmission Control Protocol (TCP).

TCP is reliable, connection oriented protocol. By connection oriented, we mean that a connection must be established between both ends before either can transmit data. It ensures that communication is error-free and in sequence.

iv) **Application Layer**

As said earlier, it is closer to combined session, presentation, and application layer of OSI model. It allows user to run various applications on Internet. These applications are File Transfer Protocol (FTP), remote login (TELNET), email (SMTP), WWW (HTTP). The session layer of OSI model is almost dropped in TCP/IP.

Check Your Progress 2

- 1) Following are functions performed at the Internet layer:
 - Define the datagram, which is the basic unit of transmission in the Internet.
 - Define the Internet addressing scheme.
 - Move data between the Network Access Layer and the Host-to-Host Transport Layer.
 - Route datagrams to remote hosts.
 - Fragment and reassemble datagrams.
- 2) When datagrams arrive too quickly for processing, the destination host or an intermediate gateway sends an ICMP *source quench message* back to the sender. This message instructs the source to stop sending datagrams temporarily.

Check Your Progress 3

- 1) DNS server does two things one is accepting requests from programs for mapping domain names into IP addresses (revolvers) and another is accepting requests from other DNS servers to map domain names into IP addresses. When such a request comes, a DNS server has the following options.
 - i) It can supply the IP address because it knows it from its zone file.
 - ii) It can contact another DNS server and try to locate the IP address for the name requested. Every DNS server has an entry called alternate DNS server, which is the DNS server it should get in touch with for unresolved domains. If this server is in authority, it responds, otherwise sends the query to another server. When the query is finally resolved, it travels back until it finally reaches the resolver. This type of resolution is called ***recursive resolution***.
- 2) It simply says, I do not know the IP address for the requested domain but here is the IP address for a name server which knows more than me". The client is responsible for repeating the query to this second server. If this newly addressed server can resolve the query, it answers the query, and otherwise, it returns address of the alternate server. This process is called ***iterative resolution***.
- 3) It can return an error message because the requested domain name is invalid or does not exist.

1.11 FURTHER READINGS

- 1) Achyut S Godbole, *Web Technologies*, TATA McGrawHill, 2003.
- 2) Berhouz Forouzan, *TCP/IP Protocol Suite*, 3rd edition, TATA McGraw Hill, 2006.
- 3) <http://www.tcpipguide.com>
- 4) <http://www.cisco.com>

UNIT 2 INTERNET PROTOCOL

Structure	Page Nos.
2.0 Introduction	45
2.1 Objectives	45
2.2 Overview of Internet Protocol	46
2.3 IP Header	46
2.4 IP Address	50
2.4.1 IP Address Classes	50
2.4.2 Subnet Masks and CIDR Networks (Classless IP Addresses)	52
2.4.3 Internet-Legal Versus Private Addressing	55
2.5 IP Routing	57
2.5.1 Routing Protocol	57
2.5.2 Routing Algorithms	58
2.6 Summary	60
2.7 Answers to Check Your Progress	60
2.8 Further Readings	60

2.0 INTRODUCTION

Internet protocols were first developed in the mid-1970s, when the Defence Advanced Research Projects Agency (DARPA) became interested in establishing a packet-switched network that would facilitate communication between dissimilar computer systems at research institutions. The Internet Protocol (IP) is a network-layer (Layer 3) protocol that contains addressing information and some control information that enables packets to be routed. IP is documented in RFC 791 and is the primary network-layer protocol in the TCP/IP protocol suite. Along with the Transmission Control Protocol (TCP), IP represents the heart of the Internet protocols. IP has two primary responsibilities: providing connectionless, best-effort delivery of datagrams through an internetwork; and providing fragmentation and reassembly of datagrams to support data links with different maximum-transmission unit (MTU) sizes.

2.1 OBJECTIVES

Our objective is to introduce you with basic concept of Internet Protocol. On successful completion of this unit, you should be able to:

- have a reasonable understanding of the IP protocol architecture;
- describe the operation of IP protocol and its header format;
- understand the role and meaning of IP addressing and classes;
- describe and understand how to use subnet addressing; and
- understand the simple routing protocols.

2.2 OVERVIEW OF INTERNET PROTOCOL

Internet Protocol is the network (internet) layer protocol used by both TCP and UDP, transport layer protocols. The entire TCP or UDP datagrams (header + payload) travel through the network as a part of the IP datagrams. TCP or UDP datagram is encapsulated in IP datagram as illustrated in following *Figure 1*. It is also clear from the figure that the application layer packet is encapsulated in transport layer datagram and network layer datagram is encapsulated in link layer frame.

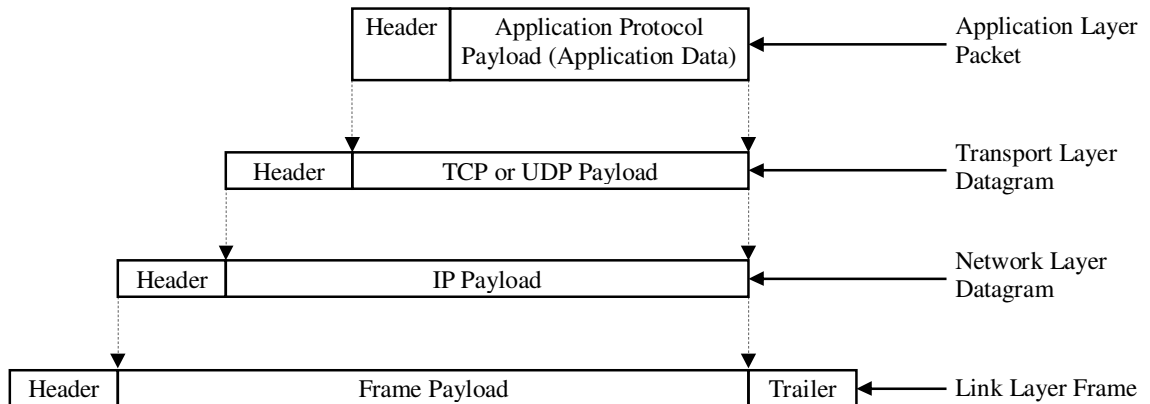


Figure 1: TCP/IP Protocols and Encapsulation

2.3 IP HEADER

The IP datagram consists of a header part and a data part. The IP header is six 32-bit words in length (24 bytes total) when all the optional fields are included in the header. The shortest header allowed by IP uses five words (20 bytes total). To understand all the fields in the header, it is useful to remember that IP has no hardware dependence but must account for all versions of IP software it can encounter (providing full backward-compatibility with previous versions of IP). The IP header layout is shown schematically in *Figure 2*. The different fields in the IP header are examined in more detail in the following subsections.

Version	HL	TOS	Total Length	
Identifier			Flags	Fragment Offset
TTL	Protocol		Header Checksum	
Source Address				
Destination Address				
Options				

Figure 2: IPv4 Header Format

Version Number

This is a 4-bit field that contains the IP version number the protocol software is using. The version number is required so that receiving IP software knows how to decode the rest of the header, which changes with each new release of the IP standards. The most widely used version is 4, although several systems are now testing version 6 (called IPng). The Internet and most LANs do not support IP version 6 at present.

Part of the protocol definition stipulates that the receiving software must first check the version number of incoming datagrams before proceeding to analyse the rest of the header and encapsulated data. If the software cannot handle the version used to build the datagram, the receiving machine's IP layer rejects the datagram and ignores the contents completely.

Header Length

This 4-bit field reflects the total length of the IP header built by the sending machine; it is specified in 32-bit words. The shortest header is five words (20 bytes), but the use of optional fields can increase the header size to its maximum of six words (24 bytes). To properly decode the header, IP must know when the header ends and the data begins, which is why this field is included. (There is no start-of-data marker to show where the data in the datagram begins. Instead, the header length is used to compute an offset from the start of the IP header to give the start of the data block).

Type of Service

The 8-bit (1 octet) Service Type field instructs IP how to process the datagram properly. The field's 8 bits are read and assigned as shown in *Figure 3*, which shows the layout of the Service Type field inside the larger IP header shown in *Figure 2*. The first 3 bits indicate the datagram's precedence, with a value from 0 (normal) through 7 (network control). The higher the number, the more important the datagram and, in theory at least, the faster the datagram should be routed to its destination. In practice, though, most implementations of TCP/IP and practically all hardware that use TCP/IP ignore this field, treating all datagrams with the same priority.

Precedence (3 bits)	Delay	Thru	Rel	Not Used
---------------------	-------	------	-----	----------

Figure 3: Type of Service

The next three bits are 1-bit flags that control the delay, throughout, and reliability of the datagram. If the bit is set to 0, the setting is normal. A bit set to 1 implies low delay, high throughput, and high reliability for the respective flags. The last two bits of the field are not used. Most of these bits are ignored by current IP implementations, and all datagrams are treated with the same delay, throughput, and reliability settings.

For most purposes, the values of all the bits in the Service Type field are set to 0 because differences in precedence, delay, throughput, and reliability between machines are virtually nonexistent unless a special network has been established. Although these flags would be useful in establishing the best routing method for a datagram, no currently available UNIX-based IP system bothers to evaluate the bits in these fields. (Although it is conceivable that the code could be modified for high security or high reliability networks.)

Total Length (or Packet Datagram Length)

This field gives the total length of the datagram, including the header, in octets. The length of the data area itself can be computed by subtracting the header length from this value. The size of the total datagram length field is 16 bits, hence the 65,535 bytes are the maximum length of a datagram (including the header). This field is used to determine the length value to be passed to the transport protocol to set the total packet length.

Identification

This field holds a number that is a unique identifier created by the sending node. This number is required when reassembling fragmented messages, ensuring that the fragments of one message are not intermixed with others. Each chunk of data received by the IP layer from a higher protocol layer is assigned one of these identification numbers when the data arrives. If a datagram is fragmented, each fragment has the same identification number.

Flags

The Flags field is a 3-bit field, the first bit of which is left unused (it is ignored by the protocol and usually has no value written to it). The remaining two bits are dedicated to flags called DF (Don't Fragment) and MF (More Fragments), which control the handling of the datagrams when fragmentation is desirable.

If the DF flag is set to 1, the datagram cannot be fragmented under any circumstances. If the current IP layer software cannot send the datagram to another machine without fragmenting it, and this bit is set to 1, the datagram is discarded and an error message is sent back to the sending device.

If the MF flag is set to 1, the current datagram is followed by more fragments (sometimes called *subpackets*), which must be reassembled to re-create the full message. The last fragment that is sent as part of a larger message has its MF flag set to 0 (off) so that the receiving device knows when to stop waiting for datagrams. Because the order of the fragments' arrival might not correspond to the order in which they were sent, the MF flag is used in conjunction with the Fragment Offset field (the next field in the IP header) to indicate to the receiving machine the full extent of the message.

Fragment Offset

If the MF (More Fragments) flag bit is set to 1 (indicating fragmentation of a larger datagram), the fragment offset contains the position of the fragments contained in the complete message within the current datagram. This enables IP to reassemble fragmented packets in the proper order.

Offsets are always given relative to the beginning of the message. This is a 13-bit field, so offsets are calculated in units of 8 bytes, corresponding to the maximum packet length of 65,535 bytes. Using the identification number to indicate which message a receiving datagram belongs to, the IP layer on a receiving machine can then use the fragment offset to reassemble the entire message.

Time to Live (TTL)

This field gives the amount of time in maximum number of **hops** that a datagram can remain on the network before it is discarded. This is set by the sending node when the datagram is assembled. Usually the TTL field is set to 15 or 30 hops.

The TCP/IP standards stipulate that the TTL field must be decreased by at least one hop for each node that processes the packet, even if the processing time is less. Also, when a datagram is received by a gateway, the arrival time is tagged so that if the datagram must wait to be processed, that time counts against its TTL. Hence, if a gateway is particularly overloaded and can't get to the datagram in short order, the TTL timer can expire while awaiting processing, and the datagram is abandoned.

If the TTL field reaches 0, the datagram must be discarded by the current node, but a message is sent back to the sending machine when the packet is dropped. The sending machine can then resend the datagram. The rules governing the TTL field are designed to prevent IP packets from endlessly circulating through networks.

This field holds the identification number of the protocol to which the packet is to be handed. The numbers are defined by the Network Information Centre (NIC), which governs the Internet. There are currently about 50 protocols defined and assigned a transport protocol number. The two most important protocols are ICMP (detailed in the section titled “Internet Control Message Protocol (ICMP)” later today), which is number 1, and TCP, which is number 6. The full list of numbers is not necessary here because most of the protocols are never encountered by users.

Header Checksum

The number in this field of the IP header is a checksum for the protocol header field (but not the data fields) to enable faster processing. Because the Time to Live (TTL) field is decremented at each node, the checksum also changes with every machine the datagram passes through. The checksum algorithm takes the ones-complement of the 16-bit sum of all 16-bit words. This is a fast, efficient algorithm, but it misses some unusual corruption circumstances such as the loss of an entire 16-bit word that contains only 0s. However, because the data checksums used by both TCP and UDP cover the entire packet, these types of errors usually can be caught as the packet is assembled for the network transport.

Sending Address and Destination Address

These fields contain the 32-bit IP addresses of the sending and destination devices. These fields are established when the datagram is created and are not altered during the routing.

Options

The Options field is optional, composed of several codes of variable length. If more than one option is used in the datagram, the options appear consecutively in the IP header. All the options are controlled by a byte that is usually divided into three fields: a 1-bit copy flag, a 2-bit option class, and a 5-bit option number. The copy flag is used to stipulate how the option is handled when fragmentation is necessary in a gateway. When the bit is set to 0, the option should be copied to the first datagram but not subsequent ones. If the bit is set to 1, the option is copied to all the datagrams.

The option class and option number indicate the type of option and its particular value. At present, there are only two option classes set. (With only 2 bits to work with in the field, a maximum of four options could be set.) When the value is 0, the option applies to datagram or network control. A value of 2 means the option is for debugging or administration purposes. Values of 1 and 3 are unused. Currently supported values for the option class and number are given in *Table 1*.

Table 1: Valid option class and numbers for IP headers

Option Class	Option Number	Description
0	0	Marks the end of the options list
0	1	No option (used for padding)
0	2	Security options (military purposes only)
0	3	Loose source routing
0	7	Activates routing record (adds fields)
0	9	Strict source routing
2	4	Timestamping active (adds fields)

Of most interest to you are options that enable the routing and timestamps to be recorded. These are used to provide a record of a datagram's passage across the internetwork, which can be useful for diagnostic purposes. Both these options add information to a list contained within the datagram. (The timestamp has an interesting format: it is expressed in milliseconds since midnight, Universal Time. Unfortunately, because most systems have widely differing time settings — even when corrected to Universal Time — the timestamps should be treated with more than a little suspicion).

There are two kinds of routing indicated within the options field: loose and strict. *Loose routing* provides a series of IP addresses that the machine must pass through, but it enables any route to be used to get to each of these addresses (usually gateways). *Strict routing* enables no deviations from the specified route. If the route can't be followed, the datagram is abandoned. Strict routing is frequently used for testing routes but rarely for transmission of user datagrams because of the higher chances of the datagram being lost or abandoned.

Padding

The content of the padding area depends on the options selected. The padding is usually used to ensure that the datagram header is a round number of bytes.

2.4 IP ADDRESSES

The Internet Protocol moves data between hosts in the form of datagrams. Each datagram is delivered to the address contained in the Destination Address (word 5) of the datagram's header. The Destination Address is a standard 32-bit IP address that contains sufficient information to uniquely identify a network and a specific host on that network.

An IP address contains a network part and a host part, but the format of these parts is not the same in every IP address. The number of address bits used to identify the network, and the number used to identify the host, vary according to the prefix length of the address. There are two ways the prefix length is determined: by address class or by a CIDR address mask. We begin with a discussion of traditional IP address classes.

2.4.1 IP Address Classes

Originally, the IP address space was divided into a few fixed-length structures called address classes. The three main address classes are class A, class B, and class C. By examining the first few bits of an address, IP software can quickly determine the class, and therefore the structure, of an address. IP follows these rules to determine the address class:

- **Class A:** If the first bit of an IP address is 0, it is the address of a class A network. The first bit of a class A address identifies the address class. The next 7 bits identify the network, and the last 24 bits identify the host. There are fewer than 128 class A network numbers, but each class A network can be composed of millions of hosts.
- **Class B:** If the first 2 bits of the address are 1 0, it is a class B network address. The first 2 bits identify class; the next 14 bits identify the network, and the last 16 bits identify the host. There are thousands of class B network numbers and each class B network can contain thousands of hosts.

- **Class C:** If the first 3 bits of the address are 1 1 0, it is a class C network address. In a class C address, the first 3 bits are class identifiers; the next 21 bits are the network address, and the last 8 bits identify the host. There are millions of class C network numbers, but each class C network is composed of fewer than 254 hosts.
- **Class D:** If the first 4 bits of the address are 1 1 1 0, it is a multicast address. These addresses are sometimes called class D addresses, but they don't really refer to specific networks. Multicast addresses are used to address groups of computers all at one time. Multicast addresses identify a group of computers that share a common application, such as a video conference, as opposed to a group of computers that share a common network.
- **Class E:** If the first four bits of the address are 1 1 1 1, it is a special reserved address. These addresses are called class E addresses, but they don't really refer to specific networks. No numbers are currently assigned in this range.

IP addresses are usually written as four decimal numbers separated by dots (periods). Each of the four numbers is in the range 0-255 (the decimal values possible for a single byte). Because the bits that identify class are contiguous with the network bits of the address, we can lump them together and look at the address as composed of full bytes of network address and full bytes of host address. If the value of the first byte is:

- Less than 128, the address is class A; the first byte is the network number, and the next three bytes are the host address.
- From 128 to 191, the address is class B; the first two bytes identify the network, and the last two bytes identify the host.
- From 192 to 223, the address is class C; the first three bytes are the network address, and the last byte is the host number.
- From 224 to 239, the address is multicast. There is no network part. The entire address identifies a specific multicast group.
- Greater than 239, the address is reserved.

The Table 2 depicts each class range with other details.

Table 2: IP Address Classes in dotted decimal format with their ranges

IP Address Class	High Order Bit(s)	Format	Range	No. of Network Bits	No. of Host Bits	Max. Hosts	Purpose
A	0	N.H.H. H	1.0.0.0 to 126.0.0.0	7	24	$2^{24}-2$	Few large organisations
B	1,0	N.N.H. H	128.1.0.0 to 191.254.0.0	14	16	$2^{16}-2$	Medium-size organisations
C	1,1,0	N.N.N. H	192.0.1.0 to 223.255.254.0	21	8	2^8-2	Relatively small organisations
D	1,1,1,0	N/A	224.0.0.0 to 239.255.255.255	N/A	N/A	N/A	Multicast groups (RFC 1112)
E	1,1,1,1	N/A	240.0.0.0 to 254.255.255.255	N/A	N/A	N/A	Future Use (Experimental)

The IP address, which provides universal addressing across all of the networks of the Internet, is one of the great strengths of the TCP/IP protocol suite. However, the original class structure of the IP address has weaknesses. The TCP/IP designers did not envision the enormous scale of today's network. When TCP/IP was being

designed, networking was limited to large organisations that could afford substantial computer systems. The idea of a powerful UNIX system on every desktop did not exist. At that time, a 32-bit address seemed so large that it was divided into classes to reduce the processing load on routers, even though dividing the address into classes sharply reduced the number of host addresses actually available for use. For example, assigning a large network a single class B address, instead of six class C addresses, reduced the load on the router because the router needed to keep only one route for that entire organisation. However, an organisation that was given the class B address probably did not have 64,000 computers, so most of the host addresses available to the organisation were never assigned.

The class-structured address design was critically strained by the rapid growth of the Internet. At one point it appeared that all class B addresses might be rapidly exhausted. To prevent this, a new way of looking at IP addresses without a class structure was developed.



Check Your Progress 1

- 1) Internet Protocol is a _____ layer protocol.
 - a) Transport
 - b) Application
 - c) Network
 - d) Data Link

.....
- 2) What is the size of “type of service” field?
 - a) 4 bit
 - b) 5 bit
 - c) 7 bit
 - d) 8 bit

.....
- 3) If DF flag is set to 1, it means the datagram:
 - a) Large and need fragmentation
 - b) Small & need reassembling
 - c) Cannot be fragmented
 - d) Is accepted and message is sent back to sending device

.....

2.4.2 Subnet Masks and CIDR Networks (Classless IP Addresses)

IP addresses are actually 32-bit binary numbers. Each 32-bit IP address consists of two subaddresses, one identifying the network and the other identifying the host to the network, with an imaginary boundary separating the two. The location of the boundary between the network and host portions of an IP address is determined through the use of a subnet mask. A subnet mask is another 32-bit binary number, which acts like a filter when it is applied to the 32-bit IP address. By comparing a subnet mask with an IP address, systems can determine which portion of the IP address relates to the network, and which portion relates to the host. Anywhere the subnet mask has a bit set to "1", the underlying bit in the IP address is part of the network address. Anywhere the subnet mask is set to "0", the related bit in the IP

address is part of the host address. For example, assume that the IP address 1100000010101000000000100010100 has a subnet mask of 1111111111111111111100000000. In this example, the first 24 bits of the 32-bit IP addresses are used to identify the network, while the last 8 bits are used to identify the host on that network.

The size of a network (i.e., the number of host addresses available for use on it) is a function of the number of bits used to identify the host portion of the address. If a subnet mask shows that 8 bits are used for the host portion of the address block, a maximum of 256 possible host addresses are available for that specific network. Similarly, if a subnet mask shows that 16 bits are used for the host portion of the address block, a maximum of 65,536 possible host addresses are available for use on that network.

If a network administrator needs to split a single network into multiple virtual networks, the bit-pattern in use with the subnet mask can be changed to allow as many networks as necessary. For example, assume that we want to split the 24-bit 192.168.10.0 network (which allows for 8 bits of host addressing, or a maximum of 256 host addresses) into two smaller networks. All we have to do in this situation is to change the subnet mask of the devices on the network so that they use 25 bits for the network instead of 24 bits, resulting in two distinct networks with 128 possible host addresses on each network. In this case, the first network would have a range of network addresses between 192.168.10.0 -192.168.10.127, while the second network would have a range of addresses between 192.168.10.128 -192.168.10.255.

Networks can also be enlarged through the use of a technique known as “**supernetting**,” which works by extending the host portion of a subnet mask to the left, into the network portion of the address. Using this technique, a pair of networks with 24-bit subnet masks can be turned into a single large network with a 23-bit subnet mask. However, this works only if you have two neighbouring 24-bit network blocks, with the lower network having an even value (when the network portion of the address is shrunk, the trailing bit from the original network portion of the subnet mask will fall into the host portion of the new subnet mask, so the new network mask will consume both networks). For example, it is possible to combine the 24-bit 192.168.10.0 and 192.168.11.0 networks together since the loss of the trailing bit from each network (00001010 vs. 00001011) produces the same 23-bit subnet mask (0000101x), resulting in a consolidated 192.168.10.0 network. However, it is not possible to combine the 24-bit 192.168.11.0 and 192.168.12.0 networks, since the binary values in the seventh bit position (00001011 vs. 00001100) do not match when the trailing bit is removed.

Classless Inter-Domain Routing

In the modern networking environment defined by RFC 1519 [Classless Inter-Domain Routing (CIDR)], the subnet mask of a network is typically annotated in written form as a “slash prefix” that trails the network number. In the subnetting example in the previous paragraph, the original 24-bit network would be written as 192.168.10.0/24, while the two new networks would be written as 192.168.10.0/25 and 192.168.10.128/25. Likewise, when the 192.168.10.0/24 and 192.168.11.0/24 networks were joined together as a single supernet, the resulting network would be written as 192.168.10.0/23. Note that the slash prefix annotation is generally used for human benefit; infrastructure devices still use the 32-bit binary subnet mask internally to identify networks and their routes. All networks must reserve host addresses that are made up entirely of either ones or zeros, to be used by the networks themselves. This is so that each subnet will have a network-specific address (the all-zeroes address) and a broadcast address (the all-ones address). For example, a /24 network allows for 8 bits of host addresses, but only 254 of the 256 possible addresses are available for use. Similarly, /25 networks have a maximum of 7 bits for host addresses, with 126 of the 128 possible addresses available (the all-ones and all-

zeroes addresses from each subnet must be set aside for the subnets themselves). All the systems on the same subnet must use the same subnet mask in order to communicate with each other directly. If they use different subnet masks they will think they are on different networks, and will not be able to communicate with each other without going through a router first. Hosts on different networks can use different subnet masks, although the routers will have to be aware of the subnet masks in use on each of the segments.

Subnet masks are used only by systems that need to communicate with the network directly. For example, external systems do not need to be aware of the subnet masks in use on your internal networks, since those systems will route data to your networks by way of your parent network's address block. As such, remote routers need to know only about your provider's subnet mask. For example, if you have a small network that uses only a /28 prefix that is a subset of your ISP's /20 network, remote routers need to know only about your upstream provider's /20 network, while your upstream provider needs to know your subnet mask in order to get the data to your local /28 network. The rapid depletion of the class B addresses showed that three primary address classes were not enough: class A was much too large and class C was much too small. Even a class B address was too large for many networks but was used because it was better than the alternatives.

The obvious solution to the class B address crisis was to force organisations to use multiple class C addresses. There were millions of these addresses available and they were in no immediate danger of depletion. As is often the case, the obvious solution is not as simple as it may seem. Each class C address requires its own entry within the routing table. Assigning thousands or millions of class C addresses would cause the routing table to grow so rapidly that the routers would soon be overwhelmed. The solution requires a new way of assigning addresses and a new way of looking at addresses.

Originally network addresses were assigned in more or less sequential order as they were requested. This worked fine when the network was small and centralised. However, it did not take network topology into account. Thus only random chance would determine if the same intermediate routers would be used to reach network 195.4.12.0 and network 195.4.13.0, which makes it difficult to reduce the size of the routing table. Addresses can only be aggregated if they are contiguous numbers and are reachable through the same route. For example, if addresses are contiguous for one service provider, a single route can be created for that aggregation because that service provider will have a limited number of routes to the Internet. But if one network address is in France and the next contiguous address is in Australia, creating a consolidated route for these addresses does not work.

Today, large, contiguous blocks of addresses are assigned to large network service providers in a manner that better reflects the topology of the network. The service providers then allocate chunks of these address blocks to the organisations to which they provide network services. This alleviates the short-term shortage of class B addresses and, because the assignment of addressees reflects the topology of the network, it permits route aggregation. Under this new scheme, we know that network 195.4.12.0 and network 195.4.13.0 are reachable through the same intermediate routers. In fact, both of these addresses are in the range of the addresses assigned to Europe, 194.0.0.0 to 195.255.255.255. Assigning addresses that reflect the topology of the network enables route aggregation, but does not implement it. As long as network 195.4.12.0 and network 195.4.13.0 are interpreted as separate class C addresses, they will require separate entries in the routing table. A new, flexible way of defining addresses is needed.

Evaluating addresses according to the class rules discussed above limits the length of network numbers to 8, 16, or 24 bits - 1, 2, or 3 bytes. The IP address, however, is not really byte-oriented. It is 32 contiguous bits. A more flexible way to interpret the

network and host portions of an address is with a bit mask. An address bit mask works in this way: if a bit is on in the mask, that equivalent bit in the address is interpreted as a network bit; if a bit in the mask is off, the bit belongs to the host part of the address. For example, if address 195.4.12.0 is interpreted as a class C address, the first 24 bits are the network numbers and the last 8 bits are the host addresses. The network mask that represents this is 255.255.255.0, 24 bits on and 8 bits off. The bit mask that is derived from the traditional class structure is called the default mask or the natural mask.

However, with bit masks we are no longer limited by the address class structure. A mask of 255.255.0.0 can be applied to network address 195.4.0.0. This mask includes all addresses from 195.4.0.0 to 195.4.255.255 in a single network number. In effect, it creates a network number as large as a class B network in the class C address space. Using bit masks to create networks larger than the natural mask is called supernetting, and the use of a mask instead of the address class to determine the destination network is called Classless Inter-Domain Routing (CIDR).

Specifying both the address and the mask is cumbersome when writing out addresses. A shorthand notation has been developed for writing CIDR addresses. Instead of writing network 172.16.26.32 with a mask of 255.255.255.224, we can write 172.16.26.32/27. The format of this notation is address/prefix-length, where prefix-length is the number of bits in the network portion of the address. Without this notation, the address 172.16.26.32 could easily be interpreted as a host address. RFC 1878 list all 32 possible prefix values. But little documentation is needed because the CIDR prefix is much easier to understand and remember than are address classes. I know that 10.104.0.19 is a class A address, but writing it as 10.104.0.19/8 shows me that this address has 8 bits for the network number and therefore 24 bits for the host number. I don't have to remember anything about the class A address structure.

2.4.3 Internet-Legal Versus Private Addressing

Although the pool of IP addresses is somewhat limited, most companies have no problems obtaining them. However, many organisations have already installed TCP/IP products on their internal networks without obtaining "legal" addresses from the proper sources. Sometimes these addresses come from example books or are simply picked at random (several firms use networks numbered 1.2.3.0, for example). Unfortunately, since they are not legal, these addresses will not be usable when these organisations attempt to connect to the Internet. These firms will eventually have to reassign Internet-legal IP addresses to all the devices on their networks, or invest in address-translation gateways that rewrite outbound IP packets so they appear to be coming from an Internet-accessible host.

Even if an address-translation gateway is installed on the network, these firms will never be able to communicate with any site that is a registered owner of the IP addresses in use on the local network. For example, if you choose to use the 36.0.0.0/8 address block on your internal network, your users will never be able to access the computers at Stanford University, the registered owner of that address block. Any attempt to connect to a host at 36.x.x.x will be interpreted by the local routers as a request for a local system, so those packets will never leave your local network.

Not all firms have the luxury of using Internet-legal addresses on their hosts, for any number of reasons. For example, there may be legacy applications that use hardcode addresses, or there may be too many systems across the organisation for a clean upgrade to be successful. If you are unable to use Internet-legal addresses, you should at least be aware that there are groups of "private" Internet addresses that can be used on internal networks by anyone. These address pools were set-aside in RFC 1918, and therefore cannot be "assigned" to any organization. The Internet's backbone routers are configured explicitly not to route packets with these addresses, so they are

completely useless outside an organization's internal network. The address blocks available are listed in *Table 3*.

Table 3: Private Addresses Provided in RFC 1918

Class	Range of Addresses
A	Any addresses in 10.x.x.x
B	Addresses in the range of 172.16.x.x-172.31.x.x
C	Addresses in the range of 192.168.0.x-192.168.255.x

Since these addresses cannot be routed across the Internet, you must use an address-translation gateway or a proxy server in conjunction with them. Otherwise, you will not be able to communicate with any hosts on the Internet.

An important note here is that since nobody can use these addresses on the Internet, it is safe to assume that anybody who is using these addresses is also utilising an address-translation gateway of some sort. Therefore, while you will never see these addresses used as destinations on the Internet, if your organisation establishes a private connection to a partner organisation that is using the same block of addresses that you are using, your firms will not be able to communicate. The packets destined for your partner's network will appear to be local to your network, and will never be forwarded to the remote network.

There are many other problems that arise from using these addresses, making their general usage difficult for normal operations. For example, many application-layer protocols embed addressing information directly into the protocol stream, and in order for these protocols to work properly, the address-translation gateway has to be aware of their mechanics. In the preceding scenario, the gateway has to rewrite the private addresses (which are stored as application data inside the application protocol), rewrite the UDP/TCP and IP checksums, and possibly rewrite TCP sequence numbers as well. This is difficult to do even with simple and open protocols such as FTP, and extremely difficult with proprietary, encrypted, or dynamic applications (these are problems for many database protocols, network games, and voice-over-IP services, in particular). These gateways almost never work for all the applications in use at a specific location.

It is always best to use formally-assigned, Internet-legal addresses whenever possible, even if the hosts on your network do not necessarily require direct Internet access. In those cases in which your hosts are going through a firewall or application proxy of some sort, the use of Internet-legal addresses causes the least amount of maintenance trouble over time. If for some reason this is not possible, use one of the private address pools described in *Table 3*. Do not use random, self-assigned addresses if you can possibly avoid it, as this will only cause connectivity problems for you and your users.



Check Your Progress 2

- 1) If the first bit of an IP address is zero, then it belongs to
 - a) Class A
 - b) Class B
 - c) Class C
 - d) Class D

.....

- 2) The Internet backbone routers are configured explicitly not to route packets with _____ address.
- Legal
 - Private
 - Public
 - Virtual
-
- 3) _____ Works by extending the host portion of a subnet mask to the left, into the network portion of the address.
- Subnetting
 - Supernetting
 - Classless addressing
 - Hyper testing
-

2.5 IP ROUTING

An important function of the IP layer is *IP routing*. It provides the basic mechanism for routers to interconnect different physical networks. This means that an internet host can function as a normal host and a router simultaneously.

A basic router of this type is referred to as a *router with partial routing information*, because the router only has information about four kinds of destination:

- Hosts which are directly attached to one of the physical networks to which the router is attached.
- Hosts or networks for which the router has been given explicit definitions.
- Hosts or networks for which the router has received an ICMP redirect message.
- A default destination for everything else.

The last two items allow a basic router to begin with a very limited amount of information and to increase its information because a more sophisticated router will issue an ICMP redirect message if it receives a datagram and it knows of a better router on the same network for the sender to use. This process is repeated each time a basic router of this type is restarted.

Additional protocols are needed to implement a full-function router that can exchange information with other routers in remote network. Such routers are essential except in small networks, and the protocols they use are discussed in Routing Protocol.

2.5.1 Routing Protocol

The Routing protocols evolved in different phases. There are some references are as below:

- RFC 1074 - The NSFNET Backbone SPF Based Interior Gateway Protocol.
- RFC 1092 - EGP and Policy Based Routing in the New NSFNET Backbone.
- RFC 1093 - The NSFNET Routing Architecture.
- RFC 1104 - Models of Policy Based Routing.

- RFC 1133 - Routing between the NSFNET and the DDN.
- RFC 1222 - Advancing the NSFNET Routing Architecture

These protocols use two important groups of routing algorithms.

2.5.2 Routing Algorithms

In this section, we discuss the Distance-Vector and Link-State, Shortest Path First (SPF) Routing Algorithms.

Distance-Vector

The term *Distance-Vector* refers to a class of algorithms that gateways use to update routing information. Each router begins with a set of routes for those networks or subnets to which it is directly attached, and possibly some additional routes to other networks or hosts if the network topology is such that the routing protocol will be unable to produce the desired routing correctly. This list is kept in a *routing table*, where each entry identifies a destination network or host and gives the “distance” to that network. The distance is called a *metric* and is typically measured in “hops”.

Periodically, each router sends a copy of its routing table to any other router it can reach directly. When a report arrives at router B from router A, B examines the set of destinations it receives and the distance to each. B will update its routing table if:

- A knows a shorter way to reach a destination.
- A lists a destination that B does not have in its table.
- A’s distance to a destination, already routed through A from B, has changed.

This kind of algorithm is easy to implement, but it has a number of disadvantages:

- When routes change rapidly, that is, a new connection appears or an old one fails, the routing topology may not stabilize to match the changed network topology because information propagates slowly from one router to another and while it is propagating, some routers will have incorrect routing information.
- Another disadvantage is that each router has to send a copy of its entire routing table to every neighbour at regular intervals. Of course, one can use longer intervals to reduce the network load but that introduces problems related to how well the network responds to changes in topology.
- Vector-distance algorithms using hop counts as a metric does not take account of the link speed or reliability. Such an algorithm will use a path with hop count 2 that crosses two slow-speed lines, instead of using a path with hop count 3 that crosses three token-rings and may be substantially faster.

The most difficult task in a distance vector algorithm is to prevent instability. Different solutions are available:

Counting to infinity

Let us choose a value of 16 to represent infinity. Suppose a network becomes inaccessible, all the immediately neighbouring routers time out and set the metric to that network to 16. We can consider that all the neighbouring routers have a piece of hardware that connects them to the vanished network, with a cost of 16. Since that is the only connection to the vanished network, all the other routers in the system will converge to new routes that go through one of those routers with a direct but unavailable connection. Once convergence has happened, all the routers will have metrics of 16 for the vanished network. Since 16 indicates infinity, all routers then regard the network as unreachable.

The growth in networking over the past few years has pushed the currently available *Interior Gateway Protocols*, which use distance-vector algorithms, past their limits.

The primary alternative to vector-distance schemes is a class of protocols known as *Link State, Shortest Path First*.

The important features of these routing protocols are:

- A set of physical networks is divided into a number of areas.
- All routers within an area have an identical database.
- Each router's database describes the complete topology (which routers are connected to which networks) of the routing domain. The topology of an area is represented with a database called a *Link State Database* describing all of the links that each of the routers in the area has.
- Each router uses its database to derive the set of optimum paths to all destinations from which it builds its routing table. The algorithm used to determine the optimum paths is called a *Shortest Path First (SPF)* algorithm.

In general, a link state protocol works as follows each router periodically sends out a description of its connections (the state of its links) to its neighbours (routers are neighbours if they are connected to the same network). This description, called a *Link State Advertisement (LSA)*, includes the configured cost of the connection. The LSA is flooded throughout the router's domain. Each router in the domain maintains an identical synchronised copy of a database composed of this link state information. This database describes both the topology of the router's domain and routes to networks outside of the domain such as routes to networks in other autonomous systems. Each router runs an algorithm on its topological database resulting in a shortest-path tree. This shortest-path tree contains the shortest path to every router and network the gateway can reach. From the shortest-path tree, the cost to the destination and the next hop to forward a datagram is used to build the router's routing table.

Link-state protocols, in comparison with vector-distance protocols, send out updates when there is news, and may send out regular updates as a way of ensuring neighbour routers that a connection is still active. More importantly, the information exchanged is the state of a router's links, not the contents of the routing table. This means that link-state algorithms use fewer network resources than their vector-distance counterparts, particularly when the routing is complex or the autonomous system is large. They are, however, compute-intensive. In return, users get faster response to network events, faster route convergence, and access to more advanced network services.

Check Your Progress 3

- 1) Which of the following device provides the basic mechanism for interconnect different physical networks.
 - a) Gateways
 - b) Bridges
 - c) Routers
 - d) Hubs

.....

- 2) _____ routing algorithm all routers within an area have an identical database.
- a) Distance-vector
 - b) Link state
 - c) Longest-path
 - d) Shortest path
-

- 3) _____ Routing algorithms use features network resources than the distance-vector routing.
- a) Link-state
 - b) Policy based
 - c) Connection based
 - d) Hops based
-

2.6 SUMMARY

In this unit, you have studied the architecture of the Internet protocol, you must have learned different fields of IP header and their role in network communication. You learned about the different IP classes and their use in network design. You also studied about the practical issues of subnetting, supernetting and the Classless IP Addresses. Further in this unit we have given you comparisons between Internet-legal addressing and private addressing. In the end of this unit we have explained the procedure of IP routing with the help of Distance-Vector and Link-State routing algorithms. The next unit of this course covers about the transport layer and its role in TCP/IP.

2.7 ANSWERS TO CHECK YOUR PROGRESS

Check Your Progress 1

- 1) C 2) D 3) C

Check Your Progress 2

- 1) A 2) B 3) B

Check Your Progress 3

- 1) C 2) B 3) A

2.8 FURTHER READINGS

- 1) Achyut S. Godbole, *Web Technologies* TATA McGrawHill, 2003.
- 2) Berhouz Forouzan, *TCP/IP Protocol Suite*, 3rd edition, TATA McGraw Hill, 2006.
- 3) <http://www.tcpipguide.com>.
- 4) <http://www.cisco.com>.

UNIT 3 TRANSPORT LAYER PROTOCOLS

Structure	Page Nos.
3.0 Introduction	61
3.1 Objectives	61
3.2 Overview of TCP	62
3.3 Transmission Control Protocol (TCP)	63
3.3.1 TCP Header	64
3.3.2 TCP Connection Establishment and Termination	67
3.3.3 TCP Connection Establishment	67
3.3.4 TCP Connection Termination	68
3.4 User Datagram Protocol (UDP)	72
3.5 Summary	74
3.6 Answers to Check Your Progress	75
3.7 Further Readings	75

3.0 INTRODUCTION

The internet work environment consists of hosts connected to networks that are in turn interconnected via gateways. Such networks are based on packet switching technology. The active elements that produce and consume messages are the *processes*. Therefore, such communication can be viewed as an inter-process communication.

As the computer communication systems are playing an important role in military, government and civilian environments, it has become essential to provide some means of interconnecting the communication networks and to provide standard inter-process communication protocols that can support a broad range of applications. In anticipation of the need for such standards, the Department of Defence (DoD), USA, decided to employ Transmission Control Protocol (TCP) as the basis of inter-process communication protocol.

3.1 OBJECTIVES

Our objective is to introduce you to the basic concepts of Transport layer Protocols. On successful completion of this unit, you should be able to:

- have a reasonable understanding of the Transmission Control Protocol architecture;
- understand the role and use of port numbers;
- describe the operation of Transmission Control Protocol and its header format;
- describe the role and meaning of User Datagram Protocol (UDP); and
- describe the role and meaning of Internet Control Message Protocol (ICMP).

Ports

When a process starts up, it registers a port number with the protocol stack. The port numbers are specified by a 16-bit number i.e., the overall available set of port numbers are 2^{16} , ranging from 0 to 65535. The various categories of ports are as under:

- 1) **Well Known Ports:** The Port numbers in the range 0-1023 are called 'Well Known Ports'. These port numbers are assigned to the server side of an application and are already reserved for specific applications by IANA (Internet Assigned Number Authority). Some of these port numbers and their associated application are given in *Table 1*.

Table 1: Port number and associated application

Port Nos.	Protocol	Application	Port No	Protocol	Application
23	TCP	TELNET	110	TCP	POP3
25	TCP	SMTP	161	UDP	SNMP
37	UDP	TIME	179	TCP	BGP
43	TCP	WHOIS	443	TCP	HTTPS
53	TCP/UDP	DNS	68	UDP	BOOTPC
67	UDP	BOOTPS	69	UDP	TFTP

- 2) **Registered Ports:** Port numbers in the range 1024-49151 are called Registered Ports. These port numbers have been publicly defined as a convenient service for the Internet community to help them avoid vendor conflicts.
- 3) **Dynamic and/or Private Ports:** The remaining port numbers in the range 49152-65535, are called Dynamic and/or Private Ports and can be used freely by any client or server application.

Remember, only one process per protocol can listen on a given port i.e., two different processes, one using UDP and another TCP can both listen on port number XXX. However, two processes using the same transport protocol cannot listen on the same port number.

In order to establish a connection between two end machines i.e., providing the process-to-process delivery of messages, we need an identifier called socket. It contains the combination of IP address of the machine along with the port number of the process running on it. For example, suppose IP address of the machine is 171.10.20.1 and a process running on it is assigned a port number of 4534, then the socket contains (171.10.20.1, 4544). Thus, a socket uniquely identifies the process from the set of processes running on a machine.

In the client server model, a pair of sockets is required i.e., one for the client (*client IP address, client port number*) and another for the server (*server IP address, server port number*).

3.2 OVERVIEW OF TCP

In the previous discussion, we have discussed that in data link layer, frames are delivered node-to-node (intermediate nodes) and in the network layer, datagrams are delivered host-to-host. But the real communication takes place between two processes running on two different machines. Therefore, the basic function of the transport layer is process-to-process delivery of messages. However, it is also necessary to identify the various kinds of processes running on the machine (e.g., e-mail, ftp, telnet). Thus, we need some kind of addressing scheme for transfer of data between the processes.

In order to deliver some information to a specific destination, we need an address for correspondence. Similarly, at the data link layer, medium access control addresses (MAC) are employed for addressing. Correspondingly, at the network layer, Internet Protocol IP addresses are required. In the same way, the transport layer uses the port numbers for addressing between two processes. These port numbers are just a way of labelling the packets. When the transport layer receives a message from an application, first of all, it registers a temporary port number for the application process and subsequently assigns it. Thereafter, it sends the data to the designated destination application program that would be running on another port number. In *Figure 1*, the client server model has been illustrated, wherein, whenever a client process starts up, a port number is assigned, say 3000 and the server process is assigned port number of 100. These assigned port numbers are put on each packet transmitted between these processes.

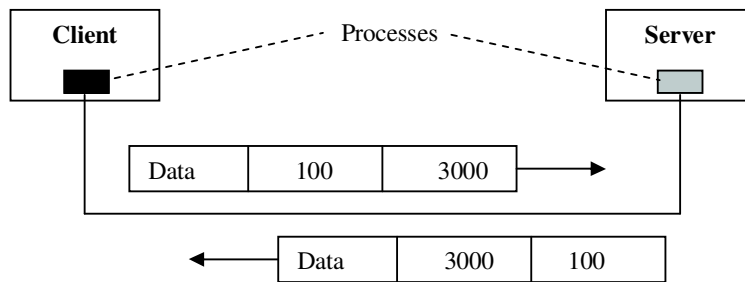


Figure: 1 Client Server Model

It may be noted that IP addresses and port numbers play different roles in communication networks. The IP address simply identifies the destination machine from the network, while the port number identifies the particular process running on that destination machine.

There are two types of protocols at the transport layer as shown in *Figure.2*.

- 1) Transmission Control Protocol (TCP)
- 2) User Datagram Protocol (UDP).

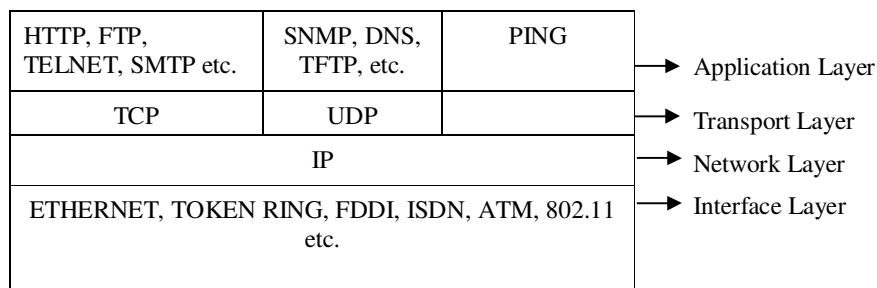


Figure 2: Protocol Stack

3.3 TRANSMISSION CONTROL PROTOCOL (TCP)

Transmission control protocol (TCP) of the transport layer provides end-to-end reliable communication. TCP is a connection oriented communication service. As the IP layer of TCP/IP protocol stack is connectionless, the TCP layer adds the reliability feature along with connection-oriented service. Basically, TCP establishes a virtual circuit between the source and the destination and the stream of bytes are transferred through that circuit such that the order of the segments remains integral.

The acknowledgement and retransmission of lost and damaged segments is guaranteed. In TCP, two processes running on two different hosts communicate with each other as shown in *Figure 3*.

The various services being offered by TCP to the processes running on application layer are as under:

- 1) **Stream data transfer:** With stream data transfer, TCP delivers an unstructured stream of bytes identified by sequence numbers. This service benefits applications because they do not have to chop data into blocks before handing it off to TCP. Instead, TCP groups bytes into segments and passes them to IP for delivery.
- 2) **Reliability:** TCP offers reliability by providing connection-oriented, end-to-end reliable packet delivery through an internetwork. It does this by sequencing bytes with a forwarding acknowledgement number that indicates to the destination, the next byte the source expects to receive. Bytes not acknowledged within a specified time period are retransmitted. The reliability mechanism of TCP allows devices to deal with lost, delayed, duplicate or misread packets. A time-out mechanism allows devices to detect lost packets and request retransmission.

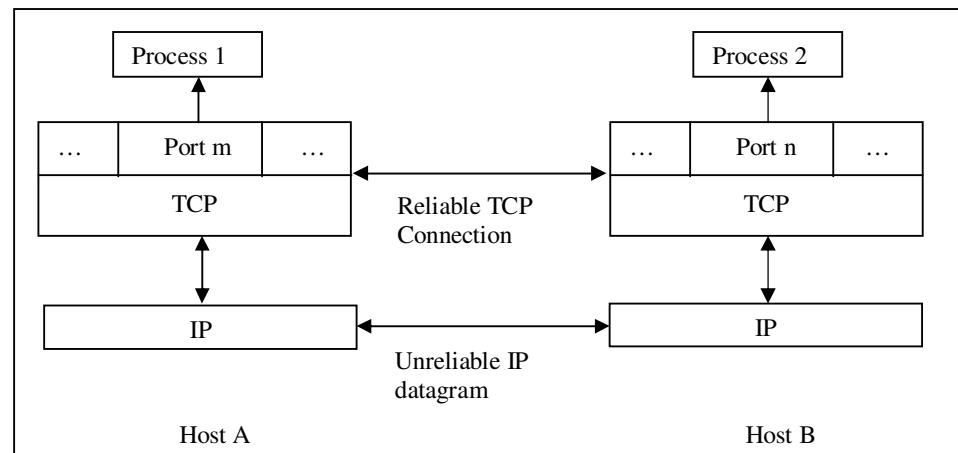


Figure 3: TCP Connection between two processes running on different machines

- 3) **Efficient flow control:** TCP offers efficient flow control, which means that, when sending acknowledgements back to the source, the receiving TCP process indicates the highest sequence number it can receive without overflowing its internal buffers.
- 4) **Full-duplex operation:** TCP offers full-duplex operation, wherein TCP processes can both send and receive data at the same time.
- 5) **Multiplexing:** TCP's multiplexing provides numerous simultaneous upper-layer conversations multiplexed over a single connection.

3.3.1 TCP Header

The TCP data unit is called a *segment*. The format of the segment is shown in *Figure 4*.

