CIT 754 : Network Programming and Design

Course Development

Programme Leader
Dr. S.1.0gunrinde
Noun, Lagos.

Course Coordinator
A.M. Balogun
Noun, Lagos.

National Open University of Nigeria

Funded by

Education Tax Fund

National Open University of Nigeria

Headquarters
14/16 Ahmadu Bello Way
Victoria Island
Lagos

Abuja Annex ()Mee
245 Samuel Adesujo Ademulegun Street,
Central Business District
Opposite Arewa Suites
Abuja

centrainfo@nou.edumg
URL: www.nou.edumg

First published 2004

ISBN 978 058 324 6

Published by I leinemann Educational Books (Nigeria) Plc
for National Open University of Nigeria

Co ntents

Page

The course, Network Programming and design is an intermediate-level course on computer networking within the Bachelor of Science in Communication Technology degreee. The course is self- contained and assumes no specific knowledge of computer networking concepts or networking programming techniques. However, you are expected to have a basic knowledge of computer operations and programming.

The overall aims of this course are to introduce you to networking concepts and networking programming techniques. Topics related to the networking architecture, included underlying mechanism are also discussed. Theoretical concepts and practical techniques are integrated with the practical analysis of case study design and programming problems.

The bottom-up approach is taken in structuring the course. We start with the basic building blocks of a computer network and how these individual blocks or units are integrated to construct a complete network. These includes how networks can be interconnected and how communications are made across networks. Following these underlying principles, we proceed to the programming techniques for making use of network resources.

There are *three* modules in this course, each comprises of 10 units of topics that you are expected to complete in 3 hours. The three modules and their units are stated below.

Module One: Network Basics and Architecture

Unit 1: Network Overview
Unit 2: Nodes and Links
Unit 3: Network Topology
Unit 4: Network Adaptors and Cabling
Unit 5: Network Operating System
Unit 6: Network Technologies
Unit 7: Encoding and Error Detection
Unit 8: OS I 7-Layer and Internet 4-Layer models
Unit 9: Network Hardware Components
Unit 10: The roles of IP, TCP,and UDP

Module Two: **intertletW** rking, Network Design and Maintenance

Unit 1: IP Naming and Addressing
Unit 2: IP Routing
Unit 3: The TCP
Unit 4: The UDP
Unit 5: Design Goals
Unit 6: Analysis of Network Requirements
Unit 7: Designing a Network Infrastructure
Unit 8: Network Implementation
Unit 9: Network Maintenance
Unit 10: Network Troubleshooting

Module Three:Overview of Network Programming

Unit 1: Introduction to Networking Programming
Unit 2: Creating a TCP Socket
Unit 3: Elementary TCP Sockets

Unit 4: Unix Standards
Unit 5: Sockets Introduction
Unit 6: TCP Client-Server Example
Unit 7: Handling Interrupted Calls
Unit 8: I/O Multiplexing: The Select and Poll Functions
Unit 9: Socket Options
Unit 10: Elementary Name and Address Conversion

From the foregoing, the content of the course can be divided into three major blocks:
1 Network Basics and Architecture
2 Internetworking, Network design, and Maintenance
3 Overview of Network Programming

Module one defines the basic building blocks of a computer network and how they are iontерconnected to form a functional unit.
Module Two defines the Naming and addressing schemes on a network. Details of the the Internet Protocol **(IP),** Transmission Control Protocol (TCP) and User Datagram Protocol (UDP) are presented. Module Three dicusses the issue of network programming. The use of sockets in programming is presented.

[illegible] course will help you do

The overall aims and objectives of this course provide guidance on what you should be achieving during your studies in general. Each unit also has its own unit objectives which state specifically what you should be achieving in the corresponding unit. To evaluate your progress continouusly, you are encouraged to refer to the overall course aims and objectives as well as the corresponding unit objectives upon the completion of each.

[illegible]

The overall aims of this course will help you to:
I. develop your knowledge and understanding of the underlying principles of computer network.
2. develop your capability to evaluate different network designs and propose your own designs for different situations.
3. develop your capability to write network applications.
4. develop your capability to write programs for developing interactive applications.

[illegible]

Upon completion of the course, you should be able to:
I identify the key elements of a computer network and how networks can be interconnected.
2 describe the application of layering models in the design of networks.
3 discuss the underlying algorithms of networking protocols.
4. develop and outline design solutions to meet specific networking requirements.
5. develop C programs in the Unix environment.
6 discuss the underlying techniques and algorithms used in network programming.
7. develop network **applications** under the Unix environment.
8. discuss the underlying principles of Internet protocols.
9 develop programs for web-based services.
10. discuss security issues in network design.

Working through this cOurse

We designed this course in a systematic \A' a y, so you need to work through it from Module one, unit 1 through to module three. unit 10. This xvil I enable you appreciate the course better.

erials

Basically, we made use of textbooks and online materials. You are expected to look for more literatures and more web refrences for further understanding. Each unit has refrences and web refrences that were used to develop them.

Online Materials

A world wide web (www)site has been developed for online readings and any activities which have an online elements. You should refer to these web sites for all the online reference materials required in this course.

The website is designed to integrate with the print-based course materials. The structure follows the structure of the units and all the reading and activity numbers are the same in both media.

A computer system suitable for connecting to the Internet is essential. This course cannot be studied without easy, regular access to such a system. The specification given below is about a minimum and should not be viewed as a precise description of what you should buy if you are intending to do so.

Below is the configuration:

P— I 100MHZ MMX + sound+video
1.2GB Hard disk
32MB RAM
24X CD-ROM
120W SPEAKERS
3.5" Floppy Disk Drive
14" SVGA Colour Monitor
Mouse + pad
Windows 95 Keyboard
56Kbps Internal Modem
Printer

The software to be put on the machine include:

Window 95 operating System
UNIX Operating System
Java (or C++) Programme Language
Microsoft Office
Norton's Anti-Virus.

Assessment

The course, Network Programming and Design requires you to attend a three-hour final examination which contributes 50% to your final grading. The final examination covers materials from all part of the course with a style similar to the Tutor-marked assignments.

The examination aims at testing your abilityto apply the knowledge you have learned throughout the course, rather than your ability to memorise the materials. In preparing for the examination, it is important for you to receive the activities and Tutor-marked assignments you have completed in each unit. The other 50% will account for all the TMA's at the end of each unit.

This section suggests the number of weeks that you are expected to spend on the three modules of 30 units, and the assignments that follow each of the unit.

We are of the opinion that each unit with its associated TMA should take one week, bringing your study period to a maximum of 30 weeks.

How to get the most from this course

Practise is important for you to learn various concepts in this course. Independent activities and case activities which are based on a particular scenario are presented in the units. The activities include open questions to promote discussion on the relevant topics, question with standard answers, and program demonstrations on the World Wide Web. You are strongly advised to do all these TMAs as they will help you in understanding the concepts. You may try to get yourself into each unit with the following steps:

I read the study unit.
2 read the textbook, printed or online references
3 perform the activities.
4 participate in group discussions.
5 complete the TMAs.
6 participate in online discussion.

This course makes intensive use of www-based materials. Specific home page address will be given to you for refrence. There are also optional readings in the units. You may wish to read these to extend your knowledge beyond the required materials, They will not be assessed.

[illegible]

About 20 hours of tutorials will be provided in support of this course. You will be notified of the dates, times and location of these tutorials, together with the name and phone number of your tutor as soon as you are allocated a tutorial group.

Your tutor will mark and comment on your assignments, keep a close watch on your progress and on any difficulties you might encounter and provide assisstance to you during the course. You must mail your TMAs to your tutor well before the due date (at least two working days are required). They will be marked by your tutor and returned to you as soon as possible.

Do not hesitate to contact your tutor by phone, e-mail if you need help The following might be circumstances in which you would find help necessary. Contact your tutor if:

- you do not understand any part of the study units or the assigned readings.
- you have difficulty with the TMAs.
- you have a question or problem with an assignment, with your tutor's comments on an assignment or with the grading of an assignment.

You should try your best to attend tutorials. This is the only chance to have face to face contact with your tutor and to ask questions which are answered instantly. You can raise any problem encountered in the course of your study. To gain the maximum benefit from course tutorials, prepare a question list before attending them.You will learn a lot from participating in discussions actively.

Summary

The course, Network Programming and Design, is intended to develop your understanding of the underlying principles of your computer networking, your knowledge of computer network design and your ability to write network programmes. This course also provides you with practical knowledge and hand-on experience in designing and implementing a local area network and its connection to the internet.

This course may not describe all networking concepts in depth but it does provide you with a strong foundation for your best to bring the knowledge and techniques you have learned into practise as this helps you to develop a more thorough understanding of the various aspects/concepts. We hope that you will find the course interesting and enjoyable and that you will be able to use the knowledge and skills gained from thiscourse throughout your career.

We wish you success in the course and hope that you will find it both interesting and useful. In the longer term, we hope you will enjoy your acquaintance with the National Open University of Nigeria and we wish you every success in your future.

Module 1: Network Basics and Architecture

Unit 1 : Network Overview

ntrOUPetiOil

This unit covers the basic concept °fa computer network. As there are a lot of jargon in the computer world, this unit also tries to introduce to you some of the key terms that 'vil I be used frequently in later units. A stand-alone computer has limited uses. ilhe only \\Tay it can exchange information with other computers is by using removable storage media such as floppy dikettes. You may not have strong feelings on this issue when using a computer at home, but in large establishments, w here different departments are usually sharing a common data source, effective networking is critical. This unit w ill guide you through networking fundamentals that are necessary for better understanding of the succeeding units.

Jlbjectives

By the end of this unit, you should he able to:

- understand the meaning of computer networks.
- appreciate the roles of computer networks iu teciay's business.
- identify the various classes of computer networks and their applications.
- argue favourably in support of computer networks as against its concerns.

3.0 Wh atm a Computer Network ?

In this unit, we use the following simple, but complete, definition of computer networks: A computer network is a set of computers that are connected and able to exchange messages. The word 'network' can also be defined as a communication s stem that links computers and computer resources in the same way that telephone system link telephone. The computer net ork can be conceived as a combination of computer and telecommunication technologies.
i.e Computer Network =Computer ± Telecommunication.

It is important to note that this definition excludes a large time-saving system with a collection of terminals attached, a type of systems that is sometimes called a nem ork. In our descriptions and discussions of networks, we assume that each station on the network is a computer that can be used independently of the network; the network is an extension of the computing environment offered by that computer.

Networks may also include some devices such as printers, used only through the network. Though these devices cannot operate in a stand alone mock. wewi II not consider such devices to be stations or modes on the network, but as a separate category of peripherals devices, accessed through the network.

3.1 Reasons for Installing Networks

Though electronic mail has el ()Ricci as a critical use of computer networks, there are other reasons for installing and using a network. The first reason for networking is for the purpose resource of sharing. Computers joined in a network offer the combined resources of all the stations to the user of each. The increase use of personal computers (PC) and workstations emphasizes the importance of this facility. The advantages offered by networked small computers over a single, large, time-shared system will be summa-rized in the following section.

3.1.1 Price/Perfomance Ratio

As workstations and personal computers becoine more powerful, they offer a superior price perfomance ratio to the mainframe. Networked computers combine the advantages of dedicated individual systems with the resource sharing offered by large time- shared systems.

3.1.2 Perfomance Quality

You may have seen, perhaps, that response time shared environment degrade as more users log in. This will enable you to appreciate the perfomance quality of the dedicated system. Networking the individual systems retains the benefits while alowing printers, plotters, modems, and other peripheral devices to be shared. Each user gets access to the resources needed, while the idle time is minimized.

3.1.3 Reliability

Network-supported resource sharing contributes to reliability, by providing alternatives to a critical re-source. Thus, if a particular printer is down or is busy, another may be available. If a disk fails,another copy of the needed file may remain accessible.

3.1.4 Accessible Resources

Networks expand the list of accessible resources far beyond those provided in most organisations. Catalogs of great libraries (and small ones), documents, public domain software, are the opportunity to use proprietary software are easily available through the network, from the most ordinary personal computers or worksta-tion.

3.1.5 Incremental Growth of Computing Power

The use of multiple computers in a network, rather than a single large system, eases growth of computing power. As more users are added, more stations are added to the network. Unlike a time - shared system where response time gradually deteriorates with heavy use, the expanded network provides each new user a dedicated processor, unaffected by the activity of other users in the organisation. Of course network traffic can become so great that access to resources on the network becomes difficult. However, when a time-shared system cannot absorb any more users, upgrading is a major financial decision. When network traffic becomes so heavy that something must be done, a small investment splits the network into two or more lightly used networks, still able to communicate with each other when necessary.

3.1.6 Value-added Communication Devices

When computers are connected, some facilities can be added to it in terms of hardware and software by way of enhancing its usage. Example of such applications are teleconferencing, multimedia applications, elec-tronic mail, etc.

3.2 Concerns of Computer Networks

Not every characteristic of a network system is an advantage. One of the major concerns of a network is the initial cost of installation. Many organisations shy away from it because of costs. For a small-or medium-scale businesses, they may not enjoy the economies of a scale. As a concomitant to the above is the cost of maintaining the network which is on the high side.

Another serious concern is the security issue. You must have heard that the most secure system is a stand-alone PC. Intruders, hackers, fraudsters etc. are examples of great threat to an installation. Also down-time can be catastrophic in terms of loss. Few hours of down-time can result into a collosal loss of money. Finally, in a single large system, all the disk space is available and can be divided among the users as needed. In a collection of workstations, the storage is distributed with the processsing power. The decision as to how much disk space each user will need is essentially static and made when the machine is purchased. If one user encounters a need for a very large block of storage, which is not available on the local machine, the fact that more than that amount of space is available on another machine may not help.

Conclusively, you should not be carried away by these concerns. The advantages derivable from net-worked systems greatly outplays its disadvantages.

3.3 Significance of Networks in the Computing Environment

In a description of the NSFnet, the National Science Foundation network for research and education, comm-nications networks are equated with the industrial revolution insignificance, and stephen Wolff of NSF is quoted as saying:

> I see it as a revolution in the way people work with one another. This kind of collaboration has never been possible before. The kinds of interactions that you can have in the network

environment are at once more rapid than mail and less demanding than the telephone. but equally absorbing. Without the network, a scientist told me. 'life as we know it would cease to exist'.

While many people still find life without networks worth living, the impact of network technology is undoubtedly substantial. Networks, interconnected with each other to form larger networks span the globe. The emergence of networked computers as the computing of form for many applications impacts every aspects of the study and use of computing. Architecture and operating systems, programming languages and tools, algorithm development and analysis all reflect the fundamental change in the nature of the system in use. The implications affect every type of application development, whether artificial intelligence, computer-human interface, databases, file systems, symbolic computation, visualisation and every other area where the presence of economical incremental increases, in processing power affects the potential accomplishments.

For you as a computing professional. who not only use the computer but also develop applications, the significance of networks is perhaps more immediate. The application development effort must often include not only writing programmes that work correctly and efficiently on a particular type of computer (and perhaps port easily to other systems as well) but also may have to cooperate with other programs running on different systems. Simple file access may require interaction with a file server. Printing may include specifying which of many printers to use. Perfomance of a programme using network resources can vary substantially, depending on the load on the network. Some of the resources that the program requires may be inaccessible because some part of the network has failed. One programme may be competing for the same resources as programmes runing on other computers.

3.4 Major Types of Networks

There are essentially three categories of computer networks. They are classified based on the *distances spanned* and *geographical locations.* Many network applications run by invoking functions of the network software, and with little or no concern about the physical characteristics of the specific network platform in use. Characteristics of the major types of network systems do intrude on application development in some cases. Also, concerns such as privacy, security, reliability, response time, and accessibility often depend on the type of network platform in use. Though the titles suggest geographic spread, the principal traits are only indirectly related to the distance covered. Let us now discuss the classes of computer networks that are available.

3.4.1 Local Area Networks (LANs)

A Local Area Network can be defined as an inter connection of autonomous computer systems to facilitate the sharing of files, applications, printers, disk space, MODEMS, other LAN resources within a restricted specified distance. In other words, this is an arrangement in which computer within a "local" area are interconnected, It is the collection of networked computers that reside within a small physical building usually not beyond I kilometre.

Local Area Networks extend the usefulness of one computer by connecting it to others. Often a computer on a network is connected to other networks as well, and has essentially unlimited access to resources and services around the world. At the local level, the Local Area Networks have very low error rates and propagation delay that is negligible for most purposes. Transmission in most local area networks is by broadcast — every station on the network receives every transmission. Thus, there are no routing decisions to make. Every packet (message unit) follows the same path, so reordering does not occur. These characteristics make LANs suitable for applications that depend on timely results, for example interactive processing involving files or other resources located on several different systems or time — critical response to a monitoring device. The broadcast mode of operation is suitable for applications requiring message exchange among a number of stations, like checking individual calendars to processor able to share in a demanding computational task.

A combination of technology and the performance expectations of LANs limits the distances they can cover. A small LAN might connect a few computers in an office, or in a home; a large LAN could extend over an office park or university campus, connecting computers and other devices in a number of buildings. Common speeds are 10Mbps (megabits per second) or I 6Mbps. Very simple network systems that allow limited sharing between two or more personal computers (for example, at home) are sold at most computer stores. The simplest of these use the computers' serial ports, standard telephone wire, and software that allows printer sharing, file transfer, and sending messages between computers.

3.4.2 Metropolitan Area Networks (MANs)

The term Metropolitan Area Networks is often applied to the new high-speed network technologies to distinguish them from wide area and local area networks. These technologies have pushed the carrying capacity of the communication links onto the gigabits range. They can be used over larger areas than LANs, and are often used to connect LANs together to form a greatly extended LAN environment. Another important potential for these networks is the ability to carry information in forms that require many more bits than convetional text or simple graphics. Clearly, the standard LAN speeds are not adequate. High speed networks, approaching gigabits per second, are needed to provide the ability to deliver the full potential of multi-media displays to the user from a source on a different machine. They are also needed to connect LANs together and deliver timely access to resources. Hence, the Metropolitan Area Network can be conceived as the interconnectivity of several LANs that spans a given geographical area. Examples are campus-wide network or a network for a large industrial complex. The coverage is usually within 10-kilometre range.

3.4.3 Wide Area Networks (WANs)

Wide Area Networks are characterised by significant propagation delay in message transmission and by high incidence of lost or damages transmissions. Message units, called packets, are passed from one intermediate mode to another until they arrive at the destination. Effective routing techniques are important to network perfomance. Packets from a single message may travel by different routes and arrive at the destination out of order. Typical transmission speeds range from 56kbps to 1.54Mbps. Since a wide area network is composed of poiut-to-point connections, connectivity is an important design consideration.

Often, the station connected to the subnet is a gateway to a local area network. Thus, the user with access to the resources of a local area network can reach out to a larger networks (or MANs) connected to other networks combine to form very complex patterns with characteristics similar to point-to-point wide area networks. Similar problems concerning routing decisions, dealing with failures, temporary conditions, etc. arise. A further complication concerns the differences among the types of local area networks: the required format for transmissions, maximum lenghts, connections for acknowledging successful delivery of a packet, etc. The stations that join two or more local area networks and address these issues are called **routers.** Tranparent access to very remote files and printers is not impossible, but is rare. Although the most common application of a wide area netwok is explicit communication (electronic mail and file transfer), another popular use is to log in to remote computer that offers a particular service. You will then realise that, essentially a wide area network is a type of network that links several cities within a particular country. Example is a network that links all the branches of a bank within the country.

In this unit, you have learned a number of key issues that relate to computer networks, its major roles in today's business and concerns. You must have learned that the combination of computing power with **high speed** data communication is regarded as potentially the most powerful influence yet on our handling of information. Although in many instances computers are used to perform their intended role in a stand-alone mode, in other situations there is a need to interwork and exchange data with other computers. You need to be aware, however, especially about the various classes of networks and their specific applications based on distance spanned. Finally, networks for computer data are one of the great success stories of the information technology industry.

mniary-

...

What you have learned in this unit concerns the fundamentals of computer net arks. its importance and concerns. It has served to convince you that many of the :cm ire., :hat now taken for granted would not be possible without the computer data networks \\ hich. Ihouh all pervasive. are largely invisible to the end-user. The units that follow shall build upon this introduction.

or Marked Assignment

A company, XYZ Nigeria Limited, has approached you for your professional advice on her computerisation project. You are expected to convince the management of the company on the enormous potentials of computer networks.

Exercise 1.1

Describe the major types of networks that you know.

Exercise 1.2

Define exhausively the term 'Computer Network'

itAR:c1.00100,And Other Resources

Microsoft corporation *Networking Essentials*, (2nd edu,) Redmond, Washington Mikrosoft Press, (1996)

Tanenbaum, A. Computer Networks,
(2nd edn,) Englewood Cliffs. New Jersey. Prentice Ilall. (1989)

Online Materials

http: // www.csc.vill.edu /—cassel / netbook / first. htm # Whatis
http: // www.esc.vill.edu /—cassel / netbook / reasons. htm 1
http: // www.esc.vill.edu /—cassel / netbook / types.htm
hap: // www.esc.vill.edu /—cassel / netbook / types.htm / # Ian
http: // www.esc .vi I Led ti /—cas se I / netbook / types.htm / *tr* man
http: // www.ese.viltedu /—cassel / netbook / types.htm / ft' wan
http: // www.awstevenson.demon .co.uk/SKYNOTES/connect.htm

Module 1:Network Basics and Architecture

Unit 2 : Nodes and Links

Page

In this unit, you will be exposed to the basic building block of a computer network from the hardware point of view. These are referred to as the nodes and links. You will also learn how these two vital components form the basic hardware of a network. The factors to be considered when selecting a link on a network is also presented. Let us now look at what you should learn in this unit, as specified in the unit objectives below.

tett

On succesful completion of this unit, you should be able to:

- discuss the basic building blocks of a computer network.
- explain the rationale for nodes and links in a network.
- discuss the factors to be considered in the choice of link.

discuss the types of signals available.

for Nodes and Links.

The term 'network' may lead you to think of computer systems in large corporations that cover several geographical locations or even cross continents. But just connecting two computer together with a cable also forms a network. No matter how small the network is, its hardware building blocks must contain device elements, which we call nodes, and physical media, which we call links, used to connect the nodes. Essentially, a computer network consists of hosts computers that are nodes of the network and communication links that connect the nodes. The nodes and links of a network are generally reliable components, yet do occassionally fail. The failure of such a component, by itself, is not usually disastrous, and methods for recovering from such failure are well known. Furthermore, the failure of a network node or link will not typically affect network functioning for the nodes that are still in service as long as other would normally be sent through the failed node (of course, network performance could be affected).

3.1 Nodes

Any device that is connected to the computer network is regarded as a node. Very often, a node is a general purpose computer or workstation, on which you may run network appliations to communicate with other on the network or local applications such as word processing or database programmes. However, as intermediate network device such as a router, which helps in transmitting data, or a printer may also be network node.

How is a node connected to the network? It needs somekind of hardware interface device for doing this. This device is usually called a *network adaptor*. Taking an IBM-compatible personal computer as an example, the network adaptor is usually called a *network Interface Card* (NIC). The NIC is plugged into an expansion slot (PCI or ISA, or on-board atimes) on the main board of a computer, the same way as you plug or connectors on the N1C for connecting links of different physical media, as shown in Figure 1.1.

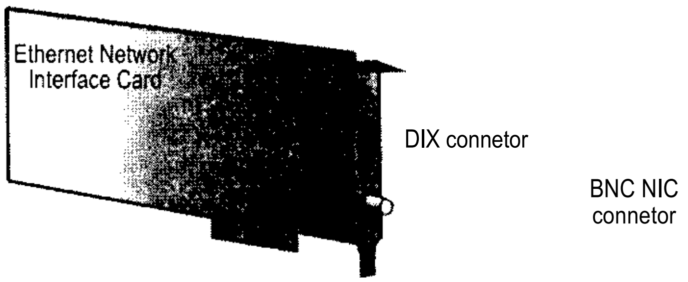


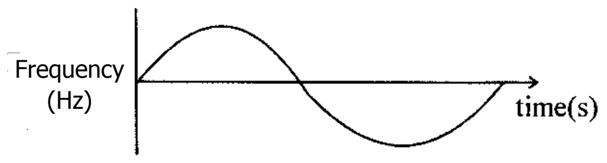
Figure 1.1 Network adaptor card (NIL) on a PC:

The network adaptor is responsible for sending data from the computer memory to the outside network and also receiving data destined for that computer from the outside network. Data exchanged between nodes are in units of frames. Computer data is composed of binary digits (bits) 0 and 1; and a frame is a block of bits. Bits are exchanged between network adaptors. There will be a more detailed discussion of network adaptors in the section on network adptors and cabling further on in this module.

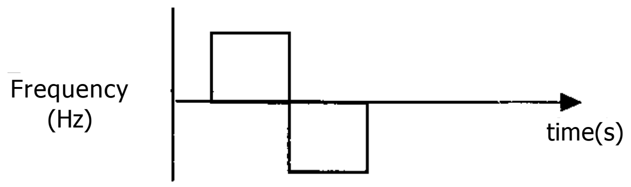
Beside the network adaptor, a node must be installed with a network operating system in order for the applications runing on it to communicate with the network. The NOS may be a separate software running on the top of ordinary operating systems. Novell Network is an example of this type of NOS. Windows NT and UNIX, on the other hard, are operating systems with built in networking features; and so they are not only responsible for the basic operations such as managing file systems and hardware resources; but also responsible for data communication across the network. In later unit in this module, we will look deeper into this issue of network operating system.

3.2 Links

Links are physical media that connect the nodes. Through a link, signals can be transmitted from one node to another. Digital links can discrete signals like electromagnetic pulse, whereas analogue links carry continuous electromagnetic signals. Figure 1.3 show the graphical representation of analogue and digital signals.



a) Analogue Signal



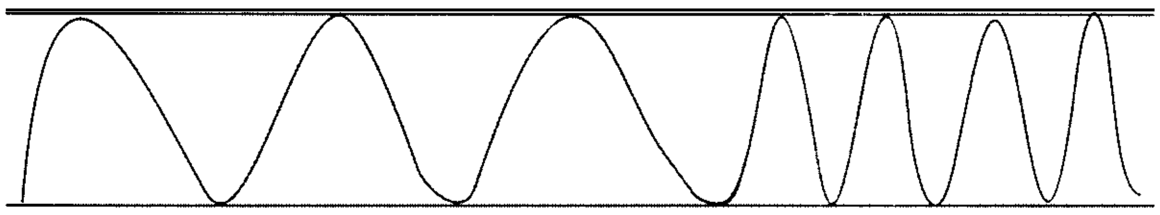
b) Digital Signal

Figure 1.3 Analogue and digital Signal representations

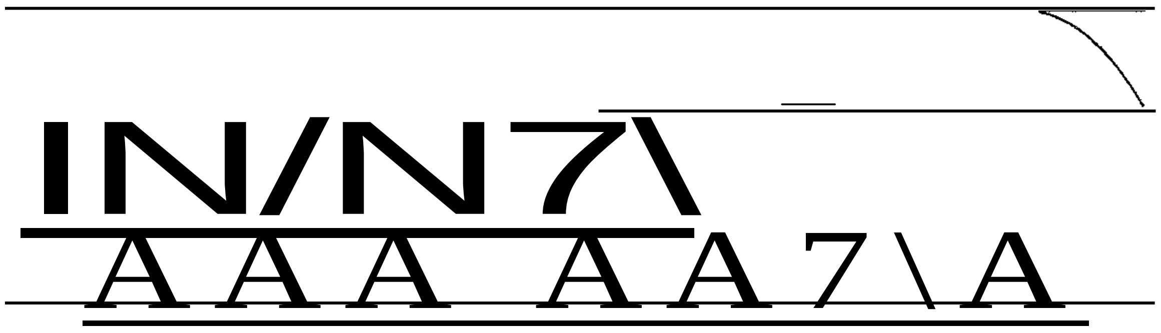
You should be aware by now that we have analogue and digital information. Analogue data is composed of continuous valves, like voice and video whereas digital data is composed of discrete values, like computer data which is composed of Os and Is. To transmit data through links, data is encoded into electromagnetic signals. For signalling, we have analogue and digital signals. An analogue signal is a continously varying electromagnetic wave, whereas, a digital signal is a sequence of discrete pulses (e.g voltage pulses in copper media). An analogue link is capable of transmitting analogue signals while a digital link is able to transmit digital signals.

3.3 Baseband and Broadband LAN

As a corollary to the previous section, two terms should be introduced: *baseband* and *boardband*. A baseband Local Area Network refers to one that is capable of carrying only one signal at a time. The entire frequency spectrum of the transmission medium is used to transmit the signal using the concept of Time Division Multiplexing (TDM). The term 'broadband' originates from the telephone world but a different meaning is used in computer networking. A broadband network refers to one using Frequency Division Multiplexing (FDM) techniques to send data through the transmission medium. The multiplexing techniques allow the frequency spectrum of the medium to be divided into channels or paths. Multiple signals can propagate in the medium at the same time through different channels. Figure 1.4 shows the pictorial representations of the two techniques.



a) Time Division Multiplexing



b) Frequency Division Multiplexing

Figure 1.4 Time Division Multiplexing Vs Frequency Division Multiplexing

3.4 Commonly used Media for Digital Links

In this course, we will mainly focus on the transmission of digital data over a digital link, as most links used in computer networks are digital links. The encoding of digital data into digital signals will be discussed in detail in unit? on encoding and error detection later in this module. Bits are discrete values which are easier to transmit through digital links. Some commonly used media for digital links are Unshielded Twisted Pairs (UTPs), coaxial cables and optical fibres. Some author's believes that optical fibre is not able to transmit digital signals. This is not quite true. Indeed, light, instead of electricity, is transmitted in optical fibre. Digital signalling in optical fibre is not done by voltage pulses, but by two different power levels of light. So optical fibre can also transmit digital signals.

To transmit bits over analogue links, a device for converting digital signals to analogue signals, and viceversa, is needed. This device, you may be already very familiar with it, is the MODEM. The term 'modem' comes from the words 'modulator' and 'demodulator.' Modulation and demodulation are the processes for converting digital signals to analogue signals and analogue to digital respectively. Here a modem can be regarded as the network adaptor, as with an NIC, for connecting the node to an analogue link. You may lay the cables and connect them to the network nodes by yourself if the network only covers one floor. However, if the network nodes are distributed very far apart, you may need to make use of one of the cabling systems that are already well developed by other companies. The largest cabling network is the telephone network. Figure 1.5 shows and illustration of modem operations.

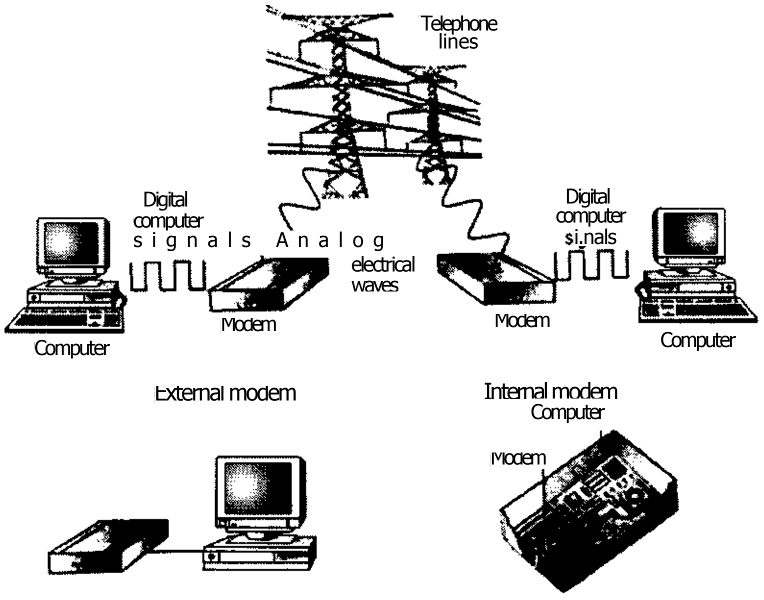


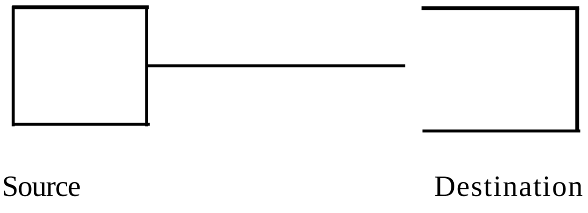
Figure 1.5 Modems convert digital signals to analogue signals, and analogue to digital

The telephone network was developed early this centry and is now a well-established global network. The telephone network is designed to transmit sounds, which are analogue signals, and therefore a modem is needed in order to transmit computer data over it.

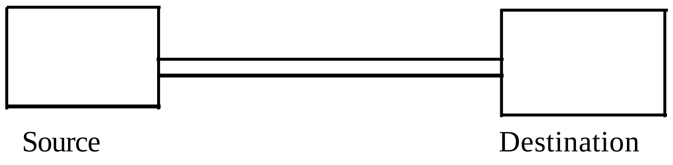
35 Mode of Data Transmission

Depending on the type of application, a link may be full-duplex, half-duplex or simplex. If two streams of data can be transmitted over the link in opposite directions at the same time, the link is said to be *full-duplex.* A practical example is the telephone network.

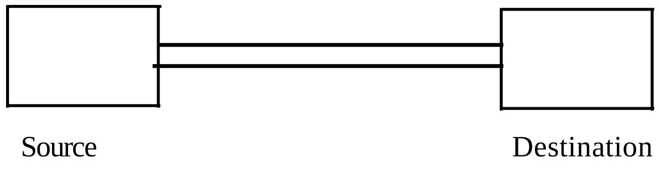
If a link allows data to be transmitted in both directions, but only in one way at one time, the link is said to be *half-duplex.* An example of this is the walkie-talkie communication system. *Simplex* links only allow data flow in one direction. Radio set and T.V set are examples. Figure 1.6 shows the different nodes of data transmission.



a) simplex



b) Half-duplex



d) full-duplex

Figure 1.6 Different modes of data transmission

In this unit, you have learned the fundamental building blocks of a network nodes and links. Their rationale in a network were also discussed. You should have learned about the two types of signals (analogue and digital), the role of a modem in a network over the telephone network. You need to be aware, also, of the different mode of data transmission in a communication system.

What you have learned in this unit concerns the fundamental building blocks of a network and their importance. It has also served to introduce you to the importance of modem in networking. The unit following shall discuss the ways of connecting many nodes together through different links.

You are required to discuss the importance of a modem in today's network to a company you are consulting for.

Exercise 1.1

Describe the Baseboard and the Broadband LAN.

Exercise 1.2

Discuss the importance of nodes and links in network.

Tanenbaum, A. *Computer Networks* (2nd ed.) Englewood Cliffs: New Jersey.
Prentice Hall, (1989).

Stallings, W and Van Slyke, R. *Business Data Communications,* (2nd ed.) New York: Macmillan College, Publishing Company, (1994) Chapter 5, pp. 124-129.

Online Materials

http://www.ccs.ed u/home/kemb/etc/partition/nodel.htm I
http://www.newton.dep.an I .govinewton/askasci/1995/compsc i/es134.HTM
http://www.com put i ng.netlli n ux/wwwboard/forum/6767.html
http://www.miine.Com/apuate06.pdf
http://www.support. Intel .Com/support+/faxmodem/4410.htm
http://www.digi I ander.io/. if/LeoTron/000/html/direction.html
http://www.weboped a.com/TERM/F/fu I I—duplex.htm I
http://www.iec.org/on I ine/tutorials/dwdm/
http://www.aciri .org/floyd.tcp—mux.html
http://www.its.bldrdoe.gov/fs-1037/dir-023/-3439.htm

Module 1:Network Basics and Architecture

Unit 3 : Network Topology

Page

This unit covers the way nodes are arranged in a network-topology. It will also exposes you to the various types of topologies and their different implementations. The unit guides you through some of the contemporary issues in topology, and the type of topology to be used for a particular application and the reason for it. Let us look at what you should learn in this unit, as specified in the unit objectives below.

By the end of this unit, you should be anle to:

- discuss freely the meaning of network topology.
- understand the various topologies available.
- determine the type of topology(ies) that suites a particular application.
- understand the various implementations of the various topologies.

3.0 Overview of Network Topology

The term 'network topology' refers to the way nodes are arranged and connected with links. Together with the physical medium chosen for implementing the links, it determines the speed of the network and the communication efficiency. Its selection depends on the geographic environment, the kind of applications running on the network and the implementation costs. Network topology is also a term that refers to the shape of the network and the layout of cabling. It shows how the various work stations (or nodes) and other network devices are linked for various reasons. In this unit, you will also learn that various topologies are available for various applications and implementations. There are three types of topology that are commonly in use: Bus, Star, and Ring. Although, we have Tree topology which is a combination of Bus and Star (otherwise called Hybrid Topology).

3.0.1 Bus Topology

The bus topology is one of the oldest types and remains one of the simplest forms to design and implement. Each node in the network is connected in sequence along a single network cable using T-shaped network interface connectors, and terminating points are placed at each end. A bus topology does not require a lot of cable and the wiring is simple. However, since devices at any point causes the entire network to go down. This single line also makes it difficult to troubleshoot and isolate faults. In addition, bottlenecks often occcur since nodes spend a great deal of time waiting for the network. To ensure no signals bounce back at the ends of the bus and interfere with the trailing signal, terminating devices absorb the signals to clear up the link. Figure 1.1 shows the arrangement of a typical bus topology.

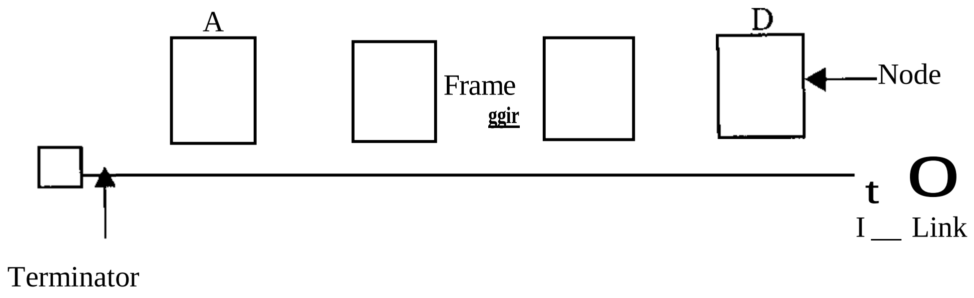


Figure 1.1 Architecture of bus topology

Suppose node A has a message to send to Node B. node A then delivers the message to the network via its network adapter. The message propagates in both directions of the bus until the ends are reached. All the nodes tapped to the bus can receive the message but only node B, which finds itself as the destination of the message, copies the massage from the network, not removes it. At the points of the bus, the terminators absorb the signals to clear up the link.

In order to ensure steady flow of data along the network, the bus topology uses a Media Access Control (MAC) protocol known as Carrier Sense Multiple Access with Collision Detection (CSMA/CD).

In this method, multiple workstations share acccess to a common network (multiple-access), but only one node can broadcast data at a time. Each node listens to the network to sense the presence of a transmission by another node (carrier sense). If no data is detected, the nodes assumes that the network is free or open and transmits its own data. If another device transmits at the same time, it detects a collision (collision detection). Thus, when workstation are ready to transmit, they check to see if more than one signal is present on the network. If a data packet is detected or a collision occurs, each station waits a random period of time and then tries sending again.

The major implementations of the bus topology are the Thick Ethernet (or 10 BASE 5). The thick ethernet uses additional device called the *tranceiver* It can cover a distance of 500 metres per segment and a total distance of 2500 metres for a maximum segments of five with the use of repeaters between segments. The thin ethernet can cover a distance 195 metres per segment and a total of 925 metres for a maximum of five segments using repeaters between segments. Let us now look at the advantages and disadvantages of the bus topology.

Advantages

i) Long distances possible especially with 10BASE 5.
ii) Noise immunity because of the STP.
iii) The architecture is conceptually simple.
iv) It is relatively inexpensive

Disadvantages

i) It is inflexible once installed especially IOBASE 5.
ii) It is fault intolerant since a break down in the bus cause the entire network to go down.
iii) It is very dificult to troubleshoot.
iv) It is susceptible to ground loops due to potential difference.
v) It uses specialized cable, e.g. 10BASE 2. If the network is changed, the cable has to be changed too.

3.0.2 Star Topology

The star topology is emerging as the most common network layout in use today. Each workstation is connected point-to-point to a single central location that is commonly referred to as a *wiring closet.* The wiring closet is a central switching station known as the *hub.* A hub is used to concentrate all the links into a single point in the network and it usually has multiple ports for multiple links to plug into. All messages must pass through the hub that contros the flow of data. This architecture makes it very easy for network administrator to re-configure the network. It permits centralized diagnostics of all network functions. One major drawback of this arrangement is that if the hub fails, the entire network will go down.
Figure 1.2 shows the arrangement of nodes in a star topology.

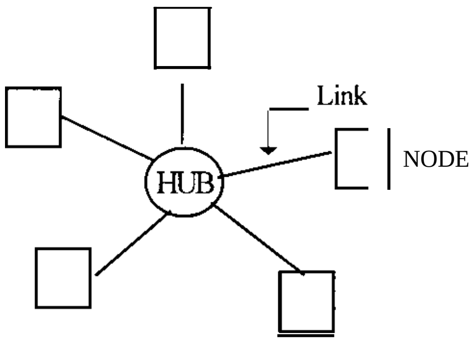


Figure 1.2 Star topology

Let us now discuss the major implementations of a star topology. Star topology usually comes in two flavours i.e. IOBASE T and IOBASE-FL. The I OBASE T uses either UTP or STP wires with RJ 45 jacks on both sides of the cable. This cable is used to link a network node with the hub, subject to a maximum distance of 100 metres. 10BASE-FL uses fibre cable to connect devices to the hub. It can cover greater distances.

We shall now consider the advantages and disadvantages of a star topology.

Advantages

i) It is fault tolerant due to partitioning. If a node is faulty, it does not affect the rest of the network.
ii) It allows for easy troubleshooting due to its architecture.
ii) It allows for ease of re-configuration.
iv) It is flexible as many buildings already has UTP cables. It is preferable to install UTP than any other cables as UTP will support other applications later.

Disadvantages

i) If area of coverage is seriously limited to 100 metres. This can be a major problem.
ii) It is sensitive to noise especially the UTP. This rules out IOBASE T out of an option for installation on factory floor environments.

3.0.3 Ring topology

The ring topology is another simple design that consists of a single cable that forms the main data path in the shape of a ring. Each node is connected to a closed loop of cable and signal travels in one direction from one node to all other nodes around the loop. Actually, each node is connected to the network via a repeater and the repeaters are connected by point-to- point links to form a ring configuration. The repeater, which is the interface at each node, is an active device (or ring interface) that has the ability to recognise its own address in the data packet in order to accept messages. The interface serves not only as a user attachment point but also as an active repeater for re- transmitting messages that are addressed to the other nodes. Figure 1.3 show the ring topology.

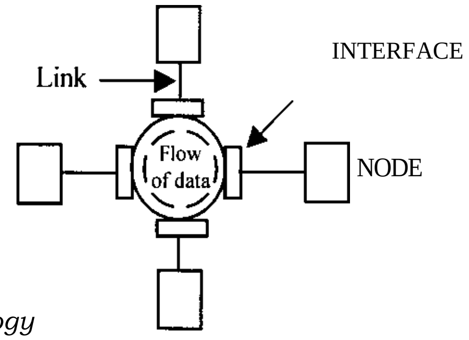


Figure1.3 Ring topology

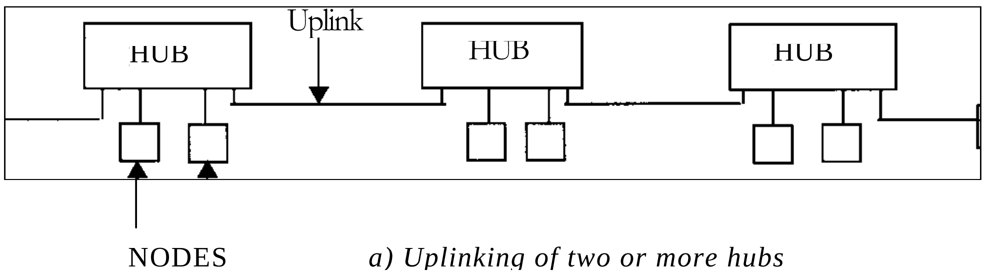
Each repeater has three connections: one to the node, the other two separate links connecting to another repeater at the other end. The repeater is responsible for receiving data from one link and immediately transmitting it bit by bit to the other link. If the repeater finds out that the message destined to the other node is attached to the message, it is then copied into the memory of the node. As there are no end points in the ring topology, you may wonder how the signals on the link are cleared, a message sent out from a node continues to circulate from one node to the other until the source node is reached. The source then absorbs the message thus clearing it from the network. the links used are uni-directional i.e. the signals transmitted in the ring only flow in one direction, either clockwise or anticlockwise.

The token ring is the MAC protocol that the ring topology uses in order to ensure that all nodes have equal access to the network. In this arrangement, the pre- condition for transmitting packets by a given node is to get hold of the token that moves around the ring. The token is a software code and the transmitting station releases the token to the next station after transmission.

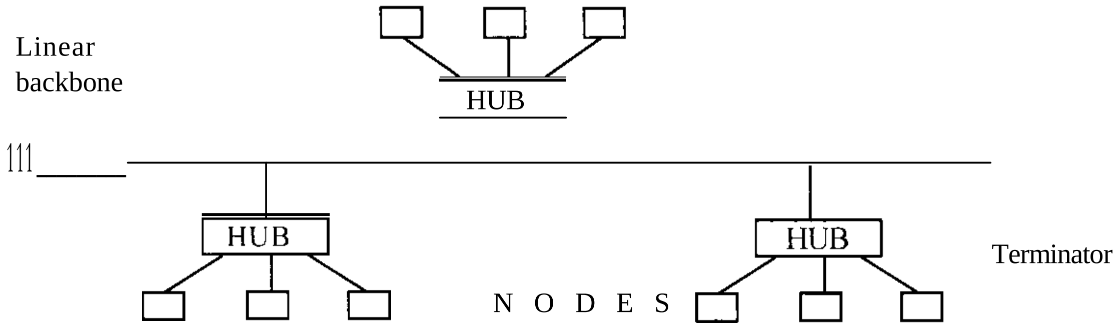
The ring topology, similar to the bus topology, cannot cope with a single point failure. Even worse, the failure of a node also hangs up the whole network, since any single point of failure in the link or any node failure makes the circulation of data in the ring impossible. As a result, some implementations of the ring topology use a dual ring configuration, where each of the nodes is connected to each other through two rings, one for normal operation and one for back-up use. An implementation of the topology is actually the configuration for the Fibre Distributed Data Interface (FDDI) and there will be discussion of this in a later unit.

3.0.4 Tree Topology

The tree topology is essentially a hybrid of the bus and the star layouts. The basic topology is similar to that of a bus, with nodes connected in sequence to a linear cable. But tree networks may have branches that contains multiple workstations that are connected point-point in a star-like pattern. Signal from a transmitting node travel the length of the medium and are received by all other nodes. Although, we have several configurations. Two or more hubs can be upl inked in a bus structure and nodes attached to them. Another is the one that has linear backbone (or bus) with hubs directly attached to the bus. Nodes are then connected

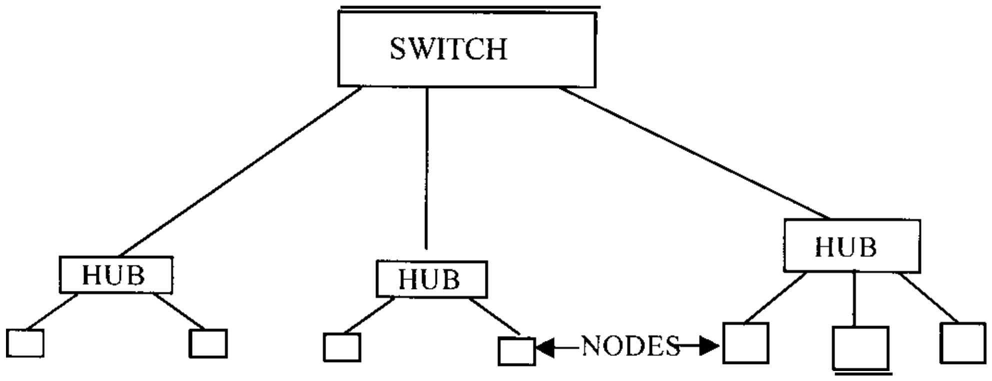


a) Uplinking of two or more hubs



b) Bus backbone with hubs attached

to the hubs. We may also have a switch that joins several hubs together. The tree topology is used when you are to network a multi-floor building. Figure 1.4 shows the tree topology, with several arrangements.



c) A switch that connects several hubs

Network topology is closely related to cabling. Some topologies may not be implented with particular type of cables. We will discuss this issue in detail in the next unit.

In this unit you have learned a number of important issues that relate to the ways in which nodes are arranged in a network. You should also have learned the various topologies available, their various implementations and applications. You need to be aware, however, of the strengths and weaknesses of each of them.

What you have learned in this unit concerns the arrangement of node in a network and its various types and implementations, including its applications. It has served to introduce you to the concept of network topology. In the next unit we will discuss how cable types affects the topology.

A client has just approached you on the proposed networking project of her newly completed 5-storey Head Office Complex. You are to give your advice on the best type of toplology to adopt and why.

Microsoft Corporation. *Network Essentials*, (2nd ed.), Redmond, Washington: Microsoft Press, 1996.
Peterson, L.L and Davie, B.S. (1996) *Computer Networks*.. A systems Approach, SanFrancisco: Morgan Kaufmann, 1996

Exercise 1.1

What do you understand by the term 'Network topology'?

Exercise 1.2

Describe the types of topology that you know.

Online Materials

http://www.es met/hypertex/we lcome/pr/topo logy. htm/
http://www.cybergeography.org/atlas/topology.htm/
http://www.weboped ia.com/quick-ref/ topologies.htm/
http://www.nordumet/connectivity/
http://www.verio.com/services/hosting/uptime.cfm

Module 1:Network Basics and Architecture

Unit 4: Network Adaptors and Cabling

Page

Having gone through the course guide, you must have had a general understanding of what this module is all about and how it fits into the course as a whole. In a network, each node actually connects to the link through a network adapter. The various types of links available will be considered in this unit. This unit will also guide you through the factors to be considered before chosing a network link. Finally, you will learn about the various connectors (or jacks) available and their applications.

By the end of this unit, you should be able to:

identity the various types of cables available.
appreciate the various types of adaptors available
carefully select appropriate cable(s) required for an application.

NglfrJipton

As discussed in the previous section, a node communicates with the network via a network adaptor. Actually, the network adaptor contains all the networking properties of a node, which includes the encoding and error detection issues discussed in a later unit. Network devices are different for different devices and also for different types of computers. Network adaptors for the computers are often called Network Interface Card (NIC). In this unit, we will explore the features of a network adaptor using a simple configuration of NICS, as an example for illustration.

N1CS are the interfacing hardware for computers to connect via a link. They are responsible for converting the computer data to a form that can be transmitted over the links, and also converting the incoming data from the link back to a form that can be understood by the computer. So an NIC must at least contain two interfacing components: one communicating with the computer, to which it is attached, called the *bus interlace*, and the other communicating with the link to the network, called the *link interlace*.

Buses are paths for data to move between different components inside a computer. The width of a bus conrols the number of bits that can be transmitted at one time; for example a 16-bit bus allows 16 bits of data to move at a time. The bus interface of an NIC transmits data to the computer through the buses. Actually, other hardware components such as video display cards and the disk controller cards also contain the bus inteface and they communicate in the same way as the NIC. Figure 1.1 shows the bus interface standards that are commonly used in PCs. Industry Standard Architecture (ISA), Extended Industry Standard Architecture (EISA) and Micro Channel Architecture (MCA) are older standards for PCs. ISA is the oldest standard supporting both 8-bit and 16-bit data paths whereas EISA is 32-bit path compatible with ISA. Micro Channel can be functioned with both 16-bit and 32-bit buses, but is not compatible with ISA. Peripheral Component Interconnect (PCI) is a 32-bit bus standard used in most Pentium computers and in the Apple Paver Macintosh. PCI also provides 'plug and play' functions and it works well with Windows '95 which is a plug and play compliant operating system.

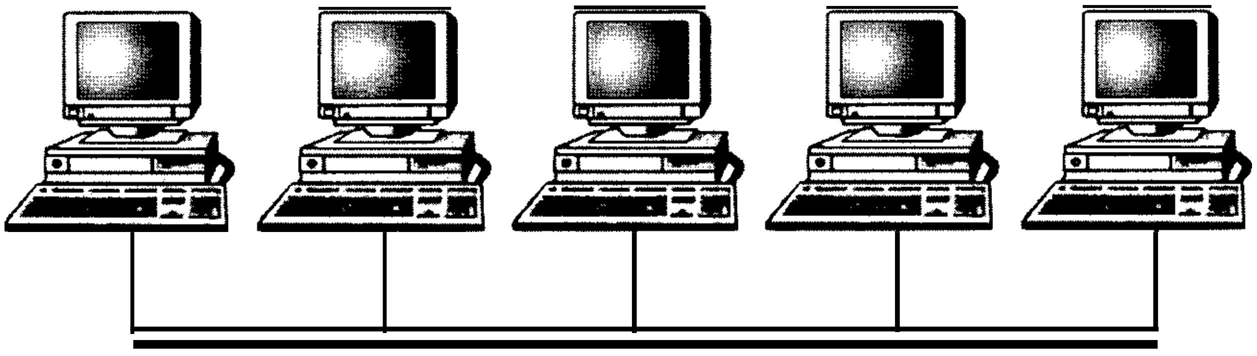


Figure 1.1 Bus Interface Standards

The link interface has connections for links to plug into. The data links can be classified into serial links or parallel links. The parallel links, which transmits or receives multiple bits in parallel through multiple tranceivers, are usually used for very short-hauled local communications such as the printer interface. The serial links, which transmit or receive data one bit at a time, are usually used for data communication at distances more than a few feet apart. As described above, data supplied to the NIC by the computer through a bus which is several bits wide needs buffering in between the two interfacing components to bridge up their speed gap. Figure 1.2 shows the block diagram for a network adaptor.

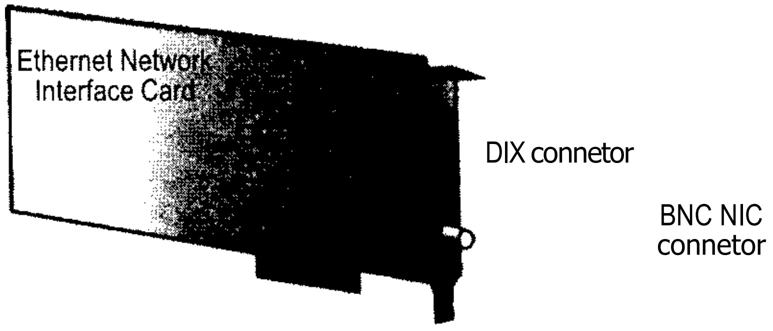


Figure 1.2 Network Adaptor

The links can only transmit electronic signals. So there must be a component in the NIC to translate the bits to electronic signals and this components is called the *tranceiver* (a term coined from the words transmitter and receiver).
Network adaptor have to be matched with the network technologies used in the network. For example, to connect a node to an Ethernet network, an Ethernet network adaptor should be installed on the node. In unit 6, we will discuss this in further details.

3.1 Cabling

You will recall at the beginning of this module, we discussed about the role of links in a network. Links are actually implementated by cabling, except in the case of wireless networks that use infrared light or radio waves for transmission through space. Cabling includes the installation of cable and outlets in the area covering the network. The principle is very similar to installing copper wires for electricity supplied; except that one electricity supply outlet (socket) can support more than one electric Appliances with the use of an adaptor, whereas one network cable outlet is usually only able to support one device. It is not easy to relocate cables once they are laid and so the cabling should be carefully planned ahead to cater for changes. Indeed, the installation cost for cables are significantly higher than the cost of the cable itself.

The selection of the most effective transmission system for a given aplication must be made in the context of a number of key design issues. There are several considerations in planning a cabling system. These include:

- Transmission characteristics which includes bandwidth, error performance, distance covered and attenuation.
- Propagation delay and response time.
- How safe is the medium in terms of security against espionage activities.
- Its mechanical strength.
- Cost implication i.e what the budget is for cabling.
- How large an area the network covers.
- How many users the network is going to support.
- How heavy the network traffic will be.

Physical dimension are sometimes considered.

The most important of these is the selection of cable type, as this affects the cost, installation method, flexibility of relocation and the reliability and speed with which data is transmitted. In sub-section following, you will learn about the characteristics and features of the three types of cables that are commonly used in today's network.

3.1.1 Coaxial Cable

Coaxial cable is a widely used wire in networking, partly due to the popularity of Ethernet. This is because Ethernet adopts a bus topology that can be easily implemented through coaxial cable being so popular are its low costs, high flexibility and simple installation. However the weak point for using coaxial cable is that any single point of failure hangs up the whole network.

Coaxial cable consists of a central core copper wire surrounded by a hollow outer cylindrical conductor, with dielectric materials filled in betweeen them, and an outer plastic cover to protect the cable from physical damage.

Figure 1.3 shows the structure of coaxial cable.

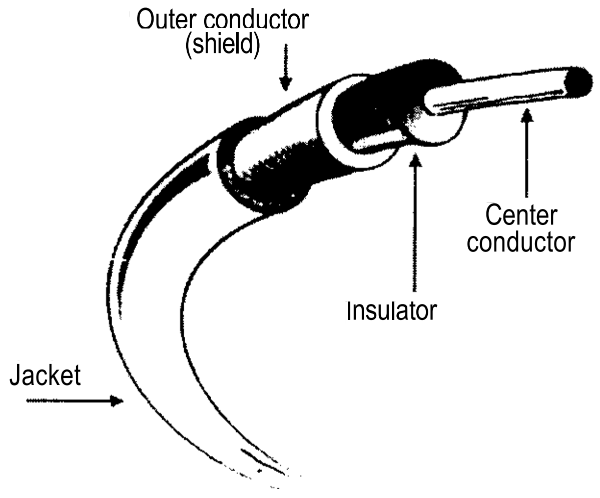


Figure 1.3 Coaxial cable

The core wire is responsible for transmitting electronic signals. The hollow cylindrical conductor protects it from outside electrical interference. The dielectric materials between them separate the two conductors and also help in keeping out electrical noise. If the core wire and the cylindrical conductor can touch each other, a short circuit is formed and electrical noise is the result, thus affecting the stableness of data transmission.

Two types coaxial cables are available: *thicknet* and *thinnet.* Thinnet cable has a shorter diameter, about 0.25 inches. Each cables segment can be joined to another by a Bayonet Navy Conductor (BNC) connector and, on attaching a node, the cable joins the NIC through a BNC T connector added on to the BNC connector, as shown in Figure 1.4

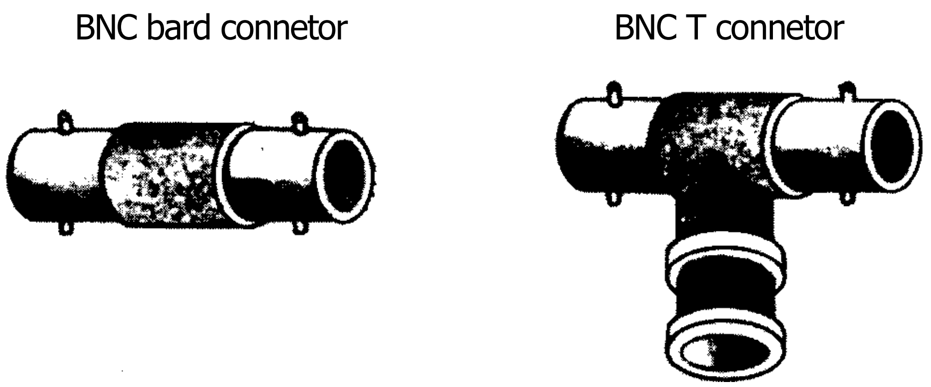


Figure 1.4 BNC cable connector and BNC T cable connector

There is a restriction that nodes must be attached to the cable at least 2.5 metres apart. Due to its high flexibility, thinnet cable is used in most network connections. As signals transmitting on a cable will attenuate, the length limit for thinnet cable is about 185 metres. Signal transmission is not reliable for cables with length longer than this limit. However, the distance covered by a thinnet cable network is not limited by its length limit because cables can be extended by connecting with a repeater. A repeater is a device that receive the signals and transmits again, so that the attenuated signal are amplified. But there must not be more than four repeaters in between two nodes, otherwise the transmission is unreliable. Thinnet cable only allows one digital signal to pass through at a time and is regarded as a baseband medium; its transmission rate is 10 Megabits per second (Mbps). Thinnet cable is also called 10Base2 cable. 10Base2 is part of the Institute of Electrical and Electronic Engineers (IEEE)'s specification IEEE 802.3. The term comes from three of its characteristics.'I0' from its transmission rate of 10Mbps, 'Base' from its baseband bandwidth and '2' from its length limit of nearly 200 metres.

Thicknet cable has a wider diameter, about 0.5 inches and is more expensive than thin coaxial cable. It has similar physical characteristics as thinnet cable, except it has a much longer lenght limit of about 500 metres. So thicknet cable is always used as a backbone for connecting several smaller networks using thinnet cables. Thicknet is harder to install due to its inflexibility. It is called a 10Base5 cable for the same reasons as in thinnet cable-with the '5' denoting the length limit of nearly 500 metres.

3.1.2 Twisted Pair Cable

As the name suggest, twisted pair cable consists of two copper wires twisted around each other. Usually two pairs of twisted pair will be used in the cable—one for transmitting data and the other for receiving data. The twisting is important as it cancels out the electrical interference from adjacent wires and the surroundings. There are two types of twisted pair cable shielded and unshielded. As you may deduce from their names, Shielded Twisted Pair (STP) cable has a protective cover around each pair of wires, whereas Unshielded Twisted Pair (UTP) does not. These two types of twisted pair are shown in Figure 1.5

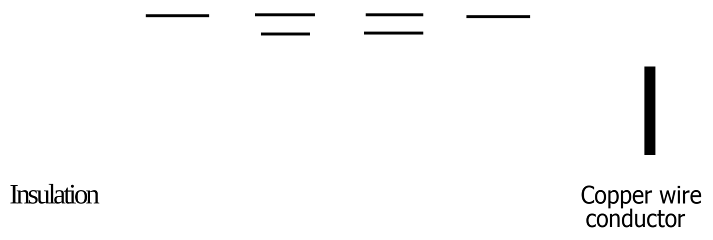


Figure 1.5 Twisted pair cable

The Electronic Industries Association and Telecommunication Industries Association's Commercial Building Wiring Standard 568 (EIA/TIA-568 standard) defines UTP as a standard in building and wiring situations. Fire categories of UTP have been specified in the EIA/TIA-568 standard and are listed in Table 1.1.

Table 1.1 The EIA/TIA-568 standard for UTP cables

Standard	Brief description
Category 1	consists of telephone cables, for voice communications and is not suitable for transmitting data.
Category 2	consists of four twited pairs and is capable of transmitting data at speeds up to 4Mbps
Category 3	consists of four twited pairs and is capable of transmitting data at speeds up to 10Mbps
Category 4	consists of four twited pairs,used in Token Ring networks and capable of transmitting data at speeds up to 16Mbps
Category 5	consists of four twited pairs and is capable of transmitting data at speeds up to 100 Mbps.

In recent years, UTP has taken thin coaxial cable's place and became the most popular cable used in networking. Beside its extremely low cost and easy installation, an important reason for its popularity is that UTP cables for data transmission are always pre-installed with telephone cables in new buildings. Since telephone cable and UTP cable for data transmission are actually the same type of cable, the pre-installation can be done with minimal effort. The topology used in UTP cabling also contribute to its popularity. UTP cabling uses star topology, usually with a hub as the central node for connecting the other nodes. An Ethernet network adopts a bus topology and a Token Ring network adopts a ring topology. So for Ethernet networks implemented with UTP and a hub, the internal struture of the hub actually is a bus. You may imagine the hub as a very short bus, instead of the traditional long bus. The appearance of the network is a star, but in actual fact bus topology is running. In this case, we say the physical network topology is star but the logical network topology is bus. In the case of Token Ring implementation with UTP and a hub, the physical network topology is star and the logical network topology is ring.

The length limit for UTP is 100 metres. Hubs may also be used to extend the cable segment. However, there should not be more than four hubs between any two nodes, otherwise signal attenuation makes the data transmission unreliable. Category 5 (often called 'Cat 5 cable') is ususally used nowadays and the transmission rate can be up to 100 Mbps which is used in Fast Ethernet networks;10Mbps is used in Ethernet networks. UTP cable is also called a 10 BASET cable. Like 10Base 2, it is part of the IEEE 802.3 specification. Similarly, the '10' stands for 10Mbps, 'base' for Baseband and the 'I' for twisted pair'.

Shielded twisted pair cable has foils rapped outer cover. This enables STP to have very strong resistance to outside inference and it is, thus, capable of transmitting data for a longer distance. STP is usually used in environments that have a high electromagnetic background, such as an electric power plant or floor of factory.

In connecting to a node, twisted pair cable is joined to the NIC by an RJ-45 connector (RJ stands for Registered Jack or Remote Jack), as shown in figure 1.6. Inded, twisted pair cable is a point to-point link, the two neds of each segment are attached with two RJ-45 connectors, one joining the NIC and the other joining a port on a hub.

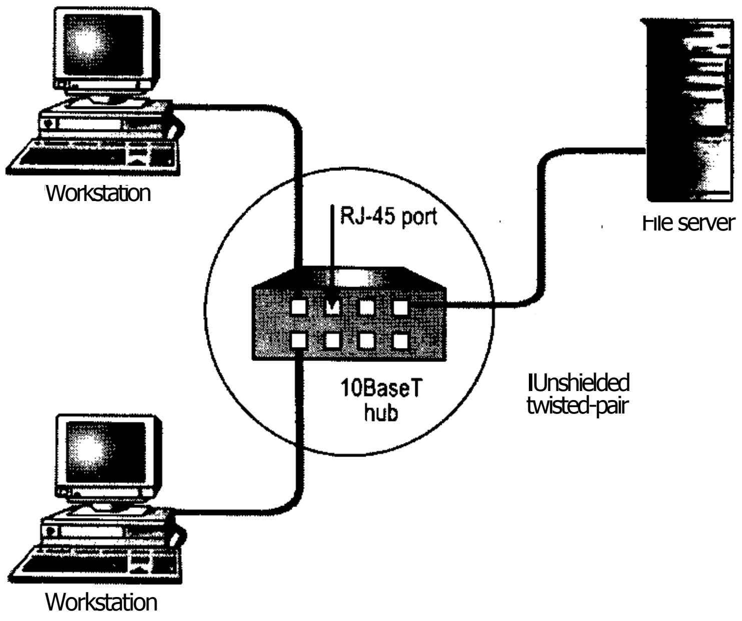


Figure 1.6 RJ-45 connector for UTP cables

3.1.3 Fibre Optic Cable

Optical fibre is an extremely thin glass strand. The fibre cable consists of an optical fibre, called the core, surrounded by a concentric layer of glass, called the *cladding,* which is in turn covered by an outer plastic jacket for protection from physical damage. It appearance is shown in figure 1.7. Unlike the coaxial and twisted pairs cables, optical fibre transmits data by sending modulated pulses of light, not electronic signals. For this reason, it is not susceptible to elecromagnetc interference. The core of the fibre optic cable is sometimes made of plastic for easier installation, but then the transmission length becomes shorter than that for a glass core.

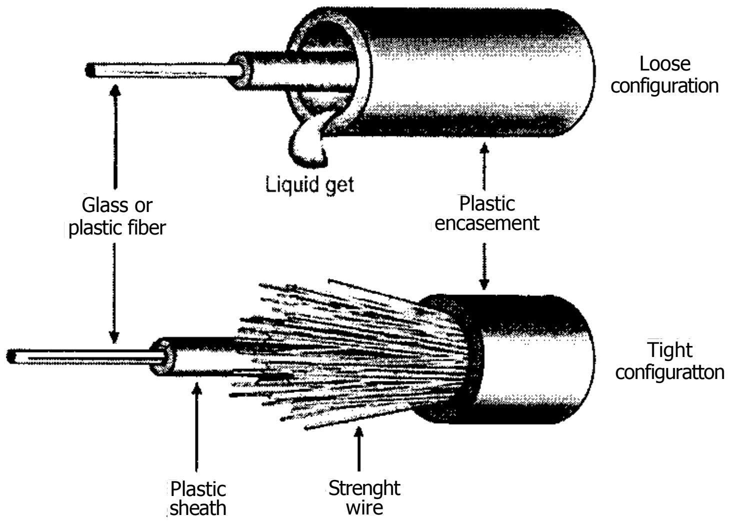


Figure 1.7 Fibre Optic Cables

Fibre optic cable is the most expensive cable used in networking, but it is capable of higher data rates. In the fibre tutorial, you came across the two types of fibre cable: *multi mode* and *single mode*. The term 'mode' is used to describe a light path through a fibre optic cable. For multimode, a light pulse propagates in a number of modes in the fibre. Since the length of each path is different, the time for the light wave to pass over a given distance will also be different. This effect is called dispersion and it limits the brandwidth of the cable. A single-mode fibre has a smaller core, which is about the same order of magnitude as the wavelength of the incident light wave. A light pulse is able to propagate in only one mode in the fibre. Typical bandwidth for multimode is about 100 Mbps with a distance limit of approximately 2 kilimetres. Single-mode fibre provides a much higher brandwidth and a longer distance limit. The connection of fibre optic cable together is difficult due to the fact that light fine precisions is required for alignment and the fact that light travels in straight line. A small flow on the fibre may lead to signal attenuation and so except installation is needed.
10 Base F is used to refer to the Ethernet specification for fibre optic cable. Like 10 Base 2, it is part of the IEEE 802.3 specification. The '10' stands for 10Mbps, 'Base' for baseband and the 'F' for fibre optic.

