# E-DAF
# EMPLOYEE DEVIATION APPROVAL FORM
# DIGINET SOFTWARE LTD.
# BANGALORE-25

PROJECT REPORT

Submitted in partial fulfillment of the requirement for the award of the degree of M.Sc. (applied science) Software Engineering of Bharathiar University, Coimbatore.

P-1225


Submitted by,
**S.CHANDRA SEKARAN**
0137S0028


Under the guidance of


External Guide
**Mr. N.Chandra Bhushan**
System Engineer

Internal Guide
**Ms.L.S.JAYASHREE,M.E**
Senior Lecturer


Estd. 1984


Department Of Computer Science and Engineering
**KUMARAGURU COLLEGE OF TECHNOLOGY**
(Affiliated to Bharathiar University)
**COIMBATORE-641006.**


**(JUNE-2004 TO SEPTEMBER –2004)**

Department Of Computer Science and Engineering

**KUMARAGURU COLLEGE OF TECHNOLOGY**

(Affiliated to Bharathiar University)

**Coimbatore-641006.**

**(JUNE-2004 TO STEPTEMBER –2004)**

# CERTIFICATE

**This is to certify that the project entitled**

**VOICE OVER INTERNET PROTOCOL (VOIP)**

**Done By**

**Mr.S.CHANDRA SEKARAN**

**0137S0028**

**Submitted in partial fulfillment of the requirements for the award of the degree of M.Sc (applied Science) Software Engineering of Bharathiar University, Coimbatore.**

**Professor and HOD** **Internal Guide**

**Submitted to University examination held on** 29-09-2004

**Internal examiner** **External examiner**

DigiNet®
Software and Communications Ltd.

# 7-10 Ground Floor, Chache Towers
50, Residency Road, Bangalore - 560 025
Tel : 91-80-22123364/65, Telefax : 91-80-51125428
www.diginetlimited.com www.diginet.ltd

Ref No. DSCL /1409/2004-2005 Date: 15.09.2004

# CERTIFICATE

This is to certify that **Mr. S. Chandra Sekaran,** of **Kumaraguru College Of Technology** Coimbatore (affiliated to **Bharthiyar University**) was associated with us as a Project Trainee to carry out his academic project for the partial fulfillment of award of **MSc. (Software Engineering).**

He has successfully completed the project titled **'Voice Over Internet Protocol (VOIP)'** in compliance with the requirement of partial fulfillment of the **Master of Science In Software Engineering.** He was associated with us during the period from July-2004 to Sep. -2004.

He acquitted himself well during his traineeship and we appreciate his efforts. We wish him well for a bright future.

For and on behalf of **DigiNet Software and Communications Ltd.,.**

HR Manager

# Acknowledgement

Before going into thick of things I would like to add a few heartfelt words for the people who have contributed directly or indirectly in making this project. First of all I wish to thank **Dr.K.K.PADMANABHAN,B.SC (Engg), M.Tech, Ph.D** Principal, and **Dr.S.THANGASAMY,Ph.D** Head of the Department, computer Science and Engineering Kumaraguru College of Technology, Coimbatore who helped and supported and encouraged me to carry out this project. I wish to express my sincere thanks to my project coordinator **Mr.K.R.BASKARAN,B.E,M.S** Assistant Professor Kumaraguru College of Technology, Coimbatore for granting me permission to do this project and her guidance through my project. I would like to thank **Ms.L.S.JAYASHREE, M.E,** Senior lecturer Kumaraguru College of Technology for his valuable help to assist me to step into the IT industry.

I would like to thank **Mr.N.Chandra Bhushan, (system engineer)** Diginet software Ltd for his valuable guidance and help. He helped me not only in technical fields but also in every field whenever I needed his help.

I really deem it a special privilege to convey my prodigious and everlasting thanks to all the Software Engineers in Diginet Software Ltd, for all their timely help and sincere suggestions during the period of the project work.