Source Port address 16 bits								Destination Port address 16 bits							
Sequence Number 32 bits															
Acknowledgement Number 32 bits															
HLEN 4 bits		Reserved 6 bits		U R G	A C K	P S H	R S T	S Y H	F I N	Window Size 16 bits					
Checksum 16 bits										Urgent Pointer 16 bits					
Options										Padding					

Figure 4: TCP Header

- 1) **Source Port Number:** This is a 16-bit number which defines the source port number for a particular application program that is sending the TCP segments.
- 2) **Destination Port Number:** This is a 16-bit number which defines the destination port number for a particular application program that is receiving the TCP segments.
- 3) **Sequence Number:** As the unit of data transfer in TCP is termed as segment, each segment's first data byte number denotes the 32-bit sequence number. Since the sequence number refers to a byte count rather than a segment count, sequence numbers in contiguous TCP segments are not numbered sequentially. For example, if a file of 5000 bytes is transferred using TCP connection and the first byte is numbered as 20002 and data divided into 5 segments each of 1000 bytes, the sequence numbers assigned to various segments are as under:

Segment 1 → sequence number: 20002
 Segment 2 → sequence number: 21002
 Segment 3 → sequence number: 22002
 Segment 4 → sequence number: 23002
 Segment 5 → sequence number: 24002
- 4) **Acknowledgement Number:** This is used by the sender to acknowledge the received data. This 36-bit field indicates the sequence number of the next byte expected from the receiver. For example, if host A has sent a segment having sequence number 2000 to host B, host B would send an acknowledgement with acknowledgement number field set to 2001 (one plus the sequence number of last received segment).
- 5) **Header Length:** The HLEN field consists of 4 bits. It indicates the length of the TCP header. The length of the TCP header can be between 20 bytes to 60 bytes i.e., HLEN field can have binary values ranging from 0101-1111 (5 to 15, 32 bit words) ($5 \times 4 = 20$, $15 \times 4 = 60$).
- 6) **Reserved:** This 6 bit field is reserved for future use. The value set in this field must be zero.

- 7) **Control Flags:** This field contains six different control flags that can control certain aspects of the TCP connection such as connection establishment, connection termination and flow control. The flags include:
 - a) **Urgent Pointer URG:** When set, the ACK indicates that the current segment contains urgent (or high-priority) data and that the Urgent Pointer field value is valid.
 - b) **Acknowledgement (ACK):** When set, indicates that the value contained in the Acknowledgement Number field is valid. This bit is usually set, except during the first message during connection establishment.
 - c) **Push (PSH):** PSH is used when the transmitting application wants to force TCP to immediately transmit the data that is currently buffered to the application without waiting for the buffer to fill. It is useful for transmitting small units of data.
 - d) **Reset (RST):** When set, RST immediately terminates the end-to-end TCP connection.
 - e) **Synchronize (SYN):** SYN is set in the initial segments used to establish a connection, indicating that the segments carry the initial sequence number.
 - f) **Finish (FIN):** FIN is set to request normal termination of the TCP connection in the direction this segment is travelling. Complete closure of the connection requires one FIN segment in each direction.
- 8) **Window Size:** The window size 16 bits field is used for flow control. It contains in bytes, the size of the window that the receiver has to maintain i.e., the value of the *receive window size*. It is basically the number of transmitted bytes that the sender of this segment is willing to accept from the receiver.
- 9) **Checksum:** This is a 16-bit field that provides bit error detection for the segment (including the header and data).
- 10) **Urgent Pointer:** Urgent data is information that has been marked as high-priority by a higher layer application. The data sent under high-priority usually bypasses the normal TCP buffering and is placed in a segment between the header and normal data. When the URG flag of control flag is set, then the urgent pointer 16-bit number indicates the position of the first octet of non-priority data in the segment.
- 11) **Options:** The option field contains 40 bytes of optional information about connection establishment. The maximum segment size (MSS) is the most commonly used option and if absent, defaults to an MSS of 536. Another option is Selective Acknowledgement (SACK), which allows out-of-sequence segments to be accepted by a receiver. The further discussion about options is beyond the scope of this book.

Characteristics of TCP

The basic characteristics of TCP are as follows:

- 1) It employs a connection-oriented service for communication.
- 2) It is a reliable source of communication i.e. guarantees delivery of messages.
- 3) It splits the messages into segments and keeps track of the order (sequence) of segments.
- 4) It employs the checksums for detecting any errors in data as well as the TCP header.

3.3.2 TCP Connection Establishment and Termination

A connection is a requirement of a reliable data delivery service. It is set up before the actual data exchange takes place. The connection is used to acknowledge the receipt of packets and retransmit those that are lost.

TCP provides a connection-oriented service over packet switched networks. 'Connection-oriented' implies that there is a virtual connection between two endpoints. There are three phases in any virtual connection. These are the connection establishment, data transfer and connection termination phases.

3.3.3 TCP Connection Establishment

In order to connect two machines, first of all both machines must initialize the communication and get a formal approval from each other before the start of data transfer. The establishment of such a connection by exchange of control messages is known as the three-way handshake. The process of the three-way handshake is as under:

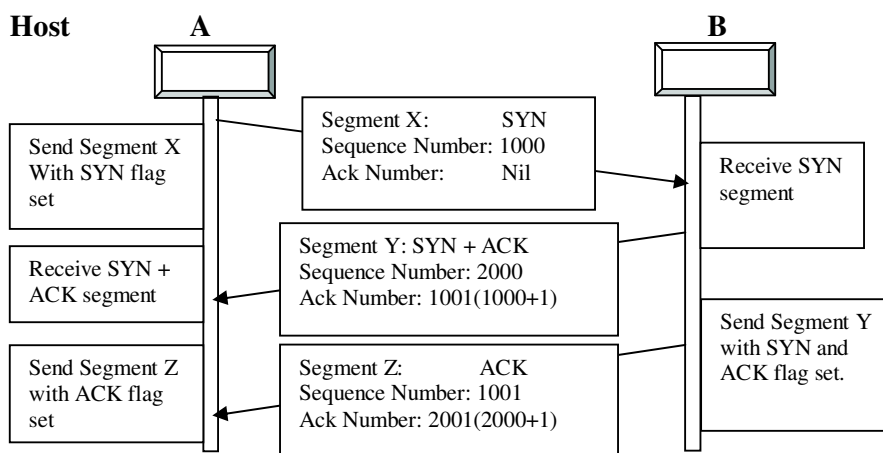


Figure 5: TCP Connection Establishment

- 1) The initiator of the session (source machine) sends a segment with SYN flag set to the destination node along with the proposed initial sequence number (ISN) in the sequence number field of the segment. Remember, the segment includes the source and destination port numbers. For example, in *Figure 5*, Host A sends a segment named X, with SYN flag set, initial sequence number is 1000 and acknowledgement number is Nil.
- 2) Upon receipt of the segment, the recipient sends a segment with SYN flag set to the initiator with the acknowledgement number set to the sequence number (ISN) + 1 and the ACK flag is also set. The sent segment is assigned a new sequence number (ISN) for its own segment. For example, in *Figure 5*, Host B, after receiving the segment, sends a segment named Y with SYN + ACK flag set, initial sequence number assigned to the segment is 2000 and acknowledgement number is 1001(sequence number of segment sent by host A plus 1).
- 3) The initiator after receiving the segment sends a segment with ACK flag set in response to the recipient's SYN, with the acknowledgement number set to the recipient's sequence number (ISN) + 1. For example, in *Figure 5*, Host A, after receiving the segment, sends a segment named Z, with ACK flag set, sequence number assigned to the segment is 1001(the first segment sent by A had a sequence number of 1000) and acknowledgement number is 2001(sequence number of segment sent by host B plus 1).

3.3.4 TCP Connection Termination

There is a saying that, “all good things must come to an end” and so it is with TCP connections. After the exchange of data between the source and destination nodes, either of them can close the connection. Thus, after the user has sent all its data and wishes to close the TCP connection, four segments are required to completely close a connection gracefully from both directions. The connection termination phases are discussed below:

- 1) The initiator (source) sends a segment with the FIN flag set. For example, in *Figure 6*, Host A sends a segment named X, with FIN flag set, initial sequence number is 2000 and acknowledgement number is Nil.
- 2) Upon receipt (destination) of the segment, the recipient issues an ACK segment, in order to confirm the receipt of the initiator. The acknowledgement number is 1 plus the sequence number received from the initiator i.e. from the FIN segment. For example, in *Figure 6*, Host B, after receiving the segment, sends a segment named Y, ACK flag set, initial sequence number assigned to the segment is 4000 and ack no. is 2001(sequence number of segment sent by host A plus 1).
- 3) If the recipient (destination) still has some data to send to the initiator (source), it sends the data. Otherwise, the recipient (destination) sends a request for termination of connection from its side by sending a segment with FIN flag set.

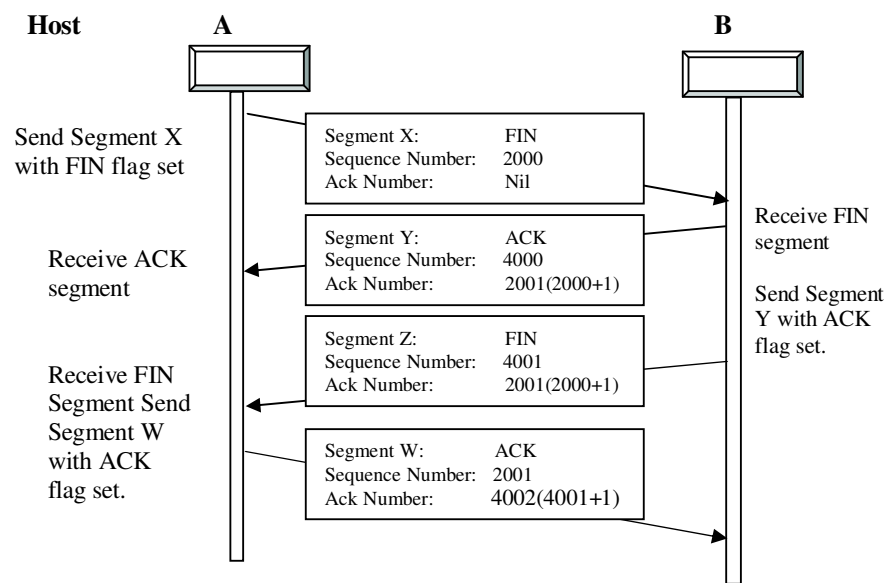


Figure 6: TCP Connection Termination

For example, in *Figure 6*, in case Host B too wishes to disconnect, it sends another segment named Z, with FIN flag set, sequence number assigned to the segment is 4001(sequence number of the last segment sent by B plus 1) and acknowledgement number is 2001(sequence number of segment sent by host A plus 1).

- 4) On receipt of this segment, the initiator sends an ACK segment, to confirm the FIN segment from the recipient (destination). For example, in *Figure 6*, Host A, after receiving the segment, sends a segment named W, with ACK flag set, sequence number assigned to the segment is 2001(the first segment sent by A had a sequence number of 1000) and acknowledgement number is 4002(the last sequence number of segment sent by host B plus 1).

Another method of closing the connection can be done by requesting the resetting of connection i.e., a host machine can send a segment with the RST bit set which will convey the other machine to immediately terminate the connection. Apart from

connection establishment and connection termination, flow control, error control and congestion control are few other responsibilities being performed by TCP.

Flow control in TCP

The flow control coordinates the amount of data that can be sent by the source between the terminals before receiving acknowledgement from the destination.

In order to speed up the communication, the source can send the complete message at a given instance without waiting for acknowledgement to arrive from the receiver. However, such a situation can cause adverse effects at the receiver side. In such situations, TCP sends data in accordance with sliding window protocol – a Flow Control Mechanism.

Sliding Window Protocol

In the sliding window protocol, a window is maintained for each connection. The window defines the size of the buffer i.e. the total number of bytes that can be sent by a terminal at a given time shown in *Figure 7*, this also shows the total number of blocks signifies the total size of the window. The sliding window also keeps track of bytes which have been sent but are unacknowledged, bytes still stored in buffer but haven't been sent etc.

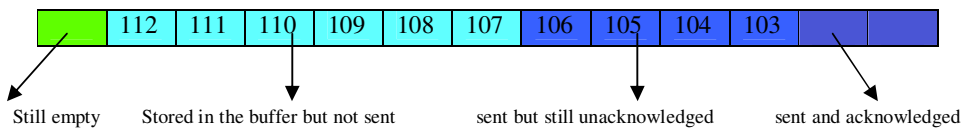


Figure 7: Sliding Window of sender node

Initially a window size is negotiated between the end terminals especially from the receiving side, while establishing a connection. Since TCP provides a byte-stream connection, sequence numbers are assigned to each byte in the stream. TCP divides this contiguous byte stream into TCP segments to transmit them. Therefore, the window principle is used at the byte level, that is, the segments sent and acknowledgements received will carry byte-sequence numbers and the window size is expressed as a number of bytes.

There are mainly two windows that are maintained i.e., one by the sender and another one by the receiver. The receiver window stores information about the various bytes received by its buffer but are still unconsumed and the other empty locations of the buffer.

With the help of these windows, a maximum limit on the number of bytes that can be sent by the sender to the receiver is decided, depending upon the total number of bytes that can be stored in the receiver's buffer at that particular instance. Thus, the mechanism of sliding window helps in controlling the flow of packets.

Error Control in TCP

The major tasks of error control in TCP include detection of out-of-order segments, lost segments, duplicate segments and corrupted segments. The above-mentioned problems are deciphered using the aid of acknowledgements, checksum and time-out period.

Out-of-Order segment

As IP is a connectionless service, IP datagrams might arrive out of order. The TCP in turn assures that the receiver does not acknowledge the received out of order segment until and unless it receives all those expected segments which precede it as shown in *Figure 8*.

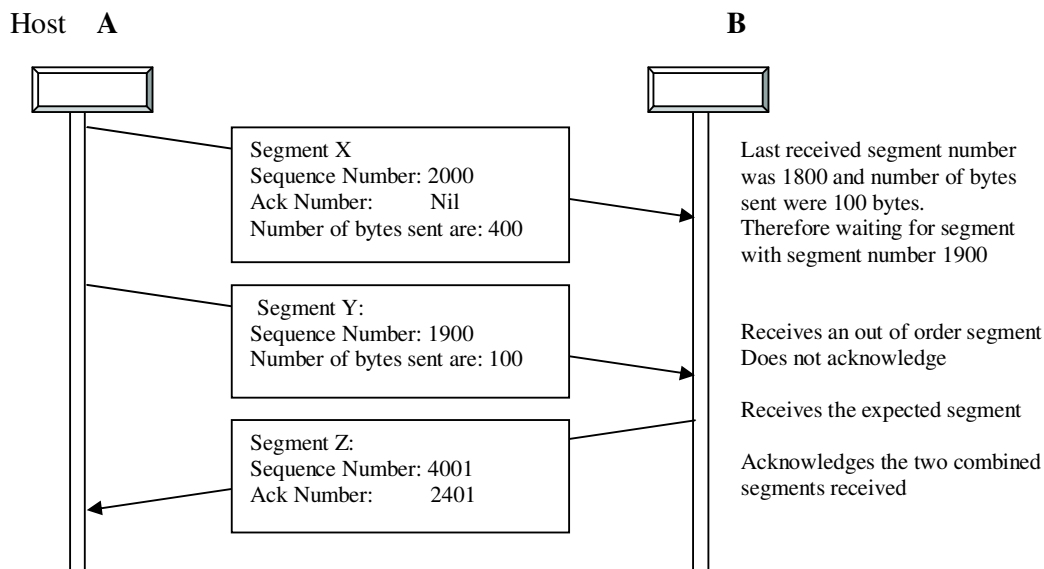


Figure 8: Out of order segments

Duplicate Segment

If the receiver receives a duplicate segment, it simply discards that segment, as a TCP segment with the same sequence number has already arrived.

Corrupted Segments

Whenever a segment gets corrupted, it is simply discarded by the destination and has to be retransmitted by the source as shown in *Figure 9*.

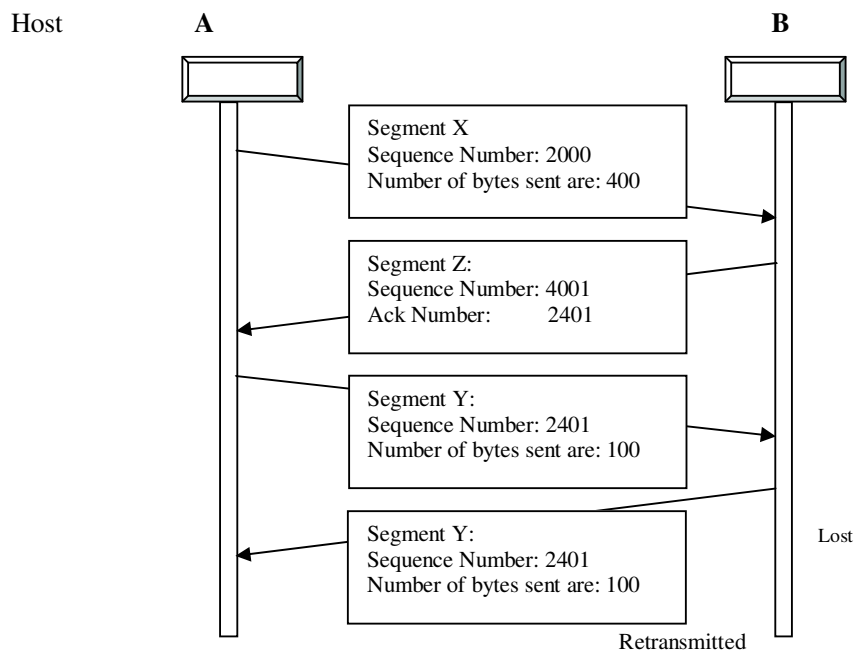


Figure 9: Corrupted Segment

Lost Acknowledgements

An acknowledgement for a segment can be lost in between. However, in TCP, the functioning of acknowledgement is as follows: An acknowledgement is a confirmation that everything up to the bytes specified by the acknowledgement

number in TCP header has been received. Thus, a latest acknowledgement overrides the previous reached /lost acknowledgements as shown in *Figure 10*.

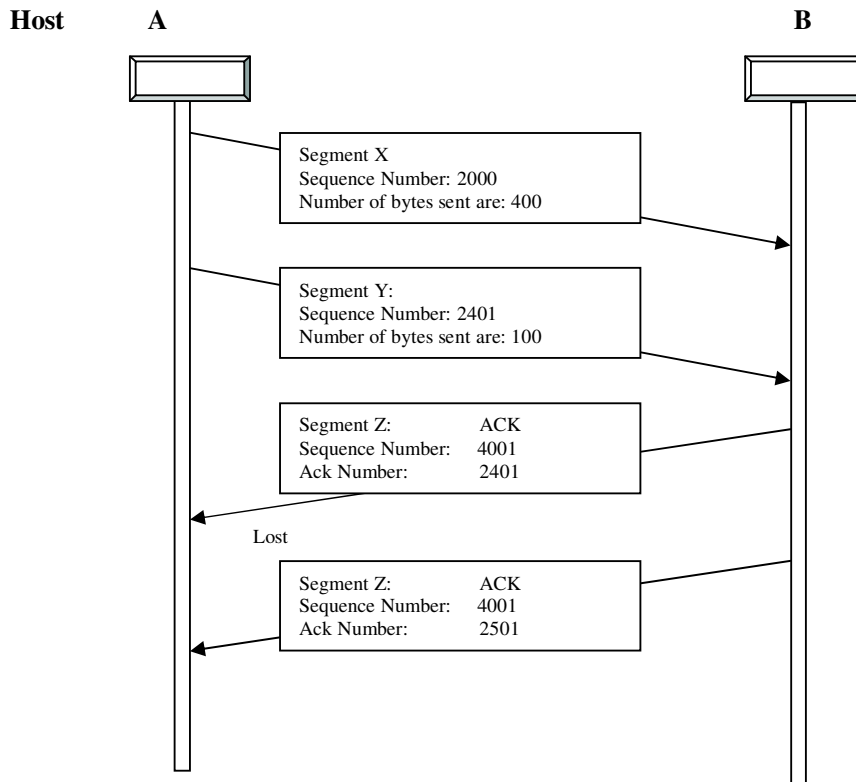


Figure 10: Lost Acknowledgement



Check Your Progress 1

1) Which of the following are transport layer protocols?

- a) TCP
- b) UDP
- c) IP
- d) Both a and b

.....

2) The combination of IP address and port numbers is known as-

- a) Socket Address
- b) Passive Address
- c) Transport Address
- d) Network Address

.....

3) Which of the following function is performed by the TCP.

- a) Process-to process communication
- b) End to end delivery of data
- c) Host to host communication
- d) None of the above

.....

- 4) How many types of port numbers are used?
- One
 - Two
 - Three
 - None of the above
-

- 5) Which of the following are parts of TCP header?
- Sequence Number
 - Acknowledgement Number
 - Checksum
 - All of the above.
-

3.4 USER DATAGRAM PROTOCOL (UDP)

UDP is an unreliable transport layer based protocol. It uses the connectionless service for providing the communication between the nodes of similar and/or different networks. The IP layer provides host-to-host communication. However, UDP provides process-to-process communication.

As UDP is connectionless, it means that no overheads will be incurred in terms of connection establishment, connection termination, acknowledgement of packets, sequence number assigned to each packet etc. Similar to the services offered by IP, UDP does not provide flow control and error control. In UDP, whenever the message is ready for transmission, it is simply transferred to the lower layers without bothering about establishment of connection. Thus, in situations where a small message has to be transmitted, UDP should be employed. On the other hand, the UDP is an unreliable source of communication. It does not guarantee accuracy, reachability and timeliness. Sequence numbers are not assigned to the packets, therefore, order of packets is not maintained and packets are unacknowledged.

UDP is a convenient protocol for multimedia and multicasting applications. It is suitable for processes that require simple request – reply communication with little interest in flow and error control.

UDP Header

The UDP packets are termed as *User Datagrams* and the header part of the user datagram has a fixed size of 8 bytes as shown in *Figure 11*.

- Source Port Number:** A 16-bit number which defines the source port number for a particular application program that is sending the UDP datagrams.
- Destination Port Number:** A 16-bit number that defines the destination port number for a particular application program that is receiving the UDP datagrams.

Source port number 16 bits	Destination port number 16 bits
Length 16 bits	Checksum 16 bits
Data	

Figure11: UDP Header

- 3) **Length:** The 16-bit field denotes the size of UDP header combined with payload data. It can range between 0 to 65,535 bytes.
- 4) **Checksum:** This is 16-bit field that provides detection of errors over the entire user datagram.

Characteristics of UDP

The basic characteristics of UDP are as follows:

1. UDP is a connectionless service.
2. It adds no non-reliable flow control to IP.
3. It serves as a multiplex/demultiplexer for sending and receiving datagrams.
4. The communication ends (end terminals) need not be synchronized.
5. There is no provision for acknowledgement of datagrams.

Applications of UDP

The standard applications using UDP include:

- Trivial File Transfer Protocol (TFTP)
- Domain Name System (DNS) name server
- Remote Procedure Call (RPC) used by the Network File System (NFS)
- Simple Network Management Protocol (SNMP).

Internet Control Message Protocol (ICMP)

The Internet Control Message Protocol notifies the sender of IP datagrams about abnormal events. ICMP is important in the connectionless environment. ICMP messages are carried in IP packets. The most commonly employed ICMP message types include:

- **Destination Unreachable:** This message type indicates that a packet cannot be delivered because the destination host cannot be reached. The reason for the non-delivery may be that the host or network is unreachable or unknown, the protocol or port is unknown or unusable.
- **Echo and Echo Reply:** These two messages are used to check whether hosts are reachable on the network. One host sends an Echo message to the other, optionally containing some data and the receiving host responds with an Echo Reply containing the same data. These messages are the basis for the Ping command.
- **Source Quench:** Sent by a router to indicate that it is experiencing congestion and is discarding datagrams.
- **TTL Exceeded:** This message indicates that a datagram has been discarded because the TTL field reached 0 or because the entire packet was not received before the fragmentation timer expired.
- **Timestamp and Timestamp Reply:** These messages are similar to the Echo messages, but place a timestamp (with millisecond granularity) in the message, yielding a measure of how long remote systems spend buffering and processing datagrams and providing a mechanism so that hosts can synchronize their clocks.



Check Your Progress 2

- 1) The UDP performs which of the following function:
 - a) Process-to process communication

- b) End to end delivery of data
 - c) Host to host communication
 - d) None of the above
 -
 -
- 2) Which of the following are parts of UDP header?
- a) Sequence Number
 - b) Acknowledgement Number
 - c) Checksum
 - d) All of the above
 -
 -
- 3) Which of the following field (s) is/are required for ordering the packets?
- a) Sequence Number
 - b) Acknowledgement Number
 - c) Checksum
 - d) All of the above
 -
 -
- 4) What is the purpose of sliding window protocol?
- a) Congestion Control
 - b) Flow Control
 - c) Error Control
 - d) All of the above
 -
 -
- 5) How is TCP different from UDP?
-

3.5 SUMMARY

This unit provides a complete overview of TCP. We have learnt that the data link layer provides node-to-node (intermediate), network layer provides host-to-host and transport layer provides process-to-process communication. The transport layer uses TCP and UDP protocols. TCP is a reliable, connection-oriented service, while UDP is an unreliable and connectionless service. As real communication takes place between two application programs, each process should be assigned a unique port number for identifying the various processes running on the same machine. There are three types of port numbers: well known, registered and dynamic ports. The combination of IP address and port number is called a socket address. The size of UDP header is 8 bytes and TCP header requires 20-60 bytes. The TCP employs error control, flow control and congestion control mechanisms. A three-way handshake is required for TCP connection establishment. The TCP connection termination requires four steps. The next unit of this course covers the application layer protocols and their role in TCP/IP.

3.6 ANSWERS TO CHECK YOUR PROGRESS

Check Your Progress 1

- 1) D
- 2) A
- 3) A
- 4) C
- 5) D

Check Your Progress 2

- 1) A
- 2) C
- 3) A
- 4) B
- 5) TCP is a connection-oriented service and UDP is a connectionless service.

3.7 FURTHER READINGS

- 1) Andrew S. Tanenbaum, *Computer Networks*, Third edition.
- 2) Behrouz A. Forouzan, *Data Communications and Networking*, Third edition.
- 3) Douglas E. Comer, *Internetworking with TCP/IP Vol.1: Principles, Protocols, and Architecture* (4th Edition).
- 4) James F. Kurose, *Computer Networking: A Top-Down Approach Featuring the Internet* (3rd Edition).
- 5) Larry L. Peterson, *Computer Networks: A Systems Approach*, 3rd Edition (The Morgan Kaufmann Series in Networking).
- 6) W. Richard Stevens, *The Protocols (TCP/IP Illustrated, Volume 1)*.
- 7) William Stallings, *Data and Computer Communications*, Seventh Edition.

UNIT 4 APPLICATION LAYER PROTOCOLS

Structure	Page Nos.
4.0 Introduction	76
4.1 Objectives	76
4.2 Domain Name System (DNS)	77
4.2.1 Hierarchical Name Space	77
4.2.2 Domain Servers	78
4.2.3 How does DNS Work in Internet	79
4.2.4 Domain Name Resolution	80
4.2.5 Messages Used in DNS	80
4.2.6 Dynamic DNS (DDNS)	82
4.3 Electronic Mail	83
4.3.1 Simple Mail Transfer Protocol (SMTP)	83
4.3.2 Message Transfer Agent	83
4.3.3 User Agent	84
4.3.4 Post Office Protocol (POP)	85
4.3.5 Internet Mail Access Protocol (IMAP)	85
4.3.6 Multipurpose Internet Mail Extension (MIME)	85
4.4 Telnet	87
4.5 File Transfer Protocol (FTP)	88
4.6 Summary	92
4.7 Answers to Check Your Progress	92
4.8 Further Readings	93

4.0 INTRODUCTION

The application layer provides means for the user to access information on the network through an application. This layer is the main interface/environment for the user to interact with the application and therefore with the network. The application layer can be assumed to be the level of TCP/IP protocol hierarchy where the user-accessed network processes reside. The applications running at the application layer are some network processes that occur above the transport layer. This includes all the processes that the users can interact with. Thus, application layer provides the services user applications communicate through the network for e.g., SMTP, FTP, Telnet and Rlogin. In this unit, we shall discuss the various standard services such as DNS, FTP, Telnet and SMTP being offered by the application layer.

4.1 OBJECTIVES

Our objective is to introduce you to the basic concept of the application layer and its protocols. On successful completion of this unit, you should be able to:

- have a reasonable understanding of the application layer;
- describe the operation of application layer protocols;
- understand the role and meaning of Domain Name System (DNS); and
- describe and understand the working of Simple Mail Transfer Protocol (SMTP), Telnet and File Transfer Protocol (FTP).

As we already know, each host machine on the Internet is assigned an IP address. The 32-bit IP address is represented in the numerical form, e.g., 202.12.32.22. However, it is very difficult for a user to remember the address of a machine in terms of an IP address. Therefore, the IP addresses have been denoted as a set of characters in English e.g., for the other name, of IP address 68.142.226.32 is yahoo.com. This human readable name in place of an IP address is known as domain name, e.g., google.com. The domain names are registered by an organization called ICANN. The domain names are not case sensitive and can contain alphabetic or numeric letters or the hyphen. Whenever a user accesses the Internet, he or she types the domain name instead of an IP address. Therefore, there is a need for a system that can convert the domain names of hosts into IP addresses and *vice-versa*. Such conversions are carried out by a system known as Domain Name System. The main function of Domain Name System (DNS) is to map the IP addresses into domain names (human readable names).

4.2.1 Hierarchical Name Space

The DNS is a large database that resides on various computers and it contains the IP addresses of various hosts on the Internet and various domains. The Domain Name System (DNS) is basically a distributed Internet directory service. In this system, the host needs to map the closest DNS computer machine which has the required information.

In order to assign the names to various machines, some kind of naming system was required. Thus, a hierarchical name space was designed such that each domain name was divided into several parts of the tree, e.g., the domain name 'yahoo.com' consists of two parts. First the domain name is read from right to left, i.e., com signifies a commercial website (upper level of hierarchical system) and yahoo denotes the name of the website (lower level of hierarchical system) see *Figure 1*.

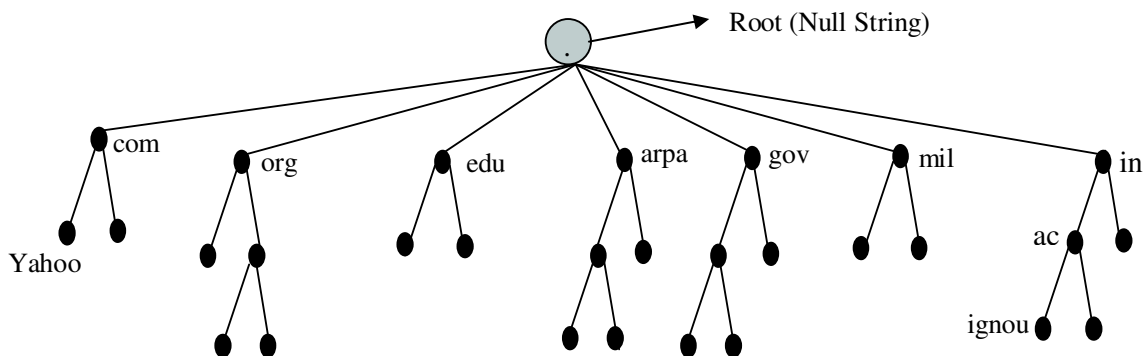


Figure 1: Hierarchical Name Space for DNS

In DNS, a tree structure has been designed such that root of the tree binds the complete tree. The maximum levels of the tree are 128 and each label of the tree can have a string of 63 characters. The root of the DNS hierarchy (tree) is designated with a period '.'. Thereafter the tree contains a group of top-level domains including familiar names like *com*, *org*, *edu*, various country-level domains like *in* (India) etc. However, the domain names at this level cannot be assigned to an individual machine. The next levels of the DNS tree are the second-level registered domains such as hotmail.com. Below the second level, local domains names can be specified. Remember, the domain names are always read from the current node up to the root of the tree as shown in *Figure 2*.

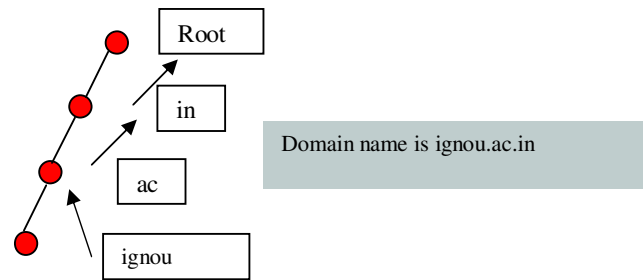


Figure 2: Labeling for domain names

The above mentioned tree structure has to be stored into the computer machine. However, storage of such a huge amount of information on a single computer system is prone to inefficient computing due to the following reasons:

1. Availability of the system.
2. Reliability.
3. Computing capacity of the system.
4. Network load on that particular transmission link.

4.2.2 Domain Servers

Due to these reasons, domain name information has been distributed among various servers known as Domain Servers. The server is responsible or has an authority over a specific region called a zone i.e., the DNS database is divided into zones. The servers in their respective zones are responsible for answering queries for their zones and are called name servers.

A name server is a server program that holds a master or a copy of a name-to-address mapping database, or otherwise points to a server that does and that answers requests from the client software, called a name resolver. Conceptually, all Internet domain servers are arranged in a tree structure that corresponds to the naming hierarchy.

A zone is simply a sub-tree of DNS and is administered separately. There are multiple name servers for a zone. There is usually one primary name server and one or more secondary name servers. A name server may be authoritative for more than one zone.

The root server is the server whose zone consists of the complete tree. Basically a root server does not keep information about domain names but actually delegates its own authority to other servers. At present, there are around 13 root servers distributed all over the world which are able to cover the entire set of domain names.

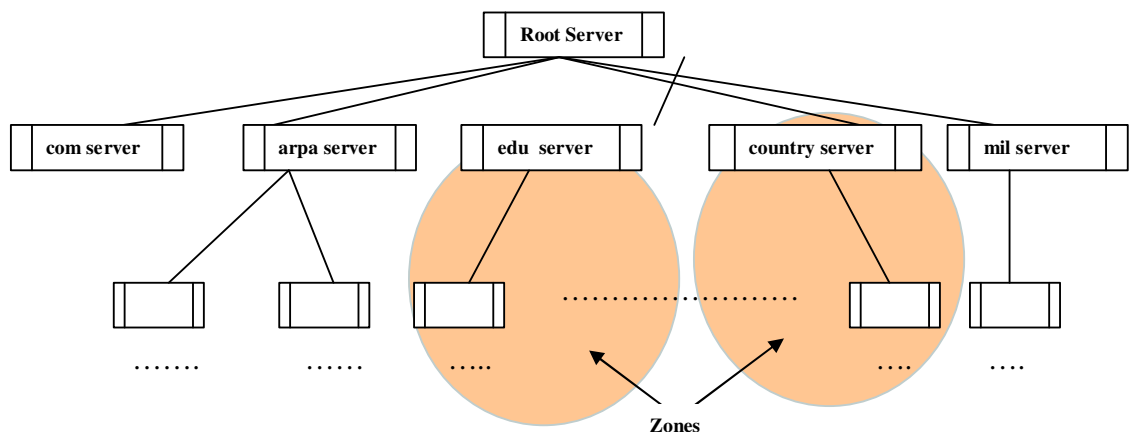


Figure 3: Name Servers

Thus, we have discussed in general the working of Domain Name Systems.

4.2.3 How does DNS Work in Internet?

The *hierarchical* system of domain names in Internet has been broadly classified into three categories:

1. Generic domain names
2. Country based domain names
3. Inverse domains

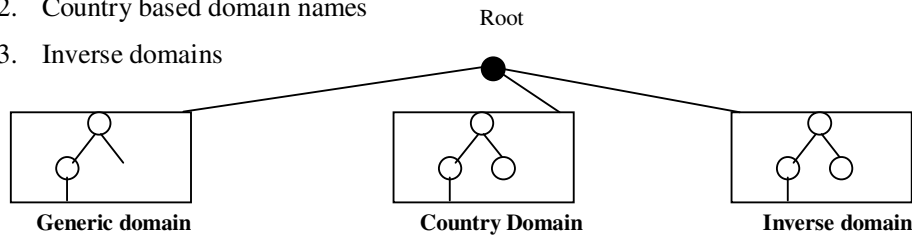


Figure 4: Hierarchy of Domains in Internet

The three-character top-level names are called the generic domains or the organisational domains. The generic domain defines registered hosts in accordance with their behaviour for example, yahoo.com uses 'com' as top level name as it signifies that yahoo is a commercial organisation. *Table 1* shows some of the top-level domains of today's Internet domain hierarchy.

Table 1: Generic Domain Names

Domain Name	Meaning
Com	Commercial organisations
Gov	Government institutions
Org	Non-profit organisations
Mil	Military groups
Edu	Educational institutions
Net	Major network support centers
Int	International organisations

There are also top-level domains named for each of the ISO 3166 international 2-character country codes (e.g., zw for Zimbabwe). These are called the country domains. Many countries have their own second-level domains underneath, which parallel the generic top-level domains.

As it is a simple matter in principle to search the database for an IP address with its symbolic name (because of the hierarchical structure), the reverse process cannot follow the hierarchy. Therefore, there is another namespace for the reverse mapping called Inverse domain. The information about the inverse domain is always stored in the domain "in-addr.arpa". Here, the IP addresses are stored in a hierarchical form such that each level of the tree depicts 8 bits of information of IP address. However, since domain names have the least significant parts of the name first, but dotted decimal format has the most significant bytes first, the dotted decimal address is shown in reverse order. For example, the domain in the domain name system corresponding to the IP address 171.10.1.2 is 2.1.10.171.in-addr.arpa. The above mapping is carried out by using a special kind of query called inverse pointer query.

4.2.4 Domain Name Resolution

The concept of mapping a domain name to an IP address and *vice-versa* is known as resolution process. The resolution process is basically a client server platform. Whenever a user needs to map an address to a domain or *vice-versa*, the DNS calls a client program called resolver. The resolver subsequently contacts the nearest DNS server (name server) with a request. In case the server has the desired information, it replies back with the results. Otherwise it suggests the resolver to other domain servers or asks other servers to provide the desired information. The resolver, after receiving the results asserts the information and thereafter delivers the desired information to the specific host process. The steps followed in the resolution are below:

- 1) The user program issues a request such as the `gethostbyname()` system call. (*This particular call is used to ask for the IP address of a host by passing the host name as a parameter*).
- 2) The resolver formulates a query to the name server.
- 3) The name server checks to see if the answer is in its local authoritative database or cache, and if so, returns it to the client. Otherwise, it will query other available name server(s), starting down from the root of the DNS tree or as high up the tree as possible.
- 4) The user program will finally be given a corresponding IP address (or host name, depending on the query) or an error if the query could not be answered.

As the resolution process is carried out with the help of queries, these domain name request queries can be one of two types: *recursive* or *iterative*. A flag bit in the domain name query specifies whether the client desires a recursive query, and a flag bit in the response specifies whether the server supports recursive queries. A recursive query requests that the server should itself issue a query to determine the requested information and return the complete answer to the client. However, an iterative query means that the name server should return what information it has available and also a list of additional servers for the client to contact to complete the query.

The concept of caching is also utilised in DNS. Whenever a name-server receives a request from a client, the name server subsequently sends the query to other servers. Later on, when it receives the response, it first stores this information in its own cache memory before sending the desired information to the client. Now onwards, if any client desires the same information, first the name-server can simply check its cache memory and resolve the query. However, such a response is designated as non-authoritative. The domain name responses from the name server can be one of two types: authoritative and non-authoritative.

4.2.5 Messages used in DNS

As DNS follows the client server paradigm, it has two types of messages: Query and Response. The messages in the DNS use a single format i.e., similar format. The general format of a query message and that of a response message are shown in Figures 5 and 6.

Header
Question Section

Figure 5: Query Message

Header
Question Section
Answer Section
Authoritative Section
Additional information Section

Figure 6: Response Message

The format of the header of the message is shown in *Figure 7*. The header section is always present and has a fixed length of 12 bytes.

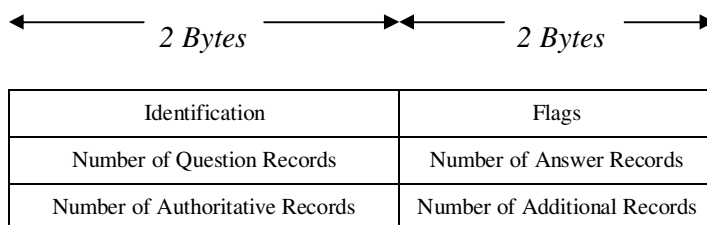


Figure 7: Header format for Query/Response

- **Identification** used by the source of the message in order to match the response of the query.
- **Flags** contains various fields that define the kind of message i.e., recursive, iterative, authoritative, non-authoritative etc.
- **Number of Question Records** contains the total number of queries asked by the resolver in the Question Section.
- **Number of Answer Records** contains the total number of answers specified in the Answer Section.
- **Number of Authoritative Records** contains the total number of Authoritative records in the Authoritative Section.
- **Number of Additional Records** contains the total number of Additional records in the Additional Section.

It may be noted that the Number of Answer Records, Number of Authoritative Records, Number of Additional Records sections will have ZERO value in the query message.

Example

Now let us take up an example and illustrate the complete procedure of mapping a domain name to an IP address.

- 1) Let us say a user opens a web browser and tries to connect to a website, say www.ignou.ac.in.
- 2) The operating system not knowing the IP Address for www.ignou.ac.in, asks the (Internet Service Provider) ISP's DNS Server for this information.
- 3) The ISP's DNS Server does not know this information, so it connects to a Root Server to find out what name server, running somewhere in the world, knows the information about ignou.ac.in.
- 4) The Root Server tells the ISP's DNS Server to contact a particular name server that knows the information about ignou.ac.in.
- 5) Thereafter, the ISP's DNS Server connects to IGNOU's DNS server and asks for the IP Address for www.ignou.ac.in.
- 6) IGNOU's DNS Server responds to the ISP's DNS server with the appropriate IP Address.
- 7) The ISP's DNS Server tells the User's operating system the IP Address for ignou.ac.in.
- 8) The operating system tells the Web Browser the IP Address for www.ignou.ac.in.
- 9) The web browser connects and starts communication with www.ignou.ac.in.

4.2.6 Dynamic DNS (DDNS)

The Dynamic Domain Name System (DDNS) is a protocol that defines extensions to the DNS in order to enable the DNS servers to accept the requests to add, update and delete entries in the DNS database dynamically. Because DDNS offers a functional superset to existing DNS servers, a DDNS server can serve both static and dynamic domains at the same time. Rather than allowing any host to update its DNS records, the secure version of DDNS uses public key security and digital signatures to authenticate update requests from DDNS hosts.

Three common utilities for querying name servers are provided with many DNS implementations:

- 1) **Host** obtains an IP address associated with a host name or a host name associated with an IP address.
- 2) **Nslookup** allows you to locate information about network nodes, examine the contents of a name server database and establish the accessibility of name servers.
- 3) **Dig** allows you to exercise name servers, gather large volumes of domain name information, and execute simple domain name queries. DIG stands for Domain Internet Groper.

The following RFCs define the Domain Name System standard and the information kept in the system:

- 1) RFC 1032 – Domain Administrator’s Guide
- 2) RFC 1033 – Domain Administrator Operations Guide
- 3) RFC 1034 – Domain Names – Concepts and Facilities
- 4) RFC 1035 – Domain Names – Implementation and Specification
- 5) RFC 1101 – DNS Encoding of Networks Names and Other Types.



Check Your Progress 1

- 1) In the following domain names, identify the name using a country based domain name
 - a) www.yahoo.com
 - b) www.ignou.ac.in
 - c) www.hotmail.com
 - d) www.aol.com

.....

.....
- 2) The domain name resolution can be carried out through which of these methods?
 - a) Iterative
 - b) Recursive
 - c) Caching
 - d) All of the above

.....

.....

.....

- 3) If the IP address of the machine is known and we want to enquire about its domain name, which one of the following is used?
- Generic domains
 - Country domains
 - Inverse domain
 - None of the above
-
-
-

4.3 ELECTRONIC MAIL

Electronic mail (e-mail) is probably the most popular service of TCP/IP. For most people, it has become an integral part of everyday life. Electronic mail provides a platform for exchanging information between two end users. It is basically used for sending and receiving mails/messages, e.g., textual, voice, graphical and video messages between end users. This section provides an overview of the TCP/IP application protocol dealing with electronic mail.

4.3.1 Simple Mail Transfer Protocol (SMTP)

The standard mechanism for electronic mail in the Internet is *Simple Mail Transfer Protocol (SMTP)*. It provides mail and message exchange between TCP/IP hosts. SMTP is based on *end-to-end delivery* i.e., an SMTP client contacts the destination host's SMTP server directly for delivering the mail. The destination host's SMTP server keeps the mail until the mail has been successfully copied into the recipient's SMTP. The SMTP is a connection service based on client-server environment and runs on port number 25 at the server side as shown in the *Figure 8*. The various components of SMTP are:

- Mail Transfer Agent (MTA)
- User Agent (UA)

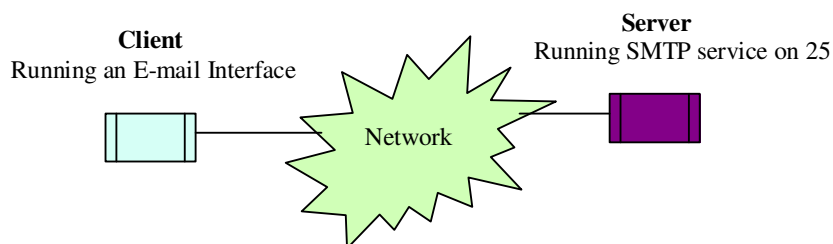


Figure 8: SMTP Service

4.3.2 Message Transfer Agent

The transfer of messages i.e., mails is delivered through an agent known as Message Transfer Agent (MTA). MTAs assist a user in sending as well as receiving the messages. The MTA consists of two shades i.e., MTA client for sending the mail while MTA server for listening/receiving the mails as shown in *Figure 10*. The MTA is generic and defines how the commands and responses should be sent back and forth. A popular example of MTA in UNIX is known as sendmail.

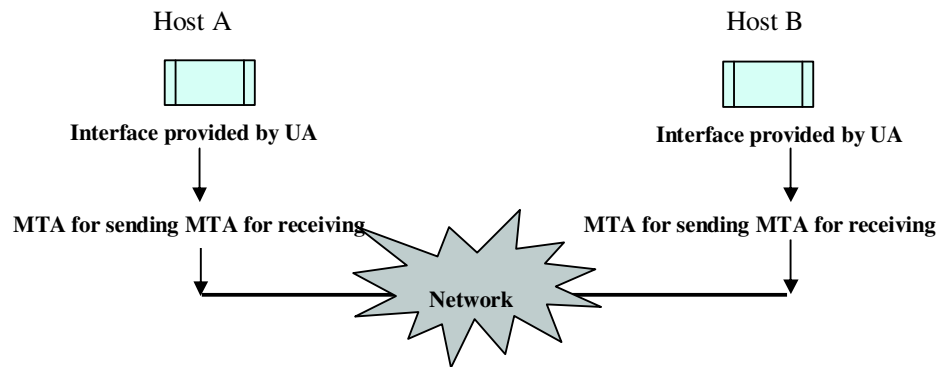


Figure 9: SMTP

4.3.3 User Agent

The user agent mainly deals with the composition of messages. The user agent provides an interface in the form as shown in *Figure 10* wherein s/he can write the message, specify the destination address (that is create an envelope). Therefore, UA puts the message into the envelope. The various services offered by the user agent are as under:

- 1) Reading the received messages
- 2) Replying to the read messages
- 3) Composing the messages
- 4) Forwarding the messages
- 5) Handling the various mailbox settings

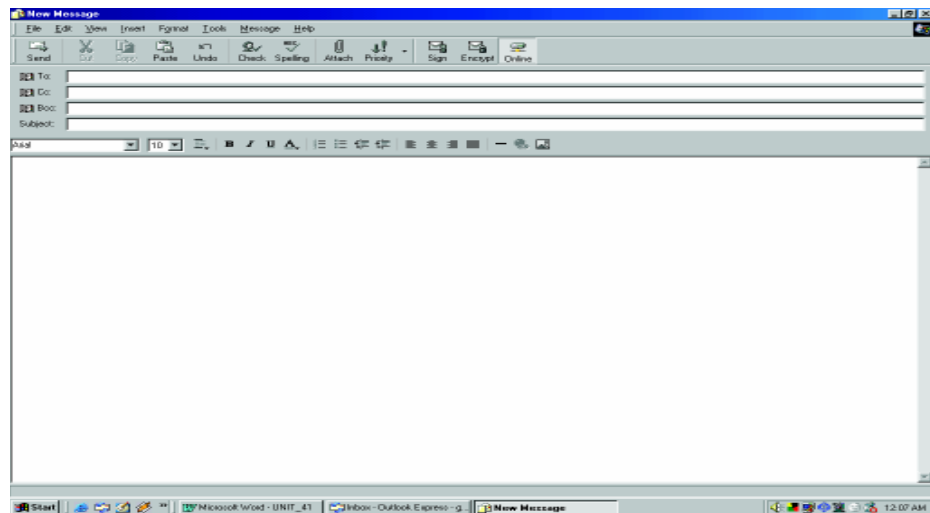


Figure 10: Interface provided by UA for composing a message

The address of a user (E-mail address) consists of two parts: the user name and the domain name. For example, amitgoel@yahoo.com is an e-mail address, wherein “amitgoel” depicts the user name, yahoo.com represents the domain name and these two parts are separated by an @ sign. Therefore, the mechanism of representing an e-mail address used by SMTP includes user name, @ and domain name. Remember that UA does not provide the facility for sending and receiving messages. The address of the destination host contains the domain name for identifying a particular network and thereafter a specific user on its network.

The interface presented by the user agents can be two types: command based or graphical interface based. The various examples of user agents with respect to command based interface are pine, mail etc. and with respect to GUI, some examples are: Netscape, outlook express etc.

4.3.4 Post Office Protocol (POP)

As SMTP is based on TCP/IP protocol, a TCP connection is required to be established between both ends. However, user machines cannot be expected to be online 24 x 7, especially a desktop machine owned by a home user. Therefore, there was a need for developing a system by which a user could receive his/her e-mails even though the machine was powered off. Therefore, most of the organisations install a SMTP server which is always online and receives e-mails on the behalf of each and every user of the organisation on its network. Basically the SMTP server acts as a “post office”.

In order to retrieve the e-mails stored in the SMTP server on the behalf of the users, a protocol called *Post Office Protocol (POP)* has been devised. It assists in downloading the e-mails from the SMTP server as shown in *Figure 11*.