3.2 A Summary of Cable Types

The following table provides a further summary of the different cable types

Table 1.2 Summary of Cable types

	Coaxial cable		Twisted Pair		
Features	Thiumet	Thickmet	Unshielded	Shielded	Optic fibre
Cost (expensiveness)	Fourth	Third	Least	Second	Most
Length Limit	185 metres	500metres	100metres	100metres	2 Kilometres
Transmission rates	10Mbps	10Mbps	100Mbps	100Mbps	100Mbps or more
Flexibility	Third	Fourth	Most	Second	Least
Resistance to electrical interference	Fourth	Third	Least	Second	Best

Installation (difficulty)	Third	Fourth	Least	Second	Most
Usage	Widely in Ethernet networks in in the past	used as a backbone to connect smaller thin coaxial cable networks	Replacing thin cables (i.e coaxial) to become the most popular cables used in Ether-net networks.	used for implementing network in environments with high electrical power plants	used as backbone for long distance connections

3.5 Cabling Vs Network Topology

Theoretically, all cable types can be used in implementing any network topology. However, this is not the case in practice. Fibre optic cable is not used in bus topology because it does not easily branch out. Besides, it is not economical to use thicknet in ring or star topologies. The relationship between the topology and cable types are summarised in Table 1.3.

Table 1.3 The relationship between the network topologies and cabling types.

Cabling	Network Topology		
	Bus	Ring	Star
Twisted pair		1	1
Thinnet	1	1	
Thicknet	1		
Fibre optic		1	1

In this unit, you have learned about the types of connectors and adaptors that we have and their specific uses. You have learned also the different types of cabling available for networking and their characteristics and and applications. You need to be aware of the factors to be considered before selecting media for networks. By now, you should be able to compare the various types of media available using some performance metrics discusses in section 3.2.

ft1i.

What you have learned in this unit concerning the rationale for different types of adaptors and connectors. It also exposes you to different type of cables and their applications. In the next unit, you will learn about one of the most important software components of a network - Network operating System- that drives these connectors, adaptors and cabling.

The UTP has become the most popular cable in today's business. Discuss.

Exercise 1.1

Describe the features that influence the choice of an effective transmission medium.

Exercise 1.2

Describe the structure of fibre Optic cables

Microsoft Corporation. *Network Essentials, 2nd* ed.Redmond, Washington: Microsoft Press, *1996*

Online materials

http://www.pcIt.cisyale.edu/pcIt/pchw/BUS.HTM
http://www.itee.uq.edu.aut-- mesh/list/msg00038.htm/
http://www.cablenetworking.co.uk/sitemap.htm http://www.netserve.anu.edu-au/operations/networks/specs.htm/ http://www.networkcables.com/company.htm
http://www.dnuoz.org/Computers/Data—Communications/Ethernet/Distributor/

Module 1:Network Basics and Architecture

Unit 5 : Network Operating System

Page

kb' ti

By now you would have had a general knowledge of what a computer network is. What we have dicussed are hardware components. You will learn in this unit about the network—what is known as the Network Operating System. This unit will guide you into some of the functions performed by NOS and some types of it Let us now look at what yoou will learn in this unit, as stated in the unit objectives below.

2.0 Objectives

By the end of this unit, you should be able to:

- explain the rationale for NOS in computer networks
- understand the basic functions of NOS
- Identify the various types of NOS that we have
- Discuss freely the specific characteristics of each of them.

3.0 Network Operating Systems (NOSs)

The importance of software to computer systems cannot be over-emphasised. The hard ware components of a network cannot work unless there is requisite software to drive them. The network operating system plays a critical role in networking architecture as it controls all the networking hardware and their communications with the network. NOSs for client/server network in which some nodes provide services like file storage and printing (known as server) and the other nodes take services (known as clients)- usually have two different programmes, one for the client side and one for the server side. In contrast to the client/server architecture is the peer-to-peer architecture, in which all the nodes can share their own resources to the other nodes and at the same time enjoy the services provided by the others. With NOSs for peer-to-peer architecture, only one programme is needed for all the nodes. However, only small networks(usually less than ten nodes) adopt the peer-to-peer architecture since the traffic congestion may be serious if all the nodes share their resources in a large network. Thus client/server architecture is the most commonly used in networking. In this section, we concentrate our discussions on NOSs for client/server architecture.

3.1 Software Components of a Network

It is pertinent to note that we have two other important components of a network apart from NOS that co-exists with it for proper functioning of the network. These are discussed in the following sub-section.

3.1.1 The Workstation Operating System

This operating system software is loaded at the workstation. This component is essential for the effective operation of the workstation. Examples of workstation operating system are Microsoft Windows(95, 97,98,2000), Microsoft Windows NT workstation, MS-DOS etc.

3.1.2 The Network Shell

The network shell is created by the network operating system but it is loaded on the workstation. This shell is formed around the operating system of the node and it filters out commands intended for the server before the operating system of the node can receive them. The shell software also determines the stations of the request at the node.

3.2 Functions of Network Operating System

There are numerous functions performed by the network operating system But in this section, the basic functions that should be performed by an NOS may be summarized as:

Support for Multiple Users

The server side of a NOS should support multiple users accessing the server's data and services concur-rently.

Support for Multiple Tasking

As multiple users may run applications on the server at the same time, the server side of a NOS should also support multiple tasking to deliver quick responses.

Support for Multiple Communication Protocols

Clients on a large network may run different NOSs and thus use different communication protocols. It is important for the server side of an NOS to support multiple communication protocols so as to allow communication with clients of different platforms.

Security Control

Different users on a network may have different right on using the data and services on the servers The server of a NOS should have capabilities for security control for network resources administration.

Scalability

A NOS should be able to cater for an increasing number of nodes on the the network.

Fault Tolerant

As the data and services on a server are shared by the network users, the daily operations may be seriously affected if a user is down. A NOS should have the ability to tolerate faults imposed on the server.

Graphical User Interface

Using graphics to display the huge amount of network data and information can help users visualize the overall network status easily. Visualisation becomes increasingly important for NOS operations.

3.3 Types of Network Operating System

You are now aware that a good NOS is one that can support an increasing number of users, increases in network traffic, different communication protocols and now appliocations. Some NOSs may be software runing on an operating system, a good example is Novell's NetWare which is a very popular NOS for PCs. Others may be integrated with an operating system like UNIX and Windows NT . As Neyware, UNIX and Windows NT are successful NOSs with significant market shares and we will discuss each of them in the next sub-section.

3.3.1 Novell NetWare

In the 1980s and early 1990s, Novell's NetWare was the most popular NOS and occupied over 60% of the market. NetWare was actually the first lage-scale network product for IBM- compatible PCs. It supports both Ethernet and Token Ring networks, which are the most popular technologies used in networking implementation. Furthermore, NetWare provides a high level of network services such as file sharing, printing management, messaging (electronic mail), security control and run ing applications across networks.

As described above, NetWare is a software runing on an Operating System (OS). There are various operating systems supporting NetWare, including Disk Operating System (DOS), Windows, OS/2, the Macintosh OS and Unix, As you can see, the operating systems are not just IBM—compatible PCs as NetWare has extended itself to other machine types. Netware is a NOS for client/server architecture and it requires the machine with the server-side programme to be a dedicated server— that is, a machine only used for providing services to other machines on the network and which cannot be used by users as a work station. Any machine on the network that wants to access the server's services should be installed with the client-side programme. The client-side programme is actually a software 'shell' for the local operating system of the machine. The shell is a programme for accepting user commands. It distinguishes between commands for the network and commands for the local OS. If the command is for the local OS, the shell passes the command back to the local OS for the execution. If the command is for the network, it is executed by the shell. This process is shown in the block diagram in Figure 1.1, for DOS-based machines.

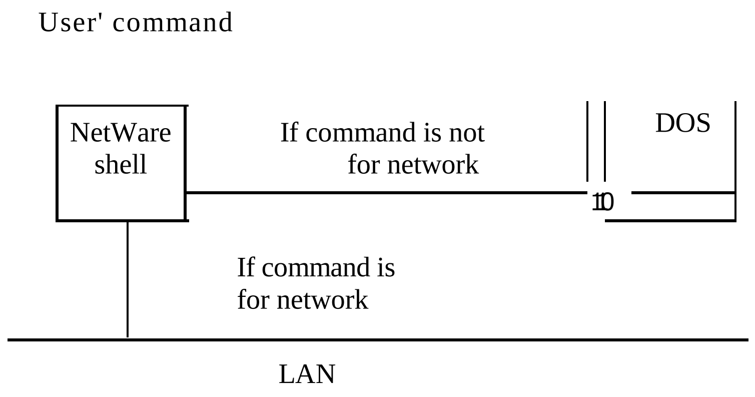


Figure .1 Block diagram for Nelware Shell

Basically, Network provides a text-mode user interface. Users have to type in DOS ___ like for accessing its services. But a simple Windows utility has been developed for accessing some basic services such as printing and file management through a graphical user interface. Netware supports communication protocols over Internetwork Packet Exchange / Sequenced Packet Exchange (IPX/SPX) and Transmision Control Protocol over Internet Protocol (TCP/IP) which you will encounter in later unit's. However, very often only IPX/SPX is used as it is a protocol developed by Novell. TCP/IP is a protocol widely used in the Internet world.

As a dedicated server is compulsory, NetWare has been criticised for not supporting peer-to-peer networking. Novell developed a product called NetWare Life for small peer-to-peer networks inthe early1990's. However, it was not as successful and was soon passed over by Windows NT which is a NOS for both client/ server and peer-to-peer architectures. NetWare allows the extension of its system services by using NetWare Loadable Modules (NLMs). Computing professional may write their own NLMs for additional or tailor—made network services for their own use.

3.3.2 Windows NT

Windows NT (NT stands for New Technology) was developed by Microsoft in the 1990's and has become increasingly popular. Newly implemented networks are opting for Windows NT as their NOS rather than NetWare. There are several reasons for this:

- With the increasing popularity of the Internet, it is always a requirement for the office networks to have servers for Internet services such as Domain Name Services (DNS) and electronic mail. Windows NT has built in services for implementing these servers. The configurations for starting up these servers are very simple too.

- Compared to the text-mode interface in NetWare, Windows NT is more user-friendly. Windows NT provides the same graphical user interface as that used in Windows. It is easy for users who are familiar with Windows to get used to it quickly.

- Windows NT supports mutiple communication protocols including TCP/IP, IPX/SPX and NetBIOS Extended user Interface (NETBEUI). You may enable all the protocols in the Windows NT network for supporting different communications. NetBEIU is usually used in LANs with no access outside networks like the Internet. TCP/IP is enabled for networks in a Wide Area Network (WAN) or having access to the Internet and IPX/SPX is used for communication with the NetWare network.

- Windows NT is most often used in implementing enterprise network with multiple NOSs since it supports the integration of network services from NOSs like Unix and NetWare. For example, it can send print jobs to printers controlled by a Unix server and it can access the data and services provided by a Netware server.

- NT File Sysrem (NTFS) is one of the file systems used by Windows NT to provide software fault tolerant services such as disk mirroring. In addition, long file names, as used in Windows 95 and Macintosh OS, are supported.

• Windows NT provides many network management tools like network traffic monitoring and protocol analysis. This helps network administrators in managing the networks.

3.3.3 Unix

When Unix was first developed by Berkeley University, it had no obvious aim as a Network Operating System. In strict definition, Unix is not a NOS, but an operating system with built-in networking features.

Various versions of Unix were then developed by multiple intitutions and companies to make Unix available on different platforms.

In the past, Unix only provided a text-mode user interface and the command used were quite difficult to remember. With the help of X-Windows, a software providing a graphical user interface for Unix, the Unix system has become more user-friendly. However, compared to NetWare and Windows NT, more expertise is still needed in managing its operations. Unix only supports the TCP/IP protocol and Internet services such as electronic mail can be implemented on it. However, the configurations for these servers are not as easy as those in Windows NT.

Unix has its advance features in administration, diagnostics, system utilities and other networking services. Besides, there are many software utilities for Unix available on the Internet as shareware. This makes Unix rich in resources for system and network management. Software applications developed for Unix on a particular platform usually run on other platforms after recompiling the source code. This shows the high portability for Unix applications and makes Unix a very good network server for runing applications serving the other machines on the network.

In this unit, you have learned the importance of software, especially Network Operating System (NOS) in a network. You should also have used this to deduce that it serves as the 'driver' for the networks hardware resources. Furthermore, you learned also about the functions performed by a NOS, and the various types of NOS and their characteristics and features. You need to be aware, however, that the importance of NOS in a network cannot be over emphasised.

What you have learned in this unit concerns the importance of NOS to a network. It has also introduced you to the various types of NOS available, their features, and characteristics. Numerous functions performed by the NOS were dicussed too. The next unit shall build upon this.

A computer network is useless cluster of PCs, cables, adapters etc. without the appropriate NOS to drive it. Discuss.

Exercise 1.1

What are the functions of NOS?

Exercise 1.2

Describe the structure/features of UNIX

Microsoft Corporation. *Network Essentials* (2nd ed). Redmond, Washington: Microsoft Press, 1996.
Peterson, L.L. and Davie, *B.S.Computer Networks:* A system Approach, 1996

Online materials

http:.//www.fc it. coed u. ustieduMetwork/software.htm
http:.//www.m icrosoft.com/traineertisyl labi/2151Afinal.asp
http:.//www.infoworld,com/cgi-bin/displayTC.p/?97poy.win3.htm
http:.//www.novel I .corn
http:.//www.novell.com/press/arch ive/1997/03/pr 97033. htm/

Module 1:Network Basics and Architecture

Unit 6 : Network Technologies

Page

....

In unit 3, we discussed about the various network topologies available. The way the nodes that are attached to a network accesses the medium is very important as digital links can only support the transmission of a single electronic signals at one time. In this unit, you will learn the various ways by which nodes access the medium in order to avoid collision. Let us now look at what you will learn in this unit, as enumerated in the unit objectives below.

By the end of this unit, you should be able to:

- understand the concept of collision.
- illustrate how media access control mediate access by multiple nodes to a comma link.
- illustrate the various MAC protocols available.

....

A link may be shared by a number of nodes. However, digital links can only support the transmission of a single electronic signal at one time. More than one signal using the same link may result in a collision of signals as shown in figure 1.1. That means only one node can use the link at one time. So, which node should use the link first? Which node should be next? Does it mean that there should be a controller to determine the order? There are several network technologies developed to mediate access by mu ltipe nodes to a common link; two of these are Ethernet (CSMA/CO) and Token Ring.

We will have a closer look at these technologies to see how they work. As the network adaptor determines the network properties of a node, you need to buy different network adaptors for a node to connect to networks of different technologies.

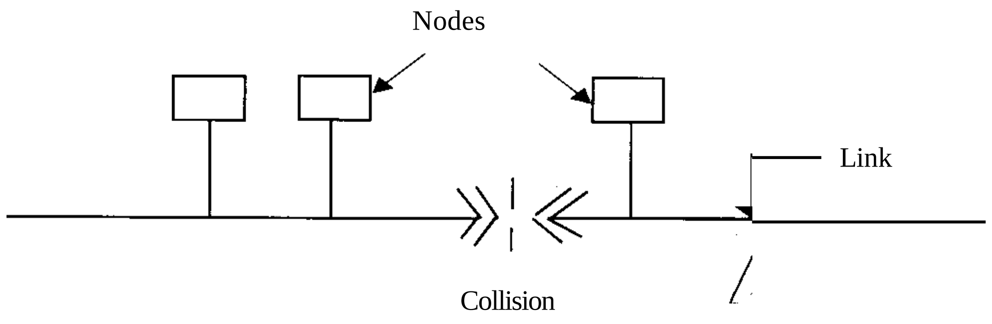


Figure 1.1 collision of two signals on a common link

The network technologies discusssed in this section are LAN implementations only. The technologies used in WANs will be discussed in Modules Two: Network Design and Maintenance.

3.1 Ethernet (CSMA/CD)

Carrier Sense Multiple Access with Collision Detection (CSMA/CD) is a technology used to share a common link among multiple nodes. Ethernet is an implementation of CSMA/CD which was developed by the Xerox Palo Alto Research Centre (PARC) in 1970s and it is now the most popular technology used for LAN implementation.

In Ethernet networks the network adaptor of each node always monitors the link to see if there is another node transmitting data on it. This actually is the meaning of Carrier Sense (CS) in CSMA/CD. All the nodes attached to the link know about the state of the link; that is, whether it is idle or being used by someone. A node transmits data to the link immediately when it finds the link free as show in figure

1.2. Multiple Access (MA) means that it is a technology used in multiple access network. There may be more than one node which wants to transmits data finding the link idle and submit their data into the link at the same time. In this case, a collision occurs. Collision Detection (CD) means that nodes are able to detect collisions during data transmission. If a collision is detected, the node transmits a jamming signal and immediately stops transmitting. This ensures all the nodes on the link know there has been a collision. Each sending node then waits for a random unit of time before retransmitting so as to reduce the probability of collision again. If there is a further collision or retransmission, the sender doubles the waiting time for retransmission until reaching a limit. The flowchart representing the generic carrier sense protocol is as shown in figure 1.3

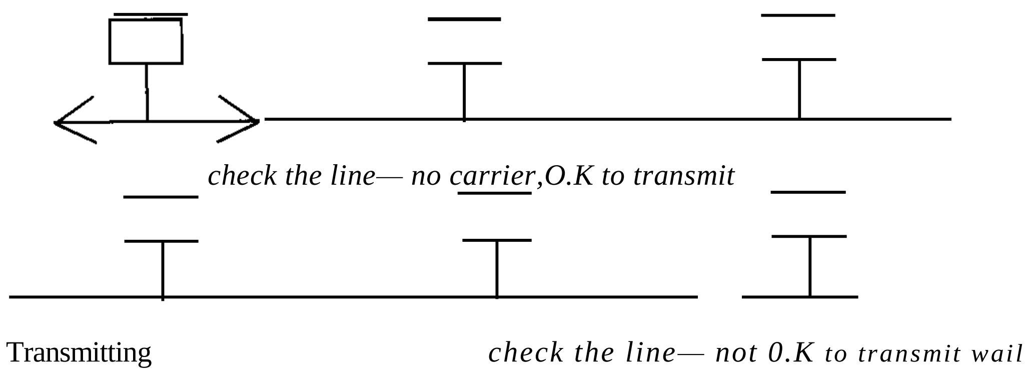

check the line— no carrier,O.K to transmit
Transmitting
check the line— not 0.K to transmit wail

Figure 1.2 Computer transmits data when the cable is free

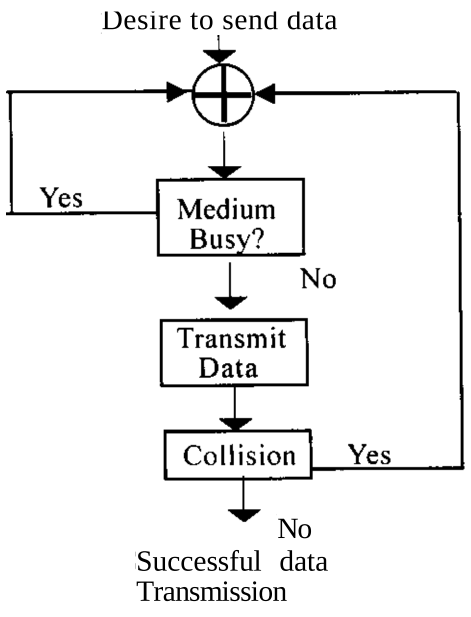


Figure 1.3 Operation of the Carrier Sense Protocol

Obviously, a considerable amount of bandwidth is wasted in collision. Consider the worst case that a collision occurs between two nodes A and B at the two ends of the link as shown in figure 1.4. We assume the end-to-end propagation delay to be *a*. At time *t,,,* node *A* find the link idle until it receives the first bit of the frame from node *A*. If *B* starts transmitting data at the same time just before the frame from node *A* arrives (at a time near $t_o + a$), a collission occurs and node *B* detects it immediately. However, node *A* will not see the collision until the corrupted frame reaches it at a time about $t_o + 2a$.

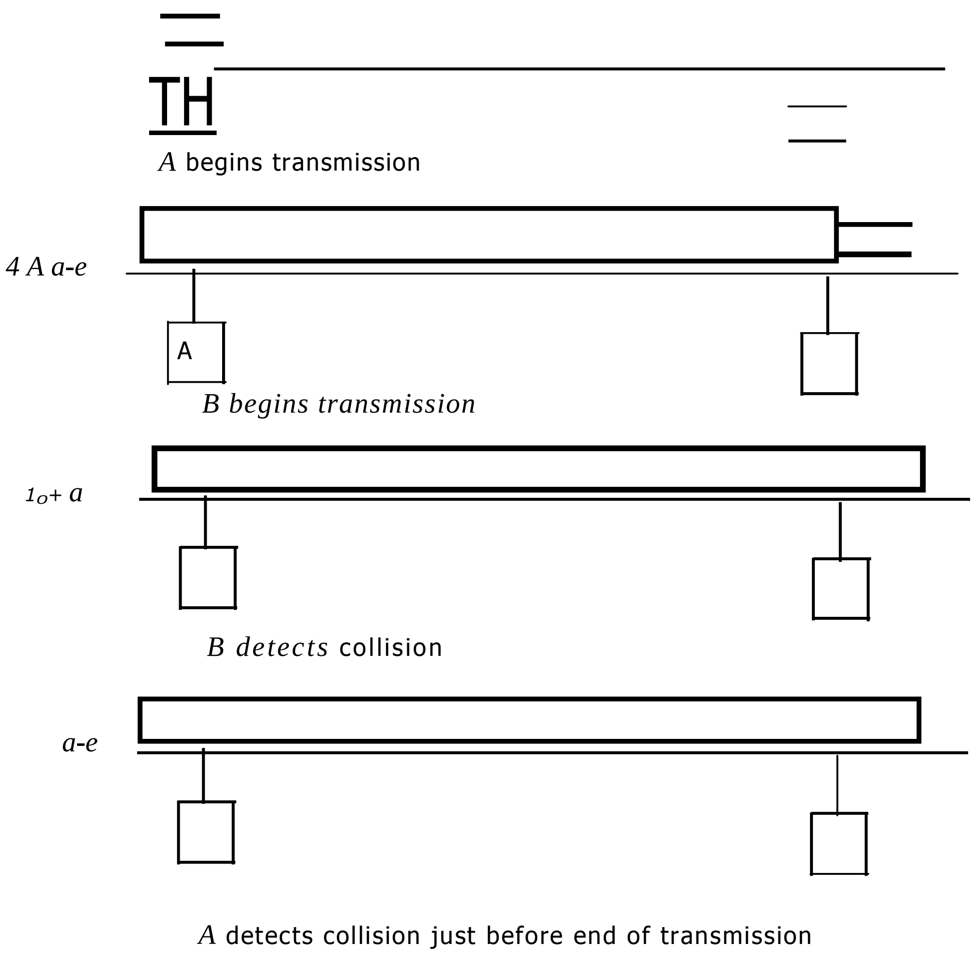


Figure 1.4 Collision in Ethernet

If the transmission time for node *A is* shorter than *2a,* node *A will* complete the transmission before knowing about the collision. This results in an unreliable transmission: node *A* thinks that the transmission is successful but actually it is not. So the transmission time has to be greater than *2a.* is imposes a minimum frame size in the Ethernet.

The following shows that the minimum size for an Ethernet frame should be *2aC,* where *C* is the speed of the link in bits per second.

Assume the time for transmiting a frame be *t* and the frame size be *S.:*

t =S/C

As discussed above, the transmission time has to be greater than `*2a..*

$>= 2a$

$S /C >=2a$

$S >=2aC$

In order to reduce collision, some of the algorithms may be `*p* - persistent. What does this means? When a node has a frame to send and the link is busy, its network adaptor waits and listens to the link until it goes idle and then transmits the frame immediately. If more than one node has the same status as this,collision may easily occur once the link goes idle, since the network adaptor may send out their own said to be 'I - persistent'. That is, a network adapter has a probability of *p (0 <p<l)* on sending

frames when the link goes to idle. Each adapter may then send (Cut their frame at a different time even though they find the link idle. Collision may be greatly reduced.

3.1.1 Ethernet Address

Each node on an Ethernet network has a unique Ethernet address. This address is burned into the Read Only Memory (ROM) of the network adaptor so if the network adaptor of a node is replaced, its Ethernet address is also changed. The form of an Ethernet address is 8 bytes long, composed of a sequence of six hexadecimal numbers separated by colons; *a4:23:2b:e4:b1:d2* is an example. Indeed,each Ethernet address is unique in the world. To ensure this, each Ethernet device manufacturer is allocated a different prefix of Ethernet address. For example, Advanced Micro Devices (AMD) is assigned with a heading 8:0:2. Each frame (block of data) transmitted on an Ethernet will contain the Ethernet address of the destination node. Each adapter on the Ethernet will suck each frame up from the network and compare the destination address on the frame with its own Ethernet address to see whether the frame is destined to itself. This actually is an example of `unicost' addressing. In the case of a message broadcast (that is, a message addressed to all the nodes on the network) the broadcast address, which is an Ethernet address of all Is, is used. Beside unicast and broadcast there is a multicast operation. More than one destination address is included in the frame. A special field in the frame will be set to indicate the frame including multicast information.

3.1.2 Ethernet Implementation

Ethernet is usually implemented with coaxical cable. Etehrnet adpts a bus topology and so any node can be easily tapped into the bus. On coaxical cable, a T-connector is used to branch out to a node. Each node should be at least 2.5 metres apart, otherwise signal reflection may give a false collision indication. Each cable segiment should not exceed 185 metres. If you want to build a network in an area exceeding this limit, you may use a repeater to extend the network. A repeater will receive signals from one cable segment, amplify and transmit it to another as shown in figure 1.5. However, you may not use more than four repeaters between any two nodes in the network. As introduced in the last section, a signal sent out from a node will propagate in both directions along the bus until the end. It is the same case in Ethernet implementation. A terminator at the end will absorb the signals and thus prevent it from bouncing back and interfering with the trailing signal. Nowadays, UTPs are also commonly used in Ethernet.

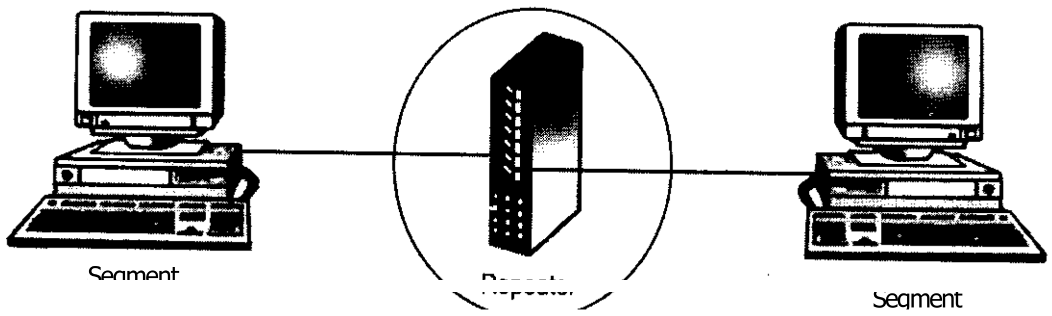


Figure 1.5 Repeater amplifies attenuated signal

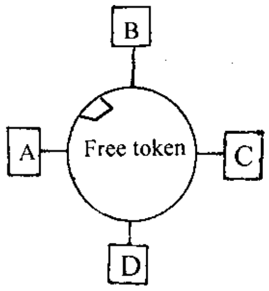
Ethernet uses the Manchester encoding scheme, which is described in the next unit, to encode the bits into electronic signals. The transmission rate for Ethernet is 10Mbps, but this number does not reveal the full facts. If there is only one user using the network, then 10Mbps transmission rate can be achieved. However, if there are two users using the network, collision may occur and retransmission is needed; so the effective transmission rate drops. If there are ten users on the network, the effective transmission rate is approximately 7Mbps. This drops again to 3.5 Mbps for a 100 users. Newly developed Ethernet technologies have a faster speed. Fast Ethernet supports 100Mbps and is now emerging as a standard. UTP catyegory 5 cable is used to implement the network. Developed by an alliance of 28 companies, Gigabit Ethernet supports 1,000Mbps. UTP category 5 cable is also used in its implementation with a length limit shortened to around 25 metres.

3.2 Token Ring

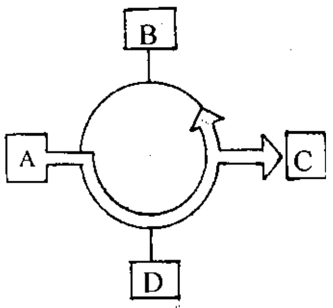
Token Ring is another network technology used for sharing a common link on a network. It has many different implementations such as IBM's 16 Mbps Token Ring, IEEE 802.5 Token Ring and the Fibre Distributed Data Interface (FDDI). In this section, we use FDDI as an example for illustrating Token Ring technology as it covers most of the details.

Token Ring is implemented on a ring topology, in constrast to the bus topology in Ethernet. For a ring topology, nodes are connected with several point-to-point links and frames flour in one direction - a node receives frame from its upstream neighbour and passes them to its downstream neighbour, and copies the frame to its internal memory if it finds the frame destinated to itself.

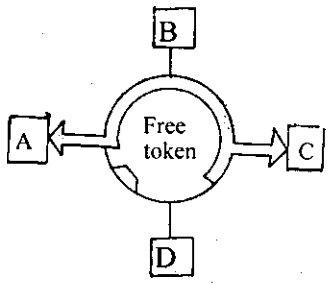
What algorithm is adopted by Token Ring in controlling which node send data first? It uses a token. A token is actually a special sequence of bits circulating around the ring as ordinary frames. Any node which wants to send data out must hold the token. In other words, nodes on a Token Ring network transmits data in turn. When a node has data to send and the token arrives at it, it takes off the token (that is, does not pass it to its downstream neighbour) and sends the data out. The data is then passed to the nodes on the network until reaching the destination node which copies the frame of data into its internal memory. The frame continues to travel after reaching the destination node until it comes back to the sender node which absorbs the frame and releases the token back to the network. Figure 1.6 shows a better visual view of this operation.



Sender A looks for token. When it comes around, A captures it.



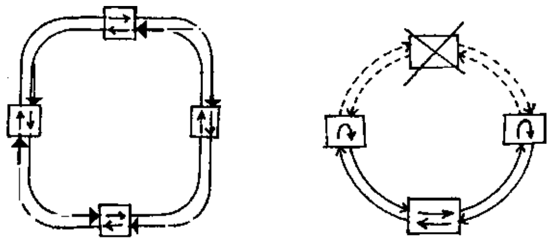
A send frame and it is received by C.



When first bit of the frame loops back, A releases token.
A absorbes its transmitted frame.

Figure 1.6 Data flow in a Token Ring Network

In the case of Fibre Distributed Data Inferface, fibre option cable is used for the implementation. In addition to the normal Token Ring implementation of a single ring topology, FDDI uses a dual ring configuration; that is, each node is connected to two independent rings and data is transmitted in opposite directions in these two rings. Indeed, the second ring is used for fault tolerance, it is not used in normal operations. The whole network fails for any single point failure in a ring topology. The second ring is used to loop back the frames in case of a single node failure or a single break in the link as shown in figure 1.7.



(a)

Figure 1.7 Fault Tolerance in FDDI

The transmission rate for FDDI networks can be up to 100Mbps. The maximum number of nodes that can be attached to the network is about 500 and there should not be a distance longer than 2 kilometres between any two nodes. FDDI uses the 4B15B encoding scheme which is described in the next unit. In the actual implementation of FDDI there are still two critical problems which need to be addressed:

- How long is a node allowed to hold the token for transmission?
- What should be done if the token is missing?

For the first problem, we try to define the time a mode is allowed to hold the token as the Token Holding Time (THT). If the node which holds the token can transmit as much data as it wants - that is, setting the THT to infinity - the node may hold the token for a very long time and this would keep other nodes waiting for a very long time. However, if the THT is too short, the node has a small piece of data to send and may still need to wait for the token several times. Here we try to define another term, Ring Latency (RL), which is the time needed for the token to circulate around the ring once when there is no node which wants to send data. The worst case scenario for short THT is that there is only one node that has data to send.

Consider the following example of such case:

Example

Suppose there is only one node which has data of size 20Kb to send. And assume that a node can send

4Kb data in one THT. Then:

* Number of times the needed to hold the token = 20Kb/4Kb = 5
- Time for sending the data without waiting = 5 x THT
- Waiting time for the token to come = RL
- Total waiting time = 4 x RL

Total time for sending the data = (5 x THT) + (4 x RL)

If the THT is so short that it is comparable with the RL, then the total waiting time is nearly equal to the total sending time. So, obviously, the TUT should be sent to a large value for better utilisation of the ring.

The second problem of the missing token may happen if the node holding it fails or there is a bit error in the token. In these cases, the toekn should be regenerated. But before looking into this problem, we first need to talk about how to discover that a token is missing.

If a node is operating normally, each node on the ring should receive the data transmitted from other nodes or the token in a periodic time. The maximum idle time for a node not receiving anything should be less than 'TUT + RL'. Try to think through why this is so. This is the case where a node holds the token and transmits data using the whole HT The node on its upstream has to wait for the sending node to complete the transmission (the TAT) as well as the time for the frame to circulate a round (a little bit less than the RL), adding up to give `THT+RL! for this reason, each node sets a timer for this time limit, if the node receives data transmission within the time limit, the timer is reset. But if it does not receive anything after the timer expired, it will send out a special data frame called the 'claim frame' to find the token. This then comes to the process of token regeneration, called the 'token claim process'. If the claim frame comes back to the sender node, then the node knows that there is a missing token, and it will insert a valid token back to the ring.

Ethernet is much more popular than Token Ring network implementation mainly due to its low costs and simple configurations. However, it is not as reliable as Token Ring in data transmission. This problem becomes prominent on a network supporting a large number of users because of high collision rate; but Ethernet still provides an efficient network for small LANs. Besides, Ethernet networks can be easily upgraded to the fast Ethernet standard, which supports 100Mb transmission rate.

4.0 Conclusion

In this unit, you have learned that a digital link can only support the transmission of a single electronic signal at one time. This has necessitated the need for media access control - CSMA/CD and Token Ring - for checking, avoiding, and controlling collision of messaes in a link. You need to be aware also about the characteristics and applications of the two techniques.

What you have learned in this unit concerns the various techniques for preventing collision in a link. It has showned the strengths and weaknesses of Token Ring and Ethernet architectures. In the next unit, we shall look at the encoding mechanisms and how errors are detected in a transmission.

a) For a link of speed 10Mbps and propagation delay of 51.2 x 10^{-}as; find what the minimum frame size will be.

b) A company, OPQ solutions, provides Pentium PC for all her staff, with a total of 10 different applications to shared. There may also be heavy sharing of data among the users. What would you recommend for the company: Ethernet or Token Ring? Why.

Exercise 1.1

Discuss the collision issues in a network Marking Scheme.

Exercise 1.2

Discuss the structure of a Token Ring network.

Microsoft corporation *Network Essentials,(2nd* Edn.) Redmond, Washington: Microsoft Press, 1996.

Peterson, L.L. and Davie, B.S. *Computer Networks:* A system Approach,1996.

Online Materials

http://www.rad.com/networks/1997/ethernet/Ethernet.html
http.//www.rad .com/networks/1997/nettut/token-ring.htm I
http://www.howstuffworks.com/etherneta.htm
http://www.quest.com/disclosures/netdisclosure417.html
http://www.peworld.no/pcworld/nettverksskolen/dels.shtm.40

Module 1:Network Basics and Architecture

Unit 7 : Encoding and Error Detection

Page

You will recall that links are physical media transmiming electronics signals which actually only contains the values 0 and Ito deliver computer data. In this unit, we will look into how data can be encoded into signals transmitted in a link, and how transmission errors can be detected and corrected to ensure reliable delivery of data. Let us look at what you will learn in this unit, as stated in the unit objectives below.

[illegible]

By the end of this unit, you should be able to:

- understand the rationale for encoding and error detection in data transmission.
- illustrate the various encoding schemes available.
- understand the various error detection schemes.

The first step in turning nodes and links into usable building blocks is to understand how to connect them in such a way that bits can be transmitted from one node to the other. You will recall, as we studied in the last unit, that signals propagates over physical link. The task, therefore, is to encode the binary data that the source node wants to send into the signals that the links are able to carry, and then to decode the signals back into the corresponding binary data at the receiving node. We will consider this problem in the context of a digital link, in which case, we are are most likely working with two discrete signals. We refer to this generically as the high signal and the low signal. Although in practise these signals would be two different voltages on a copper-pased link and two different power levels on an optical link. As we discussed in the earlier units, most of these functions are performed by a network adaptor - a piece of hardware that connects a node to a link. The network adaptor contains a signalling component that actually encodes bits into signals at the sending node. Therefore, signals travel over a link between network adaptors (see Figure 1.1).

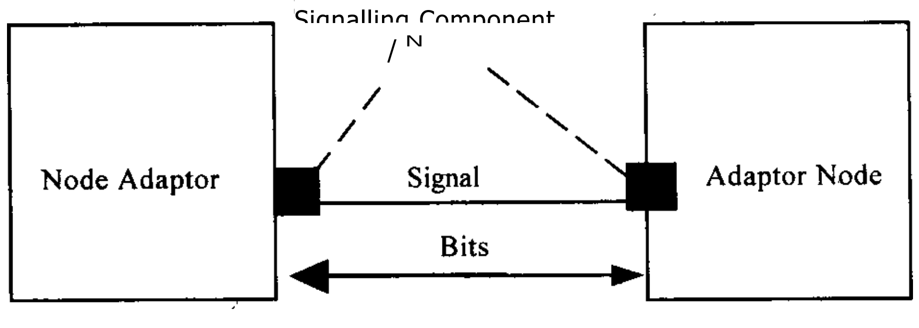


Figure 1.1 Signals travel between signalling components, hits flow between adaptors.

3.1 Shannon's Law and Modem

There has been enormous body of work done in the related areas of signals processing and information theory, studying everything from how signals degrade over distance to how much data a given signal can effectively carry. The most notable piece of work in this area is a formula known as Shannon's theorem. Simply stated, Shannon's theorem gives an upper bound to the capacity of a link, in terms of

bits per second (bps) as a function of the signal-to-noise ratio of the link, measured in decibels (d13). Shannon's law can be used to determine the data rate at which a modem (a device that transmits digital data over analog links) can be expected to transmit data over a noise-grade phone line without suffering from to high an error rate. We assume that a voice-grade phone connection supports a frequency range of between 300Hz to 3300Hz.
Shannon's law is typically given by the following formula:

$$C = B\log, (1 + S/N)$$

Where C is the achievable channel capacity, B is the bandwidth of the line (3300 Hz-300Hz =3000Hz), S is the average signal power, and N is the average noise power. The signal-to-noise ratio (SIN) is usually expressed in decibels (d [3), related as follows:

$$d1{:}3 = 10 \times \log_{ia} (S/N)$$

Assuming a typical decibel ration of 30 dI3 , this means that S/N = 1000.
Thus, we have

$$C = 3000 \times \log_2(1001)$$

which equals approximately 30 kbps, roughly the limit of today's 28.8 kbps modems. Modems will only be able to achieve higher rates if the quality of signals-to-noise ratio of the phone network improves, or by using compression.

3.2 Encoding Schemes

There are many encoding schemes. We are going to consider four of the common techniques used for encoding. They are discussed in the following sub-sections.

3.2.1 Non-Return-to-Zero (NRZ)

In this scheme, the obvious thing to do is to map the data value **I** onto the high signal and data value 0 onto the low signal. This is exactly the mapping used by this scheme called *cryptically enough, non-return-to-zero (NRZ).* For example, signal that corresponds to the transmission of a particular sequence of bits.

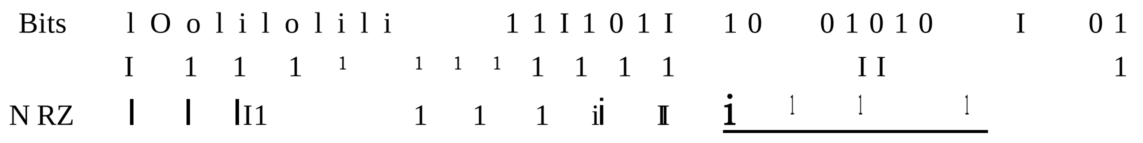


Figure 1.2 NRZ encoding of a bit stream

The problem with NRZ is that a sequence of several consecutive Is means that the signal stays high on the link for an extended period of five, and similarly, several consecutive Os means that **the** signal stays low for a long time. There are several problems caused by long stream of Is and Os. First, a prolong low signal may really correspond to the absence of a signal. This, the receiver cannot distinguish between a long string of Os and a dead link. Second, a sustained high signal potentially confuses the receiver because the receiver uses the average signal level, called the *baseline* to distinguish between the high and the low signal. Too many consecutive **I**s causes this average to change, a situation known as *baseline wander.* Third, frequent transmitions from high to low and vice versa are necessary to enable clock recovery, as described below.

Intuitively, the clock recovery problem is that both the encoding and the decoding processes are given by a clock — every clock cycle the sender transmits a bit and the receiver recovers the bit. The sender's and the receiver's clocks have to be precisely sychronised in order for the receiver to recover the same bits the sender transmits. If the reciver's clock is even the slightest bit faster or slower than the sender's clock, then it does not correctly decode the signal. You could imagine sending the clock over a separate wire to the receiver, but this is typically avoided because it makes the cost of cabling twice as high. So instead, the receiver derives the clock from the received signal —the clock recovery process. Whenever the signal changes, such as on a transition from I to 0 or from 0 to I, then the receiver knows it is at a clock cycle boundary, and it can resynchronise itself However, a long period of time without such a transition leads to having lots of transitions in the signal, no matter what data is being sent.

3.2.2 Non Return to Zero Inverted (NRZI)

One approach that addresses this problem of infrequent transition, called the *non-return-to-zero-invented (NRZ1),* has the sender make a transition from the current signal to encode a I and stay at the current signal to encode a O. This solves the problem of consecutive Is but obviously does nothing for consecutive0s. The NRZI is as illustrated - figure 1.3.

3.2.3 Manchestor

An alternative, called *Manchester encoding,* does a more explicit job of merging the clock with the signal by transmitting the exclusive — OR of the NRZ — encoding data and the clock. Just think of the local clock as an internal signal that alternates from low to high; a low/high pair is considered one clock cycle. The Manchester encoding is also illustrated in figure 1.3. Observe that the Manchester encoding results in 0 being encoded as a low-to-high transmition. Because both Os and Is results in a transition to the signals, the clock can be effectively recovered at the receiver.

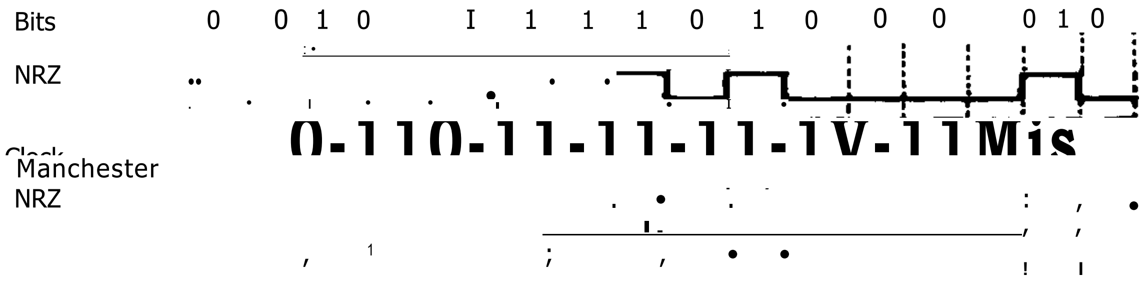


Figure 1.3 Different encoding strategies

The problem with the Manchester encoding scheme is that, it doubles the rate at which signals transitions are made on the link, which means that the receiver has half the time to detect each pulse of signal. The rate at which the signal changes is called the *link's band rate.* In the case of the Manchester encoding, the bits rate is half the band rate, so the encoding is considered only 50% efficient. You need to keep in mind that if the receiver has been able to keep up with faster band rate required by the Manchester encoding in figure 1.3, then both NRZ and NRZI could have been able to transmit twice as many bits in the same time period.

3.2.4 4B/5B

A final encoding that we will consider is called 4B/5B. It attempts to address the inaefficiency of the Manchester encoding without suffering from the problem of having extended duraions of high or low signals. The idea of 4B/5B is to insert extra bits into the bit stream so as to break up long sequences of Os and Is. Specifically, every 4 bits of actual data are encoded in a 5-bit code that it then transmitted to the reciver; hence the name 4B/5B. The five-bit codes are selected in such a way that each one has no more than one leading 0 and no more than two trailing Os. thus, when sent back to break no pair of 5 - bit codes results in more than three consecutive Os being transmitted. The resulting 5-bit codes, are then transmitted using the NRZI encoding, which explains why the code is only concerned about consecutive Os-NRZI already solves the rpoblem of consecutive **l**s. You should note that the 4B/5B encoding results in 80% efficiency.

In Table 1.1, you will see the 5-bit codes that corresponds to each of the 16 possible 4-bit data symbols. Notice that since 5 bits are enough to code 32 different codes, and we are using only 16 of these for data, there are 16 codes left over that we can use for other purposes.

Table 1.1 4B/5B Encoding

4-Bit Data Symbol	**S-Bit code**
0000	11110
0001	01001
0010	10100
0011	10101
0100	01010
0101	01011
0110	01110
1000	01111
1001	10010
1010	10011
1011	10110
1100	10111
1101	11011
1110	11100
1111	11101

Of these, code 11111 is used when the line is idle, code 00000 corresponds to when the line is dead, and **00100 is** interpreted to mean halt. Of the remaining 13 codes, seven of them are not valid because they **violate the one-leading!** two-trailing -zero rule, and the other six represent various control symbols.

3.3 Error Detection

There must be errors on transmitting the data across the links. Bit errors are sometimes introduced **into frames.** This happens, for example, because of electrical interference or thermal noise. Although **errors are rare,** especially on optical links, some mechanism is needed to detect these errors so that **corrective** actions can be taken. Otherwise, the end used is left to wonder why the C program that **successfully** complied just a moment ago suddenly has a symbol error in it, when all that happened in **the interim is that** it was copied across a network file system. The error type actually is simple, in this **case: a 0 is** wrongly recognised as a **1** or vice - versa. There is a long history of techniques for dealing **with** bit errors in computer systems, dating back to Hamming and Reed/Solomon codes that were **developed** for use when storing data on a magnetic disks and in early core memories.

There are several techniques for error detection and control, we shall consider three of them.

3.3.1 Cycle Redundancy Check (CRC)

The basic idea behind any error detection scheme is to add redundand information to a frame that can **be used to** determine if errors have been introduced. In the extreme, one could imagine transmitting **two complete** copies of the data. If the two copies are identical at the receiver, then it is probably the case that both are correct. If they differ, then an error was likely introduced into one (or both) of them, **and they** must be discarded. Fortunately, we can do a lot better than sending *n* redundancy bits for an *n* -bit message. In general, we need to send *only k* redundancy bits for a n-bit message, where *k<<n*. On an Ethernet, for example, a frame carrying up to 12,000 bits (1500 bytes) of data requires only 32-bit CRC code, or as it is commonly expressed, uses CRC-32. Such a code will catch the overwhelming majority of errors.

You should note that you don't reliably detect errors in a 12,000 bit message with only 32 redundant bits unless you have some fairly powerful mathematics behind you. In this case, the theoretical fundations of the CRC is noted in finite fields. To start, think of an n-bit message as being represented by an n-1 degree polynomial, where the value of each bit (0 or 1) is the coefficient for each term in the polynomial. For example, a message containing the bits 10011010 corresponds to the polynomial M(x)= *x'* + x^4 $+x^3$+x' We also let *K* be the degree of some divisor polynomial C(x). For our example, suppose C(x) = x^3+x^1+1. In this case, *k*= 3. The answer to the question "where did C(x) comes from ?" is "You look if up in a book" Again, we will discuss this more below.

What we are going to do is transmit a polynomial that is exactly divisible by C(x); the transmitted polynomial will be P(x). Then, if some errors occurs, it will be as if an error then. E *(x)* has been added to P(x). When the recipient of the message divides P(x) + E(x) by C(x), the result will be 0 in only two cases. E(x) is 0 (i.e; there is no error, or E *(x)* is exactly divisible by C(x). A clever choice of C(x) can make the second case extremely rare, so that one can safely conclude that a 0 result means that the message is error free, and a non-zero result means that an error occured.

We first take M(x) and multiply it by Al$^($, to obtain V^0±X'+X^6+X^4 or 10011010000 in our example. Then we divide this by C(x) which corresponds to 1101 in this case. Figure 1.4 shows the polynomial long division operation. This calculation is just like integer long division with two differences: C(x) divides any polynomial B(x) that is of the same degree as C(x), and the remainder is the exclusive - OR (XOR) of B(x) and C(x).

```
                          1 I 1 1 1 0 0 1
Generator —11. 1 1 0 1  | I 0 0 1 1 0 1 0 0 0 0-0 Message
                        | 1101
                        |  1 001
                        |   1101
                        |  1 000
                        |   1  1  01
                        |      1 1 0 0
                        |      1 1 0 1
                        |             1100
                        |             1101
                        |                 1000
                        |                 1101
                        |1 0 1                    A—R e m i n d e r
```

Figure 1.4 CRC Calculation using polynomial long division

Thus in the first step of our example, we see that C(x) which is 1101 in this case, divides into the polynomial 1001 since they are of the same degree, and leaves a remainder of 100 (1101 XOR 1001). The next step is to bring down digital from the message polynomial until we get another polynomial with the same degree as C(x), calculate the remainder again, and continue until the calculation is complete. You can see from Figure 1.4 that the remainder of the example calculation is 101. So we know that 10011010000 minus 101 would be exactly divisible by C(x), and this is what we send. The minus operation in polynomial arithmetic is the logical XOR operation, so we actually send 10011010101. Note that this turns out to be just the original messagewith the remainder for the CRC calculation appended to it. The recipient deivides the received polynomial P(x) + E(x) by C(x) and, if the result is 0, concludes that there were no errors. If the result is non-zero, it may be necessary to discard the errored message; with some codes, it may be possible to correct a small error (e.g if the error affected only one bit). A code that enables error correction is called *error - connecting code* or ECC.

Now we will consider the question of where the polynomial C(x) comes from. Intuitively, the idea is to select the polynomial so that it does not divide evenly into a message that has errors introduced into it, or said another way, we want to ensure that C(x) does not divide evenly into polynomial E(x) that represents the error. For example, an error in the single bit *i* can be expressed as E(x) = *xi.* If we select C(x) such that the first and the last term are non-zero, then we already have two terms that cannot divide evenly into the one term E(x). Such a C(x) can, therefore, detect all single-bit errors.

Six versions of C(x) are widely used in link-level protocols. They are shown in Table 1.2. For example, the Ethernet and FDDI networks use CRC-32, while HDCC uses CRC-CCITT. ATM uses CRC-8 and CRC-10.

Table 1.2 Common CRC polynomials

CRC	
CRC-8	$x^8+x^2+x^1+1$
CRC-10	$x^{10}+x^9+x^5+x^4+x^1+1$
CRC-12	$x^{12}+x^{11}+x^3+x^2+1$
CRC-16	$x^{16}+x^{15}+x^2+1$
CRC-CCITT	$x^{16}+x^{12}+x^5+1$
C RC-32	X^{3}2±x26+x23+x22±x16+x12+x I 1±x10±x8+ x7±x6+x4+x2.4.x+

Finally, we note that the CRC algorithm, while seemingly complex, is easily implemented in hardware using k-bit shift register an XOR gates.

3.3.2 Two-Dimensional Parity

Two-dimensional parity is exactly what the name suggests. Whereas simple parity involves adding an extra bit to a 7-bit code to balance the number of Is in the byte-e.g odd parity sets the eighth bit to I if needed to give an odd number of Is in the byte, and even sets the eighth bit to I if needed to give an even number of Is in the byte - two dimensional parity does a similar calculation for each bit position across each of the bytes contained in the frame. This results in an extra parity byte for the entire frame, in addition to a parity bit for each byte. Figure 1.5 shows how two-dimensional, even parity works for an example frame containing 6 bytes, of data. Notice that the third bit of the parity byte is I since there is an odd number of Is in the third bit across the 6 bytes in the frame.

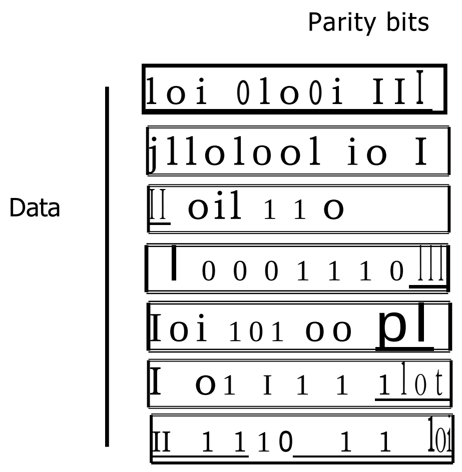


Figure 1.5 Two-dimensional parity Two-dimensional parity catches all 1-,2-, and 3- bit errors, and most 4-bit errors.

3.3.3 Internet Checksum Algorithm

Another approach to error detection is examplified by the Internet Checksum. Although it is not used at link level, it nevertheless provides the same sort of functionality as CRCs and partly.

The idea behind the Internet Checksum is very simple —you add up all the words that are transmitted and that transmit the result of that sum. The result is called the *Checksum*. The receiver performs the same calculation on the received data and compares the result with the received checksum. If any transmitted data, including the checksum itself, is corrupted, then the results will not match, so the receiver knows that an error occured. Unlike CRCs, however, this checksum does not have very strong error detection properties. For example, the checksum will not detect any errors resulting from words being misordered.

You can imagine many different variations on the basic idea of checksum. The exact scheme used by the Internet Protocols works as follows. Consider the data being checksum as a sequence of I 6-bit integers. Add them together using 16-bit ones complement arithmetic and then take the ones complement of the result. That 16-bit number is the checksum.

The reason for using an algotithm like this despite its weaker protection against errors than a CRC is simple: this algorithm is much easier to implement in software. Experience in the ARPANET suggested that a checksum of this form was adequate. One reason it is adequate is that this checksum is the last line of defence in an end-to-end protocol; the majority of errors are picked up by stronger error detection algorithms at the link level.

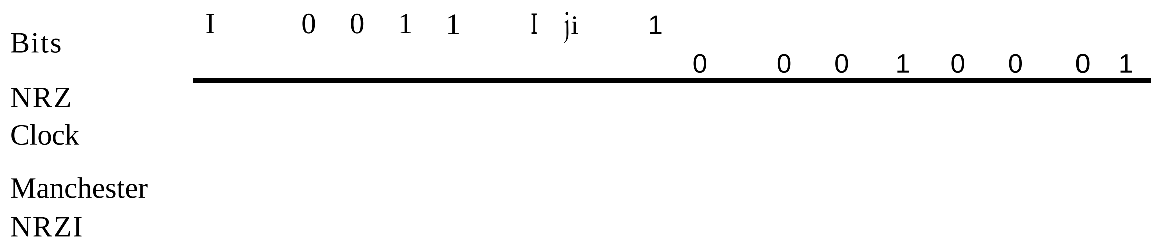
[illegible] Conclusion

In this unit, you have learned about how errors are introduced when messages are transmitted across the network. You also learned about the various encoding schemes, methods of error detection and control were descussed also. The strength and weaknesses of each of the schemes were considered. You need to be aware that error correction is at various levels if you are to have error - free transmission. A lot of important issues that relate to the encoding and error detection have been considered.

5.0 Summary

What you have learned in this unit borders on how errors are introduced into messages transmitted over the network. You also learned how they can be corrected. You should be aware of the different encoding schemes, too. In the next unit, we will consider the protocol issues and layering models.

Show the NRZ, Manchester and NAZI encodings for the bit pattern shown in following: Assume that the NAZI signal starts out low.



Peterson, L.L. and Davie, B.S. *Computer Networks:* A systems Approach,: A Systems Approach, (1996) pp 94-98.

Peterson, Lt. and Davie, B.S. *Computer Networks:* A systems Approach (1996) , : A Systems Approach, (1996) pp 105-110

Module 1: Network Basics and Architecture

Two OSI-7-Layer and Internet 4-Layer Models

Page

Internetwork achtecture and inter-protocol communication are the principal topics of this unit. You will be exposed to the layering internetwork achitectures. This will help you to understand how and why you choose to implement the layering structures and the correspondence functions of each layer are discusssed to identify the differences between the two models i.e, OSI model and Internet architecture. You will also realise that the ISO defined a 7-layer in networking model to help vendors create interoperable network implementations. The Internet architecture adopted a 4-layer model. We shall study both architectures in this unit.

By the end of this unit, you should be able to:

- describe the concept of the OSI model and Internet model, and their differences.
- explain the concept of layering internetwork structure.
- understand the routine for layering in network protocols.

[illegible] Layering and Protocols

A network consists of a collection of interconnected and heterogenous nodes that permits the exchange of units of information of data, among one another. An orderly exchange of data requires that each node conform to some pre-established agreement or rules. These rules specify the formats and relative timing of messages to be exchanged among the workstations. A network protocol establishes these rules, standards, or connections. The idea of layering network architecture was proposed in the 1970s.

The advantage of the layering concepts is to simplify the complexity of the network design with a structure model. Different implementaitons are able to produce compatible networking equipment according to the layering protocols. In a layering structure, as shown in figure 1.1, the role of layer N is to offer service to the upper layer N+1 and to request services from the lower layer N-1. The communication between the source and destination hosts is: layer N on one machine comes on a conversation with layer N on another machine. The rules and convertions used in this conversations are collectively known as the layer N protocol. A protocol provides a communication service which higher level processes use to exchange messages. At each level, peers communicate directly with each other over a link. On the other hand, peer-to-peer communication is indirect Each protocol communicates with its peer by passing messages to a lower-peer corresponding peer in the remote host.

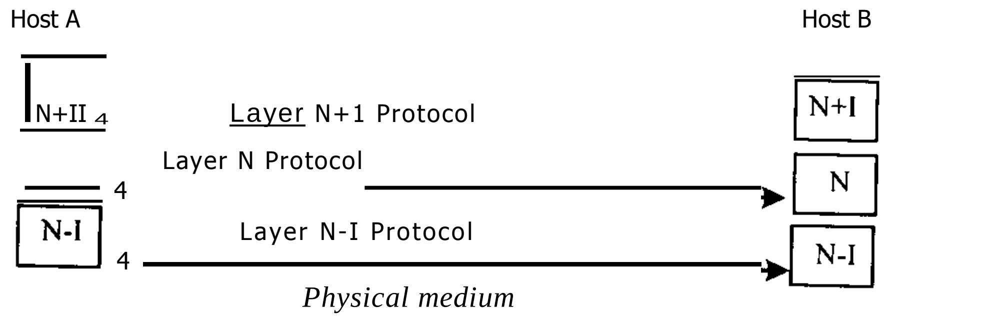


Figure 1.1 Layer network architecture

The set of layers and protocols is called the network architecture. The specifications of the architecture have to be clearly descibed to allow different manufacturers to produce the correspondence layer devices. Hence, in a network with heterogenous systems, the communication channels are still able to be established as all machines are using the defined protocols.

In the layering concepts, layer N is said to be the service user for layer N-1; and layer N-1 is the service provider to layer N. Layer N-1 service, are provided to layer N at a service Access Port (SAP), which is simply a location at which layer N can request layer N-1 services.

3.1 The OSI Reference Model

In 1974, the International Organisation for Standardization (ISO) defined a 7-layer model to clarify various tasks in communicaitons systems. The model is called the ISO Open System Interconnect (OS!) Reference Model. Each layer performs specific functions which allow application software on different system architecture to communicate with each other as if they were operating on the same system. In the layered approach, it is possible to use different network application programs. Two hosts using the OSI model and connected by network devices are shown in Figure 1.2.

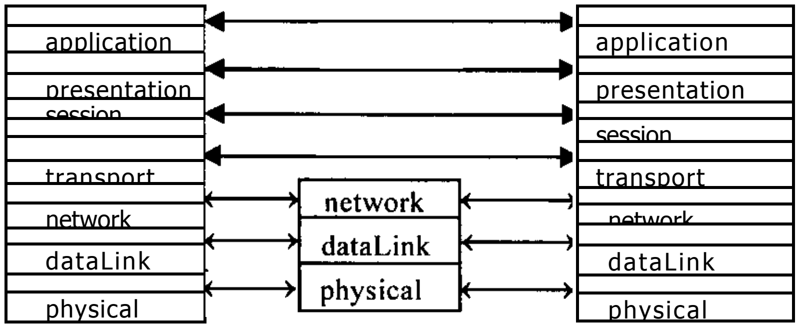


Figure 1.2 Host connecting using the OSI Reference Model

OSI defines the fuctional specification of each layer. However, it does not provide the defunctions of software and hardware to implement the model. The goal of the model is to set a standard for the communication product vendors.

3.2 Functionality of each Layer

The seven layers in sequence from bottom to top are: physical layer, data link layer, network layer, transport layer, session layer, presentation layer, and application layer. Each layer communicates with its peer using services which the layer below provides, as can be seen in figure 1.2. You should note that the transport layer and the layers above it are end-to-end layers and that they do not know anything about the network below. The network devices do not provide services on or above the transpo rt layers are intact through the transmission. Hence, the data on those layers from source hosts are directly handled by their peer layers on the receiving end. The functionality of each layer is described below.

I) Physical Layer

This layer defines electrical signalling on the transmission channel; how bits are converted into electrical current, light pulses or any other physical form; and the ability to detect signalling errors in the network media. A network device functioning at this layer only is called a repeater. The function of a repeater is used to extend the physical distance of internetworking hosts.

2) **Data Link Layer**

This layer defines how the network layer packets are transmitted as bits. It accomplishes this task by having the sender break the input data up into a data frame, (a few hundred bytes), transmits the frames sequentially and process the acknowledgement frame sent back by the receiver. The data link layer creates and recognizes frame boundaries. The frame formatting and the CRS which checks for errors in the whole frame, are incorporated at this layer.

3) Network Layer

The layer defines how information from the transport layer is sent over networks and how different hosts are addressed. Each host in the network must have at least one unique address. Your home address is an analogy of a host address. Your friends can send letters to you through your home address. In a similar fashion, the network address for a host in network close other mechines to address the correspondence host. According to the network address, this layer controls the forwarding of messages between stations. It determines how packets are routed from source to destination. Packets of data are transmitted to other networks through the use of network devices (routers/gateways). Routers, or gateways as they are sometimes called in tutorial terminology, are able to provide this function from the physical layer to the network layer.

4) Transport Layer

The transport layer provides for end-to-end transmission of data. This implies that network devices between the sender and the receiver do not change anything on the data of the transport layer. From figure 1.2, you can see that hose A and host B communicate directly with each other. In other words, the source machine carries on a direct conversation with a similar program on the destination machine. However, the lower three layers of each host work with their peer layers in the network devices. This is why we say that the transport layer is an end-to-end transmission of data. It takes care of the data transfer ensuring the integrity of data, if desired, by the upper layers (i.e, with a guarantee that it will be delivered in the same order that it was sent). This layer ensures the correctness of data transmission, the right sequence (i.e, that it is received in the same order as it was sent) and that it is transmitted in a timely manner. In case a block of data is too large to transmit, the transport layer splits data from the session layer into smaller units.

5) Session Layer

This layer is responsible for establishing and terminating network connections as well as arranging sessions into logical parts. A session allows ordinary data transport, as does the transport layer, ordinary data transport, as does the transport layer, but it also provides same enhanced services which are useful in some applications. This layer is also responsible for range-to-station address translation.

Some protocols do not allow both sides to undertake the same operation at the same time. A locking control in the session layer provides flogs that can be exchanged. Only the host holding the lock may perform the critical operation. This is to prevent inconsistent data updating in the application. Another session service is synchronisation. A typical example of using synchronisation is in transferring a large file over networks. When transfering a two-hour file, communication might be aborted after 'say' one hour of transmission. Retransmission of the file can be achieved by adding check points into the data stream, with the session layer. So that after a crash only the data transfered after the last check point have to be repeated.

6) Presentation Layer

This layer is responsible for data translation (formatting the data) and data encryption and decryption processes. A typical example of a presentation service is encoding data in a standard, agreed way. In order to make it possible for computers (e.g ASC II, GB, BIG5, and so on) the data structures to be exchanged can be defined in an abstract way. In the case of incompatible data formats, information displayed on the host is meaningless data. The application layer handles these abstract data structures and converts them from the representation used inside the computer.

7) Application Layer

This is the interface between the application and the user. It is used for those applications that are specifically written to run over the network. The application layer specifies the protocols to be used between the application programs. The application interface defines interns such as screen layouts, escape sequences for inserting and deleting text, moving the cursors, and so on.

3.3 The Protocol Data Unit in the OSI Layers

To synchronise the communication between host, specific headers for each layer are designed to indicate the start or the end of the communication. In a particular layer, the start header trigger's the layer's software to accept the data, and then the receiving process stops at the end of the stop header.
The concept of the header and the data is relative, depending on the perspective of the layer currently analysing the information unit. For example, an information unit consists of a data link layer header and the data that follows. The data link layer's data, however, can potentially contain headers from upper laeyrs. Finally, not all layers need to oppend headers. For example, the layer two data unit is formatted by the physical layer without having any encapsulation to the unit.
Figure 1.3 illustrates the 051 principles in operation. An application X has a message contained in the application process (AP) data block to send to application Y in a remote host. A header is attached to the data of each stage of the layer in the sending host. The whole encapsulated data unit is transmitted to the destination. In the receiving host, each layer strips off its peer layer header and passes the remainder up to the next layer.

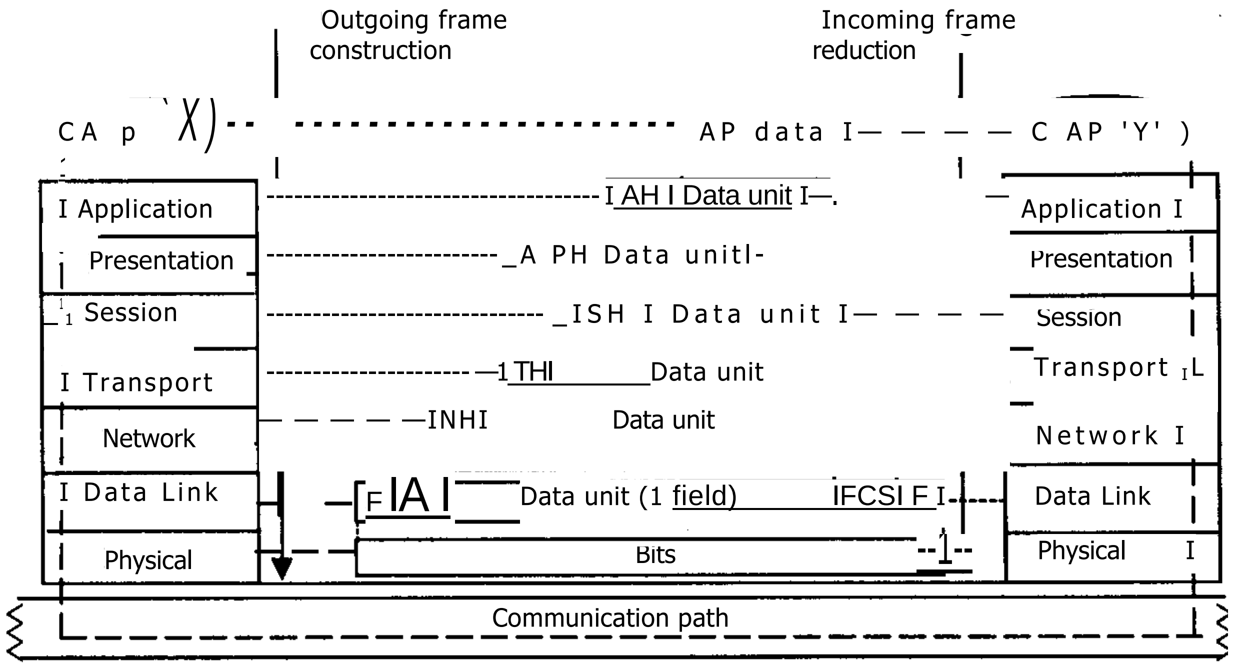


Figure 1.3 A protocol data unit in the OS! modeL

3.4 Important Terms and Concepts for the OSI Reference Model

We shall be looking at the important terminologies and concepts in OS! Reference model. These are as discussed in the following sub-sections.