I also thanks to my classmates and friends whose mutual co-operation and understanding were with me throughout my course session. I take this opportunity to give sincere regards to my parent for their constant support, encouragement, understanding and love. Without it, it would have been impossible for me to achieve all that I have.

**S.CHANDRA SEKARAN**

# ABSTRACT

**Voice over IP (VoIP)** uses the **Internet Protocol (IP)** to transmit voice as packets over an IP network. So VoIP can be achieved on any data network that uses IP, like Internet, Intranets and Local Area Networks (LAN). Here the voice signals are digitized, compressed and converted to IP packets and then transmitted over the IP network. At the other end the IP packets are uncompressed, quantized and converted to voice signal. This process is used both ways.

Since it became popular, the Internet has attracted many developers to develop software applications for better communications than offered by plain text. Among these applications are those that support voice communication. The project is intended to develop a voice communications application, which offers a real time point-to-point voice service providing virtual duplex voice communication over the Internet.

The major aim of the project is to develop an application for the Linux OS. The application will offer a real time point-to-point voice service providing virtual duplex voice communication over the Internet.

The objective of the project is to reduce bandwidth; I-Phone uses ADPCM compression to compress speech. ADPCM reduces the I-Phone bandwidth from 128-Kbits/sec to 32Kbits/sec. For network communications, I-Phone uses the Berkeley Sockets interface under the Linux OS. The GUI for the Linux version was developed for X Windows.

Internet Telephony will be providing real time voice communication over the Internet. This software have been tested on two machines that will be placed side by side. The targeted time difference between the moment the user on machine A will start talking and the moment the user on machine B will hear on the user on machine A, should be less than 100 msec.

I-Phone provides a point-to-point voice communication over the Internet.

# 1.0 INTRODUCTION

## 1.1 PROJECT OVERVIEW

The major aim of the project is to develop an application for the Linux OS. The application will offer a real time point-to-point voice service providing virtual duplex voice communication over the Internet.

The objective of the project is to reduce bandwidth; I-Phone uses ADPCM compression to compress speech. ADPCM reduces the I-Phone bandwidth from 128-Kbits/sec to 32-Kbits/sec. For network communications, I-Phone uses the Berkeley Sockets interface under the Linux OS. The GUI for the Linux version was developed for X Windows.

- PC-To-PC VOIP software for Linux

Internet Telephony will be providing real time voice communication over the Internet. This software will be tested on two machines that will be placed side by side. The targeted time difference between the moment the user on machine A will start talking and the moment the user on machine B will hear on the user on machine A, should be less than 100 msec.

I-Phone provides a point-to-point voice communication over the Internet.

I-Phone will include a daemon which will notify the user when a client someone@somewhere.net will be trying to contact the user via I-Phone.

An easy to use graphical user interface will be developed under X Windows.

## 1.2 Organisation profile

### DIGINET Software and Communications LTD.

Diginet: Partners of TCS (Tata Consultancy Services) is a world class software led IT services. Diginet is a IT service company providing a range of value added software services to:
Hardware product companies
Software product companies
End-user in large and medium business organization.

### ABOUT DIGINET Software and Communications LTD.

The information is the hallmark of today's world. A drive for productivity and the ability to offer quality solutions on information superhighway are the key to development. Diginet (Pvt.) Ltd. has mirrored

The essence of true development since 1998 by enhancing growth with the presence of social justice. In promoting and cherishing the growth of those associated with clients who are the true partners in progress.
There is no shortcut to success so as in the case of IT industry too. It is never possible without innovation, an eye for vision, a strong will to succeed and unlimited quality service. Quality objectives, precise and time bound, are the root criteria for success and development is not an exception with diginet.

Diginet will leave no stone unturned to reach its customer to the topmost rung of ladder success. A result that is translated at Diginet, i.e. - in tune with technology with time and trust, truth and tradition, and requirement is the principle assets. Diginet has two divisions working at the moment - Training division as CompuHome and a software development division. It is the development division that is offering this project training as detailed in this document.