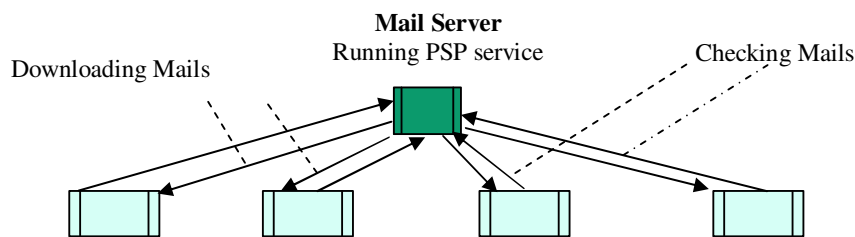


Figure 11: POP

4.3.5 Internet Mail Access Protocol (IMAP)

In POP, whenever a user accesses the mails from the mail server, i.e., downloads the mails, instantly those accessed mails are deleted from the mail server. Thus, POP is not suitable for people accessing their mails from various locations, i.e., cyber cafe, home, hotel etc. POP does not provide the facility for creating folders, organising the mails on the mail server etc.

In order to avoid the deletion of the mails from the mail server, another protocol called Internet Mail Access Protocol (IMAP) has been devised. In addition to services offered by POP, IMAP provides the following services:

- 1) The user can create, rename or delete the mailbox on the mail server.
- 2) The user can check the header part of the mail before downloading the message

4.3.6 Multipurpose Internet Mail Extension (MIME)

The SMTP protocol sends the mails i.e., the messages only in a Network Virtual Terminal seven-bit ASCII format. That is, it will support only those languages that can be represented by a seven-bit ASCII format. Therefore, messages written in languages such as German, Russian and French cannot be sent through SMTP. Moreover, SMTP does not support the transfer of video files, audio files and binary files. Thus, there was a need for developing a mechanism for allowing the transfer of in compliant formats.

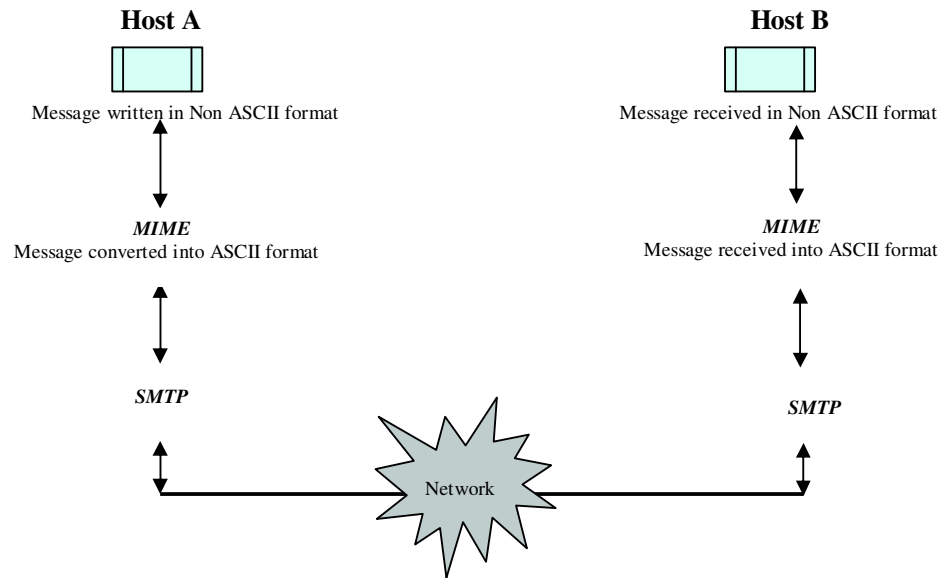


Figure 12: MIME

A protocol called Multipurpose Internet Mail Extension (MIME) supports transfer of Non-ASCII formats through SMTP. Primarily, MIME converts the non-ASCII formats into ASCII format and passes the data to SMTP. Consequently, the SMTP sends the ASCII form of data to the destination machine. The SMTP service at the destination machine passes the ASCII form of data to MIME which in turn converts the data into non-ASCII format as shown in *Figure 12*. Remember that MIME is simply an extension of SMTP and is not a mail protocol.

MIME defines five headers which can be supplemented with the original SMTP header in order to define the following parameters:

- **MIME Version:** It defines the version of MIME being used. The current version of MIME is 1.1
- **Content -Type:** It defines the type of data used in the body of the message, e.g., text, image, video, audio, postscript etc.
- **Content -Transfer-Encoding:** It defines the method to encode the mail message into 0's and 1's. The various kinds of encoding techniques are as follows: ASCII-7 bit, Non-ASCII-8 bit, Non-ASCII-Binary, Non-ASCII-Base64 and Non-ASCII-Quoted-printable.
- **Content-id:** It identifies the complete message in a multiple-message scenario.
- **Content-Description:** It describes whether the body of the message is text, image, video audio etc.



Check Your Progress 2

- 1) What is the purpose of Message Transfer Agents?
 - a) Creation of envelopes
 - b) Transfer of messages
 - c) Writing the messages
 - d) None of the Above

.....

.....

2) Which field of MIME header describes the method to encode the message?

- a) Content-type
- b) Content-id
- c) Content-transfer-encoding
- d) Content-description

.....

.....

3) How is the Non-ASCII formats converted into ASCII format?

- a) MIME
- b) POP
- c) IMAP
- d) SMTP

.....

.....

4) Give a difference between POP and IMAP.

.....

.....

4.4 TELNET

The most fundamental method of data communication employed on a network is the ability to perform remote execution i.e. calling an application on a remote terminal. TELNET is a widely known application protocol that provides remote execution capability.

TELNET is a popular client server application program. TELNET is an abbreviation for **Terminal Network**. TELNET is a standard application protocol that provides an interface, through which a program on a host i.e., *TELNET client* can access the resources of another host i.e., *TELNET server*. TELNET provides an environment such that the client acts as a local terminal connected to the server as shown in *Figure 13*. Basically, TELNET is a utility whereby a user first logs into a remote machine and thereafter the user can access the files / programs located remotely.

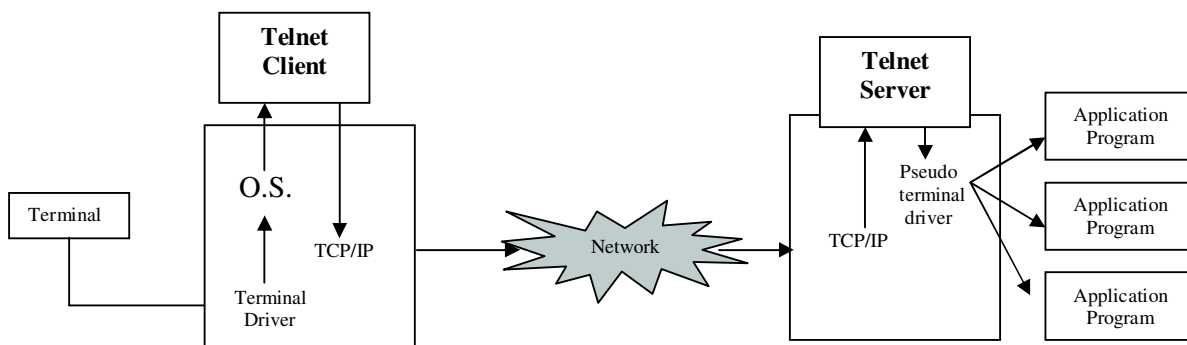


Figure 13: TELNET

The working of TELNET is as follows: The user types on the host machine /terminal that runs a driver known as terminal driver (a module of the operating system). It basically receives the keystrokes typed by the user on the terminal and passes the corresponding characters to the operating system. The operating system in turn sends these characters to the TELNET client. As discussed in SMTP, the language/format acceptable to the terminal might be different from the format allowed by TELNET. Therefore, the characters received by the TELNET client from the terminal driver are converted into a generic character set known as Network Virtual Terminal (NVT) characters. The transformed form of characters is sent to the TCP/IP protocol stack of the local machine.

The characters travel through the network using TCP/IP and finally reach the destination's operating system. The operating system passes the NVT characters to the TELNET server. The TELNET server in turn transforms the characters into characters understandable by the destination machine. The set of characters is sent to the operating system through a special kind of driver called pseudoterminal driver. As the TELNET server cannot directly interact with the operating system, the pseudoterminal driver helps the TELNET server in simulating a terminal. Thus, these characters from the operating system are delivered to the appropriate application program of the remote machine.

4.5 FILE TRANSFER PROTOCOL (FTP)

The transfer of files between two machines is one of the most common operations in a network. The standard mechanism for copying files from one machine to another is known as File Transfer Protocol (FTP).

Although it might seem quite simple to transfer data between two hosts, there are many issues which are to be resolved in such a transfer. For instance, two systems may use different file name conventions, different representation of data, different directory structures etc. These issues have been resolved with the help of FTP. FTP employs a client server environment. In order to access remote files in FTP, the user must firstly authenticate to the server before it allows the file transfer. Remember, FTP uses a connection-oriented service i.e., it is necessary to have both hosts up and running TCP/IP to establish a file transfer. After the establishment of connection, the data transfer between client and server takes place.

FTP uses TCP as a transport protocol to provide reliable end-to-end connection. The FTP server listens to connections on well-known port number 20 for transfer of data (data connection) and 21 for establishment / termination of connection (control connection). The control connection performs the task of authentication with the help of login and thereafter follows the TELNET protocol. As it is necessary to log into the remote host, the user must have a user name and a password to access files and directories.

The FTP application is built with a user interface at the top of the protocol, a Data Transfer Process (DTP) and protocol interpreter (PI) as shown in *Figure 14*. On the client side, the user interface communicates with the protocol interpreter which is in charge of the control connection. This protocol interpreter (PI) has to communicate any necessary control commands to the remote system. On the Server side, the protocol interpreter, besides its function of responding to the TELNET protocol, has to initiate the data connection. During the file transfer, the data management is performed by DTPs. After a user's request is completed, the server's PI has to close the data connection.

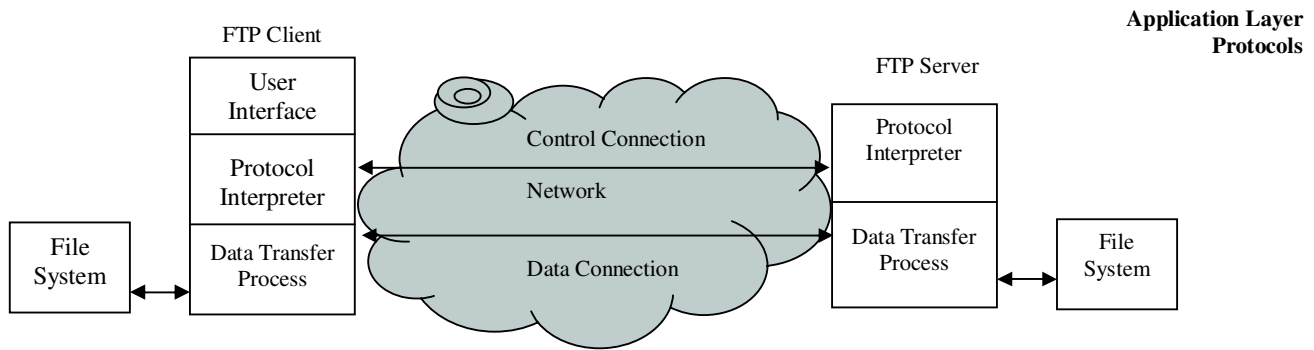


Figure 14: File Transfer Protocol (FTP)

The basic steps performed by a client during an FTP session are as given below:

1) **Connect the user to a remote host**

First the user has to authenticate the server machine about the user's authenticity with the help of a valid user name and password. The various commands that are required for connection establishment are:

- On the command prompt type "ftp".
- Then write a command "open" along with the IP address of the server machine, e.g., open 202.12.141.81. The open command imitates a login session on the remote machine.
- After connecting to the remote host, the remote machine enquires about the user name and its corresponding password.
ftp> login
ftp> password
- The other commands are user and pass. The purpose of user command is to identify a remote user id. The purpose of pass command is to authenticate the remote user id.

2) **Select a directory**

After connection is established, the client searches for the appropriate directory using "cd" (change directory) command. The user can select a local directory with the lcd (local change directory) command.

3) **List files available for transfer**

Commands like dir or ls are used for listing the various files in a particular folder depending upon the operating system.

4) **Define the transfer mode**

As the data transfer can take place between dissimilar systems, transformations of the data into an understandable form is required. The user has to mainly decide on two aspects of the data handling: first, the method through which the bits will be moved from one place to another and secondly, the different representations of data. In order to deal with the above issues, the following commands are used:

- **Mode:** It specifies whether the file is to be assumed as record structure in a byte stream format.
- **Type:** It specifies the character set being used for representation of data, i.e., ASCII, Image etc.

5) Copy files to or from the remote host

The following commands can be used to copy files between FTP clients and servers:

- **Get command:** It copies a file from the remote host to the local host.
- **Mget command:** It copies multiple files from the remote to the local host.
- **Put command:** It copies a file from the local host to the remote host.
- **Mput command:** It copies multiple files from the local host to the remote host.

6) Disconnect from the remote host

In order to disconnect a connection, the following commands can be used:

- **Close:** It disconnects from the remote host but leaves the FTP client running.
- **Quit / bye:** It disconnects from the remote host and terminates FTP.

FTP Illustration**Example 1**

```
C:\] ftp www.ymcaie.ernet.in
Connected to ymcaie.ernet.in 220
ymcaie FTP server (Version 4.1) ready.
Name: Amit
Password required for Amit.
Password: *****
User Amit logged in.
ftp> put file.txt file1.txt
200 PORT command successful.
150 Opening data connection for file.txt (3453 bytes).
226 Transfer complete.
Local: file.txt remote: file1.txt 3453 bytes received in 0.082 seconds
ftp> close
221 Goodbye.
ftp> quit
```

Example 2

```
C:\] ftp www.ymcaie.ernet.in
Connected to ymcaie.ernet.in 220
ymcaie FTP server (Version 4.1) ready.
Name: Amit
Password required for Amit.
Password: *****
User Amit logged in.
ftp> ls /usr/amit
200 PORT command successful.
150 Opening ASCII mode.
226 Transfer complete.
ftp> close
221 Goodbye.
ftp> quit
```

Anonymous FTP

The first step required for establishing a connection with a remote host is a valid user id and password. However, there are few FTP servers that are available for general public access. Thus, in order to provide access to those public remote servers, a user need not have valid user id. *The user name for such servers is anonymous and password is guest.* Remember, the user access is limited in such cases.

Example 3

```
C:\] ftp www.ymcaie.ernet.in
Connected to ymcaie.ernet.in 220
ymcaie FTP server (Version 4.1) ready.
Name: anonymous
331 guest login OK, send "guest" as password
Password: guest
ftp> get file.txt file1.txt
200 PORT command successful.
150 Opening data connection for file.txt (3453 bytes).
226 Transfer complete.
Remote: file.txt Local: file1.txt 3453 bytes received in 0.072 seconds
150 Opening ASCII mode.
226 Transfer complete.
ftp> close
221 Goodbye.
ftp> quit
```



Check Your Progress 3

- 1) Which command is used in ftp for connection establishment?
 - a) open
 - b) get
 - c) put
 - d) quit

.....
- 2). Which command is used in ftp for transfer of data from local machine to remote machine?
 - a) open
 - b) get
 - c) put
 - d) quit

.....

.....
- 3). How many times is the control connection required in an FTP session?
 - a) Once
 - b) Twice
 - c) Many times
 - d) None of the above.

.....

.....

- 4) How many times is the data connection required in an FTP session?
- Once
 - Twice
 - As many times as required
 - None of the above.
-
-
-

4.6 SUMMARY

This unit provides a complete overview of application layer of TCP/IP. Till now we have studied that there are various kinds of application layer related protocols. Some of them have been discussed in this unit, e.g., DNS, SMTP, TELNET and FTP. The domain name system is a client server based application that maps the domain names in the form of IP addresses. DNS maintains a hierarchical structure in order to store the information about a huge set of domain names. The name space is distributed among DNS servers. Every server is assigned a responsibility for a particular zone. There are 13 root servers located in different geographical regions. Each zone has its own primary server known as zone server. Whenever a user requests for an IP address, the name resolution, through a function called resolver, maps the domain name to the IP address and *vice-versa*.

The SMTP protocol provides a mechanism for providing a user interface and transfers the messages on the network. It has the following components: UA and MTA. The UA provides the interface for composing, reading, replying and forwarding the mail messages. The MTA performs the actual transfer of messages, on the network. MIME is a supplement to SMTP for allowing Non-ASCII format messages like video, audio etc. to be transferred through SMTP. POP and IMAP are extensions of SMTP that allow the messages to be stored into the mail server, so that later on, a user can access the mail server and read his/her mails.

TELNET protocol is a client server based application that provides remote login. FTP is a TCP/IP based application protocol employed for transferring files from one host to another. The basic commands required for connection establishment are: open, site etc. The commands used for data transfer are: *get*, *mget*, *put* and *mput*. The commands used for terminating a connection are: quit and close. The next block of this course covers the concepts of network programming.

4.7 ANSWERS TO CHECK YOUR PROGRESS

Check Your Progress 1

- B
- D
- C

Check Your Progress 2

- B
- C
- A

- 4) POP downloads the mail messages on its client end and those mails are deleted automatically at the mail server end. However, IMAP avoids such deletions.

Check Your Progress 3

- 1) A
- 2) C
- 3) A
- 4) C

4.8 FURTHER READINGS

- 1) Andrew S. Tanenbaum, *Computer Networks*, Third edition.
- 2) Behrouz A. Forouzan, *Data Communications and Networking*, Third edition.
- 3) Douglas E. Comer, *Internetworking with TCP/IP Vol.1: Principles, Protocols, and Architecture* (4th Edition).
- 4) James F. Kurose, *Computer Networking: A Top-Down Approach Featuring the Internet* (3rd Edition).
- 5) Larry L. Peterson, *Computer Networks: A Systems Approach*, 3rd Edition (The Morgan Kaufmann Series in Networking).
- 6) W. Richard Stevens, *The Protocols (TCP/IP Illustrated, Volume 1)*.
- 7) William Stallings, *Data and Computer Communications*, Seventh Edition.



Indira Gandhi National Open University
School of Computer and Information Sciences

BCS-052
NETWORK
PROGRAMMING AND
ADMINISTRATION

Block

2

FUNDAMENTALS OF TCP/IP PROGRAMMING

UNIT 1

TCP/IP Programming Concepts	5
------------------------------------	----------

UNIT 2

Socket Interface	23
-------------------------	-----------

UNIT 3

Socket Programming	46
---------------------------	-----------

PROGRAMME DESIGN COMMITTEE

Prof. Manohar Lal, SOCIS, IGNOU, New Delhi	Prof. Arvind Kalia, Dept. of CS HP University, Shimla	Sh. Akshay Kumar Associate Prof. SOCIS, IGNOU, New Delhi
Prof. H.M Gupta Dept. of Elect. Engg. IIT, Delhi	Prof. Anju Sehgal Gupta SOCIS, IGNOU, New Delhi	Dr. P.K. Mishra, Associate Prof. Dept. of CS, BHU, Varanasi
Prof. M.N Doja, Dept. of CE Jamia Millia, New Delhi	Prof. Sujatha Verma SOS, IGNOU, New Delhi	Sh. P.V. Suresh, Asst. Prof. SOCIS, IGNOU, New Delhi
Prof. C. Pandurangan Dept. of CSE, IIT, Chennai	Prof. V. Sundarapandian IITMK, Trivendrum	Sh. V.V. Subrahmanyam, Asst. Professor SOCIS, IGNOU, New Delhi
Prof. I. Ramesh Babu Dept. of CSE Acharya Nagarjuna University, Nagarjuna Nagar (AP)	Prof. Dharamendra Kumar Dept. of CS, GJU, Hissar	Sh. M.P. Mishra, Asst. Prof. SOCIS, IGNOU, New Delhi
Prof. N.S. Gill Dept. of CS, MDU, Rohtak	Prof. Vikram Singh Dept. of CS, CDLU, Sirsa	Dr. Naveen Kumar, Asst. Prof. SOCIS, IGNOU, New Delhi
	Sh. Shashi Bhushan Associate. Prof. SOCIS, IGNOU, New Delhi	Dr. Subodh Kesharwani, Asst. Prof. SOMS, IGNOU, New Delhi

COURSE CURRICULUM DESIGN COMMITTEE

Prof. Naveen Garg, Professor IIT-Delhi	Dr. D.P Vidyarthi Associate Professor SC&SS, JNU, New Delhi	Sh. M.P. Mishra, Asst. Prof. SOCIS, IGNOU, New Delhi
Sh.Shashi Bhushan Associate Professor SOCIS, IGNOU, New Delhi	Sh. Akshay Kumar Associate Professor SOCIS, IGNOU, New Delhi	Dr. Naveen kumar, Asst. Prof. SOCIS, IGNOU, New Delhi
Ms Seema Gupta Asst. Professor, IP University		

SOCIS FACULTY

Sh. Shashi Bhushan, Director	Sh. Akshay Kumar Associate Professor	Dr. P.V. Suresh Associate Professor
Sh. V.V. Subrahmanyam, Associate Professor	Dr. Naveen Kumar Reader	Sh. M.P. Mishra Asst. Professor
Dr. Sudhansh Sharma Asst. Professor		

PREPARATION TEAM

Dr. D.K. Lobiyal (<i>Content Editor</i>) School of Computer and System Sciences, JNU	Md. Sohrabuddin Dept. of Computer Engineering Jamia Millia Islamia, New Delhi	Language Editors Dr. Malathi Nair & Dr. Nandini Sahu School of Humanities, IGNOU
Shri Amit Goel Dept. of Computer Engineering YMCA Institute of Engineering Faridabad	Shri V.P. Vishwakarma Amity School of Engineering and Technology, New Delhi	

Course Coordinator: Dr. Naveen Kumar, Reader, SOCIS, IGNOU, New Delhi

PRINT PRODUCTION:

Sh. Tilak Raj, S.O.(P), MPDD, IGNOU, New Delhi

July, 2013

© Indira Gandhi National Open University, 2013

ISBN:

All rights reserved. No part of this work may be reproduced in any form, by mimeograph or any form, by mimeograph or any other means, without permission in writing form the Indira Gandhi National Open University.

Further information on the Indira Gandhi National Open University courses may be obtained from the University's office at Maidan Garhi, New Delhi-110068.

Printed and published on behalf of the Indira Gandhi National Open University by the Director, SOCIS.

Laser Composer : Tessa Media & Computers, C-206, A.F.E-II, Jamia Nagar, New Delhi-25

Print:

BLOCK INTRODUCTION

“Socket” or “an end point for communication” which is implemented with the socket interface (C function calls) and used to support client/server communication. The Block 2: **Fundamentals of TCP/IP Programming** provides the necessary details required for a programmer to develop client/server applications in the TCP/IP. Because the BSD socket interface has become a standard hence, we have explained most of the socket system, calls and their characteristics in this block. The purpose of this course is to explain network communication in the context of Unix. The special importance will be given to those system calls concerned with the creation, management, and use of sockets. If you want to see more information about the system calls you can get in the Unix programmer’s manual (man). Also, the fundamental concepts are covered including network addressing, well known services, sockets and ports. This block requires an understanding of the C programming language and Unix.

UNIT 1 TCP/IP PROGRAMMING CONCEPTS

Structure	Page Nos.
1.0 Introduction	5
1.1 Objectives	5
1.2 Client Server Communication	6
1.2.1 Designing Client/Server Programs	7
1.2.2 Socket Concepts	11
1.2.3 IP Address and Ports	12
1.2.4 Byte Ordering	13
1.2.5 Sketch of Networking Connection	13
1.2.6 Active and Passive Sockets	14
1.3 Socket Fundamentals	15
1.4 Networking Example	18
1.5 Summary	20
1.6 Answers to Check Your Progress	20
1.7 Further Readings	22

1.0 INTRODUCTION

We are moving in the world of IT where information plays important role in our daily life. But exchange of information is also becoming important issue in today's world due to which most organisations in India and outside world use a substantial number of computers and communication tools. Computer networks (of an organisation or the Internet) allow the user to access programs and remote databases. The commonly used protocol suit, which provides these facilities, is called as TCP/IP. You might ask yourself, why should I learn about working of TCP/IP protocol or its programming? The answer is simple, as the Internet becomes more popular, more applications are developed. You should understand how TCP/IP these protocols work and also how you can develop your own application and protocols according to your need. At present TCP/IP has become an essential built-in component of most of the operating systems, including Unix, Windows 95, Windows 98, Windows NT, OS/2, and Linux etc. Before we can start writing network programs, we must understand how these systems (Internet or Intranet) actually communicate, which means that we have to learn a little about how the client and server communicate and work. First let's understand the client-server paradigm and next we will go in the details of socket and its structure which are used by network programs for communication.

1.1 OBJECTIVES

Our objective is to introduce you with basic concept of TCP/IP programming. On successful completion of this unit, you should be able to:

- have a reasonable understanding of the TCP/IP network architecture;
- describe the operation of simple client/server architecture;
- understand the role, meaning and structure of sockets;
- describe and understand how to use sockets; and
- demonstrate an understanding how to write client/server programs.

1.2 CLIENT SERVER COMMUNICATION

Computer Networks have revolutionised our lives. We have a lot of examples where we make use of computer network like, when we withdraw money from ATM machines or when we write e-mails or when we make computerized rail reservation (*Figure 1*).

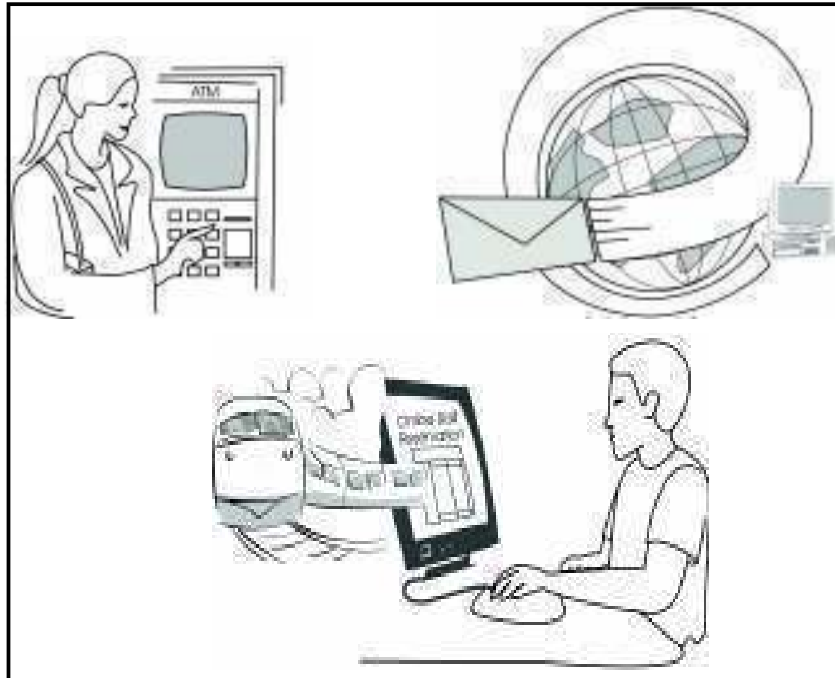


Figure 1: Use of Computer Network

All the above examples are nothing but computers communication. You must be wondering how these computers talk to each other and understand each other also? Because you are computer science students you may be thinking can I make them talk? Of course yes, this block is all about computer communication. But first of all try to understand how computers talk or network communication happens. Most of the computer networks use client/server model (*Figure 2*).

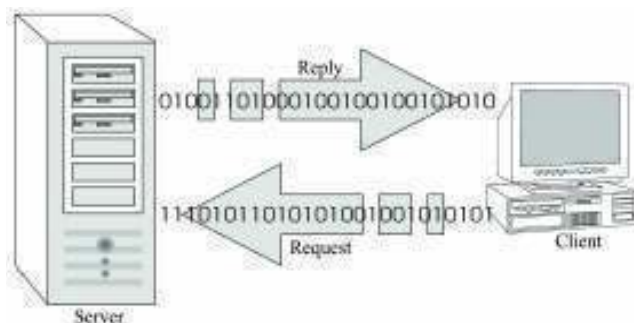


Figure 2: Client/server communications model

Client/server computing means one computer (or more) is acting as client computer one is behaving as serve which preferably of high configuration but can you think why we make a high configuration computer as a server? In simple words, client is defined as requester of services and server is defined as provider of services. Client and Servers are nothing but programs, which generally run on different machines/computers.

When client wants some information or wants to perform some task by server it has to contact the server. Therefore, the client initially needs the address of the server, and tries to contact the server using that address. Server is a program which is always running and waiting to serve clients. Server welcomes the client (if free) as the client contacts it. After contacting the server, the server welcomes that client (if free), the client informs about its own address to servers so that server can reply the client also. When both clients and servers exchange important information to each other connection establishes. Once with connection or link established, both client and server could exchange any information with each other. Lets take an analogy of doctor and patients to understand the client server paradigm in our daily life. Doctor can be considered as a server and patients as clients, patients want to get cured or want some information, so patient (s/he) should know the address of Doctor to whom s/he can contact. We assume doctor is always available for the patient, client inform about himself/herself. Now, doctor knows about patient details and history so they both can contact to each other according to their need.

We can say that a server is a program that is waiting for client so that it can do something for the clients and client program is defined as requester of services. Since server has to complete the task of many clients (may be thousands) that's why it should be of high configuration and have high speed, memory and disk space. Lets summarize the client/server communications.

A server is a process that is able to carry out some functions or services, like transferring files, translating host names to IP addresses, or any other task like finding inverse of a matrix.

- 1) Server is started on some computer system.
- 2) Client is started, either on same server computer or an different computer.
- 3) Client sends a request across the network to the server using server's network address which client should know before contacting.
- 4) Server listens the request and welcomes the client.
- 5) Client tells about its local address to server.
- 6) Connection established.
- 7) Both communicate to each other/servers serve the client like:
 - Print the files for the client,
 - Read or write a file present at the server etc.
- 8) When client's request is over and work is done, server goes back in waiting for other clients.

1.2.1 Designing of Client/Server Programs

Architecture of client and server program involves different issues, according to the system requirements. Here, in this section we have explained you about the characteristics and categories of client and server programs.

a) Characteristics of client program

Here we are explaining you about different characteristics of clients programs so that while writing programs you can always take care of these given characteristics. As you know, client program is an application program that becomes a client temporarily when remote access is needed.

- It is invoked directly by a user, and executes only for 1 session.
- Client program actively initiates contact with a server.

- Client program can access multiple services as needed, but actively contacts one remote server at a time.
- Client program does not require special hardware or sophisticated operating system.

b) Characteristics of server program

Server program is a special-purpose, program dedicated to provide generally one service, but server program can handle multiple remote clients at the same time. Server program has following characters:

- It is invoked automatically when a system boots, and continues to execute through many sessions. (inetd).
- Runs on a shared computer.
- Waits passively for contact request from arbitrary clients.
- Accepts contact from arbitrary clients, but offers a single service.

c) Iterative and concurrent programs

When server knows that client is going to take short amount of time, server program handles the client request one by one in linear manner. Such server are known as interactive or sequential server programs. But think, if server does not know how much time client will take, server may get busy with only one long request and other poor clients will keep waiting. In case server does not know about the time clients will take, servers typically handles it concurrent fashion, and such server is called concurrent server program. Most client software achieves concurrent operation because the underlying Operating System allows users to execute client programs concurrently or because user can execute client software simultaneously. A concurrent server invokes another process to handle each client request so that original server is always free to serve the clients. The program structure of iterative and concurrent server are given in the *Figure3*.

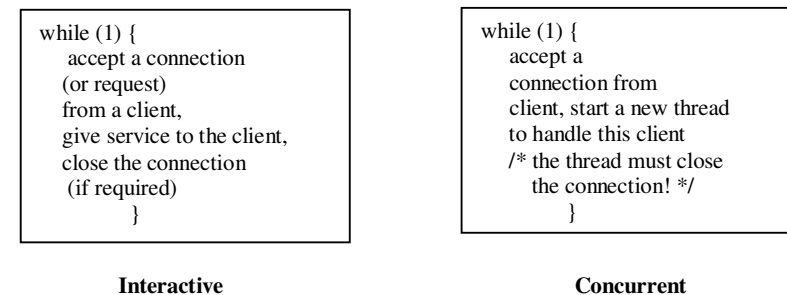


Figure 3: Program structure of interactive and concurrent servers

d) Standard and non-standard programs

Broadly client applications programs are categorised into standard and non-standard application programs. Standard application programs are those that use standard services like file transfer, electronic mail etc. of TCP/IP application layer. The application program which uses the services definitely privately or locally by the organisation according to its need is called non-standard application. For example, a site of private database system of some institute. Remember, standard application services are assigned to well known (universally recognised) port number. Like file transfer protocol is universally assigned 21 port number. Non-standard applications are assigned to locally known (internally or privately recognised) port numbers. The site of Indira Gandhi National Open University is assigned to 3128 port number. System Administrator can assign these services to different port number according to its need and the options given by the authorities. It is important for network programmers to

understand the differences between these standard and non-standard services and their port numbers.

e) **Connection and connectionless programs**

When you design client/server program you must select the type of communication mode. You can either choose connection oriented communication mode or connection less communication mode, which directly refer to the transport layer protocols of TCP/IP. As you know TCP (Transmission Control Protocol) provides connection oriented services and UDP (User Datagram Protocol) provides connectionless services. If TCP is used between client & server, we can say it is connect oriented communication and if UDP then its connectionless communication. Connection oriented and Connection less services are distinguished in the scale of reliability-level. TCP provide reliability in communication between client and server, which means that whatever data is sent, the sender knows data is delivery properly or not. In connection-oriented communication underlying protocol (e.g., TCP) verifies the data delivery by acknowledgement and if not transmitted properly, data is automatically resent. It is also computes a checksum of data packets to provide the guarantee of non-corruption of data. TCP use the sequence numbers to maintain the proper sequencing and ordering of the data.

In contrast to this, connectionless services do not provide assurance about reliable delivery of data (no acknowledgement) or ensure that whether data will reach to the entitled process or not. For example, when client/server send some data to server it may be lost, resend, duplicated, can get corrupted, can be non-sequential or may be received or delivered out of order. But, instead of these problems and disadvantages the connectionless services (UDP) provides best-effort delivery. Retransmission is not performed but application must detect lost data and retransmit if needed. It is having very less overheads because of its simplicity; it provides very high transmission speed even in the wide area networks.

f) **Stateful and Stateless programs**

State information is the information that is maintained by a server about its states (some information between requests) of the ongoing interaction with client programs. If a server program is maintaining the states of ongoing communication it is called as stateful server program. FTP is the stateful server, otherwise it is stateless (like, HTTP server). Maintaining these states of the interaction improve the efficiency of the server. Maintaining small amount of information about the states reduce the size of message that client and server exchange. It is also helps the server to understand on different requests quickly; in addition to that it may also add some security features in the communication. In contrast the stateless server avoids the possibility of wrong or incorrectly reply, because sometime state information can become incorrect, duplicated, delivered out of order or client reboot between the communications.

Generally, server programs are designed into four categories accordingly to their capability of providing concurrency and their use of communication mode. These are summarized here in the *Figure 4*.

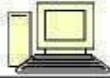
	Iterative	Concurrent
Connectionless	Iterative connectionless	Concurrent Connectionless
Connection-Oriented	Iterative connection-oriented	Concurrent Connection-oriented

Figure 4: Categories of server program

These four categories of server are further explained here:

- i) **Concurrent Connectionless:** It is rarely used server design, where server create a new process to attend each request from the client, this concurrently provides the efficiency in the system but the time to create new process must be considerably less than the time required to compute the request. Generally, in this concurrent connectionless server design cost put for process creation become higher than the efficiency achieved with the concurrency. So, it is uncommonly used.
- ii) **Concurrent Connection-oriented:** Most general design of server, it is suitable for service, which needs reliable transport of data in wide area networks and it can handle multiple clients simultaneously. It further has two different implementations:
 - Concurrent process to handler connections.
 - A single process and asynchronous I/O to handle multiple connections. (Process repeatedly waits for an I/O on any of the connections it has open and handles that request).
- iii) **Iterative Connectionless:** Most commonly used server, this kind of server is suitable for services that need a less amount of processing for each request. Iterative servers are generally stateless, simple and less vulnerable to failures.
- iv) **Iterative Connection-oriented:** It is less commonly used server. It is used for the services that need less amount of processing for each request but the reliability in communication is a necessary. But these kinds of servers waste their time in connection establishment and termination.



Check Your Progress 1

- 1) Why the 'Server Computer' is high speed, memory and disk space?

.....

.....

.....

.....

- 2) Write the difference between interactive and concurrent programs.

.....

.....

.....

.....

- 3) List different categories of services.

.....

.....

.....

.....

1.2.2 Socket Concepts

As we have studied earlier in the Unit 1 that socket is basically a combination of IP address and port number, but socket structure contains many more things. To understand in detail let study the concept of connections and association in this section.

a) Connections and Associations

As you know when client wants to communicate to server it first establishes “Connection”. Here connection is nothing but a communication link between client program and server program. As we have defined in our present section, when connection is established between client and server, they exchange a set of important information that is known as “Association”. Lets see what exactly we mean by association.

The term association is used for the 5-tuple’s that completely specify the two processes that comprise a connection.

```
Association {
    protocol,
    local address,
    local process,
    remote address,
    remote process.
}
```

Let’s understand the ‘Association’ with the help of one example. Machine ‘A’ is local host and wants to establish connection with Machine ‘B’ which is a remote host. Now Machine ‘A’ has to define which protocol it will use to establish connection like TCP or UDP (in-case of TCP/IP). Remote and local address of machines, are like ‘IP address’ of machine in TCP/IP, which actually identify the location of machine in network.

Local Process and Remote Process are used to identify the specific process in system that is involved in establishing connection. For example, **Association** needed for connection in TCP/IP is (UDP, 192.23.15.10, 1211, 192.64.10.12, 2102) where UDP is a protocol. The association with 5-tuple’s is also known as full association but we also have **half association**.

Like:

```
{protocol, local address, local process}
&
{protocol, remote address, remote process}
```

Which define each half of the connection. This half Association is also called as ‘**Socket**’ or ‘**transport address**’. The term Socket’s popularized by Berkley Unix networking system. In the next section we will discuss Socket in detail.

b) What is Socket?

The basic building block for computer communication is the *socket*, which is an end-point of communication. Sockets allow communication between two different processes on the same or different machines. Let’s see the concept of sockets in general term. In term of electricity we use “Electrical socket” which use to get electrical power plug an appliance’s power cord into a socket in the wall. When you want to talk to your friend through telephone, you can connect your telephone with different end-points (Telephone-Socket) available in your home as shown in the

Figure 5. The same concept we can realize when we talk about the Socket for communication between machines.

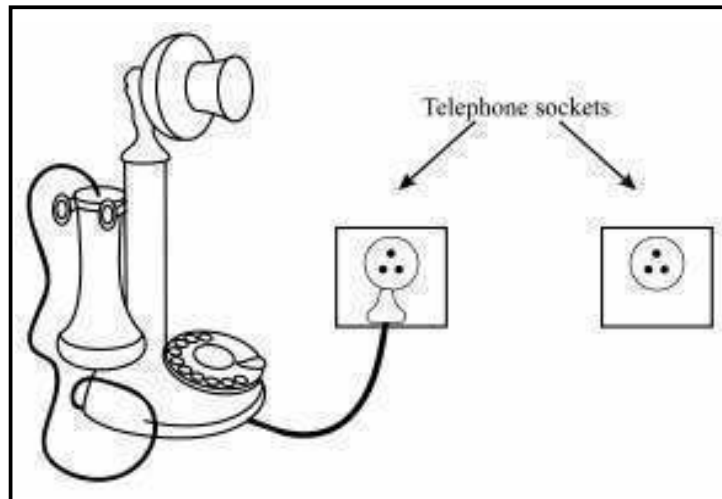


Figure 5: Socket is an end-point of communication

Why is it called a socket? Because of the analogy to plugging a wire into something that can understand what is carried by the wire. When two applications want to talk to each other so, one application tells the OS to open a socket and through socket it uses communication protocol, at receiver side one socket will be ready with the open socket. Once connection is established application can send and receive data.

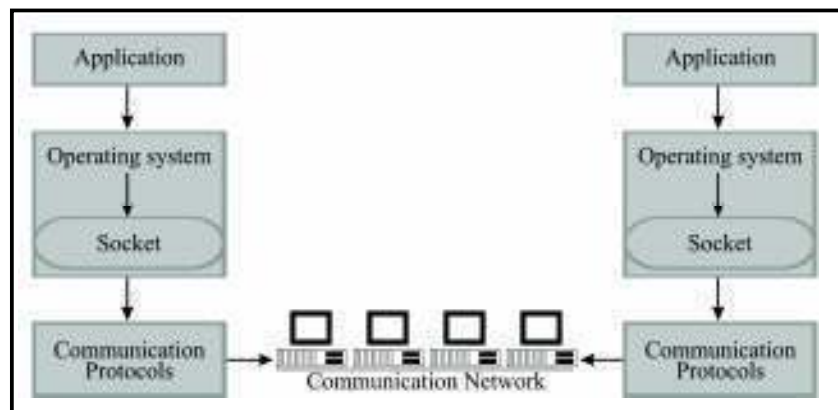


Figure 6: Sockets are basic building block of communication

1.2.3 IP Address and Port

An internet address or IP address is number made of 32 bit or 4 bytes (number between 0 and 255) written in the form of 'dotted notation' for example "192.168.10.10" **is a valid IP address** usually one computer has only one IP address but a computer might have more than one such address, if computer has more than one NIC (Network Interface Card) connected to internet. Therefore, when we mean host we should understand, IP address that identifies a host Interface not a host but usually we mean it to host.

When we talk about Network communication we say IP address identifies the host computer. However, usually more than one application program running on a computer use the network. Because of this reason TCP/IP use another level of address called port number, which identify the process running on computer.

Let's see an example, you want to call me, you know the IGNOU phone number is 29536207 but in IGNOU we have many teachers that is why you have to use my extension 2229, here in analogy to network communication, telephone number is IP address and extension number is as Port Number. Explain number is used, as another level of address is to resolve among teachers in IGNOU.

In networking, communication address is made up of an IP address and a Port Number. Port Number is a 16-bit identifier and each host has 65536 ports. In TCP/IP some ports are reserved for specific application for example, port number 21 is reserved for FTP, Port number 23 is for telnet, 80 for HTTP.

Name	Port Range	Use
The Well Known Ports	0 to 1023	Reserved for common services like telnet, ftp.
The Registered Ports	1024 to 49151	Registered for companies and organisation
The Dynamic and/or Private Ports	49152 to 65535	Available for the rest of world to use.

1.2.4 Byte Ordering

Some computers are “big endian”. This refers to the representation of objects such as integers within a word. In big endian machine, the high byte of an integer is stored in low byte address. In little endian, the high byte of an integer is stored in high byte address as shown in *Figure 7*. A Sun Sparc is big endian. So the number $2 + 3 * 256$ would be stored as:

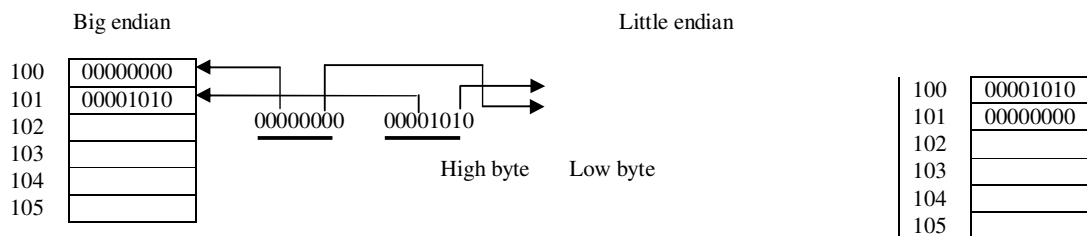


Figure 7: Data representation in big and little endian

		3	2
3	2	1	0

A “little endian” machine stores them the other way. The 386 is little endian.

2	3		
3	2	1	0

If a Sparc sends an integer to a 386, what happens? The 386 sees a data “2 + 3 * 256” as “2 * 16777216 + 3 * 65536”.

1.2.5 Sketch of Networking Connection

The client/server model is used to divide the work of networks into two parts. One part knows how to do a certain task or how to provide certain service, this part is known as server program. Other part defines how to talk to user and how to connect to server program, known as client program. Server may give service to many different clients either simultaneously (concurrent services) or one after the other (Iterative services). On the other hand client talks to single user at a time. In network

program we can have different possibilities depending on the needs. But first of all we will discuss only simple one. Let's see the following example where one client talks to one server. The main feature of client is to provide convenient user interface, hiding the details of how the server communicates from the user. The client needs to first establish a connection with server, on given address. Generally, client does the following steps:

- 1) Create a socket.
- 2) Specify the address and service port of the server program.
- 3) Establish the connection with the server.
- 4) Send and receive information.
- 5) Close the socket when finished, terminating the conversation.

To establish and complete a connection the server-side would follow these steps:

- 1) Create a socket.
- 2) Listen for incoming connection requests from clients.
- 3) Accept the client connection.
- 4) Send and receive information.
- 5) Close the socket when finished, terminating the conversation.

1.2.6 Active and Passive Socket

The client application always starts the communication. It creates a socket and actively attempts to connect to the server program. This is known as active open or active socket. On the other hand, the server application creates a socket and passively listens for incoming connection requests from clients. This socket is passive open, that's why this socket is called as passive socket. This is also same as the telephone system where one phone is always waiting for incoming calls.

When client initiates a connection, the server notices that same process is trying to connect with it. By accepting the connection the server establishes a logical communication path between two programs. (Generally, in practical situation, the act of accepting a connection creates a new socket. The original socket remains unchanged so that it can continue its availability to clients for additional connections when the server no longer wishes to listen for connection).

Check Your Progress 2

- 1) What are the types in association?
.....
.....
.....
- 2) What is a Socket?
.....
.....
.....
- 3) Write the difference between active and passive socket.
.....
.....
.....

1.3 SOCKET FUNDAMENTALS

To standardize network programming, Application Programs Interface's (API's) have been developed. An API is a set of declarations, definitions and procedure's followed by programmers to write clients server programs. For network programs commonly developed API's are Socket Interfaces, Transport Level Interface (TLI), Stream Interface, Thread Interface and Remote Procedural Call (RPC). Let's discuss socket interface in details but first see the data types defined in socket interface.

There are `u_char` `u_short` and `u_long` data type.

```
u_char    // character unsigned 8 bit
u_short   // short integer unsigned 16 bit
u_long    // long integer unsigned 32 bit integers
```

From application point of view, the differences between network protocols are address schemes used. For TCP/IP, an ideal API would be one that understand IP addressing and port numbers. Since the socket library is designed to be used for multiple protocols, addresses are referenced by a generic structure as follows:

```
struct sockaddr {
u_char  sin_len;      /* length of structure */
u_short sa_family;    /* address family: AF_XXX value */
char sa_data[14];     /* up to 14 byte of protocol specific address*/
};
```

Originally `sa_len` was not there in the structure. The `sa_family` field specifies the type of protocol. But in case of TCP/IP, this field is always set to `AF_INET` (where `INET` stand for Internet). The remaining 14 bytes (`sa_data`) of this structure are always protocol dependent for TCP/IP, IP addresses and port numbers are placed in this field. The contents of the 14 bytes of protocol dependent addresses are interpreted according to the type of address. (In TCP/IP this is defined in `<netinet/in.h>` header file.)

Internet socket address structure: The structure `sockaddr_in` describing an address in the Internet domain is defined in the file `<netinet/in.h>`. The application program that use the TCP/IP need a specific type of socket address structure, which can hold IP address, port number and protocol family. This structure is called as `sockaddr_in`., which have five fields.

```
struct sockaddr_in
{
u_char    sin_len;      /* length of structure */
u_short   sin_family;   /* for Internet it is AF_INET */
u_short   sin_port;     /* 16-bit port number */
struct in_addr sin_addr; /* 32-bit address */
char      sin_zero[8]; /* unused, reserved for future */
};
```

In this structure first and last field are normally not used. A `sockaddr_in` structure contains an `in_addr` structure as a member field. `In_addr` has the following structure.

```
struct in_addr {  
    u_long  s_addr;    /* 32-bit address */  
};
```

Note one thing here about the **sock_addr** & **sockaddr_in** structures; these both structures are compatible with each other. Both structures are 16 bytes in size and first two bytes of both structures are the family field. Thus, struct `sockaddr_in` can always be cast to struct `sock_addr`. It is important to remember that these fields always need to be set and interpreted in network byte order (this topic we will discuss in unit 3 of this Block).