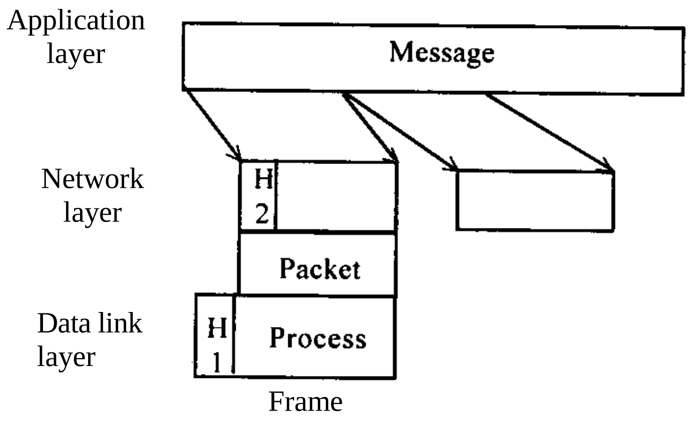
3.4.1 Addressing

You will recall that each host in the internetwork host at least one unique address for identification. There are two types of addresses employed in network technologies. These are link layer addresses and network layer addresses. Link layer addresses (also called physical or hardward addresses) are unique for each network interface. For some devices which have two or more interface, there are multi-houred hosts in the network. Routers and other systems connected to multiple physical networks can have multiple link laeyr addresses. The hardware address works at the data link layer of the OSI Reference Model.

Network layer addresses (also called vertical or logical addresses) exist at the network layer of the OSI Reference Model. This type of address can be modified according to the setting of the application. Unlike link layer addresses, which usually exist within a flat address space, network layer addresses are usually hierarchical. Hierarchical addresses make adress sorting and recall easier by eliminating large blocks of logically similar addresses through a series of comparison operations. For example, we can eliminate all other countries if an address specifies Nigeria as its country. Network layer addresses differ depending on the protocol family being used, but they typically use similar logical divisions to find computer systems on an internetwork.

3.4.2 Frames, Packets and Messages.

In the area of data communication, the terms 'frame' pockee and 'message' are interrelated. The term 'frame' represents an information unit whose source and destinaiton is a link layer entity. The term 'packet' denots an information unit whose source and destinaiton is a network layer entity. Finally, the term 'message' denotes an information unit whose source and destinaiton entity exists above the network layer. The term message is also used to refer to particular lower layer information units with a specific, well - defined purpose. The relationship of frames, packets and messages is shown in Figure 1.4.



Fgure1.4 The relationship between frames, packets, and messages

3.5 The Internet Architecture

The success of the Internet may be attributed to an overwhelming computer Internetwork. The Internet achitecture is also known as the Transmission Control Protocol over Internet Protocol (TCP/IP) architecture. The structure of the Internet Achitecture is represented in figure 1.5. The TCP/IP was developed by ARPANET (Advanced Research Project Agency Network) of the US Department of Defence, and totally funded by them.

At the lowest level are a wide variety of framework access protocols. These protocols are implemented by a combination of hardware and software. To set up a TCP/IP network, one of the first things that you have to do is to choose the kind of network access protocols you will use. For example, one might find Ethernet or FDDI Protocols at this layer. The second layer consists of a single protocol - the Internet Protocol (IP). This is the Protocol that supports the interconnection of multiple networking technologies into a logical internetwork. The third layer contains two main protocol —the Transmission Control Protocol (TCP) and the User Datagram protocol (UDP).

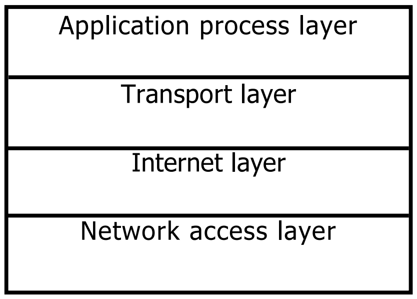


Figure 1.5 *Layers in the internet architecture*

On top of the transport layer is the application process layer in the Internet achitecture. Application Protocols such as File Transfer Protocol (FTP), Telnet, simple Mail Transfer Protocol (SMTP, or electronic Mail) are run in this layer.

Let us now look at the feature of the Internet architecture.

1) You will quickly notice that the layering concept of the Internet Architecture is different from that of the OS! model. As shown in figure 1.5, the number of layers in the Internet architecture is reduced to four instead of the seven used in the OSI model.

2) The Internet Protocol (IP) working at the Internet layer is the most critical factor for the development of the Internet architecture. It defines a common method for exchanging packets among a wide collection of networks. There are many transport Protocols available above the Internet layer. The most commonly used transport Protocols are TCP or UDR Below the Internet layer, the Internet architecture also supports a variety of network technologies such as Ethernet, FDDI and Asynchronous Transfer Mode (ATM).

Despite the fact that the Internet architecture and the OSI Reference Model differs, we can arrange the functionality of the Internet Protocols to fit into the OS1 model, as shown in figure 1.6.

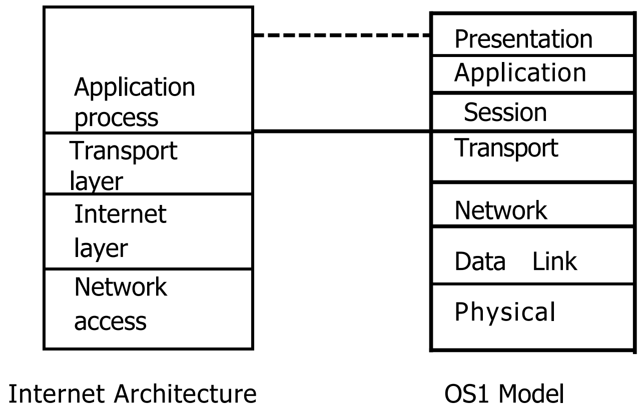


Figure 1.6 Comparison of the interne: architecture and the OSI Model

The application process layer in the Internet model combines the functions of the top three layers in the OS! model. Both models consist of the transport layers in the middle of the structure.The network layer in the OSI model is similar to the internet layer in the Internet model. Finally, the network access layer in the internet model combines the functions of the data link layer and the physical layer in the OS! model.

3.6 Layers and Protocols in the Internet Architecture

The table below lists out the layers in the Internet architecture and the corresponding protocols (see Table 1.1).

Table 1.1 Layers and protocols in the Internet architecture

Layer	Protocol
Application process	DNS*, TFTP*,SNMP*, FTP,SMTP
Transport	TCP, UDP
Internet	IP
Network access	Ethernet, Token Ring, FDDI, ATM

∗ DNS— Domain Name Server
∗ TFTP— Trivial File Transfer Protocol
∗ SNMP— Simple Network Mangement Protocol

3.7 Functionality of Internet Architecture's

Layers

As in the OSI model, each of the four layers in the Internet model performs a different role. This section focuses on the functions of each layer in the Internet.

1) Network Access Layer

The network access layer is the bottom layer of the Internet architecture. This layer provides the ability to transmit the data blocks received from the Internet layer. The data block is called a datagram in the Internet layer. For the transmission of data to media, the network access layer must be aware of the details the network hardware on which it is running. This is the layer that is responsible for translating a logical address to a physical address of the destination.

2) Internet Layer

This layer provides similar features as the network layer of the OS! model. This Layer defines the size and the format of the data to be transmitted. The size of each data block depends on the type of network devices being used. The Internet layer consists of two protocols, the Internet Protocol (IP) and the Internet Control Message Protocol (ICMP). You will recall that IP is central to the TCP/IP protocol suite. All information being sent via TCP/IP must use IP. That is IP performs, the address function. The source address and and the destination address are included in the IP data block. The other protocol that runs on this layer is ICMP, which performs a number of control, error reporting and information functions. ICMP is usually used for diagnostics of the network connectivity.

3) Transport Layer

The specifications of this layer in the Internet architecture is similar to that of the OSI model. The transport layer of the internetwork architecture has two major protocols: Tranmsmission Control Protocol(TCP) and User Datagram Protocol (UDP). TCP is a connection-oriented approach and is reliable, but with relatively large overheads on the connection setup. On the other hand, UDP is a connectionless approach. This protocol can provide a faster connection rate for communication and is generally used when the amount of data being transmitted is small.

4) Application Layer: This is the layer the user should be most familiar with. The user can applications for use over the Internet. Application layer protocols include FTP, Telnet, Routing Information Protocols (RIP), Network File System (NFS), Hypertext Transfer Protocol (HTTP), SMTP, and many others. These protocols are used to provide application level services. For example, FTP defines a number of services for transfering files.

In thii unit, you have learned a number of important issues relating to network protocols. You should have realized also that the ISO defined a 7-layer internetwork model to help vendors create interoperable network implementations. The Internet architecture adopted a 4-layer model, which is a simplified OSI architecture. Due to the rapid growth of the Internet, the studies of both architectures are very vital in network technologies today.

This unit has introduced you to the internetwork structures of the OSI model and the Internet architecture. By understanding the layering concept, you can identify the functions of each layer in the models. The unit that follow shall introduce you to various network devices and where they come in on the protocol stock of both the OSI model and the Internet architecture.

[redacted]

a) What are the advantages of using the layering models?
b) Compare the OSI model and the Internet model

Exercise 1.1

Discuss the addressing issues in networking.

Exercise 1.2

Compare these terms, frames, packets and messages.

SIMS'

Peterson L.L. and Davie, *B.S Computer Networks: A System Approaches,* Morgan Kanfman, *(1996).*

Scweber, W.L. *Data Communications, McGraw* - Hill , (1988)

Stallings, W and Van Slyke, R *(1994)Bussiness Data Communications*, Macmillan Publishing Company,

Online Materials

http//www.rad.com/networks/1996/X25/X25.htm
http://www.rad.com/networks/1997/ethernet/Ethernet.html
http://www.rad.com/networks/I997/nettut/repeaters.html
http://www.rad.corninetworks/1997/nettut/hub.html

Module 1:Network Basics and Architecture

Unit 9 : Network Hardware Components

In this unit, we will explore the different categories of network devices. You will also discover that devices are classified into different types according to their characteristics. Understanding the characteristics of different network devices is useful for the development of internetwork sytems. You will also see that we can classify the type of network device based on the layer of protocol on which the device performs the exchange of data between source and destinaiton. Let us look at what you will learn in this unit as specified in the unit objectives below.

By the end of this unit, you should be able to:

- understand the relationship between network devices and protocols.
- describe the features of different network devices.
- demonstrate the use of network devices in different network designs.

In the last unit, we looked at the characteristics of both the 7-layer OSI model and the 4-layer Internet model, you may now begin to wonder how the packets or data can be transfered through a multitude of network devices. To understand network communication, you need to understand the concept of network devices. In this unit, the purpose is to introduce you to the most commonly used network services. You should be aware that the layer protocol of the upper (four) layers is usually implemented in software approaches. The lower layers (physical to network), however, are related to the specifications of hardware components. Network devices are designed according to those specifications, therefore most of them are only applicable to a particular layer protocol in the communication. Essentially, a variety of netweork devices can be used to achieve the network connection required based on the different network traffic requirements and characteristics of the network configurations.

3.1 Types of Network Devices.

In the last section, you learned that you can classify network devices based on the layer of protocol on which the devices performs the exchange of data between source and destination. Network devices can usually provide higher capabilities if undertaking the data exchange of a higher layer protocol. In the following sub-sections, we shall look at the various types of network devices. Their specific features and characteristics shall be considered too.

3.1.1 Hubs and Repeaters

A hub is the most primitive network devices. In order to simulate a star topology, a hub concentrates the wires to a common point in the network. The hub allocates a port to each host and the hosts are interconnected through the hub. The hub works as a centre of distribution in matters of signals from and to the different parts of the network. Figure 1.1 show the configuration of a star topology usin the hub.

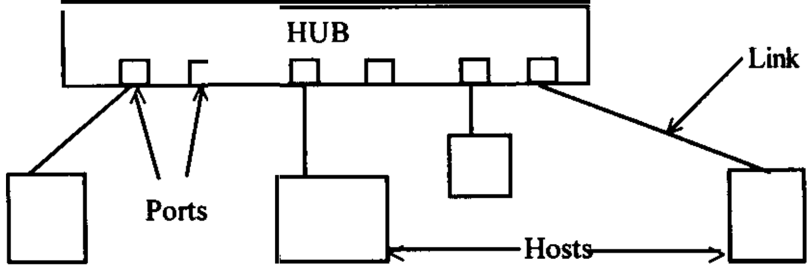


Figure 1.1 Hub devices network configuration

A repeater, as the name implies, copies or repeats signals that it receives. It, however, also amplifies all received signals before retransmission (without changing the frequency). The signal amplification occurs in the physical layer and the analogue signal (including noise and data) will be amplified by the repeater. This happens in order to compensate for the attenuation which the wareform experienced during the journey to the repeater, before sending the signal on its way.

The advantage of using a repeater is to extend the geographical distance of the network coverage. By stategically placing repeaters along a network device, the distance between adjacent computers can be extended to connect a network over a larger area. Note that repeaters connects two segments of the same network to avoid attennation problems. It cannot be used to extend the network distance to infinity. The amplification of noise signals in repeaters will degrade the data quality, the number of repeaters connected to a network segment is limited to a specific number - maximum of four repeaters inthe case of 10Base 2 and 10Base 5 configurations.

3.1.2 Bridges

Bridges technology is able to provide more features than hubs or repeaters. Bridges are used to develop a more complicated network. A bridge is a device which allows two networks that used the same technology to be connected together. It operates at the data link layer which controls data flow, handles transmission errors, provides physical (as opposed to logical) addressing and manages access to the physical medium. Bridges provide these functions by using various link layer protocols that dictate specific flow control, error handling, addressing and media access algorithms. It examines incoming frames, makes forwarding decisions based on information contained in the frames and forward the frames to the destination. It therefore improves the performance of a network system. You should realise that a collision detection system requires a computer to delay transmission after a data collission occurs. The more computers there are connected to the network, the more delays are to be suffered. The operations of bridges are transparent to upper layer protocols (network layer or above) and are not required to examine upper layer information. This means that they can rapidly forward traffic representing any network layer protocol.

Since link layer information often includes a reference to an upper layer Protocol, bridges can usually filter this parameter. This filtering can be useful in dealing with unnecessary broadcasts and multicast packets. Bridges provides a lot of advantages, including the following:

a) Since only a certain percentage of traffic is forwarded, the bridge diminishes the traffic experienced by devices on all connected segments.
b) It acts as a firewall for some potentially damaging network errors.
c) Bridges allow for communication between a larger number of devices than would be supported on any single LAN connected to the bridge.
d) It extend the effective length of LAN, permitting the attachment of distant stations that were not previously connected.

3.1.2.1 Transparent Bridging

Transparent bridges are so named because their presence and operation are transparent to the network hosts. They are able to learn the network's topology by storing the source address of incoming frames from all attached networks. If, for example, a bridge sees a frame arrive on line 1 from host A, the bridge concludes that host A can be reached through the network connected to line I. Through this process, transparent bridges build a table such as the one shown in figure 1.2

The transparent bridging table is uses for traffic forwarding. When a frame is received on one of the brides interface's the bridge looks for the frame's destination address in its internal table. If the table contains an association between the destinaiton address and any of the bridge's parts aside from the one on which the frame was recived, the frame is forwarded out of the indicated port. If no association is found, the frame is flooded to all ports except the inband port. Broadcasts and multicasts are also flooded in this way.

Host Address	Network Number
·^ C W tz• 0 • • •	•• N CM •• en • • •

Figure 1.2 *A transparent bridging table*

You can see that transparent bridge can separate a LAN into different segments. Instead of traversing incoming frames to all other interfaces of the bridges, the reduction of unnecessary traffic forwarding can improve network response tones as seen by the user. The extent to which traffic is reduced and response tones are improved depends on the volume of intersegment traffic relative to the total traffic and the volume of broadcast and multicast traffic.

3.1.3 Switches

The purpose of switching is to be able to reduce the traffic congestion in different network topologies like Ethernet, Token Ring and Fool. To improve network response time, we have to reduce broadcast traffic and increase bandwidth. A switch can achieve this purpose by filtering broadcast traffic; while switching technology is able to increase the bandwdth of network.

In order to reduce the traffic flood between different segments, switches apply technologies like bridges. It stores a table of physical addresses to determine the segment on which a datagram needs to be transmitted, thus reducing traffic. Since the operating speed of switches is much faster than that of bridges, the performance of switching technology is better able to support high volume traffic on networks.

3.1.3.1 Switching in the Ethernet Environment

Traditional Ethernet is a half-duplex technology with a maximum brandwidth of 10Mbps. However, the theoretical 10Mbps brandwidth is not generally equal to the network throughout. This is because data collisions frequently occurs inthe network media. The probability of data collision is increased when the number of connected hosts is increased. Today, throughout on traditional Ethernet LANs suffers even more because users are running network intensive software, such as client-sever applications, which cause hosts to transmit more often and for greater period of time.

An Ethernet LAN switch improves brandwidth by separating collision domains and selectively forwarding traffic to the appropriate segments.

Figure 1.3 shows the topology of a typical Ethernet network in which a LAN switch has been installed.

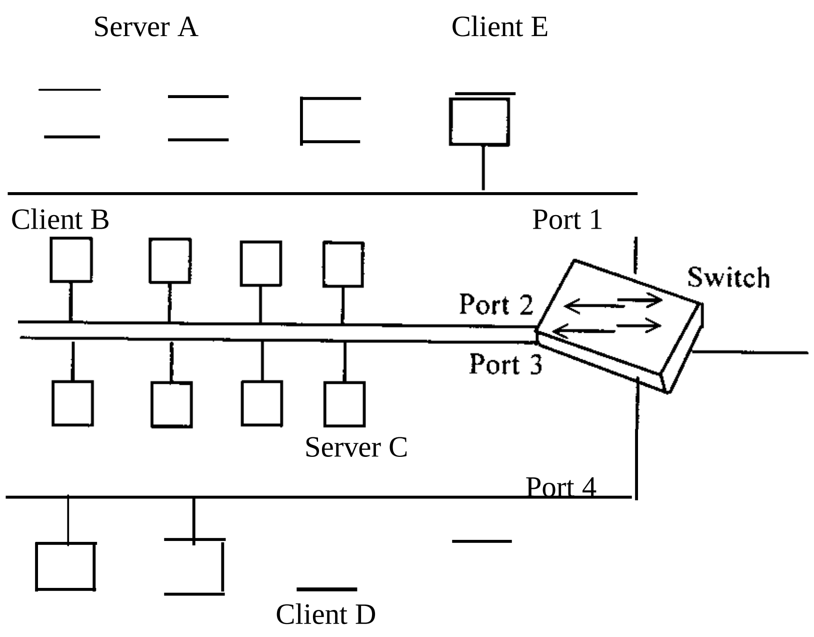


Figure 1.3 Ethernet Switching

From figure 1.3, each Ethernet segment is connected to a port on the LAN switch. If server A on port 1 wants to communicate with client B on port 2, the LAN switch sends Ethernet frames from port Ito port 2, thus sparing port 3 and port 4 from frames destined for client B. If the other pair, say, server C and client D, have to establish a connection at the same time, does server A send data to client B? The answer is yes. The LAN switch can forward frames from port 3 to port 4 at the same time if forwarding frames from port 1 to port 2. Note that the LAN switch need not forward any frame if server A needs to send data to client E, which also resides on port I. The advantage of spreading users over different collision areas is significant. The overall network throughout can be increased as data collision is reduced. Many LAN switch installations assign just one user port to a host, which gives that user an effective bandwidth of 10Mbps.

3.1.4 Routers and Gateways

The router is sometimes called a gateway. It is a device which routes data between network of different technologies such as Ethernet, Novell or IBM Token Ring. The special features of routers are:

- It provides support for multi-protocols.
- It provides multi-interfaces for different networks.
- It operates at the Internet Protocol layer.

Routers, usually consists of multiple interfaces to transfer information across and internetwork from source to destination. However, the Internet is not a network with homogenous technologies. It is necessary to build source network devices which are able to interchange data between different technologies. Routers are designed for this purpose. A router has an address on the network; a bridge does not. Network frequently use a router as an intermediate destination. There are thousands of routers linking the world through the Internet.

A router examines packets of data if they are addressed to it. After checking the packet's destination protocol address, the router detemines that it either knows or does not know how to forward the packet, it typically drops the packet if it does not know the next hop. If it does, it changes the destination physical address to that of the next hop and transmits the packet. If the next hop is not the final destination, a similar process will continue in the following hop until it reaches the final destination.

3.2 Bridges versus Routers

The major differences between bridges and routers are:

a) Bridging occurs at the data link layer of the router OSI Reference Model, while routing occurs at the network layer.

b) All interfaces in routers have corresponding addresses. It is possible to connect to routers through their interface addresses. Bridges does not have such facility.

c) A bridge has to examine all data in order to determine its destination. If a source and destination are in the same network segment, but on different interfaces of a bridge, data from the source has to pass through the bridge in order to reach the destination. This, obviously, could create bottlenecks if the capability of the bridge is not sufficient. However, the capabilities of routers are more powerful than those of bridges. Moreover, routers work at a higher layer. Instead of checking the physical address at the data link layer, as bridges do, routers examines the network address of each incoming data block in order to detemine the corresponding path. Since routers are able to provide network layer services, data from different network segments can be exchanged via routers. Thus, routing and briding accomplish their tasks in different ways. This is because the capabilities of routers are more powerful than those of bridges.

3.3 Routing Components

Routing are intermediate devices that perform the routing operaion. Routing involves two basic operations.

- The detemination of optional routing paths.
- The transport of information groups (typically called packets) through an internetwork.

The cost of the routing operation is weighted by a number of factors - for example, the path length that is used by routing algorithm to determine the optional path to a destination. Routing algorithm generate and maintain routing tables to search the optimum path. The routing tables contain route information which varies depending on the routing algorithm used.

A routing table stores the information for the next hop of the network. Destination or next hop associations tells a router that a particular destination can be reached by sending the packet to a particular router representing the 'next hop' on the way to the final destination.When a router receives an incoming packets, it checks the destination address and attempt to associate with thi3 address with a next hop. Figure 1.4 shows an example of a destination/next hop routing table.

To reach network	send to
27	Node A
57	Node **B**
17	Node C
24	Node A
53	Node A
16	Node **B**
36	Node A

Figure 1.4 Destination/Next hop routtn$_s$ table

Besides saving the next hop information, routing tables also contain information about the 'cost' for each path. If multiple paths are available to reach the destination, routers compare metrics to determine the optimal routes for the packets. Metrics differ depending on factors such as interface speeds, reliability and the path length.

In order to perform these routing functions, routers must talk to one another (and maintain their routing tables) through the transmission of a variety of messages. The information stored in the routing table is constantly updated. Routers compute the routing path based on the information stored in routing tables. Path length is the most common routing metric. Some routing protocols all network administrators to assign arbitrary costs to each network link. In this case, path length is the sum of the costs associated with each link traversed. Other routing protocols define the hop count, a metric that specifies the number of passes through intemetworking products that a packet must take on the route from a source to a destination

3.4 Routing Algorithm

You will recall that routers need to deternine the path for each outgoing pocket. In order to choose the optimum, routers compute the path length that is used by routing algorithms to determine the optional path to a destination. Path determination can be obtained according to the routing algorithm operating in the routers. Routing algorithms use a variety of metrics that affect the calculation of optional routes. The following paragraph analyse these routing attributes:

Optimality

As the name suggests, optimality refers to the ability of the routing algorithm to select the 'best' route. The best route depends on the metrics and metric weightings used to make the calculation. Commonly used factors to calculate optimality are the number of hops and the transmission delay between the source and destination; but these are not sufficient in some situations. For example, user A wants to send data to user B and user A can reach user B from either router 1 to router 2 or router 1 to router 3 to router 2. There is an assumption that the reliability of the data circuit between router **1** and router 2 is not stable. In this situation, you can define a special weight on the data circuit between router 1 and router 2. Hence, the corresponding optimality of router 1 to router 2 is reduced, and the other alternative is chosen for the communication.

Simplicity

Routing algorithm **are** also designed to be as simple as possible. As millions of packets are being sent in and out of the routers, it is impractical to design a complicated process for the routing mechanism. Efficiency is particularly important when the software implementing the routing algorithm must run on a computer with limited physical resources.

Rapid Convergence

In the internet world, the network systems are always changing. Network devices could be up or down at any time and at any place, while network topologies are dynamic. To cater for these dynamic changes of networks, the routing information in routers has to be updated frequently. However, the problem of inconsistency of routing tables occurs when the scale of the network is large.Routing algorithms must converge rapidly to alleviate this inconsistency. Convergence is the process of agreement, by all routers, on optimal routes. Routers distribute routing update messages. After getting the updated information, routers have to recalculate the optimal routes and talk to one another over the new setting. Routing algorithm that converge slowly can cause routing loops or network outages.
Slow convergence of the routing tables brings the problem of loops as shown in figure 1.5. A packet arrives at router 1 at time "t1'. The routing information in router 1 has already being updated, so it

knows that the optional route to the destination calls for router 2 to be the next hop. However, the routing information in router 2 has not been changed yet, so it believes that the optional next hop is router I. You can imagine that the packets are bounding back and forth between router I and router 2, which is called the routing loop. The looping is stopped until the change of the routing information in router 2.

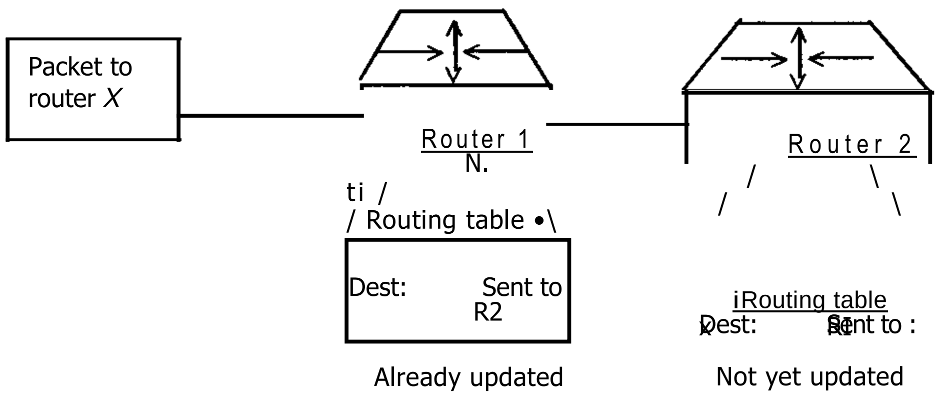


Figure 1.5 show convergence and routing loops

Flexibility

Routing algorithms should quickly and accurately adopt to a variety of network circumstances.
Routing algorithms can be programmed to adopt to changes in network delay and other variables.

In this unit, you have learned a number of important network devices; their features and characteristics. You have also learned about the relationship between network devices and the protocol stack of both the OS! Reference Model and the Internet architecture. In general, an integrated solution of using different devices is widely adopted in internetwork development for most applications.

[redacted]

What you have learned in this unit concerns the features and characteristics of the various network devices. It has served to introduce you to the relationship between network devices and protocol stacks. The unit that follow shall explain in details the roles of IP, TCP, and UDP in internetworking.

What happens if the convergence of routers is not fast enough?

Exercise 1.1

Discuss the importance of switches in a network

Exercise 1.2

Compare a Router with a gateway

Peterson L.L and Davies B.S. *Computer Network: A System Approaches,* Morgan Kaufmann, 1996.

Schweber, W.L. *Data Communications*. Mc Graw Hill, 1998.
Stallings, W and Van Slykw, R . *Business Data Communications*, Macmillan publishing Company,1998.

On line Materials

http://www.rad .com/networks/1997/nettut/repeaters .htm I
http://www.rad .com/networks/1997/nettut/h ub.htm I
http://www.rad.com/networks/1994/bridges/bridges.htmpp intro
http.//www.rad.com/networks/1997/nettut/router.htm I
http.//www. rad .com/networks/1997/nettut/gateway. htm I

Module 1:Network Basics and Architecture

[illegible]

Page

A well-known Internet protocol suite-the Transmission Control Protocol (TCP) and Internet Protocol (IP) suite is presented in this unit. It is a common misunderstanding that the Internet only consists of the TCP and the IP. The reason for this misunderstanding might be due to the success and popularity of TCP and the IP in the Internet community. In fact, the Internet architecture includes a lot of different technologies and in this unit, we shall concentrate on the main core of these: TCP, IP, and UDP. Let us look at what you will learn in this unit, as specified in the Unit Objectives below.

By the end of this unit, you should be able to:

- identify the different roles of IP, TCP and UDP in Internet communication.
- describe the features of the Internet protocol.
- discuss the TCP and UDP mechanism.

In the late 1960s, the Advance Research Projects Agency (ARPA) of the US Department of Defense formed the first packet switching network in collaboration with US universities and other research organisations. All these participants built the ARPANET network. The Transmission Control Protocol and Internet protocol were developed in 1974 for all hosts in the ARPANET.

In 1984, the ARPANET was splited into two parts. One represents the early stages in the growth of the Internet and the other is called MILNET to represent the military network. The predominant layer protocols of the Internet architecture are shown in Figure 1.1 The Internet layer is particularly important because every host in the Internet must have a unique internet address (or IP address). These adresses are 32-bit numbers and written as four decimal numbers (e.g. 192. 168. 2123).

The popularity of the Internet architecture is contributed to by its open standards characteristics. It allows the integration of systems of all sizes, from different system vendors, with different operating systems. You may consider the Internet architecture as a common structure for internetwork communication.

There is no gainsaying the fact that you have seen the letters 'TCP/IP' frequently in the internet world. The acronym TCP/IP, stands for two terms -TCP for Transmission Control Protocol and IP for Internet Protocol. Relating to TCP, there is another important protocol working at the same layer as TCP. It is the User Datagram Protocol for layer 2 in the Internet architecture. The functions of the Internet protocol such as naming, address administration and packet rating are explained in this unit. In order to address a host on a network, we need an IP address to locate the corresponding host. Hence, there are other protocols such as the Routing Information Protocol (RIP), ICMP, ARP, and Reverse Address Resolution Protocol (RARP) to provide routing, addressing and diagnostic services. On top of the IP layer, the TCP and UDP are the major technologies for layer 3 in the Internet architecture. The concept of TCP and UDP is introduced in order to help you understand the services provided via TCP or UDP for different applications. We also examine the packet format in detail to show how TCP and UDP operates.

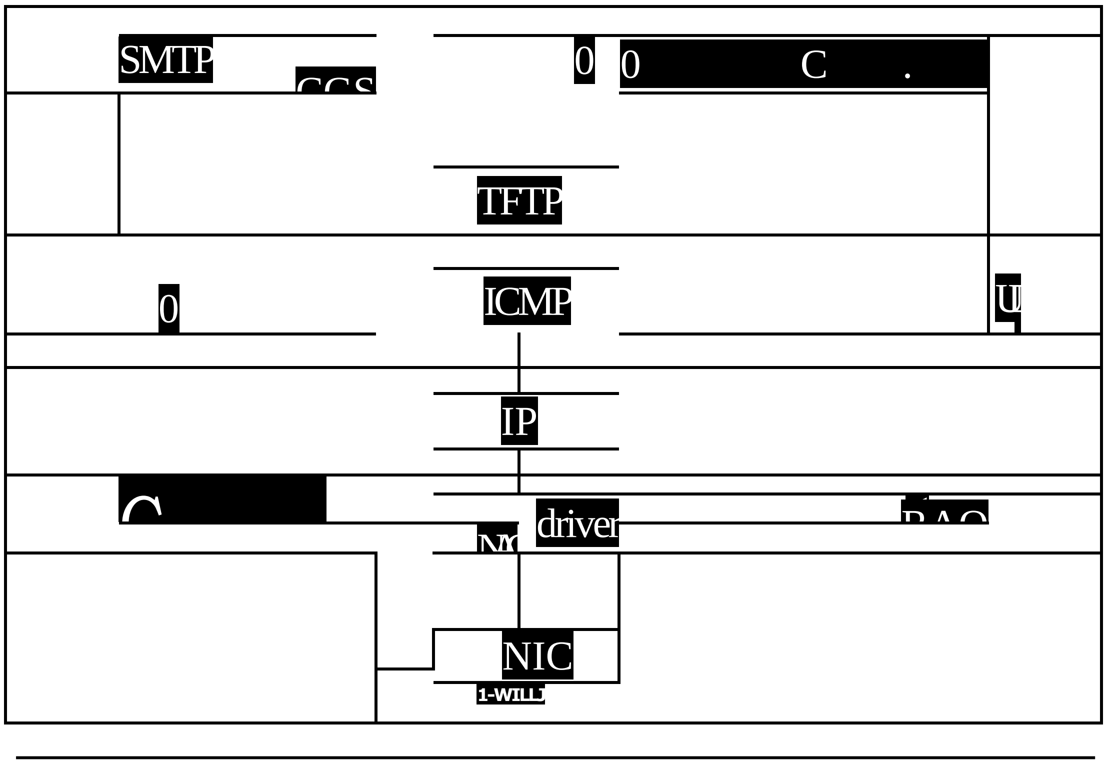


Figure 1.1 the Internet protocol suite and applications

Key: RARPD = Reverse Address Resolution Protocol Deamon
NFS = Network File System
NIC = Network Interface Card
SNMP = Simple Network Management Protocol
TFTP = Trivial File Transfer Protocol

3.1 Functions of IP, TCP, and UDP

The Transmission Control Protocol (TCP) and the Internet Protocol (IP) are the two best known protocols in the Internet protocol suite. The Transmission Control protocol over Internet Protocol (TCP/IP) is a set of protocols developed to allow computers to communicate across a network. It was developed by a community of researchers centred around the ARPANET. It is because of these researchers that TCP and IP are the best KNOWN of the protocols. TCP/IP is used to refer to the whole Internet protocol family. Besides TCP in the transport layer, the User Datagram Protocol (UDP) also provides an important role in transport layer protocols. Both TCP and UDP are responsible for breaking up the message into packets or datagrams, and reassembling them at the other end. TCP is a more intelligent protocol. It is able to resend any data that gets lost and puts packets back in the right order. In a real situation, a single station in a network can perform different tasks such as sending email and retrieving files from remote sites via the File Transfer Protocol (FTP). Hence, one of the functions of the transport layer protocol are to provide multiplexing communication abilities.

The responsibility of the Internet Protocol is to provide services in the network layer. TCP or UDP send data blocks to IP as tells IP the Internet address of the computer at the other end. Afterwards, it is the duty of IP to deliver the data to the receiving ends. You can imagine it as being similar to sending a letter through the post office. What you have to do is to write the destination address and the rest of the task will be carried out by the post office. The post office performs similar task of IP.

These three protocols;TCP, IP, and UDP, are the dominant transmission protocols of Internet services. The Internet protocols can be used to communicate across any set of interconnected networks. They are equally well suited for Local Area Networks(LANs) as they are for Wide Area Networks (WANs).
You may have noticed that the structure of the Internet protocol suite is very interesting. In the application process layer, we have a variety of applications such as Telnet, email, FTP, Ping, and so on, to support different transport layer protocols such as TCP, UDP, and the Internet control Message Protocol (ICMP). Below the IP layer, it supports different network technologies such as Ethernet, Token Ring and Token Bus. However, there is no alternative but IP in the **IP** layer of the Internet protocol means that we will explore this first.

3.2 Internet Protocol

The Internet Protocol is the Internet layer protocol that controls the routes of data across an intemetwork. Besides the internetworking routing function, IP provides fragmentation and reassembly of datagrams, and error reporting. The meaning of fragmentations is to divide a large datagram into several smaller datagrams. In a large network, an originating host may not know all of the size limit that a datagram will come across along its path to destination.

Hence, a datagram which is too large for some intermediate devices(routers) will be fragmented by the intermediate devices. It ig then up to the destination host to gather up incoming fragments and to reassemble the original datagram. The importance of the IP layer can be seen in Figure1.2. You may notice that almost **all** services in the Internet layer of the Internet architecture are working with IP technologies.

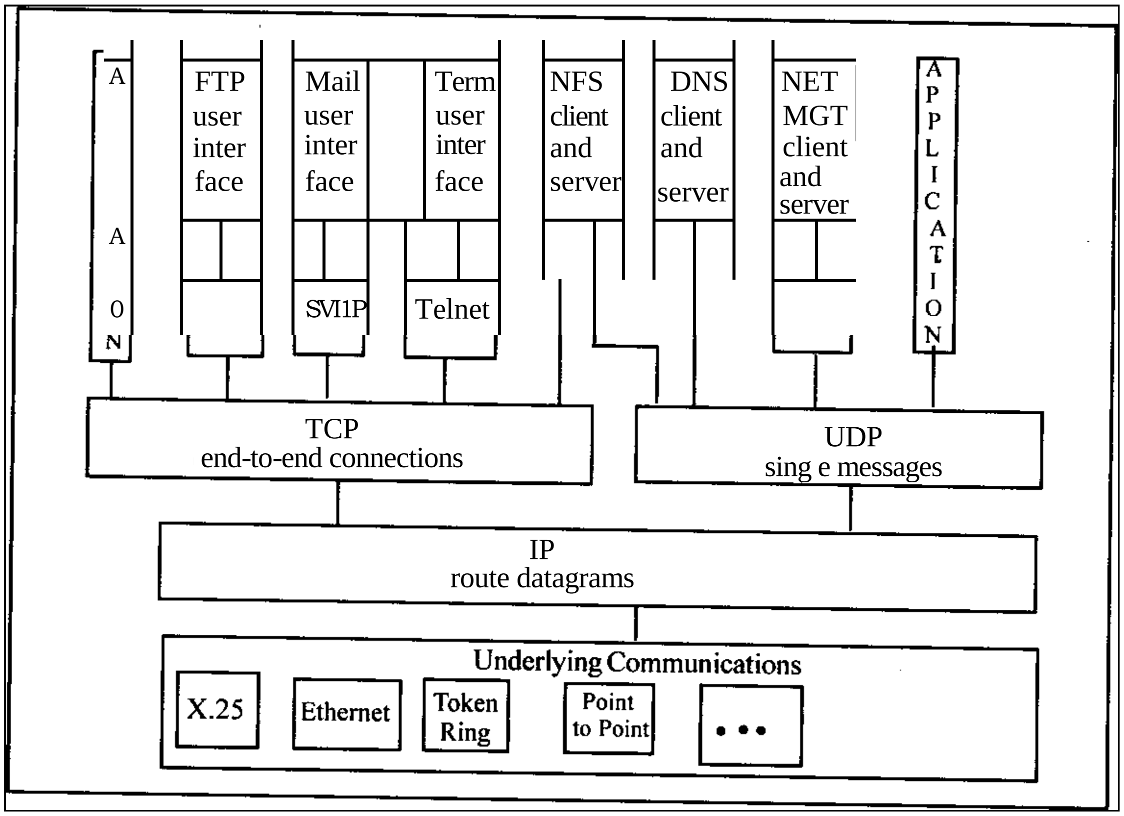


Figure 1.2 Different protocols in the TCP/IP protocol suite.

3.3 The Format of an IP Packet

In this section, we shall take a closer look at format, in terms of the fields, of a typical IP packet.Figure 1.3 shows the structure of an IP packet.

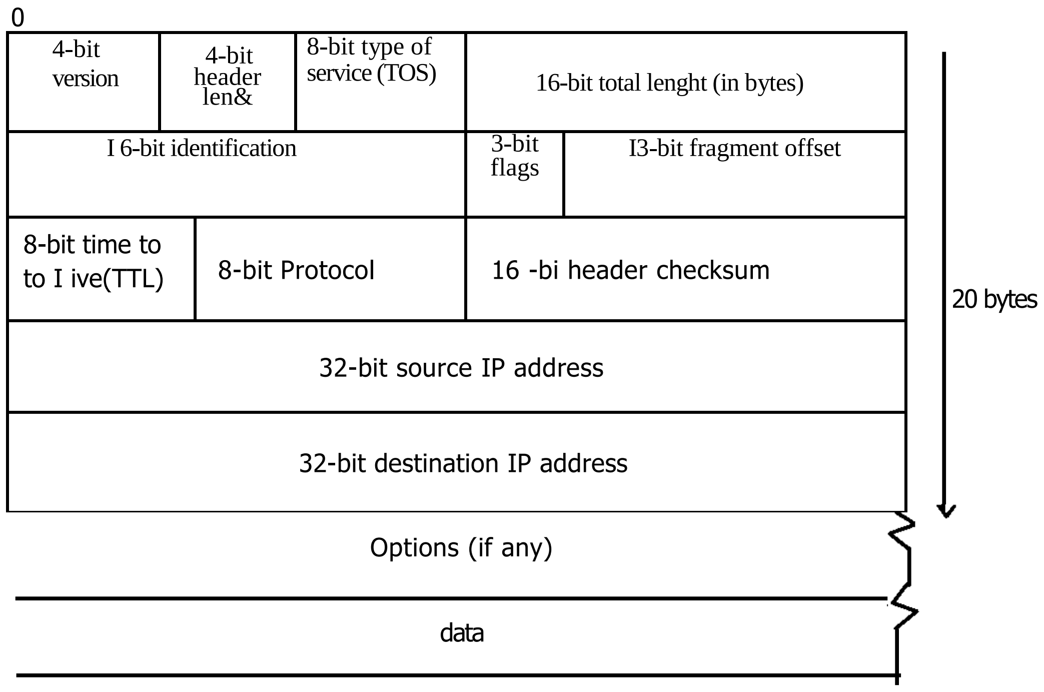


Figure 1.3 The IF packet format

The fields of the IP packet and their functions are as follows:

a) Version indicates the version of IP currently used. The current version 4 or IPv4 in short form

b) IP Header Length(HL)_ indicates the datagram header lenght in 32-bit words. If there are options set, the header length is 20 octets (bytes)

c) Type of Service (TOS) _ specifies how a particular upper layer protocol would like the current datagram to be handled. The priority of the datagrams can be defined by a 3-bit precedence field. Four TOS bits are also used to minimise delay, and maximise through put and reliability. The TOS for different applications may be different. For example, the interactive log in applications, such as Telnet and Unix remote log in (rlog in), want a minimum delay since these applications are interactively inter-faced by a human. It is sensitive to a user for the time delay of the application. Hence, the bit for minimising delay in TOS is set and the rest of the bits will not be so. To consider a file transfer application, FTP the data transfer of the most important point. Hence, the bit for maximising through out is set in the TOS.

d) Total Length_ lenght of the entire IP packet measured in octets, including data and header. The maximum packet lenght is 64 octets.

e) Fragmentation Field_ identification, flags and fragment offset indicate the packets' sequence in fragmentation. Receiving hosts reassemble the fragmented packets to the sending sequence according to the information in this field.

O **Time to Live (TTL)**_ indicates the maximum number of seconds/hops that a datagram will be allowed to remain in the network. The TTL is a counter that gradually decreases down to zero, at which point the datagram is discarded. This keeps packets from looping endlessly.

g) Protocol— identifies which upper layer protocol receives incoming packets after IP processing is corn pleted.

h) Header checksum—a 16- bit field that helps ensure IP header integrity.

i) Source **IP address_specifies** the sending IP address.

j) Destination I' address —specifies the receiving IF' address.

k) Options— provides additional information on features of the IP header. Options such as dedicated rating, security or time stamp operation can be available.

I) Data— contains upper layer information, including TCP or UDP data

In this unit, you have explored different Internet protocols, including TCP and UDP for the transport layer, and IP for the IP layer. The mechanisms of how these protocols work in the Internet were presented to help you develop a clear concept of TCP/IP and UDP **issues. You also learned other related protocols such as** routing protocols, the Address Resolution Protocol **(ARP) and ICMP.**

What you have learned in this unit borders on the different Internet protocols, TCP and UDP for the transport layer, and IP for the IP layer. The units that follow shall build upon this fundamentals.

6

Describe the format of an IP packet **and discuss the functions of each field.**

Exercise 1.1

Discuss thedevelopment of IP and TCP

Exercise 1.2

Compare TCP with UDP

Peterson, L.L. and Davie, *Computer Network: A System Approaches,* Morgan Kaufmann, 1996.
Schweber, W.L. *Data Communications,* McGrawHill, 1988.

Online Materials

http.//www.home.tu-clausthal.de/ —inof/tcptut/tcptut2.html
http://www.home.tu-clausthal.de/ —inof/tcptut/tcptut2.htm #ip
http://www.oline.vuse.vanderbiltedu/federation/mod 1 fhtm

Module 2: Internetworking, Network Design and Management

Page

In this unit, you will learn about the physical addressing of hosts and how it relates to the IP names and addresses. You will also learn about the various class of IP address and their applications. The issue of subnets, and how it can be used to expand a network will be dealt with too. Let us now look at what you will learn in this unit as specified in the unit objectives below:

By the end of this unit, you should be able to:

- discuss IP naming and address translation mechanism
- understand the relationship between the physical address and the IP address
- discuss the various classes of IP address available

You will recall that the Ethernet address is hardware address- sometimes called a physical address. From its name, it is an address related to the network interface. Each network interface has a built in 48 — bit address to identify itself. This physical address is fixed and you need not configure the setting of the physical address. That mean, theoretically, we can have 2" different physical addresses in the world without duplication. If the total number of hosts on the Internet were larger than the Ethernet address numbering limit (2"), the network would collapse because of the duplication of Ethernet addresses. Although, at present, we are still not facing the problem of running out of physical addresses for network interfaces.

While the physical address works at the networks acess layer, the IP address works at the Internet layer which provide a network access similar to the requirements of the network layer in the OSI model. In most applications, we address the destination host with its Internet layer address scheme which is the IP address in the TCP/IP protocol suite. Note that the transmission of packets in Ethernet is broadcast mode, which implies that wherever a packet is sent from a host, the packet will pass through the whole network of the particular host. This mean that each host has to keep an eye on the network media in order to extract data sent to it. A host absorbs a packet and passes it to the upper layer whenever it detects a packet sent to its address. A network interface recognizses packet sent to it by checking the destination address in the network access layer which is the physical address to determine whether to absorb the packet or not.

There is no problem for a user to send data to the receiving host via its IP address since the mapping between the physical address and the **IP** address will be done by the ARP and RARP. The details of address resolution protocols will be considered later in the unit.

3.1 Addressing and Naming

Note that every host in a network has to have a unique address, an IP address. However, the numeric IP address is not that meaningful to humans. It is, therefore, reasonable to identify hosts with names and that the names should reflect the specific purpose of the machine. The idea of an address scheme in the TCP/IP suite consists of 32 bits and is usually represented with a four numeric number such as:

192.123.44.23. In the following sub-sections, we shall consider in details the concept of internet host naming and addressing.

3.1.1 Naming

The naming scheme in the intemet is based on the concept of domains. Host names within a domain are delegated by the domain administrator. A domain can be divided into different subdomains. Each sub-domain can consists of hundred or thousand hosts in the naming system.

InterNic is the organisation which controls the internet naming scheme. Every domain name has to be registered with interNic. The domain is structured in a hierarchical model. There are several common naming connections:

- corn — commercial organisation
- edu — educational institution
- gov — government
- mil —Military organisation
- net — systems performing network services
- org — non-profit organisation

The structure of host names may consists of two to five labels. For example, the host name: `www.oauffe.edu.ng' consists of the name of the host, 'www', and the domain name, 'oauife.edu.ng'. The domain name is relative. oauife.edu.ng is the subdomain name of 'edu:ng and 'edu.ng is a subdomain of ng', 'lig' is the root of the name structure.

3.1.2 Addresses

The IP address is a 32-bit binary value. Each host in the intemet has a unique IP address to -locate its position. In the case of hosts with multiple network interfaces, we assign different IP addresses to indicate these are multihoured devices. To analyse the IP address of a host, the simplest case is to divide the IP address into two parts. The first part represents the network address of the host and the rest is the host address in the network. A subnet technique can be applied to IP address schemes to partition a single network address into multiple subnetworks.

The format of IP addressing supports five different network classes. The network classes are designed to apply for different scales of networks. Table 1.1 shows pattern of the various classes of IP address

Table 1.1 Different classes of IF address

First 4 bits of IP address	Network class
OXXX	Class A
1 0 XX	Class B
I1OX	Class C
1 1 1 0	Class D
1 1 1 1	Class E

Let us now consider the features of each class of IP address (or network class).

- Class A networks — allocates 7 bits for the network address field and start with a number between 0 and 127. The host address field has 24 bits. They are mainly for use with a few very large networks since they can support very large number of host addresses.

- Class B networks — allocate 14 bits for the network address field and start with a number between 128 and 191. The host address field has 16 bits. It provides a good balance between network address and host address space.

- Class C networks — allocates 21 for the network address field and start with a number between 192 and 223. The host address field has 8 bits. Eight bits implies a maximum number of 254 hosts in a class C network. This class of network is the most popular and it is suitable for small scale network.

- Class D networks — starts with a number between 223 and 239. The class is reserved for multicast applications. Multicast is a restricted form of broad cast Data sent from a host with a class D IP address will be delivered to group of systems which belong to the same multicast group. One if the major advantages of using multicast is to reduce the redundancy of sending save information from an application host to multiple receivers. Consider a video broadcast application with a server and two users (assuming this application does not support the multicast feature), the server has to send data separately to user 1 and user 2. This will greatly affect the performance of the network as the number of users increases. The situation will be totally different if the application supports the multicast feature. When all the users belong to the same multicast group, the server only needs to send the data once for all users. The server is not required to send data one by one. The network loading will be increased for this type of network applications.

- Class E networks — are reserved for experiemental use. Class E addresses start with a number between 240 and 255.

The IP addresses are written in dotted decimal format. For example, am IP address 192. 173. 15. 23 is a class C network. The network address is 192.173.15.0 and the host address 23. The 32- bit IP address is divided into four 8-bit numbers and separated with a dot. Similarly, an IP address 10.122.12.34 is a class A network. The network address is 10.0.0.0, and the host number 122.12.34.

3.2 Network Addressing Guidelines

There are some guidelines to follow when selecting the correct IP address. The following criteria are vital for network addressing and you should always bear in mind these roles:

- The IP address of each host must be unique. Duplicated IP addresses on different host will cause the corresponding hosts to be unable to communicate and even to halt.
- The network address cannot begin with the number 127. The number 127 in class A is reserved for internal loop back functions. The loopback address of a host is used to cheek the correctness of the network configuration for the host. An IP datagram sent to the loopback address will return back to the network system itself If the system is properly configured with the network setting, the loopback process is OK. However, if the datagram is not able to return back to the system, it implies there is something wrong in the network setting or the hardware interface.
- The first octet in a network address cannot be 255. The octet 255 is used for broadcast mode of the network.
- The first octet in a network address cannot be 0. Zero is used to indicate that the address is the local network.
- Host address cannot be 255. This means that all the bits will be set to Ps and the address is regarded as if it in broadcast mode.
- Host address cannot all be O's 3.3

Subnets

You will recall that an organisation with a class A or class B network is able to support a large number of hosts in the same network. However, it does not make sense to attach several thousand hosts in a single network address. Broadcast messages generating from thousands of hosts in the network may deteriorate the network performance to an unacceptable level. Besides the performance issue, the utilisation of the IP address is another concern for class A and class B. Hence, it is reasonable to divide class A, B or even C network addresses into multiple subnet network addresses. The format of a subnet IP address is as shown in figure 1.1.

Network address	Subnet address field	Host address

Figure 1.1 Subnet IP address

The first part designates the network address; the second part designates the subnet address; and final part designates the host address. Subnets provides extra flexibility for network administrators. For example, lets's assume that an organisation has been assigned a network 135.15.0.0 class B by interNIC. This implies the network of the organisation can support at most 65534 hosts. However, the network administrator of the organisation can use a subnet mask to subdivide the 135.15.0.0 into a group of subnetworks. This is done by borrowing bits from the host portion of the address and using them as a subnet field.

Assume the network administrator assigns 8 bits of subnetting; the third octet of a class B IP address provides the subnet number. Now, you can consider 135. 15.1 is the subnet I of the network 135.15.0.0, 135.15.2 is the subnet 2, and so on.

Logically, the network administrator is able to increase the number of network addresses to more than 200 networks for the organisation. It seems to be more flexible for the administrator to design the IP network with different segments.

The number of bits borrowed for the subnet address is variable. Hence, it is possible to determine the number of subnets by choosing different subnet masks.

Subnet mask make use of the format and representation of IP addresses. Subnet masks have IS in all bits except those bits that specify the host field. Let's refer back to our example, the subnet mask is: 255. 255. 255.0 for the class B that specifies 8 bits of subnetting. The subnet mask that specifies 8 bits of subnetting for class B address 135.15.0.0 is 255.255.255.0. Hence, the network address of IP addresses 135.15.1.15 and 135.15.1.35 are both 135.15.1.0. We use the following example to illustrate the variable length of subnet mask.

Assume the subnet mask is

11111111 11111111 1111 0000 00000000; that is, 255.255.240.0:

- IP address: 135. 15.17. 12
- Subnet mask: 255. 255. 240.0

This implies the corresponding address of the station:

- Network address: 135. 15.0.0
- Subnet address: 0.0.16.0
- Host address: 1.12

3.3.1 Subnet Benefits

With the use of the subnet tech ique network administrators can divide a network into multiple subnetworks and connect subnetworks with roofers. Benefits derivable from this arrangement include;

- reducing network congestion by redirecting traffic and reducing broadcasting it; can significantly improve the performance of Ehemet network.
- better utilisation of the **IP address**
- it's easier **to control the network segment by dividing hosts into different subnetworks.**

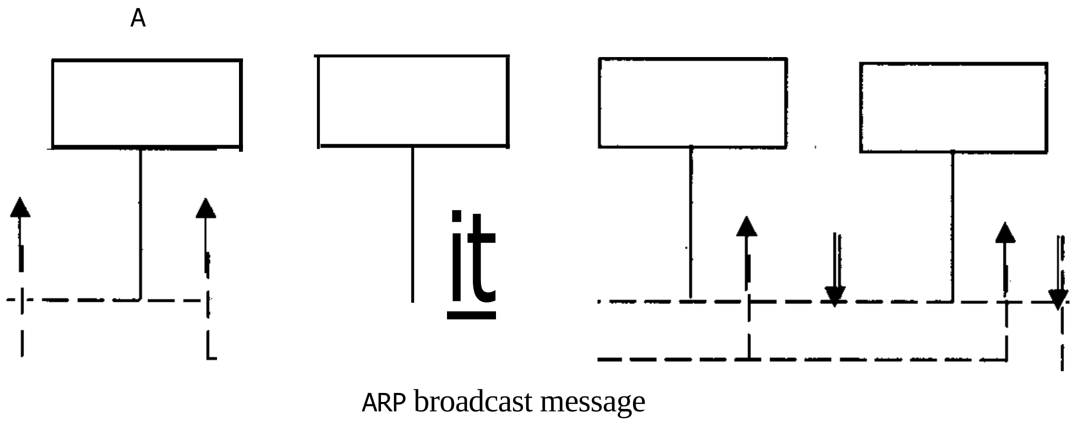
3.4 Physical Network Address Resolution

In some media (e.g., IEEE802 Local Area Networks), physical addmses and IP addresres are found dynamically through the use of the Address Resolution Protocol and the Reverse Address Resolution Protocol. ARP uses broadcast messages to determine the physical address of a particular IP address. The system on the local network applies ARP to an automatically look up the physical addresses.

Consider the diagram in Figure 1.2, a uses at the host A with IP address 192.168.75.5 sent data to the receiving host B with IP address 192.1683525: However, there were other hosts affached to the network. In order to identify the physical address of the host B, host A sent a broadcast message to discover the

physical address of the host B.

When host B received the ARP broadcast message and know that host A was looking for its physical address, host B sent its physical address information to host A. After receiving the physical address of the host B, host A updated its ARP cache. The ARP cache maintains the recent mapping from IP addresses to physical address.



Reply from host B to host A with its MAC address

Figure 1.2 An ARP broadcasting message on a network

In the ARP cache of host A, the physical address of host B was recorded for further use. To examine the ARP cache of host A, there is the ARP command, which sets an option to show all entries in the cache:

Host 8(192.168.75.25) at 8:0:21:4:f 2:bb

The host names, IP addresses and the corresponding physical addresses were stored in the cache, similarly, as host A communicated with host C, the ARP cache stored another entry for host C.
The reverse process of ARP, Reverse Address Resolution protocol, is known as Reverse ARP or RARP. A station uses RARP to broadcast messages to determine the Internet address associated with the hardware address of itself.

3.5 Address Administration

As discussed in the earlier section, a host without an IP address can determine its IP address by using the RARP request and reply. However a major limitation of RARP is that RARP request cannot be forwarded by routers. Hence, network administrators are not able to design RARP clients in different network segments with a single RARP server. In this section, we wilt discuss two alternatives- the Bootstrap Protocol and the Dynamic Host Configuration Protocol- which are used to overcome the limitations of RARP.

3.5.1 The Bootstrap protocol

The Bootstrap protocol works as a client-server model. The client- server model mean that a host will offer services to the other hosts. The service provider is called the *server* and the other are the client. There are two well- known ports for BOOTP: 67 for server and 68 for the BOOTP client. A BOOTP clients broadcast a request message with port number 68 to the network. All the BOOPT servers on the network are awakened to look at each broadcast request. When a BOOPT server replies to the request from the client, the reply message provides network information for the client to configure its parameter. A useful way to get to understand the BOOTP process is by analysing the BOOTP packet format shown in figure 1.3.

IP header I	UDP header	BOOTP request /reply
20 bytes	8 bytes	300 bytes

3.5.2. Dynamic Host Configuration Protocol

Similarly to the BOOTP model, DHCP also works as a client-server are intecture. The designated server assigns network addresses and lease configuration parameters to dynamically configured clients. DHCP clients can obtain the initialisation parameters from DHCP servers.
DHCP provides automatic, dynamic and manual allocation schemes for IP addressing:

- with automatic allocation, DACP assigns a permanent IP address to a client.
- with dynamic allocation, DACP assigns an IP address to a client for a limited period of time.
- with manual allocation, DACP assigns on IF address to the client through the Network Administrator.

Note that DACP provides a framework for passing configuration information to hosts on a TCP/IP, it also provides configuration parameters to Internet hosts. Table 1.1 shows the description of fields in a DHCP message.

Table 1.1 Description of fields in a DHCP message

Field	Octet(s)	**Description**
OP	1	Message op code/message type: 1 = BOOTREQUEST, 2= BOOTRE PLY
I Type	1	Hardware Address length. 6 bytes in 10Mb Ethernet
Hops	I	Client sets to zero, but can be used by a relay agents
Xid	4	Transaction ID, a random number chosen by the client; this number is used for each request and response between client and server
Sec	2	Number of seconds elapsed since client began address acquisition or renewal process
(Unused)	2	Reserved for future use
Giaddr	4	If the client already known its IF address, this field is filled bz the client
Yiaddr	4	Server provides the client address
Siodder	4	IP address of next server to use in bootstrap
Giaddr	4	Relay agent (i.e.; pooxy server) IF address, used in booting via a relay agent
Chaddr	16	Client hardware MAC (Media Access Control) address
Sname	64	The server host name is a null terminated string (optional)
File	128	Boot file name
Specinf	64	vendor — specific area is used for various extensions to BOOTP

3.6 Domain Name Server (DNS)

Paul Mockapetris designed DS1S in 1984 to solve escalating problems with the old name- to-address mapping system. The old system consisted of a single file known as the host tables, maintained by the Standford Research Institute's Network Information Centre (SRI – NIC). As host names tricked in SRT – NIC would add them to the table—a couple times a week. Systems administrators would grab the latest version (Via ftp) and update their domain name servers.

The domain name system is a global network of servers that translate host names like www.oauife.edu.ng into numerical IP address, like 204.62.131.129, which computers on the Net use to commutate with each other. Without DNS, we'd all be memorising long numbers instead of intuitive URLs or email addresses. As you might guess, the domain system is fairly complicated, in fact, there are entire books on the subject. If you want to team more, there are several good resources on the web.

3.6.1 Authoritative Name Servers

Authoritative Name Server store information for the host names of subdomains. Each Authoritative Name Server hold in its database the name-to-address mappings for the group of host if administers.

Basically, each domain should have its own Authoritative Name Server. The local hostnames are assigned by the network administrator. In order to let other DNSs refer to a particular domain, the local DNS of that domain has to be refined by the other DNSs. The Authoritative Name Servers for every domain is officially registered with InterNIC.

In this unit, you have learned about the physical addressing of hosts and how it relates to the IP names and addresses. You also learned about the various class of IP addresses. We presented also how you can use the idea of subnets to expound a network. The issue of address resolutions were considered too.

What you have learned in this unit borders on physical addressing of hosts and how it relates to the IP names and addresses. The units that follow shall build upon these fundamentals.

a) What do you understand by the term "IP Address"
b) Discuss the various classes of IP Address
c) Discuss the benefits of subnetting in networks.

Exercise 1.1

How do we resolve the physical and network address?

Exercise 1.2

Discuse the importance of D N S

Stevens, R. *TCP/IP illustrated, 1996 V I, pp.* 53 to 68.

Droms, R. RFC 2131 *'Dynamic Host Configuration Protocol;* (1997)

Online Materials

http:www.cis-ohio-state.edu/htbin/rfc/rfc2131.htm
http://www.ipprimer 2nd/level.NetM DNS
http://www.cisco.com/warp/public/779/smbiz/service/knowledge/tepip/dhcp.htrn

Module 2: Internetworking, Network Design and Management

Page

1111111111111111

In this unit, you will learn about the concept of routing (of packets) over the Internet. You will also learn about the types of routing that we have and their specific advantages and disadvantage. The issues of RIP, ICMP, and Open Shortest Path First (OSPF) will be discussed. Let us now look at what you will learn in this unit as specified in the Unit objectives below:

[redacted]

By the end of this unit, you should be able to:

- Understand the concept of routing in a network.
- Discuss the adaptive and non adaptive routing schemes, including their merits and demerits.
- Appreciate the importance of routing tables in a network.
- Understand the routing protocols available in TCP/IP Suite.

In recent years, IP networking has gained enormously in popularity, spurred largely by the growth and popularity of the Internet. This section introduces the tasks, issues, and tools involved to effectively manage a collection of **IP** routers to form a stable, reliable IP network on which many organizations are coming to depend. Router operates at the network layer of the OSI model. Each individual network in a routed environ-ment is identified with a unique network address (or **IP** address). A router initially needs to be given the address of the network that it is directly connected to. Routers then learn about which distant networks exist and the best or optimal paths to them from other routers. When a router receives a packet that has a destination network address different from the source network address but reachable though a different port, it retransmits the packet out of that network. A router does not modify the addressing information in the packet; each router on an inter connected network only has to know destination network to currently forward the packet.

Routers are able to handle multiple paths between a source and destination easily. When a router learn of a distant network form another router, it also learns how many hop away that network is, that is, how many routers a packet must pass through, then, the one with the learn number of hops is used to foreword packets. This is know as distant — vector routing algorithm. Routers broadcast information about all known networks peridicatlly using Router Information Protocols (RIP) packets, usually every 30 seconds.

3.1 Routing Table Maintenance

Each machine in the IP network simply forwards the IP packets to the next hop without computing the entire path to the destination, all of the machines, and especially all of the routers, must have a consistent view of how to get to each destination. In other words, it is essential that their routing table be synchronised with each other. To understand why, consider what happens if router A and router B both believes that the other router is the correct next hop towards the destination 10.0. 0. 1. When router A receives a packet destined for 10. 0. 0.1, it will forward it to router B. Router B will consult its routing table and determine that the next hop is router A and forward it there. The result is known as a routing loop, and may involve more than just two routers.

IP routing specifies that IP datagrams travel through internetworks one hop at a time. As the next hop may or may not be the final destination, each intermediate device needs to match the destination address of the datagram with an entry in the current node's routing table. Each node involved in the routing process tries its best to forward packets based on internal information, regardless of whether the packets get to their final destination. An example of a routing table is as shown in Table 1.1. Whenever there is any error in the forward packets or a network is unreachable, it is not the node's responsibility to provide error reporting back to the source.

Table 1.1 *An IP Routing Table*

Network/host	Next hop
135.125.0.0	173.21.23.1
201.4.12.0	173.21.23.1
- - -	
215.93.3.0	212.34.20.254
134.122.0.0	212.34.20.254
	- - - -

In the next sub-sections, we will consider the various routing algorithms, and their merits and demerits

3.1.1 Static Routing (or Non-adaptive Routing)

The simplest routing technique to understand and implement is the static routing. In static routing each router is manually configured with the list of destinations, and the next hop to reach those destinations, by a configuration file stored on a stable storage (or cache). Making sure that all the routing tables are consistent is left to the network administrator. It is up to the administrator to make sure that no routing loop appears, and that all destinations are reachable from all routers. If consistency is lost, a packet never makes it to its destination. The advantage s of static routing are:

a) The static routing is predictable and this is because the network administrator computes the routing table in advance. The path a packet takes between two destinations is always known precisely, and can be controlled exactly.

b) Static routing does not impose any overhead on the router or the network. This is because no periodic broadcast of RIP information every 30 seconds is required.

c) Static routing is easy to configure on a small network. The network administrator simply tells each router how to reach every network segment to which it is not directly attached.

However, this routing techniques has some disadvantages, and these are:

a) The price of its simplicity is a lack of scalability. For a large network with several routers, computing
an appropriate route from every router to every destination is difficult. The task of precomputing routing tables quickly becomes a burden, and is prone to error.

b) When a network segment moves, or is added one would have to update the configuration of every router on the network. If one misses any, in the best case, segment attached to that router will be unable to reach the moved or added segment.

c) Finally, because static routing is, by definition static, it cannot use redundant network links to adopt to a failure in the network. If we add additional interfaces to one of the routers, and the routing table is left unch:enged, hosts on the new link would not be reached if the existing links failed because it could not adapt to any change. This inability to adopt to network failures, even when redundant paths are available is a big problem.

3.1.2 Dynamic (or Adaptive) Routing

This routing technique is a more flexible solution that allows the routers to compute their routing tables dynamically, bases on information provided by the other routers in the network. In general, the rooters speak a protocol that communicates information about the current functional topology of the network. From this information, the router computes one or more next hop rooters for each destination, trying to produce a path to the destination that is as close to optiomal as possible. If nothing interferes with the flow of rooting information between the routers, and if they all implement the protocol correctly, they will all compote routing tables that are consistent with each other. The advantages of dynamic routing technique are as follows:

a) The issue of scalability and adaptability is one of the major advantages of this schame. A dynamically routed network can grow more grickly and larger, and is able to adapt to changes in the network topology brought about by this growth or by the failure of one or more network components.

b) With a dynamic routing protocol, routers learn about the topology of the network by communicating with other routers. Each rooter announces its presence, and the routers it has available, to the other routers on the network. Therefore a reconfiguration is not needed if one adds a new router or **a new** segment. This reduces the chance that errors will occur.

c) The ability to learn about changes to the network's configuration means that the network redundant paths, then a partial network failure appears to the routers as if some segments got moved and they can be reached via alternate paths.

The disadvantages of dynamic routing are state below:

a). The complexity of this technique is one of its major disadvantages. Communicating information about network topology is not as simple as that. In addition, If a router is going to adopt to **changes** in the network, it must be prepared to remove old or unusable information from its routing table. How it determines what is old or unusable adds to the complexity of the routing protocol, **and** this complexity tends to lead to errors in the protocol's implication.

b). In order to communicate information about the topology of the network, routers must periodically send messages to each other using a dynamic rooting protocol. These messages must be sent a cross network segments just like any other packets.

But unlike other packets in the network, these packets do not contain any information to or from **a** user. Instead, they contain information that is only useful to the routers. This, from the users' point of view. those packets are pure overhead. On a low-speed link (especially wireless **link),** these messages can consume much of the available bandwidth, especially if the network is large **or** unstable. These packets increase the bandwith overhead and the network latency.

c). Finally, some or all of the machines in a network may be unable to speak any dynamic routine protocol, or they may not speak a common protocol. If that is the case, static routing may be the only option.

3.1.3 Hybrid Routing

With all the disadvantages listed of both static and dynamic routing, one may be wondering what the best choice is. There is a reasonable middle ground that limits the complexity of dynamic routing sacrificing its seal ability. This middle ground is a hybrid scheme.

In a hybrid routing scheme, some parts of the network use static routing, **and** some parts use dynamic routing. Which part use static or dynamic routing is not important, and many options are possible. One of **the most** common hybrid schemes is to use static routing on the fringes of the network (i.e. access network) and to use dynamic routing in the core and distribution networks.

The advantages of using static routing in the access network is that these networks are where the user machines are typically located; these machines often have only one or two router attachments, so the burden of configuring static routing is limited. It may even be possible to define nothing more than a default route on the stub networks. Also, due to the limited commections to these networks, we usually do not need to reconfigure routing on a stub network when it get moved to a new place in the network. On the other hand, distribution and core networks often have many router corrections, and therefore many different routes to maintain. Therefore, routers in these components of the network usually cannot get by with a default route. Furthermore, routers in the core and distribution networks usually need to be informed of changes on the connectivity of access network. While it is certainly possible to inform each router manually when a changes occurs, it is usually easier and more practical to allow dynamic routing.

3.2 The Routing Information Protocol

In this section, we introduce one of the most popular routing protocols in the TCP/IP suite. The reason for choosing the Route Information Protocol (RIP) to illustrate the concept the routing protocol is due to its popularity and its simplicity, RIP is a routing protocol originally designed for the Xerox PARC Universal Protocol. RIP became associated with both Unix and the TCP/IP suite in 1982. Popular NOSs such as Windows NT 4.0 only supports RIP as their routing protocol. Hence it is important to understand the concept of RIP.

Each entry in a RIP routing table (as show in Table 1.2) provides information, including the ultimate destination, the next hop on the way to that destination, and a metric. The metric is equal to the distance in number of hops to the destination. Other information can also be presented in the routing table, including various timers associated with the route.

Table 1.2 An RIP Routing Table

Destination	Next hop Metric	Metric	Timer
Network 135.23.0.0	Router 1	3	11
Network 198.23.43.0	Router 4	5	21
Network 202.168.2.0	Router 4	3	15
Network 212.15.23.0	Router 3	4	45

3.3 Open Shortest Path First

Since a router with RIP will send routing information to its neighbour router every 30 second, it may take a relatively long period to propagate its information across the whole network. The delay of the update in routing information may affect a router's ability to choose the right route for the data. Convergence of routing information implies the stabilising after something changes, such as a router or a link going down. Open Shortest Path First (OSPF) is a newer technology than RIP as an interior gateway protocol. It overcomes the major problem of the RIP, which is show convergence. Unlike the concept of hop count applied in RIP, OSPF is a link state protocol. In a link state protocol, a router does not exchange distances with its neighbours. Each router frequently check the status of its link to each of its neighbours, sends this information to its other neighbours, which then propagate the link status throughout the autonomous system. An autonomous system is a system in which a group of routers are under the control of an organization. A corporate or enterprise network, for example, can be considered as an autonomous system. The link status of each connection between routers in the autonomous system will be updated frequently and build a complete routing table.

In a network running the OSPF routing protocol, routers will send 'Hello' messages to other routers to check if these neighbour are awake. A designated router in the autonomous system is selected to be responsible for updating its adjacent neighbours with the latest network topology information. When the status of a link is changed, the designated router will be responsible for making sure that all routers in the network know the update status of the link for the sake of retrieving current routing information, there are five message type used in the OSPF protocol to exchange link status.

Hello ____ this is used to identify neighbours, to find out about an existing designated router, and as a heart heart signal.

Database Description ___ this is used to initialise a router so that it can find out what data is missing in its routing table.

Link State Refused— this is used to ask for data that a router has discovered is missing from its routing information.

Link State Update— this is used to dynamically report changes in network status.

Link State ACK— when a router receives a Link State Update, it will send an acknowledgement to the sending router.

3.4 The Internet Control Message Protocol

The Internet Control Message Protocol (ICMP) can be regarded as a network helper. It perfomes a number of tasks within an IP environment. ICMP is able to report routing failures back to the source. In addition, ICMP provides helpful message such as the following:

- To test node reachability across an Internetwork with echo and reply message
- To stimulate more efficient routing with redirect message
- Time exceeded message to inform sources that a datagram has exceeded its allocated time to exist within the Internetwork.
- Rooter advertisement and rooter solicitation messages to determine the addresses of rooters on directly attached subnetworks.

Conclusion

In this unit, you have learned about the IP routing techniques, their various types, including their advantages and disadvantages. You also learned about the various IP routing protocols and their usefulness. The important of the routing table were discussed also,

5.0 Summary

What you have leaned in this unit borders on IP routing schemes, their merits and dements. You also learned about the various routing protocols available and their importance. Finally, you learned about the importance of proper management of routing tables. The units that follow shall build upon these fundamentals.

6.0 Tutor........Marked Assi

a. Explain the concept and importance of Routing Table Maintenance.

b. Write short notes on Static and dynamic routing schemes.

What are their advantages and disadvantages?

Exercise 1.1 Discuss
the IP routing

Emilie 1.2
Discuss the 1 CMP

Stevens, R. TCP/IP Illustrated, (1996) VI, Pg 69- 80, Pg 85-95

Online Materials
http://www.primer2ndlevel.net/fi Routing
http://www.2.sco.com: I996/Net AdminG/BookCHAPTER-5.html
http://www.oline.vuse.Vanderbiltedu/federation/mod-51htm

Module 2: Internetworking, Network Design and Management

Unit 3: The Transmission Control Protocol

Page

In this unit, you will learn about the vital role played by TCP in TCP/IP protocol suite. You will also learn abort ports and socket addresses. How TCP provides correction- oriented data transport will be discussed too. Let us now see what you will learn in this unit as stated in the unit objectives below:

By the end of this unit, you should be able to:

- analyse the importance of TCP in the TCP/IP protocol suite
- understand the difference between ports and socket addresses
- discuss the synchronisation process in a TCP correction.

The Internet transport layer is implemented by the Transmission Control Protocol and User Datagram Protocol. TCP provides correction- oriented data transport, while UPP is correction less. The correction- oriented data transport of the TCP correction is a reliable application-to — application transmission. For applications using TCP as the transport layer protocol, TCP is able to provide reliable services. When data is lost or duplicated during the transmission, TCP will automatically discard the duplicated data and handle the retransmission of the missing data. TCP provides full-duplex, acknowledged and slow-controlled services to upper layer protocols. In TCP communication, each outgoing data unit from a sender will have to wait for its acknowledgement from the receiver. This guarantees the delivery of data to the destination.

Furthermore, the acknowledgement signal can also be used to control the data flow between hosts. When a system has sent data out without receiving the corresponding acknowledgement, the system will stop further transmission of data until the receipt of the acknowledgement. Hence, TCP is able to provide a flow control feature. Each data unit in TCP is called a 'segment.'

Segments are transmitted in a continuous, unstructured octet stream where octets are identified by sequence numbers. Similar to other transport layer protocols, TCP can also support numerous simultaneous upper layer conversations. By using different TCP port numbers, hosts can simultaneously set up multiple connections with different destinations.

3.1 Ports and Socket Addresses

To address a connection for each application, the IP address of the corresponding host is not sufficient to fulfill the requirement of multiple connections. It is required to provide its TCP port number, accompanied with the host address, to specify a particular connection. The range of port number is from 0 to 2^{16}-1 for TCP or UCP.

Ports in the range 0 to 1023 are well-known services ports. For example, the port numbers of FTP, Telnet and the Simple Mail Transfer Protocol (SMTP) services are 21,23, and 25 respectively. The services of these ports are standardized in Internet applications.

The combination of the IP address and the port used for communication is called a socket address. The socket address is unique for a session of communication between hosts. Let's use FTP services as an example. A client makes two session of FTP connections to a FTP sever with IP address 153.34.2.56. In order to correctly separate the packets for both sessions, the packet headers contain two different socket addresses for the client 153.34.2.56, port 4233, and 153. 342.56, port 3245. The first FTP session uses the socket 153.34.2.56, port 4233, and the next session uses 153.34.2.56, port 3245.

Hence, there is no problem for the server to reply correct data to appropriate applications.

3.2 Connections

Before starting a TCP communication, it is necessary to establish a connection between the hosts. The scenario of the connection procedure involves six steps. They are described below:

a. A server host open a TCP port for services --e.g, so for a Web Server. The server listens to the port and accepts connection if there is any request from clients.

b. A client initiates a request to start a connection to the server at the given port and IP address.

c. The client sends a synchronisation segment with an initial sequence number- lets say 100

3.3 Flow Control in TCP

Figure 1.1 shows a simple flow control. After the client sends a segment with the sequence number 201, the server delivers two segments back to the client. Note that a sender does not have to wait for acknowledgement of each packet before sending more data. This approach can improve the data through pot of the communication.

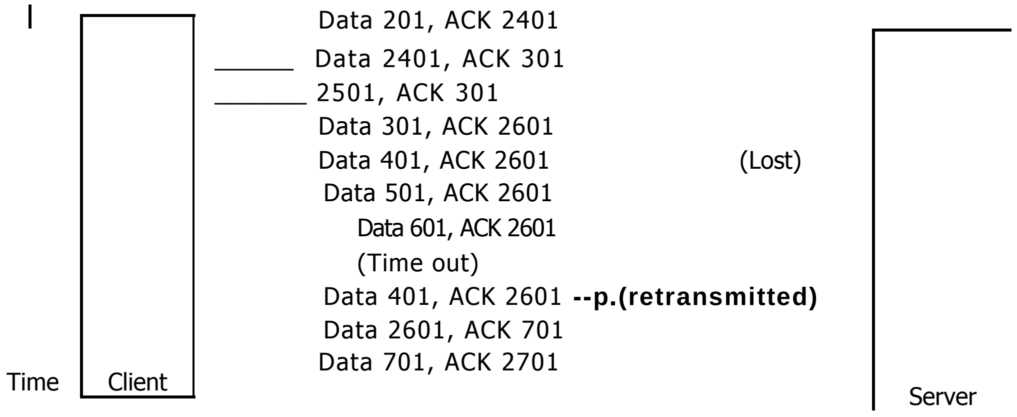


Figure 1.1 Data transfer and retransmission

As one mentioned earlier, TCP is a reliable transport protocol which provides the retransmission corrupted or lost segments. Referring to Figure 1.1, the segment, Data 401, ACK 2601' is lost in the transmission. Since the server does not receive the segment 'Data 401, ACK 2601', there is no further acknowledgement or reply to the client. After a predefined period of time, the client determines that the segment 'Data 401 ACk 2601' is lost and the client starts retransmission of the segment. The communication can proceed properly if the acknowledgement segment is sent to the client. You may note that the last two segments have the ACK number701. It is because Data 501 and Data 601 have been received by the server. After the finish of the Data transfer, it is necessary to close the connection session. The close procedure involves the following request reply process:

1. An application has finished its work and it tells TCP to close the connection.
2. The host sends a close segment to inform its partner that it will send no more data.
3. The partner replies the close request segment and stops its application.
4. The partner will also send a close segment to the host to confirm the end of communication.
5. After receiving the close segment for the partner, the host replies with the acknowledgement seg-ment and stops its application.

3.4 Retransmission Timeout

After sending a segment, TCP sets a timer and listens for an acknowledgment (ACK). In the case of no ACK reply, the sender assumes the segment is lost in transmission and it will retransmit the segment to the receiver. However, this leads to a direct question of how long the timeout should be. If the retransmission is too short, duplicated segments may be retransmitted frequently to the network. Unnecessary burden on network traffic degrades the performance of the network.

On the other hand, if time outs are too long, it will prevent prompt recovery when a segment really haS been destroyed, and will decrease the network response time. In order to design a suitable timeout parameter for TCP communication, an algorithm with adaptive feature is developed according to the network status. In this section, we introduce Karn and Jacobson's Algorithm to determine the timeout parameter.

In order to determine the timeout parameter, the network traffic response time is a good reference to adjust the time parameter. A round trip time (RTT) is defined as the time taken between the transmissions of data and the arrival of matching acknowledgements.

You can get the value of RTT by using the 'ping' command with an option `—s'. The smoothed round trip time (SRTT) can be obtained by using the average of a number of send/ ACK trails' RTT. A more responsive measure is to use a weighted sum, computing a SRTT as below:

New SRTT= (1- * (previous SRTT) + A * (latest RTT) where 0 1 1

We can control the weight of the old SRTT to the new SRTT. The typical value of 1 is 1/8 in practice.

The SRTT plays a major role in determining the timeout parameter. Since the individual RTT may vary from the SRTT, it is a reasonable thing to estimate the timeout parameter with SRTT and a smoothed deriation (SDEV)

Time out = SRTT + 2*SDEV

and

DEV= 1(last RTT) — (old SRTT)1

In this unit, you have learned about the important role of TCP in communication over the Internet.

You also learned about the ports and socko addresses and their relationship. The reason why TCP is regarded as connection-oriented was presented too.

What you have learned in this unit focuses on the vital role played by TCP in internetworking.

You also learned about ports and socket addresses. The units that follow shall build upon this.

a) With appropriate example, explain the difference between ports and socket addresses

b) With the aid of a diagram, explain the concept of flow control in TCP Communication.

Exercise 1.1

Discuss the Internet Transport Layer

Exercise 1.2

Write on retransmission time out

Stevens, R. *TCP/IP Illustrated.* 1996, V.I, PP 223- 226

Felt, S. TCP/IP: *Architecture, Protocols, and Implementation.* New York: Mc Graw, 1993.

Online Materials

http://www.oline.vuse.Vanderbilt.eduffederation/mod}fhtm

http://www.pcIt.cis.yale.edu/pcIt/comm./tepip.htm

Module 2: Internetworking, Network Design and Management

Unit 4: The User Datagram Protocol

Page

In this unit, you will learn about the basic operations of the UDP and its importance in transport–layer protocol. You will also learn about the important features of the UDP that makes it preferable to TCP for some applications. Finally, the various Internet applications that uses UDS will be discussed including the checksum for error detection. Let us now look at what you will learn in this unit as stated in the unit objec-tives below

By the end of this unit, you should be able to:

- discuss the underlying principles behind UDP transport—layer protocol.
- understand why UDP is preferred to TCP in some applications.
- use the UDP checksum to detect errors in the data transmitted.

11111 [redacted] 1111111111

The UDP is a simple transport-layer protocol. It is described in RFC 768. The application units a datagram to a UDP socket, which is encapsulated as either an IPv4 datagram or an IP v 6 datagram, which is then sent to its destination. But there is no guarantee that a UDP datagram ever reaches its final destination. If you want to be certain that a datagram reaches its destination, we must build lots of features into our applications: acknowledgments from the other end, time-outs, retransmissions, and the like. Each UDP datagram has a record. If the datagram reaches its final destination correctly (that is, the packet arrives without a checksum error), then the length of the datagram is passed to the receiving application.

You should note that UDP provides a correctionless services as there need not be any long-term relationship between a UDP client and server. For example, a UDP client can create a socket and send a datagram to a given sever and then immediately send another datagram on the same socket to a different server. Similarly a UDP server can receive five datagram in a row on a single UDP socket, each from five different clients. The UDP datagram provides:

- a source port – a 16-bit port number
- a destination port – a-16 bit port number
- the length (of UDP header + data) – a-16 bit count of octets in the UDP datagram
- a UDP checksum – a 16-bit field; if zero, then there is no checksum or else it is a checksum over a pseudo – header + UDP data area.

UDP uses a pseudo- header to verify that the UDP datagram has arrived at the correct host address with correct port number. The UDP pseudo – header consists of the source and destination IP addresses. Note that the source address, destination address and protocol field are taken from the IP packet header. The purpose of the UDP checksum is to check the correctness of a datagram. The UDP pseudo – header is show in figure 1.1 below:

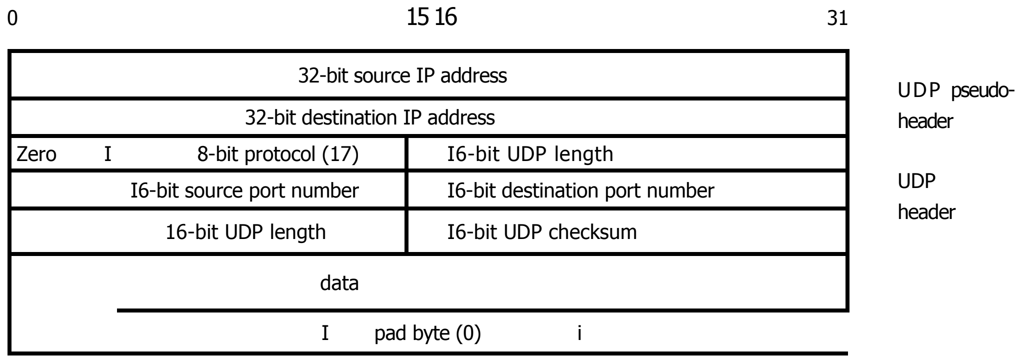


Figure 1.1 The UDP pseudo – header, used in computing the UDP checksum

3.1 Features

Now you might be wondering why an application developed would ever choose to build an application over UDP rather than over TCP. Is not TCP always preferable to UDP since TCP provides a reliable data transfer service and UDP does not? The answer is no, as many applications are better suited for UDP for the following reasons:

a) No Correction Establishment: Note that TCP uses a three-way handshake before it starts to transfer data, UDP just blasts away without any final preliminaries. This UDP does not introduce any delay to establish a correction. This is probably the principal reason why DNS runs over UDP rather than TCP-DNS would be more slower if it ran over TCP.

b) No Connection State: TCP maintains connection state in the end systems. This connection state includes receive and send buffers, congestion control parameters, and sequence and acknowledgement number parameters. These state information is needed to implement TCP'S reliable data transfer service and to provide congestion control. UDP, on the other hand, does not track any of these parameters. For this reason, a server devoted to a particular application can typically support many more active clients when the application runs over UDP rather than TCP.

c) Small Segment Header Overhead – The TCP segment has 20 bytes of header overhead in every segment, whereas UDP only has 8 bytes of overhead.

d) Unregulated Send Rate: TCP has a congestion control mechanism that throttles the sender when **one** or more links between sender and receiver becomes excessively congested. This throttling can have a severe impact on real–time applications, which can tolerate some packet loss but require a minimum send rate. On the other hand, the speed at which UDP send data is only constrained by the rate at which the application generates data, the capabilities of the sources (CPU, chock rate, etc.) and the access bandwidth to the Internet. We should keep in mind, however, that the receiving host does not necessarily receive all the data–
when the network is congested, a significant fraction of the UDP- transmitted data could be lost due to router butter over flow.
Thus, the receive rate is limited by network congestion ever if the sending rate is not constrained.

3.2 Internet Applications and Transport Layer Protocols

Table 1.1 lists popular Internet applications and the transport protocols that they use. As we expect, e-mail, remote terminal access, the web and file transfer run over TCP—these applications need the reliable data transfer service of TCP. Nevertheless, many important applications run over UDP rather than TCP. **UDP is** used for RIP routing table updates, because the updates are sent periodically, so that lost updates are replaced by more up-to-date updates. UDP is used to carry network management data e.g. SNMP. UDP is preferred to TCP in this case, since network management must often run when the network is in a stressed state—precisely when reliable, congestion – controlled data transfer is difficult to achieve. Also as we mentioned earlier, DNS runs over UDP, thereby avoiding TCP'S connection establishment delays.

As shown in the Table 1.1, UDP is also commonly used today with multimedia applications, such as Internet phone, real-time video conferencing, and streaming of stored audio and video. You will recall that interaction real-time applications, such as Internet phone and video conferencing react very poorly to TCP's congestion control. For these reasons, developers of multimedia applications often choose to run the applications over UDP instead of TCP. Finally because TCP cannot be replayed with multicast, multicast applications

Application	Application-layer Protocol	Underlying Transport Protocol
Electronic mail	SMTP	TCP
Remote terminal access	Telnet	TCP
Web	HTTP	TCP
File transfer	FTP	TCP
Remote file server	NFS	Typically UPP
Streaming multimedia	Proprietary	Typically UDP
Internet telephony	Proprietary	Typically UDP
Network management	SNMP	Typically UDP
Rooting protocol	RIP	Typically UDP
Name translation	DNS	Typically UDP

Table 1.1 Internet applications and their underlying transport protocols

Although, as commonly done today, running multimedia applications over UDP is controversial to say the least. As we mentioned above, UDP lacks any form of congestion control. But congestion control is needed to present the network from entering a congested state in which very little useful work is done. If everyone were to start streaming high bit-rate video without using any congestion control, there would be so much packet overflow at routers that no one would see any thing. Thus, the lack of congestion control in UDP is a potentially serious problem. May researchers have proposed new mechanism to force all sources, including **UDP** sources, to perform adaptive !thigestion control.

3.3 UDP Checksum

The UDP checksum provides for error detection. UDP at the sender side performs the one's complement of **the** sum of all the 16-bit words in the segment. This result is put in the checksum field of the UDP segment. **But note that** the check soon is also calculated over a few of the fields in the IP header in addition to the UDP segment. But we ignore this detail in order to see the forest through the trees. When the segment arrives (if it arrives!) at the receiving host, all 16-bit words are added together, including the checksum. If this soon **equals 1111111111111111,then** the segment has no detected errors. If one of the bits is a zero, then we know **that errors** hare been introduced into the segment.

Here we give a simple example of the checksum calculation. An example, suppose that we have the following three 16-bit words:

0110011001100110
0101010101010101
0000111100001111
The sum of the first two of these 16-bit words is:
0110011001100110 0101010101010101
1011101110111011 Adding the third word to the above sum gives:
1011101110111011
0000111100001111
1100101011001010

The 1 's complement is sustained by converting all the O's to 1 's and converting all the 1 's to O's. At the receiving end, all four 16-bit words are added, including the checksum. If no errors are introduced into the segment, then clearly the sum at the receiver will be 1111111111111111. If one of the bits is a zero, then we know that errors have been introduced into the segment.

In this unit, you have learned about the underlying principles of the UDP transport-layer protocol. You also learned about some of the advantages of the UDP over the TCP and why it is preferred in some applications. Finally, you should have understood how UDP checksum can be used to check errors in data transmitted. over the UDP transport-layer protocol

[illegible]

What you have learned in this unit focuses on the underlying principles of the UDP transport layer protocols. It also borders on the features of the UDP and how UDP check soon can be used to detect errors in data transmission. The units that follows shall build upon these revelations.

[illegible]

a) Explain why UDP is preferred to TCP in some applications.
b) Discuss the principle of the operation of the UDP checksum

11111111S111111111

Staven, R. *TCP/IP Illustrated, 1916* v.i pp. 145-153.
Feit, S. *TCP/IP: Architecture, Protocols, and Implementation,* New York: MC Graw,1993.

Online Materials

http://www.ippriimer. 2ndlevel.net/#TCPUNP
http://www.psc.edu/networking/tcp-friendly.html
http://www.-net.cs.umass.edu/kurose/transport/UDP.html

Module 2: Internetworking„ Network Design and Management

Network Design Goal

In this unit, we will present the guidelines for designing a computer network, including both Local Area Network (LANS) and Wide Area Networks (WAN S). You will also learn about the goals of network design in general. This is because network requirements should be carefully analysed before the network is built. We will also focus on the role of the designer in this process. Let us now see what you will learn in this unit as stated in the unit objectives below.

By the end of this unit, you should be able to:

- analyse the trade-offs among key network variables.
- evaluate the costs required and the benefits provided by the network.
- determine the specific size and type of circuits and equipment that constitute the network system.

After studying the previous three unit, you should be familiar with various concepts in computer networking. In the unit, we will move a step further on to the design of a network. The design process involves all of the network components and this can help you consolidate everything you have learned.

Network design is an important phase in developing a network. It is important to know that there is no absolute answer in network design. However, one goal you must achieve is that the network must work. In this unit, you will find that network design is actually a trade-off between cost and availability. What does availability means? The answer may be different to a user and a network designer. A user may care about the response time, the thorough put, and the reliability of a network. A network designer, on the other hard, may be very concerned about the flexibility towards changes and the manageability of the network. Network designers should always try to maximise availability and minimise cost.

You can set up a network without having a design phrase. However, it is not a good practice, especially forthe development of a large network. The flexibility and operate path of the network greatly depend. On the initial design of a network. Network design is actually an interactive process. You must first analyse the requirements of the network by way of stated goals. Then you have to make choices in various technical options in designing the infrastructures. After building the network, maintenance and troubleshooting help enhance its performance.

Indeed, requirements for network design solutions have changed a great deal in recent years, as new technologies have emerged quickly. These changes make design more difficult than ever. The trend is towards increasingly complex environments involving multiple media, multiple protocol interconnection to networks outside any single organisation's domain of control. Carefully designing networks can reduce the hardships associated with growth as a networking environment evolves.

3.1 The Designer's Role

The technical details of design can be extra-ordinarily complex. Before they reach a decision, companies frequently seek outside expert opinion, spend substantial time meeting with key vendors, and carry out detailed technical and economic analyses. The designer is often the coordinator of these activities and may not have the in-depth technical knowledge of other experts. But experts knowledge is only of real value if the designer provides the management framework. The telecommunications business opportunity checklist facilitates the client's business-centered imput to the design dialogue, and the telecommunication platform map translates these into criteria for design. Managing network design involves ensuring that these are then revived in terms of the following four key network design valuables.

- The capability of the network: What business functions does it deliver?
- Its flexibility: How easy will it be to add and extend business functions?

- The quantity of service: How reliable is it?
- Cost: What's the price tag, and is it worth it?

A complete network design process includes a high-level analysis of client preferences and trade off with respect to important design features that frame the capabilities and limitations of the network.

3.2 Design Goals

Technology solutions that ignore the under business vision typically occur as a result of technology-generated problems. Flexibility refers to the ease and speed with which changes can be made to any part of the telecommunication network platform and to the range of changes that can be made without having to replace, redevelop, or throw out existing network components. One way to build flexibility into a network architecture to permit the addition or deletion of small capacity increments is by relying on public network services, or virtual private network arrangements with carriers.

Public network service offerings have build-in flexibility because they consist of dial-up connections established as needed and have the ability to change the capacity of the service through a simple order change with the carrier that provides the service.

Scalability refers to the network's ability to cope with the growth of the organisation, such as supporting more users and applications. Adaptability refers to the network's ability to adopt future technologies. A network should not contain an element which would limit the adoption of new technologies.

Scalability and adaptability match user's concerns about flexibility. Statistics gathered on monitoring, and experiences, accumulated in managernent of a network, will be important to the improvement or future redesign of the network. Designers may regard mangeability as one of their design goals. Indeed, manageability is also important in ensuring the ongoing stability of operations and availability of resources of the network.

However, users may not have the same concerns about the manageability of a network as previously discussed.

A designer is responsible for weighing the costs required and the benefits provided by the network, so cost effectiveness is another main concern, Large organisations increasingly rely on electronic data for managing business activities and a lot of companies even start their business through the Internet. It is foreseeable that the associated costs of developing a network will continue to rise, and consequently it is increasingly important for a designer to maintain a balance between network capabilities and network development cost.

In this unit, you have learned about the network design goals. You also learn about the key design considerations in setting up a network. Requirements for network solutions and trader offs were considered, and their effects evaluated.

What you have learned in this unit borders on the network design goals. We also focused on key design issues for a network. The units that follow shall build upon these issues

a. Explain the role of a designer in any network design.

Exercise 1.1
Discuss the role of a designer in a network.

Exercise 1.2

What are the goals of any network design?

7.0 References and

Peterson, Li. and Davie, *B.S. Computer Networks: A systems Approach*, San Francisco Morgan Kaufiinann., 1996.
Keen, P.G.W. and Cummins, J.M.Netvvorks in *Action: Business cliouves and Telecommunication_Decisions, Belmont,* California, USA: Wadsworth Publishing Company, 1994

Online Materials

http://owww.techweb.cmp.comincinetdesigniouline.htm
http.//www.techwe.cmp.cominc/netdesignintmain.htm/

Module 2: Internetworking Network Design and Maintenance

Unit 6: Analysis of Network Requirements

1.0-Introduction:'.

In this unit, you will learn about the analysis for network requirements. The issue of traffic analysis will be discussed. You will also learn about the geographical considerations, future expansion and network simula-tion. Let us now see what you will learn in this unit as stated in the unit as stated in the unit objectives below.

2.0 Objectives

By the end of this unit, you should be able to:

- understand the issues involved in analysing network requirement.
- understand the traffic analysis of a network.
- know how to apply network simulation for solutions to problems.

3.0 Overview

As client/server information systems and distributed computing applications have put increasing demands on the local area network infrastructure for data traffic transfer, network architects and technology providers have responded with alternative solutions. One solution to the network bandwidth crunch is to offer higher-speed shared-media network architecture such as 100Base T, 100VG-AnyLAN, and isochronous Ethernet, It is important to note that implementation of switched LAN Architectures only changes the wiring center technology and, as a result, the manner in which workstations set up point-to-point communications to each other. In other words, network interface cards. network interface card drivers, and media do not change. For this reason, installing a LAN switch is often the first, easiest alternatives chosen when network bandwidth demands exceed current supply. To go from an Ethernet shared-media hub with an Ethernet LANs switch. Note that there are things that we can do in the design stage to avoid security loopholes. For example, you may control the access to Internet work devices like bridges and routers and local traffic in individual LANs of the branches so that users cannot reach the bock bore in appropriately.

3.1 Traffic Analysis

You may find that your users need constant changes in network functionality, in response to changing business conditions and changing technologies. Upgrading and resigning of the existing network, rather than designing a network from scratch, is what designers usually face. For example, the increasing popularity of voice and video-based network applications may add pressure to increase the, bandwidth of the current network. In addition to considering the cost, a designer may also have to work out a careful procedure for upgrading so as to avoid distruption of the services provided to the users.

It is important to analysis the traffic patterns, between attached clients and servers before swapping out hubs and installing switches. To minimise the interswitch traffic, which represents a potential bottleneck, the fonowing suggestions will suffice:

a) It is important to have those workstations and servers that communicate most often with each other on the same switch, since switches differ in cascadability and speed of inter switch communications connections.
b) Another benefit of analysing traffic patterns before installing the switch is the ability to identify those users and \ orkstations that need a dedicated switch port and those that can reside on a shared LAN segment attached to a switched ports.

'Hie following are a few general guidelines for switch port allocation:

a) Servers and UNIX working stations should ideally have their own suritch ports.
b) Distributed computing power users with frequent queries to servers should be able to connect to switch ports via shared LAN segments of up to eight users.
c) Casual or light traffic users accessing only e-mail and terminal, character-based programs can be connected to switch ports via shared LAN segments of 50 or more users.

The ability of LAN switchers to support multiworkstation LAN segments on a single switch port may vary among switchers. The number of workstation addresses which can be supported by a given switch port may vary as well.

3.2 Geographical Considerations

Network nodes may be confined to a single office within one floor or one building. However, nodes may also be distributed in several locations across cities, countries or even continents. As we discussed previously, we cannot use Ethernet in a very large area since the technology imposes a distance limit between the nodes. Very often, nodes in one location will be connected to form a LAN with technologies like Ethernet and Token Ring. Then, several LANS in different geographical locations will be connected through Internet work devices like bridges and routers to form a WAN as shown in Figure1.1. Since data flow will usually be more intensive within the same location, such a design can isolate the network traffic to individual locations, thus not overloading the whole network. In WAN connections, WAN technologies like dedicated lines, X.25 have to be used. These technologies provide a speed of data transfer. Relatively slower than those of LAN technologies.

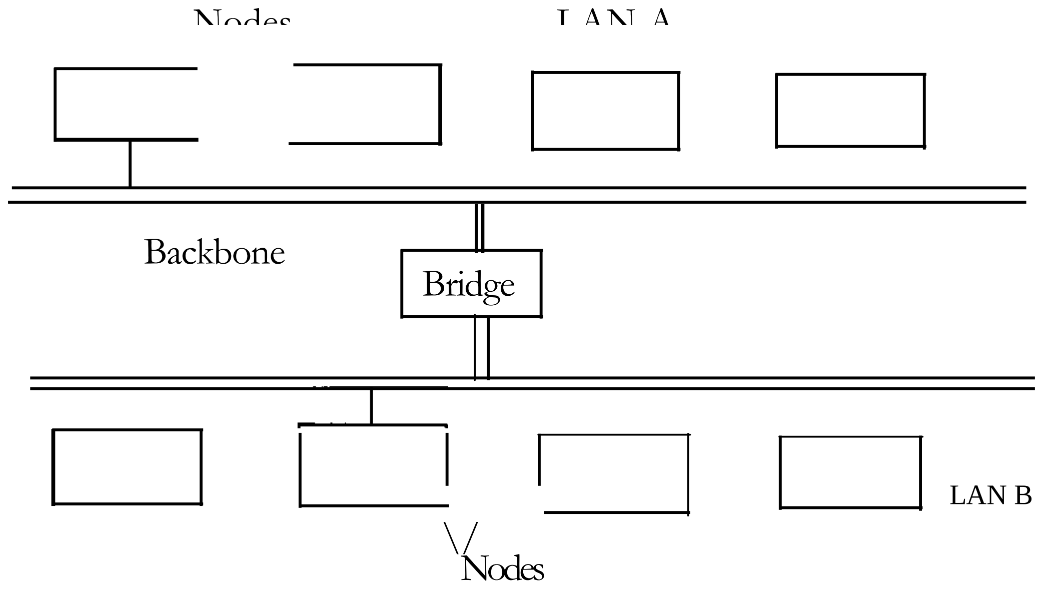


Figure 1.1 Bridge connecting two different LANs

Network applications usually adopt a client /server architecture. A central database located in a server, or distributed in several servers is accessed by clients distributed a cross the network. In this case, the servers should be connected to the network through a link with a larger bandwidth and a higher speed compared to links connecting the clients in a separate segment, as shown in Figure 1.2. So, when multiple clients access the server database, The link connecting the server will not become a bottleneck slowing the response time of the application.

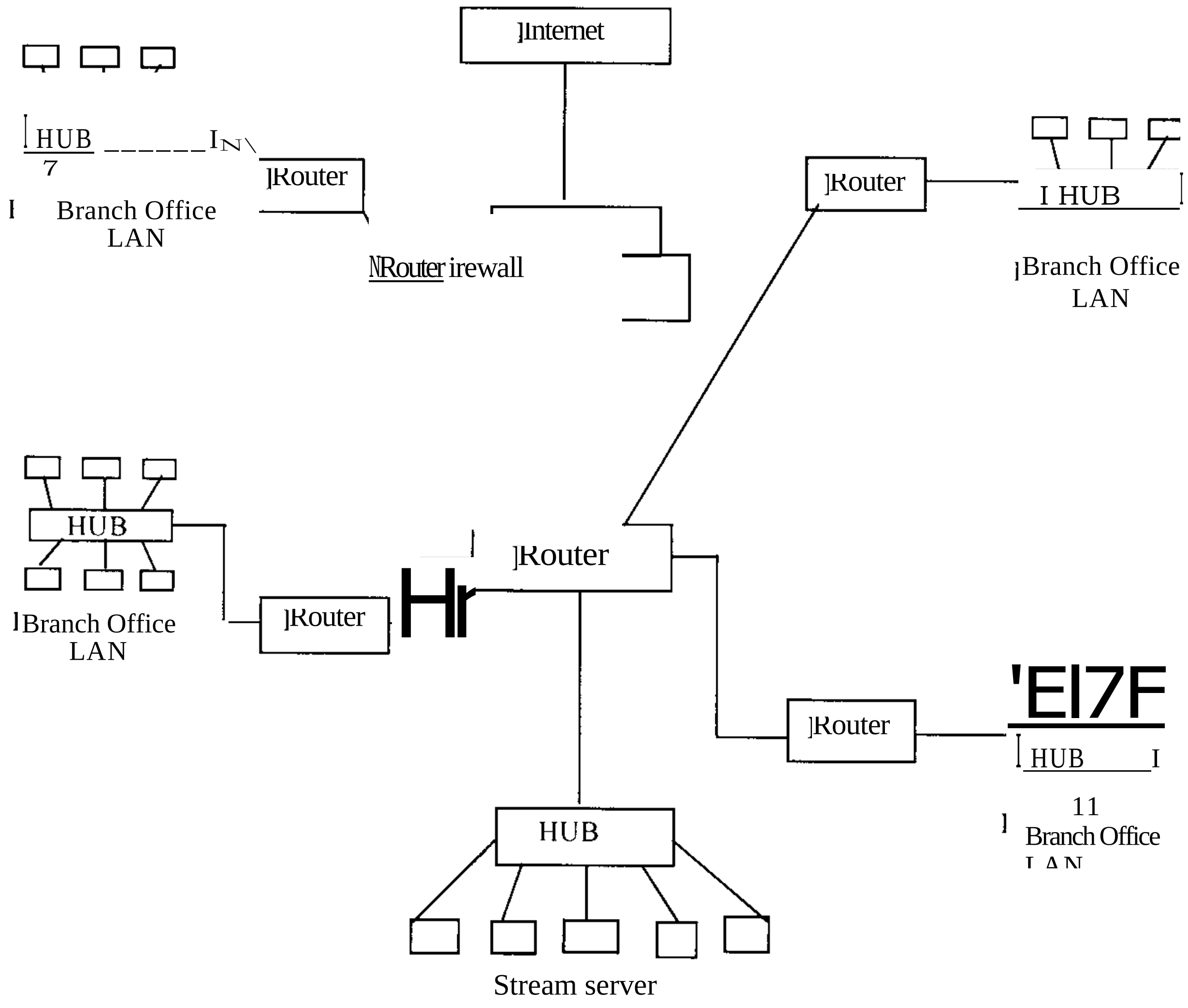


Figure 1.2 Network traffic in client /server architecture

3.3 Future Expansion

Users have changing demands on the functions of a network in response to changing business conditions and information technologies that are developing very fast. It is abv ions that a network design cannot cater for all the demands of the future. However, the network should be designed with an eye toward future technologies, so that new technologies can be adopted for upgrading the network when required.

As stated in the unit on design goals, scalability and adaptability should be catered for in designing a network. These are important metrics for measuring a network's ability in dealing with future expansion. Scalability involves the capability of the network to deal with the growth of the organisation; that is, the increasing number of users, applications and data volume that lead to the increasing utilisation of the network. For example, the organisation may have a plan to use video-based application in the future. The designer should then pay more attention to the bandwidth requirement of the network to be designed as video-based applications put a lot of pressure on the bandwidth of the network. For both LAN and WAN, the recurring costs typically tend to dominate. Therefore, in considering the trade-offs among the design variables mentioned in the design goals unit, you should First think of how to reduce the recurring costs.

3.4 Network Simulation

Simulation of network loading can provide a lot of emperical information for designers. This helps the

designers in their design development as it can always give them a rough estimate of the real network loading, The modeling should consist of the installation of a working network and monitoring traffic for a given number of users, applications and network topology. Simulation prevents over-complication of the design solution and highlights any area where the network design solution does not meet the specified needs.
A similatoin can give you a rough idea of the performance of the network. Certainly, if you input more information about the modeling, you can achieve better results. Furthermore, you can try extrapolating to the predicted future number of users, applications and topology.

In this unit, you have learned a number of important issues that relate to the major roles of analyzing the network requirements. You also learned about the key design issues like traffic analysis, geographical considerations, future expansion and network simulation.

What you have learned in this unit borders on issues that affects the analysis of the network regiments. The units that follow shall build upon this fundamental

a. In any network design, traffic analysis is very vital to its effective functioning. Discuss.

Exercise 1.1
What will you put in place in a network design to cater for future explosion.

Exercise 1.2
Discuss the geographical consideration in a network.

Goldman, J.E. *Local Area Network- A Client/Sever Approach.* (2nd ed.) New York: John Wiley & Sons, Inc, 1997

Held, G. *LAN Performance—Issues and Answers*, (2nd edu.), chichester, Uk.John Witey & Sons Limited. 1998

Online Materials
http://www.techweb.comp.com/nc/netdesignintmain.htm/
http://www.techweb.cmp/Htmnetdesigncom/nc/ outline http

Module 2: Internetworking, Network Design and Management

Unit 7: Designing a Network Infrastructure

Page

What you will learn in tis unit borders on the issue in designing a network infrastructure. You will also learn about the details of how bridges, switches and routers actually help in internetworking. You will appreciate the issue of backbone strategy and model components. Let us now see what you will learn in this unit as stated in the unit objectives below

By the end of this unit, you should be able to:

- understand why an organisation would want to implement LAN-to-LAN internetworking.
- understand the basics of internetworking design, incluing decisions about bridging, routing, or switching
- understand the importance of protocols to successful internetworking design and implementation.

First of all, one would like that you appreciate how the effective bandwidth of a link varies with number of nodes. Table 1.1 shows the variation of the effective bandwidth of an Ethernet network.

Table 1.1 Effective bandwidth of an Ethernet Network

Number of nodes in the segment	Utilization rate (approximate)	Effective bandwidth	Effective bandwith per node
1	100%	10 Mbps	10Mbps /1 = 10Mbps
10	70%	7 Mbps	7Mbps /102 = 700Kbps
100	35%	3.5 Mbps	3.5Mbps /108 = 35Kbps

You can obviously see from the above table that the effective bandwidth per note decreases with increase in the number of nodes in the network. This leads to serious congrestion problem, which eventually slows down the network.

Segmentation is usually the first approach to reducing share media congestion. Segmentation is the process of spliting a single collision domain into two or more collision domains. Bridging and switching, which operate in the data-link layer, (layer two of the OSI model) can be used in segmentation and to create separate collision domains. This results in more bandwidth being available to individual nodes. This is called *microsegmentation.* Switches can also isolate each node connecting to one of its ports into a collision domain, and this are usually used in implmenting microsegmentation.

Routers can also be used in segmenting a collision domain. However it is a more intelligent device which is usually used in isolating broadcast domains and implementating segmenting policies. You should notice that the packets broadcast using the netwok layer(layer three of the OSI model) address can still pass through all the collision domains. The segments connected through bridges or switches represent a single broadcast domain. Although bridges and switches will not forward collision, they will forward broadcast packets. Hee rwe introduce the term broadcast radiation which refers to the way that the broadcasts are transmitted out from a source node causing all the nodes in the same broadcast domain to undertakes extra processing. Routers can act as a broadcast filter to isolate broadcast radiation within a segment. In the following section, you will learn about the details of how bridges, switches and routers actually help in internetworking.

3.1 Bridging Switching and Routing

Bridging, switching and routing are common internetworking process in network segmentation, among which, switching is a relatively new technology. In the following sub-sections, we will look at the specific details of these terms.

3.1.1 Bridging

Bridges also provide certain features that routers cannot achieve. Under certain circumstances, we may choose bridges to implement the internetworking. The advantages of using bridges are given below. These are useful guidelines when choosing between routing and bridging.

(a) The Installation of bridges are very simple: they do not require configuration. You do not need special knowledge like the configuration of routers. You can simply take the bridge out of the box, power it up, and attach it to a network.

(b) Pricing for bridges is usually more attractive than for routers. A bridge is a good choice in a case where a reduction of costs is needed.

(c) Bridges are network layer-protocol-independent. Nonetheless, bridges can handle multiple protocol with almost no configuration

(d) Some protocols are not routable: you cannot implement routers to connect these networks. However bridge are able to forward non-routable protocols. This remains a compelling reason for implementing bridging capabilities for supporting cetain end-to-end connectivity requirements.

3.1.2 Switching

Switching, otherwise known as LAN switching, is very similar in function to bridging. The key differenc is that switching is done in hadware, or Application Specific Integrated Circuit (ASIC) chips and is extremely fast compared to bridging. The primary purpose of a switch is to increase available bandwidth within a shared-media LAN by implimentting microsegmentation on the local LAN. Since the switch creates point-to-point connections for each packets received, share-media LANs that employ switches become switched-media LANs. Switching uses address in a manner similar to bridging. LAN switches read the destination MAC address on incoming data-link layer frames, and quickly build a switched connection to the switched LAN segment which contains multiple workstations can discriminate between traffic between locally attached workstations and traffic which must be switched to another LAN switch port.

Switches work best when traffic does not have to leave the LAN segments linked to a particular LAN switch. In other words, to minimise the use of expensive WAN links or filter the traffic allowed onto high-speed backbone networks, layer 3 protocol need to be examined by a router. In some cases, this routing functionality is being incorporated into the LAN switch. Basic LAN swtches are layer 2 devices which must be complemented by either external layer 3 routers or internal layer 3 routing functionality.

3.1.3 Routing

Although both routing and bridging examine and forward data packets discriminately, they differ significantly in several key functional areas.

Unlike the bridge, which merely allows access to the internetwork (forwad-if-not-local logic), the router specifically address the data packet to a distant router. However, before it releases a data packet onto the internetwork it confirm the existence of the destination address. Only when the destination address and the quality of the intended path, does it release the carefully packed data packet. This meticulous processing activity is known as forward-if-proven-remote logic. Figure 1.1 illustrates router's use of data-link and network layer addresses.

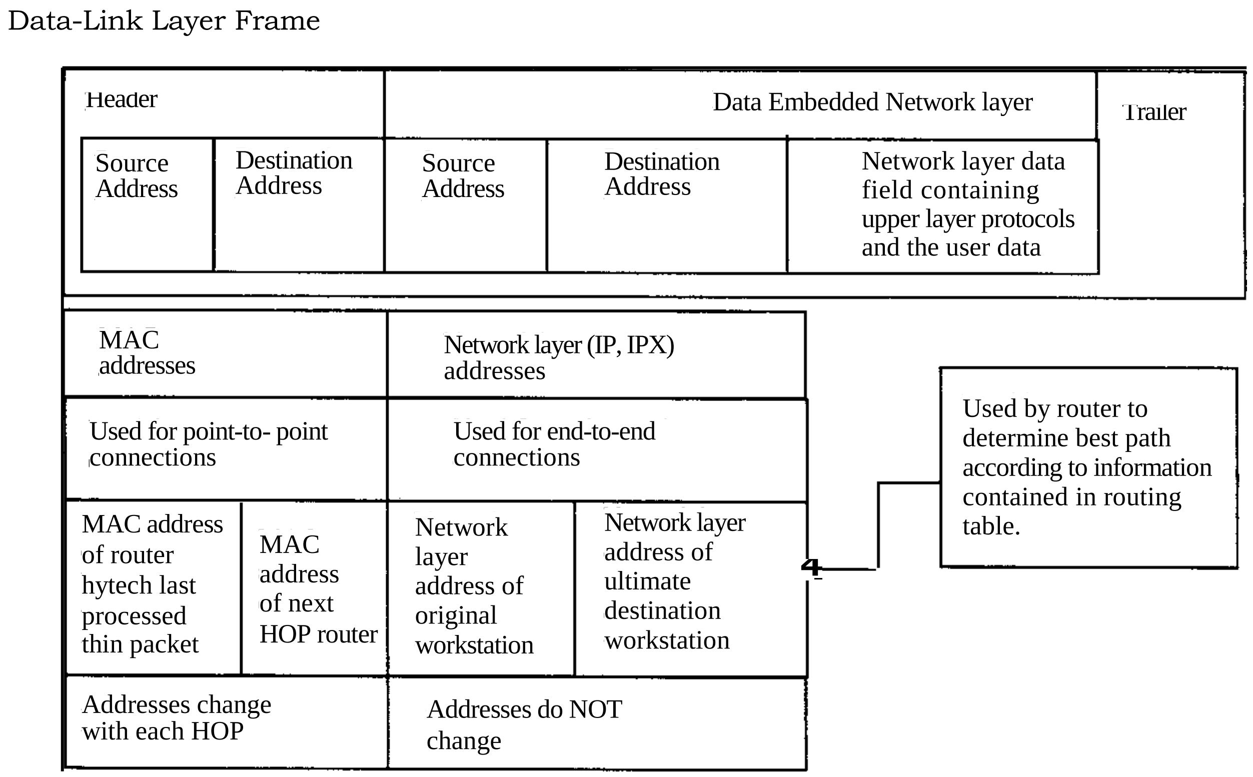

Data-Link Layer Frame
Header
Data Embedded Network layer
Trailer
Source Address
Destination Address
Source Address
Destination Address
Network layer data field containing upper layer protocols and the user data
MAC addresses
Network layer (IP, IPX) addresses
Used for point-to- point connections
Used for end-to-end connections
MAC address of router hytech last processed thin packet
MAC address of next HOP router
Network layer address of original workstation
Network layer address of ultimate destination workstation
Addresses change with each HOP
Addresses do NOT change
Used by router to determine best path according to information contained in routing table.

Figure 1.1 Router S use of Data-Link and Network Layer Addresses

Compared with bridging, routing makes more efficient use of bandwidth on large networks containing redundant paths. The effective use of a network's redundand paths allows routers to perform load balancing of total netwok traffic across two or more links between two given locations. Routers choice of the "best path" can be determined by a variety of factors, including number of hops, trasmission cost, and current line congestion. Router dynamically maintain routing tables, adjusting performance to changing network conditions.

When LANs are connected over a long disance via WAN links, router are more likely than bridges to be employed to intrface to the WAN link. Thanks to the router's ability to more accurately identity upper layer protocols, unnecessary or unwanted traffic can be kept off the relatively low-speed high-cost WAN links.

3.2 Integrated Solution

Since routers, switches and bridges are designed for different purposes, it is common to develop a network with a combination of these two technologies in order to fulfil requirements. Because of the simplicity of bridging, bridges may be employed in remote site LANs. Routers may be relied on to provide a reliable, self-healing backbone, as well as a barrier against in advertent broadcast storms in the networks. In fact, bridging is becoming less common. Most network designers now use switching instead of bridging, in most ports of their designs, and use routing as a supplement to switching. Switching can perform the jobs done by bridging and at the same time provided a higher bandwidth for each note.

3.3 Virtual Local Area Networks (VLANs)

The logical network design known as a virtual LAN depends on a physical device, the LAN swich, for its functionality. Though the original LAN switches delivered abundant bandwidth to locally attached workstation and segments, they lacked the ability to partition the switch into multiple users into corresponding separate workgroups.

VLANs are software definable through configuration software contained within the LAN switch. The use of vitual LANs allows workgroup members to be assigned to more than one workgroup quickly and easily if necessary. Subsequently, each virtual wokgroup is assigned some portion of the LAN switch backbone capacity. LAN switches which support virtual LANs use OSI layer 2 bridging functionality to logically segment the traffic within the switch into distinct virtual LANS.

Any message received by a LAN switch destined for a single workstation is delivered to that destination workstation vis an individual switched network connections. The key difference between a LAN switch which does not support virtual LANs and one that does is how it treats broadcast and multicast messages. In a virtual LAN, broadcast and mu lticast are limited to the members of that virtual LAN only, rather than to all connected devices. This prevents propagation of data across the entire network and reduces network traffic. Simply, VLANs are nothing more than logically defined broadcast / multicast groups within layer 2 LAN switches, since point-to-point traffic is handled by switched dedicated connections.

A key limitation to virtual LANs is that when members of the same virtual LAN are physically connected to separate LAN switches, the virtual LAN configuration information must be shared among multiple LAN switches. Currently, no interoperabi I ity standards exist for transmitting or sharing virtual LAN information between layer 2 LAN switches. As a result, only proprietary switch-to-switch protocols between a single vendor's equipment is possible for multi-switch virtual LANs.

IEEE 802.10 is one possibility for standardising switch-to-switch communication to support virtual LANs which span multiple switches. Originally conceived as a standard for secure data exchange on LANs which would allow workstations to set encryption and authentication settings, this standard is of interest to virtual LAN switch vendors to existing MAC sublayer frames. Instead of just holding security information, this additional 32-bit header could hold virtual LAN identifiers. To overcome the limitations on maximum data-link layer frame length, IEEE 802.10 also included specification for segmentation and reassembly of any frames which should exceed maximum length with the addition of the 32-bit header.

3.4 Backbone Strategy

The word 'backbone' is very often used to descibe the part of the network that interconnects the other parts of the network. For exampe, an organisation may hay an FDDI ring that interconnects a number of Ethernet networks. The FDDI ring is then called the network's backbone. Besides FDDI, thick Ethernet was also commonly used as the network backbone in the past. Figure 1.2 showns an FDDI backbone connecting the LANs in each flour of a build' ing. In this case, the FDDI ring has to be built across floors, providing an access point to each LAN. Since the connections of the LANs are distributed throughout the backbone, this kind of backbone is referred to as a distribute backbone. This name is also useful in distinguishing if from a collapsed backbone which provides network connections in a sinngle central location. In reality, it is rare to build an FDDI backbone within a building because of the high cost. A more cost effective design uses an FDDI backbone to conect LANs in defferent buildings. Indeed, distributed backbones are seldom used in network implementatins nowadays. Collapsed backbones are popularly used instead due to their advantages.

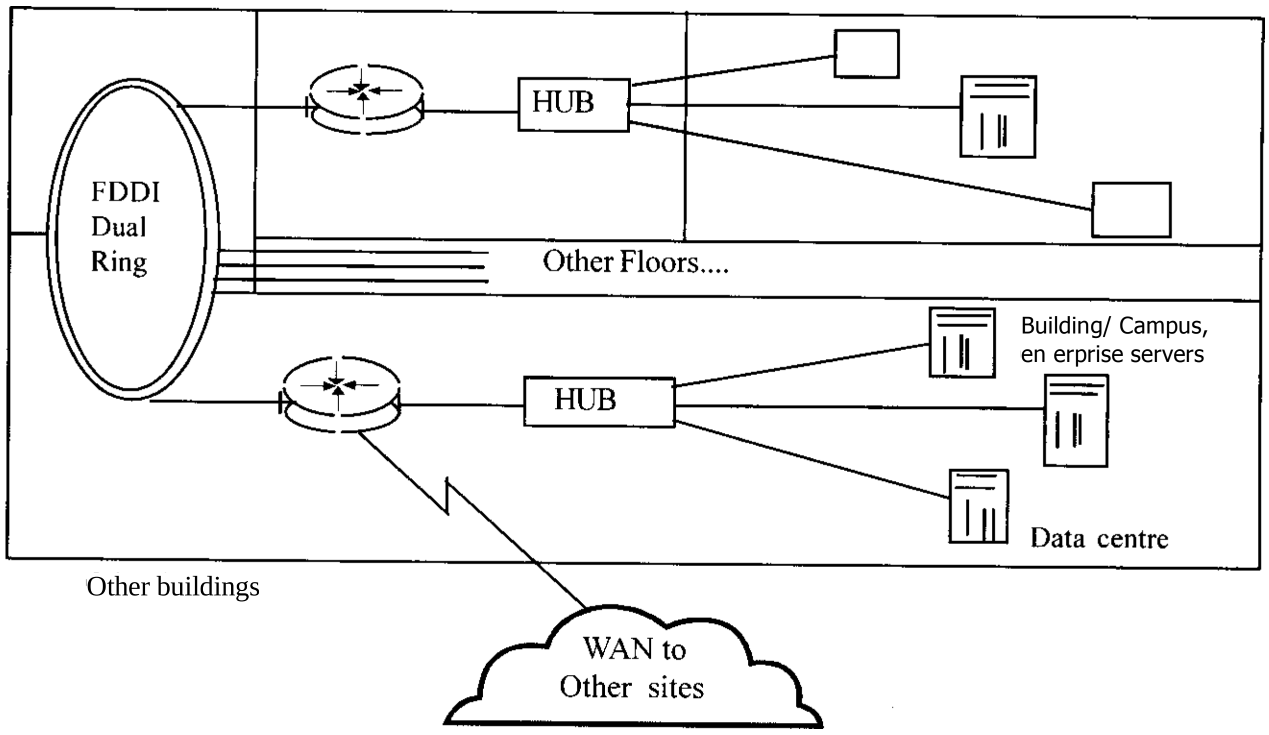


Figure 1.2 A distributed backbone in a building

3.5 Backbone Routing Options

Nowadays, organisations commonly have LANs using different network protocols. There may be Mach intosh LANs running Appletalk, UNIX LANs running TCP/IP, Novell LANs running IPX/SPX, and other. In designing a backbone to interconnect all these LANs, you have to make a deicision on the routing options of the backbone. You need to choose between a multiprotocol backbone and a single protocol backbone.

A typical example of using a multiprotocol backbone is the combination of IPX networks and TCP/IP networks. The network layer for both protocols are different. In a multiprotocol backbone, the IP packets and the TCP/ IP packets can be routed throughout a common backbone without encapsulation. The enviroment is referred to as a multiprotocol backbone (or multiprotocol routing backbone). Encapsulation here means pultting IPX packets as data frames into TCP/IP packets or vise-vesa. A multiprotocol backbone enviroment can adopt one of the two routing strategies, or both, depending on the routed protocol involved. Obviously, the routing speed is faster withut having the encapsulation process.

In a multiprotocol backbone, you may expect the mixing of routers that support different combinations of multiple protocols existing in the network. A drawback of th is is the creation of confusing situation particular)· for integrated routing. In general integrated routing is easier to manage if all the routers attached to the integrated rou ting scheme. It is a difficult task to debug the network connectivity with multiprotocols.

The design of routing can be significantly simplified for a single-protocol backbone. All routers are assumed to support a single routing protocol for a single network protocol. In this kind of routing environment all other routing protocols are ignored. Although it is a single-protocol backbone structure, it is still allowed to support multiprotocol communication.

3.6 Model Components

A three-layer hierarchical mode is generally applicable for most netwoks, from small office LANs to corporate networks. This layering actually focuses on the hierarchy of the routers as it is the main device form connecting the networks. The layers include core, distribution and access. The Figure 1.3 clearly shows that there are three layers of routers connecting to form a hierarchical network.

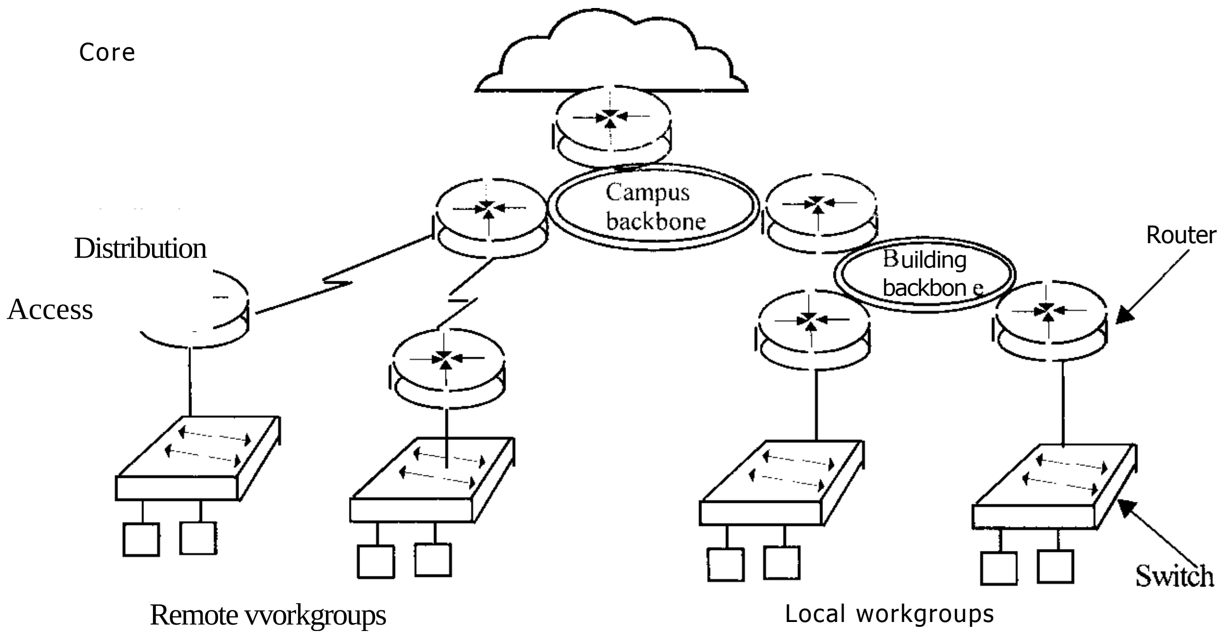


Figure 1.3 Three-layer hierarchical model

The core layer provides the connections between remote sites and acts as the gateway between the internal and external networks. Core links are usually WAN connections like X.25 and frame relay, which are typically rented from telecommunication companies. There are rarely any host directly connected to the routers in this layer. Since the main gateway is implemented in this layer, it is usually a place to install the firewall system. The firewall system is to protect the internal networking from being lacked or cracked externally.

The distribution layer is the layer that provides the connections to the various segments of the entire network.

Nodes are rarely connected at this layer also. The access layer, from its name, provides the actual link for the end users. It is at this layer that nodes are connected to the switches for the user to have access to both the internal and the external networks.

What you have learned in this unit focuses on the issues in designning a network infrastructure. You also learned about the importance of bridge, switch, and routers in internetworking. The issue of backbone strategy was looked into too.

0 Sumniaiy

The unit apparently discussed some vital issues colleen ing the design of network infrastructure. The units that follow shall build upon these issues.

6.0 Tutor Marki

(a) Explain the important of routing, bridging and switching in network design
(b) What do yo understand by the term 'Virtual Local Area Network'?

Excercise 1.1

Discuss the types of backbone that you know

Excercise 1.2

What rooting option are available in network backbone

Paloner, M. J. and Sinclair, R. B. *Advanced Networking Concepts.* Cambridge, Massachusetts USA: Course Technology, 1997.

Keen, P. G. W. and Cunimins, J. M. *Networks in Action: Business Choices and Telecommunication Decisions*, *Belmont*, Clifornia, USA: Wadsworth Publishing Company, 1994.

Goldmar, J. E. *Local Area Networks - A client/Server Approach,* New York: John Wiley & Sons. Inc, 1997.

Online Materials

http://www.rad.com/networks/1997/ipbuild/index.htm

Module 2: Internetworking, Network Design and Management

Unit 8: Network Implementation

Page

What you wil learn in this unit borders on the importance network operating system and its role in network infrastructures for both client and server machines. In this unit, we will use the Windows NT 4.0 as an examples due to its simplicity and popularity. Its features, as it affects the network infrastructure, will be looked into. Let us now see what you will learn in this unit as stated in the unit objectives below.

By the end of this unit, you should be able to:

understand the importance of the NOS in network implementation
undertand the concept of domains in netwrk planning
understand how to carry out some basic network functions

In addition to the network infrastructure, the network operating systems (NOSs) for both client and server machines play an important role in the network design. Under the scope of an NOS, there are also various issues concerning network design, such as the roles of the servers, and the grouping of servers. The level of these concerns is different from the issues discussed on the previous unit. They are closer to the users, or we may say closer to the implmentation level. These concerns are different for different NOSs. In this unit, we take Windows NT 4.0 as an examples because of its popularity. We will assume that you still remember the features of Windows NT, introduce in Unit5?
There are quite a lot online materials in this unit. Some of them ae very useful in helping you understand the architecture of an NT network. However, some of them only present the procedures for doing a particular confirmation task in NT.

3.1 Planning

In planning the implementation of an NT network, it is important to understand the concept of domains. This concept only applies to NT networking. A domain is a logical grouping of users, servers and resources under a single centrlistion administration. There are different domian models which are describe in the online materials. You will find later in this unit that it is not a must to set up a domain in NT networking, you may establish a workgroup instead. However, for implementation of enterprises networks, setting up of domains is important for networking management maintenance, trouble shooting and future expansion. Besides choosing a domain model, you have to make decisions about several design options. These include:

(a). the netwok protocol to be used
(b) the hardware requirement for the servers
(c) netwok service to be implemented e.g WINS, DHCP
(d) administration tool kits to be used
(e) interoperating with machines using other NOSs like UMX and NetWare.

The online material provide clear guidelines on all these topics for you to make the right choices.
The is the need also to plan for the growth in the network. That is, in addition to estimating traffic that will be caned by a network, it is equally important to consider network growth. Doing so permits you to determine if a netwok structure will be able to accomodate a building in workstation wage of the network and or increse in network users over a period of time.

3.2 Network Protocols

In unit 5, we provide a detailed comparison of the three commonly used protocols: NetBEU1, Novell IPX/ SPX and TCP/IP that are available in Windows NT. TCP/IP is recommended for its stronger abilities in routabilty, scalabilty, interoperability and so on. You may enables all the protocols in an NT network. However, the network will suffer from heavy traffic in return. If a server wants to send out a broadcast message to the network, the message will be sent out one time for each of the protocols enabled. That means three times for three network protocols enabled. This imposes a heavy load on network traffic

3.3 TCP/IP Properties

As TCP/IP is always enabled in NT networks, you need to know about the configurrtions of various TCP/IP properties. You learned in Unit 5 and so should be faimiliar with the concepts of IP addressing and host names.

After setting up your first NT server, you have to perform source configuration before going on to the installation of the client machines. You may first see what administrative tools are available in windows NT. These tools are not only useful in network configuration discussed in this unit, but also provides various functions in resources management network monitoring and troubleshooting which one discussed in the later units.

3.4 Client Machine Configuration

You should be aware that windows 95 is freguently used as the client machine in the NT network. Windows 95 is a very popular deskstop operating system on the PC platform. It was developed by microsoft to take place of the outdated operating system DOS. Nowadays, PCs are often installed with Windows 95, even on machines where no networking abilities are required. Besides Windows 95, Windows NT Servers, Windows NT workstations, UNIX, LAN Manager clients and NetWare clients may be NT clients.

LAN Manager is an NOS for PC LANs developed by Microsoft in the old days. It runs on DOS and was later replaced by Windows for Workgroup, which indeed is a graphical version of LAN Manager. A LAN Manager client is used as an NT client for machines installed with DOS in the network. For UNIX as an NT client, TCP/IP must be enabled as one of the network protocols. For Netware clients as an NT client, IPX/SPX must be enabled as one of the network protocols and network services like FILE and PRINT Services for NET WARE (FPNW) have to be enabled.

As Windows as is the most common client of an NT Network, it is useful to know about the detailed steps in setting up such client. The online readings will assist greatly in this regard. You only need to configure the network properties of Windows 95, through the network icon in the Control Panel.

3.5 Connectivity

Have you successfully set up your own NT server? In this section we will investigate the connection of machines in an NT network. You have already come across the concepts of domain and various domain model in the previous unit. The issue of trust is very important in communication between domains, there should be authentication at the machine level to know whether a domain is a trusted one or not. A one way trust relationship may be built between domains A and B, with domain B being the trusting domain and domain A the trusted domain. Be aware that in this case, users in domian B cannot access the resources in domain A. If th is is also wanted domain A has to trust domain B and then a two-way trust relationship is built between these two domains.

Users in a domain can access the resources of its domain and those in the trusting domains. However, in Ike next unit on resource management you will find that a trust relationship is not sufficient for users in one domain to use resources in other domains, it is only a basic requirement. After establishing a trust relationship, the administrator in the trusting domain has to give the corresponding users or groups in the twisted domain the appropriate permission of a resource before they can use it.

Frequently, there is more than one NOS in your network. You may have NT and Novell servers and their clients in the same network. The interoperability between different NOSs is vital for smooth network operations and managment. For a client to access different servers it needs redirector programmes. A redirector programme handles packets sent in the network protocol that it can understand. For example, if a Windows NT client wants to access the resourcess on a Novell server, it must have the redirector for Novell server loader. This client is already loaded with a Microsoft redirector for accessing and Window NT Server since it is an NT client.

3.6 Account Types

Let us now quickly look at the various types of accounts that are available in NT network enviroment.

3.6.1 Built in Accounts

There are two built-in accounts: *Guest* and *Administrator*. The Guest account is used in order to enable occasional users to log on to a server locally for short time. The Guest account is disabled by default. The Administrator account is the most powerful user account for managing the overall network configurations and operations.
It is also important for you to know where the accounts are created. In each NT server or NT workstation is a directory database for storing the user account information. A distinction is made between Domain and Local accounts, depending on the location where the account information is stored.

3.6.2 Domain User Accounts

Domain user accounts are created through the utility programs, user Manager for Domains. The information of the domain account is stored in the Directory Database, called the Master Directory Database. A copy of the database is stored in each of the domains. In domain models, domain user accounts are given to users. Each user can log on to the domain using a single account from any computer in the network. The single user account is an important goal in designing NT enterprise networks.

3.6.3 Local User Accounts

Local users are created on an NT Member server or an NT workstation through their own utility program, user Manager. The information is stored in their own Directory Database. If a user has a local user account of computer *A*, the user can only log on to Computer *A*. Local accounts are created in a workgroup model in which no centralised administration is available.

3.7 Permissions

For ease of management, you rarely assign permissions of the resources directly to a user account. Users are usually contained in groups, and permissions are assigned to groups. Permissions are associated with the resources. Files and printers are typical resources in the network. Different permissions are available for different file system. Do you remember which two file systems are available in Windows NT? They are NTFS and FAT. NTFS is more frequently used for better security and resources arrangement.
Resource sharing is a feature of NT networking. Sharing allows users to access a resource across the network. If a folder is shared, users can connect to the folder and access the files over the network.