A socket is defined in Operating System as structure, which has five fields. (Remember the five tuple's in association, same here in the structure of socket). Let's see these fields of structure in the *Figure 8*:

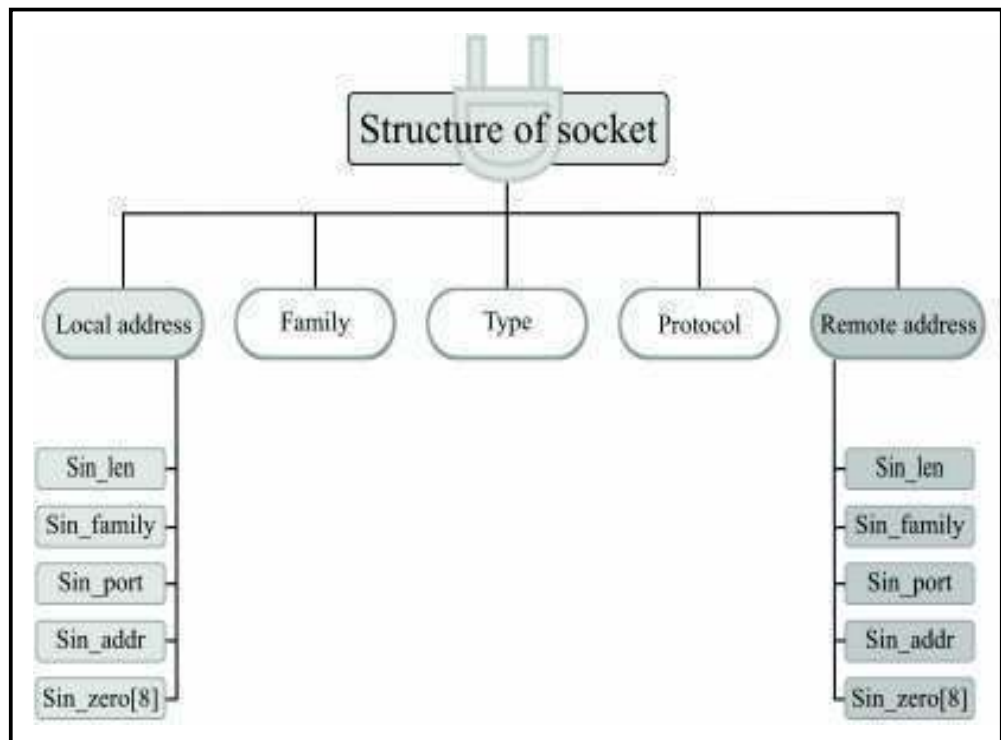


Figure 8: Structure of socket

- 1) **Family:** An application program specifying a socket family defines the protocol group needed for communication. It can be Unix domain protocols, Internet domain protocol, Xerox NS protocols and IMP link layer (IMP is Interface Messaging Protocol) as given below. In case of TCP/IP this field is `AF_INET`.
- 2) **Type:** An application program specifying a socket type, which indicates the desired communication style for the socket. This field defines the type of socket like stream socket, data-gram socket and raw socket. Each socket has a type associated with it. The socket type determines the socket communication properties like reliability, ordering, and prevention of duplication of messages. The basic set of socket types is defined in the `sys/socket.h` header file. Total we

have five types of Socket options available. Let's discuss in details what are these types of sockets:

- i) **Stream Socket:** Stream socket is used with connection oriented protocol. It provides stream model of communication. A *stream* socket provides for the bi-directional, reliable, sequenced, and unduplicated flow of data without record boundaries. This service is reliable and connection oriented. Transport protocols support this socket type and ensure that the delivery is in-order, without loss or error or duplication, if not, then abort the connection and report the error to the user. Message boundaries are not present in stream sockets. It can be used in both Unix and Internet domain. In case of Internet domain it uses TCP, which is connection-oriented protocol.
 - ii) **Datagram Socket:** Datagram sockets are used with connection-less protocols. It provides datagram model of communication. It supports bi-directional flow of data, between sender and receiver and data is not promised to be in sequence, reliable, or unduplicated. That's why in case of datagram socket, a process receiving messages on a datagram socket may find messages duplicated, and, possibly, in an order different from the order in which it was sent. An important characteristic of a datagram socket is that record boundaries in data are preserved. This type of socket is generally used for short messages, such as a name server or timeserver, since the order and reliability of message delivery is not guaranteed.
 - iii) **Raw Socket:** Raw Socket, as its name indicates, provides access to internal network protocols and interfaces. A raw socket allows an application direct access to lower-level communication protocols (like, IP layer in case of TCP/IP). Raw sockets are developed for advanced programmers who want to build new protocols or who are interested to use the low level feature, that are not directly accessible through a normal interface. Raw sockets are normally datagram-based, though their exact characteristics are dependent on the interface provided by the protocol.
 - iv) **Sequenced Packet Socket:** It provides sequenced, reliable, and unduplicated flow of information. A sequenced packet socket is similar to a stream socket, with the exception that record boundaries are preserved. This interface is provided only as part of the NS socket abstraction, and is very important in most serious NS applications.
 - v) **Reliably Delivered Message Socket:** The reliably delivered message socket has similar properties to a datagram socket, but with reliable delivery. There is currently no support for this type of socket, but a reliably delivered message protocol similar to xerox's Packet Exchange Protocol (PEX) may be simulated at the user level.
- 3) **Protocol:** The protocol specifies a particular protocol to be used with the socket. Normally only a single protocol exists to support a particular socket type within a given protocol family. However, it is possible that many protocols may exist; in that case a particular protocol must be specified.
 - 4) **Local Socket Address:** A pointer to a buffer of type struct sockaddr_in that contains the local address to which the socket is to be bound.
 - 5) **Remote Socket Address:** A pointer to a buffer of type struct sockaddr_in that contains the remote address to which the socket is to be bound.

Now let's summarise the characteristics of socket:

- A socket is referenced by an integer. That integer is called a socket descriptor.
- A socket exists as long as the process maintains an open link to the socket.

- You can name a socket and use it to communicate with other sockets.
- Sockets perform the communication when the server accepts connections from them, or when it exchanges messages with them.
- You can create sockets in pairs (only for sockets in the AF_UNIX address family).

1.4 NETWORKING EXAMPLE

As we have discussed earlier that client program can use the connection oriented reliable service or connection less unreliable services. So you should know the algorithms or steps for these client program.

Connection oriented and Concurrent client

Here we have given you an algorithm, where client opens the socket, sends a request and gets the response.

Step 1: Identify the IP address and port No. of a server.

Step 2: Open a socket.

Step 3: Choose TCP protocol (a connection oriented protocol) and an unused protocol port.

Step 4: Connect Socket to the server.

Step 5: Using application level protocol, communicate with server.

Step 6: Close the connection, when the job is over.

Connectionless Client

Step 1: Same as step 1 in connection oriented client

Step 2: Same as step 2 in connection oriented client

Step 3: Choose UDP Protocol (a connectionless protocol)

Step 4: Inform the server about the client address so that server can communicate the client.

Step 5: Same as step 5 connection oriented client

Step 6: Same as step 6 connection oriented client

Now you learnt the client program algorithm, but without server these are not useful so you must learn how to design different kinds of server programs. As we have early classified them into 4 types in the section, let's make there algorithms one by one.

Iterative and Connection oriented server

Step 1: Create a socket, and associate it with the well-known address (which is known to the client).

Step 2: Make the socket in passive mode.

Step 3: Welcome the connection request from the client, and create a new socket for the connection

Step 4: Read request from client repeatedly and give them reply according to application protocol.

Step 5: When the job of particular client is over close that connect and rather step 3.

Here at a time, a single process handles one connection from client, only one when it finish its request than only it accepts next connect request.

Iterative and connectionless server

- Step 1: Same as step 1 iterative and communication Server
- Step 2: Same as step 2 iterative and communication Server
- Step 3: Repeatedly read the next request from the client and send the reply according to the protocol.

Connectionless and concurrent server

- Master 1:** Create socket and fix it to some address (well known to client) offered
Socket must be left unconnected.
- Master 2:** Receive next request from client and create copy of the same (child process).
- Slave 1:** Receive the specific request
- Slave 2:** Reply accordingly to application protocol and give response to client
- Slave 3:** Terminate the child process after completion of the task.

Connection oriented concurrent server

- Master 1:** Create socket and fix it to the address, which is known to the client.
Socket must be unconnected.
- Master 2:** Make the socket as passive socket
- Master 3:** Continuously receive the connection request from client and create slave server process to handle the communication with client.



Check Your Progress 3

- 1) What are the data types defined by the socket interface?
.....
.....
.....
.....
.....
- 2) Write the structure of socket address.
.....
.....
.....
.....
.....
- 3) Write the algorithm for interactive connection oriented server.
.....
.....
.....
.....
.....

1.5 SUMMARY

In this unit we have discussed different issues related to client server communication so that you will have good understanding of different servers and their services like connection oriented and connectionless. The procedure of basic client and server was also explained in detail. After completing the theoretical conceptualization of client server communication we have defined the socket, which is considered as an end point of communication in network programming. The different components related to the sockets like, IP address, port, address family and socket types are also explained in this unit. The concept of byte ordering and its use in networking is explained briefly with the help of an example which you will again study in detail during the unit socket programming. Further the differences between Active and Passive Socket was explained and at the end of the unit we have given you some algorithms and procedure of different client server communication. In the next unit *Socket Interface* we have given basic socket calls, which will provide you the knowledge of the functions and their characteristics.

1.6 ANSWERS TO CHECK YOUR PROGRESS

Check Your Progress 1

- 1) Server is a program that is waiting for client so that server can do something for the client and client program is defined as requester of services. Because server has to complete the task of many clients (may be thousands) that's why it should be of high configuration and have high speed, memory and disk space.
- 2) When server knows that client is going to take short amount of time then server program handles the client request one by one in linear manner, which is known as interactive or sequential server. In this, if server don't know how much time client will take, server will get busy with one request itself and other poor clients will wait. A concurrent server invokes another process to handle each client request so that original server can always be free to serve client. That's why we can say practically it is more efficient.
- 3) Server programs are designed into four categories accordingly to their capability of providing concurrency and their use of communication mode.

These are following:

- 1) Concurrent Connectionless
- 2) Concurrent Connection-oriented.
- 3) Interactive Connectionless
- 4) Interactive Connection-oriented

Check Your Progress 2

- 1) The term association is used for the 5-tuple's that completely specify the two processes that comprise a connection.

```
Association {  
    Protocol,  
    local Address,  
    local process ,  
    remote address,  
    remote process.  
}
```


- 2) The basic building block for computer communication is the *socket*, which is an end-point of communication. Sockets allow communication between two different processes on the same or different machines.
- 3) The client application always starts the communication. It creates a socket and actively attempts to connect a server program. This is known as active open or active socket. On the other hand the server application creates a socket and passively listens for incoming connection from clients that is passive open situation. That's why this socket is called as passive socket.

Check Your Progress 3

- 1) The main data types defined in this socket interface are `u_char`, `u_shorts` and `u_long` data type, its meaning are defined below:
 - `u_char` : character unsigned 8 bit
 - `u_shorts` : short integer unsigned 16 bit
 - `u_long` : long integer unsigned 32 bit integers
- 2) The general socket address is given below:

```
struct sockaddr {
    u_char  sin_len;    /* length of structure */

    u_short sa_family; /* address family: AF_XXX value */

    char sa_data[14]; /* up to 14 byte of protocol specific address*/
}
```

Where `sa_len` indicates the length of the address structure. The `sa_family` field specifies the type of protocol. The remaining 14 bytes (`sa_data`) of this structure are always protocol dependent for TCP/IP, IP addresses and port numbers are placed in this field. The contents of the 14 bytes of protocol dependent addresses are interpreted according to type of address.

- 3) The algorithm for Interactive and Connection oriented server is as given below:
 - Step 1: Create a socket, and associate it with the well-known address (which is known to the client).
 - Step 2: Make the socket in passive mode.
 - Step 3: Welcome the connection request from the client, and create a new socket for the connection.
 - Step 4: Read request from client repeatedly and give them reply according to application protocol.
 - Step 5: When the job of particular client is over close that connect and rather step 3.

Here at a time, a single process handles one connection from client, only one when it finishes its request than only it accepts next connect request.

1.7 FURTHER READINGS

- 1) Andrew S. Tanenbaum, *Computer Networks*, Third edition.
- 2) Behrouz A. Forouzan, *Data Communications and Networking*, Third edition.
- 3) Douglas E. Comer, *Internetworking with TCP/IP Vol.1: Principles, Protocols, and Architecture* (4th Edition).
- 4) William Stallings, *Data and Computer Communications*, Seventh Edition.
- 5) W. Richard Stevens, “*UNIX Network Programming*”, Prentice Hall.

UNIT 2 SOCKET INTERFACE

Structure	Page Nos.
2.0 Introduction	23
2.1 Objectives	23
2.2 Elementary Socket System Calls	23
2.2.1 Socket System Call	24
2.2.2 Bind System Call	27
2.2.3 Connect System Call	28
2.2.4 Listen System Call	30
2.2.5 Accept System Call	31
2.3 Elementary Data Transfer Calls	34
2.4 Closing a Socket	38
2.5 TCP and UDP Architectures	39
2.6 Networking Example	41
2.7 Summary	43
2.8 Answers to Check Your Progress	44
2.9 Further Readings	45

2.0 INTRODUCTION

Socket is considered as an end-point, which is used by a process for bi-directional communication in which another socket is associated with another process. As we discussed earlier a socket is like, a file descriptor available in Unix for file communication (I/O) like, open (), create (), close (), read () and write (). Similarly for network communication (I/O) socket behaves like, socket descriptor by which we can read () write () data. File is like a sequence of characters which we can read using repeated read operation in connection oriented mode and in connectionless mode we have to get whole message in a single read operation. But because of the complexity and the requirements of network communication we need many more system calls that we will discuss in this unit.

2.1 OBJECTIVES

Our objective is to introduce you, with the elementary socket calls. After successful completion of this unit, you should be able to:

- have a reasonable understanding of the Elementary System Calls;
- understand the use of basic data transfer calls;
- describe the TCP and UDP Client/Server Architecture; and
- demonstrate an understanding the simple network programs.

2.2 ELEMENTARY SOCKET SYSTEM CALLS

As we discussed in unit 1 when a socket is created, it is allocated a slot in the file descriptor table exactly the same way a file is. Once this socket is associated with a file, we can do I/O such as read and write. Recall that generally sockets are for network communication and network I/O deals with a completely different set of problems than file I/O, but wherever possible, actions corresponding to file I/O are

accomplished. But what is the difference between file I/O and networking I/O? let's find out:

- The client-server relationship is not symmetrical. The client and server processes must be assigned the responsibilities.
- Connectionless protocols do not have an open command because any operation could come from any process.
- Peer process names and file names are important for authority uses.
- More parameters must be specified (protocol, local address, local port server address, server process).
- All computers do not share the same data format (e.g., little endian versus big endian). For details see unit 3.

We now describe the elementary system calls which require to develop Network programs.

2.2.1 Socket System Call

In the early 1980s, with ARPA project, University of California at Berkeley had the responsibility to transport the TCP/IP protocol suit to Unix operating system. It was decided to use Unix system calls with addition to new system calls, if required, as the result new socket interface developed, which become popular as Berkeley UNIX or BSD (Berkeley Software Distribution) version 4.1. We are going to discuss BSD Unix system calls with you in this section.

The socket system calls is used by any process to create a socket for doing any network I/O. The structure of socket, we have already discussed as general but here we will discuss in detail with programming concept. The structure of this is given below.

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

```
int socket (into family, int type, int protocol );
```

Condition

Returns

Successfully	Small integer value called the socket descriptor
Unsuccessful	Error (-1)

Socket function creates a socket, it sets values for only family, type and protocol of socket structure the other fields are set by other functions or by operating system. If socket system call creates socket successfully then it returns small integer value called the socket descriptor sockfd (which is similar to a file descriptor) which uniquely identifies the socket. If socket is not created it means there is an error and it returns -1. This socket descriptor sockfd, is used by other functions to refer the socket. (see the socket description tables and its contents in the *Figure1*).

Whenever you will use socket system call you should include both of these header files sys/types.

#include<sys/types.h>: This header file contains definitions of a number of data types used in system calls.

#include<sys/socket.h>: The header file socket.h includes a number of definitions of structures needed for sockets.

Socket header files contain data definitions, structures, constants, macros, and options used by socket subroutines. An application program must include appropriate header files to make use of structures or other information a particular socket subroutine requires. Some other header files are also important for Unix socket programmers: These are given in *Table 1*. You should remember that header files required for some particular calls or API may differ from system to system. So you should always check it with the online manual available in Unix.

Table1: Socket Header Files

Socket Header File	Description
inet.h	takes the place of both in.h and inet.h header files. It defines values common to the Internet.
ip.h	defines values used by the Inter-network Protocol (IP) and details the format of an IP header and options associated with IP.
netdb.h	defines the structures used by the “get” services.
errno.h	defines the errors that can be returned by the socket library when requests are made to it.
sockcfg.h	describes the socket configuration structure
socket.h	defines most of the variables and structures required to interface properly to the socket library.
sockvar.h	is needed when compiling the socket configuration file
tcp.h	describes those options that may be set for a socket of type SOCK_STREAM.
uio.h	describes the structures necessary to use vectored buffering of the socket library.

Socket Descriptor

As you know, in Unix, if some application needs to perform input/output function, it calls the “open” function to create the file descriptor which has further access to the file. Unix uses descriptor as an index into process descriptor table, which follows the pointers to the data structure that holds all details about file. Similarly, when some application calls “socket” the operating system allocates a new data structure as shown in Figure 1(a) and 1(b) to hold the details required for communication and enter the pointer into the description table.

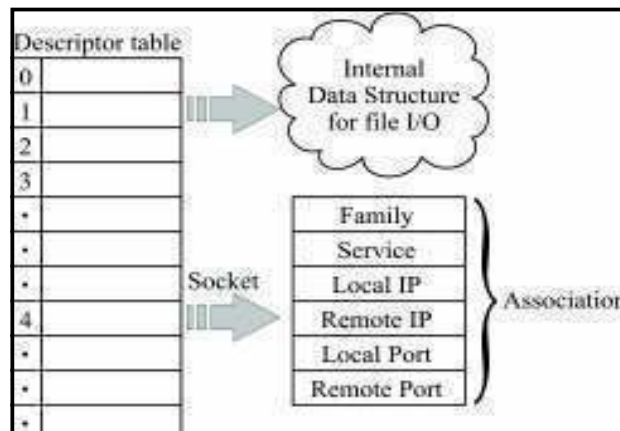


Figure 1 (a)

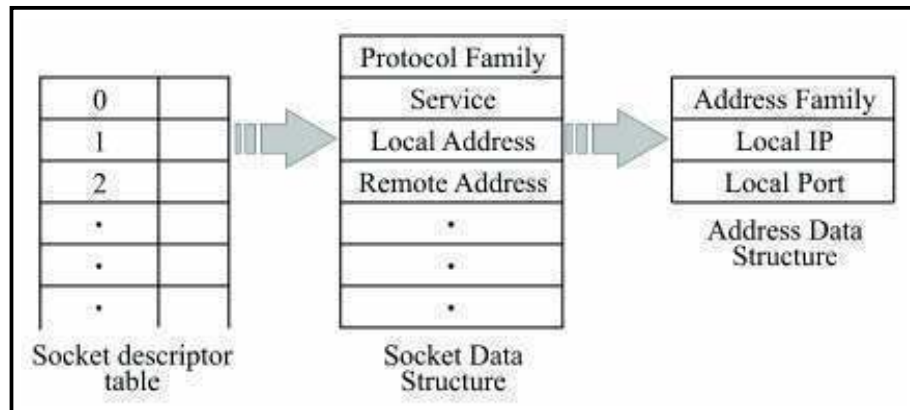


Figure 1 (b)

Figure 1: Socket Descriptor

Socket descriptor defines an address family to provide the freedom to different protocol families for choosing their own representation for their address. A single protocol family can use more than one address families to define address representation. TCP/IP protocol family uses a single address representation; this address family is symbolically represented by AF_INET (Address family internet). *Don't get confused with the PF-NET, both denote the address family in TCP/IP.*

Socket family: As you know socket family defines the protocol group needed for communication. At the time of programming you should choose one of the given options but because we are concerned about TCP/IP you need to remember AF_INET.

```
AF_UNIX /*Unix internal protocols */
AF_INET /* internet protocols */
AF_NS /* Xerox NS protocols */
AF_IMPLINK /* IMP link layer */
```

Socket types: The Type parameter in socket system call specifies the semantics of communication. Sockets are typed according to the communication properties visible to a user. Mostly Processes can communicate only between sockets of the similar type. If underlying communication protocols support, communication between different types of sockets can happen. As we have discussed earlier also you have following options available with “socket”,

/*Standard socket types available*/

```
#define SOCK_STREAM /* stream socket */
#define SOCK_DGRAM /*datagram socket*/
#define SOCK_RAW /*raw socket*/
#define SOCK_SEQPACKET /*sequence packet socket */
#define SOCK_RDM /*reliably-delivered message*/
```

Protocol: The protocol specifies a particular protocol to be used with the socket. Normally only a single protocol exists to support a particular socket type within a given protocol family.

Protocol is dependent on the services we are using in our communication. For example, if we are using Stream Socket type we must use a protocol that provide a connection-oriented service, like, TCP. But if we are using Datagram Socket type

which provide connectionless services then available protocol is UDP. Please see the Table 2 given below):

Table 2: Protocol corresponding to socket type and socket family

	AF_UNIX	AF_INET	AF_NS
SOCK_STREAM	YES	TCP	SPP
SOCK_DGRAM	YES	UDP	IDP
SOCK_RAW	NOT DEFINED	IP	YES
SOCK_RDM	NOT DEFINED	NOT DEFINED	SPP

All combinations of socket family and type are not valid. The *Table 2* (reference from Unix Network Programming by Richard Stevens) server shows the valid combinations along with the actual protocol that is selected by pair.

To create a TCP socket:

```
int sockfd;  
sockfd = socket (AF_INET,  
SOCK_STREAM, 0);
```

To create a UDP socket:

```
int Sockfd;  
sockfd = socket(AF_INET,  
SOCK_DGRAM, 0);
```

2.2.2 Bind System Call

After creating “socket” and before Sending/Receiving data using socket, it must be associated with a local port and a network interface address. That’s why we can say mapping of a socket to a port number and IP address is called a “binding”. Why binding is needed, as you know server use socket so it can attend client request on specific port. Socket must be associated with particular port on one network interface. But think when we can have multiple network addresses on one host and each network, we have some port addresses. In this case with the help of bind we can define, for which network address (IP address + port address) with which port is associated, otherwise imagine how much confusion can happen.

```
#include <sys/types.h>  
#include <sys/socket.h>  
  
int bind(int sockfd, struct sockaddr_in *localaddr, int addrlen);
```

Here in the above prototype of bind, sockfd is File descriptor of local socket, as created by the socket function. Localaddr is a pointer to protocol address structure of local socket. The special address ANY_ADDR can be used to allow the connection to be made on any of the host’s interfaces and addrlen is indicating length in bytes of structure referenced by address. On success, bind () returns a zero. On failure, it returns -1 with an error number.

Use of Bind

During networking programming we find that there are three user of Bind system call.

- A specific network address and port address can be registered to a client.
- Server needs to register a specific network & port address with its socket, basically it informs the system that “The address associated with me is this, and if you get any message on this address just transfer it to me”.

- In case of connectionless client (see your unit 1 of block 1 to know about connectionless and connection oriented services), it needs to assure that system has provided some valid unique addresses, so that when server sends some message it will reach the desired client. This approach of client is same like us, when we write letter we always check whether we have written own address on that envelop or not so that we can get reply on that address.

The bind system call fills the two tuple of association, Local address and Local process elements. When we will use Bind function the client needs to call socket system call because it needs to use the returned value as socket descriptor.

When binding is over, then in-case of UDP socket it is ready to send and receive datagram's. For TCP sockets, the socket is ready to connect or accept calls. Let's see what are these accept and connect call.

Let's have an example:

```
#include <sys/types.h>
#include <sys/socket.h>
#include <string.h>
#define CLIENTPORT 8090

main()
{
    int sockfd;
    struct sockaddr_in local_addr;

    sockfd = socket(AF_INET, SOCK_STREAM, 0);
    client_addr.sin_family = AF_INET; /* host byte order */
    client_addr.sin_port = htons(CLIENTPORT); /* short, network byte order */
    client_addr.sin_addr.s_addr = inet_addr("192.10.10.10");
    bzero(&(client_addr.sin_zero), 8); /* zero the rest of the struct */

    /* error checking for bind(): */
    bind(sockfd, (struct sockaddr *)&client_addr, sizeof(struct sockaddr));
    .
    .
    .
}
```

In socket programming we have different methods which may be useful to you, when you want to choose an unused port at random you can write `/* choose an unused port at random */`

and for choosing IP address you can use `/* use client IP address */`

2.2.3 Connect System Call

Generally “connect” function is used by client program to establish a connection to a remote machine (server).

Syntax of connect is given below:

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

int connect(int sockfd, struct sockaddr_in *server_addr, int serv_addrlen);

connect() returns 0 if successful or returns -1 on unsuccessful and sets errno
```


In the connect call, sockfd is a socket file descriptor returned by socket(), server_addr points to a structure with *destination* port and IP address and serv_addrlen set to sizeof (struct sockaddr).

Initial code necessary for a TCP client:

```
#include <string.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#define SERVER_IP "190.20.10.12"
#define SERVER_PORT 1729

main()
{
    int sockfd;
    struct sockaddr_in ser_addr; /* will hold the destination addr */

    if ((sockfd = socket(AF_INET, SOCK_STREAM, 0)) < 0)
    {
        perror("socket error ");
        exit(1);
    }

    Ser_addr.sin_family = AF_INET; /* inet address family */
    Ser_addr.sin_port = htons(SERVER_PORT); /* destination port */
    Ser_addr.sin_addr.s_addr = inet_addr(SERVER_IP); /* destination IP address */
    memset(&(dest_addr.sin_zero), '\0', 8); /* zero the rest of the struct */
    /* In function void *memset(void *s, int c, size_t n); The memset() function
    fills the first n bytes of the memory area pointed to be s with the constant byte c.*/

    if (connect(sockfd, (struct sockaddr *)&dest_addr, sizeof(struct sockaddr)) < 0)
    {
        perror("connect error");
        exit(1);
    }
}
```

During the bind call example earlier we have ignored the local port number, now we are concerned about the remote port. The kernel will choose a local port, and the site we connect to, will automatically get this information from client.

If connectionless client (UDP) use the connect (), then the system call just stores the server address specified by the process, so that the system knows where to send any future data the process writes to the sockfd descriptor. Also, the socket will receive only datagrams from this address. Note that the destination address is not necessary for each Datagram sent, if we are connecting a UDP socket.

Connect system call result in actual connection establishment between the local and remote system in case of connection-oriented protocol. At this point some agreement for the future data exchange happens like buffer size, amount of data, acknowledgement etc., as we have discussed earlier in Unit 3 of Block 1. This exchange of information between client and server are known as 3-way handshake,

To use connect function, first of all client needs to call the socket (), then connect () function sets value of server socket address and client socket address is either provided by bind system call or set by the operating system.

2.2.4 Listen System Call

TCP server, binds to a well-known port and waits for a client to try to connect to it. This process is called as “listening” and it is performed by calling the listen () system call.

The Listen system call is used in the server in the case of connection-oriented communication to prepare a socket to accept messages from clients. It creates a passive socket (recall the concept of passive and active mode of socket) from an unconnected socket. ‘Listen’ initializes a queue for waiting connections. Before calling listen, socket must be created and its address field must be set.

Usually Listen() is executed after both socket() and bind() calls and before accept() [accept system call will be discussed in next sections].

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

int listen(int sockfd, int Max_conn)

On success, listen() returns “0”
On failure, it returns “-1” and the error is in errno.
```

Listen takes two parameters, first the socket you would like to listen on and another is the Maximum number of request that will be accepted on the socket at any given time. Socket you would like to listen on is represented here as **sockfd** is the usual socket file descriptor from the socket() system call. Maximum number of connections allowed on the incoming queue at any given time is represented by **backlog**. On the server all incoming connections requests from clients come and wait in one special **queue** (from this queue server choose the connection and accept the request). We have to define the limit on queue that how many connections can wait in a queue. Generally this limit is set to 20. But you can set it with any other number also, for example,

listen (socket, 5), call will inform the operating system that server can only allow five client sockets to connect at any one given time.

If the queue is full, then the new connection request will be rejected, which will result in an error in the client application. Queued connections are removed from the queue and completed with the accept() function, which also creates a new socket for communicating with the client.

Example:

```
#include <string.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>

#define SERVERPORT 1729

int main()
{
    int sockfd;
    struct sockaddr_in ser_addr;

    if ( (sockfd = socket(AF_INET, SOCK_STREAM, 0)) < 0)
    {
        perror("socket error");
    }
}
```

```

    exit(1);
}

ser_addr.sin_family    = AF_INET;
ser_addr.sin_port      = htons(SERVERPORT);
ser_addr.sin_addr.s_addr = inet_addr(INADDR_ANY);
memset(&(ser_addr.sin_zero), '\0', 8); /* zero the rest of the struct */

if ( bind(sockfd, (struct sockaddr *)& ser_addr, sizeof(struct sockaddr)) < 0 )
{
    perror("bind");
    exit(1);
}
if ( listen(sockfd, 5) < 0 ) /* Inform the operating system that server can allow
only 5 clients*/
{
    perror("listen error");
    exit(1);
}
.
.
.
}

```

2.2.5 Accept () System Call

After listen system call accept() completes the process of establishing connection between the server and the client. Remember accept system call is used for stream sockets server like TCP server. It removes the first waiting connection request from the queue of pending connections, creates a new socket with the same properties of old socket (which you specified in accept call) and allocates a new file descriptor for the socket. But think what will happen if there is no pending connection waiting in the queue?

In this case we can have two possibilities one *socket is marked as non-blocking* another *socket is not marked as non-blocking*. In this first case **accept()** blocks the caller until a connection is present and if the socket is marked non-blocking and no pending connections are present on the queue, **accept()** returns an error.

Let's see the syntax of accept() system call

```

#include "sys/types.h"
#include "sys/socket.h"

int accept(int socket, struct sockaddr *clientAddress, int *addressLength);

On success, accept() returns the new socket descriptor.
On failure, it returns -1 and the error is in errno.

```

Here in the code accept() takes three arguments first one is **socket** which represent socket on which server is listening the requests, that's why this is also called as **Listening –socket**. Socket on which the server has successfully completed socket(), bind(), and listen() system call. Second is ***clientAddress** which point to an address (struct sockaddr_in). We can use this address to determine the IP address and port of the client. Third and last one ***addressLength** is generally an integer that will contain the actual length of address structure of client (*addressLength should be set to size of (struct sockaddr_in). Accept() returns -1 on error. If it succeeds, it returns a non-

negative integer that is a descriptor for the accepted socket. Let's see one example, which contains the coding showing all the steps we follow till accept().

Example code for accept ().

```
#include <string.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#define SERVERPORT 1729

int main()
{
    int sockfd, newsock, len;
    struct sockaddr_in ser_addr, client_address;

    if ( (sockfd = socket(AF_INET, SOCK_STREAM, 0)) < 0)
    {
        perror("socket error");
        exit(1);
    }
    if ( bind(sockfd, (struct sockaddr *)& ser_addr, sizeof(struct sockaddr)) < 0 )
    {
        perror("bind error");
        exit(1);
    }
    if ( listen(sockfd, 5) < 0 )
    {
        perror("listen error");
        exit(1);
    }
    len = sizeof(client_address);
    while(1)
    {
        if ((newsock = accept ( sockfd, (struct sockaddr *)&cliaddr, &len))<0)
        {
            perror("accept error");
            exit(1);
        }
    }
}
```

If accept is successful it returns new socket descriptor returned by accept system call which completes all the five elements of 5-tuple associations while the old socket which we passed in accept () contains only three elements of 5 tuple association (except remote address and remote process). As you know most of the server are concurrent in practical situation so all go for new socket automatically as a part of accept () as given in the following code:

Concurrent Service

```
int sockfd, newsockfd;
if ((sockfd = socket(...)) < 0)
    error_handle("socket error");
if (bind(sockfd, ...) < 0)
    error_handle("bind error");
if (listen(sockfd, ...) < 0)
```

```

        error_handle("listen error");
for ( ; ; ) {
    newsockfd=accept(sockfd, ...); /* blocks */
    if (newsockfd < 0)
        error_handle("accept error");
    if (fork() == 0) {
        close(sockfd);           /* child */
        do_process(newsockfd);   /* process the request */
        exit(0);
    }
    close(newsockfd);           /* parent */
}

```

When a connection request is received and accepted, the process forks, with the child process serving the connection and the parent waiting for another connection request. Let's see the code for iterative server also:

The iterative server

```

int sockfd, newsockfd;
if ((sockfd = socket(...)) < 0)
    error_handle("socket error");
if (bind(sockfd, ...) < 0)
    error_handle("bind error");
if (listen(sockfd, ...) < 0)
    error_handle("listen error");
for ( ; ; ) {
    newsockfd=accept(sockfd, ...); /* blocks */
    if (newsockfd < 0)
        error_handle("accept error");

    do_process(newsockfd);   /* process the request */
    close(newsockfd);
}

```

Here the server handles the request using the connected socket descriptor, newsockfd. It then terminates the connection with a close and waits for another connection using the original descriptor, sockfd, which still has its remote address and remote process unspecified.



Check Your Progress 1

- 1) Which of the following is returned on success in socket system call?
 - a) Negative integer value
 - b) Big integer value
 - c) Small integer value
 - d) Text "success"

-
-
-
- 2) Which of the following header file contains the definition of data types?
- a) Socket. h
 - b) type. h
 - c) type. h
 - d) data type. H
-
-
-
- 3) Listen system call is used on the server in the case of
- a) Connection-less common
 - b) Connection-oriented comma
 - c) Simplex Comma
 - d) Duplex Comma full
-
-
-

2.3 ELEMENTARY DATA TRANSFER CALLS

Till now we have studied about connection establishment between client and server, let's start the discussion about data transfer between them.

Once a connection is established between sockets, an application program can send and receive data. Sending and receiving data can be done with any one of the several system calls given in this section. The system calls vary according to the amount of data to be transmitted and received and the state of the socket being used to perform the data transfers. The system call pairs (read, write), (send, recv), (sendto, recvfrom) can be used to transfer data (or communicate) on sockets. Let's discuss each pair of system call one by one. The read() system calls allows a process to receive data on a connected socket without receiving the sender's address. The write system calls can be used with a socket that is in a connected state, as the destination of the data is implicitly specified by the connection. The sendto() subroutine allows the process to specify the destination for a message explicitly. The recvfrom() subroutine allow the process to retrieve the incoming message and the sender's address. The use of the above system call varies from protocol to protocol.

read() and write()

read()

This is used to receive data from the remote machine, it assumes that there is already an open connection present between two machines and it is possible only in case of TCP (the connection oriented protocol in TCP/IP).

```
#include "sys/types.h"
#include "sys/socket.h"

int read(int sockfd, char void *buff, int buff_len );
```

Condition	returns
If successful	Numbers of bytes read
If EOF	0
If Error	-1

Here in the above syntax of read() first *sockfd* is socket descriptor, *buff* is a pointer to the buffer where we can store the data and *buff_len* is length of buffer or capacity of buffer.

As given above on success, read() return the number of bytes read, if error occurs it returns -1 (and *errno* is set appropriately) and if it returns zero that means it read all data in file and EOF (end of file) has come.

write

When we want to a send some data to a process running on remote machine we use write() system call. Write() assumes that connection is already open between sender and receiver machine, that's why this is limited to process using TCP. Write () function attempts to write the specified number of bytes from the specified buffer to the interface buffer specified socket descriptor (sockfd).

```
#include "sys/types.h"
#include "sys/socket.h"

int write(int sockfd, const void *buff, int buff_len );
```

Condition	returns
If successful	Numbers of bytes written
If Error	-1

Here in the above syntax of **write ()** first argument *sockfd* is socket descriptor, *buff* is a pointer to the buffer from where we can write the data and *buff_len* is length of buffer or capacity of buffer. As given above in figure on success, the numbers of bytes written are returned (where zero indicates nothing was written). On error, -1 is returned, and *errno* is set appropriately.

send, sendto, recv, and recvfrom

send, sendto, recv, and recvfrom system calls are similar to the standard read and write system calls but these calls require some additional arguemnts like *flag*, **dest_addr*, *addrlen* and **sour_addr* as given below :

Name of additional Argument(s)	Name of system call
<i>flags</i>	send() and recv().
<i>flag</i> , <i>*dest_addr</i> , <i>addrlen</i>	sendto()
<i>flag</i> , <i>*sour_addr</i> , <i>*addrlen</i>	recvfrom()

You can note *addrlen* argument in *sendto* is an integer value while in *recvfrom* it is a pointer to an integer value. You can compare the syntax of these system calls in the

code given below. You can easily note that the first three arguments *sockfd* , **buff* and *nbytes* are similar to first three arguments of *read()* and *write()* argument.

```
#include "sys/types.h"
#include "sys/socket.h"
int send(int sockfd, char *buff, int nbytes, int flags);
int recv(int sockfd, char *buff, int nbytes, int flags);
int sendto(int sockfd, char *buff, int nbytes, int flags,
           struct sockaddr *dest_addr, int addrlen);

int recvfrom(int sockfd, char *buff, int nbytes, int flags,
            struct sockaddr sour_addr, int* addrlen);
```

The use of these system calls depends on the protocol used in your socket association. When we use TCP commonly we use *send()* and *recv()*, although *sendto()*, *recvfrom()*, and *read()* and *write()* can also work. In case of UDP , if you have bound a IP address and Port, you should use *recv()* for reading, and *send()* for sending. If you have not used *bind* on the UDP socket it is better to use *sendto()* and *recvfrom()*.

The different flag values are defined in the **sys/socket.h** header file. Flag can be defined as a nonzero value if the application program requires one or more of the following flag values:

MSG_OOB	Sends or receives out-of-band data.
MSG_PEEK	Looks at data without reading.
MSG_DONTROUTE	Sends data without routing packets
MSG_MPEG2	Sends MPEG2 video data blocks

Send ()

The *send()* function initiates transmission of a message from the specified socket to its peer. The *send()* function sends a message only when the socket is connected.

sockfd is socket descriptor, note that socket must be in connected state before sending data, **buff* is a pointer to data to be transmitted, *nbytes* indicates number of bytes to be sent and *flags* *toolbars* control flags for special protocol features; set to 0 in most cases. On success, *send()* returns the number of bytes actually sent. On failure, it returns -1 and the *errno*.

The following code example shows how you can send data to a connected socket.

```
int i;
/* sockfd is the connected socket file descriptor */
i = send( sockfd, "Hello World!\n", 13, 0);
if( i > 0 ){
    printf("sent %d bytes\n", i);
}
```

sendto()

```
int sendto(int sockfd, char *buff, int nbytes, int flags, struct sockaddr *dest_addr, int addrlen);
```

The *sendto()* function sends a message through a connection oriented or connectionless socket. If *sockfd* specific is a connectionless socket, the message is

sent to the address specified by *dest_addr. If sockfd specifies is a connection oriented socket, *dest_addr and addrlen parameters are ignored.

*dest_addr indicate destination address for data to busiest, addrlen represent length of destination address structure and other arguments are similar to send() function explained above. On success, sendto() returns the number of bytes sent. On failure, it returns -1 and the error is in errno.

recv()

```
int recv(int sockfd, char *buff, int nbytes, int flags);
```

The recv() function receives a message from a socket. The recv() call can be used on a connection oriented socket and, connectionless socket. If no messages are available at the socket, the recv() call waits for a message to arrive if the socket is blocking. If a socket is nonblocking, -1 is returned and the external variable errno is set.

The flags parameter can be set to MSG_PEEK, MSG_OOB, both, or zero. If it is set to MSG_PEEK, any data returned to the user still is treated as if it had not been read, i.e., the next recv() re-reads the same data.

If successful, recv() returns the number of bytes received, otherwise, it returns -1 and sets errno to indicate the error. recv() returns 0 if the socket is blocking and the connection to the remote node failed.

The following code example shows how you would use recv on a connected socket.

```
int i;
char buff[ 250 ];
/* clear out buff */
memset( buff, 0, sizeof( buff ) );
/* sockfd is the connected socket file descriptor */
i = recv( sockfd, buff, sizeof(buff), 0);
if( i < 0 )
{
    printf("error in recving data %d\n", i);
}
if( i == 0 )
{
    printf("socket closed remotely\n");
}
if( i > 0 )
{
    printf("received %d bytes\n", i);
    printf("data :\n%s", buff);
}
}
```

recvfrom ()

```
int recvfrom(int sockfd, char *buff, int nbytes, int flags, struct sockaddr *from, int *addrlen);
```

The recvfrom() system call receives a message from a socket and capture the address from which the data was sent. Unlike the recv() call, which can only be used on a connected stream socket or bound datagram socket, recvfrom() can be used to receive data on a socket whether or not it is connected. If no messages are available at the socket, the recvfrom() call waits for a message to arrive if the socket is blocking. If a socket is nonblocking, -1 is returned and the external variable errno is set.

2.4 CLOSING A SOCKET

There are two ways to close a socket . First is **close(sockfd)** and another is **shutdown(socked, prio)**. Let's check what is the difference between both. In **close()** you can close the socket immediately, while in **shutdown()** you can close the socket, after flushing buffers. You can indicate what buffers shutdown should flush with an integer **prio**. The following example(s) show how to use shutdown, and close, on a connected socket.

```
/* close socket immediately */
• close( sockfd );
• shutdown( sockfd, prio);
/* close socket, and dont recieve any more */
shutdown( sockfd, 0);
/* close socket, and dont send any more */
shutdown( sockfd, 1);
/* close socket, and dont recieve, or send any more */
shutdown( sockfd, 2);
```

close(): The **close()** function is used to delete a socket descriptor created by the **socket()** function.

```
#include <socket.h>
#include <uio.h>
int close (socket )
int socket;
```

close() deletes the socket descriptor, **sockfd**, from the internal descriptor table maintained for the application program and terminates the existence of the communications endpoint. If the socket was connected, the connection is terminated.

shutdown() : The **shutdown()** function is used to gracefully shut down a socket.

```
#include <socket.h>
#include <uio.h>
int shutdown (sockfd, prio)
int sockfd, prio;
```

The input path can be shut down while continuing to send data, the output path can be shut down while continuing to receive data, or the socket can be shut down in both directions at once but the data queued for transmission is not lost in **shutdown()**. If **prio** is 0, further receives are disallowed. If **prio** is 1, further sends are disallowed. If **prio** is 2, further sends and receives are disallowed.

Check Your Progress 2

- 1) When **read ()** system calls returns 0 (zero) it means?
 - a) Error has occurred
 - b) Buffer is empty
 - c) Buffer is full
 - d) End of file has occurred

2) Which of the following header file contains the definition of data types?

- a) Disconnected
- b) Ended
- c) Listing
- d) Connected

.....

.....

.....

3) The recvfrom() function receives a message from socket and capture the

- a) Message from peer socket
- b) Address from which the data was sent
- c) Size of buffer from which the data was sent
- d) Port number from the peer socket.

.....

.....

.....

2.5 TCP AND UDP ARCHITECTURES

UDP architectures

As you know broadly the services in TCP/IP are divided in connection oriented and connection-less services, for which TCP and UDP protocols are responsible. It is important for you to understand the architecture of these UDP and TCP communication.

UDP Client algorithm

To understand the communication architecture between UDP client and server, let use different socket calls.

Here we have explained you the step of UDP client algorithm, first you create a socket, then bind it to a local port remember if bind is not used, the kernel will select a free local port), establish the address of the server, write and read from it, and then terminate. In case, client is not interested in a giving reply then there is no need to use bind.

UDP server algorithm

These are steps generally involved in UDP server, here first you create a socket, bind it to a local port, accept and reply to messages from client, and if you want terminate.

The UDP client/server architecture

The system calls are different for a client-server using a connectionless protocol, from the connection-oriented case. The diagram given below in *Figure 2* shows a typical sequence of systems calls for both the client and the server in connection-less (UDP) communication:

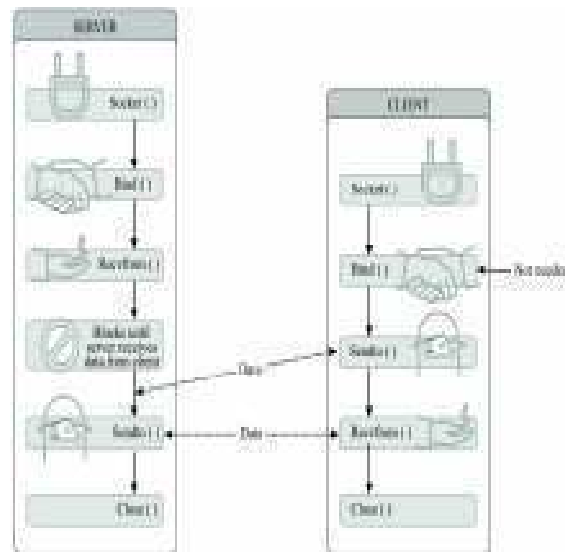


Figure 2: UDP Architecture

TCP architectures

Similarly to understand the architectural communication between TCP client and server we have different socket call in the following section.

TCP Client Algorithm

Here the sequence of steps for connection-oriented client are given. First you will create a socket, bind it to a local port (we usually do not call bind), establish the address of the server, communicate with it, if you want, terminate.

TCP Server Algorithm

These are algorithm sequence for connection oriented server, in this case you first create a socket, bind it to a local port, set up service with indication of maximum number of concurrent services, accept requests from connection oriented clients, receive messages and reply to them and then, finally terminate.

TCP Client/Server Architecture

Typical sequence of system calls to implement TCP clients and servers are given below in the *Figure 3*.

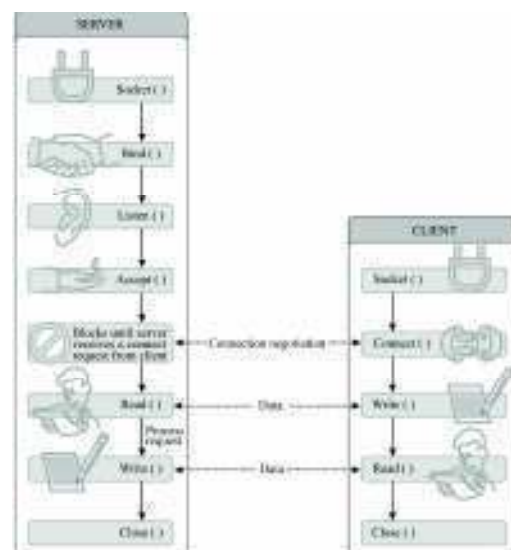


Figure 3:TCP Architecture

2.6 NETWORKING EXAMPLE

An example of an UDP echo client and server is given in this section. The client code is as follows:

UDP echo client

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <string.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <sys/types.h>

#define LINES 128
#define PORT 7200
```

} **Include all the necessary header files.**

```
int main(int argc, char *argv[])
{
    int sockfd;
    int bytesent;
    int byterec;
    struct sockaddr_in ser_addr;
    struct sockaddr_in echo_addr;
    socklen_t len= sizeof(ser_addr);
    char sendline[LINES];
    char recvline[LINES + 1];

    if ((sockfd = socket(AF_INET, SOCK_DGRAM, 0)) < 0)
    {
        perror("socket error");
        exit(1);
    }
    /* fill address completely with 0's */
    memset(& ser_addr, '\0', sizeof(ser_addr));
    ser_addr.sin_family = AF_INET;
    ser_addr.sin_addr.s_addr = htonl(INADDR_ANY);
    ser_addr.sin_port = htons(PORT); ser_addr

    while (1)
    {
        printf("==>");
        fflush(stdout);
        fgets(sendline, LINES, stdin);
        if (strcmp(sendline, "exit\n") == 0)
            break;
        bytesent = sendto(sockfd, sendline, strlen(sendline), 0,
            (struct sockaddr *) &ser_addr, sizeof(ser_addr));
        if (bytesent == -1)
        {
            perror("sendto error");
            exit(1);
        }
        if (strcmp(sendline, "Terminate!\n") != 0)
        {
            if ((byterec = recvfrom(sockfd, recvline, LINES, 0, & echo_addr, &len)) < 0)
            {
```

```
        perror("recvfrom error");
        exit(1);
    }
    recvline[byterec] = '\0';
    printf(" from server: %s\n", recvline);
}
}
close(sockfd);
printf("echo client normal end\n");
return 0;
}
```

UDP echo server

The server code is as follows:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <string.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <sys/types.h>

#define LINES 128
#define PORT 7200

int
main(int argc, char *argv[])
{
    int sockfd;
    int byterec;
    struct sockaddr_in ser_addr;
    struct sockaddr_in cli_addr;
    socklen_t len=sizeof(cli_addr);
    char recvline[LINES + 1];

    if ((sockfd = socket(AF_INET, SOCK_DGRAM, 0)) < 0)
    {
        perror("socket error");
        exit(1);
    }

    memset(& ser_addr, '\0', sizeof(ser_addr));
    ser_addr.sin_family = AF_INET;
    ser_addr.sin_addr.s_addr = htonl(INADDR_ANY);
    ser_addr.sin_port = htons(PORT);
    bind(sockfd, (struct sockaddr *) & ser_addr, sizeof(ser_addr));
    while (1)
    {
        if ( (numrec = recvfrom(sockfd, recvline, LINES, 0,
                               (struct sockaddr *) & cli_addr, & len)) < 0)
        {
            perror("recvfrom error");
            exit(1);
        }
        fprintf(stdout, "connection from %s, port %d. Received %d bytes.\n",
                inet_ntoa(cli_addr.sin_addr),
                ntohs(cli_addr.sin_port),
                byterec);
    }
}
```

```

fflush(stdout);
recvline[byterec]='\0';
if (strcmp(recvline, "Terminator!\n") == 0)
{
    fprintf(stdout, "a client want me to terminate\n");
    fflush(stdout);
    break;
}
if (sendto(sockfd, recvline, byterec, 0, &cli_addr, len) < 0)
{
    perror("sendto error");
    exit(1);
}
}

fprintf(stdout, "server normal end\n");
fflush(stdout);
return 0;
}

```



Check Your Progress 3

- 1) Draw the sequence of system calls required for implementing TCP Clients and server.

.....

.....

.....

.....

.....

.....

- 2) Draw the sequence of system calls required for implementing UDP clients and server.

.....

.....

.....

.....

.....

.....

2.7 SUMMARY

We have discussed different elementary system call and their uses in connection establishment, termination and data transfer in TCP and UDP client server architecture. This unit describes the sockets programming interface. The special needs of network communication are discussed in general terms, and socket types and addressing schemes are introduced. In the first section we have covered the elementary system call to provide you basic knowledge about socket programming. The system calls for data transfer were discussed in detail with an example. We have

also covered the different techniques to close the socket, which will definitely help you during programming. The unit concludes with annotated algorithms of client and server communication programs. In the next unit *Socket Programming* we have given advanced socket calls, which will provide you the in-depth knowledge of the functions and their characteristics that are required for developing network applications.

2.8 ANSWERS TO CHECK YOUR PROGRESS

Check Your Progress 1

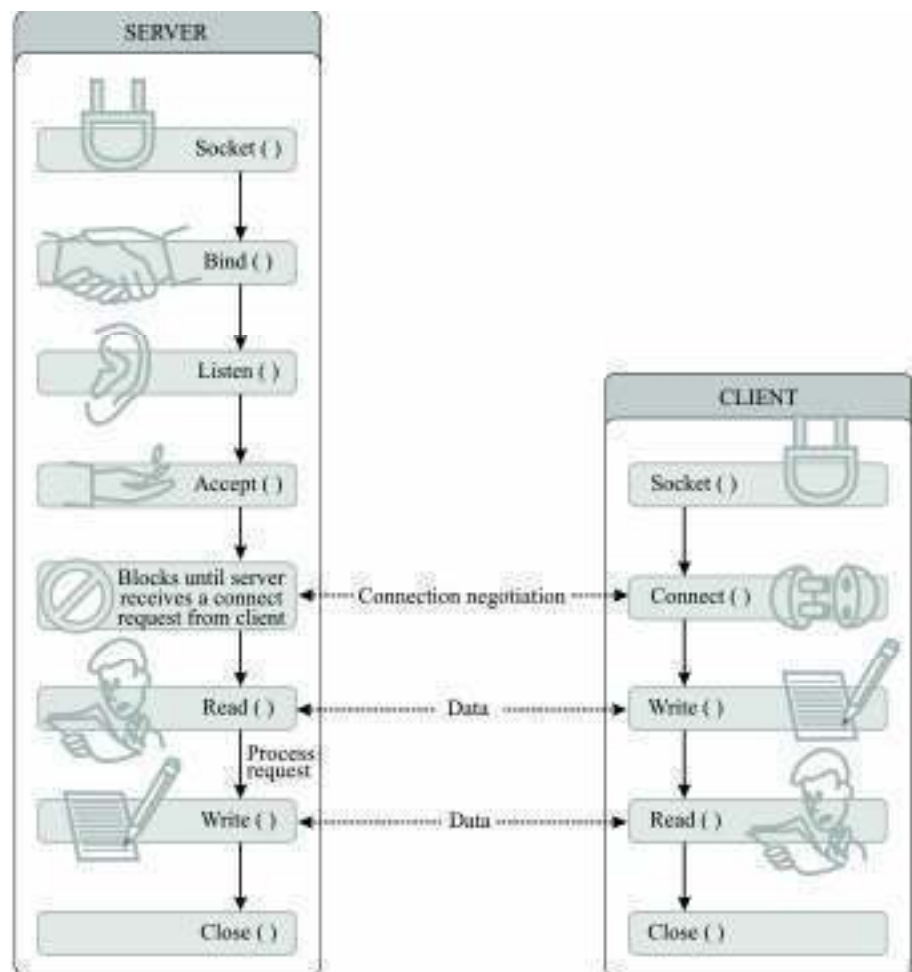
- 1) C
- 2) C
- 3) B

Check Your Progress 2

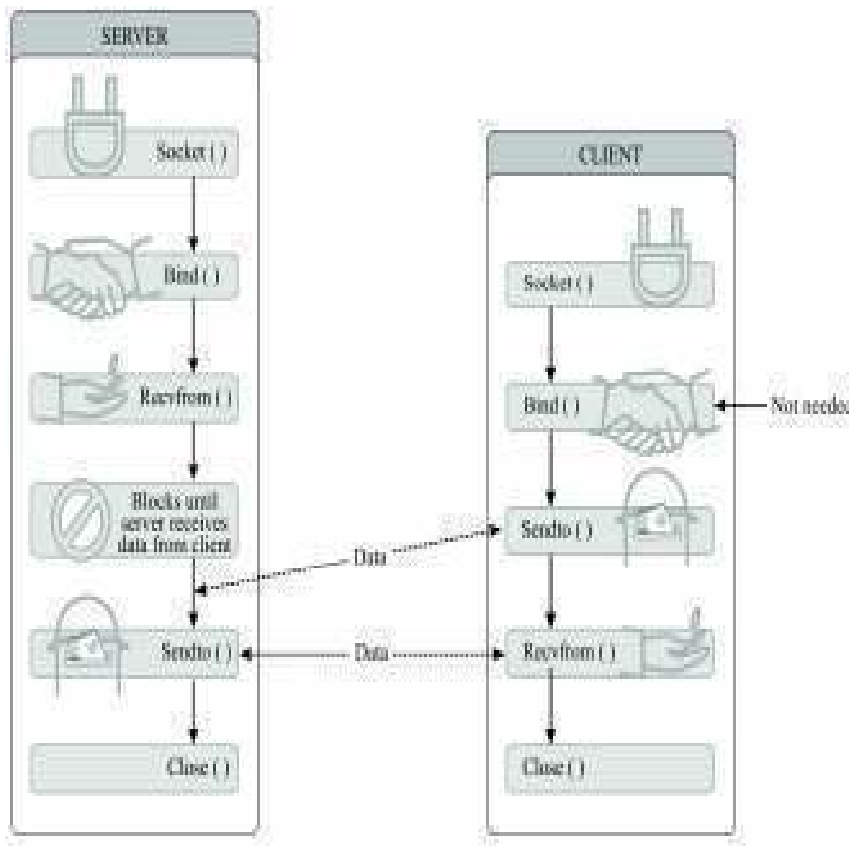
- 1) D
- 2) D
- 3) B

Check Your Progress 3

- 1)



2)



2.9 FURTHER READINGS

- 1) Andrew S. Tanenbaum, *Computer Networks*, Third edition.
- 2) Behrouz A. Forouzan, *Data Communications and Networking*, Third edition.
- 3) Douglas E. Comer, *Internetworking with TCP/IP Vol.1: Principles, Protocols, and Architecture* (4th Edition).
- 4) William Stallings, *Data and Computer Communications*, Seventh Edition.
- 5) W. Richard Stevens, *The Protocols (TCP/IP Illustrated, Volume 1)*.
- 6) W. Richard Stevens, *"UNIX Network Programming"*, Prentice Hall.

UNIT 3 SOCKET PROGRAMMING

Structure	Page Nos.
3.0 Introduction	46
3.1 Objectives	46
3.2 Advance System call	47
3.2.1 Data Transfer	47
3.2.2 Byte Operations and Addressing	48
3.2.3 Socket Options	55
3.2.4 Select System Call	57
3.3 Raw Socket	58
3.4 Multiple Recipients	60
3.4.1 Unicasting	60
3.4.2 Broadcasting	60
3.4.3 Multicasting	60
3.5 Quality of Service Issues	61
3.6 Summary	63
3.7 Answers to Check Your Progress	63
3.8 Further Readings	64

3.0 INTRODUCTION

In the previous unit, we have discussed different elementary system calls required for connection establishment, termination and data transfer in TCP and UDP client server architecture. But because of the complexity and the requirements of network communication like, byte ordering problem, addressing schemes, multiple recipients communication and the need of personal protocols development we need many more system calls for network communication that we will discuss in this unit.

3.1 OBJECTIVES

Our objective is to introduce you the advanced socket calls, which will provide you the in-depth knowledge of the functions and their characteristics that are required for developing network applications. After successful completion of this unit, you should be able to:

- have a reasonable understanding of the Advance System calls;
- understand the use of advance data transfer calls;
- describe the Byte Operations and Addressing functions;
- know the use and applications of socket options;
- understand the concept of raw sockets;
- demonstrate an understanding of the network programs for multiple recipients, and
- know the Quality of Service issues in TCP/IP.

3.2 ADVANCE SYSTEM CALL

This section will describe the syntax and semantics of some of the important system calls, organised by their functionalities. The syntax of these calls are given in C language, where the meaning and use of each system call is defined. Then the syntax with necessary parameters and header files are given. They usually return a negative value to indicate an error and a non-negative return value to indicate success. .

3.2.1 Data Transfer

readv(): The readv() function is used to input data from a socket in scatter mode when the input is to be noncontiguous.

```
#include <socket.h>
#include <uio.h>
int readv ( sockfd, buff, buffcnt)
int sockfd;
struct buffec *buff;
int buffcnt;
```

The buffec structure is defined as:

```
struct buffec
{
    caddr_t    buff_base;
    int        buff_len;
};
```

The readv() function can be called with either a socket or file descriptor. The readv() function is same as read(), but scatters the input data into the **buffcnt** buffers specified by the members of the buff array: buff[0], buff[1], ..., buff[buffcnt-1]. Each **buffec** entry specifies the *base address* and *length* of an area in memory where data should be placed. readv() function returns the total number of bytes read on success, 0 is returned if an end-of-file condition exists, otherwise, the value -1 is returned with error code.

writev(): The writev() function is used to output data from a socket in gather mode when the data are not contiguous.

```
#include <socket.h>
#include <uio.h>
int writev ( sockfd, buff, buffcnt )
int sockfd;
struct buffec *buff;
int iovcnt;
```

writev() is also as write(), but gathers the output data from the buffcnt buffers specified by the members of the buff array: buff [0], buff [1], .., buff [buffcnt-1]. writev() function always writes a complete area before proceeding to the next buff entry. It returns the total number of bytes actually written on success, otherwise the value -1 is returned with error code.

recvmsg() : The recvmsg() is used to receive incoming data that has been queued for a connected or unconnected sockets.

```
#include <socket.h>
#include <uio.h>
int recvmsg ( sockfd, messg, flags )
int sockfd;
struct messghdr messg [ ];
int flags;
```

```
struct messghdr
{
    caddr_t    messg_name;
    int        messg_namelen;
    struct buffec *messg_buff;
    int        messg_bufflen;
    caddr_t    messg_accrights;
    int        messg_accrightslen;
};
```

Data is received in “scatter” mode and placed into noncontiguous buffers. The argument **messg** is the pointer to an object of byte structure, **messghdr**, used to minimize the number of directly supplied parameters. Here **messg_name** and **messg_namelen** specify the source address if the socket is unconnected. The **messg_buff** and **messg_bufflen** describe the “scatter” locations, as described in **read()**. A buffer to receive any access rights sent along with the message is specified in **messg_accrights**, which has length **messg_accrightslen**. It returns the number of bytes received if successful. 0 is returned if an end-of-file condition exits, otherwise, the value -1 is returned with error code.

```
#include <socket.h>
#include <uio.h>
int sendmsg ( sockfd, messg, flags )
int sockfd;
struct messghdr messg [ ];
int flags;
```

sendmsg(): It is used to send outgoing data on a connected or unconnected socket.

Data is sent in gather mode from a list of noncontiguous buffers. The argument **messg** is a pointer to an object of byte structure **messghdr**. This structure is same as defined in **recvmsg()** above. If successful, it returns the number of bytes sent. Otherwise, the value -1 is returned with error code.

3.2.2 Byte Operations and Addressing

Byte Order Conversions: Different types of computers can store data in different ways. A common data format was decided upon for communication between hosts via TCP/IP. The following functions convert data from the host format to the network format, or *vice-versa*. Before going into the details of the data conversion first we can understand the use of these functions.

Function Name	Descriptions
htons	“host to network, short”, for converting a two-byte integer from host order format to network order
htonl	“host to network, long”, for converting a four-byte integer from host order to network order
ntohs	“network to host, short”, for converting a two-byte integer from network order to host order
ntohl	“network to host, long”, for converting a four-byte integer from network order to host order.

Note the pattern of the function names.

Operation	From format	“to”	To format	Data length
htonl	host	to	network	long(4byte)
htons	host	to	network	short(2byte)
ntohl	network	to	host	long(4byte)
ntohs	network	to	host	short(2byte)

There are two type of byte ordering:

- Big endian / host byte ordering: most significant bit first.
- Little endian / Network byte ordering: least significant bit first

In the first form, also called 'Big Endian', The Most Significant Byte (MSB) is kept in the lower address, while the Least significant Byte (LSB) is kept in the higher address. In the second form, also called 'Little Endian', the MSB is kept in the higher address, while the LSB is kept in the lower address. As we know (recall unit 5) different computers use different byte ordering (or different endianness), usually depending on the type of CPU they have. The same problem arises when using a long integer: which word (2 bytes) should be kept first in memory? the least significant word, or the most significant word?

In a network protocol, however, there must be a predetermined byte and word ordering. The IP protocol defines what is called 'the network byte order', which must be kept on all packets sent across the Internet. The programmer on a Unix machine is not saved from having to deal with this kind of information. We'll see how the translation of byte orders is solved when we get down to programming.

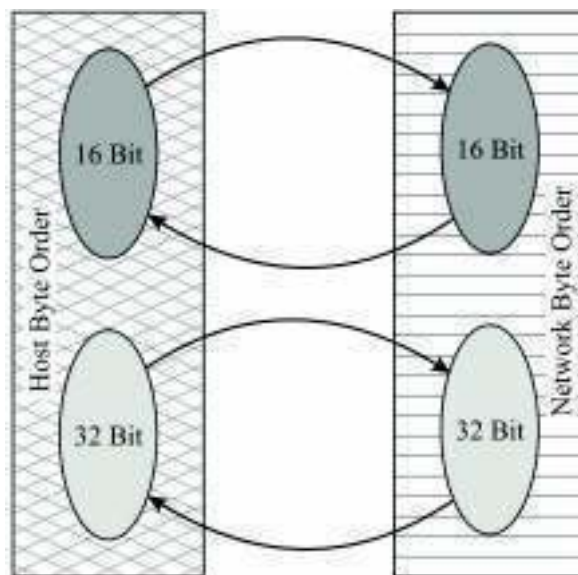


Figure 1: Byte Order Conversion

Here we will take one example to show you how to convert an integer before sending it through a socket and after reading how it can be converted back into the original form:

- 1) `J=htonl(i); /* converting a four-byte integer from host order to network order */`
- 2) `write_data(sockfd, &i, sizeof(i)); /* write data on the socket sockfd*/`
- 3) `read_data(sockfd, &i, sizeof(i)); /* read the data*/`
- 4) `i=ntohl(i); /* convert a four-byte integer from network order to host order */`

If your host order is different from network order and you fail to compensate for it, then your sockets will not work. The sockets API provides a number of conversion functions, including ones that will convert host-order values into network order and vice versa. The details of these functions for converting byte order are given in the following section:

htonl() : It converts 32-bit quantities from host byte order to network byte order.

```
#include <sys/types.h>
#include <inet.h>
unsigned long htonl ( hostlong );
unsigned long hostlong;
```

The htonl() function is provided to maintain compatibility with application programs ported from other systems that byte-swap memory. These systems store 32-bit quantities in right-to-left byte order instead of the usual left-to-right byte order assumed by network protocols. If successful, returns the converted value.

htons() : It converts 16-bit quantities from host byte order to network byte order.

```
#include <sys/types.h>
#include <inet.h>
unsigned short htons ( hostshort );
unsigned short hostshort;
```

The htons() function is provided to maintain compatibility with application programs ported from other systems that byte-swap memory. These systems store 16-bit quantities in right-to-left byte order instead of the usual left-to-right byte order assumed by network protocols. If successful, returns the converted value.

ntohl() : It converts 32-bit quantities from network byte order to host byte order.

```
#include <sys/types.h>
#include <inet/in.h>
unsigned long ntohl( netlong );
unsigned long netlong;
```

ntohl is portable to other environments, including most UNIX systems, that implement BSD sockets. If successful, returns the converted value.

ntohs() : It converts 16-bit quantities from network byte order to host byte order.

```
#include <sys/types.h>
#include <inet/in.h>
unsigned short ntohs(netshort);
unsigned short netshort;
```

Ntohs () is portable to other environments, including most UNIX systems, that implement BSD sockets. If successful, returns the converted value.

Address Conversion

An Internet address is usually written and specified in the dotted-decimal notation. Internally it becomes part of a structure of type in_addr. To convert between these two representations functions are available. In the following section we will discuss these address conversion function in details.

- inet_aton: ASCII to number (string to binary)
- inet_ntoa: number to ASCII (binary to string)
- inet_addr: like inet_aton.

inet() : The inet functions are a set of routines that construct Internet addresses or break Internet addresses down into their component parts. Routines that convert

between the binary and ASCII ("dot" notation) form of Internet addresses are included. Here, in the given sections, these functions are briefly defined.

unsigned long inet_addr (cp) : inet_addr() interpret character strings (char *cp) representing numbers expressed in the Internet standard dotted (".") notation, returning numbers suitable for use as Internet addresses.

unsigned long inet_network (cp) : inet_network() interpret character strings representing numbers expressed in the Internet standard dotted (".") notation, returning numbers suitable for use as network numbers.

char inet_ntoa () : inet_ntoa() takes an Internet address and returns an ASCII string representing the address in dot notation.

unsigned long inet_makeaddr (net, lna) : inet_makeaddr() takes an Internet network number(net) and a local network address (lna) or host number and constructs an Internet address from it.

int inet_lnaof () : inet_lnaof() break apart Internet host addresses, returning the network number .

int inet_netof () : inet_lnaof() break apart Internet host addresses, returning the local network.

You must remember that all the Internet addresses are returned in network byte order and all network numbers and local address parts are returned as integer values in host byte order. Values specified using the Internet standard dotted notation take one of these forms:

- 1) **a.b.c.d** → If 4 parts are specified, each part is interpreted as a byte of data and assigned, from left to right, to the four bytes of an Internet address.
- 2) **a.b.c** → In this case the last part is interpreted as a 16-bit quantity and placed in the right-most two bytes of the network address. This makes the three-part address format convenient for specifying Class B network addresses as 128.net.host.
- 3) **a.b** → In this case when two parts are specified the last part is interpreted as a 24-bit quantity and placed in the right-most three bytes of the network address. This makes the two-part address format convenient for specifying Class A network addresses as net.host.
- 4) **a** → When only one part is given, the value is stored directly in the network address without any byte rearrangement.

The inet_addr() function returns an Internet address if successful, or a value of -1 if the request was unsuccessful. The inet_network() function returns a network number if successful, or a value of -1 if the request was malformed.

inet_aton()

```
#include <types.h>
#include <socket.h>
#include <in.h>
#include <inet.h>
int inet_aton( cp , in)
const char    *cp;
struct in_addr *in;
```

As you know inet_aton() converts an ASCII representation of an Internet address to its network internet address. inet_aton() interprets a character string representing

numbers expressed in the Internet standard. notation, returning a number suitable for use as an Internet address.

inet_ntoa()

```
#include <types.h>
#include <socket.h>
#include <in.h>
#include <inet.h>
char *inet_ntoa ( in )
const struct in_addr in;
```

The `inet_ntoa()` converts an Internet network address to its ASCII “d.d.d.d” representation. `inet_ntoa()` returns a pointer to a string in the base 256 notation “d.d.d.d”. Internet addresses are returned in network byte order (bytes ordered from left to right). `inet_ntoa()` points to a buffer which is overwritten on each call.

Get useful information

There are number of situations when we need to obtain the official name of a host and we should know about its network address or vice versa. Similarly, sometime we need the name of the local host, the port number associated with a service or the name assigned to a socket there may be many more other cases when we want to get some information of client or server using some known information for this purpose. We have lots of get functions in our socket library which you can use accordingly. Here we have given the details of some get* system call description.

gethostbyaddr(): It is used to obtain the official name of a host when its network address is known.

```
#include <netdb.h>
struct hostent *gethostbyaddr ( addr, len, type )
char *addr;
int len, type;
```

A name server is used to resolve the address if one is available; otherwise, the name is obtained (if possible) from a database maintained on the local system. `gethostbyaddr()` returns a pointer to an object with the structure `hostent`.

```
struct hostent
{
    char *h_name;
    char **h_aliases;
    int h_addrtype;
    int h_length;
    char **h_addr_list;
}; #define h_addr h_addr_list [0]
```

Parameters:

h_name	Official name of the host.
h_aliases	A zero terminated array of alternate names for the host.
h_addrtype	The type of address being returned; currently always AF_INET.
h_length	The length, in bytes, of the address.
h_addr_list	A zero-terminated array of network addresses for the host.
h_addr	The first address in <code>h_addr_list</code> ; this is for backward compatibility.

The `gethostbyaddr()` function returns a pointer to the `hostent` structure, if successful. If unsuccessful, a null pointer is returned, and the external integer `h_errno` is set to indicate the nature of the error.

gethostbyname() : It is used to obtain the network address or list of addresses of a host when its official name is known.

```
#include <netdb.h>
struct hostent *gethostbyname (name)
char *name;
```

A name server is used to resolve the name if one is available; otherwise, the address is obtained (if possible) from a database maintained on the local system. The `gethostbyname()` function returns a pointer to an object with the structure `hostent` as defined above. The `gethostbyname()` function returns a pointer to the `hostent` structure, if successful. If unsuccessful, a null pointer is returned, and the external integer `h_errno` is set to indicate the nature of the error.

gethostname() : It is used to obtain the name of the local host.

```
#include <socket.h>
#include <uio.h>
int gethostname ( name, namelen )
char *name;
int namelen;
```

Here name is the *official name* by which other hosts in the communications domain reference it. Generally, a host belonging to multiple communication domains, or connected to multiple networks within a single domain, has one official name by which it is known. The `gethostname()` function returns the standard host name for the local system. The parameter `namelen` specifies the size of the name array. If `gethostname()` is successful, a value of 0 is returned. A return value of -1 indicates an error.

getservbyname() : It is used to obtain the port number associated with a service when its official name is known.

```
#include <netdb.h>
struct servent *getservbyname ( name, proto )
char *name, *proto;
```

This information is acquired from a service database maintained by the local system. `getservbyname()` returns a pointer to an object with this given structure:

```
struct servent
{
    char    *s_name;
    char    **s_aliases;
    int     s_port;
    char    *s_proto;
};
```

Parameters:

s_name	The official name of the service.
s_aliases	A zero-terminated list of alternate names for the service.
s_port	The port number at which the service resides.
s_proto	The name of the protocol to use when contacting the service.

The getservbyname() function sequentially searches from the beginning of the network service database until a matching service name is found, or until the end of the database is reached. It returns a pointer to the servent structure, if successful. If unsuccessful, a null pointer is returned.

getsockname() : It obtains the name assigned to a socket, which is the address of the local endpoint and was assigned with a bind() function.

```
#include <socket.h>
#include <uio.h>
int getsockname ( sockfd, name, namelen )
int sockfd;
struct sockaddr *name;
int *namelen;
```

getsockname() returns the current name for the specified socket. The namelen parameter should be initialized to indicate the amount of space pointed to by name. On return, it contains the actual size of the name returned (in bytes). If getsockname() is successful, a value of 0 is returned. A return value of -1 indicates an error, and the error code stored in the global integer errno indicates the nature of the error.



Check Your Progress 1

- 1) readv() & writev() functions are used when data are?
 - a) Synchronous
 - b) Asynchronous
 - c) Contiguous
 - d) Non-Contiguous

.....
- 2) Which of the following function is used for converting a two-byte integer from network order to host order?
 - a) h tons()
 - b) h tonl()
 - c) n tons()
 - d) n tohl()

.....
- 3) When we want to know the official name of a host and its network address is known, which of the following function is used?
 - a) getservby name
 - b) getsockname()
 - c) gethostbyname()
 - d) gethostbyaddr()

.....

3.2.3 Socket Options

It is possible to set and get a number of options on sockets via the *setsockopt* and *getsockopt* system calls. These options include such things as marking a socket for broadcasting, not to route, to linger on close, etc. The general forms of the calls are:

```
setsockopt(sockfd, level, optname, optval, optlen);
and
getsockopt(sockfd, level, optname, optval, optlen);
```

The parameters to calls are as follows: *sockfd* is the socket on which the option is to be applied. *Level* specifies the protocol layer on which the option is to be applied; The actual option is specified in *optname*. *Optval* and *Optlen* point to the value of the option and the length of the value of the option, respectively. For *getsockopt*, *optlen* is a value-result parameter, initially set to the size of the storage area pointed to by *optval*, and modified upon return to indicate the actual amount of storage used. An example should help clarify things. It is sometimes useful to determine the type (e.g., stream, datagram, etc.) of an existing socket; programs described below will perform this task.

```
#include <sys/types.h>
#include <sys/socket.h>
int type, size;
size = sizeof (int);
if (getsockopt(sockfd, SOL_SOCKET, SO_TYPE, (char *)
&type, &size) < 0) {
```

SO_TYPE get the socket option after the getsockopt call, type will be set to the value of the socket type. Because the socket were a datagram socket, *type* would have the value corresponding to SOCK_DGRAM.

Lets see the details of getsockopt() and setsockopt() system calls in the following section. Then you will understand the above example in more detail:

setsockopt() : The setsockopt() function is used to manipulate options associated with a socket.

```
#include <socket.h>
#include <uio.h>
#include <inet.h>
#include <tcp.h>
int setsockopt ( sockfd, level, optname, optval, optlen )
int sockfd, level, optname;
char *optval;
int optlen;
```

Options can exist at multiple protocol levels; they are always present at the uppermost socket level. Remember when you are manipulating socket options, the level at which the option resides and the name of the option must be specified properly. level is specified as SOL_SOCKET to manipulate options at the socket level. To manipulate options at any other level, the protocol number of the appropriate protocol is supplied.

For explaining it, let's take an example, the case for changing TCP protocol option the following parameters of `setsockopt()` should be passed as given below:

level	set to the protocol number of TCP.
optval	identify a buffer that contains the option value.
optlen	length of the option value in bytes.
optname	parameter and any specified options are passed uninterpreted to the appropriate protocol module for interpretation.

Options at other protocol levels vary in format and name. These options are recognized at the socket level. Each can be set with `setsockopt()` and examined with `getsockopt()`.

There are a number of socket options which either specialise the behaviour of a socket or provide useful information. These options may be set at different protocol levels and are always present at the uppermost "socket" level. These options (some of these options are given below) allow an application program to customize the behaviour and characteristics of a socket to provide the desired effect.

Option	Parameter Type	Parameter Meaning
SO_BROADCAST	int	Non-zero, requests permission to transmit broadcast datagrams (SOCK_DGRAM sockets only).
SO_DONTROUTE	int	Non-zero, requests bypass of normal routing; route based on destination address only.
SO_KEEPALIVE	int	Non-zero, requests periodic transmission of messages, keep alive (protocol-specific).
SO_OOBINLINE	int	Non-zero, requests that out-of-band data be placed into normal data input queue as received.

getsockopt(): The `getsockopt()` function is used to retrieve options currently associated with a socket.

```
#include <socket.h>
#include <uio.h>
int getsockopt (sockfd, level, optname,
optval, optlen)
int sockfd, level, optname;
char *optval;
int *optlen;
```

When retrieving socket options, the level at which the option resides and the name of the option must be specified. In continuation of the previous example to indicate that an option is to be returned by the TCP protocol the following parameters of `getsockopt()` should be indicated as given below:

Level	set to the protocol number of TCP.
Optval	identify a buffer in which the value for the requested option is to be returned.
Optlen	It initially contains the size of the buffer pointed to by the optval and modified on return to indicate the actual size of the value returned. If no option value is to be returned, optval can be supplied as 0.
optname	is passed uninterpreted to the appropriate protocol module for interpretation.

3.2.4 Select System Call

The select() system call is used to synchronise processing of several sockets operating in non-blocking mode. When an application calls recv or recvfrom it is blocked until data arrives for that socket. While the incoming data stream is empty program can do some other job or the situation when a program receives data from multiple sockets. The select function call solves this problem by allowing the program to choose all the socket handles to see if they are available for non-blocking reading and writing operations.

```
#include <socket.h>
#include <uio.h>
int select ( nfds, readfds, writefds, exceptfds, timeout )
int nfds;
fd_set *readfds, *writefds, *exceptfds;
struct timeval *timeout;
int fd;
fd_set fdset;
FD_SET ( fd, &fdset ) /* Includes a particular descriptor fd in fdset. */
FD_CLR ( fd, &fdset ) /* Removes fd from fdset. */
FD_ISSET ( fd, &fdset ) /* non-zero if fd is a member of fdset, zero
otherwise */
FD_ZERO ( &fdset ) /* Initializes a descriptor set fdset to the null set. */
int fd;
fd_set fdset;
```

Sockets that are ready for reading, ready for writing, or have a pending exceptional condition can be selected. If no sockets are ready for processing, the select() function can block indefinitely or wait for a specified period of time (which may be zero) and then return. Select() examines the I/O descriptor sets whose addresses are passed in readfds, Writefds, and exceptfds to see if some of their descriptors are ready for reading, are ready for writing, or have an exceptional condition pending, respectively. The first nfds descriptors are checked in each set (in other words, the descriptors from 0 through nfds-1 in the descriptor sets are examined). readfds - A pointer to a set of file and socket descriptors that are to be polled for non-blocking reading and writing operations. writefds, exceptfds are same as readfds, except these sets contain the file/socket handles to poll for non-blocking writing operations and error detection. On return, select() replaces the given descriptor sets with subsets consisting of those descriptors that are ready for the requested operation. The total number of ready descriptors in all the sets is returned.

If timeout is a non-zero pointer, it specifies a maximum interval to wait for the selection to complete. If timeout is a zero pointer, the select blocks indefinitely. To affect a poll, the timeout argument should be non-zero, pointing to a zero-valued timeval structure.

If successful, select() returns the number of ready descriptors that are contained in the descriptor sets, or 0 if the time limit expires. Otherwise, the value -1 is returned, and the error code.

Example:

```
fd_set fdset;
struct timeval tv;      // sock is an initialized socket handle
tv.tv_sec = 2;
tv.tv_usec = 500000; // tv now represents 2.5 seconds
FD_ZERO(&fdset);
FD_SET(sock, &fdset); // adds sock to the file descriptor set
/* wait 2.5 seconds for any data to be read from any single socket */
select(sock+1, &fdset, NULL, NULL, &tv);
if (FD_ISSET(sock, &fdset))
    recvfrom(s, buffer, buffer_len, 0, &sa, &sa_len);
else
    /* do some other work */
```

3.3 RAW SOCKET

Raw sockets are those sockets, which offer the programmer the possibility to have absolute control over the data, being sent or received through the network. They are very useful when someone needs to create his own protocol but can you think what is a need of creating special protocols? You know with raw sockets you can have your own encrypted traffic tunnels which will enhance your security, you can also optimize the voice and video conference protocols, which may increase the quality and the performance of the sessions.

In this section, you will learn the basics of using raw sockets to insert an IP protocol based datagram into the network traffic. You will learn how to build raw socket scanners or how to perform operations that need to send out raw sockets. Basically, you can send any packet at any time, whereas using the interface functions for your systems IP-stack (connect, write, bind, etc.).

The basic concept of low-level sockets is to send a single packet at one time, with all the protocol headers filled in by the program. Unix provides two kinds of sockets that permit direct access to the network.

- SOCK_PACKET
- SOCK_RAW

SOCK_PACKET receives and sends data on the device link layer. This means, the NIC specific header is included in the data that will be written or read. For most networks, this is the ethernet header. Of course, all subsequent protocol headers will also be included in the data.

The socket type we are going to use, is SOCK_RAW, which includes the IP headers and all subsequent protocol headers and data. A standard command to create a datagram socket is:

```
socket (PF_INET, SOCK_RAW, IPPROTO_UDP);
```

The moment it is created, you can send any IP packets over it, and receive an IP packets.

Note that even though the socket is an interface to the IP header, it is transport layer specific. That means, for listening to TCP, UDP and ICMP traffic, you have to create 3 separate raw sockets, using IPPROTO_TCP, IPPROTO_UDP and IPPROTO_ICMP.

With this knowledge, we can, for example, already create a small sniffer, that dumps out the contents of all tcp packets we receive.

As you see, we are skipping the IP and TCP headers which are contained in the packet, and print out the payload, the data of the session/application layer, only.

```
int fd = socket (PF_INET, SOCK_RAW, IPPROTO_TCP);
char buffer[8192]; /* single packets are usually not bigger than 8192 bytes */
while (read (fd, buffer, 8192) > 0)
printf ("Caught tcp packet: %s\n",
buffer+sizeof(struct iphdr)+sizeof(struct tcphdr));
```

To inject your own packets, all you need to know is the structures of the protocols that need to be included. It is recommended to build your packet by using a struct, so you can comfortably fill in the packet headers. Unix systems provide standard structures in the header files (eg. <netinet/ip.h>). You can always create your own structs, as long as the length of each option is correct. Now, by putting together the knowledge about the protocol header structures with some basic C functions, it is easy to construct and send any datagram(s). To make use of raw packets, knowledge of the basic IP stack operations is essential.



Check Your Progress 2

- 1) How Many parameters are in setsockopt () system call?
 - a) three
 - b) four
 - c) five
 - d) two

.....

.....

.....

.....
- 2) Which of the following option should be used for by passing the normal routing?
 - a) SO_KEEPALIVE
 - b) SO_BYPASS
 - c) SO_BYPASSROUTER
 - d) SO_DONTROUTE

.....

.....

.....

.....

- 3) Which of the following sockets has permission to directly access the network?
- a) SOCK_STREAM
 - b) SOCK_DGRAM
 - c) SOCK_PACKET
 - d) SOCK_TCP

.....
.....

3.4 MULTIPLE RECIPIENTS

There are different ways of transmitting a message over a network, one way in which a data is sent from a single source to a specified destination is known as unicasting, another way is multicasting in which data is sent to a specified group of destinations and in broadcasting where data is sent to all connected destinations. Let's discuss these transmission ways in detail:

3.4.1 Unicasting

Unicast is the term used to describe communication where a piece of information is sent from one point to another point. In this case there is just one sender, and one receiver. In unicast transmission a packet is sent from a single source to a specified destination, is still the predominant form of transmission on LANs and within the Internet. For example in our earlier sections we have given the implementation of unicasting.

3.4.2 Broadcasting

Broadcast is the term which is very familiar to all of us, and it has a traditional meaning associated with TV and radio. It generally means as a transmission that can be received by everyone having the correct equipment. In this case there is just one sender, but the information is sent to all connected receivers. Broadcast transmission is supported by most of the LANs, for example the ARP (address resolution protocol) uses this to send an address resolution query to all computers on a LAN.

Sending Broadcasting message

To send a broadcast message, first of all a datagram socket should be created as given below:

```
sockfd = socket(AF_INET, SOCK_DGRAM, 0);
```

Then the socket option should be set for allowing broadcasting:

```
int b_on = 1;
```

```
setsockopt(sockfd, SOL_SOCKET, SO_BROADCAST, &b_on, sizeof(on));
```

```
/*as we have studied earlier in the course*/
```

```
sin.sin_family = AF_INET;
```

```
sin.sin_addr.s_addr = htonl(INADDR_ANY);
```

```
sin.sin_port = htons(MYPORT);
```

```
bind(sockfd, (struct sockaddr *) &sin, sizeof(sin));
```

3.4.3 Multicasting

Multicasting is the term used to describe communication where a piece of information is sent from one point to a set of other points. In this case there is one sender, and the information is distributed to a set of receivers. One example of this we can see in email and chatting groups on the Internet.

IP multicast provides dynamic many-to-many connectivity between a set of senders (at least 1) and a group of receivers. The format of IP multicast packets are identical to that of unicast packets and is distinguished only by the use of a special class of destination address (class D IP address), which denotes a specific multicast group. Most of the LAN's are able to support the multicast transmission mode. It is inherently supported by the shared LANs, because in that case all packets reach all network interface cards connected to the LAN.

Sending multicasting message

To send a multicast datagram, specify an IP multicast address in the range 224.0.0.0 to 239.255.255.255. A socket option is available to override the default for subsequent transmissions from a given socket:

```
struct in_addr addr;
setsockopt(sock, IPPROTO_O_IP, IP_MULTICAST_IF, &addr, sizeof(addr));
```

Where “addr” is the local IP address of the desired outgoing interface.

If a multicast datagram is sent to a group to which the sending host itself belongs (on the outgoing interface), a copy of the datagram is, by default, looped back by the IP layer for local delivery. Another socket option gives the sender explicit control over whether or not subsequent datagrams are looped back:

```
u_char loop;
setsockopt(sock, IPPROTO_O_IP, IP_MULTICAST_LOOP, &loop,
sizeof(loop));
```

where *loop* is set to 0 to disable loopback, and set to 1 to enable loopback. This option improves performance for applications that may have no more than one instance on a single host (such as a router demon), by eliminating the overhead of receiving their own transmissions.

To receive multicast datagrams sent to a particular port, it is necessary to bind to that local port, leaving the local address unspecified (i.e., `INADDR_ANY`). To receive multicast datagrams sent to a particular group and port, bind the local port, the local address set to the multicast group address. Once bound to a multicast address, the socket cannot be used for sending data.

More than one process may bind to the same `SOCK_DGRAM` UDP port or the same multicast group and port if the *bind* call is preceded by:

```
int on = 1;
setsockopt(sock, SOL_SOCKET, SO_REUSEPORT, &on, sizeof(on));
```

All processes sharing the port must enable this option. Every incoming multicast or broadcast UDP datagram destined to the shared port is delivered to all sockets bound with the port. For backwards compatibility reasons, this does not apply to incoming unicast datagrams. Unicast datagrams are never delivered to more than one socket, regardless of how many sockets are bound to the datagram's destination port.

3.5 QUALITY OF SERVICE ISSUES

“Quality of service” has become a big buzzword. This term conveys about as much useful information about what the technology offers as being told that it is “high performance”.

The Internet is designed on top of the Internet Protocol, a packet-switching technology that is designed to get packets from point “A” to point “B” in whatever

way is most effective, without the user necessarily having any ability to know what route will be taken. In fact, some packets in the same data stream may be sent along different routes. Packets may be stored for a while before being forwarded to their destination, or even dropped and retransmitted.

For most applications, such as simple file or message transfers, this is perfectly fine. However, there are applications where this sort of service is simply of “too low quality”. In these cases, the nature of how the data is delivered is more important than merely how fast it is, and there is a need for technologies or protocols that offer “quality of service”. This general term can encompass a number of related features; common ones include the following:

- **Bandwidth Reservation:** The ability to reserve a portion of bandwidth in a network or interface for a period of time, so that two devices can count on having that bandwidth for a particular operation. This is used for multimedia applications where data must be streamed in real-time and packet rerouting and retransmission would result in problems. This is also called resource reservation.
- **Latency Management:** A feature that limits the latency in any data transfer between two devices to a known value.
- **Traffic Prioritization:** In conventional networks, “all packets are created equal”. A useful QoS feature is the ability to handle packets so that more important connections receive priority over less important one.
- **Traffic Shaping:** This refers to the use of buffers and limits that restrict traffic across a connection to be within a pre-determined maximum.
- **Network Congestion Avoidance:** This QoS feature refers to monitoring particular connections in a network, and rerouting data when a particular part of the network is becoming congested.

So, in essence, quality of service in the networking context is analogous to quality of service in the “real world”. Some applications, especially multimedia such as voice, music and video, are time-dependent and require a constant flow of information more than the raw bandwidth.

Key Concept: The generic term quality of service describes the characteristics of how data is transmitted between devices, rather than just how quickly it is sent. Quality of service features seek to provide more predictable streams of data rather than simply faster ones. Examples of such features include bandwidth reservation, latency minimums, traffic prioritization and shaping, and congestion limitation. Quality of service is more important for special applications such as multimedia than for routine applications such as those that transfer files or messages.

To support quality of service requirements, many newer technologies have been developed or enhanced to add quality of service features to them. This includes the ability to support isochronous transmissions, where devices can reserve a specific amount of bandwidth over time to support applications that must send data in real time.



Check Your Progress 3

- 1) The transmission in which data is sent to a specific group of destination is called.
 - a) Unicasting
 - a) Multicasting
 - b) Broadcasting
 - c) All the above

-
-
-
- 2) Which of the following protocol uses the transmission in local area network?
- a) ARP
 - b) BRP
 - c) TCP
 - d) UDP
-
-
-
- 3) Which of the following condition is applied to disable the loop back in setsockopt () system call.
- a) loop is set to 1
 - b) loop is set to 0
 - c) loop is set to 2
 - d) loop is set to 1
-
-

3.6 SUMMARY

In this unit we have discussed different practical issues about socket programming. In the first section we have covered the advance system call to provide you enhanced knowledge about socket programming. The address conversion and byte ordering routines were discussed with practical syntax of these routines. To help you for designing your own protocols raw sockets were discussed with their use in network programming. We have also covered the techniques to send data to all the connected computers or some group of computers in a network. In the end of this unit we have compared the services of TCP/IP protocols, which will definitely help you in TCP/IP programming. In the next block *Lab Manual* we have given you some exercises in the lab sessions based on these system calls.

3.7 ANSWERS TO CHECK YOUR PROGRESS

Check Your Progress 1

- 1) D
- 2) C
- 3) D

Check Your Progress 2

- 1) C
- 2) D
- 3) C

Check Your Progress 3

- 1) B
- 2) A
- 3) B

3.8 FURTHER READINGS

- 1) Andrew S. Tanenbaum, *Computer Networks*, Third edition.
- 2) Behrouz A. Forouzan, *Data Communications and Networking*, Third edition.
- 3) Douglas E. Comer, *Internetworking with TCP/IP Vol.1: Principles, Protocols, and Architecture* (4th Edition).
- 4) William Stallings, *Data and Computer Communications*, Seventh Edition.
- 5) W. Richard Stevens, *The Protocols (TCP/IP Illustrated, Volume 1)*.
- 6) W. Richard Stevens, *“UNIX Network Programming”*, Prentice Hall.



Indira Gandhi National Open University
School of Computer and Information Sciences

BCS-052
NETWORK
PROGRAMMING AND
ADMINISTRATION

Block

3

NETWORK ADMINISTRATION WITH LINUX

UNIT 1

Introduction to Network Administration	5
---	----------

UNIT 2

Network Administration Activities	20
--	-----------

UNIT 3

Network Configuration and Setting	46
--	-----------

UNIT 4

Network Management and Security	70
--	-----------

PROGRAMME DESIGN COMMITTEE

Prof. Manohar Lal, SOCIS, IGNOU, New Delhi	Prof. Arvind Kalia, Dept. of CS HP University, Shimla	Sh. Akshay Kumar Associate Prof. SOCIS, IGNOU, New Delhi
Prof. H.M Gupta Dept. of Elect. Engg. IIT, Delhi	Prof. Anju Sehgal Gupta SOCIS, IGNOU, New Delhi	Dr. P.K. Mishra, Associate Prof. Dept. of CS, BHU, Varanasi
Prof. M.N Doja, Dept. of CE Jamia Millia, New Delhi	Prof. Sujatha Verma SOS, IGNOU, New Delhi	Sh. P.V. Suresh, Asst. Prof. SOCIS, IGNOU, New Delhi
Prof. C. Pandurangan Dept. of CSE, IIT, Chennai	Prof. V. Sundarapandian IITMK, Trivendrum	Sh. V.V. Subrahmanyam, Asst. Professor SOCIS, IGNOU, New Delhi
Prof. I. Ramesh Babu Dept. of CSE Acharya Nagarjuna University, Nagarjuna Nagar (AP)	Prof. Dharamendra Kumar Dept. of CS, GJU, Hissar	Sh. M.P. Mishra, Asst. Prof. SOCIS, IGNOU, New Delhi
Prof. N.S. Gill Dept. of CS, MDU, Rohtak	Prof. Vikram Singh Dept. of CS, CDLU, Sirsa	Dr. Naveen Kumar, Asst. Prof. SOCIS, IGNOU, New Delhi
	Sh. Shashi Bhushan Associate. Prof. SOCIS, IGNOU, New Delhi	Dr. Subodh Kesharwani, Asst. Prof. SOMS, IGNOU, New Delhi

COURSE CURRICULUM DESIGN COMMITTEE

Prof. Naveen Garg, Professor IIT-Delhi	Dr. D.P Vidyarthi Associate Professor SC&SS, JNU, New Delhi	Sh. M.P. Mishra, Asst. Prof. SOCIS, IGNOU, New Delhi
Sh.Shashi Bhushan Associate Professor SOCIS, IGNOU, New Delhi	Sh. Akshay Kumar Associate Professor SOCIS, IGNOU, New Delhi	Dr. Naveen kumar, Asst. Prof. SOCIS, IGNOU, New Delhi
Ms Seema Gupta Asst. Professor, IP University		

SOCIS FACULTY

Sh. Shashi Bhushan, Director	Sh. Akshay Kumar Associate Professor	Dr. P.V. Suresh Associate Professor
Sh. V.V. Subrahmanyam, Associate Professor	Dr. Naveen Kumar Reader	Sh. M.P. Mishra Asst. Professor
Dr. Sudhansh Sharma Asst. Professor		

PREPARATION TEAM

Prof. M. N Doja (<i>Content Editor</i>) Dept. of Engg. , Jamia Millia Islamia, New Delhi	Dr. A. Murli M. Rao, Computer Division IGNOU	Language Editor Dr. Parmod Kumar School of Humanities, IGNOU
--	--	---

Course Coordinator: Dr. Naveen Kumar, Reader, SOCIS, IGNOU, New Delhi

PRINT PRODUCTION: Sh. Tilak Raj, S.O.(P), MPDD, IGNOU, New Delhi

July, 2013

© *Indira Gandhi National Open University, 2013*

ISBN:

All rights reserved. No part of this work may be reproduced in any form, by mimeograph or any form, by mimeograph or any other means, without permission in writing form the Indira Gandhi National Open University.

Further information on the Indira Gandhi National Open University courses may be obtained from the University's office at Maidan Garhi, New Delhi-110068.

Printed and published on behalf of the Indira Gandhi National Open University by the Director, SOCIS.

Laser Composer : Tessa Media & Computers, C-206, A.F.E-II, Jamia Nagar, New Delhi-25

Print:

BLOCK INTRODUCTION

This block *Network Administration with Linux* covers essentials of network administration and configuration based on Linux. This block provides the knowledge and skills required to work on Linux graphical and command-line tools. You learn to create, edit and search Linux files and directories, limit access within file systems by controlling permissions and ownership. Network configuration, management, monitoring and system tools are also covered in this block. This block has following four units:

- Unit 1: Introduction to Network Administration
- Unit 2: Network Administration Activities
- Unit 3: Network Configuration and Setting
- Unit 4: Network Management and Security

UNIT 1 INTRODUCTION TO NETWORK ADMINISTRATION

Structure	Page Nos.
1.0 Introduction	5
1.1 Objectives	5
1.2 Roles and Responsibilities of Network Administrator	6
1.3 Linux and TCP/IP Internetworking Concepts	6
1.4 Using Network Clients	10
1.5 Understanding System Initialization	11
1.6 User Remote Administration Services and Tools	16
1.7 Summary	17
1.8 Answers to Check Your Progress	18
1.9 Further Readings	19

1.0 INTRODUCTION

Computer network is a telecommunications network that connects a collection of computers to allow communication and data exchange between systems, software applications, and users. The computers that are involved in the network that originate, route and terminate the data are called nodes. The interconnection of computers is accomplished with a combination of cable or wireless media and networking hardware. Two devices are said to be networked when a process in one device is able to exchange information with a process in another device. Networks may be classified by various characteristics, such as the media used to transmit signals, the communications protocols used to organize network traffic, network scale, network topology and organizational scope. The best-known computer network is the Internet.

Communication protocols define the rules and data formats for exchanging information in a computer network. Well-known communications protocols include Ethernet, a hardware and link layer standard that is widely used for local area networks, and the Internet protocol suite (TCP/IP), which defines a set of protocols for communication between multiple networks, for host-to-host data transfer, and for application-specific data transmission formats. Protocols provide the basis for network programming.

1.1 OBJECTIVES

After going through this unit, you will be able to:

- know the roles and responsibilities of a Network Administrator;
- know about network client and its purpose;
- understand LINUX system initialization; and
- understand remote system administration and available tools.

1.2 ROLES AND RESPONSIBILITIES OF NETWORK ADMINISTRATOR

A Network Administrator is an individual, who is responsible for configuring, commissioning and maintenance of network infrastructure and services. It also includes the computer hardware and software systems that make up a data network. In an organization, Network Administrator generally don't typically get involved directly with users, instead focus upon configuring, monitoring and maintenance of network components within organization's LAN/WAN infrastructure. Depending on the organization and its size, the Network Administrator may also involve in design and deployment of computer networks.

Roles of a Network Administrator

The roles of a Network Administrator include activities and tasks to be performed such as configuring, commissioning and maintenance of various network devices-routers, switches, VPN gateways, security devices-Firewall and IDS/IPS, creation of De Militarized Zones (DMZ), IP addresses allocation & management. It also includes configuring and commissioning of various network services/protocols-DHCP, DNS, FTP, HTTP, NFS, etc.

Apart from roles, the Network Administrator is also responsible to

- Ensure data network connectivity
- Network monitoring and management
- Testing the network for breaches, if any
- Keeping an eye out for needed updates
- Update Access Control Lists (ACLs) time to time to regulate network traffic
- Security controls enforcement
- Preparing and implementation of security policy and standards

1.3 LINUX AND TCP/IP INTERNETWORKING CONCEPTS

Linux is a Unix-like computer operating system assembled under the model of free and open source software development and distribution. Linux was originally developed as a free operating system for Intel x86-based personal computers. It is a leading operating system and supports different computer hardware platforms like other operating systems. Linux also runs on embedded systems (a devices where the operating system is typically built into the firmware and highly tailored to the system) such as mobile phones, tablet computers, network routers, building automation controls, televisions, video game consoles and. The Android system, which is in wide use on mobile devices is built on the Linux kernel.

Typically Linux is packaged in a format known as a Linux distribution for desktop and server use. Some popular mainstream Linux distributions include Debian (and its derivatives such as Ubuntu and Linux Mint), Red Hat Enterprise Linux (and its derivatives such as Fedora), and openSUSE (and its commercial derivative SUSE Linux Enterprise Server).

Linux is the most popular network operating system (NOS) runs on a server and enables the server to manage data, users, groups, security, applications, and other networking functions. It runs based on a client/server architecture in which a server enables multiple clients to share resources. Linux allows shared file and printer access

among multiple computers in a network, typically a local area network (LAN), a private network or to other networks. Linux well supports to configure and commissioning of various network servers and services such as proxy servers, Domain name systems, Mail servers, Web servers, etc that are to be accessed through internet.

Internetworking with TCP/IP

Internetworking is the practice of connecting a computer network with other networks through the use of network gateways that provide a common method of routing data packets between the networks. The most notable example of internetworking is the Internet, which a network of networks based on many underlying hardware technologies, but unified by an internetworking protocol standard, referred to as the Internet Protocol Suite and known as TCP/IP.

TCP/IP (Transmission Control Protocol (TCP) and the Internet Protocol (IP)) is a networking model and provides end-to-end connectivity specifying how data should be formatted, addressed, transmitted, routed and received at the destination. It has four abstraction layers which are used to sort all related protocols according to the scope of networking involved.

The Figure 1 shows different networks connected to internet.