3.8 Audit Trail

The auditing of all the activities that goes on in a network is very vital to the Network Administrator. The activities are monitored under the following points:

(a) A description of the action performed.
(b) The user who performed the action.
(c) The date and time of the action.

Table 1.1 shows the event you may audit. You have to choose to audit the event on successful, failure or both.

Table 1.1 Events For Auditing in Windows NT

Event tye	Description
Logan or log off	A user logs on or off a server in the domain. Or, a user breaks the connection to a serve, without normal log off procedures.
File and object Access	A user accesses a folder, file or printer that is set for auditing.
Use of User Rights	A user exercises a right (except log on or log off)
User and Group Management	A user account or group is created, modified or deleted.
Security Policy Changes	A change is made to the rights, audit,or trust relationships policies
Restard, Shutdown, and system	A user rested or shutdown the computer, or an event has secured that affects system security or the security log.
Process Tracking	Detailed tracking information for various events, such as programme activation.

3.9 Monitoring

You may view the result in the security log through the Event Viewer. To audit specific actions to individual files or directories, you have to further specify the audit policy for that particular file or directory through Windows NT Explorer . You may choose to audit actions such as Read, Write, Execute, and Delete on the files and directories. To audit a printer, you have to further specify the audit policy to that particular printer. In addition to auditing, you may monitor the current status of the network through the Server Manager. It gives you infrmation on the users currently connected to the network, the resources shared and those in use. To monitor system resources like the procesor, memory and storage, you may use the parfomance Monitor. You may turn on the counters for one or more behaviours of a system resource. **The** performance Monitor can then give you reports or chats on the statistics gathered. You may even set an alert for a particular behaviour. The performance Monitor will notify you when the counter on that behaviour exceeds a threshold value. The statistics gathered are very helpful in locating bottlenecks in your network, leading to improvements in the network.

For ease of management, you may set up profiles or login scripts for the user accounts and groups. System policies may als be implemented to control computer settings and user cnfigurations.

3.9.1 Event Viewer

The Event Viewer was mentioned in the previous section as a tool for the perforrynce monitoring of an NT server. Here it is introduced as a tool for trobleshooting . The Event Viewer shows your records of event generated by one of the following sources: NT system, the NT services, applications running on an NT server and audited user accounts. You may use the Event Viewer to isolate problems.

3.9.2 Using Last Known Good Configuration

When you log on to an NT server successfully, it will store the current configuration as the Last Known Good Configuration. NT will try to load the Last Known Good Configuration on booting when the system is experiencing an error on loading a device driver or on a user's request. The Last Known Good Configuration is used for incorrect configurations. It cannot solve problems arising from hardware damage or corrupted drivers or files.

3.9.3 System Recovery

When a sever error, also known as a STOP error, occurs on an NT server, the system by default unite an event to the system log; alerts administrators; dumps system memory to a file to use for debugging; and then automatically reboots the server.

However, you may configure the recovery options on a STOP error to suit your particular needs. You may choose to turn on or off the following options; a in Table 1.2.
It is advisable to turn to all the optionsto ensure minimum recovery time; and it is easier to provide disgnostic information to outside support.

Table 1.2 Recovery Options using NT

Options	**It turned on**
Write and event to the system log	Windows NT server writes an event containing information about the error to the system log when a STOP error occurs.
Send an administrature alert	Windows NT server sends an alert to administrators when a STOP error occurs.
Write debugging information	Window NT server writes the contents of system memory to a log file when a STOP error occurs.
Automatic reboot	Computer reboots automatically after a STOP error.

3.10 Emergency Repair Process

The NT server provides the Emergency Repair Process for system recovery. However, you must have the following before you can use it:

(a) The original installation disks and CD-ROM, in case files are detected as missing or corrupted.
(b) The Emergency Repair disk, which is computer - specific. An Emergency Repair disk created for computer will not work on the other ones

In this unit, you have learned about the importance of operating system and its role in network infrastructure. You also learned about the configuration of servers and clients in network implementation. The issue of network accounts, monitoring, and planning were discussed too.

What you have learned in this unit focuses on network planning and implemention. The importance of NOS and its role in accounts management, monitoring and recovery procedures were discussed. The units that follow shall build upon this.

(a) Discuss the types of accounts that you know in networks
(b) What is the importance of audit trail in networking?

Excercise 1.1

Discuss the types of monitoring that you know in a network

Excercse 1.2

Discuss the Emergency Repair Process

Goldman, J. E. *Local Area Networks - A Client / Server Approach, New* York: John Wiley & Sons, Inc., 1997.

Held, G. LAN *Performance - Issues and Answers,* (2nd edu.) New York: John Wiley & Sons, Inc., 1996.

Online Materials

http://www.cif.ac.uz/smac/winnt/ptz_o.htm
http://www.coted.umn.edu/win NT/Cook Book/Win95Client/TCPIP/Default.html
http://web66/coted.umn.edu/winnt/cookbook/win95client/MSClient/Default.html
http://web66/coted.umn.edu/winNT/cookbook/win95client/identification/Default.html

Module 2: Internetworking, Network Design and Management

Unit 9: Network Maintenance

Table of Contents

In this unit, you will learn about the basics of network maintenance and network management. You will also learn about the five functional areas of network management system, and the components of network management. Let us now see what you will learn in this unit as stated in the unit objectives below.

By the end of this unit, you should be able to:

understand the importance of network maintenance and management.
explain the functional area of network management.
discus the major components of a network management system.

Even after setting up a network, your job has not ended. You need to perform network maintenance tasks. The primary objectives for network maintenance are to:

Keep the network running smothly and effectively
Look for potential problem and prevent them from happening

To meet these objectives, you should have a good understaning of network management and monitoing. However, some palanning on the following ascepts may also help you a lot in doing the tasks.

(a) Backups

A reliable backup system should be planned. This should be included in the initial design of your network. It may impose duplicate costs to the network, but your organisation may loose more in any case of data loss that occurs.

(b) Standardisation

It is important to keep everything on the network, including hardware and software components, operation procedures, configuration and file formats as uniform as possible. This makes network management upgrading and troubleshooting much easier.

(c) Documentation

Documenttion is extremely important for network maintenance and trouleshooting. It should include:

A map of the whole network which includes the details of the hardware components, addressing, cabling;
Server information, including the applications run and the data stored on each server and their backup pains;
A record of all past problems, including descriptions, solutions, dates and procedures.

3.1 Network Management

Network management refers to how computer networks monitor and manage active function including the following, known as the functional areas.

- Fault management detects, displays, and maintains records of alarm conditions.
- Configuration management handles additions, deletions, and changes to the network.
- Accounting management tracks the measurable use and cost of network resources.
- Performance management is designed to optimise the speed of the network and to ensure effective utilisation of the network resources.

Security management deals with such issues as logging on and protection of the network from external attack from intruters or unauthorised users (Lockers) and crackers.

If you have a simple network with only three devices, then it might be just as easy to manage each device individually. But if your network is larger and more complex, with dozens (or even hundreds!) of devices that are logically grouped according to class, then a management tool might make sense.

3.2 Network Managment System

Network management tools often require huge workstations to run their software, requiring hundreds of MB (or even GB) of RAM. That worstation is called the Network Management Station (NMS). The most common type of NMS issue is Hewlett Packard's OpenView. At the most basic level, the NMS determines whether or not objects on the LAN are "alive". If an object "dies", you will want to know about it right away as well. If a brand new objects appears, you will want to know about it right away as well. The NMS consists of two parts: the *management* entity and the *management agent* A management agent is installed on each managed device, which may be a computer or a network device like a hub and router. The management agents monitor network traffic and behaviour on the managed device and gather statistics. The data is stored in its management database. The management entity regularly sends requests to the agents, or user initiation. The agent responds to the request database to the management entity.

3.3 Key Elements of Network Management System

The key elements of Network Management System are:

(a) **Agent:** A software that performs function, on behalf of its principal. It can be a stationary or mobile agent, but most of the time, they are mobile.

(b) **Management Station/Manager:** These are the nodes and devices (or stand alone devices) that are attached to the network and they are to be managed.

(c) **Management Infrmation Base:** As we said in the last sectiom, the Management Information Base (NIB) serves as the database or repository for information and data collected along the network.

(d) **Network Management Protocol:** The management staiton and agents are linked by a network mangement protocol e.g. Simple Network Management Protocol(SNMP), Common Management Information Protocol (CMIP) etc.

3.4 Network Management Tools

There are several network management tools available that provide a range of services. For example, they can monitor nodes such as repeaters, routers bridges, hubs, workstaitons, servers, and hosts. If a router is down the software graphically displays this condition ¯id may sound an alarm to the network management station. Some mangement packages have the ability to reboot 'tie router from the NMS. These packages can monitor network traffic and provide analysis of traffic based oi, the collected by intelligent nodes, such as routers and hubs.

Some managemet packages are designed to be modular, so specific functions can be added as your network grows. Modules, can be added that manage domain name services, user accounts, e-mail services need backup and restore services, file server configuration printer sevices and telecommunication services. Figure 1.1 illustrates how these modules are used within a main management module to compose an integrated network management softwre package.

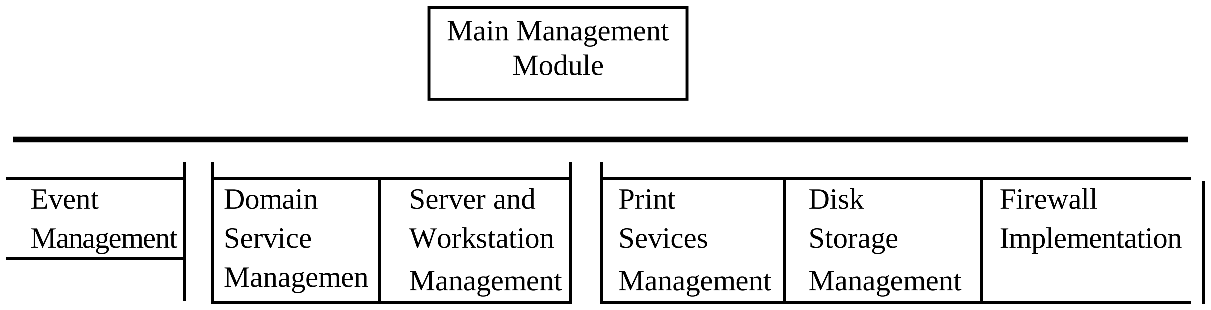


Fig. 1.1. Use of modules within a main management module

3.5 Peripheral Management

OpenView an example of network management package, has modules that enable you to manage key peripherals such as disk services and print services. One such module.
Storage provides client/server hierarchical storage management. OmniStorage spreads data across magnetic, optical, and tape storage facilities. For instance, in most businesses, the current fiscal years accounting data are accessed frequently and would be stored on the accounting server's hard device. When the fiscal year is closed out, that data could be automatically compressed and migrated to an optical disk for future reference. Software like OMNISTORAGE can provide disk capacity management, by automatically storing data where it can be kept most economcally. Data are migrated from expensive magnetic media to less-expensive tapes or CD-ROMs. In the process, critical hard drive space is freed for immediate production use.
The most powerful tool you have as a network manager is your ability to plan. Planning can be performed at all stages of network operation. Today's network manager has to stay current on the business needs of his or her organisation. As you plan the installation of new equipment include a phase for testing the equipment before it is installed and brought "alive" onto the network. This will reduce "downtime" due to problems in equipment preparation such as loading storage and table entry (or entries) on a new router.

In this unit, you have learned about the rudiments of network maintenance. You also learned about network management and its functional areas. The basic tools used in network management were considered and the protocol requirements needed for effective network mangement was explained.

What you have learned in this unit focuses on isues concerning network maintenance and management. We also considered the functional areas of network management and the protocols required to do such. The last unit will build upon this.

(a) What do you understand by the term "Network Management". Discuss its functional areas.

Excercise 1.1

Discuss the Network Management Tools

Excercise 1.2

What do you understand by Peripheral Management?

Palmer, M. J. and Sinclair R. B. *Advanced Networking Concepts* pp. 187-97, (1997)

Online Material

http://www.netman.cit.bufallo.edu/
http://www.simpleweb.org/
http://www.nmf.org/

Module 2: Internetworking. Network Design and Management

Unit 10: Networking Troubleshooting

Table of Contents

In this unit, you will learn about the basic of troubleshooting and how it is carried out. You will also learn about the basic troubleshooting equipemnts used in computer networks. In addition to these, you will see how a network problem or fault can be isolated. Let us now see what you will learn in this unit as started in the unit objectives below.

By the end of this unit, you should be able to:

- explain basic network troubleshooting approaches.
- isolate a network problem.

describe network troubleshooting equipment and how it is used.
discuss how design issues affect network troubleshooting.

Troubleshooting problems on your network offers as much challenge as the initial network design. You may spend hours finding a defective tranceiver or a workstation NIC that intermittently sends a network-wide broadcast storm. A network slowdown might be caused by one of 20 possible sources. Implememting the solution to your network problem often will take only a short time compared to the time you spend isolating it.

This unit gives you several tools for network troubleshooting. These include developing overall troubleshoot-ing approaches that will save you time. Obtaining the right troubleshooting equipment is importanty also. We present information on several types of troubleshooting equipment and the types of situations where it is used. Finally, we discuss the varitey of problems that occur on networks and where to look for solutions.

3.1 Steps to Investigate Network Problems

Let us consider the Table 1.1 below. It shows the logical steps to be followed in order to investigate the isolate network problems.
If the problems is reported by the network user, listen carefully to his or her description. Even if he or she does not use the right tem imology, the information still can be of value. Part of the challenge for you is to ask the right questions so you get as much information as possible.

Table 1.1 Investigating Network Problems

Steps To Investigate Network Problems
Carefully listen to the problem report.
Obtain error message or other details of the problem
Check first for the single solutions.
Determine if others are expriencing the problem
check to see what your management software indicates
Check for any power outages
Keep a log of reported problems

An obvious, but sometimes overlooked step is to record the error message at the time it appears. If you try to recall the message from memory, you may loose some important information. For example, the error 'Network' not responding, can head you to a different set of troubleshooting steps than the message. "Network timeout error". These messages are similar and might be confused through recollection. The first message could mean a damaged NIC. The second message could mean that your database server is down and the

application is waiting to obtain data.
Another important step is to start with simple solutions. A printer may not be printing because the print server next to it is unplugged from the power or cable is off. A repeater may be done because it is kept in a janitor's closet and the janitor has mistakenly unplugged the power. A new office employee may take a segment down by removing the cable from the wall connector in order to rearrange the office furniture. Or the new employee may use the wrong IP address as he or she configures the workstation software.
Determine if anyone else is experiencing the problem. For example, several people may report they cannot load a word processing software package.

This may be due to a problem at the server they use to load the software. If only one person is experiencing this problem, however, it may point to trouble on one workstation.
If you have event monitoring on your network management software, check to see if any recent triggered events have been logged. Also check to determine if any devices are in a caution or warning status.
Power interruptions are a common source of network difficulties. Sometimes a bridge, router, hub, or other piece of equipment fails to boot properly following a power outage. Some power irregularities, such as a power sag or a small surge, go unnoticed. But the sag or surge may impact one or more network devices, even though no other equipment seems affected. If you manage a large campus-wide metwork, a localized power outage may take down only part of your network, leaving those who still have power perhaps (including the network manager) unaware there is a problem.

3.2 Troubleshooting Equipment

Several tools that are commonly used in network troubleshooting are discussed in the following sub-sections.

(a) Time - Domain Reflectometers (TDRs) and Optical Time - Domain Reflectometer (OTDRs)

A TDR can quickly locate open and short circuits, sharp bends and imperfections in twisted pair and coaxial cables. It sends a signal along the cable. The defects on the cable reflect the signal to the TDR, at different amplitutes depending on the problem. The TDR calculates distance to a defect by measuring the time the signal travels. It is commonly used in network troubleshooting as well as in network installation. OTDR is used to locate breaks, measure the attenuation and splice or connector losses in fibre optic cable.

(b) Cable Testers (Scanners)

If you suspect the cables cause the network problem, you may check them with a cable tester. Besides the physical connectivity, a cable tester can test and report on cable conditions like near-end crosstalk (NEXT), attention and noise. A cable tester also performs the functions of a TDR. They are available for tested pair and causal cable. There is also similar testing equipment available for fibre optic cable.

(c) Network Monitor

Network monitors track packets crossing a network. They gather information about packet sizes and types, error packets, and overall utilisation and other information. But they do not decode the packet. The information gathered is useful for creating profiles for network traffic, planning for network expansion, determining intruders, establishing a baseline and distributing traffic more efficiently.

(d) Protocol Analysers

A protocol analyzer helps you in network traffic analysis by capturing and decoding the data frames. It presents the protocol layers information recorded in a frame in a readable format. This provides you information for analysing network behaviour such as network bottlenecks, faulty network equipments and protocol problems, for example.

e) Tranceiver Monitor

Tranceiver have low visibility on a network, but they play a critical role. These small devices are part of the attachment unit interface (AU!) for linking backbone cabling into network and computer equipment such as bridges, routers, hubs, andmonitor can detect transceiver problems workstations. A defective tranciever can be hard to identify without the right equipment. The tranceiver related to power, signal reception, and collision handling.

(f) MAU Analyser

A MAU analyzer is used on token ring networks and provides information similar to a cable scanner. It generates a singal for the purpose of locating opens, shorts, and faulty cable conditions. Also, it can determine if the MAU is functioning properly.

3.3 Network Problems

Network cabling is the most common source of problems on a network. Cabling problems have many symptoms, such as disconnecting workstations, slow network services, a high level of packet errors, and unreliable data transmission. If you have reports of any of these problems, one place to start is by investigating the cable plant. There are several things to check related to cabling problems, including the following:

Cable length
Cable Type
Terminators
Grounding
Cable impedance
An open or short
RFI and EMI
Connectors
Distance between connections

Figure 1.1 summarises cable troubleshooting. The following sub-sections discusses the various problems that a cable can experience.

3.3.1 Cable length

According to the IEEE specifications, Ethernet and token ring networks must be able to send a packet to the most distance node in a given amount of time. If a network segment is extended beyond the IEEE specifications, there will be communicatoin problems affecting all nodes on that segment. If you use a protocol analyser or RMN probe to analyse an overly long cable segment, it will show excessive packet collisions, expecially late collisions. A late collision occurs when a note can put a majority of the packet on the cable but experiences a collision toward the end of the transmission of the packet. Another way to detect a segement that is too long is to use a cable scanner or TDR and short the distance of the cable.

3.3.2 Cable Types

In places where users can install their own cabling, you may discover some nonstanding cabling. For example, network managers have found TV coaxial eable and antenna cable used instead of 50-ohm network cable. The labeling on the cable often will indicate it is the wrong type. A multimeter also will quickly show that the impedance is not 50 ohms.

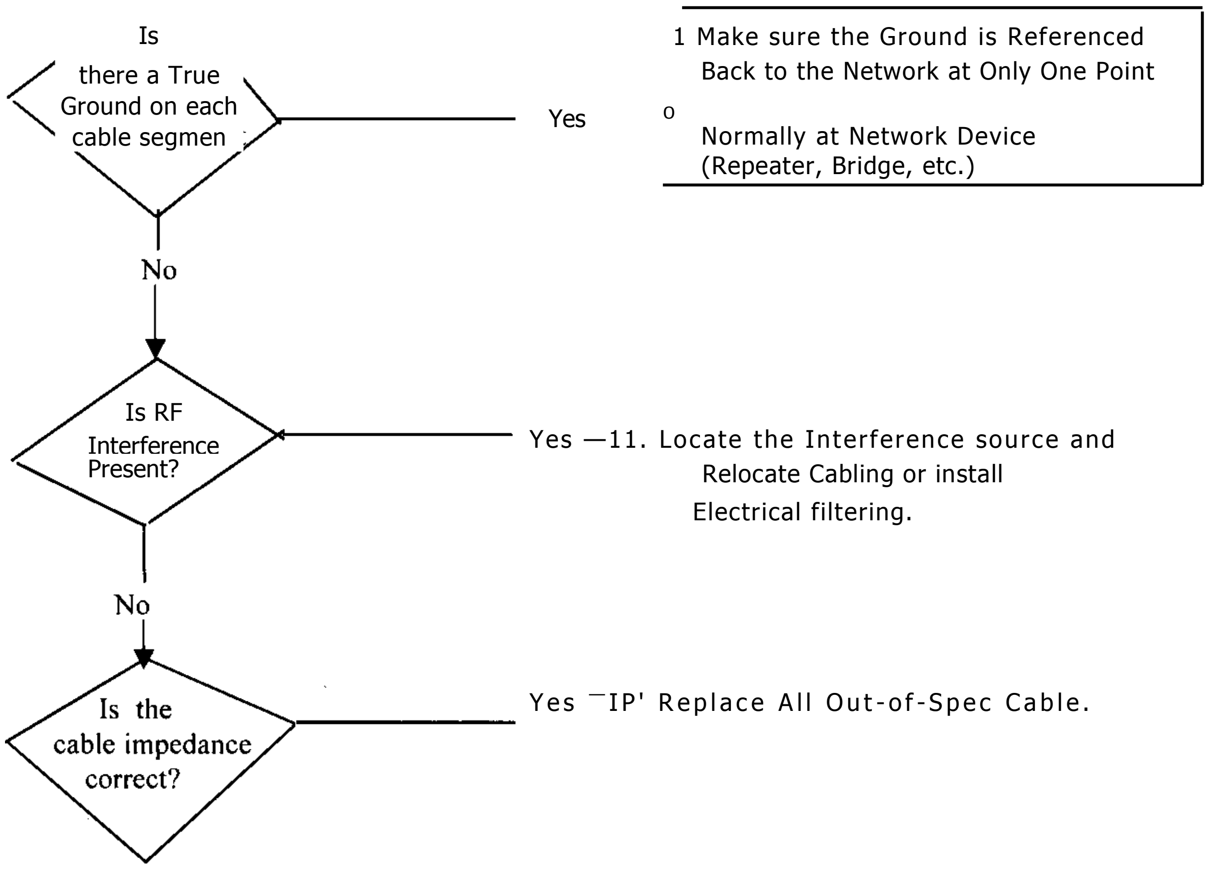

Is there a True Ground on each cable segmen
Yes
1 Make sure the Ground is Referenced Back to the Network at Only One Point
Normally at Network Device (Repeater, Bridge, etc.)
No
Is RF Interference Present?
Yes —11. Locate the Interference source and Relocate Cabling or install Electrical filtering.
No
Is the cable impedance correct?
Yes ¯IP' Replace All Out-of-Spec Cable.

Figure 1.1 Summary of cable troubleshooting.

3.4 Ethernet Troubleshooting

In addition to checking the cable problems, there are several elements to check when an Ethernet network is experiencing difficulty. Check to be sure that all servers and workstations are configured to use the correct frame type for your network.

If the server Ethernet LANS are connected to a WAN, each LAN will have a unique network number. Servers, such as Novell NetWare, are configured to use the network number. Make sure that all servers on the same network use the same network number in the address configuration. Routers also must be configured with the correct network number.

Also on an Ethernet network, when your protocol analyser detects late collisions or excessive CRC errors, check the repeaters, tranceivers, and bridges for problems.

3.5 Token Ring Troubleshooting

Cable problems are a good place to begin when troubleshooting a token ring network. If there are no detectable cabling problems and the entire network is done, check on whether beaconing is occuring. If the active monitor has a NIC that is functioning intermittently, you will need to remove the station from the hub so another active monitor can be designated. Replace the NIC and return the station to the hub. Also, check the standby monitor stations and their NICs to ensure all are functioning.

Another device to check is the MAU. A protocol analyser can help you determine if the MAN is not working. Further, if the cable, connectors, and NICs show no problems, install a different MAU to determine if it is the source. If you attach a new station and problems occur, check to be certain the NIC is properly set for the network speed.

3.6 Troubleshooting Fibre Optic Cable

Fibre optic cable is used for network backbones and for high-speed network segments such as FDDI. Troublesshooting fibre optic cable presents some special problems, because the signal source is light and the medium is glass or plastic (usually glass). Fibre cable is expecially susceptible to damage and should be examined for breaks or opens when problems occurs. An OTDR is used to locate opens or measure power loss on a calbe run. As we mentioned in Unit 4 of Module One, power loss is called attenuation. For example, network problems will result if attenuation is greater than 1.5d 8/km on an FDDI ring.

Fibre Optic Cable connections and terminations must be more precise than for metal cabling and take longer to install. Durable connections and terminations depend on whether the cable installer has the right tools and has been careful in his or her work. Improperly connected or terminated cable will cause problems. Dirty conectors also cause problems on fibre optice cable runs. Examine connections and terminations for dirt or for poor installatin when you have problems on a fibre optic cable.

Another problems area is the angle of the bend around corners. This si called the angular circumference. The maximum angular circumference depends on the cable characteristics and the number of stands. Light will not transmit when th angular circumference is too great.

3.7 Problems of Network Devices

In this section, we will look at the specific problems of some network devices. We will also examine what we can do in terms of troubleshooting and how we can fix those problems.

(a) MC Problems

NIC problems are common and can affect the entire network segment or network equipment such as a repeater or hub. Many NIC vendors include dignostic software to help you determine if the NIC is working properly.

When you set up a NIC or if problems occurs, check to be certain the vendor's network drivers are at the most current version level. Some vendors have updated drivers that can be down loaded from the Internet if you have an immediate need.

A NIC may have problems because it is set up to use the wrong frame type or the wrong cable type. For example, if a NIC does not connect to the network, check the cable type used. Some NICs can accept coaxial of twisted pair cabling. Be certain the NIC setup information matches the cable conected to the NIC. Occassionally, a NIC will malfunction and broadcast continuously. This situation can result in a network slowdown. A protocol analyser will help you trace the problem to the offending NIC. With the protocol analyser, you can obtain the address of the NIC and trace its location. The easiest solution to try with a malfunctioning NIC is to reseat (remove and reinstall) the board in the device having problems.

(b) Print Servers

Print server problems are usually very apparent, because nothing appears at the printer after a print request is made. Print server capability has two components. The first component is at the printer end. The printer may be attached to a stand-alone hardware print server connected to the network, or it may have an internal print server card. Another option is to use a printer attached to an existing PC workstation and to have print server software on the workstation. This option is less frequently used because stand-alone print severs and print server cards are more reliable and competivitely priced.

The second component of print services is the print server software on the file server or host. This software captures a printout and sends it to a printer que that is "attached" (through software) to the print server (remote server) at the printer end. The queue is able to delvier the printout to the remote server because each remote server has a unique address. When printing problems occur, you will need to check the printer's print server and the print server software on the file server or host.

(e) File Server

File sever operating systems can be tuned for network performance. For example, received packets are stored in server communication buffers until the server is ready to process the packet. A bottleneck or slow response will occur if the buffer are too small or if there are too few buffers to match the demand on the server. It may be necessary to rellocate or expand server memory to meet the need. Some vendors offer packages that automatically tune server parameters to meet changing demand. The tuning occurs automatically while the server is functioning.

The server NIC also can create bottlenecks and slow response. As demand increases on a server, it may be necessary to upgrade the NIC or to install additional NICs. Novell servers are easily configured to use several NICs. Scalable servers are an advantage when you anticipate growth in demand. Severs with scalable CPUs can be upgraded when bottlenecks are created by an overloaded CPU. Microsoft NT version 3 supports up to four server CPUs. Microsoft NT version 4 is scalable to eight server CPU.

(d) Gateway

A malfunctioning gateway can have several symptoms. The most apparent symptom is that a node is missing. For example, if an SNA gateway to an IBM mainframe is down, the mainframe is no longer accessible to the network and will appear to be missing. The solution is to reboot the gateway or to replace it. **The** gateway on a Novel system may be a NIC in the server. If a Novell gateway is down, replace the NIC. Malfuntioning gateways also can generate bad packets and high error rates on the network. A protocol analyser or RMON probe can be used to trace the problem to the gateway.

(e) Repeaters

Some repeater problems are relatively simple to troobleshoot. For exasmple, if the network traffic is not going through a repeater, check to determine if a repeater segment is partitioned. Check the segment cable, connections, and NICs for problems. Correct any problems you find and reset the partitioned segment on the repeater. If the entire repeater is down, check the power source and any fuses in the repeater.

Sometimes a malfunctioning repeater will send bad or corrupted packets. This can be determined by placing a protocol analyser on the network and viewing the traffic into and out of the repeater. The easiest solution to this problem is to replace the repeater and return it to the vendor for repair.

Excessive collisions, network slowdown, or a network bottleneck can be caused by an overload repeater. In this situation, you may need to make some design changes such as installing one or more bridges to segment sections of the network and prevent overloading.

(f) Bridge

A defective bridge can generate bad packets, excessive network traffic and network slowdowns. Information about this situation can be bathered from a protocol analyser or from RMON probe data. Check the bridge configuration and power supply. Try resetting the bridge to see this cures the problem. If none of these things work, you may need to replace the bridge.

g) Router Problems

As with other network equipment malfunctioning routers produce bad packets, slow network reponse, and a high rate of collisions. When a router malfunctions, check the routing table and related areas. Check the following potential trouble spots:

- Is the routing software up-to-date?
- Is the power supply working properly?
- Is the router memory working?
- For PC-based routers, is the hard drive full or fragmented?
- For PC-based routers, are the hard drive and controller working?

Thtreporter symptons also may be due to an overloaded router. In this case, it may be necessary to redesign the network or to segment that portion of the network Another solution may be to replace the router with an intelligent hub.

(b) Intelligent Hubs

Many hubs are designed in a modular fashion, so all of the hub may be working except a single board, such as a bridge module. If one module is defective, the hub isolates the problem and shuts it down. The board can then be removed and replaced with no interruption of service to other hub modules. The symptoms of a defective module are similar to those for repeaters, bridges, and routers. These include bad packets, packet encapsulation errors, network errors, slow down network response, and problem reports from users on a particular module. Intelligent hubs can be monitored with the help of software on the hub or software at a network management station.

If there is a generalised failure of a hub, check fuses and the power supply. Also check the hub backplane to be sure it is working. If your network mangement software or protocol analyser detect packet timing errors or excessive collisions, check the retiming module in the hub. Often this is a single board that can be replaced when it malfunctions.

3.8 Further Network Problems

Let us now look at some network problems that are software-based and how they can be troubleshoted and fixed.

(a) Rebooting Network Equipment

Anytime you detect a problem with network equipment, try rebooting it. This technique works in many cases because most network equipment is CPU-based. As can happen on an CPU, some registers may get out of synchronisation, register pointer may be lost or a critial register may be empty. Rebooting the equipment forces the CPU register to reset to a known state, and the equipment may be fully operational again.

(b) IF or Network Address Problems

Information cannot be routed to its proper destination if there are addressing problems. The most common source of addressing problems is IP addressing on TCP/IP networks. For example, network problems can occur if two or more nodes are using the same IP address. This can result in extra network traffic and show or no response at the nodes in conflict. The assignment of IP addresses is often the responsibility of the network manager. You can avoid conflicts by maintaining an IP address database showing the address assigned to each node. IP address conflicts can be traced by means of a protocol analyser or an AMON probe.

(c) Mail Systems

Network traffic and bottlenecks sometimes are caused by mail systems that go awry. For example, in the early versions of a popular mail system, there were instances where a single message was transmitted thousands of times to a single destination. The problem was corrected by a software fix.

Defective mail systems can create severe network problems. Make certain your mail server or servers are set up in the most efficient way and that the software is kept up to date. Another source of mail system problems can develop when there are several mail servers and mail systems.

Addressing and forwarding problems can occur wherein some mail goes into an endless loop between servers. This problem might happen on a college campus where academic users have VAX mail, administrative users have Microsoft Mail or Exchange, and physical plant users have their own server with CC: MAIL.

3.9 Isolating Network Problems

The amount of difficulty you may experience in isolating a network problem is related to the size of your network, its complexity, and the type of network. Small token ring network are relatively easy to troubleshoot because of their star configuration. If a beaconing network cannot resolve its problem without your help, there is usually a downed node. Management software, TDR, or a protocol analyser can be used to locate the node and isolate it for repair. The network continues running after the node is removed.

Problems are more difficult to isolate on an Ethernet network, particularly if they are intemittent. A defective NIC, repeater, bridge, router or other piece of equipment can result in similar network symptoms, such as network slowdowns. Network troubleshooting equipment such as a protocol analyser, is very effective for resolving problems. SNMP - compliant equipment also helps, especially on complex networks, so you can use network management software to isolate problems.

3.10 Eliminating Problems Through Design

Often the best solution to problems is found in network design. If you begin the design with a knowledge of your user and their business, you will be able to avoid bottlenecks and excessive traffic problems from the beginning. For example, a business campus may use ID card readers in the cafeteria and gym areas. The sever for the card readers may be located in the cafeteria's main office. If your network design includes the ability to segment this area from the rest of the network, business functions will not be slowed due to the high traffic from the card readers.

In this unit, you have leaned about the basics of network troubleshooting. You also leaned about the equipment used for troubleshooting. We disclosed the possible problems of some network devices and how they can be fixed. The software problems of a network and solutions were considered too. Note that it is impossible to have a solution that applies to all problems and scenarios. However, there are still general gu id I ines for network troubleshooting.

What you have learned focused on troubleshooting of networks and devices used for such. It has served to introduce you to the various problems of a network (both hadware and software) and how they can be fixed. We now move to Module Three, where we will discuss the issues of network Programming.

(a) Discuss the steps involved when investigating network problems.

(b) Explain the oprations of any three troubleshooting equipment that you know in compouter networks.

(c) What problems can occur or happen to four network devices, and how can they be fixed.

Exercise 1.1

(a) How would you troubleshoot fibre option cable

Exercise 1.2

(a) Discuss how you would isolate a problem on a network.

Palmer, M.J. and sinclair, R.B. *Advanced Networking Concepts,* 1997 PP. 119-208

Online Materials

None

Module 3: Overview of Network Programming

[illegible] Network Programming

Page

In this unit, you will learn about the basics of client-sever architecture and TCP connections in order to refresh yourself. Yo will also learn about the construct of simple programmes in networking, especially sockets. Establishing a connection to a sever, error handling, and to read and display sever's reply will be looked into. Let us now see what you will learn in this unit as specified in the unit objectives below.

By the end of this unit, you should be able to:

- understand the Client-Sever architecture.
- write simple network programmes.
- establish a link to a severs.

Most applications can be divided into two pieces: a *client* and *sever*. We can draw the communication link between the two as shown in Figure 1.1

client
communication link
server

Figure 1.1 Network application: client and server

There are numerous examples of clients and severs that most readers are probably familiar with: a Web browser (a client) commuicating with a Web server; an FTP client fetching a file from an FTP sever; a Telnet client that we use to log in to a remote host through a Telnet sever on that remote host.
Client s normally communicate with one sever at a time, although using the Web browser as an example, we might communicate with many different Web severs over, say, a 10-minute time period. But from the server's perspective at any given point in time it is not usual for a sever to be communicating with multiple clients. We show this in figure 1.2. Later in this text we will cover several different ways for a server to handle multiple clients at the same time.

Although we think oft he client application communicating with the sever application, networking protocols are involved. In this text we focus on the TCP/IP protocol suit, also called the Internet protocol suite. For example, Web clients and servers communciatie using the TCP protocol. TCP, in turn, use the IP protocol, and IP communicates with a datal ink layer of some form. For example, if the client and server are on the same Ethernet we would have the arrangement shown in Figure 1.3.

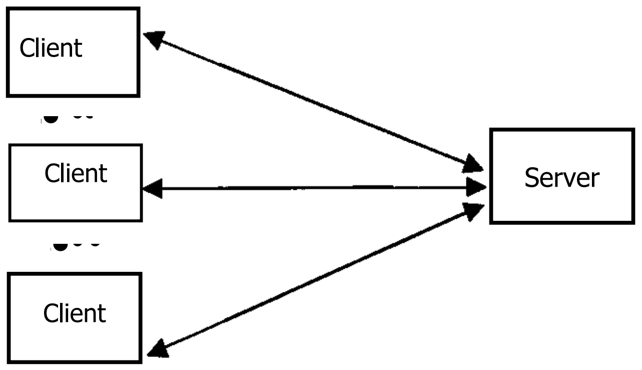


Figure 1.2 Server handling multiple Clients at the same time

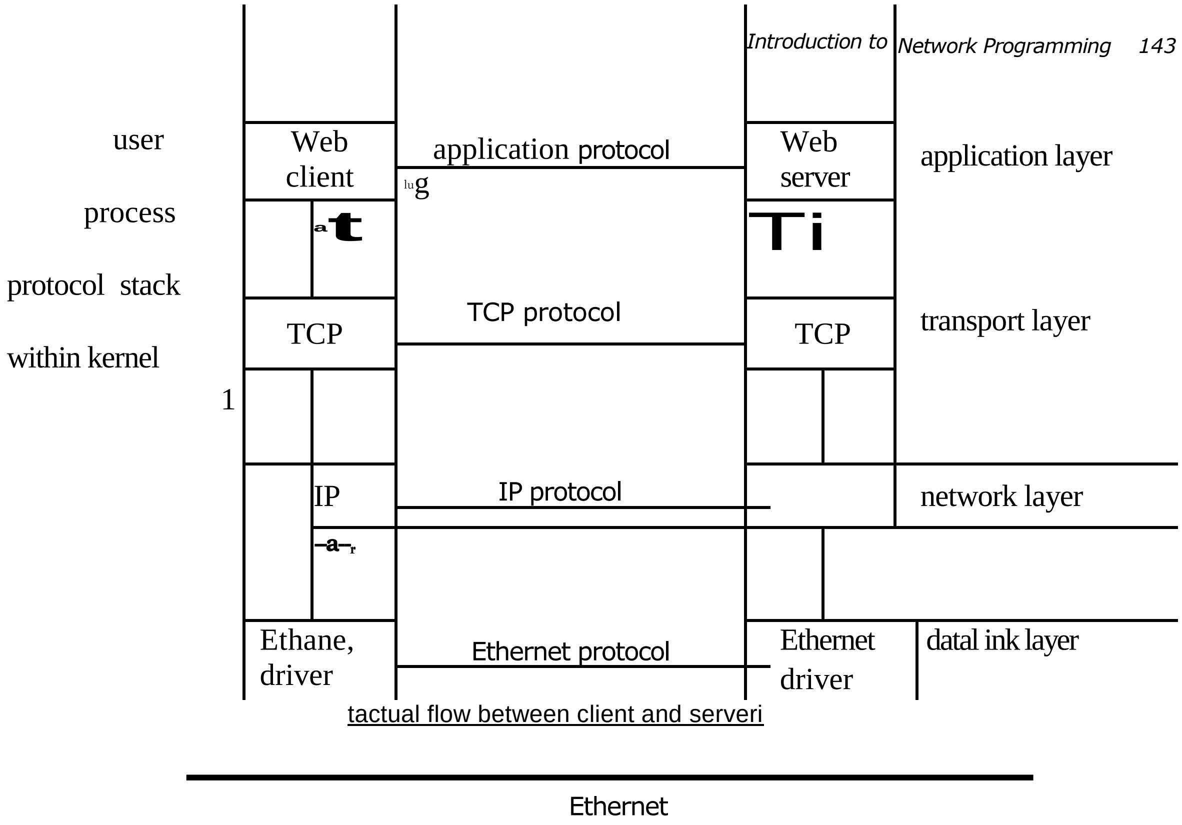


Figure 1.3 Client and server on the same Ethernet communicating using TCP.

Even though the client and server communcate using an application protocol, the transport layer communicate using TCP, and so on, we note that the actual flow of information between the cleint and server goes down the protocol stack on one side, across the network, and up the protocol stack on the other side.
We also note that the client and server are typically user prosesses, while the TCP and IP protocols are normally part of the protocol stack within the kernel. We have lableled the four layers on the right side of Figure 1.3.

TCP and IP are not the only protocols that we discuss. Some clients and servers use the UDP protocol instead of TCP. Furthermore, we have used the term "IP" but the protocol, which has been in use since the early 1980s, is officially called *IF version* 4 (IPv4). A new version, *IF version 6* (IPv6) was developed during the mid-1990s and will probably replace IP v4 in the years to come. Initial implementations of IPv6 were available at the time of this writing, and this text covers the development of network application using both IPv4 and IPv6. Appendix A provids a comparison of IPv4 and IPv6, along with other protocols that we will encounter.

The client and server need not be atached to the *local area network* (LAN) as we show in Figure 1.3. Instead, in Figure 1.4 we show the client and sever on different LANs, with the both LANs connected to a *wide area network* (WAN) using routers.

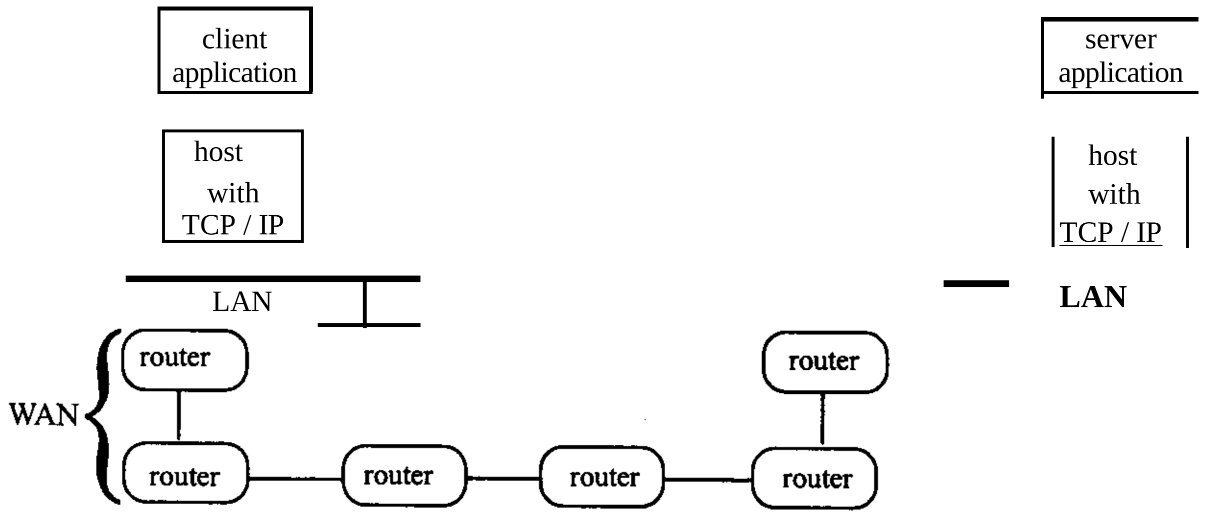


Figure 1.4 Client and server on different LANs connected through a WAN

Routers are the building blocks of WANs. The largest WAN today is the Internet, although many companies build their own WANs and these private WANs may or may not be connected to the Internet.
The remainder of this chapter provides an introduction and overview to the various topics that are covered in detail later in the text. We start with a complete exmaple of a TCP client, albeit a simple one, that demonstrates many of the function calls and concepts that we encounter throughout the text. This cleint works with IP version 4 only, and we show the changes required to work with IP version 6. A better solution is to write protocol-independent cleints and servers. This units also show a complete TCP server that works with our client.

To simplify all the code that we write, we define our own wrapper functions for most of the system funtions that we call throughout the text. We can use these most of the time to check for an error, print an appropriate message, and terminate when an errors occurs. We also show the test network, hosts, and routers used for most examples in the text, along with their hostnames, EP addresses, and operating systems. Most discussions of Unix these days include the term *Posix,* which is the standard that most vendors have adopted. We descrie the history of *Posix* and how it affects the APIs that we describe in this text, along with the other players in the standards area.

3.1 A Simple Daytime Client

Let us consider a specific example to introduce many of the concepts and terms that we will encounter throughout the book. Figure 1.5 is an implementation of a TCP time-of-day client. This cleint establishes a IC connection with a server and the server simply sends back the current time and date in a human- readable format.

--*intro/daytimetcpcli.c*

```
# include "unp.h"
int
main (int   argc,     char     **argv)?
{
    int sockftd, n;
         char recvline (MAXLINE + 1);
```

```
7      struct sockaddr_in servaddr;

8          if (argc ! = 2)
9              err_qu it ("usage: a .out <I Paddress>");

10     if ( (sockfd = socket (AF-INET, SOCK_STREAM, 0) ) < 0)
11.            err_sys ("socket error");

12.        bzero (& servaddr, size of (servaddr);
13.        servaddr. sin_family = AF_INET;
14.        servaddr. sin_port = htons(13); /* daytime server */
15.        if (inet_pton (AF-INET, argv[1], & servaddr.sin_addr
           )< = 0)1
16.            err_quit ("inet_pton error for %s", argv [1] );

17.        if (connect (sockfd, (SA *) &servaddr, sizeof (servaddr) ) < 0)
18.            err_sys ("connect error");

19.        while ( (n = read (sockfd, recvline, MAXLINE)) > 0) {
20.            recvline [n] = 0;        /* null terminate * /
21.            if (fputs(recvline, stdout) == EOF)
22.                err_sys ("fputs error");
23.
24.    if (n < 0)
25.        err_sys ("read error");

26         exit (0);
27
```

-- *intro/daytimecpcli.c*

Figure 1.5 TCP daytime cleint

This is the format that we use for all the source code in the text. Each nonblank line is numbered. The text describing portions of the code begins with the starting and ending line numbers in the left margin, as shown shortly.Sometimes the paragraph is preceded by a short descriptive bold heading, providing a summary statement of the code being described.

The horizontal rules at the begining and end of the code fragment specify the source code filename: the file daytime cpcli.c in the directory intro for this example. Since the source code for all the examples in the text is freely available (see the Preface), thios lets you locate the appropriate source file. Compiling, runing, and especially modifying these programmes while reading this text in an excellent way to learn the concepts of network programming.

Throughout the text we will use indented, parenthethical notes such as this to describe implementation details and historical points.

If we complile the program into the defaul a.out file and execute it, we have the following output

```
solaris % a.out 206.62.226.35                  our input
 Fri Jan 12 14:27:1996                          the program :1s output
```

Whenever we display interactive input and output we show our typed input in a bold font and the computer output like this. *Comments are added on the right side in italics.* We always include the name of the system as part of the shell prompt (solar is this example) to show on which host command was run. The hostnarnes usually describe the operating system.

There are may details to now consider in this 27-line program. We mention them briefly here, in case this is your first encounter with a network programme, and provide more ifnormation on these topic later in the text.

Include our own header

We include our own header, u np.h, which we show in Section D.1. This header includes numerous system headers that are needed by most network programmes and defines various constants that we use (eg., MAXLI NE).

Command-line arguments

This is the definition of the ma in fucntion along with the command-line arguments. We have written the code in this text assuming an ANSI (American National Standard Institute) C compiler.

Create a TCP socket

The socket function creates an Internet (ALI NET) stream (SOCK_STREAM) socket, which is a fancy name to a TCP socket. The functions returns as small integer descriptor that we use to identify the socket in all future function calls (e.g., the calls to connet and read that follow).

The if statement contains a call to the `socket` funtion, an assignment of the return value to the varialbe named sockfd, and then a test of whether this assigned value is less than 0. While we could break this into two C statements,

```
    sockfd = socket (AF_INET, SOCK_STREAM, );
    if (sockfd < 0)
```

it is a common C idiom to combine the two lines. The set of parentheses around the funtion call and assignment are required, given the precedence rules of C (the less-than operator has a higher precedenee than assi ngment). As a personal style issue, the author always places a space between the two opening parentheses, as a visual indicator that the left-hand side of the comparison is also an assignment. (The author firs saw this style in the Minix source code [Tanenbaum 1989] and has copied it ever since) We use this same style in the while statement later in the program.

We will encounter many differet uses of the term `socket.` First, *the aplication programming interface*, or API, that we are using is called the *sockets* API. In the preceding paragraph we refer to a function named `socket` that is part of the *sockets* API. In the preceding paragraph we also refer to a "TCP socket," which is synonymous with a "TCP endpoint."

If the call to socket fails, we abort the program by calling our own `err_sys` function. It prints our error message along with a description of the system error that occured (e.g, "Protocol not supported" is one possible error from socket) and terminates the process. This function, and a few others of our own that begin with `err_,` are called throughout the text. We describe them in Section D.4

Specify Sever's IP Address and Port

We fill in an Internet socket address structure (`a sockadr_in` structure named `servaddr)` with the server's **IP** address and port number. We set the enitre structure to 0 using `bzero,` set the address family to AF_I NET, set the port number to 13 and set the IF address to the value specified as the first command-line argument (argv[1 D. The IF address and port number fields in this structure must be in specific formats: we call the library function. `htons` ("host to network short") to convert the binary port number, and we call the library function i net_ pton ("presentation to numeric") to convert the ASCII command-line argument (such as 206.62.226.35 when we ran this example) into the proper format.

Establish Connection with Server

The connect function, when applied to a TCP socket, establishes a TCP connection with the server specified by the socket address structure pointed to by the second argument. We must also specify the length of the socket address structure as the third argument to connect , and for Internet socket address structure we always let the compiler calculate the length using C's s i zeof operator.

Read and Display Server's Reply

We read the server's reply and display the result using the standard I10 fputs function. We must be careful when using TCP because it is a bytestream protocol with no record boundaries. The server's reply in normal a 26-byte string of the form

```
Fri Jan 12 14:27:52 1996 \r\n
```

where kr is the ASCII carriage return and kn is the ASCII linefeed. With a byte-stream protocol these 26 bytes can be returned in numerous ways: a single TCP segment containing all 26 bytes of data, in 26 bytes. Normally a single segment containing all 26 bytes of data is returned, but with larger data sizes we cannot assume that the server's bytes of data is returned, but with larger data sizes we cannot assume that the server's replf is returned by a single read. Therefore, when reading form a TCP socket we *always* need to code the read in a loop and terminate the loop when either read returns 0 (i.e., the other end closed the connection) or less than 0 (an error).

In this example the end of record is being denoted by the server closing the connection. This technique is also used by HTTP, the Hypertext Transfer Protocol. Other techniques are available, for eampole, FTP (File Transfer Protocol) and SMTP (Simple Mail Transfer Protocol both mark the end of a record with the 2-byte sequence of an ASCII carriage return followed by sequence of an ASCII linefed. Sun RPC (Remote Procedure Call) and the DNS (Doman Name System) place a bianry count countaining the record length in front of each record that is sent when using TCP. the important concept here is that TCP itself
provides no record makers: if the application wants to delineate the end of records, it must do so itself and there are a few common ways to accomplish this.

Terminate Programme

exit terminates the programme. Unix always closes all open descriptors when a process termiantes, so our TCP socket is now closed.

As we mentioned, the text goes into much more detail on all the points that we just described.

32 Protocol Independence.

Our programme in Figure 1.5 is *protocol dependent* on IP version 4 (IPv4). We allocate and initialise a **sockaddr_in** structure, we set the family of this structure to AF_IN **ET,** and we specify the first argument to socket as **A F_I NET.**

If we want to modify the programme to work under IP version 6 (IPv6) we must change the code. Figure 1.6 shows a version that works under IPv6, with the changes, highlighted in a bolder font.

_introldaytimetcpcliv6. c

```
# include "unp.h"

int
 main (int argc, char nargy);
{
        int              sockftd, n;
        char         recvline (MAXLINE + 1);
          struct sockaddr_in6 servaddr;

          if (argc !/ = 2)
                err_quit ("usage: a.out <I P address>);
      if ( (sockfd = socket AF-INET6, SOCK_STREAM, 0) ) < 0)
                  err_sys ("socket error");

 bzero (& servaddr, sizeof (servaddr);
  servaddr. sin6 _family = AF_INET6;
  servaddr. sin6_port= htons (13); /*daytime server*/
  if (ineLpton (AF-INET6, argv (1), & servaddr. sin-addr < = 0)
    err-quit ("inet-pton error for %s", argv [1] )

  if (connect (sockfd, (SA *) & servaddr, sizeof (servaddr) ) <0)
   err_sys ("connect error");

 while ( (n = read (sockfd, recvline, MAXLINE)) >0)
    recvline [n] = 0;          /*null terminate */
    if (fputs (recvline, stdout)== EOF)
     err_sys("fputs error");
 }
  if (n <  0)
    err_sys ("read error");
 exit (0);
}
```

intro/daytimetcpcl1v6.c

Figure 1.6 Version Figure 1.5 for IP VerSion 6

Only five lines are changed but what we now have is another protocol-dependent programme, this time dependent on IP version 6. It is better to make the programme *protocol independent.*
Another deficiency in our programmes is that the user must enter the server's IP address as a dotted-decimal number (e.g., 206.62.226.35 for the IPv4 version). Humans work better with name instead of numbers (e.g., laptop. koha la .com orjust laptop).

3.3 Error Handling: Wrapper Fundons

In any real-world programme it is essential to check *every* funtion call for an error return. In Figure 1.5 we check for errors from socket, i net_pton , connect, read, and fputs, and when one occurs we call our own functions err quit and err_sys to print an error message and terminate the programme. We find that most of the time this is what we want to do.
Since terminating on an error is the common case, we can shorten our programmes by defining a *wrapper function* that performs the actual function call, test the return value, and terminates on an error. The convention we use is to capitalise the name of the function, as in

```
sockfd= Socket (AF_I NET, SOCK_STREAM, );
```

Our wrapper function is shown in Figure *1.7.*

lib/wrapsock.c

```
int
Socket (int family, int type, int protocol )
{
    int        n;
    if( (n = socket (fa mi ly, type, protocol) ) <0)
        err_sys (socket error");
    return (n) ;
}
```

lib/ wrapsock.c

Figure 1.7 Our wrapper function for the socket function

Whenever you encounter a function name in the text that begins with an uppercase letter, that is a wrapper function of our own. It calls a function whose name is same but beginning with lowecase.
When describing the source code that is presented in the text, we always refer to the lowest level funtion being called (e.g.. Socket and not the wrapper function (e.g.,, Socket).

While these wrapper fucntions might not seem like a big savings, when we discuss threads latter we will find that the thread functions do not set the standard Unix errno varriable when an error occurs; instead the errno value is the retun value of the function. This means that every time we call one of the phtluead functions we must allocate a variable, save the return value in that variable, and then set errno to this value before calling err_sys. To avoid cluttering the code with braces, we can use C's comma operator to combine the assignment into ern° and the call of err_sys into a single statement as in the follwing:

```
it       q
!Wm= pthread_mutex lock (& ndone mutex)) ! = 0)
```

```
        errno = n, err sys ("pthread_mutex lock error") ;
```

Alternately we could define a new error function that takes the system's error number as an argument But we can make this piece of code much easier to read as just

```
        pthread_mutex_look(&ndone mutex);
```

by defining our own wrapper function shown in Figure 1.8.

_____________________ *lib/wrappthread.c*

```
void
pthread_mutex_lock (pthread_mutex_t "rnptr)
{
    irt      n;

    if ( (n = pthread mutex_lock (mptr) == 0)
          return;
    errno = n;
    err_sys ("pthread_mutex lock error");
801
```

——————————————*lib/ wrappthread.c*

Figure 1.8 Our wraper function for pthread_mutex lock.

Throughout the rest of this bok we will use these wrapper fucntions unless we need to check for an explicit error and handle it in some form other than terminating the process. We do not show the source code for all our wrapper function, but the code is freely available.

Unix errno Value

When an error occurs in a Unix function (such as one of the socket functions), the global variable errno is set to a positive value indicating the type of error and the function normally returns- 1 . Our e rr_sys function looks at the value of err no and prints the corresponding error message string (e.g., "Connection timed out" if errno equals ETIMEDOUT)

The value of errno is set by a function only if an error occurs. Its value is undefined if the function does not return an error. All of the positive error values are constants with an all uppercase name beginning with E and are normallydefined in the < sys / err no. h> header. No error has the value of O./

Storing errno in a global variable does not work with multiple threads that share all global variables. Throughout the text we use phrases of the form the connect function returns ECONN REFUSED" as shorthand to mean that the function returns an error (typically a return value-1) with ermo set to the specified constant.

3.4 A Simple Daytime Server

We can also write a simple version of a TCP daytime server, which work with the client from Seciton 3.1. We use the wrapper functions that we described in the previous section and show this server in Figure 1.9.

____ intro/daytimetcpsrv.c

```
#include "unp.h"
#include <time.h>
```

```
i n t
main(intargt,char**argy)

     int instenfd,connfd;
    structsockaddr_in servaddr;
        char buff[MAXLINE];
        time_t      'ticks;
           listenfd = Socket (AF_INET, SOCK_STREAM, 0) ;
           bzero (&servaddr, sizeof (servaddr) ) ;
           servaddr . sin_family = AF_INET;
           servaddr sin_addr . s_addr = htonl (INADDR_ANY);
           servaddr. sin_port = hton(13); /* daytime server */
           Bind (listenfd, (SA *) & servaddr, sizeof (servaddr) );
           Listen (listenfd, LISTENQ);
           for (    ;   ;   )
                     connfd = Accept (listenfd, (SA *) NULL, NULL);
                     ticks = time (NULL);
                      snprintf (buff, sizeof(buff), :%.24s \ r \ n", ctime
                     (&ticks)     )   ;
                       Write (connfd, buff, strlen (buff ) ) ;
                      Close (connfd) ;
    }
}                                                  intro/daytimetcPsrv.c
```

Figure 1.9 TCP daytime server

to Conclusion

In this unit, you have learned about the client-server arthitecture. You also learned about the construct of simple network programmes like establishing a link with a server. You should also use this techniques and method to read and display server's reply.

What you have learned in this unit concerns the basics of network programming and how you can read and display a server's reply. It also focused on to establish a link with a server. The units that follow should build upon these fundamentals.

a) Write a simple programme to display today's date and the current time.

Excereise 1.1

How would you handle errors in a network using swagger funtions.

Excercise 1.2

Discuss the protocol independence

7.0 Refe

Stevens, W.R. *Unix Network Programming,, (lnd ed)* Vol 1, Pretice Hall PTR., *1998.*

Module 3: Overview of Network Programming

Unit 2: Creating a TCP Socket

Introduction

In this unit, you will learn about the meaning of the socket and how to create a TCP socket. You will also learn about road-map to client-server and how to terminate a connection. You should be able to know the topology of the network by writing some lines of codes.

2³/⁴¹ O'Neal*

By the end of the unit, you should be able to:

- create a TCP socket
- understand how to terminate a connection
- understand the BSD networking
- use network programming to determine the topology of the network.

rea e Oe

The creation of the TCP socket is identical to the client code.

Bind server's well-known port to socket

The server's well-known port (13 for the daytime service) is bound to the socket by filling in an Internet socket address structure and calling bind. We specify the IP addres as I NADDR_ANY, which allows the server to accept a client connection on any interface, in case the server host has multiple interfaces. Later we will see how we can restrict the server to accepting a client connection on just a single interface, if we so desire.

Convert socket to listening socket

By calling `listen` the socket is coverted into a listening socket, on which incoming connections from clients will be accepted by the kernel. These three steps, `socket`, `binds`, and listen , are the normal steps for any TCP server to prepare what we call the *listening descriptor (I* `istenfd` in this example).

The constant L STEN Q is from our un `p.h` header. It specifies the maximum number of client connections that the kernel will queue for this listening descriptor.

Accept client connection, send reply

Normally the server process is put to sleep in the call to `accept`, waiting for a client connection to arrive and be accepted. A TCP connection uses what is called a *three-way handshake* to establish a conection and when this handshake completes, `accept` returns, and the return value from the function is a new descrtiptor `(connfd)` that is called the connected descriptor. This new descriptor is used for communication with the new client. A new descriptor is returned by `accept` for each client that connects to our server.

The style used throughout the book for an infinite loop is

```
for ( ; ; ) (
```

The current time and date is returned by the library function time, which returns the number of seconds since the Unix Epoch: 00: 00: 00 January 1, 1970, UTC (Coordinated Universal Time). The next library function, `-ctime`, converts this integer value into a human readable string such as

```
Fri Jan 12 14: 27: 25 1996
```

A carriage return and linefeed are appended to the string by `snprintf` and the result is written to the client by write:

3.1 Road map to Client-Server Examples in the Text

Two client-server examples are used predoninantly throughout the text to illustrate the various techniques used in network programming:

- a daytime client-server
- an echo client-server

To provide a road map for the different topics that are covered in this text, we summarise the programmes that hwe develop, and the starting figure number and page number in which the source code appears. Figure 1.1 lists the versions of the daytime client, two versions of which we have already seen. Figure 1.2 lists the versions of the daytime server. Figure 1.3 lists the versions of the echo client and Figure 1.4 lists the versions of the echo server.

Figure	Page	Description
1.5	6	TCP/IPv4, protocol dependent
1.6	10	TCP/IPv6, protocol dependent
9.8	253	TCP/IPv4, protocol dependent, calls gethostbyna me and getservbyname
11.7	287	TCP, protocol indepentent, calls getaddrinfo and tcp_connect
11.12	295	UDP, protocol independent, calls getaddrinfo and udp_cl ient
15.11	411	TCP, uses nonblocking connect
28.13	779	TCP/IPv4, XTI, protocol dependent
29.7	795	TCP,XTI, protocol independent, calls netdir_getbyna me and tcp_connect
31.3	823	UDP, XTI, protocol independent, calls netdir_getbyname and udp_cl ient
31.4	826	UDP,XTI, protocol independent, receives asynchronous errors
31.7	830	UDP, XTI, ptotocol independent, reads datagrams in pieces
33.8	857	TCP, protocol dependent, uses TPI instead of sockets or XTI
E.1	929	TCP, protocol dependent, generates SIGPIPE
E.5	932	TCP, protocol dependent, prints socket reveive buffer sizes and MSS
E.13	942	TCP, protocol dependent, allows hostname (gethostbyname) or address
E. 14	943	TCP, protocol independent, allows hostname (gethostbyna me)

Figure 1.1 Different versions of the daytime client developed in the text

Figure	Page	Description
1.9	13	TCP/IPv4, protocol dependent
11.9	290	TCP, protocol independent, calls getaddrinfo and tcp_listen
11.10	292	TCP, protocol independent, calls getaddrinfo and tcp_listen
11.15	298	UDP, protocol independent, calls getaddrinfo and udp_server
12.5	338	TCP, protocol independent, runs as stand-along deamon
12.12	345	TCP, protocol independent, spawned from inetd deamon
30.5	805	TCP,XTI, protocol independent, calls netdir_getbyname and tcp_Li sten
31.6	828	UDP, XTI, protocol independent, calls netdir_getbyna me and udp_server

Figure 1.2 Different versions of the daytime server developed in the text

Terminate connection

The server closes its connection with the client by calling close. This initiates the normal TCP connection termination sequence: a FIN is sent in each direction and each FIN is acknowledged by the other end. We say much more about TCP's three-way handshake and the four TCP packets used to terminate a TCP connection.

As with the client in the previous section, we have only examined this server briefly, saving all the details for later in the book. Note the following points;

- As with the client, the server is protocol depenednt on IPv4.
- Our server handles only one client at a time. If multiple client connecitons arrive at about the same time, the kernel queues them, up to some limit, and returns them to accept one at a time. This day-time server, which requires calling two library functions, time and cti me, is quite fast. But if the server took more time to service each client (say a few seconds or a minute), we would need some way to overlap the service of one client with another client. The server that we show in Figure 1.9 is called an *intrative server*, because it iterates through each client, one at a time. There are numerous techniques for writing a *concurrent* server, one that handles multiple clients at the same time. The simplest technique for a concurrent server is to call the Unix fork function, creating one child process for each client. Other techniques are to use threads instead of fork or to pre-fork a fixed number of children when the server starts.
- If we start a server like this from a shell command line, we might want the server to run for a long time, since servers often run for as long as the system is up. This requires that we add code to the server to run correctly as a Unix daemon: a process that can fun in the background, unattached to a terminal.

Figure	Page	Description
5.4	114	TCP/IPv4, protocol dependent
6.9	157	TCP, uses select
6.13	162	TCP, uses select and works in a batch mode
8.7	216	UDP/IPv4, protocol dependent
8.9	219	UDP, verifies server's address
8.17	227	UDP, call collect to obtain asynchronous errors
13.2	352	UDP, timeout when reading server's reply using S I GA LR M
13.4	354	UDP, timeout when reading server's reply using select
13.5	355	UDP, timeout when reading server's reply using SO_RCTVI M EO
14.4	380	Unix domain stream, protocol dependent
14.6	381	Unix domain datagram protocol dependent
15.3	400	TCP, uses nonblocking1/0
15.9	408	TCP, uses two processes (fork)
15.21	423	TCP, establishes connection then sends RST
18.5	476	UDP, broadcasts with race condition
18.6	479	UDP, broadcasts with race condition
18.7	481	UDP, broadcasts, race condition fixed by using pselect
18.9	483	UDP, broadcasts, race condition fixed by using sigsetjmp and siglongjmp
18.10	485	UDP, broadcasts, race condition fixed by using 1PC from signal handler
20.6	545	UDP, reliable using timeout, retreansm it, and sequence number
21.14	585	TCP, heartbeat test to serer using out-of-band data
23.2	606	TCP, uses two threads
24.6	642	TCP/IPv4, specifies a source route

Figure 1.3 Different versions of the echo client developed in the tex.

Figure	Page	Description
5.2	113	TCP/IPv4, protocol dependent
5.12	128	TCP/IPv4, protocol dependent, reaps terminated children.
6.21	165	TCP/IPv4, protocol dependent, uses select, one process handles all clients
6.25	172	TCP/IPv4, protocol dependent, uses poll, one process handles all clients
8.3	214	UDP/IPv4, protocol dependent
8.24	234	TCP and UDP/IPv4, protocol dependent, uses select
13.14	367	TCP, uses standard I10 library
14.3	379	Unix domain stream, protol dependent
14.5	380	Unix domain datagram protocol dependent
20.4	537	UDP, receive destination address and received interface; truncated datagrams
20.15	554	UDP, bind all interface addresses
21.15	585	TCP, heartbeat test to client using out-of -band data
22.4	594	UDP, uses signal-driven I10
23.3	607	TCP, one thread per client
23.4	610	TCP, one thread per client, portable argument passing
24.6	642	TCP/IPv4, prints received source route
25.30	689	UDP, uses icmpd to receive asynchronous errors
E.17	955	UDP, bind all interface addresses

Figure 1.4 Different versions of the echo server developed in the text

3.2 OSI Model

A common way to describe the layers in a network is the International Organisation for Standardisation (ISO) *open systems interconnection* model (OS!) for comptuer communications. This is a seven-layer model, which we show in Figure 1.5 along with the approximate mapping to the Internet protocol suite.

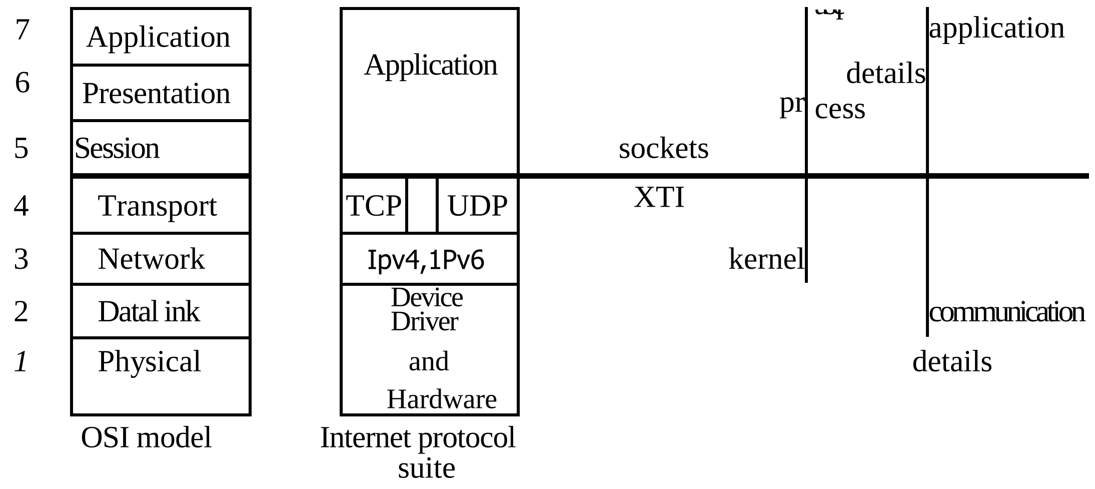


Figure 1.5 Layers in OSI model and Internet protocol suite

We consider the bottom two layer of the OSI model as the device driver and networking hardware that are supplied with the system. Normally we need not concern ourselves with these layers other than being aware of some properties of the datalink, such as the 1500-byte Ethernet MTU.

The network layer is handled by the IPv4 and IPv6 protocols, both of which we describe in Appendix A. The ctransport layers that we can choose from are TCP and UDP. We show a gap between TCP and UDP in

Figure 1.5 to indicate that it is possible for an application to bypass the transport layer and use 11³v4 or IPV6 directly. This is called a *raw socket.*

The upper three layers of the OSI model are combined into a single layer called the application. This is the Web client (browser), Telnet client, the Web server, the FTP server, or whatever application we are using. With the Internet protocols there is rarely any distinction between the upper layers of the OSI model. The two programming interfaces that we describe in this book, sockets and XTI, are interfaces from the upper three layers (the "application") into the transport layer. This is the focus of this book: how to write applications using either sockets or XTI that use either TCP or UDP.