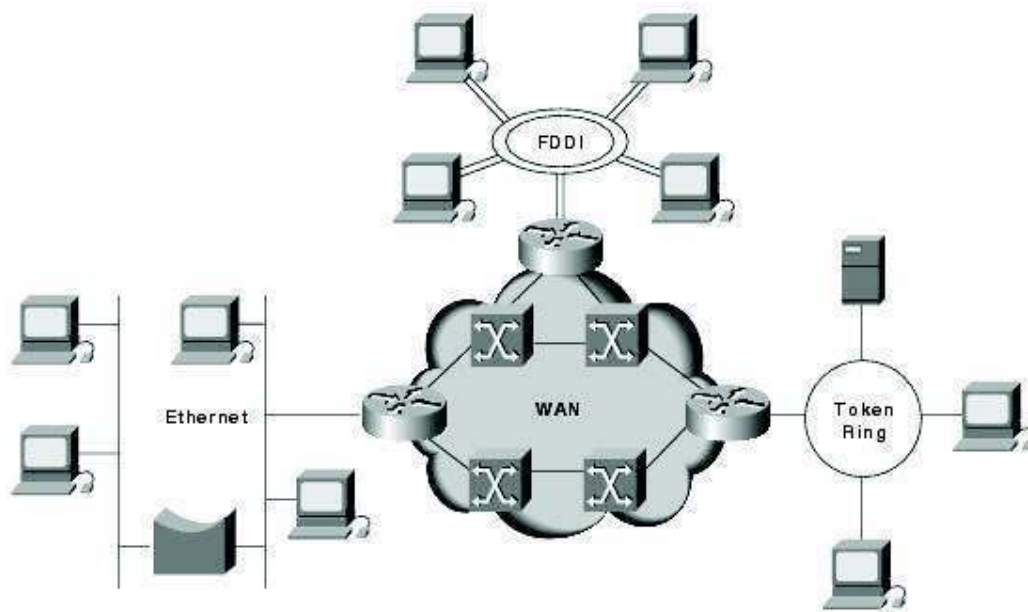


Figure 1: Different networks connected to Internet

How TCP/IP Works

TCP/IP for IP version 4 (IPv4) is a networking protocol suite that uses to communicate over the internet with other computers. It interacts with naming services like Domain Name System (DNS) and security technologies, such as IPsec for secure transfer of IP packets between computers.

Linux provides extensive support for the Transmission Control Protocol/Internet Protocol (TCP/IP) suite, as both a protocol and a set of services for connectivity and management of IP internetworks. Knowledge of the basic concepts of TCP/IP is an absolute requirement for the proper understanding of the configuration, deployment, and troubleshooting of IP-based Linux server.

TCP/IP Protocol Architecture

TCP/IP protocols map to a four-layer conceptual model. The four layers are Application, Transport, Internet, and Network Interface. Each layer corresponds to one or more layers of the seven-layer Open Systems Interconnection (OSI) model.

Figure 2 shows the TCP/IP protocol architecture.

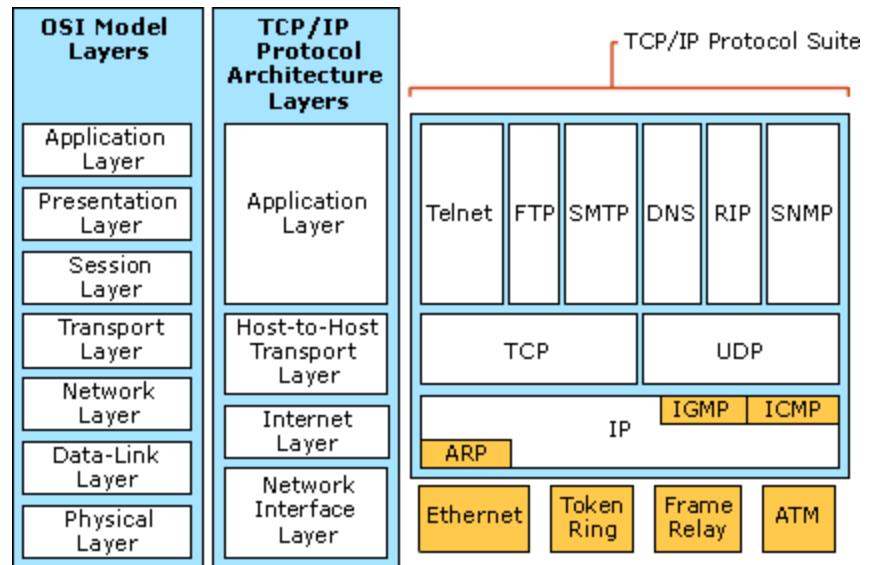


Figure 2: TCP/IP Protocol Architecture

Network Interface Layer

The Network Interface layer also called the Network Access layer that handles placing TCP/IP packets on the network medium and receiving TCP/IP packets off the network medium. TCP/IP was designed to be independent of the network access method, frame format, and medium. In this fashion, TCP/IP can be used to connect differing network types and these include local area network (LAN) media such as Ethernet and Token Ring and also WAN technologies such as X.25 and Frame Relay. The network interface layer encompasses the data link and physical layers of the OSI model.

Internet Layer

The Internet layer handles addressing, packaging, and routing functions. The core protocols of the Internet layer are IP, ARP, ICMP, and IGMP.

- The Internet Protocol (IP) is a routing protocol that handles IP addressing, routing, and the fragmentation and reassembly of packets.
- The Address Resolution Protocol (ARP) handles resolution of an Internet layer address to a Network Interface layer address, such as a hardware address.
- The Internet Control Message Protocol (ICMP) handles providing diagnostic functions and reporting errors due to the unsuccessful delivery of IP packets.
- The Internet Group Management Protocol (IGMP) handles management of IP multicast group membership.

The Internet layer is similar to the Network layer of the OSI model.

Transport Layer

The Transport layer handles and provides session and datagram communication services to Application layer. The core protocols of the Transport layer are Transmission Control Protocol (TCP) and the User Datagram Protocol (UDP).

- TCP provides a one-to-one, connection-oriented, reliable communications service. TCP handles the establishment of a TCP connection, the sequencing and acknowledgment of packets sent, and the recovery of packets lost during transmission.
- UDP provides a one-to-one or one-to-many, connectionless, unreliable communications service. UDP is used when the amount of data to be transmitted is small (such as data that fits into a single packet), when you do not want the overhead of establishing a TCP connection, or when the applications or upper layer protocols provide reliable delivery.

The TCP/IP Transport layer is similar to the Transport layer of OSI model.

Application Layer

The application layer provides services for an application program to ensure that effective communication with another application program in a network is possible. The application layer is not the application itself that is doing the communication, but with various application layer protocols.

The most widely known Application layer protocols are those used for the exchange of user information:

- The Hypertext Transfer Protocol (HTTP) is used to transfer files that make up the Web pages of the World Wide Web.
- The File Transfer Protocol (FTP) is used for interactive file transfer.
- The Simple Mail Transfer Protocol (SMTP) is used for the transfer of mail messages and attachments.
- Telnet, a terminal emulation protocol, is used for logging on remotely to network hosts.

Additionally, the following Application layer protocols help facilitate the use and management of TCP/IP networks:

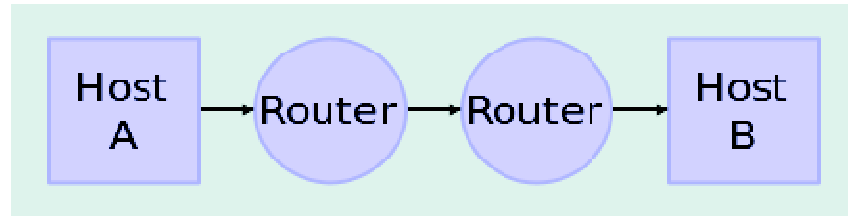
- The Domain Name System (DNS) is used to resolve a host name to an IP address.
- The Routing Information Protocol (RIP) is a routing protocol that routers use to exchange routing information on an IP internetwork.
- The Simple Network Management Protocol (SNMP) is used between a network management console and network devices (routers, bridges, intelligent hubs) to collect and exchange network management information.

The TCP/IP Application layer encompasses the responsibilities of the Session, Presentation, and Application layers of OSI model.

How Data Transmitted

Figure 3 shows, how data transmitted from source computer to destination computer.

Network Topology



Data Flow

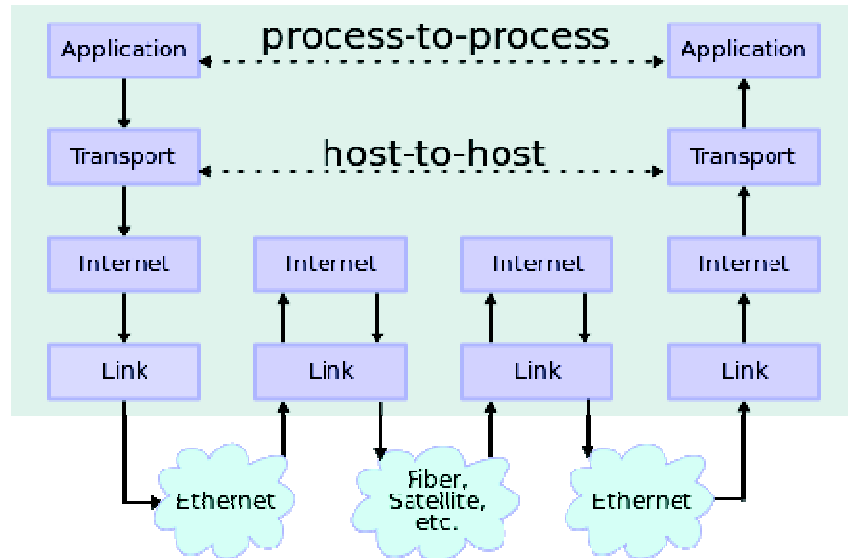


Figure 3: Data transmission between two Hosts

1.4 USING NETWORK CLIENTS

A client is a piece of computer hardware or software or both or a computer program that sends a service request to a server/computer program or accesses a service made available by a server/computer program. A network client is a client that can interact with servers/computer program through network/internet. The term client was first applied to devices that were not capable of running their own stand-alone programs, but could interact with remote computers via a network.

For example, web browsers are clients that connect to web servers and retrieve web pages for display. Email clients retrieve email from mail servers. Online chat uses a variety of clients, which vary depending on the chat protocol being used. Multiplayer video games or online video games may run as a client on each computer. The term "client" may also be applied to computers or devices that run the client software or users that use the client software. Similarly, the devices such as laptops, notebooks, palmtops, tablet PCs, smart phones, and other such devices also called network clients through which services requests can be sent or services can be retrieved to/from servers that are providing services. Figure 4 shows network clients that are being used in a cloud computing model.

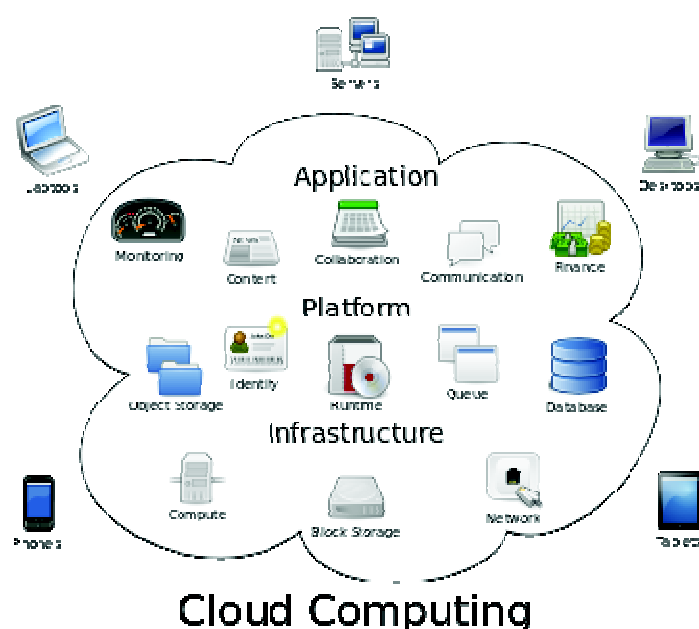


Figure 4: Network clients in a Cloud computing model

1.5 UNDERSTANDING SYSTEM INITIALIZATION

Here system means the combination of the computer hardware and the software (generally it the operating system installed on the computer hardware). Linux system (the computer installed with Linux operating system) initialization means how system gets started after power is on. The Linux startup process is the process of Linux-operating system initialization. It is in many ways similar to the BSD and other Unix style boot processes, from which it derives.

In Linux, the flow of control during a boot is from BIOS (Basic Input/output System), to boot loader, to kernel. The kernel then starts the scheduler and runs the first program *init* (which is mostly responsible to run startup scripts for each runlevel), at which point the kernel goes idle unless called externally.

init (short for initialization) is a program for Unix-based computer operating systems that spawns all other processes. It runs as a daemon and typically has PID 1. The *boot loader* starts the kernel and the kernel starts *init*. If some one has to delete *init* without a replacement, the system would encounter a kernel panic on the next reboot.

Overall system initialization process

- i) The *BIOS* performs hardware-platform specific startup tasks
- ii) Once the hardware is recognized and started correctly, the BIOS loads and executes the partition boot code from the designated boot device, which contains phase 1 of a Linux boot loader. Phase 1 loads phase 2 (the bulk of the boot loader code). Some loaders may use an intermediate phase (known as phase 1.5) to achieve this since modern large disks may not be fully readable without further code.

- iii) The *boot loader* often presents the user with a menu of possible boot options. It then loads the operating system, which decompresses into memory, and sets up system functions such as essential hardware and memory paging, before calling 'start_kernel()'.
'start_kernel()' then performs the majority of system setup (interrupts, the rest of memory management, device initialization, drivers, etc.) before spawning separately, the idle process and scheduler, and the Init process (which is executed in user space).
- iv) The Init process executes scripts as needed that set up all non-operating system services and structures in order to allow a user environment to be created, and then presents the user with a login screen.

The standard sequence for initializing a Linux system is as follows:

- Power on the System
- Initializing the BIOS
- Bootloader
- Kernel initialization
- Starting from "init"

Initializing the BIOS

- The BIOS (Basic Input/Output System) is the interface between the hardware and software at a very basic level, it provides all the basic instructions used by the operating system.
- The BIOS begins by executing an auto-ignition test (POST), and then it searches for devices.
- After the POST, a boot device is selected from a list that is configurable in the BIOS.
- The BIOS reads and executes the first physical sector of the boot media selected on the system, which is usually contained in the first 512 bytes of hard disk.

Bootloader

The bootloader is usually contained in the first sector of the disk and then read and executed by the BIOS. The storage space that reads the BIOS is not sufficient to contain all the bootloader, but just a part sufficient enough to start the rest of the bootloader, which is usually contained in a configuration file stored elsewhere on the disc. Hence the start is done in two steps:

- Launch via BIOS
- Launch a file under boot

The bootloader is designed to load and run the system kernel. The standard bootloader is GRUB but can also shift to LILO.

Kernel initialization

The kernel in Linux handles all operating system processes, such as memory management, task scheduling, I/O, interprocess communication, and overall system control. This is loaded in two stages - in the first stage the kernel is loaded into memory and decompressed, and a few fundamental functions such as basic memory management are set up. Control is then switched to the main kernel start process. Once the kernel is fully operational, it looks for an init process to run, which sets up a user space and the processes needed for a user environment and ultimate login. The kernel itself is then allowed to go idle, subject to calls from other processes.

The kernel initialization includes:

- The detection and initialization of devices. It means any device drivers compiled into the kernel are called and try to locate their corresponding devices.
- Mounting the root file system in read-only mode
- Loading the initial process "init"

The kernel initialization is very rapid and therefore it is very difficult to follow visually. One can read system generated log file to check what happened during kernel initialization. Generally log file can be stored under */var/log/dmesg*

Initialize "init"

Init (initialization) is the father of all processes. Its primary role is to create processes from a script stored in the file */etc/inittab*. This file usually has entries which cause *init* to spawn gettys on each line that users can log in. It also controls autonomous processes required by any particular system.

- "init" is the main process, it will always have a **PID** value: **1**.
- "init" reads its configuration from the */etc/inittab* file, that contains the settings for the system at every level of execution.

Run Levels

A run level is a software configuration of the system which allows only a selected group of processes to exist. The processes spawned by *init* for each of these run levels are defined in the */etc/inittab* file.

"init" defines the following run levels:

- **Level 0:** Stop (not to be attributed to the *initdefault*)
- **Level 1, S:** single user mode (only the root user can log). Typically used for maintenance.
- **Level 2:** Multi-user mode without NFS network
- **Level 3:** full mode for multiple users including network
- **Level 4:** User Configurable duplicate but the level 3 by default.
- **Level 5:** X11 (including network)
- **Level 6:** Restart

Runlevel (System V)

The ability to change runlevel offers easy interaction with administrators; this allows to switch between different levels of startup.

Scripts services are in */etc/rc.d/init.d*. Each runlevel correspond to a */etc/rc.d/rcX.d* directory, where **X** is the runlevel.

System Shutdown

On shutdown, *Init* is called to close down all user space functionality in a controlled manner, again via scripted directions, following which *Init* terminates and the Kernel executes its own shutdown.

To stop the system, use commands like:

```
#Shutdown -h now
```

```
#halt
```

```
#poweroff
```

```
#init 0
```

Check Your Progress 1

1. Write the main responsibilities of a Network Administrator.

.....
.....
.....
.....
.....

2. What are the various run levels of Linux system?

.....
.....
.....
.....
.....

1.6 USER REMOTE ADMINISTRATION SERVICES AND TOOLS

Remote administration is an approach being followed to control either a computer system or a network or an application or all three from a remote location. A remote location may refer to a computer in the next room or one on the other side of the world. Generally, remote administration is essentially adopted when it is difficult or impractical to a person to be physically present and do administration on a system's terminal.

Requirements to perform Remote Administration

Network connectivity is essentially needed to perform remote administration. The network could be either Local Area Network (LAN) connectivity or internet connectivity depending on remote location. It means, any computer with an internet connection or on a LAN can be remotely administered.

For non-malicious administration, the user must install or enable remote administration software/tool on the host system then the user/client can access the host system from another computer using the remote tool.

Common Services for which remote administration is used

Generally, remote administration is needed for user management, file system management, software installation/configuration, network management- Network Security/Firewalls, VPN, Infrastructure Design, Network File Servers, Auto-mounting etc. and kernel optimization/ recompilation.

The following are some of the tasks/ services for which remote administration need to be done:

General

Controlling one's own computer from a remote location (e.g. to access the software on a personal computer from an internet café).

ICT Infrastructure Management

Remote administration essentially needed to administer the ICT infrastructure such as the servers, the routing and switching components, the security devices and other such related.

Shutdown

- Shutting down or rebooting a computer over a network

Accessing Peripherals

- Using a network device, like printer
- Retrieving streaming data, much like a CCTV system

Modifying

- Editing another computer's registry settings
- Modifying system services
- Installing software on another machine
- Modifying logical groups

Viewing

- Remotely assisting others
- Supervising computer or internet usage
- Access to a remote system's "Computer Management" snap-in

Hacking

Computers infected with malware such as Trojans sometimes open back doors into computer systems which allow malicious users to hack into and control the computer. Such users may then add, delete, modify or execute files on the computer to their own ends.

Remote Desktop Solutions for Linux

Most people who are used to a Unix-style environment know that a machine can be reached over the network at the shell level using utilities like telnet or ssh. And some people realize that X Windows output can be redirected back to the client workstation. But many people don't realize that it is easy to use an entire desktop over the network. The following are some of proprietary and open source applications that can be used to achieve this.

SSH (Secure Shell)

Secure Shell (SSH) is a proprietary cryptographic network tool for secure data communication between two networked computers that connects, via a secure channel over an insecure network, a server and a client (running SSH server and SSH client programs, respectively). The protocol specification distinguishes between two major versions that are referred to as SSH-1 and SSH-2.

The best-known application of the tool is for access to shell accounts on Unix-like operating systems-GNU/Linux, OpenBSD, FreeBSD, but it can also be used in a similar fashion for accounts on Windows.

SSH is generally used to log into a remote machine and execute commands. It also supports tunneling, forwarding TCP ports and X11 connections, it can transfer files using the associated SSH file transfer (SFTP) or secure copy (SCP) protocols. SSH uses the client-server model.

SSH is important in cloud computing to solve connectivity problems, avoiding the security issues of exposing a cloud-based virtual machine directly on the Internet. An SSH tunnel can provide a secure path over the Internet, through a firewall to a virtual machine

OpenSSH (OpenBSD Secure Shell)

OpenSSH is a tool providing encrypted communication sessions over a computer network using the SSH protocol. It was created as an open source alternative to the proprietary Secure Shell software suite offered by SSH Communications Security.

Telnet

Telnet is used to connect a remote computer over network. It provides a bidirectional interactive text-oriented communication facility using a virtual terminal connection on internet or local area networks. Telnet provides a command-line interface on a remote host. Most network equipment and operating systems with a TCP/IP stack support a Telnet service for remote configuration (including systems based on Windows NT). Telnet is used to establish a connection to Transmission Control Protocol (TCP) on port number 23, where a Telnet server application (telnetd) is listening.

Experts in computer security, recommend that the use of Telnet for remote logins should be discontinued under all normal circumstances, for the following reasons:

- Telnet, by default, does not encrypt any data sent over the connection (including passwords), and so it is often practical to eavesdrop on the communications and use the password later for malicious purposes; anybody who has access to a router, switch, hub or gateway located on the network between the two hosts where Telnet is being used can intercept the packets passing by and obtain login, password and whatever else is typed with a packet analyzer.
- Most implementations of Telnet have no authentication that would ensure communication is carried out between the two desired hosts and not intercepted in the middle.
- Several vulnerabilities have been discovered over the years in commonly used Telnet daemons.

rlogin

rlogin is an utility for Unix-like computer operating systems that allows users to log in on another host remotely through network, communicating through TCP port 513.

rlogin has several serious security problem- all information, including passwords is transmitted in unencrypted mode. rlogin is vulnerable to interception. Due to serious security problems, rlogin was rarely used across distrusted networks (like the public internet) and even in closed networks.

rsh

The remote shell (rsh) can connect a remote host across a computer network. The remote system to which rsh connects runs the rsh daemon (rshd). The daemon typically uses the well-known Transmission Control Protocol (TCP) port number 514. In security point of view, it is not recommended.

PuTTY

PuTTY is a free and open source terminal emulator application which can act as a client for the SSH, Telnet, rlogin, and raw TCP computing protocols and as a serial console client. The name "PuTTY" has no definitive meaning, though "tty" is the name for a terminal in the Unix tradition, usually held to be short for Teletype. PuTTY was originally written for Microsoft Windows, but it has been ported to various other operating systems as well

VNC (Virtual Network Computing)

VNC is a remote display system which allows the user to view the desktop of a remote machine anywhere on the internet. It can also be directed through SSH for security

Install VNC server on a computer (server) and install client on local PC. Setup is extremely easy and server is very stable. On client side, set the resolution and connect to IP of VNC server.

One can download VNC software from the following URLs:

<http://www.realvnc.com/download.html>

<http://www.tightvnc.com/download.html>

<http://www.uvnc.com/>

FreeNX allows to access desktop from another computer over the internet. One can use this to login graphically to a desktop from a remote location. One example of its use would be to have a FreeNX server set up on home computer, and graphically logging in to the home computer from work computer, using a FreeNX client. One can download FreeNX software from the following URLs:

<https://help.ubuntu.com/community/FreeNX>

<http://ubuntuforums.org/showthread.php?t=97277&highlight=freenx>

<http://freenx.berlios.de/> (FreeNX homepage)

Wireless Remote Administration

Remote administration software has recently started to appear on wireless devices such as the BlackBerry, Pocket PC, and Palm devices, as well as some mobile phones.

Generally these solutions do not provide the full remote access seen on software such as VNC or Terminal Services, but do allow administrators to perform a variety of tasks, such as rebooting computers, resetting passwords, and viewing system event logs, thus reducing or even eliminating the need for system administrators to carry a laptop or be within reach of the office.

AetherPal and Netop are some of the tools used for full wireless remote access and administration on Smartphone devices.

Disadvantages of Remote Administration

Remote administration has many disadvantages too apart from its advantages. The first and foremost disadvantage is the security. Generally, certain ports to be open at Server level to do remote administration. Due to open ports, the hackers/attackers takes advantage to compromise the system. It is advised that remote administration to be used only in emergency or essential situations only to do administration remotely. In normal situations, it is ideal to block the ports to avoid remote administration.

Check Your Progress 2

1. Why remote administration is needed? Explain.

.....

.....

.....

.....

.....

2. List the different network clients.

.....

.....

.....

.....

.....

3. What are the different remote administration tools?

.....

.....

.....

.....

.....

1.7 SUMMARY

In this unit, different roles and responsibilities of a network administrator are clearly explained. The TCP/IP and its role in data transmission from source to destination is made clear. System initialization process and importance of remote administration also covered.

1.8 ANSWERS TO CHECK YOUR PROGRESS

Check Your Progress 1

- 1) Network Administrator is responsible to
 - Ensure data network connectivity

- Network monitoring and management
- Testing the network for breaches, if any
- Keeping an eye out for needed updates
- Update Access Control Lists (ACLs) time to time to regulate network traffic
- Security controls enforcement
- Preparing and implementation of security policy and standards

2) The following are various run levels of a Linux system

Level 0: Stop (not to be attributed to the initdefault)

Level 1, S: single user mode (only the root user can log). Typically used for maintenance.

Level 2: Multi-user mode without NFS network

Level 3: full mode for multiple users including network

Level 4: User Configurable duplicate but the level 3 by default.

Level 5: X11 (including network)

Level 6: Restart

Check Your Progress 2

1) The need for remote administration is for

- User management
- ICT Infrastructure management
- Problem Diagnosis and Troubleshooting
- File system management
- Software installation/configuration
- Network Management- Network Security/Firewalls, VPN, Infrastructure Design, Network File Servers, Auto-mounting etc.

2) The different network clients are

Computers, Notebooks, Palmtops, Tablet PCs, Smart phones and other such related.

3) The following are different remote administration tools

SSH, Telnet, rsh, rlogin, openSSH, VNC, etc

1.9 FURTHER READINGS

1. Computer Networks by Andrew S Tanenbaum , Fifth Edition
2. SA2, Redhat System Administration I & II, Student Workbook
3. Cisco Certified Network Associate Study Guide, Seventh Edition by Todd Lammle
4. Redhat Enterprise Linux System Administration
5. <http://en.wikipedia.org/wiki/Internetworking>
6. http://en.wikipedia.org/wiki/Remote_administration

UNIT 2 NETWORK ADMINISTRATION ACTIVITIES

Structure	Page Nos.
2.0 Introduction	20
2.1 Objectives	20
2.2 Managing Software Packages	20
2.3 File Systems	23
2.4 Managing Users	29
2.5 System and Kernel Management	34
2.6 Basic Troubleshooting	38
2.7 Summary	43
2.8 Answers to Check Your Progress	44
2.9 Further Readings	45

2.0 INTRODUCTION

System administration is a critical activity that includes installation of software , configuration, commissioning, monitoring and finally problem diagnosis & troubleshooting of systems. It is a challenging task to the system administrator to ensure the uptime of the systems all times. Creation of a suitable file system with adequate disk partitions requires the system administrator enough working knowledge. Once system commissioning is over, the administrator has to hardening of OS in terms of removing unnecessary packages that come along with OS, close unnecessary ports and other such related. System and kernel management is another activity where the administrator has to keep on doing as part o regular system resources monitoring. It is very essential to the system administrator to know and use at least some useful tools/commands that are being used as part of problem diagnosis and troubleshooting so that system health can be maintained always and in turn make the systems uptime always.

2.1 OBJECTIVES

After going through this unit, you will be able to:

- know how to manage software packages;
- understand what is a file system and its structure;
- know how to manage users;
- understand the management of system and kernel; and
- find commands that are useful for troubleshooting.

2.2 MANAGING SOFTWARE PACKAGES

Software Packages are archives of files that include all the files that make up a piece of software(such as an application itself, shared libraries, development packages containing files that needed to build software against a library, etc) and also has a set of instructions to make them work. A package is properly integrated into the

on distribution it has been built for installation paths, dependencies, desktop integration, and proper startup scripts for servers, etc.

Package manager

Generally, in modern Linux distributions like openSUSE, software installation is done with a package manager. Package manager is a collection of software tools to automate the process of installing, upgrading, configuring, and removing software packages for a computer's operating system. It works on top of RPM (Red hat Package Manager) and gets software packages from repositories (such as online servers, CDs and DVDs), solves the dependencies and installs them on your system. The package manager also makes it easy to remove packages later or to update them. The number of packages available for installation depends on which repositories you have added. Package manager typically maintains a database of software dependencies and version information to prevent software mismatches and missing prerequisites.

Package management systems

Package management systems are designed to save organizations time and money through remote administration and software distribution technology that eliminate the need for manual installs and updates. This can be particularly useful for large enterprises whose operating systems are based on Linux and other Unix-like systems, typically consisting of hundreds or even thousands of distinct software packages.

Package Metadata

Packages are distributions of software, applications and data. Packages also contain metadata that contains

- Summary (software name, etc)
- Description
- A list of files contained in the package
- The version of the software it contains as well as the release number of the package
- When, where and by whom it has been built
- What architecture it has been built for
- Checksums of the files contained in the package
- The license of the software it contains
- Which other packages it requires to work properly

After installation, metadata is stored in a local package database.

Package dependencies

The important aspect of the packages archive is the relations they contain. The packages also relates files to other packages as the packaged applications need an execution environment (like other tools, libraries, etc.) to actually run the application. Package dependencies are used to express such relations.

For example, package A needs the packages of B, C and D to be installed in order to work properly.

Package dependencies are transitive, which means that when package A needs package B, and package B needs package C, package A also needs package C, which is why you sometimes end up with lots of packages to install although you just wanted that one application.

Dependencies on libraries (typically packages with a name that starts with "lib") are very common and pretty much every single application depends on a set of library packages.

Packages and package dependencies are very important aspects of Linux distributions (as well as other BSD and UNIX systems) because they provide a modular way to set up and manage an operating system and its applications. This is especially true for library packages. For example, the package `openssl` contains cryptographic libraries that are used by many applications and other libraries (e.g. for SSL encryption). When a new, improved version of `openssl` is available, all the applications that use it will benefit from it just by upgrading that single package to the newer version. It is also a very efficient way to maintain a stable and secure system. It means, when a security hole, exploit or bug affects a library used by one or many applications, upgrading the single package will fix it for all of them. Figure 1 Shows the linkages between user systems, package manager, repository, meta data, packages and package dependencies.

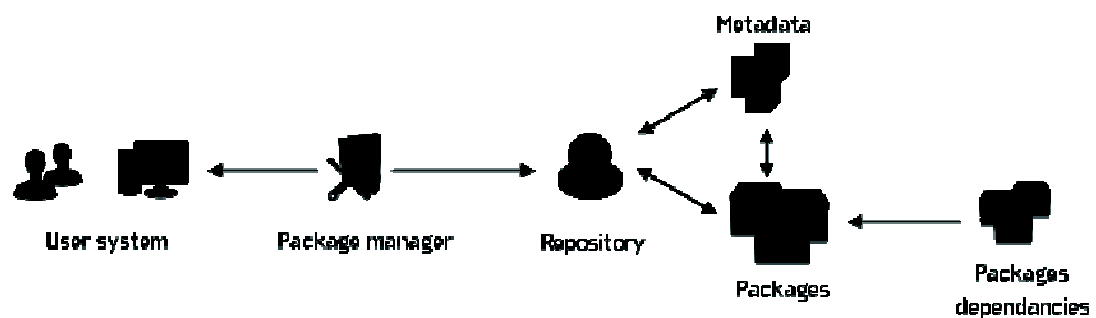


Figure 1: Linkage between user systems, package manager and repository

Package formats

In Linux distribution, the native software comes in three kinds of package formats.

- `tgz` (tar gzip files), which are basically source code archives and can hold anything the package maintainer thinks useful. Apart from the archive format itself, it is necessary to extract the files, but there is nothing standardized about the content of a `tgz` file. They need to be compiled in order to run the software.
- `rpm` (RPM Package Manager), which are pre-compiled archives that are created by Red Hat Linux and standardized by the LSB (Linux Standard Base- to lower the overall costs of supporting the Linux platform)..
- `deb` (Debian) which are pre-compiled archives that are used on Debian based system.

However, if the archives format noticed the system of the required dependencies, they don't provide the dependency management capability and they will just present any encountered problem to the user at first sight, and leave it for user to decide what to do.

For instance if you want to install a RPM package **A** that has dependencies to RPM package **B**, RPM will not automatically install package **B** but just tell you that it needs package **B** and stop. It's up to the user to install package **B** and then afterwards package **A**. Now imagine package **B** has dependencies on package **C** and package **D** and package **D** has dependencies to package **E** and so on and so on. You end up chasing package dependencies manually down all the branches of this very huge tree.

rpm is a powerful package manager, which can be used to build, install, query, verify, update, and erase individual software packages. A package consists of an archive of files and meta-data used to install and erase the archive files. The meta-data includes helper scripts, file attributes, and descriptive information about the package. Packages come in two varieties: binary packages, used to encapsulate software to be installed, and source packages, containing the source code and recipe necessary to produce binary packages.

One of the following basic modes must be selected: Query, Verify, Signature Check, Install/Upgrade/Freshen, Uninstall, Initialize Database, Rebuild Database, Resign, Add Signature, Set Owners/Groups, Show Querytags, and Show Configuration.

Use of rpm with some options

The general form of an rpm install command is

rpm { -il--install } [install-options] PACKAGE_FILE ...

The general form of an rpm upgrade command is

rpm { -Ul--upgrade } [install-options] PACKAGE_FILE ...

This upgrades or installs the package currently installed to a newer version. This is the same as install, except all other version(s) of the package are removed after the new package is installed.

rpm { -Fl--freshen } [install-options] PACKAGE_FILE ...

This will upgrade packages, but only if an earlier version currently exists. The *PACKAGE_FILE* may be specified as an **ftp** or **http** URL, in which case the package will be downloaded before being installed.

2.3 FILE SYSTEM

A file system is a way in which files are stored, accessed, overwritten, and deleted on a media format by using a computer and the operating system (OS) installed onto that computer. Different types of OS typically have different file systems, though they are often somewhat similar in design and how files are accessed through the OS.

Generally, a number of different types of file systems that have been designed and implemented, with new types being created and used by newer OS versions. A file system at its most basic level be defined as the way in which data is stored in individual files on a computer hard drive or other media, and how that data is then accessed again in the future. File system have methods and data structures that an operating system uses to keep track of files on a disk or partition and the way the files are organized on the disk. A computer user will typically work on a computer to create, alter, save, and delete a variety of files for different purposes.

Some file systems are used on local data storage devices; others provide file access via a network protocol (e.g. NFS, SMB, etc). Some file systems are "virtual", in that the "files" supplied are computed on request (e.g. procfs) or are merely a mapping into a different file system used as a backing store. The file system manages access to both the content of files and the metadata about those files. It is responsible for arranging storage space; reliability, efficiency, and tuning with regard to the physical storage medium are important design considerations.

Aspects of file systems

Following are main aspects of file systems:

Space management

File systems allocate space in a granular manner, usually multiple physical units on the device. The file system is responsible for organizing files and directories, and keep track of which areas of the media belong to which file and which are not being used.

File system fragmentation

Fragmentation occurs when unused space or single files are not contiguous. Once file system is built, files are created, modified and deleted. When a file is created, the file system allocates space for the data. Some file systems permit or require specifying an initial space allocation and subsequent incremental allocations as the file grows. If files are deleted, the space reserved for it will be allocated and use by other files. This creates used and unused areas of various sizes. This is called as a free space fragmentation. When a file is created, and there is no contiguous space available for its initial allocation, the space must be assigned in fragments. When a file is modified such that it becomes larger it may exceed the space initially fragmented.

Filenames

A filename is used to identify a storage location in the file system. Most file systems have restrictions on the length of filenames. In some file systems, filenames are case-insensitive (i.e. filenames such as 'ABC' and 'abc' refer to the same file); in others, filenames are case-sensitive (i.e. the names 'ABC' and 'abc' refer to two separate files).

Most modern file systems allow filenames to contain a wide range of characters from the Unicode character set. Most file system interface utilities have restrictions on the use of certain special characters, disallowing them within filenames (the file system may use these special characters to indicate a device, device type, directory prefix, or file type). However, these special characters might be allowed by enclosing the filename with double quotes ("). For simplicity, special characters are generally discouraged within filenames.

Directories

File systems typically have directories (also called folders) which allow the user to group files into separate collections. This may be implemented by associating the file name with an index in a table of contents or an inode in a Unix-like file system. Directory structures may be flat (i.e. linear), or allow hierarchies where directories may contain subdirectories.

Metadata

Metadata is a information that is typically associated with each file within a file system. The length of the data contained in a file may be stored as the number of blocks allocated for the file or as a byte count. The time that the file was last modified may be stored as the file's timestamp. File systems might store the file creation time, the time it was last accessed, the time the file's meta-data was changed, or the time the file was last backed up. Other information includes the file's device type (e.g. block, character, socket, subdirectory, etc.), its owner user ID and group ID, its access permissions and other file attributes (e.g. whether the file is read-only, executable, etc.).

Design limitations

All file systems have some functional limit that defines the maximum storable data capacity within that system. These functional limits are a best-guess effort by the designer based on how large the storage systems are right now and how large storage systems are likely to become in the future. Disk storage has continued to increase at near exponential rates, so after a few years, file systems have kept reaching design limitations that require computer users to repeatedly move to a newer system with ever-greater capacity.

File system complexity typically varies proportionally with the available storage capacity. The file systems of early 1980s home computers with 50 KB to 512 KB of storage would not be a reasonable choice for modern storage systems with hundreds of gigabytes of capacity. Likewise, modern file systems would not be a reasonable choice for these early systems, since the complexity of modern file system structures would consume most or all of the very limited capacity of the early storage systems.

Types of file systems

The following are some of the file system types and classified into disk/tape file systems, network file systems and special-purpose file systems.

Disk file systems

Disk file systems are the file systems which manage data on permanent storage devices. Magnetic disks are the most common of such devices. A disk file system takes advantages of the ability of disk storage media to randomly address data in a short amount of time. In disk files systems, data access is fast. It supports multiple users (or processes) to access various data on the disk. Some of the example disk file systems are FAT (FAT12, FAT16, FAT32), exFAT, NTFS, ext3, ext4, etc.

In a disk file system, there is typically a master file directory, and a map of used and free data regions. Any file additions, changes, or removals require updating the directory and the used/free maps. Random access to data regions is measured in milliseconds so this system works well for disks.

Optical file system

Optical media use different file systems than hard disks or flash media, because of the write-once nature of most optical discs. Even rewritable discs use different file systems because of the way that rewritable media is managed. ISO 9660 and Universal Disk Format (UDF) are two common formats for Compact Discs, DVDs and Blu-ray discs.

Flash file systems

A flash file system is a file system designed for storing files on flash memory devices. These are becoming more common as the number of mobile devices is increasing, the cost per memory size decreases, and the capacity of flash memories increases.

Tape file systems

A tape file system is a file system and tape format designed to store files on tape in a self-describing form. Magnetic tapes are sequential storage media with significantly longer random data access times than disks. Tape requires linear motion to wind and unwind potentially very long reels of media. This tape motion may take several seconds to several minutes to move the read/write head from one end of the tape to the other.

Network file systems

A network file system is a file system that acts as a client for a remote file access protocol, providing access to files on a server. Examples of network file systems include clients for the NFS, AFS, SMB protocols, and file-system-like clients for FTP and WebDAV (Web Distributed Authoring and Versioning). WebDAV is an extension of the Hypertext Transfer Protocol (HTTP) that facilitates collaboration between users in editing and managing documents and files stored on World Wide Web servers.

Shared disk file systems

A shared disk file system is one in which a number of machines (usually servers) have access to the same external disk subsystem (usually a SAN). The file system arbitrates access to that subsystem, preventing write collisions. Examples include GFS2 from Red Hat, GPFS from IBM, etc.

Special file systems

A special file system presents non-file elements of an operating system as files so they can be acted on using file system APIs. This is most commonly done in Unix-like operating systems, but devices are given file names in some non-Unix-like operating systems as well.

Flat file systems

A flat file system is a system of files in which every file in the system must have a different name. Flat file system stores all its files in the same directory. In this system, every file has a different name since there is only one list of files. Flat file systems cannot contain multiple tables.

In Windows 95 and most other operating system today, files are managed in a hierarchical file system with a hierarchy of directories and subdirectories, each containing a number of files (or subdirectories). The operating system allows more than one file to have the same name as long as it is stored in a different directory. Early versions of the Macintosh and DOS operating systems used a flat file system.

File systems and operating systems

Many operating systems require support for more than one file system. Sometimes the OS and the file system are so tightly interlink, due to this, it is difficult to separate out the file system.

There needs to be an interface provided by the operating system software between the user and the file system. This interface can be textual (such as provided by a command line interface, such as the Unix shell) or graphical (such as provided by a graphical user interface, such as file browsers).

Unix-like operating systems

Unix-like operating systems create a virtual file system, which makes all the files on all the devices appear to exist in a single hierarchy. This means, there is one root directory, and every file existing on the system is located under it somewhere. Unix-like systems can use a RAM disk or network shared resource as its root directory. Figure 2 shows a sample Unix directory structure.

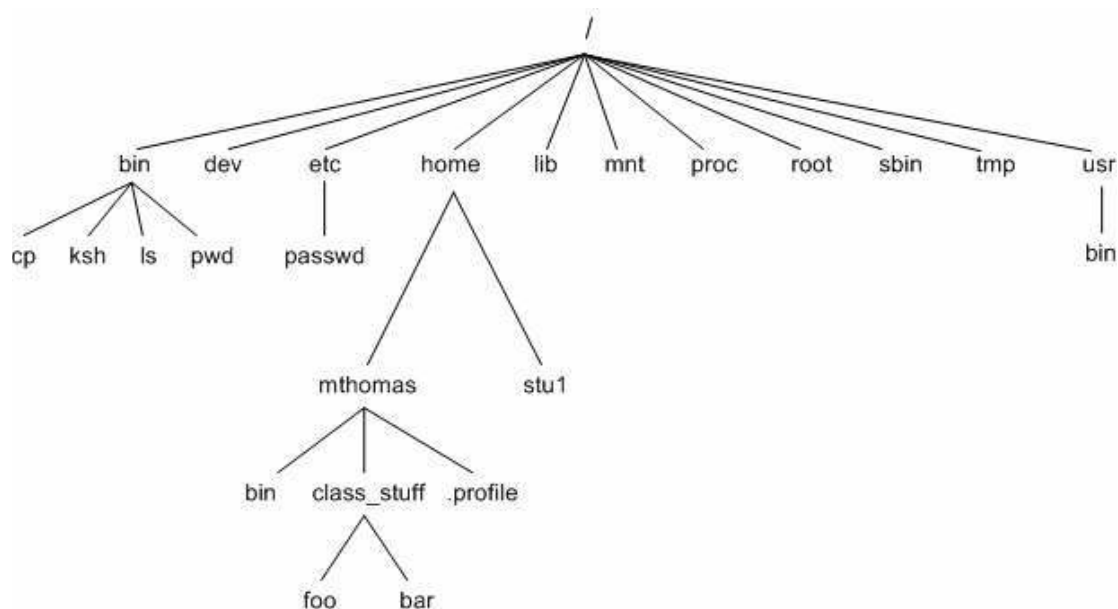


Figure 2: Unix directory Structure

In order to access a file system in UNIX, first it needs to mount it. Mounting a file system is simply making the particular file system accessible at a certain point in the Unix directory tree. When mounting a file system, it does not matter, if the file system is in a hard disk partition, CD-ROM, floppy, or USB storage device. You simply need to know the device name associated with the particular storage device and a directory you would like to mount it to. Unix-like operating systems often include software and tools that assist in the mounting process.

In many situations, file systems other than the root need to be available as soon as the operating system has booted. All Unix-like systems therefore provide a facility for mounting file systems at boot time. System administrators define these file systems in the configuration file *fstab* (*vfstab* in Solaris), which also indicates options and mount points.

In some situations, there is no need to mount certain file systems at boot time, although their use may be desired thereafter. There are some utilities for Unix-like systems that allow the mounting of predefined file systems upon demand.

Removable media have become very common with microcomputer platforms. They allow programs and data to be transferred between machines without a physical connection. Common examples include USB flash drives, CD-ROMs, and DVDs. Utilities have therefore been developed to detect the presence and availability of a medium and then mount that medium without any user intervention.

An automounter is a program or software facility which automatically mounts file systems in response to access operations by user programs. This is usually used for file systems on network servers, rather than relying on events such as the insertion of media, as would be appropriate for removable media.

Linux Operating System

The Linux file system is a hierarchically structured tree where every location has its distinct meaning. Linux supports many different file systems. It is similar to Unix file system. The file system is a tree-shaped structure. The root of the tree, which is called the 'file system root' but is always depicted as being above all other and is

identified by the slash character "/". It is the highest place you can go to and beneath it are almost always only directories. Figure 3 shows the sample directory of a Linux file system.

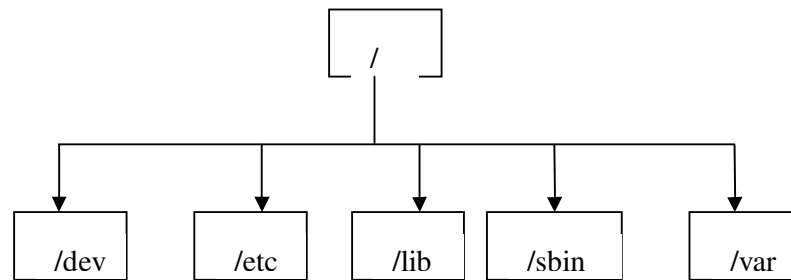


Figure 3: Sample Linux File Structure

The /dev directory contains file system entries which represent devices that are attached to the system. These files are essential for the system to function properly.

The /etc directory is reserved for configuration files that are local to your machine. No binaries are to be put in /etc.

The /lib directory should contain only those libraries that are needed to execute the binaries in /bin and /sbin.

The /proc directory contains special files that either extract information or send information to the kernel.

The /sbin directory is for executables used only by the root user.

The /var directory writes log files

Microsoft Windows File System

Windows makes use of the FAT, NTFS, exFAT and ReFS file systems. Windows uses a *drive letter* abstraction at the user level to distinguish one disk or partition from another. For example, the path C:\WINDOWS represents a directory WINDOWS on the partition represented by the letter C. Drive C is most commonly used for the primary hard disk partition, on which Windows is usually installed and from which it boots. Generally, Drive A and Drive B are used for two floppy drives. This tradition has become so firmly ingrained that bugs exist in many applications which make assumptions that the drive that the operating system is installed on is C.

File system commands

The following are some of the file system commands in Linux

- mount (to mount a file system)

syntax: #mount [-lhV]

- h prints a help message;
- V prints a version string; and just
- l lists all mounted file systems (of type type).

Note: use 'man' utility in Linux to know more on mount command

- umount (to unmount a file system)

syntax : #umount [-hV]

- V Print version and exit.

-h Print help message and exit.

Note: use 'man' utility in Linux to know more on umount command

- df [-hk] (to know file system disk space)

-h print sizes in human readable format (e.g., 1K 234M 2G)

-k prints in kilobytes

Note: use 'man' utility in Linux to know more on df command

- du [-bB] (estimate file space usage)

-b in bytes

-B in Block-size

Note: use 'man' utility in Linux to know more on du command

2.4 MANAGING USERS

The most important responsibilities of any system administrator is managing and monitoring of users. This section explains on how a new user account is created, deleted and other such related in a Linux operating system. The following explains on different types of users.

Users

Generally, users on a system are identified by a username and a userid. The username is something that users would normally refer to, but as far as the operating system is concerned this is referred to using the userid (or uid). The username is typically a user friendly string, such as your name, whereas the userid is a number. The userid numbers should be unique (one number per user).

Special Users

There are some default users on all systems when first installed. Other than the root user however these can normally be disabled from logging in. One should be more careful about deleting usernames as sometimes these are used by different tasks running on the system. Special users normally have names such as sys, bin, adm etc.

The root user has an userid of 0, which has a special meaning. The root user has full permissions to do anything on the system. It is not bound by any of the permissions on the system. There are some tasks that can only be performed by root. To use root permissions, one would normally login under a normal userid and "su" to root or use sudo as required.

Switch User (su)

One of the features of Linux is the ability to change userid when logged into a system. The command 'su' is sometimes referred to as superuser, however this is not completely correct. In the early days of UNIX it was only possible to change to the root user, which made for the superuser command however it is now possible to change to any user using the su command. It is more correct to refer to the command as the switch user command.

The switch user command "su" is used to change between different users on a system, without having to logout. The most common use is to change to the root user, but it can be used to switch to any user depending upon the user's settings. To switch to a different user other than root, then the username is used as the last option on the command.

With regard to user accounts there are three basic types of Linux user accounts: administrative (root), regular, and service.

The Linux administrative root account is automatically created at the time of installation of Linux, and it has administrative privileges for all services on Linux Operating System. The root account is also known as super user. Regular users have the necessary privileges to perform standard tasks on a Linux computer such as running word processors, databases, and Web browsers. Regular users can store files in their own home directories. Since regular users do not normally have administrative privileges, it is not possible to delete critical operating system configuration files. Services such as Apache, Squid, mail, games, and printing have their own individual service accounts. These accounts exist to allow each of these services to interact with your computer.

Each user on a Red Hat Enterprise Linux system is assigned a unique user identification number, also known as a UID. UIDs below 500 are reserved for system users such as the root user and service users.

User group

A user group is a group of one or more users. A user can be a member of more than one group. In Red Hat Enterprise Linux, when a user is added, a private user group (primary group) is created—meaning that a user group of the same name is created and that the new user is the sole user in that group.

The user information is stored in the system files such as `/etc/passwd` and `/etc/shadow` files, and that additionally, group membership information is stored in the `/etc/group` file. The following explains on how user accounts created, modified and deleted with appropriate commands along with examples:

Understand fields in `/etc/passwd`

The password file name in Linux is ‘passwd’ and is stored in `/etc` directory. The path of a password file in Linux is ‘`/etc/passwd`’.

The `/etc/passwd` file stores essential information, which is required during login i.e. user account information. The structure of each entry (record) in a password file is in a seven(7) colon format. The `/etc/passwd` contains one entry per line for each user (or user account) of the system. All fields are separated by a colon (:) symbol. Total seven fields as follows.