Why do both sockets and XTI provide the interface from the upper three layers of the OSI model into te transport layer? There are two reasons for this design, which we note on the right side of Figure 1.5. first, the upper three layers handle all the details of the application (FTP, Telnet, or HTTP, for example) and know little about the communication details. The lower four layers know little about the application but handle all the communication datail: sending data, waiting for an acknowledgement, sequencing data that arrives or of order, calculating the verifying checksums, and so on. The second reason is that the upper three layers often form what is called a user process while the lower three layers are normally provided as part of the operating system kernel. Unix provides this seperation between the user process and the kernel, as do many other contemporary operating systems. Therefore the interface between layers 4 and 5 is the natural place to build the application programming interface (API).

3.3 BSI) Networking History

The sockets API orginated with the 4.2BSD system, released in 1983. Figure 1.6 shows the development of the various BSD releases, noting the major TCP/IP development. A few changes to the sockets API also took place in 1990 with the 4.3BSD Reno release, when the OSI protocols went into the BSD kernel.

The path down the page from 4.2BSD through 4.4BSD are the releases from the Computer Systems Research Group (CSRG) at Berkeley that required the recipient to already have a source code license for Unix. But all of the networking code, both the kernel support (such as the TCP/IP and Unix domain protocol stacks and the socket interface), along with the applications (such as the Telnet and FTP clients and servers),were developed independently from the AT&T-derived Unix code. Therefore starting in 1989 Berkeley provicded the first of the BSD networking releases, which contained all of the networking code and various other pieces of the BSD system that were not constrained by the Unix source code license. These release were "publicly available" and eventually available by anonymous FTP to anyone on the Internet.

The final releases from Berkeley were 4.4BSD-LIte in 1994 and 4.4BSD-Lite2 in 1995. We note that these two releases were then used as the base for other system: BSD/OS, FreeSBD, NetBSD, and OPenBSD, all four of which are still being actively developed and enhanced.

Many Unix systems started with some version of the BSD networking code, including the sockets API, and we refer to these implementations as *Berkeley-derived implementations.* Many commercial versions of Unix are based on System V Release 4 (SVR4) and some Of these have Berkeley-derived networking code (e.g, UnixWare 2.x), while the networking code in other SVR4 systems has been independently derive (e.g, solaris 2.x). We also note that the Linux system, a pupular, freely available implementation of Unix,
does not fit into the Berkeley-derived classification: its networking code and sockets API were developed

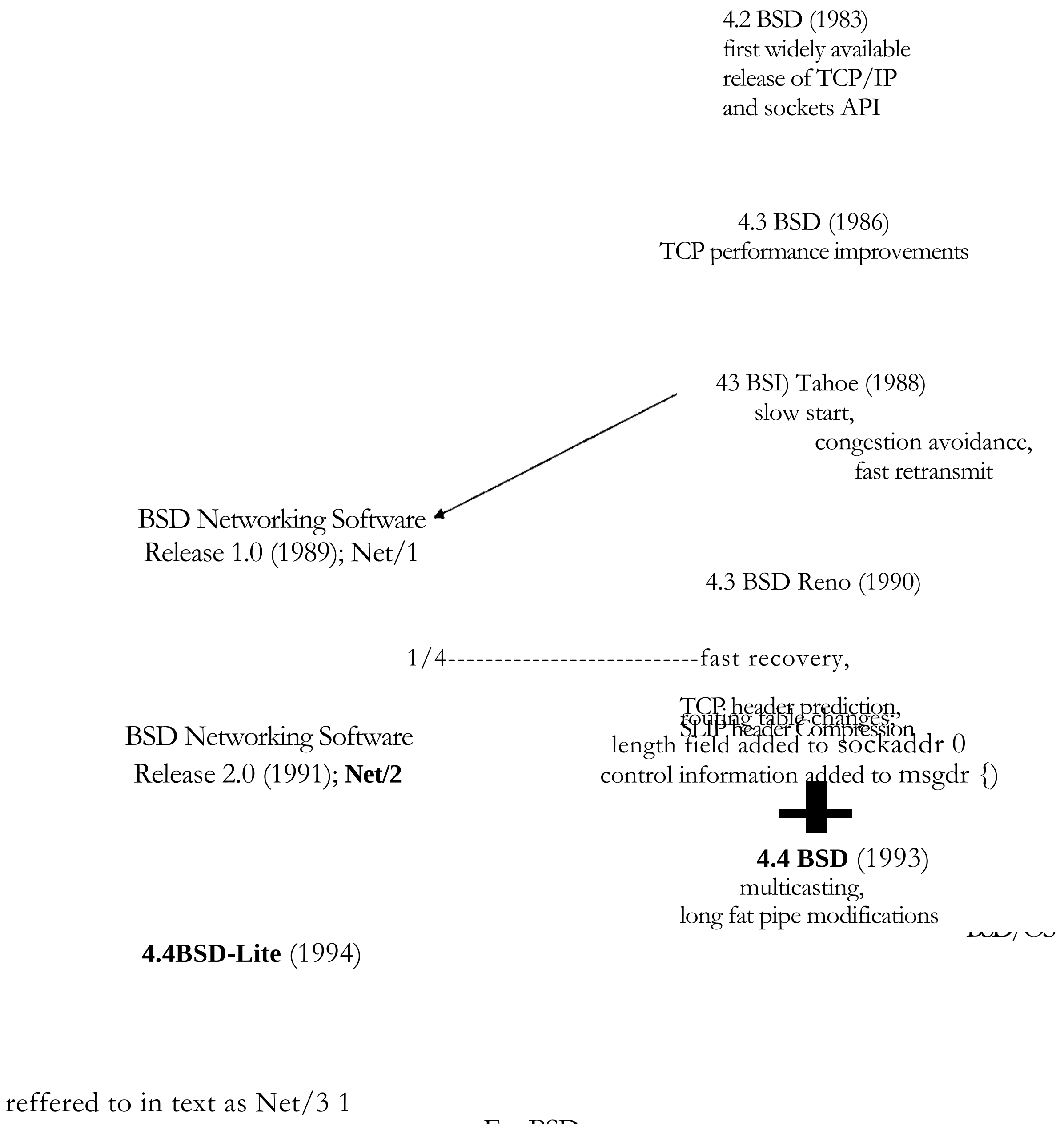


Figure 1.6 History of various BSD releases

3.4 Test Networks and Hosts

Figure 1.7 shows the various networks and hosts used in the examples throughout the text. For each host we show the operating system and the type of hardware (since some of the operating systems run on more than one type of hardware). The name within each box is the hostname that appears in the text.

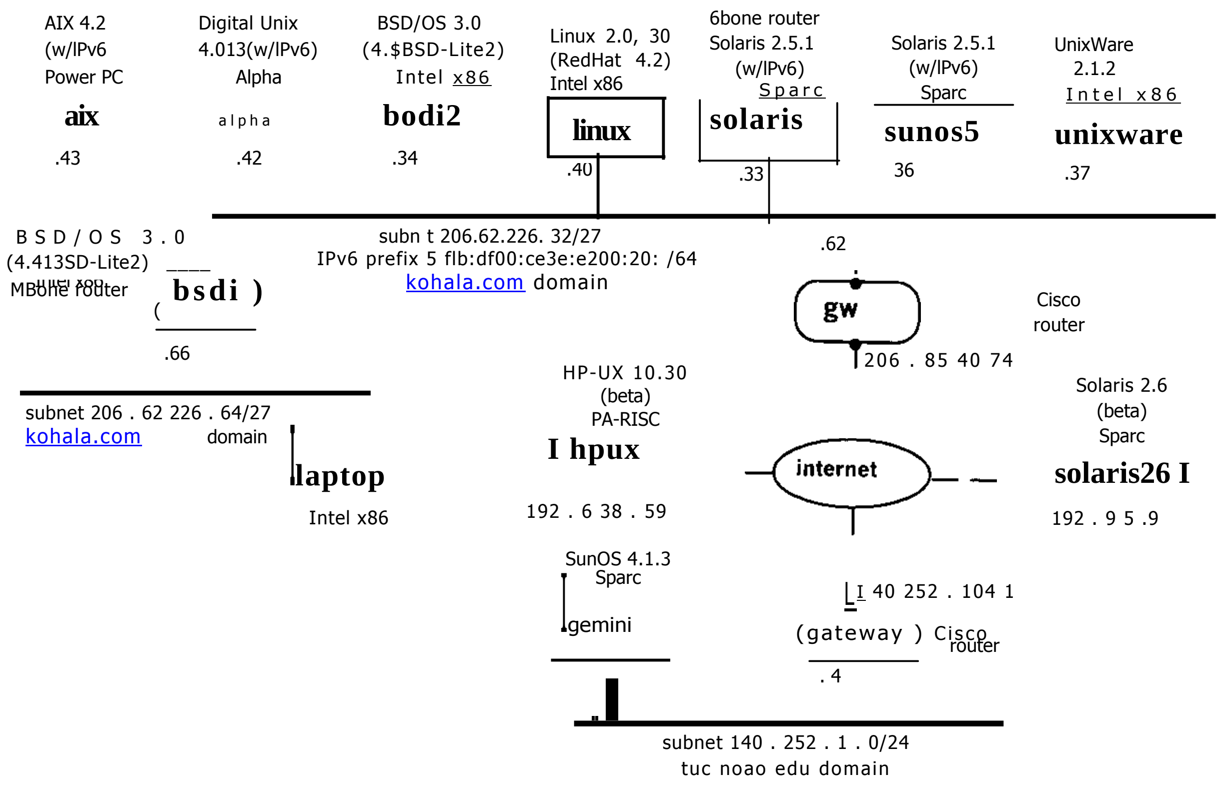


Figure 1.7 Networks and hosts used for most examples in the text

The hosts on the top two Ethernets with the subnet addresses 206.62.226.32/27 and 206.62.226.64/27 are all in the kohala.com domain. The hosts on the bottom Ethernet with the subnet address 140.252.1.0/24 are all in the tuc.noao.edu domain, which is run by the National Optical Astronomy Observatories. The notation /27 adn /24 indicates the number of consecutive bits starting from the leftmost bit of the address used to identify
the network and subnet.

Also in Figure 1.7 we draw nodes that function as routers with rounded corners, and nodes that are only hosts with square corners. We follow this convention throughout the book, as sometimes the distinction between a host and a router is important.

3.5 Discovering Network Topology

We show the network topology in figure 1.7 for the hosts used for the examples throughout this text, but the need to know your own network topology to run the examples and exercises on your own network. Although there are no current Unix standards with regard to network configuration and administration, two basic commands are provided by most Unix systems and can be used to discover details of a network: netstat and ifconfig. We show examples on some different systems from Figure 1.7. Check the manual pages for these commands on your system to see the details on the information that is output. Also be aware that some vendors place these comamnds in an administrative directory, such as / sbin or / usr / sbin, instead of the normal /nsr / bin, and these directories might not be in your normal shell search path (PATH).

1. netstat i provides information on the interfaces. We also specify the n flag to print numeric addresses,

```
linus % netstat - ni
Kernel Interface table

[face  MTU  Met   RX-OK    RX•ERR RX• DRP RX-OVR TX-OK TX-ERR TX-DRP TX-OVR    Flags
b      3584   0      32         0       0      0    32      0      0      0     BLRU
eth0   1500   0  483929         0       0      0 449881     0      0      0     BRU
```

The loopback interface is called lo and the Ethernet is callet `eth0`. The next example shows a host with IPv6 support.

```
alpha % netstat .ni
NameMut Network         Address                 Ipkts lerrs     Opkts Oerrs Coll
In0 15W <Link>          08:00:2b:37:64:26       11220    0       4893  0     4
In0 15C0 DLI            none                    11220    0       4893 0      4
In0 1500 206. 62 . 226. 206. 62 . 226.42        11220    0       4893  0     4
In0 1500 IPv6 FE80: :800:2B37: 6426             11220    0       4893  0     4
I no 15W I Pv6 5F1B: FOO: CE3E E200:20: 800: 2B 37:642611220 0 4893 0        4
In0 1536 <Link>         Link#3                  12432    0      12432 0      0
100 15% 127             127.0.0.1               12432    0      12432  0     0
lo0 1536 IPv6           :1                      12432    0      12432 0      0
tun0 576 <Link>         Link#4                  0        0          0 0      0
kin0 576 IPv6           :206.62.226.42          0        0          0 0      0
```

2. nentstat - r shows the routing table, which is another way to determine the interfaces. We normally specify the -n flag to print numeric addresses. This also shows the IF address of the default router.

```
a ix % netstat -rn
Routing tables
Destination             Gateway Flags       Refs    Use   MTU   Netif Expire

Route tree for    Protocol    Family    2 (Internet):
default              206 . 62 . 226 . 62    UG     0     0     -    en0
127/8                127.0.0.1              U      0     0     -    100
206.62.226.3227      206.62.226.43          U      4   475     •    en0

Route tree for    Protocol    Family 24   (Internet v6 ):
                     0.0.0.0        UC          0   0      1483  sit°         =>
default fe80::2:0:800:2078:e3e3     UG          0   0         -   en0
:1                   ::1                 UH     0        0 16896  100
5f1b:c1f00:c3e3:e200:20:180
                     link#2              UC     0   0      1500   en0
1980:116             link#2              UC     0   0       15W    en0
fe80:;2:0:800:2078:   e3e3
                     link#2              UHDL 1     0       15W    en0
f01:116              : :1                U      0   0         -   100
f02116        fe80: :800:5afc:2b36       U      1   3      1503   en0
ff11:116             ::1                 U      0   0         -   100
1112:116      fe80: :800:5afc:2b36       U      0   0      15oo   eno
```

(We have wrapped some of he longer lines to align the output fields.)

3. Given the interface names, we execute ifconfig to obtain the details for each interface.

```
linux % ifconfig eth0
eth0 Link encap : 10Mbps Ethernet HWaddr 00:A): 24: 9C: 43: 34
inetaddr:206.62.226.40 Bcast:206.62.226.63 Mask:255.255.255.224
UP BROADCAST RUNNING MULTICAST MTU: 1500 Metric : 1
RX packets : 484461 errors: 0 dropping: 0 dropped:0 overruns:0
TX packets : 450113 errors: 0 dropping: 0        dropped:0 overruns:0
Interrupt : 10 Base address : 0x300
```

This shows the IP address, subnet mask, and broadcast addrsss. The MULTI CAST flag is often an indication that the host supports multicasting.

```
alpha % ifconfig In°
In0 : flags=c63<UP. BROADCAST, NOTRAILERS, RUNNING, MULTICAST, SIMPLEX>
met
206.62.226.42 netmask ffffffe0 broadcast 206.62.226.63 ipmtu 1500
```

Some implementations provide a -a flag that prints the information on all configured interfaces.

4. One way to find the IP address of many hosts on the local network is to ping the broadcast address (which we found in the previous step).

```
bsdi % ping  206.62.226.63
PING 206.62.226.63 (206.62.226.63): 56 data bytes
64 bytes from 206.62.226.35: icmp_seq -0 tt1=225 time=0.316 ms
64 bytes from 206.62.226.40: icmp_seq=0 ttl =64 time=1.369 ms (DUP!)
64 bytes from 206.62.226.34: i cm p seq=0 tt1=225 time=1.822 ms (DUP!)
64 bytes from 206.62.226.42: icmp_seq tt1=64 time=2.27 ms (DUP!)
64 bytes from 206.62.226.37: icmp_seq=0 111=64 time=2.717 ms(DUP!)
64 bytes from 206.62.226.33: icmp_seqO ttl =225 time=3.281 ms (DUP!)
64 bytes from 206.62.226.62: icmp_seq=0 tt1=225 time-3731 ms (DUP!)
  ?                              type our interrupt key (DEL)
   206.62.226.63 ping statistics
   packets transmitted, 1 packets received, +6 duplicates, 0% packet loss round-trip min/avg/max = 0.316 /
 2.215 / 3.731 ins
```

4.0 Conclusion

In this unit, you have learned about the meaning of a socket and how to creat a TCP socket. You also learned how to terminate a connection and how to write network program to determine the topology of a network.

.0 Summary

What you have learned in this unit borders on socket and how to create a TCP socket. You also learned about how to terminate a connection. These schemes and techniques will afford you the opportunity to determine the topology of a network.

6.0 Tutor Marked Assignment

Explain how to create a TCP socket.

Stevens, W.R. *Unix Network Programming.* (2nd ed.) Vol. 1 Prentice Hall, PTR. (1998).

Module 3: overview of Network Programming

Unit 3: Elementary TCP Sockets

Page

it ufroduction

In this unit, we will describe the elementary socket functions required to write a complete TCP client and server. YOu will also learn about the Unix to provide concurrency when numerous clients are connected to the same server. Let now see what you will learn in this unit as specificed in the Unit objectives below.

2.0 Objectives

By the end of this unit, you should be able to:

- describe the elementary socket functions required to write a compelte TCP client and server.
- explain how the functions work for each operation described.
- undrestand the protocol used for queues and TCP three-way handshake.

ticground

This chapter describes the elementary socket fiinctions required to write a complete TCP client and server. We first describe all of the elementary sockets that we will be using and then develop the client and server in the next chapter. We will work with this client and server throughout the text, enhancing it many times. We also describe concurrent servers, a common Unix technique for providing concurrency when numerous clients are connected to the same server at the same time. Each client connection causes the server to fork a new process just for that client. In this chapter we consider only the one-process-per-client model using fork.

Figure 1.1 shows a time line of the typical scenario that takes place between a TCP client and server. First the server is started, then sometime later a client is started that connects to the server. We assume that the client sends a request to the server, the server processes the request, and the server sends back a reply to the client. This continues until the client closes its end of the connection, which sends an end-of-file notification to the server. The server then closes its end of the connection and either terminates or waits for a new client connection.

(a) **Socket** Function

To perform network I/O, the first thing a process must do is call the socket function, specifying the type of communication protocol desired (TCP using IPv4, UDP using IPv6. Unix domain stream protocol, etc.).

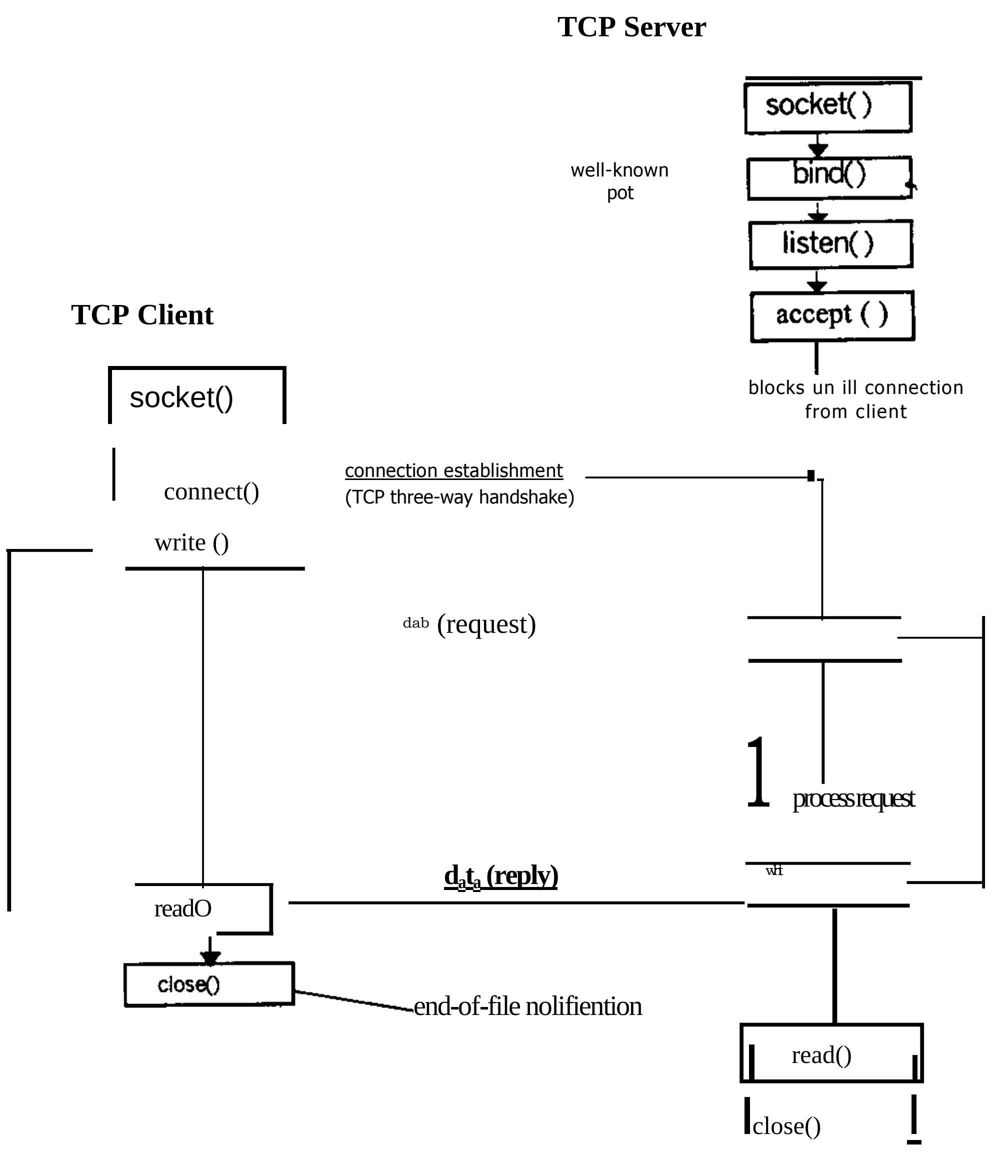


Fig 1.1Socket functions for elementary TCP client server

```
# include <sys/socket

int socket(i ntfamily, int int protocol);
            Returns:nonnegative descriptor if OK,-1 on
```

The family specifies the protocol family and is **one of the constants shown in Figure 1.2. The socket** *type* is one of the constants shown in Figure 1.3. Normally the protocol argument to the socket function is set to 0 except for raw sockets.

Not all combinations of socketfamily **and** *type* are valid. Figure 1.4 shows the valid combinations, along with the actual protocol that is selected by the pair. The boxes marked "Yes" are valid but do not have handy acronyms. The blank boxes are not supported.

family	Description
ALINET	IPv4 protocols
A F_INET6	IPv6 protocols
AF_LOCAL	Unix domain protocols
AF_ROUTE	Routing sockets
AF_KEY	Key socket

Figure 1.2 Protocol family constants for socket function.

type	Description
SOCK_STREAM	stream socket
SOCK_DGRAM	datagram socket
SOCK_RAW	raw socket

Figure 1.3 type of socket for socket function

	AF INET	AF INET6	AF_LOCAL	AF_ROUTE	AF_KEY
SOCK_STREAM	TCP	TCP	Yes		
SOCK_DG RA M	UDP	UDP	Yes		
SOCK_RAW	IPv4	1Pv6		Yes	Yes

Figure 1.4 Combinations of family and type for the socket function

On success the socket function returns a small nonnegative integer value, similar to a file descriptor. We call this a *socket descriptor*, or a *soc10*. To obtain this socket descriptor, all we have specified is a protocol family (1Pv4, IPv6, and Unix) and the socket type (stream, datagram, or raw). We have not yet specified either the local protocol address or the foreign protocol address.

AF_xxx versus *PF_xxx*

The A F_Prefix stands for "address family" and the PF_pfefix stands for "protocol family." Historically the intent was that a single protocol family might support multiple address families and that the PF_value was used to create the socket and the A F_ value was used in socket address structures. But in actuality, a protocol family supporting multiple address families has never been supported and the <sys/soc ket. h> header defines the P F_ value for a given protocol to be equal to the AF_value for that protocol. While there is no guarantee that this equality between the two will always be true, should anyone change this for existing protocol, lots of existing code would break. To conform to existing coding practice, we use only the AF_constants in this text, although you may encounter the PF_ value, mainly in calls to socket.

b) connect Function

The connect function is used by a TCP client to establish a connection with a TCP server.

```
# include <sys/socket.h>
int connect(' nt sockfd, co n st struct sockaddr *servaddr, soc kl e n_t addrIen);
                                        Returns; 0 if OK,-1 on error
```

sockfd is a socket descriptor that was returned by the socket function. The second and third arguments are a pointer to a socket address structure, and its size. The socket address structure must contain the IP address and port number of the server. We saw an example of this function in Figure 1.5.

The client does not have to call bind (which we describe in the next section) before calling connect: the kernel will choose both an ephemeral port and the source IP address if necessary.

In the case of a TCP socket, the connect function initiates TCP's three-way handshake. The function returns only when the connection is established or an error occurs. There are several different error returns possible.

I. If the client TCP receives no response to its SYN segment, ETI M EDO UT is returned. 4.4BSD, for example, sends one SYN when conned is call, another 6 seconds later, and another 24 seconds later. If no response is received after a total of 75 seconds, the error is returned.

   Some systems provides administrative control over this timeout; see Apendix E of TCPvl .

2. If the server's response to the client's SYN is an RST, this indicates that no process is waiting for connections on the server host at the port that we specified (i.e., the server process is probably not running). This is *hard error* and the error ECON NREFUSED is returning to the client as soon as the RST is received.

   An RST, meaning "reset", is a type of TCP segment that is sent by TCP when something is wrong. Three conditions that generate an RST are when a SYN arrives for a port that has no listening sever (what we just described), when TCP wants to abort an existing connection, and when TCP reveives a segement for a connecton that does not exist. (TCPvl pp. 246-250 contains additional information.)

3. If the client's SYN elicits an ICMP destination unreachable from some intermediate router, this is considered a soft error. The client kernel saves the messages but keeps sending SYNs with the same time between each SYN as in the first scenario. But if no response is received after some fixed amount of time (75 seconds for 4.4BSD), the saved ICMP error is returned to ther process as either EHOSTUNREACH or EN ETUN REACH.

We see these different error conditions with our simple client from Figure 1.5. We first specify the local host (127.0.0.1), which is running the daytime server and see the normal output.

```
solaris % daytimetcpcli 127.0.0.1
Tue Jan 16 16: 45: 07 1996
```

To see a different format for the returned reply, we specify the local cisco router.

```
solaris % daytimetcpcli 206.62 226.62
Tue, May7,1996 ii: 01:33-MST
```

Next we specify an IP address that is on the local subnet (206.62.226) but the host ID (55) is nonexistent. That is, there does not exist a host on the subnet with a host ID of 55, so when the client host sends out ARP requests (asking for that host to respond with its hardware address), it will never receive an ARP reply.

```
solaris%daytimetcpcli 206 62.26.55

connect error: Conneciton timed out
```

We only get the error after the connect time out (which we said was 3 minutes with Solaris 2.5). Notice that our err_sys function prints the human-readable string associated with the ETI M EDO UT error.
Our next example is to the host gateway, which is a Cisco router, that is not running a daytime server.

```
solans%daytimetcpcli140.252.1.4
connect error: Connection refused
```

The server responds immediately with an RST.
Our final example specifies anti) address that is no teadable on the Internet. If we watch the packets with tcpdump, we see that a router six hops away returns an ICMP host unreadable error.

```
solaris % daytimetcpcli 192.3.4.5
connect error: No route to host
```

As with the ETI M EDOUT error, in this example the connect returns the EH OST UN REACH error only after waiting its specified amount of time.
In terms of the TCP state transition diagram, connect moves from the CLOSED state (the state in which a socket begins when it is created by the socket function) to the SYN_SENT state and then, no success, to the ESTABLISHED state. If the connect fails, the socket is no longer usable and must be closed. We cannot call connect again on the socket. In Figure 11.6 we will see that when we call connect in a loop, trying each IP address for a given host until one works, each time connect fails we must close the socket descriptor and call socket again.

c) bind Function

The bind function assigns a local protocol address to a socket. With the Internet protocols the protocol address is the combination of either a 32-bit 1Pv4 address or a 128-bit IPv6 address, along with a 16-bit TCP or UDP port number.

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

int connect (int socicfd, const struct sockaddr *rnyaddr, socklen taddrien);

                                          Returns; 0 if OK, -1 on error
```

The second argument is a pointer to a protocol-specific address and the third argument is the size of this address structure. With TCP, calling bind lets us specify a port number, an IP address, both, or neither.

- Servers bind their well-known port when they start. If a TCP client or server does not do this, the kernel chooses an ephemeral port for the socket when either connect or listen is called. It is normal for a TCP client to let the kernel choose an ephemeral port, unless the application requires a reserved port, but it is rare for a TCP server to let the kernel choose an ephemeral port, since servers are known by their well-known port.

- A process can bind a specific IP address to its socket. The IP address must belong to an interface on the host. For a TCP client, this assigns the source IP address that will be used for IP datagrams sent on the socket. For a TCP server, this restricts the socket to receive incoming client connections destined only to this IF address

  Normally a TCP client does not bind an IP addess to its socket. The kernel then choose the source IP address when the socket is connected, based on the outgoing interface that is used, which in turn is based on the route required to reach the server .
  If a TCP server does not bind an IP address to its socket, the kernel uses the destination IF address of the c lei nt's SYN as the server's source IP address.

  As we said, calling bind lets us spevidy the IF address, the port, both, or neither. Figure 1.5 summarises the values to which we set the sin_addr adn sin_port, or the sin6_addr and sin6_port, depending on the desired result.

Process specifies		Result
IP address	port	
wildcard	0	kernel chooses IP address and port
wildcard	nonzero	kernel choosesIPaddress, process specifies port
local IP address	0	process specifies IP address, kernel chooses port
local IF address	nonzero	process specifies IP address and port

Fig. 1.5 Result when specifying IP address and / or port number to bind.

If we specify a port number of 0, the kernel choose an ephemeral port when bind is called. But if we specify a wildcard IP address, the kernel does not choose the local IF address until either the socket is connected (TCP) or until a datagram is sent on the socket (UDP).

With IPv4 the *wildcard* address is specified by teh constant I NA D D R_AN Y, whose value is normally 0. This tells the kernel to choose the IP address. We saw the use of this value in Figure 1.9 (Unit 1) with the assignment.

```
struct sockaddr_in        servaddr;
servaddr.sin_addr.s_addr = htonl (I NADDR ANY);       /* wildcard*/
```

While this works with IPv4, where an IP address is a 32-bit value that can be represented as a simple numeric constant (0 in this case), we cannot use this technique with 1Pv6, since the 128-bit Ipv6 address is stored in a structure. (In C we cannot represent a constant structure on the right-hand side of an assign-ment). To solve this problem, we write

```
struct sockaddr_in6 4serv;
serv.sin6_addr = in 6addr_any;       rwildeard */
```

The system allocates and initialises the in 6a d d r_a fly variable to the constant I N 6A D D R_A N Y_I NIT. The <netinet/in.h> header contains the extern declearatin for i n6addr_a ny.

The value of I NADDR_ANY (0) is the same in either network or host byte order, so the use of htonl is not really required. But since all th I NADDR constants defined by the <netinet / in.h> header are defined in host byte order, we should use htonl with any of these constants.

If we tell the kernel to choose an ephemeral port number for our socket, notice that bind does not return the chosen value. Indeed, it cannot return this value since the second argument to bind has the const qualifier. To obtain the value of the ephemeral port assigned by the kernel we must call getsockname to return the protocol address. A common example of a process binding a nonwildcard IF address to a socket is on a host that provides Web servers to multiple organisations.

First, each organisaton has its own domain name, such as www.organisation.com. Next, each organisation's domain name maps into a different IF address, but typically on the same subnet. For example, if the subnet is 198.69.10, the first organisation's IF address could be 198.69.10.128, the next 198.69.10.129, and so on. All of these IF addresses are then *aliased* onto a single network interface (using the alias option of the ifconfig command on 4.4BSD, for example) so that then) layer will accept incoming datagrams destined for any of the aliased addresses. Finally, one copy of the HTTP server is started for each organisation and each copy binds only the IP address for that organisation.

A common error from bind is EADDR IN USE ("Address already in use").

d) listen Function

The listen function is called only by a TCP server and it performs two actions.

1. When a socket is created by the socket function, it is assumed to be an active socket, that is, a

client socket that will issue a connect. The listen function converts an unconnected socket into a passive socket, indicating that the kernel should accept incoming connection requests directed to the socket. In terms of the TCP state transition diagram the call to listen moves the socket from the CLOSED state to the LISTEN state.

2. The second argument to this function specifies the maximum number of connections that the kernel should queue for this socket.

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

int listen(int socfd, int backlog) ;

                                   Returns: 0 if OK,-1 on error
```

This function is normally called after both the socket and bind functions and must be called before calling the accept function.

To understand the backlog argument we must realise that for a given listening socket, the kernel maintains two queues:

1. *An incomplete connection* queue, which contains an entry for each SYN that has arrived from a client for which the server is awaiting completion of the TCP three-way handshake.
2. *A completed connection queue,* which contains an entry for each client with whom the TCP three-way handshake has completed. These sockets are in the ESTABLISHED state.

Figure 1.6 depicts these two queues for a given listening socket.

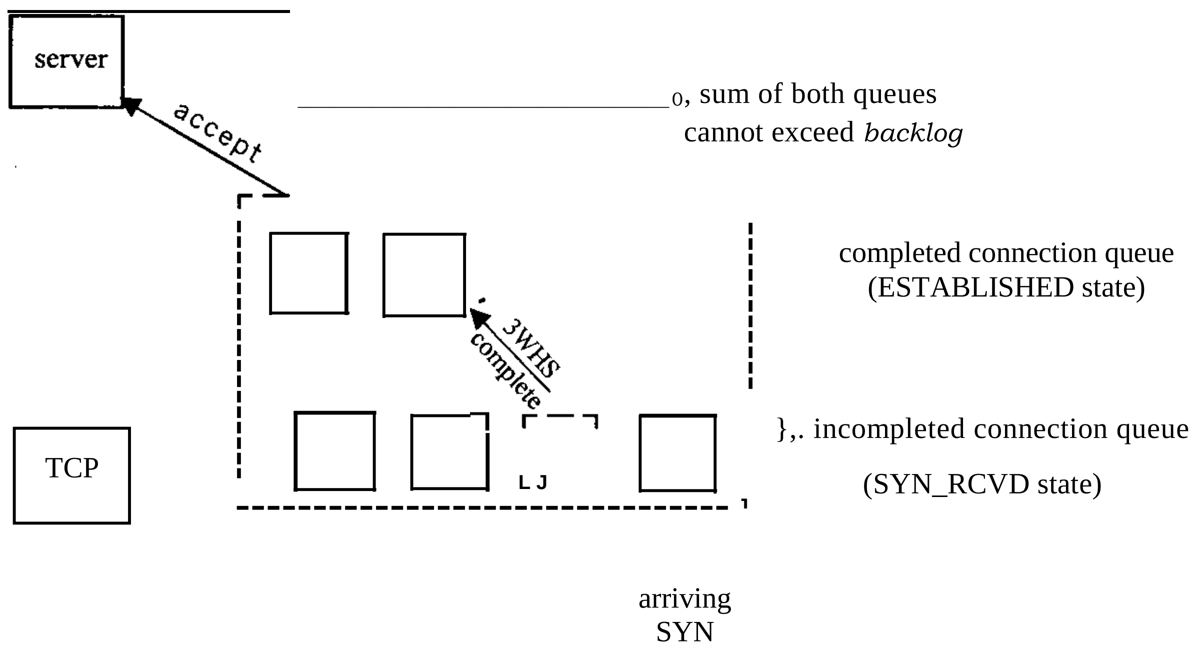


Figure 1.6 The two queses maintained by TCP for a listening socket.

When a SYN arrives from a client, TCP creates a new entry on the incomplete queue and then responds with

[illegible]

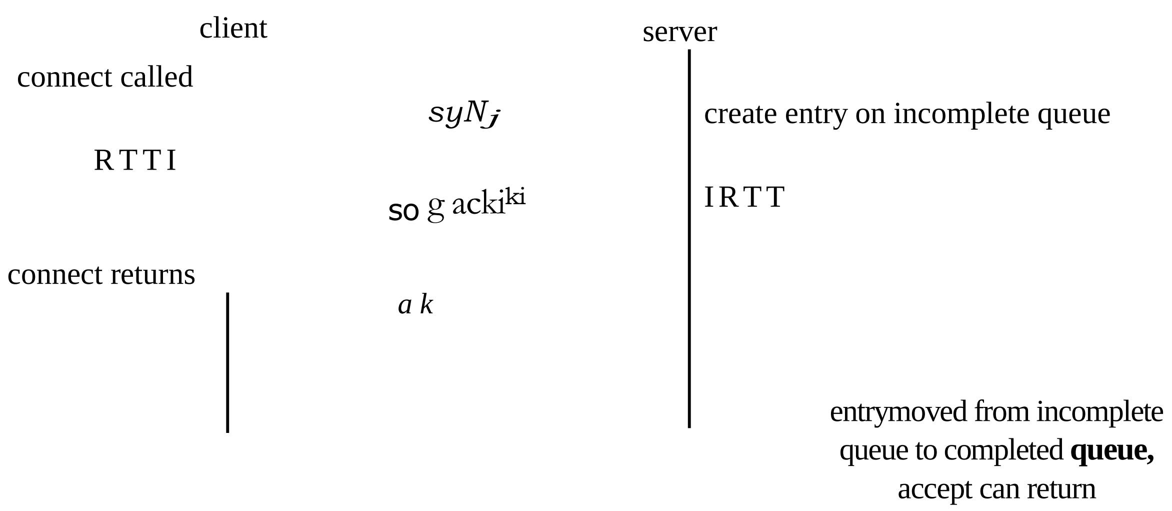

client
server
connect called
syNj
create entry on incomplete queue
RTTI
so g ackiki
IRTT
connect returns
a k
entrymoved from incomplete queue to completed **queue,** accept can return

Figure 1.7 TO' three-way handshake and the two queue for a listening socket

entry will remain on the incomplete queue until the third segment of the three-way handshake arrives (the client's ARK of the server's SYN), or until the entry times out. (Berkeley-derived implementatins have a timeout of 75 seconds for these incomplete entries.) If the three-way handshake completes normally, the entry moves from the incomplete queue to the end of the completed queue. When the process calls accept, which we describe in the next section, the first entry on the completed queue is returned to the process or, if the queue is empty, the process is put to sleep until an entry is placed onto the completed queue.

There are several points to consider about the handling of these two queues.

- The *backlog* argument to the listen function has historically specified the maximum value for the sum of both queues.
- Berldey-derived implementations add a fudge factor to the *backlog* that we specify: it is multiplied by 1.5. For example, the commonly specified *backlog* of 5 really allows up to eight queued entries on these systems, as we show in Figure 1.10
- Do not specify a *backlog* of 0, as different implementations interpret this differently. Some implememtations allow one queued connection, while others do not allow any queue connections. If you do not want any clients connecting to your listening socket, then close the listening socket.

ao Conclusion

In this unit, you have learned about the elementary socket functions and how those functions works. You also learned about how concurrency is handled in network programming

The methods and techniques that you have learned in this unit will enable you to know the protocols used for queue and three-way handshake for TCP connections.

SO Smmnaty

What you learned in this unit focuses on the elementary socket programming and the various socket functions and their usage. We looked at the concurrency issues and how it is being done when several clients are counected to a sewer. The next unit will build on this.

cag

a) Describe the operations of the socket functions that you know.

Exercise 1.1

How does bind fouction works?

Exercise 1.2

How would you perform 110 operation in the network using socket function?

.Rthttn **and Other** Itesow'ees

Stevens, W.R. *Unix Network Programming* (2nd ed.) Vol. 1, Prentice Hall, **PTR,** (1998).

Module 3: Overview of Network Progfamming

Unit 4: U ANStandards

In this unit, you will learn about the Unix Standards and the importance of Posix and Open Group in Unix standardisation.

You will also learn about the involvement of IETF in this standardisation and how it has affected the Unix versions and its portability. The issue of 64-bit architectures will be addressed and its effects on network programming. Let us now see what you will learn in this unit as stated on the unit objectives below.

ObjectiVes

By the end of this unit you should be able to:

- understand the relevance of standardisation in Unix.
- describe the implementation of Posix and Open Group
- understand the importance of portability in unix environment.
- appreciate why 64-bit architectures is good for network programming.

Most activity these days with regard to Unix standardisation is being done by Posix and The Open Group.

Posix is an acronym for "Portable Operating System Interface." Posix is not a single standard, but a family of standards being developed by the Institute for Electrical and Eletronics Engineers, Inc., normally called the *MIFF.* The Posix standards are also being adopted as international standards by ISO (the International Standard Organisation for Standardisation) and the IEC (the International Electrotechnical Commission), called ISO/IEC.

The first of the Posix standards was IEEE Std 1003.1-1988 (137 pages) and it specified the C language interface into a Unix-like kernel covering the following areas: process primitives (fork, exec, signals, timers), the environment of a process (user IDs, process group), files and directors i.e (all the I / 0 functions), terminal I/O, the system databases (password file and group file), and the tar and cpio archive formats.

The first Posix standard was a trial use version in 1986 known as "IEEEIX." The name posix was suggested by Richard Stallman.

The standard was updated in 1990 by IEEE Std 1003.1-1990 (356 pages), which was also International Standard ISO/IEC 9945-1: 1990. Minimal changes were made from the 1988 to the 1990 version. Appended to the title was "Part 1: System Application Programme Interface (API) [C Language]" indicating that this standard was the C language API.

The next of the Posix standards was IEEE Std 1003.2-1992, and its title contained "Part 2: Shell and Utiliities." It was established in two volumes, totaling about 1300 pages. This part defines the shell (based on the System V Bourne shell) and about 100 utilities (programmes normally executed from the shell, from awk and ba se n a me to vi and ya cc). Throughout this text we refer to the standary as *Posix.* 2.

Next came IEE std 1003. lb -1993, formerly known as IEEE PI003' .4. This was an update to the 1003.1_1990 standard to include the realtime extensions developed by the P1003.4 working group. The 1003.1b-1993 standard added the following items to the 1990 standard: file synchoronisation, asynochrous 110, semaphores, memory management (nunap and shared memory), execution scheduling, clocks and timers, and messages queues. The 1003.1b-1993 standard totaled 590 pages.

The next Posix standard was IEEE Std 1003.1, 1996 Edition [IEEE 1996], which includes 1003.1-1990 (the base API), 1003.1b-1993 (realtime extensions), 1003.1c-1995 (pthreads), and 1003.1i-1995 (technical corrections to 1003.1b). This standard is also called ISO! IEC 9945-1: 1996. Three chapters on threcads were added for a total size of 743 pages. Throughout this text refer to the standard as *Posix.1*

This standard also contains a Foreword stating that ISO/IEC 9945 consists of the following parts:

- Part 1: System application programme interface (API) [C language],
- Part 2: Shell and utilities, and
- Part 3: System administration (under development).

The Posix work that affects most of this book is IEEE Std 1003: lg: Protocol Independent Interfaces (PII), a product of the P1003.1g working group. This is the networking API standard and it defines two APIs, which it calls DNIs (Detailed Network Interfaces):

I. DNI/Socket, based on the 4.4BSD sockets API.
2 DNI/XTI, based on the X/Open XPG4 specification.

Work on this standard started in the late 1980s as the P1003.12 working group (later renamed P1003.1 g, but as of this writing, the standard is not complete (but getting close!). Draft 6.4 (May 19%) was the first draft to obtain more than 75% approval from the balloting group. Draft 6.6 (March 1997) appears to be the final draft [IEEE 1997a]. Somtime in 1998 or 1999 a new version of IEEE Std 1003.1 should be printed to include the P1003.1g standard.

Even though the P1003.1g standards is not officially complete, this book uses the features from Draft 6.6 of this standard whenever possible. Throughout this text we refer to this draft as *Posit. I g.* For example, the third argument to the connect function (Section 4.3) is shown as a soc klent datatype, even though this is new with Posix. 1 g. Similarly we describe the new Posix.lg soc katrn a rk function (Section 21.3) and provide an implementation of it using the ioctI function. We also use the Posix.Ig protocol value of AF-LOCAL instead of A F- UNIX for Unix domain sockets. Differences between current practice and Posix.lg are noted throughout the book. Although no vendors today support Posix.lg (since it is not final), once the standard is complete vendor support should be forthcoming.

Work on all of the Posix standards continues and it is a moving target for any book that attempts to cover it. The current status of the various Posix standards is available from http://www. pasc. org/sta ndi ng/sd 1 1 . htm I .

3.2 The Open Group

The Open Group was formed in 1996 by the consolidation of the X/Open Company (founded in 1984) and the Open Software Foundation (OSF founded in 1988). It is an international consortium of vendors and end-user customers from industry, government, and academia.

X/Open published the *X/Open Portability Guide*, Issue 3 (XPG3) in 1989. Issue 4 was published in 1992 followed by Issue 4, Version 2 in 1994. This latest version was also lcnown as "Spec 1170," with the magic number 1170 being the sum of the number of system interfaces (926), the numbe of headers (70), and the number of commands (174). The latest name for this set of specifications is the "X/Open Single Unix Specification although it is also called "Unix 95."

In March 1997 Version 2 of the Single Unix Specification was annoucned. Products conforming to this specification can be called "Unix 98." We refer to this specification as just "Unix 98" throughout this next. The number of interfaces required by Unix 98 increases from 1170 to 1434, although for a workstation this jumps to 3030, because it includes the CDE (Common Desktop Environment), which in turn requires the X Window System and the Motif user interface. Details are available in Posey 1997] and http: // www. opengroup. org/public/tech/unix/versions2.

We are interested in the networking services that are part of Unix 98. These are defined in [Open Group 1997] for both the sockets and XTI APIs. This specification is nearly identical to Draft 6.6 of Posix. 1g.

3.3 Internet Engineering Task Force

The *IETF*, the Internet Engineering Task Force, is a large open international community of network designers, operatior, vendors, and researchers concerned with the evolution of the Internet architecture and the smooth operation of the Internet. It is open to any interest individual.

The Internets standards process is documented in RFC 2026 [Bradner 1996]. Internet standards normally deal with protocol issues and not with progamming APIs. Nevertheless, two RFCs [Gilligan et al. 1997] and [Stevens and Thomas 1997] specify the sockets API for IP version 6. These are informational RFCs, not standards, and were produced to speed the deployment of portable applications by the numerous vendors working on early releases of IPv6. Standards bodies tend to take a long time. Nevertheless, at some time the IPv6 APIs will probably be standardised more formally.

3.4 Unix Versions and Portability

Most Unix systems today conform to some version of Posix.1 and Posix 2. We must use the qualifier "some" because as updates to Posix occur (e.g., the realtime extensions in 1993 and the pthreads addition in 1996) it takes vendors a year or two (sometimes more) to incorporate these latest changes.

Historically most Unix systems show either a Berkeley heritage or a System V heritage, but these differences are slowly disappearing as most vendors adopt the Posix standards. The main differences still existing deal with system administration, one area that no Posix standard currently addresses.

The focus of this book is on the forthcoming Posix. 1 g standard, with our main focus on the sockets API. Whenever possible we use the Posix functions.

3.5 64 bit Architectures

During the mid to late 1990s the trend is toward 64-bit architectures and 64-bit software. One reason is for larger addressing within a process (i.e., 64-pointers) that can address large amounts of memory (more than 2^{32} bytes). The common programming model for existing 32-bit Unix systems is called the *ILP32* model, denoting that integers (I), long integers (L), and pointers (P) occupy 32 bits. The model that is becoming most prevalent for 64-bit Unix systems is called *LP64* model, meaning only long integers (L) and pointers (P) require 64 bits. Figure 1.1 compares these two models.

Datatype	ILP32 model	LP64 model
char	8	co 1/4o p 3 3
short	16	
i n t	32	
long	32	
pointer	32	

Figure .1 Comparison of number of bits to hold various datatypes for ILP32 and LP64 models.

From a programming perspective the LP64 model means we cannot assume that a pointer can be stored in an integer. We must also consider the effect of the LP64 model on the existing APIs.

ANSI C invented thesize_t datatype, and this is used, for example, as the argument to ma I I oc (the number of bytes to allocate), and the third arugment to read and write (the number of bytes to read or write). On a 32- bit system size_t is a 32-bit value, but on a 64-bit system it must be a 64-bit value, to take advantage of the larger addressing model. This means a 64-bit system will probably contain a typedef of size_t to be an unsigned long. The networking API problem is that some drafts of Posix.Ig specified that function arguments containing the size of a socket address structures have thesize_t datatype (e.g., the third argument to bind and connect). Some XTI structures also had members with a datatype of long (e.g., the t-i nfo and Lopthdr structures). If these had been left as is, both would change from 32-bit values to 64-bit values when a Unix system changes from the ILP32 to the LP64 model. In both instances there is no need for a 64-bit datatype: the length of a socket address structure is a few hundred bytes at the most, and the use of long for the XTI structure members was a mistake.

What we will see are new datatypes invented to handle these scenarios. The sockets API uses the s oC k I e n_t datatype for lengths of socket address structures and XTI uses the t_sca I a r_t and t_u sc a I art dataytypes. The reason for not changing these values from 32 bits to 64 bits is to make it easier to provide binary compatibility on the new 64-bit systems for application compiled under the 32-bit systems.

4.6 . COnc1usion

In this unit, you have learned about the standardisation in Unix and the importance of Posix and Open Group in this standardisation. You also learned about the role of IETF in this process and portability of Unix. Finally, we discussed about the effect of 64-bit architectures on network programming because of its larger addressing of memory.

5.0 Summary.

....

What you have learned in this unit focuses on Unix standards and relevance of Posix and Open Group. We discussed the role of IETF in this process and how it has affected the portability of Unix. You learned also about the importance of 64-bit architectures in programming. The next units shah build upon this.

6.0 Tutor Marked Assignment

a) What are the importance of Posix and Open Group in Unix standards.

Excericse 1.1

What are the roles of IETF in networking

Excercise 1.2

Dissusss the 64 bit Architectures

7.0 References and other Resources

Stevens, W.A. *Unix Network Programming (2nd ed)* Vol 1 Prentice Hall, PTR, 1998.

Module 3: Overview of Network Programming

Unit 5: Sockets Introduction

duel

In this unit, you will learn about the sockets application program interface and the sockets address structures. You will also learn about the address conversion functions between a text representation of an address and the binary value that goes into a socket address structure. Let us now see what you will learn in this unit as stated in the Unix objectives below.

Obitetives

By the end of this unit, you should be able to·

- describe the sockets API
- understand address conversion functions
- describe the socket address structures in a protocol-independent fashion.

3.0 Background

This unit begins the description of the sockets API (application program interface). We begin with socket address structures, which will be found in almost every example in the text These structures can be passed in two directions: from the process to the kernel, and from kernel to the process. The latter case is an example of a value-result argument, and we will encounter other examples of these arguments throughout the text.

The address conversion functions convert between a text representation of an address and the binary value that goes into a socket address structure. Most existing IPv4 code uses inet_addr and i n et_nto a , but two new functions, in et_pt i on and in et_nto p, handle both IPv4 and IPv6.

One problem with these address conversion functions is that they are protocol dependent on the type of address being converted· IPv4 or IPv6. We develop a set of functions whose names begin with sock_that work with socket address structures in a protocol-independent fashion. We will use these throughout the text to make our code protocol independent.

3.1 Socket Address Structures

Most of the socket functions require a pointer to a socket address structure as an argument. Each supported protocol suite defines its own socket address structure. The names of these structure begins with sockad d r_ with a unique suffix for each protocol suite.

11Pv4 Socket Address Structure

An IPv4 socket address structure, commonly called an "Internet socket address structure," is named sockadd r_i n and defined by including the <net i net/ in. h> header. Figure 1.1 shows the Posix.lg definition.

```
struct in_addr                          /* 32-bit I Pv4 address */
   in_addr t s_addr;                    /*network byte ordered*/


}I
struct sockaddr-in
unit 8_ sin Jen;                        /*length of structure(16)*/
sa family t      sin family             /*AF-INET*/
in_port t        sin_port;              1*16-bit TCP or UDP porthumber*/
                                        /*network byte ordered*/
struct in•addr        sin•addr,         /* 32-bit I Pv4 address*/
                                        /" network byte ordered*/
char             sin zero(8)            /*unused*/
1;
```

Figure I. I: The Internet (IPv4) socket address structure: sockaddrin

There are several points we need to make about socket address structures in general, using this example. The length member, si n_l en, was added with 4.3BSD-Remo, when support for the OSI protocols was added. Before this release, the first member was si n_fa mily, which was historically an unsigned short. Not all vendors support a length field for socket address structures and Posix.Ig does not require this member. The datatype that we show, uint8i, is typical, and datatypes of this form are new with Posix.lg (Figure 2).

Having a length field simplifies the handling of variable-length socket address structures.

Even if the length field is present, we need never set it and need never examine it, unless we are dealing with routing sockets. It is used within the kernel by the routines that deal with socket address structures from various protocol families, (e.g. the routing table code).

Posix.lg requires only three members in the structure: sin_fa mi ly, sin_addr, and si n_port. It is acceptable for Posix-compliant implementation to define additional structure members, and this is normal for an Internet socket address structure. Almost all implementations add the si n_ze ro member so that all socket address structures are at least 16 bytes in size.

We show the Posix.Ig datatypes for the s_add r, si n_fa m i ly, and si n_port members. The i n_addr_t datatype must be an unsigned integer type of at least 32 bits, i n_port_t must be an unsigned integer type of at least 16 bits, and- safamily_t can be any unsigned integer type. The latter is normally an 8-bit unsigned integer if the implementation supports the length field, or an unsigend 16-bit integer if the length field is not supported. Figure 1.2 lists these three Posix-defined datatypes, along with some other Posix.Ig datatypes that we will encounter.

Datatype	Description	Header
intg t	signed 8-bit integer	<sysitypes.h>
Lane t	unsigned 8-bit interger	<sysitypes.h>
int16 t	signed 16-bit integer	<sysitypes.h>
uint16 t	unsigned 16-bit integer	csysitypes.h>
int32 t	signed 32-bit integer	<sysitypes.h>
uint32 t	unsigned 32-bit integer	<sysitypes.h>
sa family t	address family of socket address structure	<sys/socket.h>
socklen t	length of socket address structure, normally Li int 32_t	<sys/socket.h>
in_addr t	IPv4address,normallyuint 32_t	<nehnet/in.h>
in_port t	TCP or UDP port, normally uint 16_t	<netinet/in.h

Figure 1.2 Datatypes required by Posix.1g.

You will also encounter the datatypes u_char, u_short, u_i nt, and u_l ng, which are all unsigned. Posix. 1 g defines these with the note that they are obsolescent. They are provided for back ward compatibility.

Both the IPv4 address and the TCP or UDP port number are always stored in the structure in network byte order. We must be cognisant of this when using these members.

The 32-bit IPv4 address can be accessed in two different ways. For example, if serve is defined as an Internet socket address structure, then serv.si n_addr references the 32-bit IPv4 address as an n_add r structure, while serv.si n_addr.s_addr references the same 32-bit IPv4 as an in_addr_t (typically an unsigned 32-bit integer). We must be certain that we are referencing the IPv4 address correctly, especially when it is used as an argument to a function, because compilers often pass structures differently from integers.

- The si n_zero member is unused, but we always set it to 0 when fillingin one of these structures. By convention, we always set the entire structure to 0 before filling it in, not just the s i n_zero member.

Socket address structures are used only on a given host: the structure itself is not communicated between different hosts although certain fields (e.g. the IP address and port) are used for communication.

Generic Socket Address Structure

Socket address structures are *always* passed by reference when passed as a argument to any of the socket functions. But the socket functions that take one of these pointers as an argument must deal with socket address structures from *any* of the supported protocol families.

A problem is how to declare the type of pointer that is passed. With ANSI C the solution is simple: void *is the genetic pointer type, But the socket functions predate ANSI C and the solution chosen in 1982 was to define a *genetic* socket address structure in the <sys/soc ket. h> header, which we show in Figure 1.3

```
struct sockaddr
   uint8_t          sa_len ;
   safamily_t (sa_family;        P address family : AF_xxx value ' /
   char             sa_data[14]; Pprotocol-specific address * /
```

Figure 1.3 The generic socket address structure: sockaddr

The socket functions are then defined as taking a pointer to the generic socket address structure, as shown here in the ANSI C function prototype for the bind function:

```
int bind(int, struct sockaddr_in      serv:          socklen_t);
```

This requires that any calls to these functions must cast the pointer to the protocol specific socket address structure to be a pointer to a generic socket address structure.

For example:

```
struct sockaddr_ in serv ;       / ' I Pv4 socket address structure */
/* fill in serv{} */
bind(sockfd, (struct sockaddr *) &serv, sizeof (serv));
```

If we omit the cast "(struct socicaddr *) "the C compiler generates a warning of the form warning: passing arg 2 of 'bind' from incompatible pointer type" assuming the system's headers have an ANSI C prototype for the bind function.

From an application programmer's point of view, the *only* use of these generic socket address structures is to cast pointers to protocol-specific structures.

IPv6 Socket Address Structure

The 1Pv6 socket address is defined by including the <netinet/in.h> header, and we show it in Figure 1.4.

```
struct in6_addr (
 uint8_t s6_addr[16];                /*128.bit I Pv6 address */
                                      /" network byte ordered*/
   ;

#define SIN6_LEN          /*required for compile-time test*/

struct sockaddr_in6 (
 uint8i          sin6Jen              /* length of this struct (24)*/
 sa_family_t      sin6_family;        /* AF_INET6 */
 in_port_t       sin6_port ;          /* transport layer port# */
                                      /* network byte ordered */
 uint32_t        sin6_flowinfo;       /* priority & flow label*/
                                      /* network byte ordered */
 struct in6_addr sin6_addr;           /* IPv6 address */
);                                    /* network byte ordered */
```

Figure 1.4 IPv6 socket address structure: sockaddr_i n6.

Note the following points about Figure 1. 4:

- The S I N 6LEN constant must be defined if the system supports the length member for socket address structures.
- The 1Pv6 family is AF_I **N ET6,** whereas the 1Pv4 family is A F_I N ET.
- The members in this structure are ordered so that if the s oc ka ddr_i n6 structure is 64-bit aligned, so is the 128-bit si n6_a d d r member. On some 64-bit processors, data access of 64-bit values are optimised if stored on a 64-bit boundary.
- The sin6_flowinfo members is divided into three fields.
    - ∗ the low-order 24 bits are the flow label,
    - ∗ the next 4 bits are the priority,
    - ∗ the next 4 bits are reserved.

Comparison of Socket Address Structures

Figure 1.5 shows a comparison of the four socket address structures that we encounter in this text: IPv4, IPv6, Unix domain and datalink. In this figure we assume that the socket address structures all contain a 1-byte length field, that the family field also occupies 1 byte, and that any field that must be at least some number of bits is exactly that number of bits. Two of the socket address structures are fixed length, while the Unix domain structure and the datalink structure are variable length. To handle variable-length structures whenever we pass a pointer to a socket address structure as an argument to one of the socket functions, we pass its length as another argument. We show the size in bytes (for the 4.4B SD implementation) of the fixed-length structures beneath each structure.

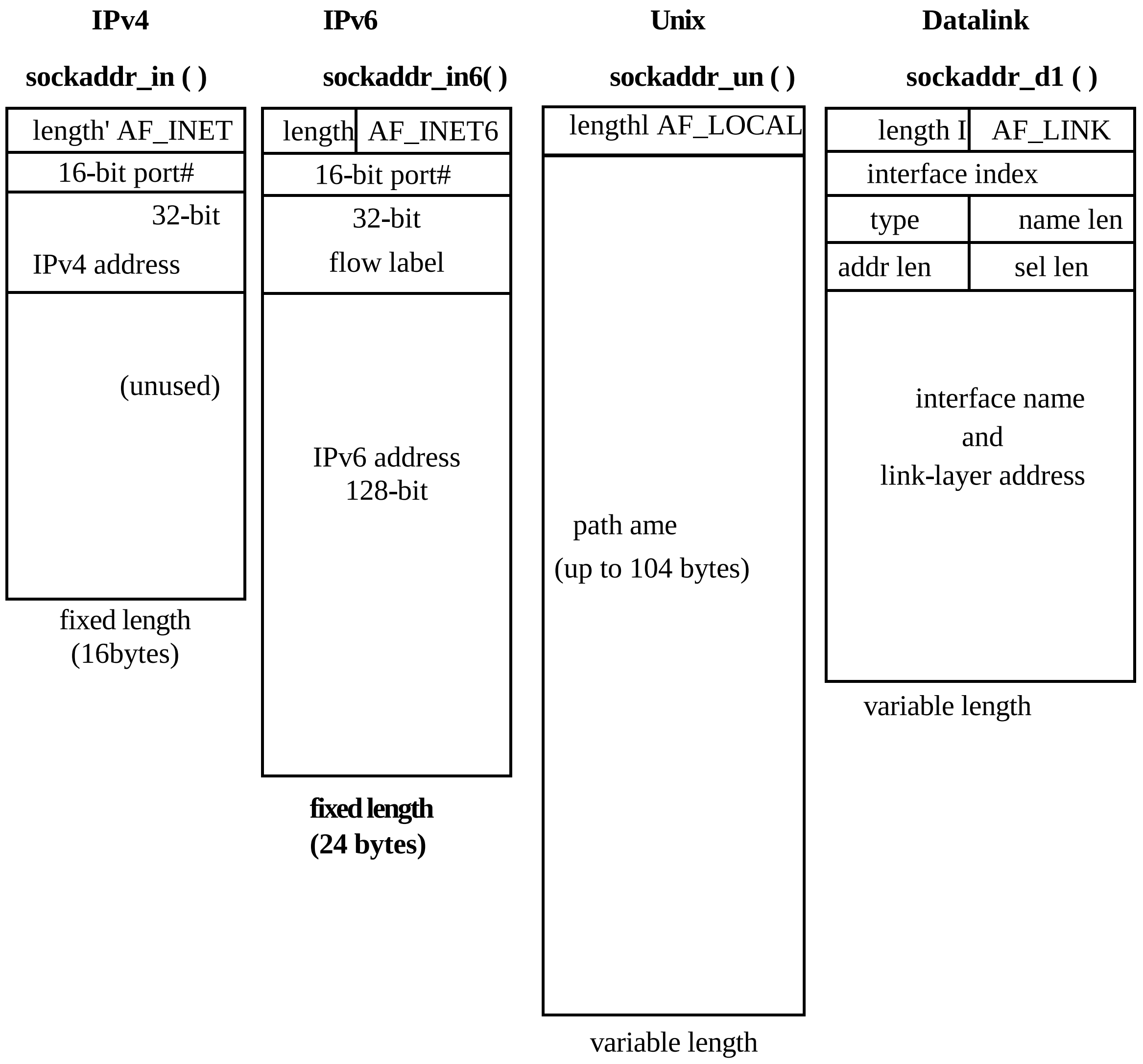


Figure 1.5 Comparison of various socket address structures

3.2 Value Result Arguments

We mentioned that when a socket address structures is passed to any of the socket functions, it is always passed by reference. That is, a pointer to the structure is passed. The length of the structure is also passed as an argument. But the way in which the length is passed depends on which direction the structure is being passed: from the process to the kernel, or vice versa.

1. The three functions bind, connect, and sendto pass a socket address structure from the process to the kernel. One argument to these three functions is the pointer to the socket address structure and another arguments is the integer size of the structure, as in

```
strut sockaddr_in serv;
   fill   in serv{} */
connect (sockfd, (SA *) &serv, sizeof (serv)) ;
```

Since the kernel is passed both the pointer and the size of what the pointer points to, it knows exactly how much data to copy from the process into the kernel. Figure1.6 shows this scenario.

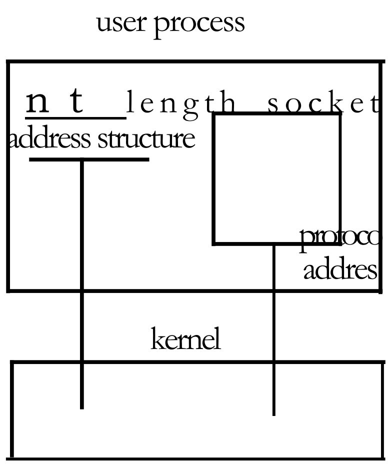


Figure 1.6 Socket address structure passed from process to kernel.

We will see in the next unit that the datatype for the size of a socket address structure is actually socklen_t and not int, but Posix.lg recommends that socklett defined as ui nt 32_t.

2. The four functions accept, reufrom, getsockname, and getpeerbame pass a socket address structure from the kernel to the process, the reverse direction from the previous scenario. Two of the arguments to these four functions are the pointer to an integer containing the size of the structure, as in.

```
struct     sockaddr_un     cli ;          / *     Unix     domain      * /
socklen_t           len

len = sizeof     (cli) ;                    / * len     is    a    value   *
getpeername (unixfd,      (SA* )      &cli,      &len )
/ *     len     may     have    changed     * /
```

The reason that the size changes from an integer to be a pointer to an integer is because the size is both a *value* when the function is called (it tells the kernel the size of the structure, so that the kernel does not write past the end of the structure when filling it in) and a *result* when the function returns (it tells the process how much information the kernel actually stored in the structure). This type of argument is called a *value-result*
argument. Figure 1.7 shows this scenario.

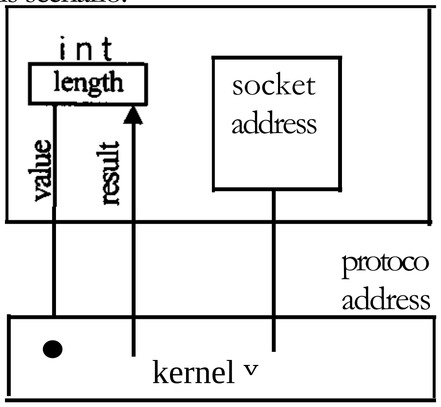


Figure 1.7 Socket address structure passed from kernel to process.

When using value-result arguments for the length of socket address structures, if the socket address structure is fixed length (Figure 1.5), the value returned by the kernel will always be that fixed size: 16 for an IPv4 sockaddr_in and 24 for an IPv6 sockaddr_i n6, for example. But with a variable-length socket address structure (e.g., a Unix domain sockaddr_un the value returned can be less than the maximum size of the structure.

With network programming the most common example of a value-result argument is the length of a returned socket address structure. But we will encounter other value-result arguments in this text:

- The middle three arguments for the select function
- The length arguments for the getsockopt function.
- The msgnamelen and msg_controllen members of the m sgh d r structure, when used with recvmsg .
- The ifc_len member of the ifconf structure.
- The first of the two length arguments for the sysctl function.

3.3 Byte Ordering Function

Consider a 16-bit integer that is made up of 2 bytes. There are two ways to store the 2 bytes in memory: with the low-order byte at the starting address, known as *little-endian* byte order, or with the high-order byte at the starting address, known as *big-endian* byte order. We show these two formats in Figure 1.8.

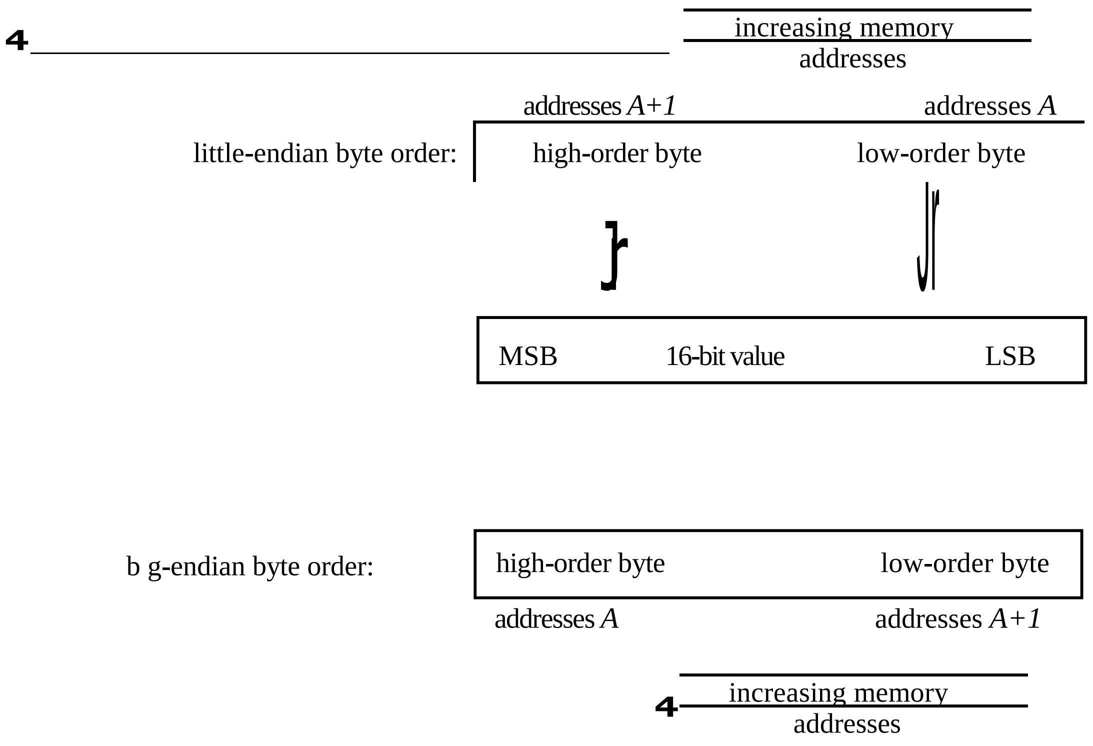


Figure 1.8 Little-endian byte order and big-endian byte order for a 16-bit integer

In this figure we show increasing memory addresses going from right to left in the top, and from left to right in the bottom. We also show the *MSB* (most significant bit) as the leftmost bit of the 16-bit value and the *LSB* (least significant bit) as the rightmost bit.

Unfortunately there is no standard between these two byte orderings and we encounter systems that use both formats. We refer to the byte ordering used by a given system as the *host byte order* The program shown in Figure 1.9 prints the host byte order.

intro/byteorder c

```
# include"unp.h"

int
main ( int argc, char *"argv)
(
    union (
        short s;
        char    c[sizeof (short)];
    } un ;

    un.s = 0x0102 ;
    printf ("%s:" CPU_VENDOR_OS);
    if (sizeof (short)== 2) {
        if (un .c[0]= = 1 && un.c[1] = = 2)
            printf ("big-endian \ n");
        else if (un.c[0] = = 2 && un .c [1] = = 1)
            printf ( "little-endian \ n");
        else
            printf ( "unknown \ n" );
    } else
        printf ( "sizeof (short) -= %cl \ n", sizeof (short));

    exit (0);
}
```

intro/byteorderc

Figure 1.9 Program to determine host byte order.

We store the 2-byte value Ox0102 into the short integer and then look at the two consecutive bytes c(10) (the address *A* in Figure 1.8) and 41] (the address *A+1* in Figure 1.8) to determine the byte order.

The string CP U_VE N DO R_OS is determined by the GNU a utoconf program when the software in this book is configured, and it identifies the CPU type, vendor, and operating system release. We show some exaples here in the output from this program when run on the various systems in Figure 1.7 in Unit 2.

```
aix%      byteorder
powerpc•ibm-aix4 2 0 0 b g end an

alpha%byteorder alpha•dec-
osf4.0 little.endian

bsdi%byteorder i386-
pcbsdi3.0:little-endian

gemini%byteorder sparc.sun-
sunos4.1.4:big-endian

hpux%bytearder hppal.1-hp-
hpux10.30:big-endian
```

```
I nux%byteorder 1586-pc-
linux-gnutittle-endian

solans%byteorder sparc-sun-
solans2 5 1:big-endian

unixware%byteorder 1386-
univel-sysv4 2MPhttle-endian
```

We have talked about the byte ordering of a 16-bit integer, and obviously the same discussion applies to a 32-bit integer.

We must deal with the byte ordering differences as network programmers because networking protocols must specify a *network byte order* For example, in a TCP segment there is a 16-bit port number and a 32-bit IPv4 address. The sending protocol stack and the receiving protocol stack must agree on the order in which the bytes of these multi-byte fields are transmitted. The Internet protocols use big-endian byte ordering for these multibyte integers.

In theory an implementation could store the fields in a socket address structure in host byte order and convert to and from the network byte order when moving the fields to and from the protocol headers, saving us from having to worry about this detail. But both history and Posix.Ig specify that certain fields in the socket address structures be maintained in network byte order. Our concern is therefore converting between the host byte order and the network byte order. We use the following four functions to convert between these two byte orders.

```
#include        <netinet/in.h>

uint16_t       htons(uint16_t host 1 6bitvalue);

uint32_t htonl(uint32_t host32bitvalue);
Both return: value in network byte order
                                uint 1 6_t ntohs(ui nt 1 6_t netbsbitvaue);

                                uint32_t ntohl(uint32_t net32bitvalue);

Both return: value in host byte order
```

In the names of these functions *h* stands for *host, n* stands for *network, s* stands for *short,* and / stands for *long.* The terms *shot* and *long\are* historical artifacts from the Digital VAX implementation of 4.2BSD. We should instead think of *s* as a 16-bit value (such as a TCP or UDP port number) and / as a 32-bit value (such as an IPv4 address). Indeed, on the 64-bit Digital Alpha, a long integer occupies 64 bits, yet the htonl and ntoh I functions operate on 32-bit values.

When using these functions we do not care about the actual values (big endian or little endian) for the host byte order and the network byte order. What we must do is be certain to call the appropriate function to convert a given value between the host and network byte order. On those systems that have the same byte ordering as the Internet protocols (big endian), these four functions are usually defined as null macros.

4.0 **Condusiun**

In this unit, you have learned about the sockets application program interface and sockets address staletures. You also learned about the address conversion functions between a text representation of an address and the binary value.

11$0" 1:$0,1000,11

What you have learned in this unit borders on sockets application program interface and the sockets address structures. You also learned about the address conversion. What you will learn in the next few units will build upon this issues.

6.0 Thtor Marked Assignment.

a) Describe the structure of the IPv4 socket address structure

Exercise 1.1

Describe the structures of a socket address.

Exercise **1.2**

Write on Byte ordering functions

etentes and Other: esoureetc•.:

Stevens, W.R. *Unix Network Programming.* (2nd ed.) Vol. 1, Prentice Hall, PTR, 1998.

Module 3: Overview of Network Programming

Unit 6: TCP Clio!manitServer Example

1.0 Introduction

In this unit, you will learn about the use of elementary functons and how they can be used to wirte a complete TCP client-server example. You will also learn how to perform simple I-0 operations between a server and a client. Let us now see what you will learn in this unit as stated in the unit objectives below.

24 Objectives

By the end of this unit, you should be able to:

- perform a read operation from a standard input and writes the line to the server.
- read a line from a network input and echo the line back to the client
- read and echoed line from the client and print it on its standard output.

We now use the elementary functions from the previous chapter to write a complete TCP client-server example Our simple example is an echo server that performs the following steps:

1 The client reads a line to text from its etmielarcl input and writes the line to the server.

2 The server reads the line from its network input and echoes the line back to the client

3. The client reads the echoed line and prints it on its standard output.

Figure 1.1 depicts this simple client-server along with the functions used for input and output.

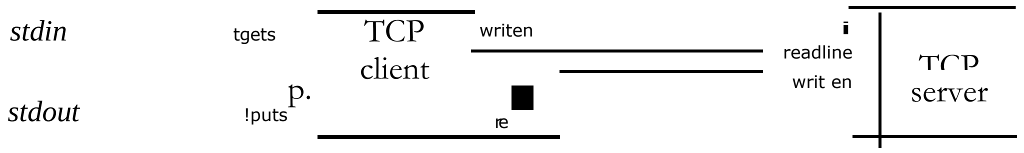


Figure 1.1 Simple echo client and server

We show two arrows between the client and sever but this is one full-duplex TCP connection. The fgets and fputs functions are from the standard I10 library and the writen and readline funcitons.

While we develop our own implementation of an echo sever, most TCP/IP implementations provide such a server, using both TCP and UDP. We will also use the server with our own client.

A client-server that echoes input lines is a valid, yet simple, example of a network application. All the basic steps required to implement any client-server are illustrated by the example. To expand this example into your own application all you need to do is change what the server does with the imput it receives from its clients.

Besides running our client and server in its normal mode (type in a line and wacth it echo) we examine lots of boundaryconditions for this example: what happens when the client and server are started; what happens when the client terminates normally; what happens to the client of the server process terminates before the client is done; what happens to the client if the server host crashes; and so on. By looking at all these scenarios and understanding what happens at the network level, and how this appears to the sockets API, we will understand more about what goes on at these level and how to code our applications to handle these scenarios.

In all these examples, we have "hard-coded" protocol-specific constants such as address and ports. There are two reasons for this First, we need to understand exactly what needs to be stored in the protocol-specific address structure. Second, we have not yet covered the library functions that make this more portable.

We note now that we will make many changes to both the client and server in successive units as we learn more about network programming.

3.1 TCP Echo Server: **main** Function

Our TCP client and server follow the flow of functions that we diagramed in Figure 1.1 (Unit 3) We show the concurrent server program in Figure 1.2.

Create socket, bind server's well known port

A TCP socket is created. An Internet socket address structure is filled in with the wildcard address (I NADDR_ANY) and the server's well-known port (SERV_PORT, which is defined as 9877 in our unph header). Binding the wildcard address tells the system that we will accept a connection destined for any local interface, in case the system is multihomed. It should be greater than 1023 (we do not need a reserved port), greater than 5000 (to avoid conflict with the ephemeral ports allocated by many Berkeley-derived implementations), less than 49152 (to avoid conflict with the "correct" range of ephemeral ports), and should not conflict with any registered port. The socket is coverted into a listening socket by listen.

Wait for client connection to complete

The server blocks in the call to accept, waiting for a client connection to complete.

Concurrent server

For each client, fork spawns a child, and the child handles the new client. As discussed in Section 3.0 (b) Unit 3, the child closes the listening socket and the parent closes the connected socket. The child than call str_echo (Figure 1.3) to handle the client.

tcpcliserv/tcpservOl.c

```
#include "unp.h"

int
main (intargc,char"argv)
{
    int     listenfd,confd;
    pid_t childpid;
    socklen_t clilen;
    struct sockaddr in cliaddr, servaddr;

    I itenfd = Socket (AF_IN ET, SOCK_ STREAM, 0) ;

    bzero(&servaddr, sizeof (servadr));
    servaddr.               = AF_INET;
    servaddr. sin_addr. s_addr = htonl (INADDR_ANY) ;
    servaddr. sin_port = htons (SERV_PORT) ;

    Bind ( listenfd, (SA *) &servaddr, sizeof (servaddr));

    Listen (listenfd, LISTNEQ);

    for( ; ; ){
      clilen = sizeof (cliaddr);
      connfd = Accept (listenfd, (SA*) &cliaddr, &clilen);

      if ( (childpid = Fork( ))= = 0 ){       /*child process*/
          Close (listenfd);       /* close listening socket*/
          str_echo(connfd) ; /* process the request* /
          exit (0) ;
      }
      Close (connfd) ;            /* parent closes connected socket* /
    }
}
```

tcpcliservficpservOlc

Figure 1.2 TCP echo server

3.2 TCP Echo Server: str_echo Function

The function str_echo, shown in Figure 1.3, performs the server processing for each client: reading the lines from the client and echoing them back to the client.

Read a line and echo the **line**

readline reads the next line from the socket and the line is echoed back to the client by **writen.** If the client closes the connection (the normal scenario), the receipt of the client's FIN causes the child's **read I ine to return).** This causes the **str_echo,** function to return, which terminates the child in Figure 1.2

3.3 TCP Echo Client: main Function

Figure 1.4 shows the TCP client **main** funciton.

lib/str_echo.c

```
#include "unp.h"
void
str_echo (int sockfd)
{
     ssize_t n;
      char line [MMLI NEI;
     for( ; ; )
         if ( (n = Readline (sockfd, line, MAXLINE)) = = 0
            return;              /*connection closed by other end * /
         Writen(sockfd, line, n);
     }
```

libistr_echo.c

Fiqure 1.3 str echo function: echo lines on a socket.

tcpcliserv/tcpcliOl.c

```
#include "unp.h"

int
main ( int argc,char**argv)
{
      int   sockfd;
     struct sockaddr_in servaddr ;

      if (argc != 2)
         err_quit {"usage: tcpcli <I Paddress>" );

     sockfd = Socket(AF_I NET, SOCK-STREAM, 0 ) ;
      bzero (&servaddr, sizeof (servaddr ));
      servaddrsiniamily = AF_INET ;
     servaddr. sin-port - htons (SERV_PORT) ;
     Inet_pton (AF_INET, argv [1], &servaddr. sin_addr);

      Connect (sockfd, (SA *) &servaddr, sizeof (servaddr));

       str cli (stdin, sockfd);              /*doit all*/
     exit (0) ;
}
```

tcpcliservitcpcli01 .c

Figure 1.4 TCP echo client

Create socket, fill in Internet socket address structure

A TCP socket is created and an Internet socket address is filled in with the server's 11^3 address and port number. We take the server's IP address from the command-line argument and the server's well-known port (SERV_PORT) is from our unp.h header.

Connect to server

connect establishes the connection with the server. The function str-c I i (Figure 1.5) then handles the rest of the client processing.

3.4 TCP Echo Client: str_cli Function

This function, shown in Figure 1.5, handles the client processing loop: read a line of text from standard input, write it to the sever, read back the server's echo of the line, and output the echoed line to standard output.

//b/sir c/i.c

```
#include "unp.h"
void
str cli (FILE -xfp, int sockfd)

     char      sendline (MAXLINE), recvline (MAXLINE) ;

     while (Fgets (sendline, MAXLINE, fp) !=NU LL) {

             Writen(sockfd, sendline, strlen (sendline )) ;

              if (Readline (sockfd, recvline, MAXLINE) = = 0)
                  err quit ( "str cli : sever terminated prematurely" ) ;

              Fputs (recvline, stdout) ;
1

```

lib/str_cli.c

Figure 1.5 str_cli function: client processing loop

Read a line, write to server

fgets reads a line of text wri ten sends the line to the server.

Read echoed line from server, write to standard output

readline reads the line echoed back from the server and fputs writes it to the standard output.

Return to main

The loop terminates when fgets returns a null pointer, which occurs when it encounters either an end-of-file or an error. Our Fget s wrapper function checks for an error and aborts if one occurs, so Fgets returns a null pointer only when an end-of-file is encountered.

3.5 Normal Startup

Although our TCP example is small (about 150 lines of code for the two ma in functions, str echo, st r_cli, read line, and writen), it si essential that we understand how the client and server start, how they end, and most importantly, what happens when something goes wrong: the client host crushes, the client process crushes, network connectivity is lost, and so on. Only by understanding these boundary conditions, and their interaction with the TCP/IP protocols, can we write robust clients and servers that can handles these conditions.

We first start the server in the background on the host b sd i .

```
bsdi % tcpserv &
[1]     21130
```

When the sewer starts, it calls socket, bind, listen, and accept, blocking in the call to accept. (We have not started the client yet.) Before starting the client, we run the netstat programme to verify the state of the server's listening socket.

```
    bsdi % netstat -a
    Prato Recv-Q Send-Q          Local Address      ForeignAddress        (state)
   tcp            0         0         * .9877                                    LISTEN
```

Here we show only the first line ou output (the heading), and the line that we are interested in. This command shows the status of *all* sockets on the system, which can be lots of output. We must specify the -a flag to see listening sockets.

The output is what we expect. A socket is in the LISTEN state with a wildcard for the local IF address and a local port of 9877. netstat prints an asterisk for an IF address of 0 (1 NA DDR_A N Y, the wildcard) or for a port of O.

We then start the client on the same host, specifying the server's IF address of 127.0.0.1. We could have also specified this address as 206.62.226.35 (Figure 1.7, unit 2.)

```
    bsdi % tcpcli 127.0.0.1
```

The client calls socket and con nect, the latter causing TCP's three-way handshake to take place. When the three-way handshake completes, connect returns in the client and accept returns in the server. The connection is established. The following steps then take place:

1. The client calls st r_c Ii, which will block in the call of fgets , because we have not typed a line of input yet.
2. When accept returns in the server, it calls fork and the child calls str_echo. This function calls readline, which calls read, which blocks, waiting for a line to be sent from the client.

3 The server parent, on the other hand, calls accept again, and blocks, waiting for the nest client connection.

We have three processes, and all three are asleep (blocked): client, server parent, and server child.

We purposely run the client and server on the same host, because this is the easiest way to experiment with client-server applications. Since we are running the client and server on the same host, netstat now shows two additional lines of output, corresponding to the TCP connection.

```
    bsdi % netstat -a
    Proto Recv-Q Send-Q Local Address    Foreign Address       (State)
   tcp          0          0 loca I host.9877    local host.1052        ESTABLISHED
   tcp          0          0 localhost.1052      localhost.9877         ESTABLISHED
   tcp          0          0 '.9877              **                    LISTEN
```

The first of the ESTABLISHED lines corresponds to the server child's socket, since the local port is 9877. Th second of the ESTABLISHED lines is the client's socket, since the local port is 1052. if we were running the client and server on different hosts, the client host would display only the client's socket, and the server

host would display only the two server sockets.
We can also use the ps command to check the status and relationship of these processes.

```
bsdi % ps-i
 PID  PPID      WCHAN     STAT      TT      TIME COMMAND
 19130 19129    wait      Is        p1    0 : 04.99 -ksh (ksh)
 21130 19130    netcon    I         pl    0 : 00.06 tcpserv
 21131 19130    ttyin               pl    0 : 00.09 tcpcli 127.0.0.1
 21132 21130    metio     I         pl    0 : 00.01 tcpserv
 21134 21133    wait      Ss        p2    0 : 03.50 -ksh (ksh)
 21149 21134              R+        p2    0 : 00.05 ps -1
```

(We have removed several columns of output that do not affect this discussion.) In this output we ran the client server from the same window (p I , which stasnds for pseudo-terminal number 1) and ran the ps command from another (p2). 'The PD and PPID columns shows the parent and child relationships. We can tell that the first tcpserv line is the parent and the second tcpserv line is the child since the PPID of the child is the parent's PD. Also the PPID of the parent is the shell (ksh).

The STAT column for all three of our network processes is "I" meaning the process is idle (i.e, asleep). The plus sign at the end of two of the STAT columns indicates that the process is in the foreground process group of its control terminal. If the process is asleep, the WCHAN column specifies the condition. 4.4BSD prints netcon if the process is blocked in either or connect, netio if the process is blocked on socket input or output, and ttyin if the process is blocked on terminal PO. The WCHAN values for our three network processes make sense.

3.6 Normal Termination

At this point the connection is established and whatever we type to the client is echoed back.

```
bsdi % tcpcli 127.0.0.1          we showed this line earlier
hello ,world                     we now type this
hello,   world                   and the line is echoed
good bye
good bye
AD                               Control D is our terminal EOF character
```

We type in two lines, each one is echoed, and then we type our terminal EOF character (Control-D) which terminates the client. If we immediately execute netstat we have

```
bsdi % netstat -a I grep 9877
   tcp        0      0    local host.1052    local host.9877 TIME_WAIT
   tcp        0      0 *.9877                                LISTEN
```

The client's side of the connection (since the local port is 1052) enters the TIME_WAIT state, and the listening server is still waiting for another client connection.
(This time we pipe the output of netstat into grep, printing only the lines with our server's well-known port. But doing this also removes the heading line.)
We can follow through the steps involved in the normal termination of our client and server.

1. When we type our EOF character, fgets returns a null pointer and the function following str di (Figure 1.5) returns.

2. When str_cli returns to the client ma in function (Figure 1.4), the latter terminates by calling exit.

3. Part of process termination is the closing of all open descriptors, so the client socket is closed by the kernel. This sends a FIN to the server, to which the server TCP responds with an ACK. This is the first half of the TCP conneciton termination sequence. At this point the server socket is in the CLOSE_WAIT state and the client socket is in the F I N_WA T_1 state.

4. When the server TCP receives the FIN, the server child is blocked in a call to read line (Figure 5.3), and read I i ne then returns O. This causes the str_echo function to return to the server child main.

5. The server child terminates by calling exit (Figure 1.2).

6. All open descriptors in the server child are closed. Closing the connected socket by the child causes the final two segments of the TCP connection termination to take place: a FIN from the server to the client, and an ACK from the client. At this point the connection is completely terminated. The client socket enters the T M E_WA I-111 state.

7. Another part of process termination is for the SIGCHLD signal to be sent to the parent when the server child terminates. That occurs in the example, but we do not catch the signal in our code, and the default action of the signal is to be ignored. The child enters the zombie state. We can verify this with the ps command.

```
bsdi % ps
  PID    IT    STAT  TIME    034.48ND
19130    pl    Ss    0 :   05.08    ksh   (ksh)
21130    pl    I     0 :   00.06    ./tcpserv
21132    pl    Z     0 :   00.00    (tcpserv)
21167    pl    R+    0 :   00.10    ps
```

The STAT of the child is now z (for zombie).

We need to clean up our zombie processes and doing this requires dealing with Unix signals. In the next section we give an overview of signal handling and the following sectin continues our example.

3.7 Posh Signal Handling

A *signal* is not notification to a process that an event has occured. *Signals* are somethings called *software interrupts*. Signals usually occur *asynchronously*. By this we mean that the process doesn't know ahead of time exactly when a signal will occur.

Signals can be sent

- by one process to another process (or to itself),
- by the kernel to a process.

The S GCH LD signal that we described at the end of the previous section is one that is sent by the kernel whenever a process terminates, to the parent of the terminating process.

Every signal has a *disposition,* which is also called the *action* associated with the signal. We set disposition of a signal by calling the sigaction function (described shortly) and we have three choices for the desposition.

1. We can provide a function that is called whenever a specific signal occurs. This function is called a *signal handler* and this action is called catching the signal. The two signals SI GKI LL and S I GSTO P cannot be caught. Our function is called with a single integer argument that is the signal number and the function returns nothing. Its funciton prototype is therefore

   void handler (int signo);

   For most signals, calling sigact ion and specifying a function to be called when the signal occurs is all that is required to catch a signal. But we will see later that a few signals, SIG 10, SI G PO L L, and SI GU RG , all require additional actions on the part of the process to catch the signal.

2. We can *ignore* a signal by setting its disposition to SIG_IGN. The two signals SI GK I LL and S I GSTOP cannot be ignored.

3. We can set the *default* disposition for a signal by setting its disposition to SI G_D FL. The default is normally to terminate a process on the receipt of a signal, with certain signals also genereating a core image of the process in its current working directory. There are a few signals whose default disposition is to be ignored: SIGCHLD and SIGURG, (sent on the arrival of out-of-band data) are two that we encounter in this text that are ignored by default.

signal Function

The Posix way to extablish the desposition of a signal is to call the s iga cti on function. This gets complicated, however, as one argument to the function is a structure that we must allocate and fill in. An easier way to set the disposition for a signal is to call that signal function. The first argument is the signal name and the second argument is either a pointer to a funciton or one of the constants S I G_I G N. or SI G_D FL But signal is a historical function that predates Posix. 1 and different implementations provide different signal semantics when it is called, providing backward compatibility, whereas Posix explicitly spells out the semantics when s i gacti on is called. The solution is to define our own funciton named signal just calls the Posix sigaction function. This provides a simple interface with the desired Posix semantics. We include this funciton in our own library, along with our err_XXX functions and our wrapper functions, for example, that we specify when building any of our programs in this text.

This funciton is shown in Figure 1.6.

litilsingal c

```
# include "unp.h"

 sigfunc*
signal (mnt signo, Sigfunc Kfunc)
 {
      struct signaction act, oact;

       act.sa_handler = func;
      sigempotyset (&act.sa mask) ;
      act.sa_flags = 0;
     if (singo = = SIGALRM){
# ifdef SA_I NTERRU PT
            act.sa flags I =SA_I NTERRUPT; /'SunOS 4.x */
#endif
        } else {
#ifdef SA_RESTART
            act.sa flags I = SA_RESTART; /* SVR4, 4.4BSD ⁴/
#endif
      }
```

```
    if (signaction (signo, &act, &oact) < 0)
      return (SIG_ERR) ;
    return (oact.sa_handler);

```

lib/singaLc

Figure 1.6 signal function that calls the Posix signaction *function,*

Simplify function prototype using typedef

The normal function prototype for signal is complicated by the level of nested parentheses:

```
void (*signal (int signo, void (*func) (int) ) ) (int) ;
```

To simplify this we define the Sigfunc type in our u np.h header as

```
typedef      void      Sigfunc (int) ;
```

stating that signal handlers are functions with an integer argument and the function returns nothing (void). The function prototype is then

```
Sigfunc *signal (int signo, Signfunc *func) ;
```

A pointer to a signal handling function is the second argument to the function, as well as the return value from the function.

Set handler

The sa_handler member of the signation structure is set to the *flint;* argument.

Set signal mask for handler

Posix allows us to specify a set of signals that will be *blocked* when our signal handler is called. Any signal that is blocked cannot be delivered to the process. We set the sa_mask member to the empyt set, which means that no additional signals are blocked while our signal handler is running. Posix guarantees that the signal caught is always blocked while its handler is executing.

Set **SA_RESTART** flag

An optional flag is SA_R ES TA RT and if set, a system call interrupted by this signal will be authomatically restarted by the kernel. (We talk more about interrupted system calls in the next section when we continue our example.) If the signal being caught is not SI GALRM, we specify the SA RESTART flag, if defined. Older systems, notably SunOS 4.x, authomatically restart an interrupted system call by default and then define the complement of this flag as SA_I NTERRU PT. If this flag is defined, we set it if the signal being caught is SIGALRM,

Call signaction

We call signaction and then return the old action for the signal as the return value of the signal function.

Throughout this text we use th signal functino from Figure 1.6.

Posix Signal Semantics

We summarize the fullowing points about signal handling on a Posix-compliant system.

- Once a signal handler is installed, it remains installed. (Older systems removed the signal handler each time it was executed.)
- While a signal handler is executing, the signal being delivered is blocked. Furthermore any additioinal signals that we re specified in the sa_mask signal set passed to sigaction when the handler was installed are also blocked. In Figure 1.6 we set sa_mask to the empty set, meaning on additional

signals are blocked other than the signal being caught.

- If a signal is generated one or more times while it is blocked, it is normally delivered only one time after the signal is unblocked. That is, by default Unix signals arenot *queued* We will see an example of this in the text section. THe Posix realtime standard, 1003.1b, defines a set of reliable signals that are queued, but we do not use them in this text.

In this unit, you have learned about the applications of some elementary functn ons and how they can be used to perform simple I-0 operations between a client and a server.

Summary..

What you have learned in this unit focuses on the applications of simple functions to perform basic 1-0 operations. The next few units shall build upon these fundamentals.

..:4:4EwlottorMorloctAismonlenc

a) Describe the operations of the TCP Echo Server and TCP Echo Client functions that you know.

Exercise 1.1
Discuss the Normal startup and termination

Exercise 1.2

Describe the Pos x Signal Handling

'7.0 **Refrences** and Resources

Stevens. W.R. (1998) *Unix Network Programing, vol. I,* 2nd ed., Prentice Hall, PTR.

Module 3: Overview of Network Programming

Unit 7: Handling Interrupted Calls

•:.ft Introduction

In this unit, you will learn about how calls are handled in a client-server systems You will also learn about how to connect, abort, and terminate a call. You will be exposed to some operations like read, write _ and select the open. Let us now see what you will learn in this unit as specified in the unit objectives below.

2.0 Objectives

By the end of this unit, you should be able to.

- understand the meaning of calls in client-server environment
- initiate and abort a call
- terminate a seiner process
- understand how to crash, reboot, and shutdown a server.

3.4) Background

We used the term *slow system call* to describe accept and we use this term for any system call that can block forever. That is, the system call need never return. Most networking function falls into the catelory. For example, there is no guarantee that a server's call to *accept* will ever return, if there are no clients that will connect to the server. Similarly our server's call to read (through *readline)* in Figure 1.3 will never return if the client never sends a line for the sever to echo. Other examples of slow system calls are reads cind writes of pipes and terminal devices. A notable exception is disk I/O. which usually returns to the caller (assuming no catastrophic hardware failure.)

The basic rule that applies here is that when a process is blocked in a slow system call *anti* the process catches a signal *and* the signal handler returns, the system call *can* return an error of El NT R. sonic kernels automatically restart some interrupted system calls. For portablility when we write a programme that catches signals (Most concurrent servers catch SIGCHLD), we must be prepared for slow system calls to return El NTR. Portability problems are caused by the qualifiers "can" and "some" used earlier and the fact that support for he Posix SA RESTART flag is option. Even if the implementation supports the SA_RESTART flag, not all interrupted system calls may automatically be restarted. Most Berkeley-derived implementations, for example, never automatically restart select and some of these implementations never restart accept or recvform

To handle an interrupted accept we change the call to accept in Figure 1.2, the beginning of the loop, to be following:

```
for    ( ; ; ) {
        chlen sizeof (cliaddr);
        If ( (con nfd = accept (I istenfd, (SA) &cl iaddr, &cl len) ) c0){
           if(errno = = El NT R)
              continue;              /* back to for( )*/
         else
             err_sys ("accept error") ;
```

Notice that we call accept and not our wrapper function accept, since we must handle the failure of the function ourself.

What we are doing in this piece of code is restarting the interrupted system call ourself This is fine for accept along with the functions such as read, write, select, and open. But there is one function that we cannot restart ourself: connect If this function returns EINTR we cannot call it again, as doing so will return an immediate error. When connect is interrupted by a caught signal that is not autontaticaly restarted, we must call select to wait for the connection to complete.

3.1 Difference between wait and waitpid

We now want to illustrate the different between the wait and waitpid functions, when used to clean up terminated children. To do this we modify our TCP client as shown in Figure 1.2. The client establishes five connections with the server and then uses only the first one (sockfd[0]) in the call to str_cli. The purpose of establishing multiple connectins is to spawn multiuple childre from the concurrect server, as shown in Figure 1.1

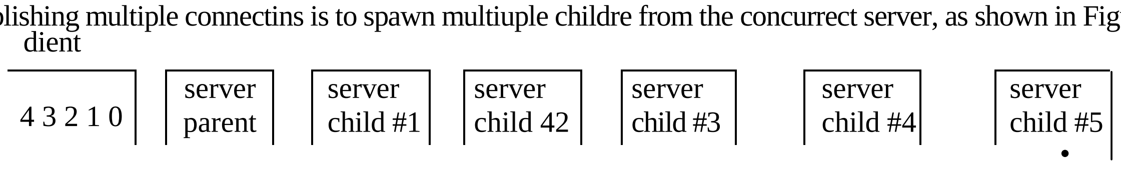


Figure 1. I Client with five established connections to same cone wrent server

tepeliserv/tepeliO4.e

```
#include "unp.h"
int
main(int argc,      char      "- *argv)
{
    int            sockfd [5] ;
    struct sockaddrj     servaddr;
    if      (argc != 2)
      err_quit ("usage: tcpcli < I Paddress>" );
      for      (i = 0; i <5; i ++) {
            sockfd [i] = Socket (AF_INET, SOCK_STREAM, 0);
            bzero(&servaddr, sizeof(servaddr) );
            servaddr.sin family =     INET;
            servaddr.sin port = htons(SERV_PORT) ;
            'net pton(AF _I NET, argvill, &servaddr.sin addr);
            connect (sockfd[i], (SA *) &sel addr, sizeokservaddr));
      1
      str_cli (stdin, sockfd [0] ); /*do it all" /
     exit (0);
}
```

tcpcliservitcpcliO4.c

Figure1.2 TCP client that estabishes five connections with server

When the client terminates, all open desctiptions are closed automatically by the kernel (we do not call close, only exit) , and all five connections are terminated at about the same time. This causes five FINs to be sent, one on each connection, which in turn causes all five server children to terminate at about the same time. This causes five SIGCPLD signals to be delivered to the parent at about the same time, which we show in Figure 1.3.

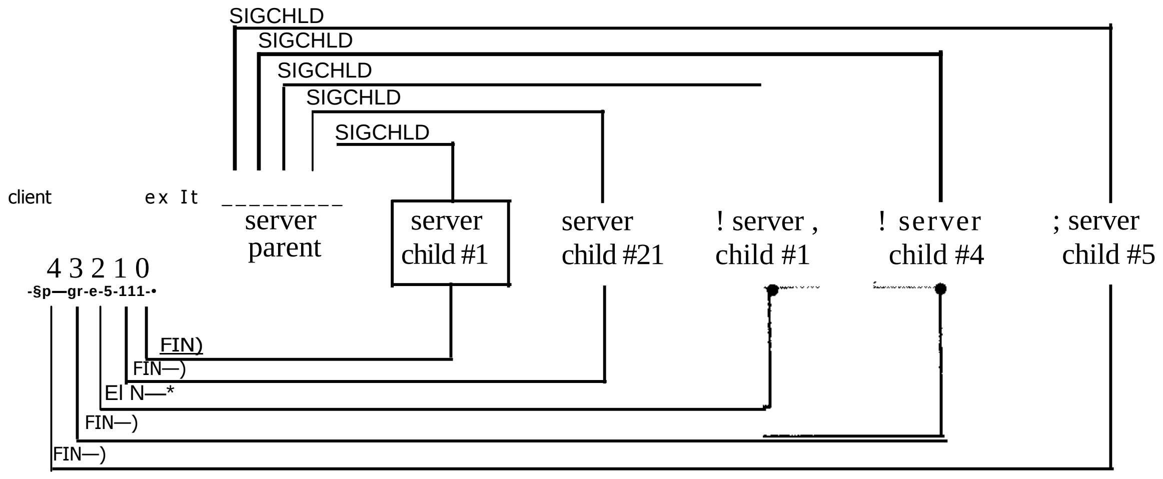


Figure 1.3 Client terminates, closing all Jive connections, terminating all five children

It is this delivery of multiple occurences of the same signal that causes the problem we are about to see. We first run the server in the background and then our new client.

```
bsdi % tcpserv03 &
[1]        21282
bsdi % tcpcl iO3 206.62.226.35
hello                                   we type this
hello                                   and it is echoed
AD                                      we then type our EOF character
child 21288 terminated                  output by server
```

The first thing we notice is that only one printf is output, when we expect all five children to have terminated. If we execute ps , we see that the other four children still exist as zombies.

```
  RI 11) TT STAT TIME COMMAND
21282 p1 s      0:00.09 it:lose/v(13
21281. pl z     0:00.00 (tcpserv03)
212E5 pl z      0:C0.00 (tcpserv03)
212E6 pl z      0:00.00 (tcpserv03)
21287 pl z      0:00.00 (tcpserv03)
```

Establishing a signal handler and calling wait from that handler are insufficient for preventing zombies. The problem is that all five signals are genereated before the signal handler is executed, and the signal handler is executed only one time because Unix signals are normally not *queued*. Furthermore this problem is nondeterministic. In the example we just ran, with the client and server on the same host, the signal handler is executed once, leaving four zombies. But if we run the client and server on different hosts, the signal handler is called only one more time. This leaves three zombies. But sometimes, probably dependent on the timing of the FINs arriving at the server host, the signal handler is executed three or even four times.

The correct solution is to call wa itpid instead of wait Figure 1.4 shows the version of our sig_ch Id function that handles S I GCHLD correctly. The version works because we can waitpid within a loop, fetching the status of any of our children that have terminated. We must specify the WNOHANG option: this tells waitpid not to block if there exist running children that have not yet terminated.

Figure 1.5 shows the final version of our server. It correctly handles a return of El NTR from accept and it

```
#include "unp.h"

void
sig_chld (int signo)
{
     pid_t pid;
    int     stat;

    while ( ( pid = waitpid (-1 &star, WNOHANG )) > 0)
          printf ("child %d terminated \ n", pid);
    return ;
}
```

tcpcliserv/sigchldwaitpid.c

Figure 1.4 Final (correct) version of **sig_chld** function that calls **waitpid**

tcpcliserv/tcpserv04.c

```
# include "unp.h"

hit
main ant argc, char ** argv)
{
    int     listenfd, connfd;
    pid_t childpid;
    socklen_t clilen;
    struct sockaddr_in cliaddr, servaddr;
    void sig_chld (int);

    listenfd = Socket (AF_IN ET, SOCK_STREAM, 0);

    bzero (&servaddr, sizeof (sdervaddr)) ;
     servaddr.siniamily = AF_INET;
    servaddr.sin_addr.s_addr = htonl (INADDR_ANY);
    servaddrsin_port = htons (SERV_PORT) ;

    Bind (listenfd, (SA *) &servaddr, sizeof (servaddr)) ;

    Listen (listenfd, LISTENQ);

     Signal (SIGCHLD, sig_chld);     */ must call waitpid ( ) *\
    for ( ; ; ) {
        clilen = sizeof (cliaddr) ;
        if ( (connfd = accept (listenfd, (SA*) &cliaddr, &clilen ) ) <0) {
           if        (erron = = EINTR)
               continue;        /* back to for ( ) */
           else
             err_sys ("acept error") ;

        if ((childpid = Fork () ) = = 0)1 *child process */
          Close (litenfd);        /* close listening socket */
          str echo (connfd) ; / * process the request*/
          exit (0);
```

again or not. In the case of the ECCON NA BO RTED error, the server can ignore the error and just can accept again.

3.3 Termination of Server Process

We now start now client-server and then kill the server child process. This simulates the crashing of the server process, so we can see what happens to the client. (We must be careful to distingusish between the chashing of the server *process,* which we are about to describe, and the crashing of the server *host,* which *we* describe in Section 1.7.) The following steps take place:

I. We start the server and client on different hosts and type one line to the client to verify that all is OK. That line is echoed normally by the server child.
2. We find the process ID of the server child and *kill* it. As part of process termination all open descriptors in the child are closed. This causes a FIN to be sent to the client, and the client TCP responds with an ACK. This is the first half of the TCP connection termination.
3. The S I GCH LD signal is sent to the server parent and handled correctly (Figure 1.5).
4. Nothing happens at the client. The client TCP reveives the FIN from the server TCP, and responds with an ACK, but the problem is that the client process is blocked in the call to fgets waiting for a line from the terminal.
5. Running netstat at this point from another window on the client shows the state of the client socket.

```
solaris % netstat I grep 9877
 Local Address Remote Address Swind Send-Q Rwind Recv-Q State
solaris.34673        bsdi.9877           8760         0 8760          0 CLOSE WAIT
```

(This is the first time we have shown thenetstat output from Solaris so we have added the headling line. This format is slightly different from the BSD output, byt the information is similar.) We also run netstat from another window on the server.

```
bsdi % netstat I     grep 9877
 tcp           0       0 bsdi.9877        sola ris.34673       F I N_WAIT_2
```

6. We can still type a line of input to the client. Here is what happens at the client starting from step I.

```
solaris %tcpcli01 206.62.226.35        start client
 hello                                  the first line that we type
 hello                                 it is echoed correctly
                                       here we kill the server child on the server host
another line                            we then type a second line to the client
str_cli: server terminated permaturely
```

When we type "another line", str_c I i calls write n and the client TCP sends the data to the server. This is allowed by TCP because the receipt of the FIN by the client TCP only indicates that the server process has closed its end of the connection and will not be sending any more data. The receipt of the FIN does *not* tell the client TCP that the server process has terminated (which in this case it has).

When the server TCP reveives the data from the client, it responds with an RST since the process that had that socket open has terminated. We can verify that the RST is sent by watching the packets with tcpdump.

7. But the client process will not see the RST because it calls readl I ne immediately after the call
to writen and readline returns 0 (end-of-line) immediately because of the FIN that was reveived in step 2. Our client is not expecting to reveive an end-of-file at this point so it quits with the error message "server terminated prematurely."

8. When the client terminates, all its open descriptors are closed.

The problem in this example is that the client is blocked in the call to fgets when the FIN arrives on the socket. The client is really working with two descriptors—the socket and the user input—and instead of blocking on input from only one of the two sources (as str_c I i is currently coded), it should block on input from either source. Indeed, this is one purpose of the select and poll functions.

14 SIGPIPE Signal

What happens if the client ignores the error return from readline and writes more data to the server? This can happen, for example, if the client needs to perform two writes to the server before reading anything back, with the first write eliciting the RST.

The rule that applies is:when a process writes to a socket that has received an RST, the SIGPIPE signal is sent to the process. The default action of this signal is to terminat the process so the process must catch the signal to avoid being involuntarily terminated.

If the process either catches the signal and returns from the signal handler, or ignores the signal, the write operations returns EP I PE.

To see what happens with SIGPIPE we modify our client as shown in Figure 1.7

tcpcliserv/str_clill.c

```
# include "unp.h"
void
str_cli (FILE xfp, int sockfd)
{
    char send[MAXLINE], recvline[MAXLINE];
    while (Fgets(sendline, MAXLINE, fp) != NULL) {
        writen(sockfd, sendline, 1) ;
         sleep(1);
         writen(sockfd, sendline + 1, strlen(sendline) - 1)
        if (Readline(sockfd, recvline, MAXLI NE) == 0)
           err quirstr_cli: server terminated prematurely");

        Fputs(recvline, stdout);
    1
1
```

tcpcliserv/str_clil 1.c

All we have changed is to call two times: the first time the first byte of data is to the socket, followed by a pause of 1 second, followed by the remainder of the line. The intent is for the first to elecit the RST and then for the second writen to generate SIGH PE

[illegible]

```
bsdi % tcpcl ill 206.62.226.34
hi there                          we v.vpe this line
hi there                          this is echoed by the server
                                  here we kill the server child
b   y   e                         then we type this line
bsdi % echo $?                    what is the Kornshell's return value of last command?
Zd9                          269 = 256 + 13
bsdi % grep SIGPIPE /usr/include/sysd/signal.h
#define SIGPIPE 13            /*write on a pipe with no one to read it* /
```

We start the client, type in one line, see that line echoed correctly, and then terminate the server child on the server host. We then type another line ("bye") but nothing is echoed and we just get a shell prompt. Since the default action of SI GRIPE is to terminate the process without generating a core file, nothing is printed by the KomShell. This is the problem with programs terminated by SI GRIPE: normally nothing is outpout even by the shell to indicate what has happened.

We musat execute echo $? to point the shell's return value, which is 269. We then print the numeric value of the constant SI G PI PE and see that the KomShell's return value is 256 plue the signal number. But if we execute this program under digital Unix 4.0, Solaris 2.5, or UnixWare 2.1.2, the KomShell's return value is 141, or 128 plus 13.

The recommended way to handle SIGPIPE depends on what the application wants to do when this occurs. If there is nothing special to do, then setting the signal disposition to S I G G N is easy, assuming that sebsequent output opeartions will cathc the error of and terminate. If special actions are needed when the signal occurs (writing to a log file perhaps), then the signal should be caught and any desired actions can be performed in the signal handler. Be aware, however, that if multiple sockets are in use, the delivery of the signal does not tell us which socket encoutered the error. If we need to know which write caused the error, then we must either ignore the signal or return from the signal handler and handle EP I PE from the `write`.

3.5 Crashing of Server Host

Our next scenario is to see what happens when the server host crashes. To simulate this we must run the client and server on different hosts. We then start the server, start the client, type in a line to the client to verify that the connection is up, disconnect the server host from the network, and type in another line at the client. This also covers the scenario of the server host being unreachable when the client sends data (i.e., some intermediate router is down after the conneciton has been established).

The following steps take place:

I. When the sever host crashes, nothing is sent out on the existing network connections. That is, we are assuming the host crashes, and is not shut down by an operator.

2. We type a line of input to the client, it is `writen` by writen, and is sent by the client TCP as a data segment. The client then blocks in the call to `read line,` waiting for the echoed reply.

3. If we watch the network with `tcpdump,` we will see the client TCP continually retransmits the data segment, trying to receive an ACK from the server. When the client TCP finally gives up (assuming the server host has not been rebooted during this time, or if the server host has not crashed but was unreachable on the network, assuming the host was still unreachable), an error is returned to the client process. Since the client is blocked in the call to `readline,` it returns an error.

   Assuming the server host had crashed and there were no responses at all to the client's data segments, the error is ET I M EDOUT. But if some intermediate router determined that the server host was un-reach able the responded with an ICMP destination unreachable message, the error is either EHOSTUNREACH or EN ETU N REACH .

Although our client discovers (eventually) that the peer is down or unreachable, there are times when we want to detect this quicker than having to wait 9 munutes. The solution is to place a timeout on the call or readline.

The scenario that we just discussed detects that the server host has crashed only when we send data to that host. If we want to detect the crashing of the server host even if we are not actively sending it data, another technique is required.

3.6 Crashing and Rebooting of Server Host

In this scenario we establish the conneciton between the client and server and then assume the serve host crashes and reboots. In the previous section the server host was still down when we sent it data. Here we will let the server host reboot before sending it data. The easiest way to simulate this is to establish the connection, disconnect the server from the network, shut down the sever host and then reboot it, and then reconnect the server host to the network. We do not want the client to see the server host shut down (which we cover in Section 3.7).

As stated in the previous section, if the client is not actively sending data to the server when the server host crashes, the client is not aware that the server host has crashed. (This assumes we are not using the SO_KEEPALIVE socket option). The following steps take place:

I. We start the server and then the client. We type a line to verify that the connection is established.
2. The server host carahes and reboots.
3. We type a line of input to the client, which is sent as a TCP data segment to the server host.
4. When the server host reboots after crashing, its TCP loses all information about connections that existed before the crash. Therefore the server TCP responds to the reveived data segment from the client with an RST.
5. Our client is blocked in the call to readline when the RST is reveived, causing readline to return the error ECONNRESET.

If it is important for our client to detect the crashing of the server host, even if the client is not actively sending data, then some other technique (such as the SO_KEEPALIVE socket option or some client-server heartbeat fimciton) is required.

3.7 Shutdown of Server Host

The previous two sections discussed the crashing of the server host, or the server host being unreadable across the network. We now consider what happens if the server host is shut down by an operator while our server process is running on that host.

When a UNix system is shut down, the i nit process normally sends the SI GTERM signal to all processes (we can catch this signal), waits some fixed amount of time (often between 5 and 20 seconds), and then sends the SIGKILL signal (which we cannot catch) to any processes still running. This gives all runing process a short amount of time to clean up and terminate. If we do not catch S GT ER M and terminate, our server will be terminated by the SIGKILL signal. When the process terminates, all open descriptors are closed, and we then follow the same sequence of steps discussed in Section 3.3. As we stated there, we must use the select or pool function in our client to have the client detect the termination of the server process as soon as it occurs.

3.8 Summary of TCP Example

Before any TCP client and server can communicate with each other, each end and must specify the socket pair for the connection: the local IP address, local port, foreign IP address, and foreign port. This figure is from the client's perspective. The foreign IP address and foreign port must be specified by the client in the call to connect. The two local values are normally chosen by the kernel as part of the co n nectfunction. The client has the option of specifing either or both of the local values, by calling bind before connect, but this is

The client can obtain the two local values chosen by the kernel by calling getsockname after trite connection is established.

4.0 Conclusion

In this unit, you have learned about the issues of calls and how they can be interrupted. You also learn about the read, write, select and open operations. These techniques are scheme will enable you to appreciate how to crash, reboot, adn shutdown a host or server.

Mary

What you have leaned in this unit focuses on the calls interruption and how they are handled. You also learn about how to crash, reboot, and shutdown a server. The units that follow shall build upon these issues.

• :·4sPgnwnt..:

a) Explain the protocols for connection Abort.

Exercises 1.2

Discuss the SIGPIPE Signal

Exercise 1.1

Differentiate between wait adn waitpid

IS:.:.:::•:Refeettects. and Other Resources'.

Stevens, W.R. Unix Network Programming, (2nd ed) Vol. 1, Pentice Hall, PTR,1998.

Module 3: Overview of Network Programming

Unit 8 I/O Multiplexing : The select and Poll Fu ictious

Page

thit

In this unit, you will learn about the meaning of I10 Multiplexing and how the select and poll functions are used to accomplish multiplexing. You will also learn how multiple descriptors can be handled through multiplexing. Let us now see what you will learn on this unit as stated in the unit objectives below.

2,0 Objectives

By the end of this unit, you should be able to:

- understand why I/0 multiplexing is necessary
- accomplish WO multiplexing through select and poll functions
- understand how I10 multiplexing handles UDP and TCP connections simultaneously.

We saw in Section 3.3 (Unit 7) that our TCP client is handling two inputs at the same time: standard input and a TCP socket. The problem we encountered was when the client was blocked in a call to read (by calling our readline function), and the server process was killed. The server TCP correctly sends a FIN to the client TCP, but since the client process is blocked reading from standard input, it never sees the end-of-file unit it reads from the socket (possibly much later in time). What we need is the capability to tell the kernel that we want to be notified if one or more I10 conditions are ready (i.e. input is ready to be read, or the descriptor is capable of taking more output). This capability is called *I/O Multiplexing* and is provided by the select and poll functions. We also cover a newer Posix.Ig variation of the former, called pselect.

I10 multiplexing is typically used in networking applications in the following scenarios:

When a client is handling multiple descriptors (normally interactive input and a network socket), 110 multiplexing should be used. This is the scenario we described in the previous paragraph.

It is possible, but rare, for a client to handle multiple sockets at the same time. We show an example of this using select in the context of a Web client.

If a TCP server handles both a listening socket and its connected sockets, I/O multiplexing is nor mally used.

- If a server handles both TCP and UDP, 110 multiplexing is normally used.
- If a server handles multiple services and perhaps multiple protocols, I10 multiplexing is normally used.

I/0 multiplexing is not limited to network programme. Any nontrivial application often finds a need to use this technique.

3.1 I10 Models

Before describing select and poll we need to step back and look at the bigger picture, examining the base difference in the five YO models that are available to us under Unix:

- blocking I/0,
- nonblocking I/O,
- 1/0 multiplexing (select and poll),
- signal driven I/0 (SIG I 0), and
- asynchronous I/O (the Posix 1 a io _functions.)

You may want to skim this section on your first reading and then refer back to it as you encounter the different I10 models described in more detail in later units

As we show in all the examples in this section, there are normally two distinct phases for an input operation:

1. waiting for the data to be ready, and
2. copying the data from the kernel to the process.

For an input operation on a socket the first step normally involves waiting for data to arrive on the network. When the packet arrives, it is copied into a buffer within the kernel. The second step in copying this data from the kernel's buffer into our application buffer.

Blocking I/O Model

The most prevalent model for I10 is the *blocking I/O model,* which we have used for all our examples so far in the text. By default, all sockets are blocking. Using a datagram socket for our examples we have the scenario shown in Figure 1.1.

We use UDP for this example, instead of TCP, because with UDP the concept of data being "ready" to read is simple either an entire datagram has been received or not. With TCP it gets more complicated, as additional variables, such as the socket's low-water mark, come into play.

In the examples in this section we also refer to recvfrom as a system call, because we are differentiating between our application and the kernel. Regardless of how recv from is implemented (as a system call on a Berkeley-derived kernel, or as a function that invokes the get msg system call on a System V kernel), there is normally a switch from running in the application to running in the kernel followed at some time later by a return to the application.

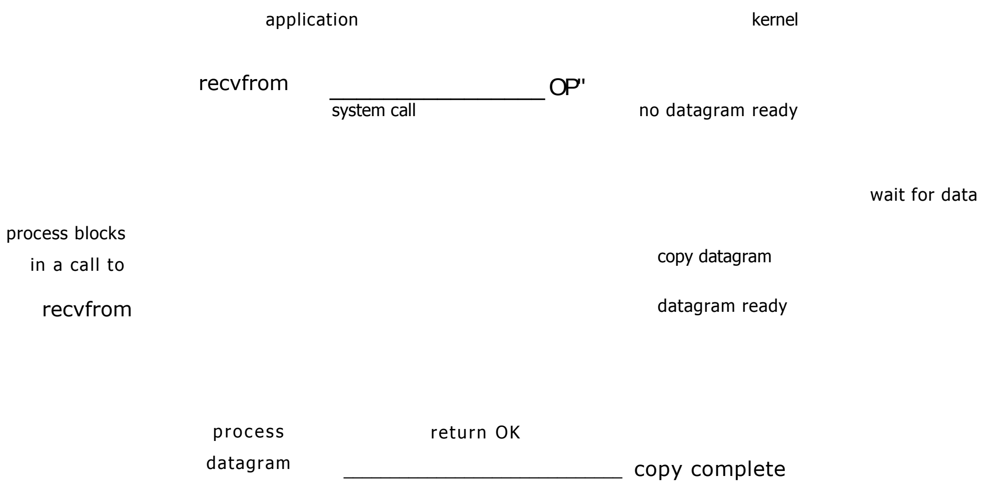


Figure 1.1 Blocking I/O model

In Figure 1.1. the process calls recvfrom and the system call does not return until the datagram arrives and is copied into our application buffer, or an error occurs. The most common error is the system call being interrupted by a signal. We say that our process is *blocked* the entire time from when it calls recvfrom until it returns. When recvfrom returns OK, our application processes the datagram.

Nonblocking I/O Model

When we set a socket nonblocking, we are telling the kernel "when an I10 operation that I request cannot be completed without putting the process to sleep, do not put the process to sleep but return an error instead."

The first three times that we call recvfrom, there is no data to return, so the kernel immediately returns an error of EWOU LD B LOCK instead. The fourth time we call recvfrom a datagiam is ready, it is copied into our application buffer, and recvfrom returns OK. We then process the data.

When an application sits in a loop calling recvfrom on a nonblocicing descriptor like this, it is *calledpolling.* The application is continually polling the kernel to see if some operation is ready. This is often a waste of CPU time, but this model is occasionally encountered, normally on systems dedicated to one function.

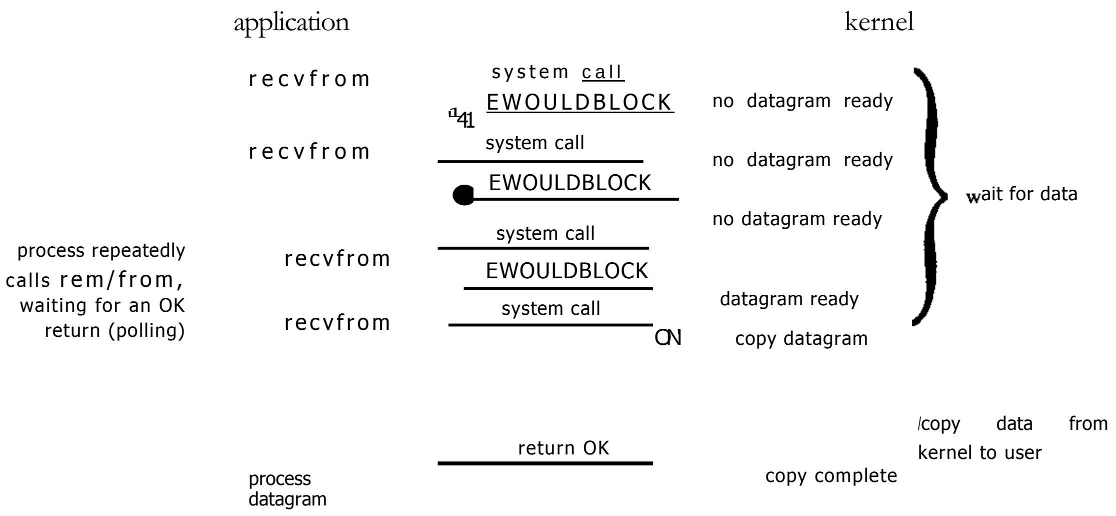


Figure 1.2 Non blocking I/O model

I/O MultiplexingWith *I/O multiplexing, we call* select or poll and block in one of these two system calls, instead of blocking in the actual 110 system call. Figure 1.3 is a summary of the 110 multiplexing model..

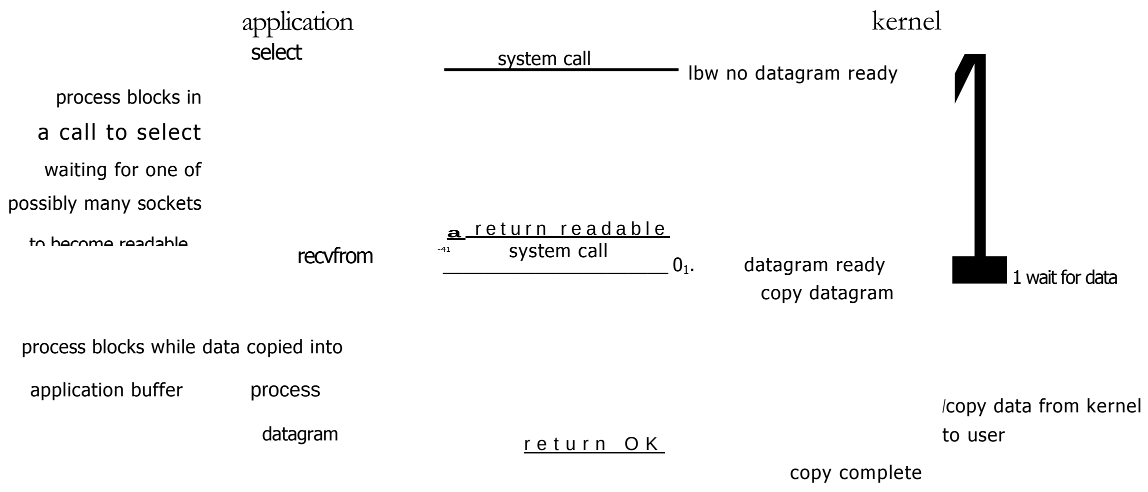


Figure 1.3 I/O multiplexing modeL

We block in a call to select, waiting for the datagram socket to be readable. When select returns that the socket is readable, we then call recvfrom to copy the datagram into our application buffer.

Comparing Figure 1.3 to Figure 11, there does not appear to be any advantage, and in fact there is a slight disadvantage because using select requires two system calls instead of one. But the advantage in using select, which we will see later in this chapter, is that we can wait for more than one descriptor to be ready.

Signal Driven I10 Model

We can also use signals, telling the kernel to notify us with the S I GIO signal when the descriptor is ready. We call this *signal driven I/O* and shown a summary of it in Figure. 1.4.

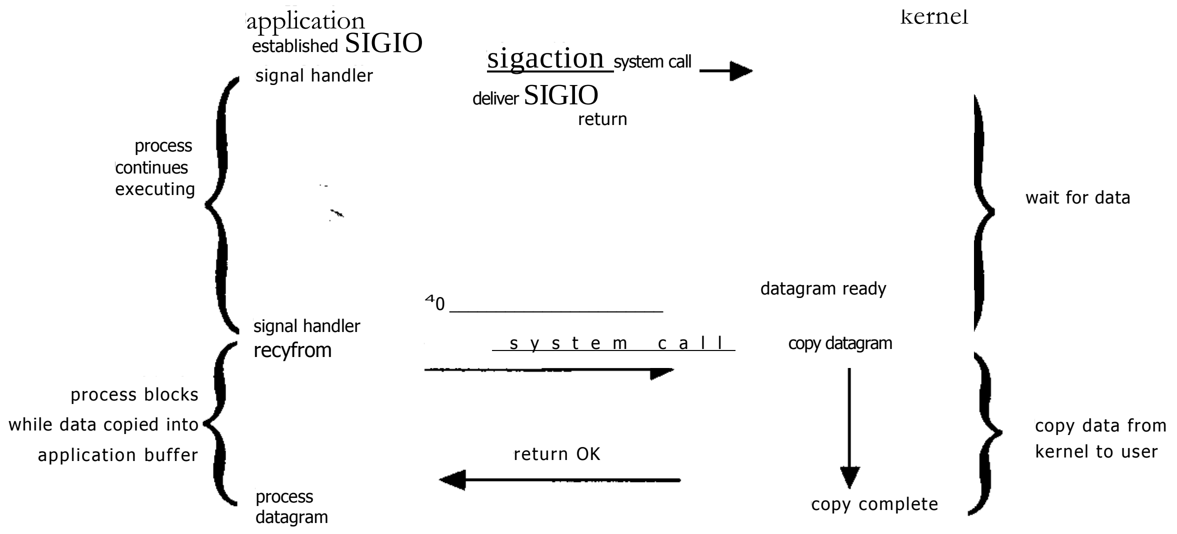


Figure 1.4 Signal driven I/O model

We first enable the socket for signal driven 110 and install a signal handler using the sigaction system call. The return from this system call is immediate and our process continues, it is not blocked. When the datagram is ready to be read, the SIGIO signal is generated for our process. We can either read the datagram \ffom the signal handler by calling remffrom and then notify the main loop that the data is ready to be processed or we can notify the main loop and let it read the datagram.

Regardless of how we handle the signal, the advantage in this model is that we are not blocked while waiting for the datagram to arrive. The main loop can continue executing and just wait to be notified by the signal handler that either the data is read to process or that the datagram is ready to be read.

Asynchronous I/O Model

Asynchronous I/O is new with the 1993 edition of Posix 1 (the "realtime" extensions). We tell the kernel to start the operation and to notify us when the entire operation (including the copy of the data from the kernel to our buffer) is complete. We do not discuss it in this book because it is not yet widespread. The main difference between this model and the signal driven 110 model in the previous section is that with signal
driven VO the kernel tells us when an I10 operation can be *initiated*, but with asynchronous 110 the

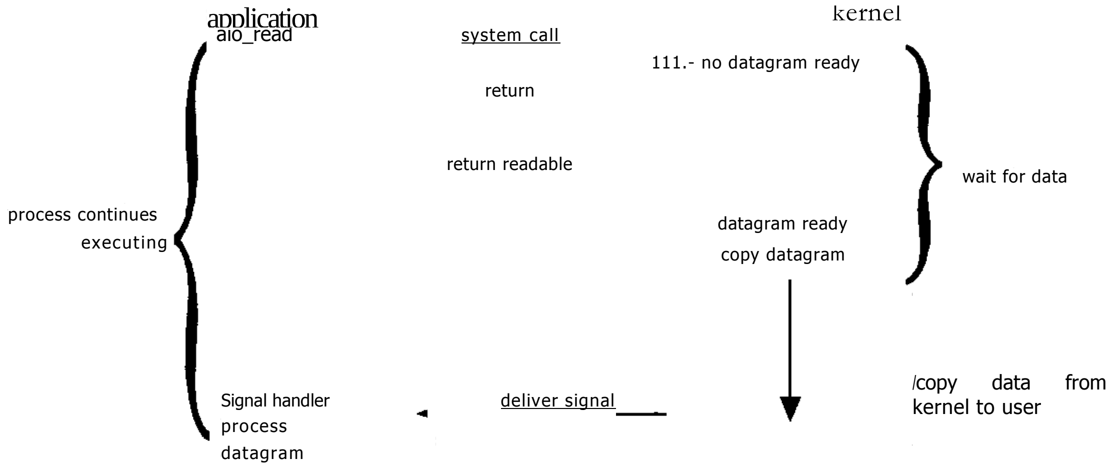


Figure 1.5 Asynchronous I/O model here

We call a io_read (the Posix asynchronous I/O functions begin with a i o_ or I io_) and pass the kernel the descriptor, buffer pointer, buffer size (the same three arguments for read), file offset (similar to I seek), and how to notify us when the entire operation is complete. This system call returns immediately and our process is not blocked waiting for the I/O to complete. We assume in this example that we ask the kernel to generate some signal when the operation is complete. This signal is not generated until the data has been copied into our application buffer, which is different from the signal driven I10 model.

Comparison of the I/0 Models

Figure 1.6 is a comparison of the five different I10 models. This shows that the main difference between the first four models is the first phase, as the second phase in the first four models is the same: the process is blocked in a call to recvfrom while the data is copied from the kernel to the caller's bufferAsynchronous 1/ 0, however, handles both phases and is different from the first four..

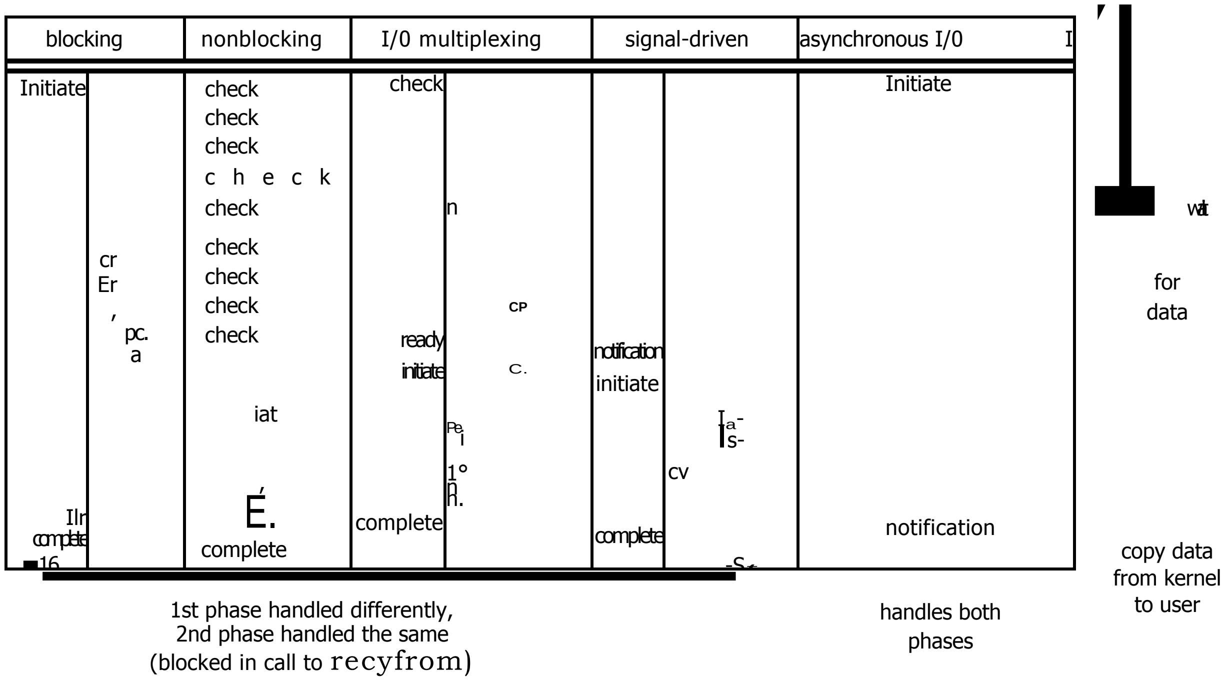


Figure 1.6 Comparison of the five I/O models

Synchronous I/O versus Asynchronous I/0

Posix. 1 defines these two terms as follows:

- A *synchronous I/O operation* causes the requesting process to be blocked until that 1/0 operation completes.
- An *asynchronous I/O operation* does not cause the requesting process to be blocked.

Using these definitions the first four 1/0 models — blocking, nonblocking, I/0 multiplexing, and signal-driven I/0 — are all synchronous because the actual I/O operation (recvfrom) blocks the process. Only the asynchronous I10 model matches the asynchronous I/0 definition.

3.2 Select Function

This function allows the process to instruct the kernel to wait for any one of multiple events to occur and to wake up the process only when one or more of these events occurs or when a specified amount of time has passed.

As an example, we can call select and tell the kernel to return only when

- any of the descriptors in the set (1, 4, 5) are ready for reading, or
- any of the descriptors in the set (2, 7) are ready for writing, or
- any of the descriptors in the set (1, 4) have an exception condition pending, or
- after 10.2 seconds have elapsed.

That is, we tell the kernel what descriptors we were interested in (for reading, writing, or an exception condition) and how long to wait. The descriptors in which we are interested are not restricted to sockets: any descriptor can be tested using select.

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


      int select (int maxfdpl, fd_set *readset, fd_set *writeset, fd_set
          *exceptset, const struct timeval *timeout) ;

                                      Returns: positive count of ready descriptors, 0 on
```

We start our description of this function with its final argument, which tells the kernel how long to wait for one of the specified descriptors to become ready. A timeva I structure specifies the number of seconds and microseconds.

```
structtimeval(
  long tv_sec        /* seconds */
  long tv_sec:        /*microseconds */
   ;
```

There are three possibilities.

1. Wait forever: return only when one of the specified descriptors is ready for UO. For this, we specify the *timeout* argument as a null pointer.
2. Wait up to a fixed amount of time: return when one of the specified descriptors is ready for I/O, but do not wait beyond the number of seconds and microseconds specified in the timeval structure pointed to by the *timeout* argument.
3. Do not wait at all: return immediately after checking the descriptors. This is called *polling.* To specify this, the *timeout* argument must point to a timeval structure, and the timer value (the number of seconds and microseconds specified by the structure) must be O.

The wait in the first two senarios is normally interrupted if the process catches a signal and returns from the signal handler.

Although the tinieval structure lets us specify a resolution in microseconds, the actual resolution supported by the kernel is often more coarse. For example, many Unix kernels round the timeout value up to a multiple of 10 ms. There is also a scheduling latency involved, meaning it take some time after the timer expires before the kernel schedules this process to run.

The c nst qualifier on the *timeout* argument means it is not rhodified by select on return. For example, if we specify a time limit of 10 seconds, and select returns before the timer expires, with one or more of the descriptors ready or with an error of El NTR, the timeval structure is not updated with the number of seconds remaining when the function returns. If we wish to know this value, we must obtain the system time before calling select, and then again when it returns, and substract the two.

The three middle arguments *readset, writeset,* and *exceptset* specify the descriptors that we want the kernel to test for reading, writing, and exception conditions. There are only two exception conditions currently supported.

1. The arrival of out-of-band data for a socket.
2. The presence of control status information to be read from the master side of a pseudo terminal that has been put into packet mode. We do not talk about pseudo terminals in this volume.

A design problem is how to specify one or more descriptor values for each of these three arguments select uses *descriptor sets,* typically an array of integers, with each bit in each integer corresponding to a descriptor. For example, using 32-bit integers, the first element of the array corresponds to descriptors 0 through 31, the second element of the array corresponds to descriptors 32 through 63, and so on. All the implementation details are irrelevant to the application and are hidden in the fd_set datatype and the following four macros:

```
void FD•ZERO(fd_set *fdset):                /*clear all bits infdset*/
void FD_S ET (i nt.fd ,fd_set "ildset):     /*turn on the bit for fd i nfdset
void FD_CLR (intfd ,fd_set *fthet):         /* turn offthe bit for fd infdset*/
int FD_I SS ET° nt fd , fd_set *fdset):     /* is the bit forp on infdset? */
```

We allocate a descriptor set of the fd_set datatype, we set and test the bits in the set using these macros, and we can also assign it to another descriptor set across an equals sign in C.

For example, to define a variable of type fd_set and then turn on the bits for descriptors 1,4, and 5, we write.