Here is an example of a *passwd* file:

root:x:0:0:root:/root:/bin/bash

bin:x:1:1:bin:/bin:/bin/bash

daemon:x:2:2:daemon:/sbin:/bin/bash

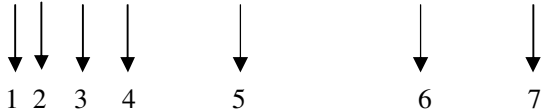
news:x:9:13:News system:/etc/news:/bin/bash

uucp:x:10:14:./var/lib/uucp/taylor_config:/bin/bash

cquoi:x:500:100:Cool.....:/home/cquoi:/bin/bash

Generally, passwd file entry looks like the following:

murli:x:1025:1024:A Murli M Rao:/usr1/murli:/bin/bash



- 1) Username: It is used when user logs in. It should be between 1 and 32 characters in length. (ex. murli)
- 2) Password: An x character indicates that encrypted password is stored in /etc/shadow file.
- 3) User ID (UID): Each user must be assigned a user ID (UID). UID 0 (zero) is reserved for root and UIDs 1-99 are reserved for other predefined accounts. Further UID 100-999 are reserved by system for administrative and system accounts/groups. (ex. 1025)
- 4) Group ID (GID): The primary group ID (stored in /etc/group file) (ex. 1024)
- 5) User ID Info: The comment field. It allow you to add extra information about the users such as user's full name, phone number etc. (ex. A Murli M Rao)
- 6) Home directory: The absolute path to the directory the user will be in when they log in. If this directory does not exists then users directory becomes /(ex. /usr1/murli)
- 7) Command/shell: The absolute path of a command or shell (/bin/bash). Typically, this is a shell. (ex. /bin/bash)

Check Your Progress 1

- 1. What are the common metadata in packages?
.....
.....
.....
.....
.....
.....
- 2. Explain the different parts of a passwd file.
.....
.....
.....
.....
.....
.....

User Administration

Add New User in Linux

Command: **useradd** (Enables a super user or root to create a new user or updates default new user information)

Syntax

Useradd [options] <user-name>

The following is more in details

useradd [-c comment] [-d home_dir] [-e expire_date] [-f inactive_time] [-g initial_group] [-G group,...] [-m [-k skeleton_dir]] [-p passwd] [-s shell] [-u uid [-o]] login

useradd -D [-g default_group] [-b default_home] [-f default_inactive] [-e default_expire_date] [-s default_shell]

Example

#useradd murli (creates a user murli with default shell, home directory, UID, GID automatically)

(or) one can use various options as listed above to have specific values

User Account Modification

Command: **usermod** (Enables a super user or root user to modify a users account)

Syntax

Usermod [options]< user-name>

The following is more in details

usermod [-c comment] [-d home_dir [-m]] [-e expire_date] [-f inactive_time] [-g initial_group] [-G group,...] [-l login_name] [-p passwd] [-s shell] [-u uid [-o]] [-L|-U] login

#usermod -G others admin murli

Delete User account in Linux

Command: userdel (Enables a super user to remove a users account)

Syntax

userdel [-r] login

[-r Files in the user's home directory will be removed along with the home directory itself and the user's mail spool. Files located in other file systems will have to be searched for and deleted manually]

Syntax

Example

#userdel -r murli (the user account 'murli' can be deleted)

Groups Administration in Linux

The file **/etc/group** contains a list of the users who belong to the different groups. As a matter of fact, whenever a large number of users may have access to the system, they are frequently placed in different groups, each of which has its own access rights to the files and directories.

It has different fields that are separated by ":",

group_name : special_field : group_number : member1, member2

Here is an example of a */etc/group* file:

root:x:0:root

bin:x:1:root,bin,daemon

daemon:x:2:

tty:x:5:

disk:x:6:

lp:x:7:

wwwadmin:x:8:

kmem:x:9:

wheel:x:10:

mail:x:12:cyrus

news:x:13:news

Group Creation

A system administrator can manage a group's account. The various tasks that a system administrator can perform include adding, modifying and deleting group account.

Command: **groupadd** (Creates a new group account)

Syntax

groupadd [-g gid [-o]] group

-g The numerical value of the group's ID. This value must be unique, unless the **-o** option (override) is used. The value must be non-negative. The default is to use the smallest ID value greater than 99 and greater than every other group. Values between 0 and 99 are typically reserved for system accounts.

Example

```
#groupadd test1
```

This will create a test1 group

Group Modification

Command: groupmod (Enables a super user or root to modify a group)

Syntax

groupmod [-g gid [-o]] [-n group_name] group

-g gid The numerical value of the group's ID. This value must be unique, unless the **-o** option (override) is used. The value must be non-negative. Values between 0 and 99 are typically reserved for system groups. Any files which the old group ID is the file group ID must have the file group ID changed manually.

-n group_name The name of the group will be changed from group to group_name.

Example

```
#groupmod test1 test2 (This will modify group name test1 to test2)
```

Group Deletion

Command: groupdel (The name of the group you wish to delete.)

Syntax

```
groupdel groupname
```

Example

```
#groupdel test2 (this will delete the test2 group)
```

2.5 SYSTEM AND KERNEL MANAGEMENT

An operating system is the first piece of software that the computer executes when you turn the machine on. The operating system loads itself into memory and begins managing the resources available on the computer. It then provides those resources to other applications that the user wants to execute.

System Management

Once system is booted, the operating system has to perform various tasks such as task scheduling, memory management, disk management , I/O and security management.

The following explains on typical services that an operating system provides.

Task scheduler

The task scheduler is able to allocate the execution of the CPU to a number of different tasks. Some of those tasks are the different applications that the user is running, and some of them are operating system tasks. The task scheduler is the part of the operating system that lets you print a document from your word processor in one window while you are downloading a file in another window and recalculating a spreadsheet in a third window.

Memory manager

The memory manager controls the system's RAM and normally creates a larger virtual memory space using a file on the hard disk.

Disk manager

The disk manager creates and maintains the directories and files on the disk. When you request a file, the disk manager brings it in from the disk.

Network manager

The network manager controls all data moving between the computer and the network.

Other I/O services manager

The OS manages the keyboard, mouse, video display, printers, etc.

The OS maintains the security of the information in the computer's files and controls who can access the computer.

System Management Commands

The following are some of the system management commands:

- `who` (who is logged in)

Syntax: `who [-abd]`

`-a, --all`

`-b, --boot`

time of last system boot

`-d, --dead`

print dead processes

Example: `[murli@imssit ~]$ who`

```
murli pts/0 Jun 2 14:15 (10.10.115.3)
```

```
murli pts/1 Jun 2 15:44 (10.10.115.3)
```

Note: use 'man' utility in Linux to know more on mount command

- `pwd` (displays path of current working directory)

Syntax: `pwd`

Example: `[murli@imssit ~]$ pwd`

```
/home/murli
```

Note: use 'man' utility in Linux to know more on mount command

Other such related command, which are to be used as part of system management are the following:

- `top`, `vmstat`, `ps`, `kill`, `df`, `du`, `mount`, `unmount`, `tar`, `cpio`, `head`, `tail`, `mkdir`, `chdir`, `chown`, `chgrp`, `chmod`, `nice`, `find`, `grep`, etc.

Note: use 'man' utility in Linux to know more on all the above commands

Kernel Management

The kernel is the main component of most computer operating systems. It is a bridge between applications and the actual data processing done at the hardware level. The kernel's responsibilities include managing the system's resources (the communication between hardware and software components). Usually, as a basic component of an operating system, a kernel can provide the lowest-level abstraction layer for the resources (especially processors and I/O devices) that application software must control to perform its function. It typically makes these facilities available to application processes through inter-process communication mechanisms and system calls. Figure 4 shows the kernel and its interactive components.

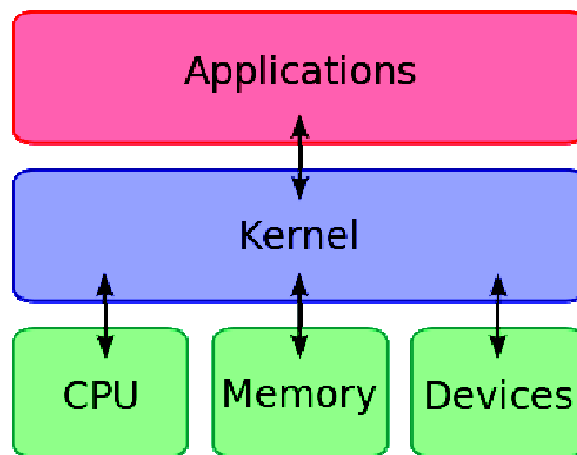


Figure 4: Kernel and its interactive components

Operating system tasks are done differently by different kernels(monolithic and micro kernels), depending on their design and implementation. While monolithic kernels execute all the operating system code in the same address space to increase the performance of the system, micro kernels run most of the operating system services in user space as servers, aiming to improve maintainability and modularity of the operating system. A range of possibilities exists between these two extremes.

The kernel's primary function is to manage the computer's hardware and resources and allow other programs to run and use these resources. Typically, the resources consist of:

The Central Processing Unit

This is the most central part of a computer system, responsible for running or executing programs. The kernel takes responsibility for deciding at any time which of the many running programs should be allocated to the processor or processors (each of which can usually run only one program at a time)

The computer's memory

Memory is used to store both program instructions and data. Typically, both need to be present in memory in order for a program to execute. Often multiple programs will want access to memory, frequently demanding more memory than the computer has available. The kernel is responsible for deciding which memory each process can use, and determining what to do when not enough is available.

Input/Output (I/O)

The I/O devices present in the computer, such as keyboard, mouse, disk drives, USB devices, printers, displays, network adapters, etc. The kernel allocates requests from applications to perform I/O to an appropriate device and provides convenient methods for using the device.

Inter-process communication (IPC)

Kernels usually provide methods for synchronization and communication between processes called inter-process communication (IPC).A kernel may implement these features itself, or rely on some of the processes it runs to provide the facilities to other processes, although in this case it must provide some means of IPC to allow processes to access the facilities provided by each other. Finally, a kernel must provide running programs with a method to make requests to access these facilities.

The Linux kernel allows drivers and features to be compiled as modules rather than as part of the kernel itself. This means that users can often change features in the kernel or add drivers without recompiling, and that the Linux kernel doesn't have to carry a lot of unnecessary baggage.

The following explains on how to see the already loaded in running kernel, how to add modules to the kernel and finally on how to remove modules from kernel.

What is Loaded?

The first utilities that you'll want to get to know are `lsmod` and `modinfo`. Open a terminal and run `lsmod`. Note that you won't need to use `sudo` or log in as root just to probe the modules in the system.

You'll see output like this when you use `lsmod`:

Module	Size	Used by
parport_pc	18855	0
ppdev	5030	0
lp	7462	0
sco	7209	2
parport	27954	3 parport_pc,ppdev,lp
bridge	39630	0
stp	1440	1 bridge
bnep	9427	2

This shows the modules that are loaded, their size, and whether they're being used by other modules. Take the `parport` module, for instance. It's being used by several other modules, but what are they? The `modinfo` utility will tell us. To do so run the following:

Run `modinfo parport`, and you'll see something like this:

```
filename:    /lib/modules/2.6.32-5-amd64/kernel/drivers/parport/parport.ko
license:     GPL
depends:
vermagic:    2.6.32-5-amd64 SMP mod_unload modversions
```

It tells us where the module is found, and its license, and that it has no dependencies.

Removing Modules

Modules can be removed using the `rmmod` utility. The usage is simple, just `rmmod modulename`. However, if we try to remove the `parport` module, we get this error:

```
ERROR: Module parport is in use by parport_pc,ppdev,lp
```

You can force module removal using `rmmod -f`, but that's not a good idea, usually. A better way to do it is to use `modprobe -r` which will automatically look to see what other modules depend on it, and unload those modules as well. If they're in use, then `modprobe` will refuse to remove them as well, unless you use the `-f` option with `modprobe` too.

Installing Modules

What if you have a module you want to load into the kernel? You can do that with `insmod` or `modprobe`.

The preferred method is `modprobe`, because it will also load any modules that the requested module depends on. For instance, if I didn't have the `parport` module loaded and went to load the `lp` or `parport_pc` modules, `modprobe` would go ahead and load `parport` as well.

To load a module using `modprobe` run `modprobe modulename`.

Blacklisting Modules

You may on occasion need to "blacklist" a module. Why would you need this feature? Sometimes a module will cause a conflict with another module, is superseded by another module, or is otherwise undesirable.

To blacklist a module, the easiest way to do it (there's usually more than one way to do things...) is to add the module to `/etc/modprobe.d/blacklist.conf`. For instance, on Debian systems the `evbug` module is automatically blacklisted because it's not something most users will need. To add a module to the blacklist, just add one line to the `blacklist.conf` file:

```
blacklist modulename
```

2.6 BASIC TROUBLESHOOTING

Troubleshooting is a form of problem solving, often applied to repair failed products or processes. It is a logical, systematic search for the source of a problem so that it can be solved, and so the product or process can be made operational again. Troubleshooting is needed to develop and maintain complex systems where the symptoms of a problem can have many possible causes.

The following are some of the commands being used in a Linux environment for problem diagnosis and troubleshooting:

- Use `tail -f` to watch log file in real time, advantage is simple you can spot error or warning message in real time.

Command : **# `tail -f /var/log/maillog`**

- Use `telnet` command to see if you get response or not. Sometime you will also see some informative message:

telnet ip port

Example(s):

```
# telnet localhost 53
# telnet localhost 25
```

- Make sure you can see PID of your service.

pidof service-name

cat /var/run/service.pid

Example(s):

```
# pidof sshd
# cat /var/run/sshd.pid
```

- You need to make sure that your DNS server or third party DNS server (ISP) is accessible. This is an important step, as many network services depend upon DNS; especially sendmail/postfix or Squid etc for example. Run dig or nslookup. No timeout should occur.

```
# dig your-domain.com
# nslookup gw.isp.com
# more /etc/resolv.conf
```

- For networking troubleshooting, make sure your ip address configuration is right, gateway, routing, hostname etc all configured. Here is list of tools on RedHat Linux to verify or modify information:

Hostname verification or setup tools

- **hostname** : To get hostname of server.
- **more /etc/sysconfig/network** : To see hostname and networking details
- **more /etc/hosts** : Make sure at least localhost entry do exist.

Ethernet configuration tools

- **ifconfig** : To see running network card information.
- **ifconfig eth0 up|down** : To enable/disable network interface
- **service network reload|restart|stop|start** : To reload (after changed made in ip config file)|restart|stop|start network interface with all properties.
- **route|netstat -rn** : To print routing table
- **ping ip-address** : To see if host is alive or dead
- **more /etc/modules.conf** : To see your network card configuration alias for eth0 exists or not.
- **lsmod** : To list loaded modules (read as drivers), here you need to see that eth0 module is loaded or not, if not loaded then use insmod to insert (load) driver.
- **dhclient** : Dynamic Host Configuration Protocol Client, run this if your Ethernet card is not getting ip from DHCP box on startup; this command does by default shows useful information.

To see if service blocked because of access control

- **iptables -n -L** : To list all iptable rules; useful to see if firewall blocks service or not.
- **service iptables stop|start** : To start/stop iptables
- **more /etc/xinetd.conf**

OR

- **more /etc/xinetd.conf/SERVICENAME** = To list configuration of xinetd server. Again useful to see if firewall xinetd based security blocks service or not (xinetd includes host-based and time-based access control)
- **more /etc/hosts.allow** : To see list of hosts allowed to access service.
- **more /etc/hosts.deny** : To see list of hosts NOT allowed to access service. **NOTE** first TCP wrappers (hosts.allow/hosts.deny) checked and then xinetd-based access control checked.
- **more /etc/path/to/application.conf** : See your application configuration file for access control. For example smb.conf and many other applications/services got own access control list in application. You need to check that as well.

Some more commands

Getting ram information

```
#cat /proc/meminfo
```

or if you want to get just the amount of ram you can do:

```
#cat /proc/meminfo | head -n 1
```

Getting cpu info

Sometimes in troubleshooting we want to know what processor we are dealing with along with how much cpu is currently being used by our OS and programs. We can do this with these two commands.

```
#cat /proc/cpuinfo
```

```
#top
```

Example :[murli@imssit ~]\$ top

Output

```
top - 14:25:45 up 10 days, 21:07, 1 user, load average: 0.00, 0.05, 0.02
```

```
Mem: 4147692k total, 4124540k used, 23152k free, 57604k buffers
```

```
Swap: 10241396k total, 562856k used, 9678540k free, 2250180k cached
```

PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND
5511	oracle	16	0	1280m	264m	262m	S	0.3	6.5	4:08.74	oracle
5665	oracle	16	0	1280m	314m	311m	S	0.3	7.8	12:13.93	oracle
17154	murli	16	0	2776	1048	772	R	0.3	0.0	0:00.16	top
1	root	16	0	2900	540	464	S	0.0	0.0	0:01.38	init
2	root	RT	0	0	0	0	S	0.0	0.0	0:00.31	migration/0
3	root	34	19	0	0	0	S	0.0	0.0	0:00.00	ksoftirqd/0
4	root	RT	0	0	0	0	S	0.0	0.0	0:00.20	migration/1
5	root	34	19	0	0	0	S	0.0	0.0	0:00.00	ksoftirqd/1
6	root	RT	0	0	0	0	S	0.0	0.0	0:00.22	migration/2
7	root	34	19	0	0	0	S	0.0	0.0	0:00.00	ksoftirqd/2
8	root	RT	0	0	0	0	S	0.0	0.0	0:00.12	migration/3
9	root	34	19	0	0	0	S	0.0	0.0	0:00.00	ksoftirqd/3

```

10 root    5 -10   0   0   0 S 0.0 0.0 0:00.00 events/0
11 root    5 -10   0   0   0 S 0.0 0.0 0:00.00 events/1
12 root    5 -10   0   0   0 S 0.0 0.0 0:00.00 events/2
13 root    5 -10   0   0   0 S 0.0 0.0 0:00.00 events/3
14 root    5 -10   0   0   0 S 0.0 0.0 0:00.00 khelper
15 root    5 -10   0   0   0 S 0.0 0.0 0:00.00 kthread

```

Network Administration Activities

Check out how much hard drive space is left

```
#df -h
```

Example :[murli@imssit ~]\$ df -h

Output

Filesystem	Size	Used	Avail	Use%	Mounted on
/dev/cciss/c0d0p5	16G	2.3G	13G	16%	/
/dev/cciss/c0d0p1	289M	28M	247M	10%	/boot
none	2.0G	0	2.0G	0%	/dev/shm
/dev/cciss/c0d0p7	7.7G	51M	7.3G	1%	/tmp
/dev/cciss/c0d0p2	202G	182G	11G	95%	/u01
/dev/cciss/c0d0p3	29G	21G	6.6G	76%	/u02
/dev/cciss/c0d0p8	4.9G	2.8G	1.9G	60%	/usr

Check out how much disk partitions usage

```
#df -k
```

Example: [murli@imssit ~]\$ df -k

Output

Filesystem	1K-blocks	Used	Available	Use%	Mounted on
/dev/cciss/c0d0p5	16126420	2307900	12999208	16%	/
/dev/cciss/c0d0p1	295561	27863	252438	10%	/boot
none	2073844	0	2073844	0%	/dev/shm
/dev/cciss/c0d0p7	8064272	51480	7603140	1%	/tmp
/dev/cciss/c0d0p2	211663320	190318352	10593068	95%	/u01
/dev/cciss/c0d0p3	30233928	21790480	6907636	76%	/u02
/dev/cciss/c0d0p8	5036284	2854352	1926100	60%	/usr

To estimate file space usame

```
#du
```

Example: [murli@imssit ~]\$ du

Output

```

16  ./kde/Autostart
24  ./kde
16  ./xemacs
104 .

```

Kill a process

```
#ps -A | grep ProgramName
```

```
#kill 7207
```

Show all network connections

```
#netstat
```

```
Example: [murli@imssit ~]$ netstat
```

Output

Active Internet connections (w/o servers)

Proto	Recv-Q	Send-Q	Local Address	Foreign Address	State
tcp	0	0	imssit:64734	imssit:1521	ESTABLISHED
tcp	0	0	imssit:64742	imssit:1521	ESTABLISHED
tcp	0	0	imssit:1521	imssit:32786	ESTABLISHED
tcp	0	0	imssit:1521	imssit:40238	ESTABLISHED

List all files that are currently open on the system

```
#ls -of
```

```
Example: [murli@imssit ~]$ ls -of
```

Output

```
..      .      .bash_logout .bashrc .emacs      .zshrc
.bash_history .kde .gtkrc      .xemacs .bash_profile
```

List all processes running

```
#ps -aef
```

```
Example: [murli@imssit ~]$ ps -aef
```

Output

UID	PID	PPID	C	STIME	TTY	TIME	CMD
root	1	0	0	May22 ?		00:00:01	init [5]
root	2	1	0	May22 ?		00:00:00	[migration/0]
root	3	1	0	May22 ?		00:00:00	[ksoftirqd/0]
root	4	1	0	May22 ?		00:00:00	[migration/1]
root	5	1	0	May22 ?		00:00:00	[ksoftirqd/1]
root	6	1	0	May22 ?		00:00:00	[migration/2]
root	7	1	0	May22 ?		00:00:00	[ksoftirqd/2]
root	8	1	0	May22 ?		00:00:00	[migration/3]
root	9	1	0	May22 ?		00:00:00	[ksoftirqd/3]
root	10	1	0	May22 ?		00:00:00	[events/0]
root	11	1	0	May22 ?		00:00:00	[events/1]
root	12	1	0	May22 ?		00:00:00	[events/2]
root	13	1	0	May22 ?		00:00:00	[events/3]
root	14	1	0	May22 ?		00:00:00	[khelper]

Check Your Progress 2

1. Write the syntax and uses of "useradd" command in Linux.

.....

.....

.....

.....

.....

2. Which command is used to display real-time running tasks in a Linux environment? Explain the significance of this command using an example.

.....

.....

.....

.....

.....

3. How the "Disc Uses" is checked in Linux? Explain with an example.

.....

.....

.....

.....

.....

2.7 SUMMARY

In this unit, the need of a package manager for installation and un-install a package is clearly explained. Different types of file systems and their need and usage was discussed. The unit also covered on how to manage user accounts and also on system and kernel management. Finally the unit explained with some examples on troubleshooting commands.

2.8 ANSWERS TO CHECK YOUR PROGRESS

Check Your Progress 1

1. Packages contains metadata that contains
 - Summary (software name, etc)
 - Description
 - A list of files contained in the package
 - The version of the software it contains as well as the release number of the package
 - When, where and by whom it has been built

- What architecture it has been built for
- Checksums of the files contained in the package
- The license of the software it contains
- Which other packages it requires to work properly

2. Generally, passwd file entry looks like the following:

```
murli:x:1025:1024:A Murli M Rao:/usr1/murli:/bin/bash
```

The diagram shows the passwd file entry with arrows pointing to the following fields:

Field Number	Field Content
1	murli
2	x
3	1025
4	1024
5	A Murli M Rao
6	/usr1/murli
7	/bin/bash

1. **Username:** It is used when user logs in. It should be between 1 and 32 characters in length. (ex. murli)
2. **Password:** An x character indicates that encrypted password is stored in /etc/shadow file.
3. **User ID (UID):** Each user must be assigned a user ID (UID). UID 0 (zero) is reserved for root and UIDs 1-99 are reserved for other predefined accounts. Further UID 100-999 are reserved by system for administrative and system accounts/groups. (ex. 1025)
4. **Group ID (GID):** The primary group ID (stored in /etc/group file) (ex. 1024)
5. **User ID Info:** The comment field. It allow you to add extra information about the users such as user's full name, phone number etc. (ex. A Murli M Rao)
6. **Home directory:** The absolute path to the directory the user will be in when they log in. If this directory does not exists then users directory becomes /(ex. /usr1/murli)
7. **Command/shell:** The absolute path of a command or shell (/bin/bash). Typically, this is a shell. (ex. /bin/bash)

Check Your Progress 2

1. 'useradd' is a command to create a new user in Linux environment

Syntax is

Useradd [options] <user-name>

or

```
useradd [-c comment] [-d home_dir] [-e expire_date] [-f inactive_time] [-g
initial_group] [-G group,...] [-m [-k skeleton_dir]] [-p passwd] [-s shell] [-u uid [ -
o]] login
```

Example:# useradd murli

2. 'top' is a command to display real-time running tasks in a Linux environment

Example :[murli@imssit ~]\$ top

Output

```
top - 14:25:45 up 10 days, 21:07, 1 user, load average: 0.00, 0.05, 0.02
```

```
Mem: 4147692k total, 4124540k used, 23152k free, 57604k buffers
```

```
Swap: 10241396k total, 562856k used, 9678540k free, 2250180k cached
```

```

PID USER   PR  NI  VIRT  RES  SHR  S %CPU %MEM    TIME+  COMMAND
5511 oracle  16   0 1280m 264m 262m S  0.3  6.5   4:08.74 oracle
5665 oracle  16   0 1280m 314m 311m S  0.3  7.8  12:13.93 oracle
17154 murli   16   0 2776 1048  772 R  0.3  0.0   0:00.16 top
   1 root    16   0 2900  540  464 S  0.0  0.0   0:01.38 init
   2 root    RT   0   0   0   0 S  0.0  0.0   0:00.31 migration/0
   3 root    34  19   0   0   0 S  0.0  0.0   0:00.00 ksoftirqd/0
   4 root    RT   0   0   0   0 S  0.0  0.0   0:00.20 migration/1
   5 root    34  19   0   0   0 S  0.0  0.0   0:00.00 ksoftirqd/1

```

3. 'df-k' is a command to know the disk usage

Example: [murli@imssit ~]\$ df -k

Output

```

Filesystem      1K-blocks    Used Available Use% Mounted on
/dev/cciss/c0d0p5 16126420 2307900 12999208 16% /
/dev/cciss/c0d0p1 295561 27863 252438 10% /boot
none            2073844    0 2073844 0% /dev/shm
/dev/cciss/c0d0p7 8064272 51480 7603140 1% /tmp
/dev/cciss/c0d0p2 211663320 190318352 10593068 95% /u01
/dev/cciss/c0d0p3 30233928 21790480 6907636 76% /u02
/dev/cciss/c0d0p8 5036284

```

2.9 FURTHER READINGS

1. Computer Networks by Andrew S Tanenbaum , Fifth Edition
2. SA2, Redhat System Administration I & II, Student Workbook
3. Cisco Certified Network Associate Study Guide, Seventh Edition by Todd Lammle
4. Redhat Enterprise Linux System Administration
5. <http://en.wikipedia.org/wiki/linux>
6. <http://en.wikipedia.org/wiki/kernel>

UNIT 3 NETWORK CONFIGURATION AND SETTING

Structure	Page Nos.
3.0 Introduction	46
3.1 Objectives	46
3.2 Configuring Networks	46
3.3 Dynamic Host Configuration Protocol (DHCP)	47
3.4 Network Domain System (DNS)	50
3.5 Network File System (NFS)	56
3.6 Web Server	63
3.7 Summary	67
3.8 Answers to Check Your Progress	68
3.9 Further Readings	69

3.0 INTRODUCTION

A computer network is a telecommunications network that allows computers to exchange data. The physical connection between networked computing devices is established using either cable media or wireless media. The best-known computer network is the Internet.

Network configuration is an activity to properly configure any network infrastructure through which various network applications/services can be run and accessed. Administrators must be able to configure IP addresses as well as other configuration files w.r.t different network services such as Dynamic Host Configuration Protocol (DHCP), Domain Name System (DNS), Email, Web Servers and other such related network services.

In this unit, you will learn configuration settings of various network services such as Dynamic Host Protocol (DHCP), Domain Name System (DNS) Network File System (NFS) and Web Server

3.1 OBJECTIVES

After going through this unit, you will be able to:

- know how to install various network services;
- configure a Dynamic Host Control Protocol;
- understand and configure a Domain Name System; and
- know on how to configure a Samba server.

3.2 CONFIGURING NETWORKS

A computer network facilitates interpersonal communications, allows sharing of files, and allows sharing of network and computing resources and other such related. To do so, there is an essential need to configure various network services those facilitate for the above network applications.

Network configuration and setup of various services in any organization is a challenging task to configure various network services such as DHCP, DNS, Web Service, Email, etc to run various applications that are to be accessed through network. The following section explains various core and essential network services that are to be required in any organization through which various applications can be run and accessed through an organizational network.

3.3 DYNAMIC HOST CONTROL PROTOCOL

The Dynamic Host Configuration Protocol (DHCP) is a network protocol used to configure devices that are connected to a network. These devices can communicate on that network using the Internet Protocol (IP). It involves clients and a server operating in a client-server model.

Dynamic Host Configuration Protocol (DHCP) automatically assigns IP addresses and other network configuration information (subnet mask, broadcast address, etc) to computers on a network. The DHCP server maintains a database of available IP addresses and configuration information. A client configured for DHCP will send out a broadcast request to the DHCP server requesting an address. The DHCP server will then issue a "lease" and assign it to that client. The time period of a valid lease can be specified on the server. DHCP reduces the amount of time required to configure clients and allows one to move a computer to various networks and be configured with the appropriate IP address, gateway and subnet mask.

How DHCP Works?

The following are the activities between DHCP Server and DHCP Client.

- Lease Request: Client broadcasts request to DHCP server with a source address of 0.0.0.0 and a destination address of 255.255.255.255. The request includes the MAC address which is used to direct the reply.
- IP lease offer: DHCP server replies with an IP address, subnet mask, network gateway, name of the domain, name servers, duration of the lease and the IP address of the DHCP server.
- Lease Selection: Client receives offer and broadcasts to all DHCP servers that will accept given offer so that other DHCP server need not make an offer.
- The DHCP server then sends an acknowledgement to the client. The client is configured to use TCP/IP.
- Lease Renewal: When half of the lease time has expired, the client will issue a new request to the DHCP server.

Download and Install the DHCP Package

Most RedHat and Fedora Linux software product packages are available in the RPM format, whereas Debian and Ubuntu Linux use DEB format installation files. When searching for these packages, remember that the filename usually starts with the software package name and is followed by a version number, for example the file name as 'dhcp-3.23.58-4.i386.rpm'.

Managing the DHCP daemon is easy to do, but the procedure differs between Linux distributions. Here are some things to keep in mind.

- Firstly, different Linux distributions use different daemon management systems. Each system has its own set of commands to do similar operations.
- Secondly, the daemon name needs to be known and in this case the name of the daemon is dhcpd.

dhcpcd.conf File

You can define your server configuration parameters in the dhcpcd.conf file which may be located in the /etc the /etc/dhcpd or /etc/dhcp3 directories depending on your version of Linux.

The dhcpcd.conf configuration file formats in Debian / Ubuntu and Redhat / Fedora are identical.

Here is a quick explanation of the dhcpcd.conf file.

How do you configure DHCP?

The following is the dhcpcd.conf file

ddns-update-style interim

ignore client-updates

subnet 192.168.1.0 netmask 255.255.255.0 {

The range of IP addresses the server
will issue to DHCP enabled PC clients
booting up on the network

range 192.168.1.201 192.168.1.220;

Set the amount of time in seconds that
a client may keep the IP address

default-lease-time 86400;
max-lease-time 86400;

Set the default gateway to be used by
the PC clients

option routers 192.168.1.1;
Don't forward DHCP requests from this
NIC interface to any other NIC
interfaces

option ip-forwarding off;

Set the broadcast address and subnet mask
to be used by the DHCP clients

option broadcast-address 192.168.1.255;
option subnet-mask 255.255.255.0;

Set the NTP server to be used by the
DHCP clients

option ntp-servers 192.168.1.100;

Set the DNS server to be used by the
DHCP clients

option domain-name-servers 192.168.1.100;

If you specify a WINS server for your Windows clients,

you need to include the following option in the dhcpd.conf file:

```
option netbios-name-servers 192.168.1.100;
```

You can also assign specific IP addresses based on the clients'

ethernet MAC address as follows (Host's name is "laser-printer"):

```
host laser-printer {
    hardware ethernet 08:00:2b:4c:59:23;
    fixed-address 192.168.1.222;
}
#
# List an unused interface here
#
subnet 192.168.2.0 netmask 255.255.255.0 {
}
```

DHCP Servers with Multiple NICs

DHCP servers with multiple interfaces pose two configuration challenges. The first is setting up the correct routing and the second is making sure only the required interfaces are listening to serve DHCP.

Routing

When a DHCP configured PC boots, it requests its IP address from the DHCP server. It does this by sending a standardized DHCP broadcast request packet to the DHCP server with a source IP address of 255.255.255.255.

If your DHCP server has more than one interface, you have to add a route for this 255.255.255.255 address so that it knows the interface on which to send the reply; if not, it sends it to the default gateway.

You can temporarily add a route to 255.255.255.255 using the following route add command:

```
root# route add -host 255.255.255.255 dev eth0
```

Create a permanent route to 255.255.255.255. This will vary according to your version of Linux

In Fedora / RedHat ,add the route to your /etc/sysconfig/network-scripts/route-eth0 file, if the route needs to be added to your eth0 interface. The following is an example:

```
#
# vi /etc/sysconfig/network-scripts/route-eth0 and add the following entry and then
save
#
```

```
255.255.255.255/32 dev eth0
```

in Ubuntu / Debian, add the route to your /etc/network/interfaces file. In this case the route is added to the eth0 interface. The following is an example:

```
#
# vi/etc/network/interfaces and add the following and then save
#
iface eth0 inet static
    up route add -host 255.255.255.255 eth0
```

Listening

Once you have defined the interface for your DHCP routing, you should also ensure that your DHCP server only listens on that interface and no others. This methodology to do this varies depending on your version of Linux.

In Fedora / RedHat, the `/etc/sysconfig/dhcpd` file must be edited and the `DHCPDARGS` variable edited to include the preferred interface. For example interface `eth0` is preferred.

```
# vi /etc/sysconfig/dhcpd then add the following and save
```

```
DHCPDARGS=eth1
```

In Debian / Ubuntu the `/etc/default/dhcp3-server` file must be edited and the `INTERFACES` variable edited to include the preferred interface. For example interface `eth0` is preferred.

```
# vi /etc/default/dhcp3-server then add the following and save
```

```
INTERFACES="eth0"
```

You will be able to verify success in one of two ways. First the `netstat` command using the `-au` options will give the list of interfaces listening on the bootp (DHCP) UDP port. The following is an example:

```
root# netstat -au | grep bootp
```

```
udp      0      0 192.168.1.100:bootps  *:*
```

```
root#
```

Secondly, your `/var/log/messages` file will also reveal the following defined interfaces used when the DHCPd daemon was restarted.

```
Jun 8 17:22:44 dhcpd: Listening on LPF/eth0/00:e0:18:5c:d8:41/192.168.1.0/24
```

```
Jun 8 17:22:44 dhcpd: Sending on LPF/eth0/00:e0:18:5c:d8:41/192.168.1.0/24
```

Once, the above messages revealed when DHCPd daemon was started, the configuration is success then go for launch.

Note: Client systems running on Linux/Windows shall be configured to use DHCP.

3.4 DOMAIN NAME SYSTEM (DNS)

A DNS server, or name server, is used to resolve an IP address to a hostname or vice versa.

It is a hierarchical distributed naming system for computers, services, or any resource connected to the internet or a private network. It associates various information with domain names assigned to each of the participating entities. Most prominently, it translates easily memorized domain names to the numerical IP addresses needed for the purpose of locating computer services and devices worldwide.

By setting up a DNS server, you become part of a hierarchy of DNS servers that make up the internet. At the top of this hierarchy is the root server, represented by a dot (“.”) below the root server are the top level domains (such as .com,org,and so on).

The basic function of name server is to answer queries by providing the information that those queries request. A DNS name server primarily translates domain and host names into IP addresses. Each domain is typically represented by a least two DNS servers. The following are different types of DNS servers:

- **Primary (master) name server** contains authoritative information about the domains that it serves. In response to queries for information about its domains, this server provides that information marked as being authoritative. The primary is the ultimate source for data about the domain. The secondary name server only carries the same authority in that it has received and loaded a complete set of domain information from the primary.
- **Secondary (slave) name server** gets all information for the domain from the primary. As is the case for the primary, DNS considers the secondary information about the domain that it serves authoritative.
- **Caching name server** simply caches the information it receives about the locations of hosts and domains. It holds information that it obtains from other authoritative servers and reuses that information until the information expires.
- **Forwarding name server** is essentially a caching name server but is useful in cases where computers lie behind a firewall and in which only one computer can make DNS queries outside that firewall on behalf of all the internal computers.

Understanding Bind

Red Hat Linux (and most other Linux and UNIX systems) implement DNS services by using the Berkeley Internet Name Domain (BIND) software. The Internet software Consortium maintains BIND (at www.isc.org/products/BIND). The basic components of BIND include the following:

- **DNS server daemon** (`/usr/sbin/named`): the named daemon listens on a port for DNS services requests and then fulfills those requests based on information in the configuration files that you create. Mostly named receives requests to resolve the host names in your domain to IP address.
- **DNS configuration files** (`named.conf` and `/var/named/*`): the `/etc/named.conf` file is where you add most of the general configuration information that you need to define the DNS services for your domain. Separate files in the `/var/named` directory contain specific zone information.
- **DNS lookup tools** to check that your DNS server is resolving host names properly. These include commands such as `host`, `dig`, and `nslookup` (which are part of the `bind-utils` software package).

To maintain your DNS server correctly, you can also perform the following configuration tasks with your DNS server:

Logging indicates what you want to log and where log files reside.)

Remote server options can set options for specific DNS servers to perform such tasks as blocking information from a bad server, setting encryption keys to you use with a server, or defining transfer methods)

There is no need to give out DNS information to everyone who requests it. Restrict access to those who request it based on the following.

Access control list can contain those hosts, domains or IP addresses that one wants to group together and apply the same level of access to DNS server. C acl records to group those addresses, and then indicate what domain information the locations in that acl can or can't access.

Listen-on ports by default, name server accepts only name server requests that come to port 53 on name server. You can add more port numbers if you want your name server to accept name-service queries on different ports.

Authentication is to verify the identities of hosts that are requesting services from DNS server, can use keys for authentication and authorization. (the key and trusted-keys statements are used for authentication.)

Quick-starting A DNS Server

The DNS server software that comes with the current Red Hat Linux is Berkeley Internet name Daemon (BIND). The following components are required to configure BIND.

- **Configuration file** (/etc/named.conf) is the main DNS server configuration file.
- **Zone directory** (/var/named) is the directory containing files that keep information about the zone that you create for your DNS server.
- **Daemon process** (/usr/sbin/named) is the daemon process that listens for DNS requests and responds with information that the named.conf file presents.
- **Debugging tools** (named -checkconf, and named-checkzone) are to determine whether the created DNS configuration correctly.

The following are the points one has to keep it in mind in creating a DNS server:

- identifying your DNS servers
- creating DNS configuration files (named.conf and /var/named/*)
- starting the named daemon
- Monitoring named activities

Configure DNS Server

The following is a step by step procedure to configure a master DNS server.

For this activity, use three systems- one Linux server, second Linux clients and third window clients.

Step 1 - bind and caching-nameserver rpm is required to configure DNS. Check them for install, if not found then install

```
[root@Server ~]# rpm -qa bind*
bind-libs-9.3.3-10.el5
bind-chroot-9.3.3-10.el5
bind-devel-9.3.3-10.el5
bind-utils-9.3.3-10.el5
bind-libbind-devel-9.3.3-10.el5
bind-9.3.3-10.el5
bind-sdb-9.3.3-10.el5
[root@Server ~]# rpm -qa cach*
caching-nameserver-9.3.3-10.el5
cachefilesd-0.8-2.el5
[root@Server ~]# _
```

Step 2 - set hostname to server.example.com and ip address to 192.168.0.254

```
[root@Server ~]# cat /etc/sysconfig/network
NETWORKING=yes
NETWORKING_IPV6=no
HOSTNAME=Server.example.com

[root@Server ~]# ifconfig eth0
eth0      Link encap:Ethernet  HWaddr 00:0C:29:11:AD:E1
          inet addr:192.168.0.254  Bcast:192.168.0.255  Mask:
          inet6 addr: fe80::20c:29ff:fe11:ade1/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:99 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:0 (0.0 b)  TX bytes:17981 (17.5 KiB)
          Interrupt:67 Base address:0x2000
```

Main configuration file for dns server is named.conf. By default, this file is not created in /var/named/chroot/etc/ directory. Instead of named.conf, a sample file /var/named/chroot/etc/named.caching-nameserver.conf is created. This file is used to make a caching only name server. You can also do editing in this file after changing its name to named.conf to configure master dns server or you can manually create a new named.conf file.

In our example, create a new named.conf file

```
[root@Server etc]# vi /var/named/chroot/etc/named.conf _
```

Or do editing exactly as shown here in image

```
options{
    directory "/var/named/";
};

zone "example.com" {
    type master;
    file "example.com.zone";
    allow-transfer {192.168.0.1};
};

zone "0.168.192.in-addr.arpa" {
    type master;
    file "0.168.192.in-addr.arpa.zone";
};
```


save this file with :wq and exit

Configure zone file

Defined two zone files example.com.zone for forward zone and 0.168.192.in-addr.arpa for reverse zone. These files will be stored in /var/named/chroot/var/named/ location.

Use two sample files for creating these files.

```
[root@Server named1# cd /var/named/chroot/var/named
[root@Server named1# cp localhost.zone example.com.zone
[root@Server named1# cp named.localhost.168.192.in-addr.arpa.zone
[root@Server named1# _
[root@Server named1# vi example.com.zone _
```

By default this file will look like this

```
$TTL      86400
@         IN SOA  @         root (
                                42      ; serial
                                3H      ; refresh
                                15M     ; retry
                                1W      ; expiry
                                1D )    ; minimum

                                IN NS   @
                                IN A    127.0.0.1
                                IN AAAA  ::1
```

Change this file exactly as shown in image below

```
$TTL      86400
@         SOA      example.com.  root (
                                42      ; serial
                                3H      ; refresh
                                15M     ; retry
                                1W      ; expiry
                                1D )    ; minimum

@         NS       server.example.com.
@         NS       client1.client.com.
server    A        192.168.0.254
client1   A        192.168.0.1
client2   A        192.168.0.2
```

Now open reverse lookup zone file 0.168.192.in-addr.arpa

By default it will look like this

```
$TTL      86400
@          SOA      example.com. root.server.example.com. (
                                1997022700 ; Serial
                                28800      ; Refresh
                                14400      ; Retry
                                3600000    ; Expire
                                86400 )    ; Minimum

                IN      NS      server.example.com
254           IN      PTR      server.example.com.
1             IN      PTR      client1.example.com.
2             IN      PTR      client2.
```

Change this file exactly as shown in image below

```
$TTL      86400
@          SOA      example.com. root.server.example.com. (
                                1997022700 ; Serial
                                28800      ; Refresh
                                14400      ; Retry
                                3600000    ; Expire
                                86400 )    ; Minimum

                IN      NS      server.example.com
254           IN      PTR      server.example.com.
1             IN      PTR      client1.example.com.
2             IN      PTR      client2.
```

Now changed the ownership of these zone files to named group

```
[root@Server named]# chgrp named example.com.zone
[root@Server named]# chgrp named 0.168.192.in-addr.arpa.zone
[root@Server named]# _
```

Now start the named service

```
[root@Server named]# chkconfig named on
[root@Server named]# service named restart
Stopping named: [ OK ]
Starting named: [ OK ]
[root@Server named]# _
```

If service restart without any error, it means successfully configured master name server.

Check Your Progress 1

1. Discuss the activities between DHCP Server and DHCP Client.

.....

.....

.....

.....

2. What are the different types of DNS servers? Explain.

.....

.....

.....

.....

.....

3.5 NETWORK FILE SYSTEM

Network File System (NFS) is a server-client protocol for sharing files between computers on a common network. It allows a computer to access directories on remote computers by mounting them on a local file system as if they were a local disk. The administrator on the NFS server has to define the directories that need to be activated, or exported, for access by the NFS clients, and administrators on the clients need to define both the NFS server and the subset of its exported directories to use. It is available on a variety of UNIX based operating systems, not just Linux. The server and client do not have to use the same operating system. The client system just needs to be running an NFS client compatible with the NFS server.

General rules to be followed

One should follow some general rules when configuring NFS.

- Only export directories beneath the / directory.
- Do not export a subdirectory of a directory that has already been exported. The exception being when the subdirectory is on a different physical device. Likewise, do not export the parent of a subdirectory unless it is on a separate device.
- Only export local filesystems.

When mounting any filesystem on a directory, one should know that the original contents of the directory are ignored, or obscured, in favor of the files in the mounted filesystem. When the filesystem is unmounted, then the original files in the directory reappear unchanged.

Some NFS key concepts

The following are some key NFS background concepts that will help in overall understanding.

The Virtual File System (VFS)

VFS interface is the mechanism used by NFS to redirect all access to NFS-mounted files to the remote server. It is done in such a way that files on the remote NFS server appear to the user like those on a local disk.

Stateless Operation

Programs that read and write to files on a local filesystem rely on the operating system to track their access location within the file with a pointer. As NFS is a network-based file system, and networks can be unreliable, it was decided that the NFS client daemon would act as a failsafe intermediary between regular programs running on the NFS client and the NFS server.

Normally, when a server fails, file accesses timeout and the file pointers are reset to zero. With NFS, the NFS server doesn't maintain the file pointer information, the NFS client does. This means that if an NFS server suddenly fails, the NFS client can precisely restart the file access once more after patiently waiting until the server returns online.

Caching

NFS clients typically request more data than they need and cache the results in memory locally so that further sequential access of the data can be done locally instead of access from server. This is also known as a read ahead cache. Caching therefore helps to reduce the amount of network traffic while simultaneously improving the speed of some types of data access. The NFS server caches information too, such as the directory information for the most recently accessed files and a read ahead cache for recently read files.

NFS and Symbolic links

One has to be careful with the use of symbolic links on exported NFS directories. If an absolute link points to a directory on the NFS server that hasn't been exported, then the NFS client won't be able to access it. Mounting a filesystem on a symbolic link actually mounts the filesystem on the target of the symbolic link. Plan carefully before doing this.

NFS Background mounting

NFS clients use remote procedure call (RPC) suite of network application helper programs to mount remote filesystems. If the mount cannot occur during the default RPC timeout period, then the client retries the mount process until the NFS number of retries has been exceeded. The default is 10,000 minutes, which is approximately a week. The difficulty here is that if the NFS server is unavailable, the mount command will hang for a week until it returns online.

Hard and Soft mounts

The process of continuous retrying, whether in the background or foreground, is called a hard mount. NFS attempts to guarantee the consistency of data with these constant retries. With soft mounts, repeated RPC failures cause the NFS operation to fail not hang and data consistency is therefore not guaranteed. The advantage is that the operation completes quickly, whether it fails or not. In such case it is better not to place critical data that needs to be updated regularly or executable programs in such a location. NFS has currently various versions such as version 2,3 and 4. Version 1 was a prototype.

NFS Daemons

NFS isn't a single program, but a suite of interrelated programs that work together to get the job done. The following are several daemons that are started when a system goes into run level 3 or multi-user mode. The mountd and nfsd daemons are run on systems that are servers. The automatic startup of the server daemons depends on the existence of entries that are labeled with the NFS file-system type in /etc/dfs/sharetab. To support NFS file locking, the lockd and statd daemons are run on NFS clients and servers.

i) automountd Daemon

This daemon handles the mounting and unmounting requests from the autofs service. The syntax of the command is as follows:

```
automountd [ -Tnv ] [ -D name=value ]
```

The command behaves in the following ways:

-T enables tracing.

-n disables browsing on all autofs nodes.

-v selects to log all status messages to the console.

-D name=value substitutes *value* for the automount map variable that is indicated by *name*.

The default value for the automount map is `/etc/auto_master`. Use the **-T** option for troubleshooting

ii) *lockd Daemon*

This daemon supports record-locking operations on NFS files. The lockd daemon manages RPC connections between the client and the server for the Network Lock Manager (NLM) protocol.

iii) *mountd Daemon*

This daemon handles file-system mount requests from remote systems and provides access control. The mountd daemon checks `/etc/dfs/sharetab` to determine which file systems are available for remote mounting and which systems are allowed to do the remote mounting. One can use the **-v** option and the **-r** option with this command.

The **-v** option runs the command in verbose mode. Every time an NFS server determines the access that a client should be granted, a message is printed on the console. The information that is generated can be useful when trying to determine why a client cannot access a file system.

The **-r** option rejects all future mount requests from clients. This option does not affect clients that already have a file system mounted.

iv) *nfs4cbd Daemon*

This daemon is for the exclusive use of the NFS version 4 client that manages the communication endpoints for the NFS version 4 callback program. The daemon has no user-accessible interface.

v) *nfslogd Daemon*

This daemon provides operational logging. NFS operations that are logged against a server are based on the configuration options that are defined in `/etc/default/nfslogd`. When NFS server logging is enabled, records of all RPC operations on a selected file system are written to a buffer file by the kernel

vi) *statd Daemon*

This daemon works with lockd daemon to provide crash and recovery functions for the lock manager. The statd daemon tracks the clients that hold locks on an NFS server. If a server crashes, on rebooting, statd daemon on the server contacts statd on the client. The client statd can then attempt to reclaim any locks on the server. The client statd also informs the server statd when a client has crashed so that the client's locks on the server can be cleared.

vii) *rpcbind: (portmap in older versions of Linux)*

The primary daemon upon which all the others rely, rpcbind manages connections for applications that use the RPC specification. By default, rpcbind listens to TCP port 111 on which an initial connection is made. This is then used to negotiate a range of TCP ports, usually above port 1024, to be used for subsequent data transfers. You need to run rpcbind on both the NFS server and client.

viii) nfs

Starts the RPC processes needed to serve shared NFS file systems. The nfs daemon needs to be run on the NFS server only.

ix) nfslock

Used to allow NFS clients to lock files on the server via RPC processes. The nfslock daemon needs to be run on both the NFS server and client.

x) netfs

It allows RPC processes run on NFS clients to mount NFS filesystems on the server. The netfs daemon needs to be run on the NFS client only.

Installing NFS

RedHat Linux installs nfs by default, and nfs is also activated when the system boots. One can check whether nfs installed or not using the RPM command in conjunction with the grep command to search for all installed nfs packages.

The following is an example to check:

```
[root# rpm -qa | grep nfs
redhat-config-nfs-1.1.3-1
nfs-utils-1.0.1-3.9
[root#
```

A blank list means that there is a need to install the required packages.

There is also a need to have the RPC rpcbind package installed, and the rpm command can tell whether it is installed or not. Use rpm in conjunction with grep, to check all the rpcbind applications installed or not.