```
FD_ZERO (Sir set) ;          /* initialize the set: all bits off */
FD_SET (1, &rset);           /*turn on bit for fd 1 */
FD_S ET (4, &rset);          /*turn on bit forfd 4*/
FD_SET (5, &rset);           /*turn on bit for fd 5 */
```

It is important to initialise the set, since unpredictable results can occur if the set is allocated as an automatic variable and not initialised.

Any of the middle three arguments to select, *readset, writeset, or exceptset,* can be specified as a null pointer, if we are not interested in that condition. Indeed, if all three pointers are null, then we have a higher precision timer than the normal Unix sleep function (which sleeps for multiples of a second). The poll function provides similar functionality. Figures C.9 and C.10 of APUE show a sl eep_us function implemented using both select and pole that sleeps for multiples of a microsecond.

The *maxfdpl* argument specifies the number of descriptors to be tested. Its value is the maximum descriptor to be tested, plus one (hence our name of *maxfdpl).* The descriptors 0,1,2, up through and including *maxfdpl -1* are tested. The constant FD_SETS I Z E, defined by including <sys/sel ect. h>, is number: of descriptor in the fd_set datatype. Its value is often 1024, but few programme use that many descriptors. The *maxfdpl* argument forces us to calculate the largest descriptor that we are interested in and then tell the kernel this value. For example, given the previous code that turns on the indicators for descriptors 1,4 and *5, maxfdpl* value is 6. The reasons it is 6 and not 5 is that we are specifying the number of descriptors, not the largest value, and descriptors start at 0.

select modifies the descriptor sets pointed to by the *readset, writeset,* and *exceptset* pointers. These three arguments are value-result arguments. When we call the function, we specify the values of the descriptors that we are interested in and on return the result indicates which descriptors are ready. We use the FD_ISSET macro on return to test a specific descriptor in an fd_set structure. Any descriptor that is not ready on return will have its corresponding bit cleared in the descriptor set. To handle this we turn on all the bits in which we are interested in all the descriptor sets each time we call select

3.3 str_cl i Function (Revisited)

We can now rewrite our st r_c I i function from Section 3.4 (Unit 6), this time using select , so we are notified as soon as the server process terminates. The problem with that earlier version was that we could be blocked in the call to fgets when something happened on the socket. Our new version blocks in a call to select instead, waiting for either standard input or the socket to be readable. Figure 1.7 shows the various conditions that re handled by our call to select.

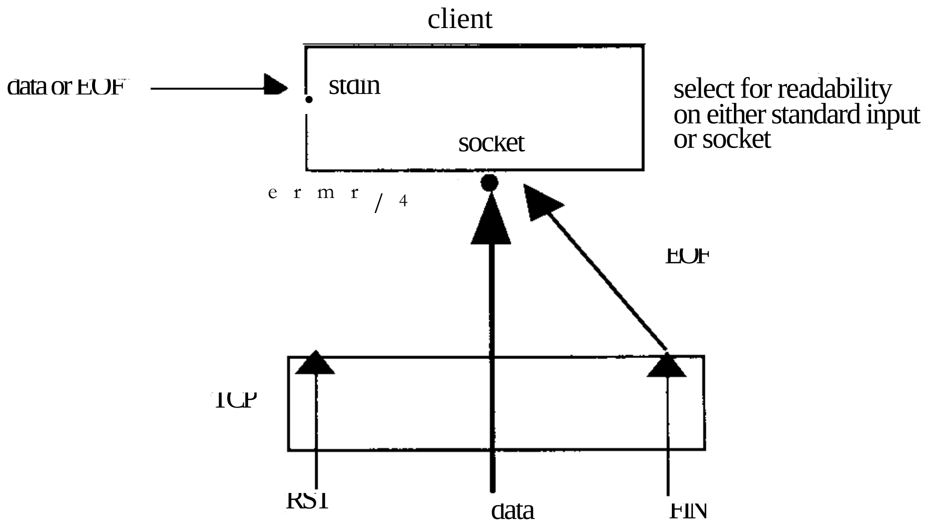


Figure 1.7 Condition handled by select in strc I i

Three conditions are handled with the socket.

1. If the peer TCP sends data, the socket becomes readable and read returns greater than 0 (i.e. the number of bytes of data).
2. If the peer (TCP) sends of FIN (the peer process terminates), the socket becomes readable and read returns 0 (end-of-file).
3. If the peer TCP sends an RST (the peer host has crashed and rebooted), the socket be comes readable and read returns -1 and errno contains the specific error code.

Figure 1.8 shows the source code for this new version.

Call select

We only need one descriptor set — to check for readability. This set is initialized by FD_ZERO and then two bits are turned on using F D_S ET: the bit corresponding to the standard I/0 file pointer fp and the bit corresponding to the socket soc kfd. The function fi len 0 converts a standard 1)0 file pointer into its corresponding descriptor,select (and poll) work only with descriptors.

select is called after calculating the maximum of the two descriptors. In the call the write-set pointer and the exception-set pointer are both null pointers. The final argument (the time limit) is also a null pointer since we want the call to block until something is ready.

Handle readable socket

If, on return from sel ect,the socket is readable, the echoed line is read with readline and output by fputs.

5elect/strcliselect01.c

```
# include "unp.h"
void
str_cli (FILE *fp, int sockfd)
{
        int      maxfdl ;
        fd_set     rset ;
       char sendline[MAXLI NE], recvline[MAXLINE];

       FD_ZER09&rset);
       for(; ;
           FD_SET(fileno(fp), &rset);
            FD_SET(sockfd, &rset); ;
           maxfdpl = max(fileno(fp), sockfd)+ 1;
           select(maxfdpl, &rset, NULL, NULL, NULL);

           if (FD_ISSET(sockfd, &rset)){ /"- .ocket is readable*/
            if (Readline(sockfd, recvline, MAXLINE) == 0)
              err_quirstr_cli: server terminated prematurely");
           Fputs(recvline, stdout);

       if (FD_ISSET(fileno(fp), &rset)) /*input is readable*/
      if (Fgets(sendline, MAXLINE, fp) == NULL)
           return;           ['all done*/
      Writen(sockfd, sendline, strlen(sendline));
   }
}
}
```

select/strcliselect01.c

Figure 1.8 Implementation of str_ cli *function using*

Handle readable input

If the standard input is readable, a line is read by fgets and writen to the socket using written.

Notice that the same four I10 functions are used as in Figure 1.5 (unit 6): fgets, writen, read line, and fputs, but the order of flow within the function has changed. Instead of the function flow being driven by the call to fgets, it is now driven by the call to select. With only a few additional lines of code in Figure 1.8 compared to Figure 1.5 (unit6), we have added greatly to the robustness of our client.

3.4 Batch Input

Unfortunately, our str_cli function is still not correct. First lets go back to our original version, Figure 1.5 (unit 6). It operates in a stop-and-wait mode, which is fine for interactive use: it sends a line to the server and then waits for the reply. This amount of time is one RTT (round-trip time) plus the server's processing time (which is close to 0 for a simple echo server). We can therefore estimate how long it will take for a given **number** of lines to be echoed, if we know the RTT between the client and server.

6 elusion

In this unit, you have learned about the I/O multiplexing and how it can be accomplished through the select and poll functions. You also learn how multiple descriptors can be handled through 1/0 multiplexing. These schemes and techniques will now assist you to appreciate how TCP and UDP connections are handled simultaneously.

What you have learned in this unit is on the issue of I/O multiplexing and how it can be realised through select and poll functions. You also learn about the handling of multiple descriptors and finally the handling of TCP and UDP connections simultaneously. The next few units will build upon these concepts.

lulor ked ASsignmela

a Explain the term "110 Models" Describe the various types that you know.

Exercise 1.1

Discuss how str_c I i Functions works.

Exercise 1.2

Write on Batch Input

7.0;i:ReccenleCOS]and OtherResounes

Stevens, W.R. *Unix Network Programming,* (2nd ed.) vol. 1, Pentice Hall, PTR, 1998.

Module 3: Overview of Network Programming

Unit 9: Socket Options

SC **ttlit$.**

In this unit, you will learn about the two improtant functions used in socket programming i.e. sets oc ko pt and getsockopt. You will also learn about how to print the default value of all the options and various socket functions. Let us now see what you will learn in this unit as stated in the Unit Objectives below.

2.0 Objectives

By the end of this unit, you should be able to:

- describe the operations of two important socket options i.e. set soc kopt and getsoc kopt .
- describe how to print the default value of all options.

3.0 Synopsis

There are various ways to get and set the options that affect a socket:

- the setsockopt and getsockopt functions,
- the fontl function, and
- the ioctl function.

This chapter starts by covering the setsockopt and getsockopt. functions, followed by an example that prints the default value of all the options, followed by a detailed description of all of the socket options. We divided the detailed descriptions into the following categories: generic, IPv4, and TCP. This detailed coverage can be skipped during a first reading of this unit, and the individual sections referred to when needed. A few option **are** discussed in detail in a later unit, such as the IPv4 and IPv6 multicasting options.

We also describe the fcntl function, because it is the Posix way to set a socket for nonblocking I/O, signal-driven I/O, and to set the owner of a socket. We save the function for later unit.

3.1 getsockopt and setsockopt Functions

These two functions apply only to sockets.

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

int getsockopt(int sockfd, int level, int op/name, void *optval, sock len_t *opt/en);

int setsockopt (m nt sockfd, int level, i nt op/name, cont void *optva/
            soc k I e nt_t opt/en);
                                                  Both return: 0 if OK, -I an error
```

sockfd must refer to a open socket descriptor. The *level* specifies the code in the system to interpret the option: the general socket code, or some protocol-specific code (e.g., IPv4, IPv6, or TCP).
optival is a pointer to a variable from which the new value of the option is fetched by set soc kopt, or into which the current value of the option is stored by get socko pt.The size of this variable is specified by the final argument, as a value for s et soc ko ptand as a value-result for get socko pt.

Figurel .1 summarise the options that can be queried by getsoc kopt or set by set soc kopt. The "Datatype" column shows the datatype of what the *optival* pointer must point to for each option. We use the notation of two braces to indicate a structure, as in tinge {} to mean a struct I inger.

There are two basic types of options: binary options that enable or disable a certain feature (flags), and options that fetch and return specific values that we can either set or examine (values). The column label "Flag" specifies if the option is a flag option. When calling getsockopt for these flag options, *optival* is an integer. The value returned in *optival* is zero if the option is disabled, or nonzero if the option is enabled.

Similarly, setsockopt requires a nonzero optval to turn the option on, and a zero value to turn the option off. If the "Flag" column does not contain a "." then the option is used to pass a value of the specified datatype between the user process and the system.

Following sections of this chapter give additional details on the options that affect a socket.

3.2 Checking If an Option is Supported and Obtaining the Default

We now write a program to check whether most of the options defined in Figure 1.1 are supported, and if so, print their default value. Figure 1.2 contains the declarations for our program.

level	*Optname*	get	set	*Description*	*flag*	*Datatype* I
SOL_SOCKET	SO_BROADCAST	.	.	permit sending of broadcast datagrarns		
	SO_DEBUG		.	enable debug tracing		
	SO_DONTROUTE	.	.	by pass routing table lookup	.	
	SO_ERROR			get pending en¨or and clear		
	SO_KEEPALIVE	.	.	periodically test if connection still alive		
	SO_LINGER			linger on close if data to send	.	
	30_00BINLINE	.	.	leave received out-of-band data inline		
	SO_RCVBUF			receive buffer size		
	SO_SNDBUF	.	.	send buffer size		
	SO_RCVLOWAT			received buffer low-water mark		
	SO_SNDLOWAT	.	.	send buffer low-water mark		
	SO_RCVTI M EO			received timeout	.	
	SO_SNDTIME0	.	.	send timeout	.	Taro c
	SO_REUSEADDR			allow local address reuse		
	SO_REUSEPORT	.	.	allow local address resue		r
	SO_TYPE			get socket type		-
	SO_USELOOPBACK	.	.	routing socket gets copy of what is sends		
PROYOU P	P_HDRINCL	.	.	IF header included with data	.	lit
	P_OPTIONS	.		IF header options		(see text)
	P_RECVDSTADDR		.	return destination IF address	.	li
	P_RECVIF	.		return received interface index	.	t
	P_TOS	.	.	type-of-service and precedence		in
	P_TTL	.		time-to-live		t
	IP_MULTICAST_IF	.		specify outgoing interface		in_addrng
	IP_MULTICAST_TTL	.	.	specify outgoing TTL		u_char
	I P_MULTICAST_LOOP	.		specify loopback		u_char
	I P_ADD_MEMBERSHIP		.	join a multicast group		ip_mreq{}
	I P_DROP_MEMBERSHIP			leave a multicast group		ip_Mf610
IPPROTO _ICMPV6	ICMP6_FILTER	.	.	specify ICMPv6 message types to pass		cmp6_filter0
PROTO_IPV6	PV6_ADDR FORM	.	.	change address format of socket		int
	PV6_CHECKSUM	.	.	offset of checksum field for raw sockets		int
	PV6_DSTOPTS	.	.	receive destination options		
	PV6_DSTOPTS	.		receive destination options		int int
	PVE_HOPLI M IT	.		received unicast hop limit	.	int
	PV6_HOPOPTS	.	.	receive hop-by-hop options	.	sokaddr0
	PV6 NEXTHOP	.	.	specify next-hop address	.	int
	PV6_PKTINFO	.	.	receive packet information	.	(see text)
	PVG_PKTOPTIONS	.	.	specify packe options		
	PVG_RTHDR	.	.	receive source route	.	
	PV6_UN1CAST_HOPS		.	default uicast hop limit		

	I P V 6 _ M U L T I C A S T _ I P		.	specify outgoing interface		n6_addr0
	I P V 6 _ M U L T I C A S T _ H O P P ,	6	.	s p e c i f y o u t g o i n g b o p l i m i t		
	I P V G _ M U L T I C A S T _ L O O P	.	.	s p e c i f y l o o p b a c k		
	I P V 6 _ P D P I _ M E M B E R S H . P	•		join a multicast group		0.6_mr••
	I P V 6 _ P R O P _ M E M B I R S H I P	.		leave a multicast group		P•6-.¹,••
P P R O T O _ T C P	T C P _ K E E P P L I V E	.	.	seconds between keepalive probes		
	I C P _ M A X P T	.		TCP maximum retransmit time		inl
	I C P _ M A X S E G			TCP maximum segment size		int
	T C P . N O D E L A Y			d i s a b l e N a g l e a l g o r i t h m		
	T C P . S T O U P O	.		interpretation of urgent pointer	.	int

Figure 1.1 Summary of socket options for getsockopt and setsockopt.

sockopt/checkopts.c

```
# include          "unp.h"
# include          <fletinet/tcp.h>        /*for TCP_xxx defines*/

union val (
     int                    i_val;
     long                   l_val;
     char                   c_va I  [10];
     struct linger          linger_val;
     struct timeval         timeval_val;
   ) val ;

static char *stock_str_flag (union val x, int);
static char *sock_str-int (union val*, int);
static char *sock_str_linger (union val*, int);
static char*sock_str_timeval (union val*, int) ;

struct sock_opts (
     char           *opt_str ;
     int             opt_level ;
     int             opt name ;
     char      *(*opt_val_str) (union val *, int) ;
 )   sock_opts [ ] = (
        "SO_BROADCAST",          SOL_SOCKET,        SO BROADCAST        sock_str_flag,
        "SO_DEBUG",              SOL_SOCKET,        SO_DEBUG,            sock_str_flag,
        "SO_DONTROUTE",          SOL_SOCKET,        SO_DONTROUTE,        sock_str_flag,
        "SO_ERROR",              SOL_SOCKET,        SO_ERROR,            sock_str_int,
        "SO_KEEPALIVE",          SOL_SOCKET,        SO_KEEPALIVE,       sock_str_flag,
        "SO_LINGER",             SOL_SOCKET,        SO_LINGER,           sock_str_linger,
        "S0_00BINLINE",          SOL_SOCKET,        SO_OOBINLINE,       sock_str_flag,
        "SO_RCVBUF",             SOL_SOCKET,        SO_RCVBUF,          sock_str_int,
        "SO_SNDBUF",             SOL_SOCKET,        SO_SNDBUF,          sock_str_int,
         "SO_RCVLOWAT",          SOL_SOCKET,        SO_RCVLOWAT,        sock_str_int,
        "SO_SNDLOWAT",           SOL_SOCKET,        SO_SNDLOWAT,        sock_str_int
        "SO RCVTIMEO",           SOL SOCKET,        SO_RCVTIMEO,        sock_str_timeval,
        "SOISNDTIMEO",           SOL_SOCKET,        SO_SN DTI M E0,     sock_str_timeval,
         "SO_REUSEADDR",         SOL_SOCKET,        SO_REUSEADDR,       sock_str_flag,
#ifdef            SO_REUSEPORT
         "SO  REUSEPORT",        SOL  SOCKET,   SO_REUSEPORT,           sock_str_flag
#el se
    "SO_REUSEPORT",                                          NULL,
*tend if                   0,            0,
      "SO_TYPE",           SOL_SOCKET,  SO_TYPE,             sock_str int,
      "SO USELOOPBACK",    SOL_SOCKET, SO_USELOOPBACK,       sock_str_flag,
      "IP_TOS",             IPROTO_IP,   IP_TOS,             sock_str_int,
      "I P_TTL              I PROTOJ P   I P_TTL,            sock_str_int,
      "TCP_MAXSEG",        IPROTO_TCP, TCP_MAXSEG,           sock_str_int,
      "TCP_NODELAY",       IPROTO_TCP,  TCP_NODELAY,         sock_str_flag
      NULL                 0,           0,                   NULL
46);
```

sockopt/checkopts.c

Figure 1.2 Declarations for our program to check the socket options

Declare unionof possible value
Our union contains one member for each possible return value from getsockopt.

Define functions prototypes
We define function prototypes for four functions that are called to print the value for a given socket option.

Define structure and initialize array

Our sock_opts structure contains all the information necessary to call getsockopt for each socket option and **then** print its current value. The final member, opt_val_str, is a pointer to one of our four function that will print **the** option value. We allocate and initilise an array of these structures, one element for each socket option.

Figure 1.3 shows our main function.

sockopt/checkpts.c

```
47ñ
45 main(int argc, char 'mg)
43 (
53      int fd, len;
51      struct sock_opts*ptr ;
52      fd = Socket )AF_I N ET, SOCK_STREAM, 0);
        for (ptr = sock_opts; ptr->opt_str != NULL; ptr++) (
              printf M,s: ", ptr->opt_str);
              if (ptr>opt_val_str == NULL)
                        printf ("(underlined) n");
              else (
                        len = sizeof (val);
                                          if (getsockopt (fd, ptr->opt_level. ptr->opt_name,
                                                          &val, &len) == -1) (
                             err_ret ("getsockopt error");
                           else {
                             prinff (*default = %s n",(ptr->opt_val_str) (&val, len));
```

SEBfaMebBEBQSS4W

```
        }
    1
    exit (0);
(33 )
```

sockopt/checkopts.c

Figure 1.3 main function to check all socket options.

Create TCP socket, go to through all options

We create a TCP socket and then go through all elements in our array. If the pt_va l_str pointer is null, the option is not defined by the implementation (which we should is possible for SO_REUSEPORT).

Call getsockopt

we call get soc kopt but do not termine if an error is returned. Many implementations define some of the socket options names even through they do not support the option. Unsupported options should elicit an error of ENOPROTOOPT.

Print option's default value

If getsockopt returns success, we call our function to convert the option value to a string, and the string.

In Figure 1.2 we showed function prototypes, one for each type of option value that is eturned. Figure 1.4 shows of these four functions, soc k_str_flag, which prints the value of a flag option. The other three functions are similar.

sockopt/ceckopts.c

```
static char strres [128] ;
static char*
sock_str flag (union val xptr, int len)
(
    if (len ! = sizeof(int))
      snprintf (strres, Lizeof (strres);, "size (%d) not sizeof (i nt)", len);
     else
             snprintf (strres, sizeof (strres),
                        "%s", (ptr->_val == 0) ? "off" : "on") ;
    return (strres);
)
```

sockopt/checkopts.c

Figure 1.4 sock_str_flag *function convert flag option to a string*

Recall that the final argument to getsockopt is a value-result argument. The first check we make is that the size of the value returned by getsockopt is the expected size. The string returned is off or on, depending whether the value of the flag option is 0 or nonzero, respectively.

Running this program under AIX 4.2 gives the following output:

```
aix % checkopt
SO_BROADCAST: default = off
SO_DEBUG: default = off
SO_DONTROUTE: default = off
SO_ERROR: default = 0
SO_KEEPALIVE: default = off
SO_LINGER: default = Lonoff = 0, I -linger = 0
SO_OOBINLINE: default = off
SO_RCVBUF:   default   =   16384
SO_SNDBUF:   default   =   16384
SO_RCVLOWAT: default = I
SO_SNDLOWAT: default = 4096
SO_RCVTIMEO: default = 0 sec, 0 usec
SO_SNDTIMEO: default = 0 sec, 0 usec
SO_REUSEADOR: default = off
SO_TYPE: default = I
SO_USELOPBACK: default = off
IP_TOS: default = 0
IP_TTL: default = 60
TCP_MAXSEG: default = 512
TCP_NODELAY: default = off
```

The value of I returned for the SO_TYPE option corresponds to SOCK_STR EA M for this implementation.

3.3 Socket States

For some socket options there are timing considerations about when to set or fetch the options versus the state of the socket. We mention these with the affected options.

The following socket options are inherited by a connected TCP socket from the listening socket SO_DEBUG, SO DONTROUTE, SO_KEEPALIVE, SOW NGER, SO 00BINLI NE, SO_RCVBUF, and SO_SNDBUF. This is important with TCP because the connected socket is not returned to a server by accept until the three-way handshake is completed by the TCP layer. If we want to ensure that one of these socket options is set for the connected socket when the three-way handshake completes, we must set that option for the listening socket.

34 Generic Socket Option

We start with a discussion of the generic socket options. These options are protocol independent (that is, they are handled by the protocol-independent code within the kernel, not by one particular protocol module such as IPv4), but some of the options apply to only certain types of sockets. For example, even though the SO_BROADCAST socket option is called "generic", it applies only to datagram sockets.

SO_BROADCAST Socket Option

This option enables or disables the ability of the process to Send broadcast messages. Broadcast is supported for only datagram sockets and only on networks that support the concept of a broadcast message (e.g. Ethernet, token ring, etc.). You cannot broadcast on a point-to-point link.

Since as application must set this socket option before sending a broadcast datagram, it prevents a process from sending a broadcast when the application was never designed to broadcast. For exampe, a UM application might take the destination IP address as a command-line argument, but the application never intended for a is a broadcast address or not, the test is in the kernel: if the destination address is a broadcast address and this socket option is not set, EACCES is returned.

SO_DEBUG Socket Option

This option is supported only by TCP. When enabled for a TCP socket, the kernel keeps track of detailed information about all the packets sent or received by TCP for the socket. These are kept in a circular buffer within the kernel that can be examined with the trpt programme.

SO_DONTROUTE Socket Option

This option specifies that outgoing packets are to bypass the normal routing mechanisms of the underlying protocol. For example, with IPv4, the packet is directed to the appropriate local interface, as specified by the network and subnet portions of the destination address. If the local interface cannot be determined from the destination address (e.g, the destination is not on the other end of a point-to-point link, or not on a shared network), EN ETU N R EACH is returned.

The equivalent of this option can also be applied to individual datagrams using the MSG_DONTROUTE flag with the send, sendto, or sendmsg functions.

This options is often used by the routing daemons (routed and gated) to bypass the routing table (in case the routing table is incorrect) and force a packet to be sent out a particular interface.

SO_ERROR Socket Option

When an error occurs on a socket, the protocol module in a Berkeley-derived kernel sets a variable named so_error for that socket to one of the standard Unis *Exxx* values. This is called the *pending error* for the socket. The process can be immediately notified of the error in one of two ways.

1. If the process is blocked in a call to select on the socket, for either readability or writability, select return with either or both conditions set.
2. If the process is using signal-driven I/O, the SIGIO signal is generated for either the process or the process group.

The process can then obtain the value of so_error by fetching the SO_ERROR socket option. The integer value returned b getsockopt is the pending error for the socket. The value of so_error is then reset to 0 by the kernel

If so_error is nonzero when the process calls read and there is no data to return, read return-1 with errno set to the value ofso_error, the value of so_error is then reset to O. If there is data queued for the socket, that data is returned by read instead of the error condition. If so_error is nonzero when the process calls write, - 1 is returned with errno set to the value of so_error and so_error is reset to O.

This is the first socket that we have encountered that can be fetched but cannot be set.

SO_KEEPALIVE Socket Option

When the keepalive option is set for a TCP socket and no data has been exchanged across the socket ineither

direction for 2 hours, TCP automatically sends a *keepalive probe* to the peer. This probe is a TCP segment

to which the peer must respond. One of three scenarios result.

I. The peer responds with the expected ACK. The application is not notified (since everything if **OK).**

2. The peer responds with an RST, which tells the local TCP that the peer host has crashed and rebooted. The socket's pending error is set to EC ON NRESET and the socket is closed.
3. There is no respond from the peer to the keepalive probe. Berkeley-derived TCPs send eight additional probes, 75 seconds apart, trying to elicit a response. TCP will give up if there is no response within 11 minute and 15 seconds after sending the first probe. If there is no response to one the socket is closed. But if the socket's pending error is set to ET I M E DO U T and the socket is closed. But if the socket receives an ICMP error in response to one of the keepaliv proebs, the corresponding error is returned instead (and the socket is still closed). A common ICMP error in this scenario is "host unreachable", indicating that the peer host not crashed but is just unrechable, in which case the pending error is set to EH OSTU N R EACH .

Undoubtedly the most common question regarding this option is whether the timing parameters an be modified (usually to reduce the 2 hour period of inactivity to some shorter value). Appendix E of TCPvl discusses how to change these timing parameters for various kernels, but be aware that most kernels maintain these parameters on a per kernel basis, not on a per-socket basis, so changing the inactivity period from 2 hors to 15 minutes, for example, will affect *all* sockets on the host the this option.

The purpose of this option is to detect if the peer *host* crashes. If the peer *process* crashes, its TCP willsend a FIN across the connection, which we can easily detect with select. (This was why we used select in Section 3.3(unit 8). Also realise that if there is no response to any of the keepalive probes (scenario 3), we are not guaranteed that the peer host has crashed, and TCP may well terminate a valid connection. It could **be** that some intermediate router has crashed for 15 minutes,and that period of time just happens to completely overlap our host's 11 minute and 15 second keepalive probe period.

This option is normally used by servers, although clients can also use the option. Servers use the option because they spend most of their time blocked wating for input across the TCP connection, that is, waiting fora aclient request. But if the client host crashes, the server process will never know about it, and the server will continually wait for input that can never arrive. This is called a *halfopen connection.* The keepalive option will detect these halfopen connections and terminate them.

Figure 1.5 sumarises the various methods that we have to detect when something happens on the other end of a TCP connection. When we say "using select for readability" we mean calling sel;ect to test whether the socket is redable.

Scenario	Peer process crashes	Peer host crashes	Peer host is unreachable
Our IC? is actively sending **data**	Peer TCP sends a FIN which we can detect immediately using select for realiability. If TCP sends another segment, peer TCP responds with RST. If TCP sends yet another segment, our TCP sends us SIGPI PE.	OurTCP will time out and our socket's pending error is set to ETIMEDOUT.	Our TCP will time out and our socke's pending error is set to EHOSTUNREACH.
Our TCP is actively receiving data	Peer TCPwill sends a FIN, which we will read as a (possibly premature) end-of-file	We will stop recceiving data,	We will stop receiving data.
Connection is idle, keepalive set	Pea TCP sends a FIN, which we can detect immediately using select for readability,	Nine Keepalive probes are sent after 2 hours of inactivity and then our socket's pending error is set to ETIMEDOUT	Nine Keepalive probes are sent after 2 hours of inactivity and then our socket's pending error is set to ETIMEDOUT.
Connection is idle keepalive not set.	Peer TCP sends a FIN, which we can detect immediately using select for readabiltity.	(Nothing)	(Nothing)

Figure 1.5 Ways to detect various TCP conditions.

SOLINGER Socket Option

This option specifies how the close function operates for a connection-oriented protocol (e.g., for TCP but not for UDP). By default, close returns immediately, but if there is any data still remaining in the socket send buffer, the sysrem will try to deliver the data to the peer.

The SO_Ll NGER socket option let us change this default. This option requires the following structure to be passed between the user process and the kernel. It is defined by including <sys / socket. h>.

```
 struct linger{
   i nt Lon off ; /* 0=off, nonzero=on */
   int I-linger; /* linger time, Posix.Ig specifies units as seconds V
1;
```

Calling setsoc kopt leads to one of the following three scenarios depending on the values of the two structure members.

1. If Lonoff is 0, the option is turned off. The value of 1_1 i nger is ignored and the previously discussed TCP default applies: close returns immediately.
2. If 1_0 noff is nonzero and I i nger is 0, TCP aborts the connectiong when it is closed. That is, TCP discards any data still remaining in the socket send buffer and sends an RST to the peer, not the normal four-packet connection terminati leaves open the possibility of another incarnation of this connection being created within 2MSL seconds and having old duplicted segments from the just-terminated connection being incorrectly delivered to the new incarnation.
3. If 1_0 n off is nonzero and 1_li nger is nonzero, then the kemal will *linger* when the socket is closed. That is, if there is any data still remaining in the socket send buffer, the process is put to sleep until either (a) all the data is sent and acknowledged by the peer TCP, or (b) the linger time expires. If socket has been nonblocking, it will not wait for the close to complete, even if the linger time is nonzero.

When using this feature of the SOW NGER option it is important for the application to check the return value from close, because if the linger time expires before the remaining data is sent and acknowledged, close returns EWOU LD B LOCK and any remaining data in the send buffer is discarded. We now need to see exactly when a close on a socket returns, given the various scenaios that we have looked at. We assume that the client wirtes data to the socket and then calls close. Figure 1.6 shows the default situation.

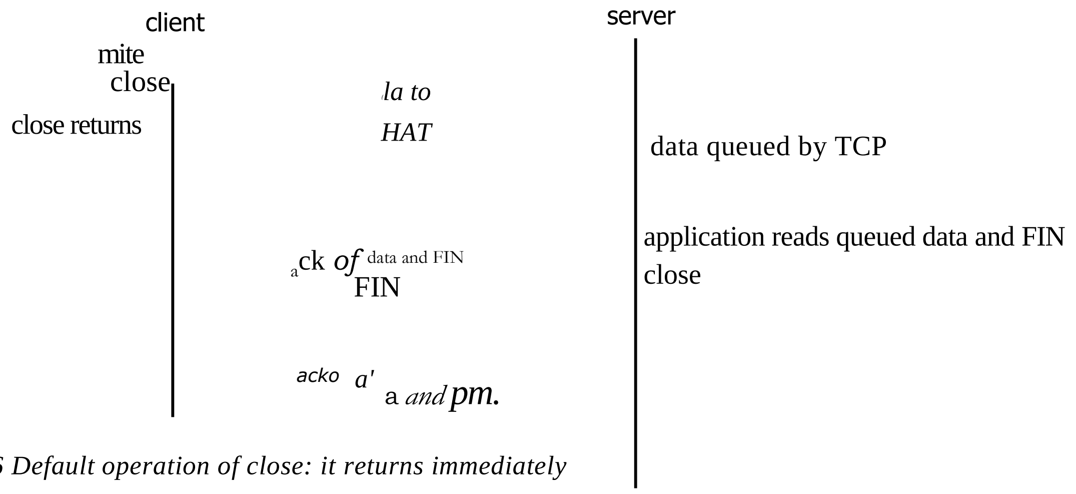


Figure 1.6 Default operation of close: it returns immediately

We assume that when the client's data arrives, the server is temporarily busy, so the data is added to the socket receive buffer by its TCP. Similarly the next segment, the client's FIN, is also added to the socket receive buffer On whatever manner the implementation records that a MN has been received on the connection). But by default the client's close returns immediately. As we show in this scenario, the client's close can return before the server reads the remaining data in its socket receive buffer. It is possible for the server host to crash before the server application reads this remaining data, and the client application will never know.

The client can set the SO_Ll N G ER socket option, specifying some positive linger time. When this occurs, the client's close does not retun until all the client's data and its FIN have been acknowledged by the serv.er TCP. We show this in Figure 1.7. But we still have the same problem as in Figure 1.6: the server host can crashbefore the server application reads its remaining data, and the client application will never know.

The basic principle here is that a successful return from close, with the SOU.. NG ER socket option set, only tell us that the data we sent (and our FIN) have been acknowledged by the peer TCP. This does *not* tell us whether the peer application

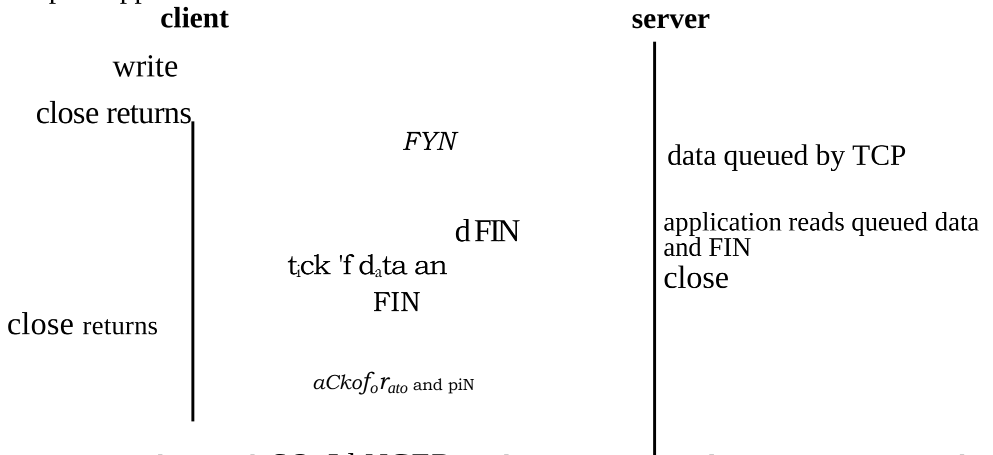


Figure 1.7 close with SO_Ll NGER *socket option set and Unger a positive value*

has read the data. If we do not set the SOW NG ER socket option, we do not know whether the peer TCP has acknowledged the data.

One way for the client to know that the server has read its data is to call shutdown (with a second argument of SHUT_WR) instead of close and wait for the peer to close its end of the connection. We show this scenario in Figure 1.8.

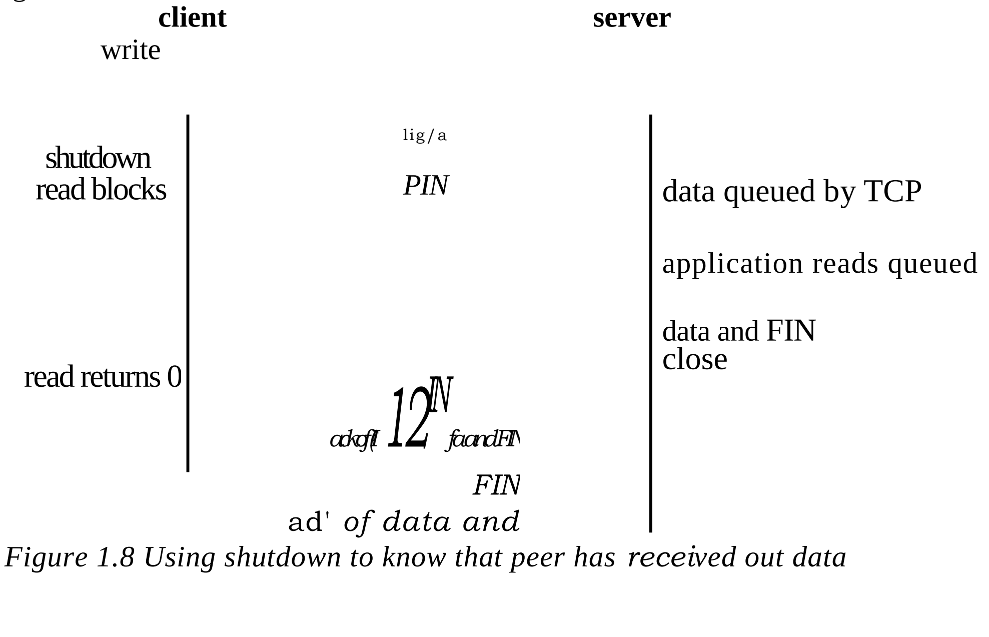


Figure 1.8 Using shutdown to know that peer has received out data

Comparing this figure to Figure 1.6 and 1.7 we see that when we close our end of the **connection, depending** on the function called (close or shutdown) an whether the SO_L I N G ER socket option is set, **return can occur** at three different times:

1. close returns immediately, without waiting at all (the default; Figure 1.6),
2. close lingers unitl the ACK of our **FIN** is received (Figure 1.7), or
3. shutdown followed by a read waits until we receive the peer's FIN (Figure 1.8)

,

In this unit, you have learned about the various socket options available in client-server programming **and their** operations. You aso learn about the way you can print the default value of all the options. **These techniques and** schemes will enable you to know the more about socket programming and addressing.

Sumivary

What you have learned in this unit borders on socket options available for client-server **application programme. You also learn how to print the default value of all the options. The next unit shall build upon this.**

Olf Thor Marked Assigzi

Explain the operations of the various generic socket options that you know.

a. As in Section 3.4

Excercise 1.1

Compare setsockopt and getsockopt.Functions

Exceroise 1.2

Write on socket states

Re

Stevens, W. R. *Unix Network Programming* (2nd ed.), Prentice **Hall, PTR, 1998.**

Module 3: Overview of Network Programming

Unit 10: Elementary Name and Address Conversion

In this unit, you will learn how functions help to convert between names and numeric values. You will also learn about the use of some specific functions to accomplish these tasks. Let us now see what you will learn in this unit as specified the unit objectives below.

jooi

By the end of this unit, you should be able to:

- understand how names are mapped with IP address
- appreciate the importance of Domain Name server in names and address resolution
- understand how specified functions are used for these operations.

Al! the examples so far in this text have used numeric address for the hosts (e.g., 206.6.226.33) and numeric port numbers to identify the sewers (e.g., port 13 for the standard daytime server and port 9877 for our echo server). We should, however, use names instead of numbers for numerous reasons: name are easier to remember, the numeric address become much longer making it much more error prone to enter an address by hand. This chapter describes the functions that convert between names and numeric value: getostbyname and gethostyaddr to convert between host-name and IP address, and getservbyname and getservbyport to convert between service names and port numbers.

The hostname functions have recently been enchanced to work with IPv6, in addition to IPv4, and we also describe these changes. This is the beginning of our move toward protocol independence.

3.1 Domain Name System

The *Domain Name System*, or DNS, is used primarily to map betrween hostnames and IP address. A hostname can be either a *simple name,* such as so I ar i s or bsd i , or *afully qualified domain name* (FQDN) suchas solaris.kohala.com.

Resource Records

Entries in the DNS are known as *resource records* (RRs). There are only a few types of RRs that affect us.

A — An A record maps a hostname into a 32-bit IPv4 address For example, here are the four DNS records for the host solaris in the kohal.com domain, the first of which is an A record:

```
Solaris IN A        206.62.22633
        IN AAAA     511b: df00: ce3e:e200:0020:0800:2078:e3e3
        IN MX        5 solaris.kohala.com
        N    MX      10 mailhost.kohala.com
```

AAAA — A AAAA record, called a "quad A" recod, maps a hostname into a 128-bit IPv6 address. The term "quad A" was chosen because a 128-bit address is four times largfer than a 32-bit address.

PTR — PTR records (called "pointer records") map IP address into hostnames. For an IPv4 address the 4 bytes of the 32-bit address are reserved, each byte is converted to its decimal ASCII value (0-255), and in-addr.arpa is then appended. The resulting string is used in the PTR query.

For an IPv6 address the32 4-bit nibbles of the 128-bit address are reversed, each nibble is converted to its corresponding hexadecimal ASCII value (0-9a-D, and ip6.int is appended.

Fr example, the two PTR records for our host solaris would be
33 . 226. 62 . 206 . in-addr arpa and 3 . e . 3 . e . 8 . 7 . 0 . 2 . 0 . 0 . 8 . 0 .
0.2 .0 .0.0.0 . 2.e.e. 3 .e.c. 0 .0 .f.d.b.i.f. 5 .ip6.int.

MX An MX record specifies a host to act as a "mail exchange" for the specified host. In the example for the host so I a ris above, two MX records are provided. The first has a preference value of 5 and the second has a preference value of 10. When mutltiple MX records exist, they are used in order of preference, starting with the smallest value.

CNAME CNAME stands for "canonical name" A common use is to assign CNAME record for common service, such as ftp and www. If people use these service name, instead of the actual hostname, it is transparent if the service is moved to another host. For example, the following could be CNAMEs for our host bsd i

```
IP           CNAME    bsdi.kohala.com
WAW          CNAME    bsdi.kohala.com
mailhost     CNAME    bsdi.kohala.com
```

It is too early in the deployment of IPv6 to know what conventions administrators will use for host that support both 1Pv4 and LPv6. In our example earlier in this section we specified both an A record and a AAAA record for host so I a ris. Some administrators place all AAAA records into their own subdomain, often named ipv6. For example the hostname associated with the AAAA record would then be sol a ris.i pv6. ko ha la .com. Sometimes this is done because the administrator of the dual-stack host does not have domain name responsibility for the entire domain but obtains responsibility for the separate ipv6 subdomain.

Instead, the author place both the A record and the AAAA record under the host's normal name (as shown earlier) and creates another RR whose name ends in -4 containing the A record, another RR whose name ends in -6 containing the AAAA record, and another RR whose name ends in -611 containing a AAAA record with the host's link-local address (which is sometimes handy for debugging purposes). All the records for another of our hosts are then.

```
aix-4    IN    A       206.62.226.43
aix      IN    A       206 . 62 . 226 . 43
         IN    MX      5   alx.kohala.com
         IN    MX      10 mailhost . kohala . corn
         IN    AAAA    5fib:c1f00:ce3e:e200:0020:0800 5afc:2b36
aix•6    IN    AAAA    5fib:df00:ce3e:e200:0020:0800:5afc:2b36
aix-611  IN    AAAA              fe80::0800:5afc:2b36
```

This gives us additional control over the protocol chosen by some applications, as we will see in the next chapter.

Resolvers and Name Servers

Organisation run one or more *name servers,* often the programme known as BIND (Berkeley Internet Name Domain). Application such as the client and server that we are writing in this text contact a DNS server by calling functions in a library known as the *resolver.* The common resolver functions are gethostbr a me and gethostbyaddr, both of which are described in this reverse mapping.

Figure 1.1 shows a typical arrangeent of application, resolves, and name servers. We write the application code. The resolver code is contained in a system library and is link-edited into the application when the application is built. The application code calls the resolver code using normal function calls, typically calling die functions gethostbyna me and gethostbyaddr.

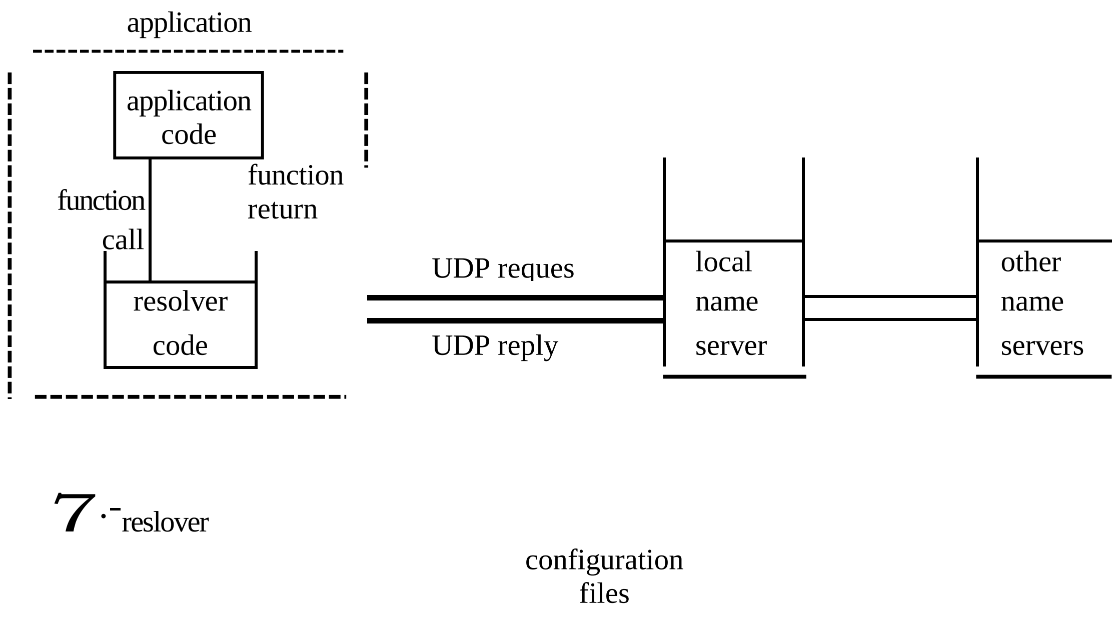


Figure 1.1 Typical arrangement of clients, resolves, and name servers

The resolver code reads its system-dependent cofiguration files to determine the location of the organisation's name servers. (we use the plural. "name servers" because most organisations run multiple name servers, even though we show only one local server in the figure.) The file / etc/resolv. co nf normally contains the IP addresses of the local name servers.

The resolver sends the query to the local name server using UDP. If the local name server does not know the answer, it will normally query other name across the Internet, also using UDR

DNS Alternatives

It is possible to obtain the name and address information without using the DNS and common alternatives are static host files or NIS (Network Information System). Unfortunately it is implementation dependent how an administrator configures a host to use the different types of name service. Solaris 2.x and HP-UX 10.30 uses die file /etc/nsswitch.cont Digital Unix uses the file /etc/svc.conf, and AIX uses the file /etc/netsvc.conf. BIND 8.1 supplies its own version named IRS (Information Retrieval Service) that uses the file /etc/ i rs. co nf. If a name server is to be used for hostname lookups, then all these systems use the file /etc/resolv.conf to specify the IP address of the name servers. Fortunately, these differences are normally hidden to the application programmer, so we just call the resolver functions such as geth ostbyn a m e and gethostbyaddr.

3.2 gethostbyname Function

Host computers are normally known by human-readable names. All the examples that we have shown so far in this book have intentionally used IP addresses instead of names, so we know exactly goes into the socket address structures. for functions such a connect and se n dto, and what is returned by functions such as accept and reevfrom. But most applications should deal with names and not address. This is especially true as we move IPv6, since IPv6 address (hex strings) are much longer than IPv4 dotted-decimal numbers. (The example AAAA record and i p6. i nt PTR record in the previous section should make this

The most basic function that looks up a hostname is gethostbyna me. If successful, it returns a pinter to a hostent structure that contains all the IPv4 addresses or call the IPv6 address for the host.

```
#include <netdb.h>

  struct hostent *gethostbyname (const char shasiname):

                 Returns: nonnull pointer if OK, NULL on error with h_err no set
```

The nonnull pointer returned by this function points to the following hostent structure:

```
struct hostent {
   char *h_name;        /*official (canonical) name of host*/
    char **h_aliases; /* pointer to array of pointers to alias name*/
    it       h_addrtype; /* host address type: AF_I N ET or AF_INET6 */
    it       h_lenght;      /*length of address: 4 or 16 */
    char ntaddr list; /* ptr to array of ptrs with I Pv4 or I Pv6 addrs*/

      #define h_addr       h_addr_list [0] /* firs address in list */
```

In terms of the **DNS, gethostbyname** performs a query for an A record for a AAAA record. This function can return either Wv4 addresses or II'v6 addresses. We summarise in Figure 1.5 the conditions under **which it returns these** two types of addresses.

Figure 1.2 showns the arrangement of the hostent structure and the information that it points to assuming **the hostname that is looked up has two alias names and three 11³v4 addresses. Of these fileds, the official hostname nd all of the aliases are null terminated C** strings.

The returned h_na me is called the ***canonical*** name of the host. For example, given the **CNAME records shown in the previous section, the canonical name of the host ftp. koh a la .com would be bsdi koha Also, if we call gethostbyname** from the host solaris with an unqualified hosmame, say solaris, the FQDN **(solaris.koha la.com)** is returned as the canonical name.

When IPV6 addresses are returned, the h_addrtype member of the hostent structure is set to AF_INET6 **and h_lenght member is set to 16. Figure 1.3** shows thes changes, with the shaded fieldss having **changed from Figure 1.2.**

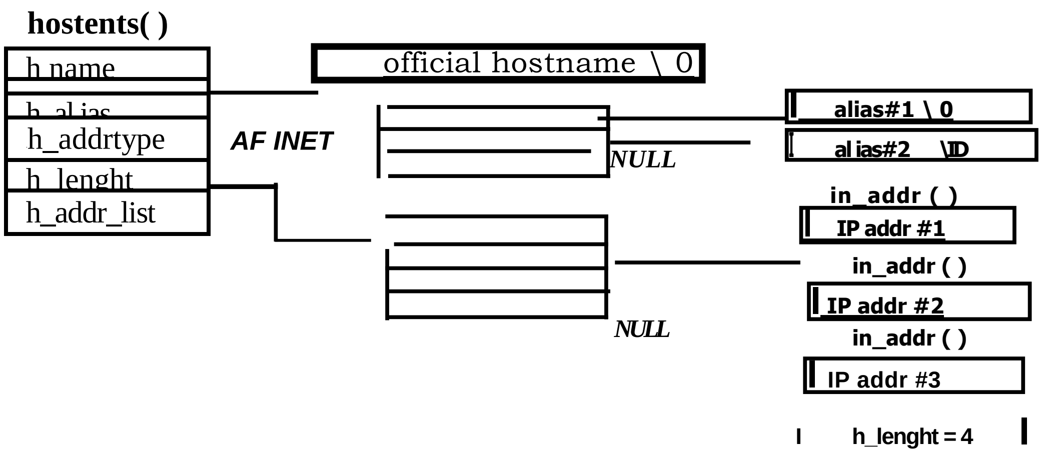


Figure 1.2 hostent structure and the information it contains

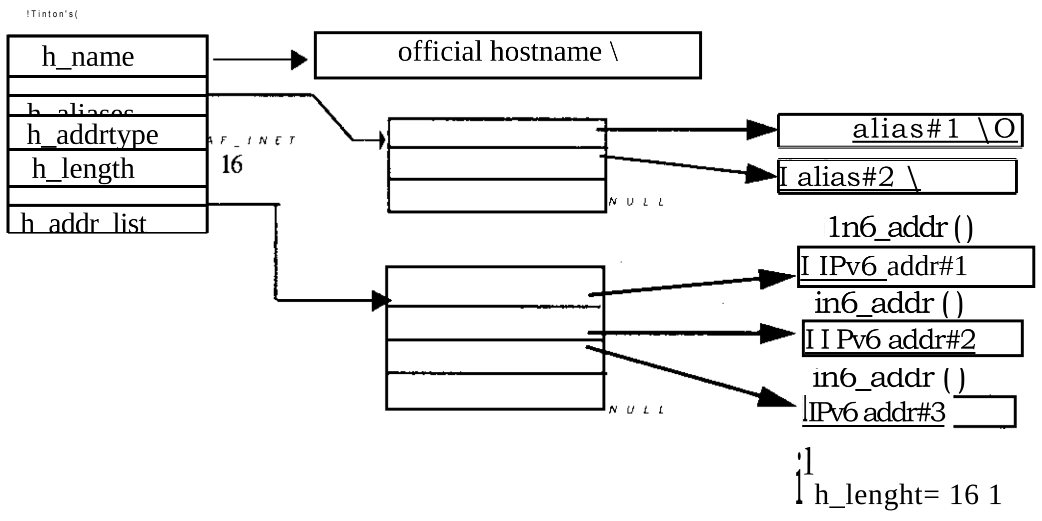


Figure 1.3 Changes in information returned in hostent structure with IPv6 addresses

gethostbyna me differs from the other socket functions that we hve described in that it does not set errno when an error occurs. Instead, it sets te integer h_errno to one of the following constants defined by including <reit. h>

- HOST_NOT_FOUND
- TRY_AGAIN
- NO_REGOVERY
- NO_DATA(denticaltoNO_ADDRESS)

The NO_DATA error means the specified name is valid, but it does not have either an A record or a AAAA record. An example of this is a hoshmme with only an MX record.

Current releases of BIND provide the function hsterror that takes an h_errn 0 value as its only argument and returns a const char * pointer to a description of the error. We show some examples of the strings returned by this function in the next example.

Example

Figure 1.4 shows a simple programme that calls gethostbyna me for any number of command-line arguments and prints all the returned information.

gethostbyna me is called for each command-line argument.

The official hostname is output followed by the list of alias names.

For this programme to support both 1Pv4 and II³v6 addresses we allow the returned address type to be either A F_I NET or ALI NET6. But we do not allow the latter unless it is defined (i.e., the hot supports IPv6).

pptr points to the array of pointers tothe individual addresses. For each address we call inet_ntop and print the returned string. Note that i net_ntop handles both IPv4 and IPv6 addresses, based on its first argument. Also notice that we defined str of length I NET6_ADDRETRLEN, which we said is large enough for the longest possible IPv6 address string. In our unp.h file we define this constant, even if the host does not support #ifdef within our code).

We first execute the programme with the name of our host solaris, which has just one IPv4 address.

```
sola ri s % hostent solaris
official hostname: solaris.kohala.com
        address: 206.62.226.33
```

Notice that the offical hostname is the FQDN. Also notice that even though this host has an 11^3v6 address, only the IPv4 address is returned. Next is a host with multiple IPv4 addresses.

```
sol a ris % hostent gemini.tuc.noao.edu
official hostname: gemini.tuc.noao.edu
            address:   140.252.1.11
            address:   140.252.3.54
            address:   140.252.4.54
            address: 140.252.8.54
```

names/hostentc

```
#include "unp.h"
int
main (int argc, char**argv)
{
      char *ptr,"pptr;
       char str [INET6_ADDRSTRLEN];
      struct hostent*hptr;

      while (--argc > 0) {
            ptr =*++argv;
             if( (hptr = gethostbyname (ptr)) == NULL) {
                  err_msg ( "gethostbyna me error for host : %s: %s",
                                   ptr, hstrerror (h_errno));
                  continue;

              print "official                %s \ n", hptr->h_name);
       for (pptr = hpt•>h_aliases; *pptr != NULL; pptr++)
               printk" \ talias: %s \ n", *pptr),
         switch (hptr->h_addrtype){
            case AF_INET:
#idef AF_INET6
              case AF_INET6:
#endif
                  pptr = hptr->h_addr list;
                  for( ;*pptr != NULL; pptr++)
                         Printk" \taddress:%s \ n",
                                Inet_ntop(hptr->h_addrtype,*pptr, str, sizeokstr)));
                     break;
              default
                  err_ret( "unknown address type");
                     break;

      }
  exit (0) ;
 }
```

names/hosteritc

Figure 1.4 Cagethostbyname *and print returned infonnation.*

Next is a name that we showed in section 3.1 having a CNAME record.

```
        seals%        hostent www
 officialhostname: bsdi.kohala.com
                          a lias:www.koha la.com
                         address: 206.62.226.35
```

As expected, the official hostname differs from our command-line argument.
To see the error strings returned by the hsterror function we first specify a non-existent hostname, and then a name that has only an MX record.

```
solaris% hostent nosuchname
gethostbyname errorfor host: nosuchname: Unknown host

so I a ris % hostent uunet.uu.net
gethostbyrameerraforhu31,uunetuu.net
Nbaddressassociataddchnarre
```

3.3 **RES_USE_INET6** Resolver Option

Newer release of BIND provide a resolver option named R ES_U SE _I N ET6 that we can set in three different ways. We can use this option to tell the resolver that we want 1Pv6 addresses returned by gethostbyname, instead oflPv4 addresses.

1 An application can set this option itself by first calling the resolver's res jnit function and then enabling the option:

```
#indude <resolv.h>

res init ();
_res.options / = RES_USE NET6;
```

This must be done before the first call to geth ostbyna me or gethostbyaddr. The effect of this option is only on the application that sets the option.

2. If the enviroment variable RE_OPTIONS contains the string inet6, the option is enabled. The effect of this option depends on the scope of the enviroment variable. If we set it in our profile file for example (assuming a Korn-Shell) with the export attribute, as in

   ```
   export RES OPTIONS=inet6
   ```

   then it affets every programme that we run from our login shell. But if we just set the variable on a command line (as we show shortly), then it affects only that command.
3. The resolver configuration file (normally /etc / resolv.conf) can contain the line

   ```
   options inter6
   ```

   Be aware, however, that setting this option in the resolver configuraion file affects *all* applictions on the host that call the resolver functions. Therefore this technique should not be used until all applications on the host are capable of handling I2Pv6 addresses returned in a hostent structure.

The first method sets the option on a per-application basis, the second method on a peruser basis, and the third method on a per-system basis

We now run our example programme from Figure 1.4 setting the enviroment variable RES_

```
solaris % RES_OPTIONS=inet6 hostent solaris a name with a AAAA record
official hostname: solaris.kohala.com
          address: 5flb:df00:ce3e:e200:20:800:2078:e3e3

solaris% RES_OPTIONS=inet6 hostent bsdi              a name without a AAAA record
 official hostname:    bsdi.kohala.com
           address:    :: ffff:206.62.226.35
           address:    :: W206.62.226.66
```

The first time we execute our programme it return the IPv6 address of the host (recall its AAAA record in Section. The second time we execute our programlanme we specify a hostname that does not have a AAAA record. Still IPv6 addresses are returned: the IPv4-mapped IPv6 addresses (Section A.5).

3.4 gethostbyname2 Function and IPv6 Support

When support for IPv6 was added to BIND 4.9.4, the function get ho st byn a m e2 was added, which has two argument, allowing us to specify the address family.

```
#include<netdb.h>

struct hostenrgethostbyname2(const char*hostname, intfamily),

                         Returns: nonull pointer if OK, NULL on error with h_ermo set
```

The return value is the same as with gethostbyna me, a pointer to a hostent structure, and this structure remains the same. The logic of the function depends on the family argument and on the R ES_U S N ET6 resolver option (which we mentioned at the end of the previous section).

Before decribing th details, Figure 1.5 summarises the operation of gethostbyna me and gethostbyname2 with regard to the new R ES_U S N ET6 option. We show in a bolder fon the values that can change:

- whether the R ES_U S E _I N ET6 option is **off** or **on,**
- whether the second argument to gethostbyname2 is **AF_INET** or **AFINET,**
- whether the resolver searches for A records or AAAA records, and
- whether the returned addresses are of length 4 or **16.**

The operation of gethostbyna rne2 is as follows:

- If the *family* argument is A F_I N ET, a query is made for A records. If unsuccessful, the function returns a nut pointer. If succeful, the type and size of the returned addresses depends on the new R ES_U SE _I NET6 resolver option: if the option is not set (the default), Tv4 addresses are returned and the h_length memeber of the hostent structure will be 4; if the option is st, 1Pv4-mapped IPv6 addresses are returned and the h- length member of the **hostent** structure will be 16.

```
solaris % RES_OPTIONS=inet6 hostent solaris a name with a AAAA record
official hostname: solaris.kohala.com
        address: 5flb:df00:ce3e:e200:20:800:2078:e3e3

solaris% RES_OPTIONS=inet6 hostent bsdi              a name without a AAAA record
official hostnaine:     bsdtkohala.com
        address:        : : W.206.62.226.35
        address:        : :ifif:206.62.226.66
```

The first time we execute our programme it return the IPv6 address of the host (recall its AAAA record in Section. The second time we execute our programlcmme we specify a hostriame that does not have a AAAA record. Still IPv6 addresses are returned: the IPv4-mapped IPv6 addresses (Section A.5).

3.4 **gethostbyname2** Function and IPv6 Support

When support for IPv6 was added to BIND 4.9.4, the function geth ost byn a me2 was added, which has two argument, allowing us to specify the address family.

```
itinclude<netdb.h>

struct hostent tethostbyname2(const charliostname, int family),

                         Returns: nonull pointer if OK, NULL on error with h_ermo set
```

The return value is the same as with gethostbyna me, a pointer to a hostent structure, and this structure remains the same. The logic of the function depends on the family argument and on the R ES_U N ET6 resolver option (which we mentioned at the end of the previous section).
Before decribing th details, Figure 1.5 summarises the operation of gethostbyna m e and gethostbyname2 with regard to the new R ES_U S N ET6 option. We show in a bolder fon the values that can change:

whether the R ES_U S El NET6 option is off or **on,**
whether the second argument to gethostbyn a m e2 is **AF_INET** or **AF_INET,**
whether the resolver searches for **A** records or AAAA records, and
whether the returned addresses are of length 4 or 16.

The operation of gethostbyna me2 is as follows:

If the *family* argument is ALI N ET, a query is made for A records. If unsuccessful, the function returns a nul pointer. If succeful, the type and size of the returned addresses depends on the new R ES_US NET6 resolver option: if the option is not set (the default), 1Pv4 addresses are returned and the h_lengt h memeber of the hostent structure will be 4; if the option is st, 1Pv4-mapped IPv6 addresses are returned and the h-length member of the hostent structure will be 16.

	RE_USE_INET6 option	
	off	**on**
gethostbyname *(host)*	Search for A records. If found, return IPv4 addresses (h_length=-4) Else error This provides backward compatibility for all existing 1Pv4 applications.	Search for AAAA records. If found, return IPv6 addresses (h_length =16). Else search for A records. If found, return 1Pv4-mapped 1Pv6 addresses(h_(ength = 16). Else error
gethostbyname2 *(host.* AF_INET)	Search for A records. If ound, return 1Pv4 addresses (h_l ength **=4).** Else error.	Search for A records. If found, return 1Pv4-mapped WO addresses (h_length = **16).** Else error.
gethostbyname2 *(host.* ALINET6)	Search for AAAA records. If found, return IPv6 addresses(h_length = **16).** Else error.	Search for AAAA records. [found, return IPv6 addresses(h_length = **16).** Else error.

Figure 1.5 gethostbyname cuidgethostbyname2 *with resolver* RE_USE _INET6 *options.*

This function can be used if the appication wants to force a search for one specfic type of address, either IPv4 or IPv6. But it is more common for applications to call gethostbyna me, nd newer versions of this function can return either IPv4 or IPv6 addresses.

One way to describe the actions of gethost byna me and the RE_USE_INET6 options is to look at its source code, which we show in Figure 1.6.

If the resolver has not yet been initialised (the RE _INIT flag is not set), res_i nit. is called. This initialisation function examines and processes the RES_OPTIONS enviroment variable. If this variable contains the string inet6 or if the resolver configuration file contains the options i net6 line, then the flag R E_USE _I N ET6 is set by res_i nit. The res_i nit function is normally called automatically by gethostbyn a m e (as we show here) the first time it is called byte application, or by gethostbyaddr. Alternately, we showed that the appliction can also call re_init and then set the RE_USE_INET6 flag explicity.

If the RE_USE_INET6 option is *not* set, the last line of the function is executed and get hostbyn a m e2 is called with an address family argument of AF_I NET. We saw in Figure 9.5 that this call searches for only A records. This provides backward compatibility for all existing applications.

If the RE_US E _I N ETS options is enabled, gethostbyname2 is called with an address family argument of A F_I NET6 to search for AAAA records (Figures 1.5). If this succeeds, gethostbyname returns. If this fails, gethostbyna me2 is called with an address family argument of AF_I NET to search for A records. If this succeeds, what is

```
strict hostent*
gethostbyname (const char *name)

    struct hostent *hp;
    if (Cres.options &            N IT) =-7 0 && resinit ( )== -1) {
            h_errno = NETDB_INTERNAL;
            return (NULL);

     if (_res.options & RES_USE_INET6) (
        hp = gethostbyname2 (name, AF_INET");
        if (hp)
            return (hp);

    return (gethostbyname2 (name, AF_INET) );
```

Figure 1.6 ***gethostbyname flunks and*** *IPv6 support.*

not apparent in Figure 1.6 is that 4-byte addresses are automatically mapped into 16-byte EPv4-mapped 1Pv6 addresses.

In summary, when the RE USE_INET6 option is enabled and the appliction calls gethostbyna me, the appliction is telling the resolver "I want only IPv6 addresses returned, period. Search for AAAA records first, but if none are found then search for A records and if they are found, returned the addresses as IPv4-mapped IPv6 addresses,"

33 **gethostbyaddr** Function

The function gethostbyaddr takes a binary IP address and tries to find the hostname corresponding to that address. This is the reverse of gethostbyna me

```
#indude<netdb.h>

structhostent ftgethostbyaddr(const char *addr; size _t len, int family);
```

Returned: nonnull pointer ijOK, NULL on error with h_errno set

This function returns a pointer to the same hostent structure that we described with gethostbyname. The field of interest in this structure is nonnallt h_name, the canonical hostname.

The *addr* arpnnent is not a char* but is really a pointer to an in_addr or in6_addr structure containing the IPv4 or IPv6 address. *len* is the size of this structure: 4 for an IPv4 addresses, or 16 for an 1Pv6 addresses. *The famio* argtnnent is either AF I N ET or AF I NET6.

In terms of the DNS, gethostbyaddr queies a name server for a PTR record in the in addrarpa domain for an 1Pv4 address, or a PM record in the ip6. int domain for an IPv6 address.

gethostbyaddr Function and IPv6 Support

gethostbyaddr has always had an address family argument, so when IPv6 support was added to **BIND** there was no need to invent another function (similar togethostbyname2) But there are a few three tests are applied in the order listed:

1. If the *family is* AF_I N ET6, the *len* is 16, and the address is an IPv4-mapped IPv6 address, then the low-order 32 bits of the address (the IPv4 portion) are looked up in the in_addr.arpa domain.
2. If the *family is* AF_I N ET6, the *len* is 16, and the address is an IPv4-compatible IPv6 address, then the low-order 32 bits of the address (the IPv4 portion) are looked up in the in_addr.arpa domain.
3. If an IPv4 address was looked up (either the family argument was AF_INET or one of the two case above were true) and the R ES_U SE_I N ET6 resolver option is set then the one returned address (a copy of the *addr* argument) is converted to an IPv4-mapped address: h_addrtype is AF_I N ET6 and h_length is 16.

The third point is usually of little importance because few applications examine the IP address returned by gethostbyaddr, since it is just a copy of the argument. Application normally call this function to examine the n_n a me member of the returned hostent structure (and possibly the aliases too).

3.6 uname Function

The u name fimctionreturns the name of the curent host. This function is nt part of the resolver library, but we cover it here because it is often used along with gethostbyname to determine the local host's IP addresses.

```
#incude <sys/utsnameh>

int uname(struct utsnamflarney,

Returns: nonnegative value if OK, -1 on error
```

This function fills in a utsna me structure whose address is passed by the caller:

```
#define _UTS_NAMES IZE              16
#define _UTS_NODESIZE              256

 struct utname (
   char sysname [_UTS_NAMESIZE];        /* name of this operating system */
   char nodename LUTS_NODESIZE];         /*name of this node*/
   char release [_UTS_NAMESIZE];        /*O. S. release level */
   char version [_UTS_NAMESIZE];        /* O. S. version level */
   char machine LUTS_NAMESIZE];         /*hardware type */
```

In this unit, you have learned about the conversion of names between conventional address and numeric values 'ion also learn about the importance of Domain Name server in the resolutions of names. These schemes also enables you t know some functions that are used for these operations.

What you have learned in this unit focuses on the resolution and conversion of names between numeric values and addresses. You learned about the vital roles of Domain Name Serers in these resolutions. You must have appreciated the specific functions used for these conversions.

a Expalin the rotes of Domian Name servers in names and addresses resolutions

Excercise 1.1

How does gethostbyname Function works?

Excercise 1.2

Discuss the uname Function

Oft **and otl**

Stevens, W. **R.** *Unix Network Programming,* (2nd ed) vol. 1 and ed., prentice Hail, **PTR.** 1998.

Index