The following is an example:

```
[root# rpm -q rpcbind
rpcbind-4.0-57
[root#
```

A blank list means that there is a need to install the required packages.

If nfs and rpcbind are not installed, they can be added fairly easily once find the nfs-utils and rpcbind RPMs. Remember that RPM filenames usually start with the software's name and a version number, as in nfs-utils-1.1.3-1.i386.rpm.

A Typical NFS Scenario

A small office has an old Linux server that is running out of disk space. The office cannot tolerate any down time, even after hours, because the server is accessed by overseas programmers and clients at nights and local ones by day. Budgets are tight and the company needs a quick solution until it can get a purchase order approved for a hardware upgrade. Another Linux server on the network has additional disk capacity (for ex. at /data partition) and the office would like to expand into it as an interim expansion NFS server.

Configuring NFS on The Server

Both the NFS server and NFS client have to have parts of the NFS package installed and running. The server needs rpcbind, nfs, and nfslock operational, as well as a correctly configured /etc/exports file.

The following explains how exports file is to be configured:

The /etc/exports File

The /etc/exports file is the main NFS configuration file, and it consists of two columns. The first column lists the directories you want to make available to the network. The second column has two parts. The first part lists the networks or DNS domains that can get access to the directory, and the second part lists NFS options in brackets.

For example for the scenario as mentioned above, suppose requires the following:

Read-only access to the /data/files directory to all networks

Read/write access to the /home directory from all servers on the xxx.xxx.1.0 /24 network, which is all addresses from xxx.xxx.1.0 to xxx.xxx.1.255

Read/write access to the /data/test directory from servers in the my-site.com DNS domain

Read/write access to the /data/database directory from a single server xxx.xxx.1.203.

In all cases, use the sync option to ensure that file data cached in memory is automatically written to the disk after the completion of any disk data copying operation.

#/etc/exports

```
/data/files      *(ro,sync)
/home            xxx.xxx.1.0/24(rw,sync)
/data/test       *.my-site.com(rw,sync)
/data/database   xxx.xxx.1.203/32(rw,sync)
```

After configuring /etc/exports file, there is a need to activate the settings, but first make sure that NFS is running correctly.

Starting NFS on the Server

Configuring an NFS server is straightforward:

Use the chkconfig command to configure the required nfs and RPC rpcbind daemons to start at boot and also activate NFS file locking to reduce the risk of corrupted data.

The following is the example:

```
root@# chkconfig nfs on
root@# chkconfig nfslock on
root@# chkconfig rpcbind on
```

Use the init scripts in the /etc/init.d directory to start the nfs and RPC rpcbind daemons.

The following are examples to use the start option, but by using stop and restart options one can stop or restart the service as when needed.

```
root# service rpcbind start
root# service nfs start
root# service nfslock start
```

```
root# service rpcbind stop
root# service rpcbind restart
```

Test whether NFS is running correctly with the rpcinfo command or not. The following I an example to list the running RPC programs.

```
root# rpcinfo -p localhost

program vers proto  port
100000  2  tcp   111  portmapper
100000  2  udp   111  portmapper
100003  2  udp  2049  nfs
100003  3  udp  2049  nfs
100021  1  udp  1024  nlockmgr
100021  3  udp  1024  nlockmgr
100021  4  udp  1024  nlockmgr
100005  1  udp  1042  mountd
100005  1  tcp  2342  mountd
100005  2  udp  1042  mountd
100005  2  tcp  2342  mountd
100005  3  udp  1042  mountd
100005  3  tcp  2342  mountd
```

```
root#
```

Accessing NFS Server Directories from the Client

In most cases, users want their NFS directories to be permanently mounted. This requires an entry in the /etc/fstab file in addition to the creation of the mount point directory.

The /etc/fstab File

The /etc/fstab file lists all the partitions that need to be auto-mounted when the system boots. Edit the /etc/fstab file for NFS directory to be made permanently available to users on the NFS.

For example, mount the /data/files directory on server test(IP address xxx.xxx.1.100) as NFS-type filesystem using the local /mnt/nfs mount point directory.

```
#/etc/fstab
```

#Directory	Mount Point	Type	Options	Dump	FSCK
xxx.xxx.1.100:/data/files	/mnt/nfs	nfs	soft,nfsvers=2	0	0

This example is used the soft and nfsvers options; The table 1 outlines these and other useful NFS mounting options:

Table 1: NFS Option

Option	Description
bg	Retry mounting in the background if mounting initially fails
fg	Mount in the foreground
soft	Use soft mounting
hard	Use hard mounting
rsiz=n	The amount of data NFS will attempt to access per read operation. The default is dependent on the kernel. For NFS version 2, set it to 8192 to assure maximum throughput.
wsiz=n	The amount of data NFS will attempt to access per write operation. The default is dependent on the kernel. For NFS version 2, set it to 8192 to assure maximum throughput.
nfsvers=n	The version of NFS the mount command should attempt to use
tcp	Attempt to mount the filesystem using TCP packets: the default is UDP.
intr	If the filesystem is hard mounted and the mount times out, allow for the process to be aborted using the usual methods such as CTRL-C and the kill command.

Configuring NFS on the Client

NFS configuration on the client requires to start the NFS application; create a directory on which to mount the NFS server's directories that are exported via the /etc/exports file, and finally to mount the NFS server's directory on local system's directory, or mount point.

Starting NFS on the Client

Use the following steps to configure NFS on the client.

Use the chkconfig command to configure the required nfs and RPC rpcbind daemons to start at boot. Activate nfslock to lock the files and reduce the risk of corrupted data.

```
root# chkconfig netfs on
root# chkconfig nfslock on
root# chkconfig rpcbind on
```

Use the init scripts in the /etc/init.d directory to start the nfs and RPC rpcbind daemons. Use the following example les to start, stop and restart the processes:

```
root# service rpcbind start
root# service netfs start
```

```
root# service nfslock start
root# service rpcbind stop
root# service rpcbind restart
```

Test NFS whether is running correctly or not with the rpcinfo command. The following is an example:

```
.
root# rpcinfo -p
program vers proto  port
100000  2  tcp   111  portmapper
100000  2  udp   111  portmapper
100024  1  udp  32768  status
100024  1  tcp  32768  status
100021  1  udp  32769  nlockmgr
100021  3  udp  32769  nlockmgr
100021  4  udp  32769  nlockmgr
100021  1  tcp  32769  nlockmgr
100021  3  tcp  32769  nlockmgr
100021  4  tcp  32769  nlockmgr
391002  2  tcp  32770  sgi_fam
root#
```

3.6 WEB SERVER

The primary function of a web server is to cater web page to the request of clients using the Hypertext Transfer Protocol (HTTP). This means delivery of HTML documents and any additional content that may be included by a document, such as images, style sheets and scripts. The server that sends your web browser the code to display a web page is called a web server. There are countless web servers all over the Internet serving countless websites to people all over the world. Whether you need a web server to host a website on the Internet a Red Hat Enterprise Linux server can function as a web server using the Apache HTTP server. The Apache HTTP server is a popular, open source server application that runs on many UNIX-based systems as well as Microsoft Windows.

A user agent, commonly a web browser initiates communication by making a request for a specific resource using HTTP and the server responds with the content of that resource or displays an error message, if not available. The resource is typically a real file on the server's secondary storage, but this is not necessarily the case and depends on how the web server is implemented.

Samba Server

Samba is a software package that comes with Red Hat Linux to share file systems and printers on a network with computers that use the session message block (SMB) protocol. SMB is the protocol that is delivered with windows operating systems for sharing files and printers. In Red Hat Linux, the Samba software package contains a variety of daemon processes, administrative tools, user tools, and configuration files.

The default Samba configuration file is `smb.conf`, which is in `/etc/samba` directory (`/etc/samba/smb.conf`). If you need to access features that are not available through the samba server configuration file you can edit `/etc/samba/smb.conf` file as required.

Daemon processes consist of `smbd` (the SMB daemon) and `nmbd` (the NetBIOS name server). The `smbd` is what makes the file sharing and printing services you add to your Red Hat Linux computer available to windows client computers. The following are some of the clients that Samba supports:

Window 9X

Window 2000

Window NT

Window ME

Window XP

Window for workgroups

Ms Client 3.0 for DOS

OS/2

Dave for Macintosh computer

Samba for Linux

As for administrative tools for samba, you have several shell commands at your disposal. You can check your configuration file using the `testparm` and `testprns` commands. The `smbstatus` command tells you which computers are currently connected to your shared

Samba configuration file (smb.conf)

```
# smb.conf is the main Samba configuration file. You find a full commented
# version at /usr/share/doc/packages/samba/examples/smb.conf.SUSE if the
# samba-doc package is installed.
```

```
# Date: 2008-05-28
```

```
[global]
```

```
workgroup = WORKGROUP
printing = cups
printcap name = cups
printcap cache time = 750
cups options = raw
map to guest = Bad User
include = /etc/samba/dhcp.conf
logon path = \\%L\profiles\msprofile
logon home = \\%L%\U\9xprofile
logon drive = P:
usershare allow guests = Yes
add machine script = /usr/sbin/useradd -c Machine -d /var/lib/nobody -s /bin/false
%m$
```

```
domain logons = No
domain master = No
security = user
usershare max shares = 100
```

```
[homes]
```

```
comment = Home Directories
valid users = %S, %D%w%S
browseable = No
read only = No
inherit acls = Yes
```

```

[profiles]
comment = Network Profiles Service
path = %H
read only = No
store dos attributes = Yes
create mask = 0600
directory mask = 0700
[users]
comment = All users
path = /home
read only = No
inherit acls = Yes
veto files = /aquota.user/groups/shares/
[groups]
comment = All groups
path = /home/groups
read only = No
inherit acls = Yes
[printers]
comment = All Printers
path = /var/tmp
printable = Yes
create mask = 0600
browseable = No
[print$]
comment = Printer Drivers
path = /var/lib/samba/drivers
write list = @ntadmin root
force group = ntadmin
create mask = 0664
directory mask = 0775

## Share disabled by YaST
# [netlogon]

[public]
comment = RAID drive share
inherit acls = Yes
path = /local/public
read only = No

```

How to configure Samba server

The following is the step by step procedure to configure a Samba Server

1. Server IP address is xxx.xxx.0.254 and machine name is Server1
2. Windows machine IP address is xxx.xxx.0.2 and machine name is client2
3. Firewall Should be disabled.

```

#chkconfig iptables off
#service iptables stop
#chkconfig ip6tables off
#service ip6tables stop

```

4. To check if all rpms required for samba are installed or not.

```
#rpm -qa | grep samba, then it displays the following
Samba-<version-name>
Samba-common-<version-name>
Samba-client-<version-name>
System-config-samba-<version-name>
```

5. If not, install it using

```
#rpm -ivh samba
```

6. Copy the original configuration file as smb.conf.bck

```
#cp /etc/samba/smb.conf /etc/samba.conf.bck
(This will keep a backup copy of your configuration file.)
```

7. Now edit the configuration file

```
#vi /etc/samba/smb.conf
```

8. To share home directory edit this part of /etc/samba/smb.conf

```
[homes]
comment=Home directories
browseable=no
writable=yes
```

- 9.To share printer edit this part of /etc/samba/smb.conf

```
[printers]
comment = All printers
path = /var/spool/samba
browseable = no
```

- 10.To configure share called IHNC SHARE edit this part of /etc/samba/smb.conf

```
[IHNC's share]
comment = Testing Samba Server
path = /samba
valid users = user1, user2
```

11. Save the file with wq command

12. To check your configuration file type testparm

```
#testparm
```

13. Create two samba users user1 and user2

```
useradd user1
useradd user2
```

14. Create smbpassword for both the users

```
smbpasswd -a user1
smbpasswd -a user2
```

15. Now you stop/start the samba services

```
#service smb stop  
#service smb start
```

16. To see what is shared from your server through samba for a particular user

```
#smbclient -L Server1 -U user1 or smbclient -L //xxx.xxx.0.1 -U user1
```

17. To see what is shared for a particular host

```
#smbclient -L mac2
```

18. Now, permanently make samba server on

```
#chkconfig smb on
```

Check Your Progress 2

1. List the components required to configure BIND.

.....

.....

.....

.....

.....

.....

.....

2. What is Samba Server? Explain its importance.

.....

.....

.....

.....

.....

.....

3.7 SUMMARY

In this unit, installation, configuration and setup of various network services such as Dynamic Host Control Protocol (DHCP), Domain Name System (DNS), Network File System (NFS) and Samba server are explained in detail with examples. This knowledge may help to understand the concepts and install, configure and commissioning of other network services such as Email, FTP and such related services also. Student has to practice in real time to have more exposure and built confidence in configuration of network services.

3.8 ANSWERS TO CHECK YOUR PROGRESS

Check Your Progress 1

1. The following are the activities between DHCP Server and DHCP Client.
 - Lease Request: Client broadcasts request to DHCP server with a source address of 0.0.0.0 and a destination address of 255.255.255.255. The request includes the MAC address which is used to direct the reply.
 - IP lease offer: DHCP server replies with an IP address, subnet mask, network gateway, name of the domain, name servers, duration of the lease and the IP address of the DHCP server.
 - Lease Selection: Client receives offer and broadcasts to all DHCP servers that will accept given offer so that other DHCP server need not make an offer.
 - The DHCP server then sends an acknowledgement to the client. The client is configured to use TCP/IP.
 - Lease Renewal: When half of the lease time has expired, the client will issue a new request to the DHCP server.
2. The following are different types of DNS servers:

Primary (master) name server contains authoritative information about the domains that it serves. In response to queries for information about its domains, this server provides that information marked as being authoritative. The primary is the ultimate source for data about the domain. The secondary name server only carries the same authority in that it has received and loaded a complete set of domain information from the primary.

Secondary (slave) name server gets all information for the domain from the primary. As is the case for the primary, DNS considers the secondary information about the domain that it serves authoritative.

Caching name server simply caches the information it receives about the locations of hosts and domains. It holds information that it obtains from other authoritative servers and reuses that information until the information expires.

Forwarding name server is essentially a caching name server but is useful in cases where computers lie behind a firewall and in which only one computer can make DNS queries outside that firewall on behalf of all the internal computers.

Check Your Progress 2

1. The following components are required to configure BIND.

Configuration file (/etc/named.conf) is the main DNS server configuration file.

Zone directory (/var/named) is the directory containing files that keep information about the zone that you create for your DNS server.

Daemon process (/usr/sbin/named) is the daemon process that listens for DNS requests and responds with information that the named.conf file presents.

Debugging tools (named `--checkconf`, and `named-checkzone`) are to determine whether the created DNS configuration correctly.

2. Samba is a software package that comes with Red Hat Linux to share file systems and printers on a network with computers that use the session message block (SMB) protocol. SMB is the protocol that is delivered with windows operating systems for sharing files and printers. In Red Hat Linux, the Samba software package contains a variety of daemon processes, administrative tools, user tools, and configuration files.

The default Samba configuration file is `smb.conf`, which is in `/etc/samba` directory (`/etc/samba/smb.conf`). If you need to access features that are not available through the samba server configuration file you can edit `/etc/samba/smb.conf` file as required.

3.9 FURTHER READINGS

1. Computer Networks by Andrew S Tanenbaum , Fifth Edition
2. SA2, Redhat System Administration I & II, Student Workbook
3. Cisco Certified Network Associate Study Guide, Seventh Edition by Todd Lammle
4. Redhat Enterprise Linux System Administration
5. <http://en.wikipedia.org/wiki/dhcp>
6. <http://en.wikipedia.org/wiki/dns>
7. <http://en.wikipedia.org/wiki/nfs>

UNIT 4 NETWORK MANAGEMENT AND SECURITY

Structure	Page Nos.
4.0 Introduction	70
4.1 Objectives	71
4.2 Networks and Security	71
4.3 User Security Management	72
4.4 Disk Security Management	74
4.5 Security Configuration and Analysis	76
4.6 Account Policies	77
4.7 Permissions and Restrictions	81
4.8 Configuring Network Settings	84
4.9 Advance Troubleshooting	85
4.10 Summary	88
4.11 Answers to Check Your Progress	88
4.12 Further Readings	89

4.0 INTRODUCTION

Network management includes the activities, methods, procedures, and tools that pertain to the operation, administration, maintenance, and provisioning of networked systems. Network Operation deals with keeping the network up and running smoothly. It includes monitoring the network to diagnose and identify problems as soon as possible, ideally before users are get affected. Network administration deals with keeping track of resources or components in the network and how these resources are assigned and do necessary steps to keep the network under control. Network maintenance is concerned with performing repairs and upgrades. For example, when the equipment must be replaced, when a router needs a patch for an operating system image and when a new switch is added to a network and so on. Maintenance also involves corrective and preventive measures to make the managed network run "better", such as adjusting device configuration parameters. Network provisioning is concerned with configuring resources or components in the network to support a given service. For example, this might include setting up the network so that a new customer can receive voice service.

A common way of characterizing network management functions is FCAPS- Fault, Configuration, Accounting, Performance and Security. Functions that are performed as part of network management include controlling, planning, allocating, deploying, coordinating, and monitoring the resources of a network. Network planning, frequency allocation, predetermined traffic routing to support load balancing, cryptographic key distribution authorization, configuration management, fault management, security management, performance management, bandwidth management, route analytics and accounting management are some of the key managerial activities that are to be performed for effective network management and ensure security. A network management system (NMS) is combination of hardware and software used to monitor and administer a computer network or networks.

Data for network management is collected through several mechanisms, including agents installed on network infrastructure, synthetic monitoring that simulates transactions, logs of activity, sniffers and real user monitoring. In the past, network management mainly consists of monitoring whether devices in the network were up or down, but today performance management has become a crucial part of the IT team's role.

As usage of Systems/Applications/ Services through network is increasing day by day, at same time Hackers/Attackers are playing a vital role to destruct the system resources and deface the Websites , etc. Security is essential to protect the resources of systems, services and applications from Hackers or Attackers and in turn protect the sensitive information and data.

Security management is an activity that includes a set of functions that are to be performed to protect telecommunications networks and systems from unauthorized access by persons, acts, or influences. Security functions are for creating, deleting, and controlling security services and mechanisms; distributing security-relevant information; reporting security-relevant events; controlling the distribution of cryptographic keying material; and authorizing subscriber access, rights, and privileges.

4.1 OBJECTIVES

After going through this unit, you will be able to:

- understand the network management and security;
- know how user management can be done in a security perspective;
- understand disk management in a security perspective;
- find account policies and specially password policy;
- find various user permissions and restrictions; and
- understand troubleshooting of network and available tools.

4.2 NETWORKS AND SECURITY

A computer network is a telecommunications network that allows computers to exchange data. The physical connection between networked computing devices is established using either cable media or wireless media. The best-known computer network is the Internet.

Network devices that originate, route and terminate the data are called network nodes. Nodes can include hosts such as servers and personal computers, as well as networking hardware. Computer networks support applications such as access to the World Wide Web, shared use of application and storage servers, printers, and fax machines, and use of email and instant messaging applications.

Networks established either through wired or wireless technologies. Wired networks can be established by using twisted pair, coaxial cable and optical fiber. Wireless networks can be established by using terrestrial microwave communications, communication satellites, cellular and PCS systems, radio and spectrum technologies. All these technologies use different network components that are to be used for routing, switching the data traffic for successful transmission. Some of the routing and switching devices involved in any communication network are the routers, switches, bridges, repeaters, hubs, etc. As many network devices will participate during data transmission from source to destination, ensuring secure communication is a challenging task.

Why do we need Security?

Security is the degree of resistance to, or protection from, harm. It applies to any vulnerable and valuable asset, such as Information Technology infrastructure, a computer network, a person, dwelling, community, nation, or organization. Due to rapid developments in internet technologies, usage of network services for ecommerce applications, banking, education and many such areas are increasing day by day and at same time hackers are also playing a vital role to destruct services and data. Security is essential to protect the network resources and in turn ensure secure data transmission.

Security services

The following are various security services or parameters to enhance the security of a systems, applications, data and are intended to counter security attacks.

- Authentication
- Authorization & Access Control
- Availability
- Confidentiality
- Integrity
- Nonrepudiation

Authentication is the act of establishing or confirming something (or someone) as authentic. It confirms the identity of a person or a system and permits one system to determine the origin of another system. It is essential in online community, where two systems are usually not directly connected

Authorization and Access Control is the level of access control that is permitted to use systems and services

Availability ensures that the system or an Application or a Service is always available to the authorized parties when needed

Confidentiality provides the secrecy of information and allows only authorized users to have access to information.

Integrity ensures that only authorized parties are able to modify computer system assets and transmitted information and provides the correctness of information.

Nonrepudiation ensures that neither the sender nor receiver of a message be able deny the transmission. It is a kind of system that includes authentication, integrity and non-repudiation should be able to detect tampered information and prevent valid information from being falsely rejected.

Deviation in any of the above security services is a breach in security.

4.3 USER SECURITY MANAGEMENT

User Management is an authentication feature that provides administrators with the ability to create users on a system or a service identify and control the state of users logged into the system and even network. User management includes the ability to query and filter users those are currently logged into the system or network, manually log out users, and control user login counts and login times.

Why do you need User Management?

Most security-conscious enterprises today implement some form of authentication and authorization for accessing network resources. In this process, user permissions

can be verified before granting access to resources, and user activity can be monitored through various logging mechanisms. In typical authentication and authorization deployments, administrators have various options available with regard to how users are authenticated, but have little control over how often users are authenticated. User Management enables administrators to control the frequency of user authentication, to ignore cached browser credentials and force the user to re-enter credentials, or to require more frequent authentication only if the user is accessing critical resources. This kind of flexibility allows administrators to implement authentication-based policies that more closely match their network security policies.

The User Management logout capability also provides more secure control over the state of users. For example, when using IP authentication mode, users are identified by the specified IP address until the IP surrogate time expires. If another person were to use that computer before the IP surrogate time expired, they would be treated as the original user. The common solution for preventing this scenario is to decrease the IP surrogate expiry time, causing the user to be challenged more often. Another key benefit of User Management is visibility into active user sessions. Using the Management Console and CLI, administrators can view all active users and filter display data by user, IP address, or realm for easier viewing. This can be useful for identifying the general login status of users or for making real-time decisions such as immediately logging off a user.

How does User Management work?

User Management is based on the concept of users logging in and logging out of a system or a network. A login is the combination of a unique IP address with a unique username in a unique domain. A user is considered logged in when first authenticated to the system or a network. Identifying users as logged in, or active, allows administrators to create flexible User

Management policies to fine tune user access and control.

The majority of User Management is done by framing and introducing a policy. Using policy, administrators can create, delete, modify users and also enforce controls on logging ins, logouts, number of login attempts and other such related.

The following are some of the policies implemented as part of user management in a security perspective:

- Create genuine user-names on a system or network or a service.
- Frequently monitor unauthorized users, if any created, logged in or connected.
- Introduce timestamps on logins and logouts of users and monitor in case of odd timings, if any activities performed.
- Keep tracking on users , who login in long time.
- Clearly add expiry date of a user at the time of user creation itself.
- Enforce policy to deactivate automatically after expiry.
- Introduce multi-level authentication such as authentication with username and password; username, password and IP address; username, password, MAC-address, etc.
- Limit the number of IP addresses associated with a single username.
- Limit the number of logins associated with a single IP address.
- Force a re-authentication to gain access to a particular network resource.
- Limit the login session time allowed in a particular timeframe.

4.4 DISK SECURITY MANAGEMENT

Disk Management is an activity to manage the drives installed in a computer like hard disk drives (internal and external), optical disk drives, and flash drives. Disk Management activities are like partition drives, format drives, assign drive letters, and other such related. For disk management can be done either with help of a tool or with a command to manage system disks, both local and remote.

The following are some of Disk Management functions:

- Create partitions, logical drives, and volumes.
- Delete partitions, logical drives, and volumes.
- Format partitions and volumes.
- Mark partitions as active.
- Assign or modify drive letters for hard disk volumes, removable disk drives, and CD-ROM drives.
- Obtain a quick visual overview of the properties of all disks and volumes in the system.
- Create mounted drives on systems using the NTFS file system.
- Convert basic disks to dynamic disks.
- Convert dynamic to basic disks, although this is a destructive operation.
- On dynamic disks, create a number of specialty volumes including spanned, striped, mirrored, and RAID-5 volumes.

Disk Management

Microsoft Windows utility that was introduced with Windows XP as a replacement to the fdisk command that enables users to view and manage the disk drives installed in their computer and the partitions associated with those drives. Figure 1 shows the snapshot of disk management utility in windows XP. Each drive is displayed followed by the layout, type, file system, status, capacity, free space, % free space, fault tolerance and overhead.

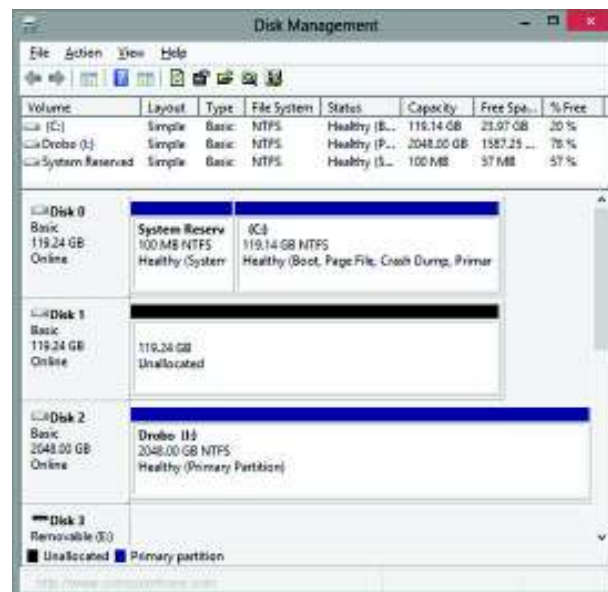


Figure 1: Disk Management in Windows XP

The following is the procedure to open Windows Disk Management

- Click Start, Settings, Control Panel.
- Double-click on Administrative Tools if in Classic View or click Performance and Maintenance and then Administrative Tools if in Category View. Note: If you do not have admin rights to the computer this will not be available.
- Once in the Administrative Tools window double-click Computer Management and then click Disk Management under the Storage section.

Similarly, disk management utilities are available in Linux environment (fdisk command) too to perform various activities as being done in Windows environment.

Disk Management in a security perspective

The following are to be followed for secure disk management:

- Create adequate number of disk partitions
- Allocate adequate storage space in each disk partition as per requirement
- Ensure minimum free space at the times
- Set password for each disk partition or a disk drive
- Scan each disk partition or a disk drive at regular intervals for viruses or worms, etc
- Enforce standard anti-virus software to check and clean viruses in a file before storing into a particular disk partition.
- Update Anti-virus software, periodically
- Disk partitions or disk drives should not be in sharing mode (put in sharing mode only on demand)
- Introduce RAID concept
- Disable disk remote access.
- Apply data encryption at disk storage level too apart from during data transmission
- Implement disk quotas by enforcing upper limits and warning alerts in case reaches to upper limit
- Disk defragmentation to be done, periodically
- Allow disk indexing service

Check Your Progress 1

1. What are various security services or parameters?

.....

.....

.....

.....

.....

.....

2. List the Disk Management functions.

.....

.....

.....

.....

.....

.....

4.5 SECURITY CONFIGURATION AND ANALYSIS

Configuring networked systems and components with adequate security controls in any organization to run or access through network is a critical task. Once, installations, configurations are done, it is very essential to look at the configurations and analyze the setting by keeping security issues in mind before deployment or immediately after deployment.

Security Configuration and Analysis can be done either manually with a check list or with help of a tool that can be used to analyze and configure computer security. Generally, users can use the tool to import one or more saved configurations to a private security database. Importing configurations builds a machine-specific security database that stores a composite configuration. One can apply this composite configuration to the computer and analyze the current system configurations against the stored composite configuration stored in the database.

Best practices for Security Configuration and Analysis

The following are the best security practices to be followed to ensure security in any networked system domain of an organization:

Restrict physical access to computers, especially domain controllers to trusted personnel

Physical access to a server is a high security risk. Physical access to a server by an intruder could result in unauthorized data access or modification as well as installation of hardware or software designed to circumvent security. To maintain a secure environment, you must restrict physical access to all servers and network hardware.

For administrative tasks, use the principle of least privilege

Using the principle of least privilege, Administrators should use an account with restrictive permissions to perform routine, non-administrative tasks and use an account with broader permissions only when performing specific administrative tasks.

Define groups and their membership

At the time of creation of users, the administrator has to define the user under a specific group depending on type of resources or applications where the user needs access.

Ensure that the system files and the registry are protected using strong access control lists.

Use strong passwords throughout your organization

Authentication methods require the user to provide a password to prove their identity. These passwords are normally chosen by the user, who may want a simple password that is easily remembered. In most cases, these passwords are weak and may be easily guessed or determined by an intruder. Strong passwords tend to be more difficult for an intruder to detect. Using strong passwords always help to provide an effective defense of resources.

Do not download or run programs that come from untrusted sources

Programs can contain instructions to violate security in a number of ways including data theft, denial of service and data destruction. These malicious programs often masquerade as legitimate software and can be difficult to identify. To avoid these programs, one should only download and run software that is guaranteed authentic and obtained from a trusted source. It is always better to ensure a current virus scanner is installed and functioning in case this type of software does inadvertently wind up on your computer.

Keep virus scanners up to date

Virus scanners frequently identify infected files by scanning for a signature, which is a known component of a previously identified virus. The scanners keep these virus signatures in a signature file, which is usually stored on the local hard disk. Because new viruses are discovered frequently, this file should also be updated frequently for the virus scanner to easily identify all current viruses.

Keep all software patches up to date

Software patches provide solutions to known security issues. Check software provider Web sites periodically to see if there are new patches available for software used in your organization.

Use some available security configuration and analysis tools (for example: secedit.exe is a Windows security configuration editor command tool) to configure, verify and analyze the network settings. This kind of tools can be used for the following:

- For frequent analysis of a large number of computers, as with a domain-based infrastructure,
- To configure security areas not affected by Group Policy settings. It includes the areas such as security on local files and folders, registry keys, and system services. Otherwise, Group Policy settings will override the local settings.
- Do not use such tools, when you are configuring security for a domain or an organizational unit.

4.6 ACCOUNT POLICIES

User accounts or simply login IDs are required to access any system's or network resources, if resources access configured in such a way that they are entry controlled. The system or network administrator has to create usernames on systems depending on need and deed. The administrator has to have a policy on user administration in terms of user account creation, deletion, modification, expire and also have a

password policy in terms of password length, complexity, minimum and maximum password aging, etc.

A user account policy is a document which outlines the requirements for requesting and maintaining an account on computer systems or networks, typically within an organization. It is very important for large sites where users typically have accounts on many systems. Some sites have users read and sign an account policy as part of the account request process.

Introduction to user and Group in Linux

Unix/Linux is a multi user and multi tasking Operating system. Redhat Linux uses User Private Group (UPG) scheme where user always get created with primary group. There is only one primary group per user.

When a user is created in Linux, the following are created

Home directory (home/username)

Mail account (/var/spool/mail/username)

Unique UID & GID

Types of Accounts

Generally, accounts fall into two types such as System accounts(system users) and Normal accounts (normal users). When an user account is created in Linux, the details are stored in the following files:

/etc/passwd
/etc/shadow
/etc/group

where

/etc/passwd contains database of all users created.

Example entry in a /etc/passwd file

```
u1:x:500:550:prog:/home/u1:/bin/bash
```

Where

u1 is User ID

X is mask password

500 is UID (user ID)

550 is GID (group ID)

prog is user ID comment field (generally it could be name of user)

/home/u1 is home directory of user ID and

/bin/bash is a shell

The /etc/shadow file contains the encrypted user passwords assigned by the password binary file. Password's are encrypted through DES (Data Encryption Standard) or MD5 (Message Digest Ver.5) Algorithm.

The/etc/group file contains Group name and GID of the group.

Managing Users

A system administration can manage a user's account. The various tasks that a system administrator can perform include adding, modifying and deleting user account.

User account Creation

To create a user, use the useradd command. The syntax is

```
root# useradd <option> <userID>
```

-u is UID

-g is primary/group name/GID

-o is : Override

-G is Secondary group

-c is Comment

-d is Home directory

-s: is Shell

Example:

root# useradd murli , it creates user ID murli and assign some default values against UID, GID, and other

user ID can also create by using all options along with the command useradd as specified above.

User ID Modification

To modify a user, use the usermod command. The syntax is:

```
root# usermod <options> <userID>
```

The options to usermod command are:

-l is Change user account

-L is to Lock the account

-U is to Unlock the account

Example

```
Root# usermod murli
```

User Deletion

To delete a user, use the userdel command. The syntax is

```
root# userdel <options> <userID>
```

The option to userdel command is

-r is recursively delete

Example

```
root# userdel murli
```

Managing Groups

System administrator can manage a group's account. The various tasks that a system administrator can perform include adding, modifying and deleting group account.

The following are various the commands:

- groupadd command is for group creation
- groupmod command is for group modification
- groupdelcommand is for group deletion

Account Policies

Account policies are required to manage users effectively. Account Policies contains the following:

Password Policy: These policy settings are used for domain or local user accounts. They determine settings for passwords, such as enforcement and lifetimes.

Account Lockout Policy: These policy settings are used for domain or local user accounts. They determine the circumstances and length of time that an account will be locked out of the system.

Account UID policy: Normal user accounts should not have UID (UserID) value 0(zero). Any account has UID value 0 will get root privileges automatically. System administrators should pay attention on UIID values at the time of account creation.

Account GID policy: Normal user accounts should not have GID (User groupID) value 0(zero). Any user account has GID value 0 will get root privileges automatically. System administrators should pay attention on GIID values at the time of user account creation.

Account creation policy: Policy should be implemented on account creation. Account will be created automatically when a user joined in the organization and needs to access on systems.

Account termination policy: Policy should be implemented on account termination. Account will be terminated automatically with or without grace period, when user superannuated or left or expired.

Kerberos Policy: These policy settings are used for domain user accounts. They determine Kerberos-related settings, such as ticket lifetimes and enforcement. Kerberos policy settings do not exist in local computer policy.

Password Policy

Password policy implementation is very essential to secure accounts and at same time secure the systems, the network and the services. The following are some of the parameters to be enforced as part of setting a password policy:

- Enforce password history
- Maximum password age
- Minimum password age
- Minimum password length
- Password strength (must meet complexity requirements)

The Passwords must meet complexity requirements policy setting determines whether passwords must meet a series of guidelines that are considered important for a strong password. Enabling this policy setting requires passwords to meet the following requirements:

- The password is at least six characters long
- Store password using reversible encryption for all users in the domain
- The password contains characters from three of the following four categories
 - English uppercase characters (from A through Z)
 - English lowercase characters (from a through z)
 - Base 10 digits (from 0 through 9)
 - Non-alphanumeric characters (for example: !, \$, #, or %)

Someone who attempts to use more than a few unsuccessful passwords while trying to log on to a system might be a malicious user attempting to determine an account password by trial and error. Account Lockout Policy settings control the threshold for this response and the actions to be taken after the threshold is reached. Generally, the number of login attempts to access any critical systems such as banking or any ecommerce applications systems are not more than three attempts. If any user fails to access such systems 3 times, system will automatically lock such account for a particular period. The lockout period depends on system sensitivity or organization. The following are the parameters to be enforced as part of account lockout policy:

- Account lockout threshold
- Account lockout duration
- Reset account lockout counter after

4.7 PERMISSIONS AND RESTRICTIONS

Privilege is defined as the delegation of authority over a computer system or a network or a service. A privilege is a permission to perform an action on a system or its resources. Examples of various privileges include the ability to create a file in a directory, or to read or delete a file, access a device, or have read or write permission to a socket for communicating over the internet. Users who have been delegated absolute control are called privileged. Users who lack most privileges are defined as unprivileged, regular, or normal users.

On Unix-like systems, the superuser (commonly known as 'root') owns all the privileges. Ordinary users are granted only enough permissions to accomplish their most common tasks.

Unprivileged users usually cannot perform the following tasks:

- Adjust kernel options.
- Modify system files, or files of other users.
- Change the owner of any files.
- Change the runlevel (on systems with System V-style initialization).
- Adjust disk quotas.
- Start or stop daemons.
- Signal processes of other users.
- Create device nodes.
- Create or remove users or groups.
- Mount or unmount volumes
- Execute the contents of any `sbin/` directory

Principle of least privilege

In security perspective, the principle of least privilege also known as the principle of minimal privilege requires that in a particular abstraction layer of a computing environment, every module such as a process, a user or a program depending on the subject must be able to access only the information and resources that are necessary for its legitimate purpose.

The following are the benefits of the principle of least privilege:

Better system stability

For example, applications running with restricted rights will not have access to perform operations that could crash a machine, or adversely affect other applications running on the same system.

Better system security

For example, running a system or a service in standard user mode increases protection against inadvertent system-level damage caused by attacks, malware, such as root kits, spyware, and undetectable viruses.

Ease of deployment

In general, the deployment of any application with fewer privileges is easy.

Superuser

The superuser is a special user account, which has all privileges and is used for system administration. Depending on the operating system, the actual name of this account might be the root, administrator or supervisor. In some cases the actual name is not significant, rather an authorization flag in the user's profile determines if administrative functions can be performed.

In Unix-like computer operating systems, root is the conventional name of the user who has all rights or permissions (to all files and programs) in all modes (single- or multi-user). Regardless of the name, the superuser always has user ID (UID) 0 (zero). The root user can do many things an ordinary user cannot, such as changing the ownership of files or directories and many such others. The superuser account always point at root's home directory.

Filesystem permissions

Most current file systems have methods of assigning permissions or access rights to specific users and groups of users. These systems control the ability of the users to view or make changes to the contents of the filesystem.

Traditional permissions

Permissions on Unix-like systems are managed in three distinct classes such as *s* user, *group*, and *others*. When a file or directory is created, its permissions are restricted by the umask of the process that created it. Files and directories are owned by a user. The owner determines the file's owner class. Distinct permissions apply to the owner. Files and directories are assigned a group, which define the file's group class. Distinct permissions apply to members of the file's group members. The owner may be a member of the file's group. Users who are not the owner or not a member of the group, such users treated under file's *others* class. Distinct permissions will apply to *others*.

The effective permissions are determined based on the user's class. For example, the user who is the owner of the file will have the permissions given to the owner class regardless of the permissions assigned to the group class or *others* class.

Permissions

The following are the three specific permissions (read, write and execute) on Unix-like systems (flavors of UNIX like all Linux versions and others) that apply to each class:

- The *read* permission grants the ability to read a file. When set for a directory, this permission grants the ability to read the names of files in the directory

- The *write* permission grants the ability to modify a file. When set for a directory, this permission grants the ability to modify entries in the directory. This includes creating files, deleting files, and renaming files.
- The *execute* permission grants the ability to execute a file. This permission must be set for executable binaries (for example a compiled C++ program) or shell scripts (for example a Perl program) in order to allow the operating system to run files. When set for a directory, this permission grants the ability to access file contents.

When permission is not set, the rights it would grant are denied. Files created within a directory will not necessarily have the same permissions as that directory.

Changing permission behavior

Unix-like systems typically employ three additional modes. These are actually attributes but are referred to as permissions or modes. These special modes are for a file or directory overall, not by a class.

- The *set user ID* or *setuid*, or SUID mode. When a file with setuid is executed or is on, the resulting process will assume the effective user ID given to the owner class. It enables users to be treated temporarily as root.
- The *set group ID* or *setgid* or SGID mode. When a file with setgid is executed or on, the resulting process will assume the group ID given to the group class. When setgid is applied to a directory, new files and directories created under that directory will inherit the group from that directory
- The *sticky bit* mode. The typical behavior of the sticky bit on executable files encourages the kernel to retain the resulting process image in memory beyond termination. On a directory, the sticky permission prevents users from renaming, moving or deleting contained files owned by users other than themselves, even if they have write permission to the directory. Only the directory owner and superuser are exempt from this.

These modes are also referred to as *setuid bit*, *setgid bit*, and *sticky bit*, due to the fact that they each occupy only one bit.

Notation of traditional file permissions

Symbolic notation

There are several ways by which permissions are represented. The most common form is symbolic notation as shown by the command `ls -l`.

For example, the following is the output after executing a command `ls -l`:

```
[murli@imssit etc]$ ls -l | more
-rw-r--r-- 1 root  root   15276 Oct 10  2006 a2ps.cfg
-rw-r--r-- 1 root  root    2562 Oct 10  2006 a2ps-site.cfg
drwxr-xr-x 4 root  root    4096 Apr 24  2009 acpi
-rw-r----- 1 root  root     450 Jan 26  2009 auditd.conf
```

The first character indicates the *type*(is file or directory, etc) and is not related to permissions. The remaining nine characters are in three sets, each representing a class of permissions as three characters. The first set represents the *user* class. The second set represents the *group* class. The third set represents the *others* class.

Each of the three characters represent the read, write, and execute permissions:

'r' if reading is permitted, '-' if it is not.
'w' if writing is permitted, '-' if it is not.
'x' if execution is permitted, '-' if it is not

The following are some examples of symbolic notation:

-rwxr-xr-x a regular file whose user class has full permissions and whose group and others classes have only the read and execute permissions.

crw-rw-r-- a character special file whose user and group classes have the read and write permissions and whose others class has only the read permission.

dr-x----- a directory whose user class has read and execute permissions and whose group and others classes have no permissions.

Numeric notation

Another method for representing Unix like permissions is an octal (base-8) notation. This notation consists of at least three digits. Each of the three rightmost digits represents a different component of the permissions: owner, group, and others.

The following shows file permissions in numeric notation (in Octal)

0000	no permissions
0111	execute
0222	write
0333	write and execute
0444	read
0555	read and execute
0666	read and write
0777	read, write and execute

4.8 CONFIGURING NETWORK SETTINGS

Setting up an ideal network in any organization requires the computer systems to host the applications; the network gateway level components such as the routers and switches; the security devices such as the firewall, the intrusion detection or prevention systems (IDS/IPS); the Anti-virus software to protect from viruses; a policy document that facilitates to implement and enforce various policies for effective usage of system resources and network services; the network design document.

For effective implementation of networked systems in any organization, all the required and involved systems or components shall be installed and configured properly. The network services/applications such as Email, DHCP, DNS, NFS, Web servers, etc will have a configuration files with default values. The system or the network administrator has to study each configuration file and set the required values in place of default values at each configuration files. It includes the IPAddresses connected with systems and services, blocking of unnecessary ports, mapping services with allowed ports, netting of public IPs with private IPs in creation of special Demilitarized Zones(DMZs), enforcing proper Access Control Lists at Firewalls, etc. Configuring the network with proper settings in any networked

environment ensures right services at right time always by protecting the environment from hackers or attackers.

4.9 ADVANCE TROUBLESHOOTING

Problem diagnosis and troubleshooting is critical activity which has to be done either manually or automatically (with help of scripts) for effective maintenance of systems and networks and their resources. It can be done locally or remotely.

Problems occur at the network level where systems are connected- due to improper configuration settings done at network components and services; at system level- the physical hardware, the installed operating systems and the system resources.

Basic network issues and troubleshooting tools

Network problems occur due to lack of network connectivity, improper software installations and configurations, insufficient or non-working network hardware, power fluctuations or no power, insufficient or no security devices (firewalls, IDS/IPS- intrusion detection or prevention system) installed in a network, network bandwidth issues, no anti-virus software or not updated anti-virus, and many such related.

Network troubleshooting tools are a necessity for every network administrator. When getting started in the networking field, it is important to have number of tools that can be used to troubleshoot a variety of different network problems and conditions.

Ping

The most commonly used network tool is the ping utility. This utility is used to provide a basic connectivity test between the requesting host and a destination host. This is done by using the Internet Control Message Protocol (ICMP) which has the ability to send an echo packet to a destination host and a mechanism to listen for a response from this host. Simply stated, if the requesting host receives a response from the destination host, this host is reachable. This utility is commonly used to provide a basic picture of where a specific networking problem may exist. For example, if an internet connection is down at an office, the ping utility can be used to figure out whether the problem exists within the office or within the network of the internet provider.

Tracert/traceroute

Once the ping utility has been used to determine basic connectivity, the tracert/traceroute utility can be used to determine more specific information about the path to the destination host including the route the packet takes and the response time of these intermediate hosts. The tracert utility and traceroute utilities perform the same function but operate on different operating systems, Tracert for Windows machines and traceroute for Linux based machines.

Ipconfig/ifconfig

One of the most important things that must be completed when troubleshooting a networking issue is to find out the specific IP configuration of the variously affected hosts. Sometimes this information is already known when addressing is configured statically, but when a dynamic addressing method is used, the IP address of each host can potentially change often. The utilities are ipconfig on Windows machines and the ifconfig utility on Linux.

Nslookup

Some of the most common networking issues revolve around issues with Dynamic Name System (DNS) address resolution issues. DNS is used by everyone using the internet to resolve commonly known domain names (i.e. google.com) to commonly unknown IP addresses (i.e. 74.125.115.147). When this system does not work, most of the functionality that people are used to goes away, as there is no way to resolve this information. The nslookup utility can be used to lookup the specific IP address(es) associated with a domain name. If this utility is unable to resolve this information, there is a DNS issue. Along with simple lookup, the nslookup utility is able to query specific DNS servers to determine an issue with the default DNS servers configured on a host.

Netstat

Netstat utility is used to display the currently active ports on a Linux machine. This is very important information to find for a variety of reasons. For example, when verifying the status of a listening port on a host or to check and see what remote hosts are connected to a local host on a specific port. It is also possible to use the netstat utility to determine which services on a host that is associated with specific active ports.

Putty

Putty utility is used to connect different systems remotely. Putty is being used to connect to a host via SSH.

Nmap

The nmap utility is one of the most versatile of network tools that is available. Regardless of how much experience a network engineer has, the nmap utility should always be available. Nmap utility can be used for the following:

- port scanning (TCP/UDP)
- version detection
- OS detection
- ping sweeps

Wireshark/tcpdump

Wireshark/tcpdump utilities are the packet scanners that have the ability to capture and analyze individual packets that are sent across a network.

Wireshark includes many different functions that provide the ability to perform a number of different analysis including filtering by conversation (i.e. IPv4, TCP, UDP..) and protocol analysis (HTTP, VoIP protocols (RTP, SIP, H.225..).

Tcpdump is another packet scanner that is available that provides the ability to analyze network traffic and is very easy to configure. Tcpdump is used on a Linux machine (various flavors) and is available for Windows as Windump.

inSSIDer

The inSSIDer utility can be used to not only scan for different networks within the 2.4 and 5 GHz ranges but also list the current signal strengths of different wireless networks within range.

Syslog server

A simple syslog server can be installed in the field to receive network events from key network elements. This information can then be recorded over time and help in determining the cause of a networking problem.

PTRG Network Monitor

The PTRG network monitor utility offers the ability to track the status of different sensors over a period of time. These sensors monitor anything from simple reachability (ping) to the response time of specific services (i.e. HTTP or POP).

System level issues and activities to be done

Generally, system level problems are due to system hardware components, improper operating system installations, improper configuration settings and other such related.

The following are some of the activities to be done by a system administrator:

- Periodically update the Operating system patches, if any
- Remove unnecessary accounts created by default
- Close all ports except the ports required to allow the services
- Harden the operating system
- Regularly monitor the system resources such as the file system, the storage, the log files and analyze.
- Regularly monitor the users and their activities
- Update Anti-virus patches, periodically

Best practices to be followed for ideal networked environment

Organizations require ideal networked systems to be established to run and access various network services or applications. The following points may be kept in mind for the purpose:

- The systems or servers should have adequate resources in terms of processing speed, memory, storage, etc
- Use the Routing and Switching devices with adequate resources
- Configure Firewall Device with proper ACLs(Access Control Lists) to control the network traffic
- Configure IDS/IPS for effective detection of intrusions/intruders
- Configure network configuration files properly for various network services such email, DNS, DHCP etc
- Implement VLANs and enforce traffic limits
- Use appropriate network monitoring tools for effective network monitoring
- Do periodic system and network performance auditing
- Do periodic security audit at system and network level
- Policy to be framed and implemented for effective usage of systems and network resources

Check Your Progress 2

1. Write the standard minimum requirement to create a strong password.

.....

.....

.....

.....

.....

2. What is the significance of Nmap utility.

.....

.....

.....

.....

.....

4.10 SUMMARY

The unit emphasized on network management and security issues in a network. The user management and disk management in a security perspective are clearly and comprehensively explained. Various user account policies along with the parameters to be set for a password file are explained. The file permissions are explained with different examples. Finally, the need of problem diagnosis and troubleshooting along with available tools was also discussed.

4.11 ANSWERS TO CHECK YOUR PROGRESS

Check Your Progress 1

1. The following are various security services or parameters
 - Authentication
 - Authorization & Access Control
 - Availability
 - Confidentiality
 - Integrity
 - Nonrepudiation
2. The following are some of Disk Management functions:
 - Create partitions, logical drives, and volumes.
 - Delete partitions, logical drives, and volumes.
 - Format partitions and volumes.
 - Mark partitions as active.
 - Assign or modify drive letters for hard disk volumes, removable disk drives, and CD-ROM drives.

- Obtain a quick visual overview of the properties of all disks and volumes in the system.
- Create mounted drives on systems using the NTFS file system.
- Convert basic disks to dynamic disks.
- Convert dynamic to basic disks, although this is a destructive operation.
- On dynamic disks, create a number of specialty volumes including spanned, striped, mirrored, and RAID-5 volumes.

Check Your Progress 2

1. To set strong password, password contains characters from three of the following four categories and minimum 8 characters.
 - English uppercase characters (from A through Z)
 - English lowercase characters (from a through z)
 - Base 10 digits (from 0 through 9)
 - Non-alphanumeric characters (for example: !, \$, #, or %)
2. **Nmap** utility is one of the most versatile of network tools and can be used for the following:
 - port scanning (TCP/UDP)
 - version detection
 - OS detection
 - ping sweeps

4.12 FURTHER READINGS

1. Computer Networks by Andrew S Tanenbaum , Fifth Edition
2. SA2, Redhat System Administration I & II, Student Workbook
3. Cisco Certified Network Associate Study Guide, Seventh Edition by Todd Lammle
4. Redhat Enterprise Linux System Administration
5. Fundamentals of network security by Eric Maiwald, Dreamtech publisher
6. Linux system security, the Administrators guide to open source security tools by Scott Mann. Ellen L. Mitchell, Second edition, Pearson Education.
7. <http://en.wikipedia.org/wiki/security>