Diginet provides the state of the art technology like COM, COM+, Active-X, ASP, 3-tier solutions etc. and limited support of its clients in India and abroad. Diginet also provides Consultancy services for all IT related matters to its clients. With the revolving strategy and re-structuring, Diginet has now started offering Web based solutions and gearing towards providing the E-Commerce / M-Commerce solutions to its existing and new clients.

## Training Tie-up

**Diginet Software and Communications** having status of ATC of **TATA CONSULTANCY SERVICES** For committed to excellence in corporate training. We have the unique advantage of combining a management perspective with in-depth technical knowledge in all our training solutions. We offer a wide range of training programs to meet the requirements of corporate clients.

## Other Tie-ups

The following are the measure Tie-ups of Diginet in the areas of Software Development, Consultancy and Training.
Project Development Partner of TCS
Technology Team Development of RGSL
R&D and technology Team Development of APSON.com (p) LTD.
Prototype Development for VFM Software Solutions (p) LTD.

## Work related areas @ Diginet:

Business Application Development
Developing Device And Device Drivers
Web enabled applications development
Client / Server Applications Development
Embedded System
Research & Development in WAP and WEB related conversing technologies
Corporate training
High-end User Training (Vocational)
Industrial Automation
Data Processing
One-Wire and Tini Technology
Palm top/Hand Held PC Application Development/ E-CRM...

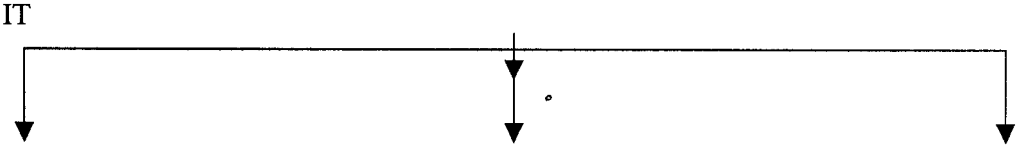
It is the policy of Diginet to design, develop, deliver, maintain and support high quality software solutions. This is done not only to meet the client's requirements but also to exceed their expectations by being their true partners to the ladder of success. Diginet extend its services to its clients by providing skilled manpower resources on contractual basis. This leads to a dedicated human resources development program.

## About Diginet's Project Training Program

In today's scenario of IT employment the expectations of employers are very high. The reason for that is very simple and obvious.
Since the entire stack of IT in the industry is just 8.7% and rests all is the other industry. Therefore the entire IT industry is based on the requirements raised from the other industry. There is a natural recession into other industry therefore the impact is on the IT.

Today IT industry is working in Unfocused manner with the focused objective, therefore even for the employment the industry is picking the manpower on time. Under this circumstances it is very difficult for the people who are planning to enter into the industry.
Our Project Training program helps in all the ways to the people who are planning to get into the industry, not only that our program helps people to even stay in the industry.
It is saying that once you define the purpose means will follow accordingly.

- IT



**BE/MCA** **Wide range of Skill sets** **experience**

Our program helps people to gain experience as well as to gain wide range of skill sets. We have a structured program, which makes people ready to be absorbed in six month by the industry with a shelter to people by providing a status as a Project Trainee till they are not getting an opportunity into the industry.

## ▪ ABOUT PEOPLE @DIGINET.

### *MANAGEMENT*

Diginet is managed by professionals with a wealth of experience in IT field. Diginet is promoted by **Mr. Chandra Bhushan**, an engineering graduate with more than 10 years of experience in IT and is co-promoted by **Mrs. Kiran C. B.** and **Mr. I. B. Sharma** and more with a combined experience of more than 15 years in IT.
M/S V. S. Ranganathan & Co., a leading financial service expert governs the Financial Management and Administration.
The Human Resources Management team is lead by **Mr. Sanjeev Kumar**, a HR professional.
Diginet has strategic and technical alliance with **M/S Namsoft Solutions**, Bangalore headed by Mr. Rajesh Ranjan, known for his technical abilities and consulting services in IT sector for more than10 years.
Apart from above, Diginet has strategic alliances with various leading industries such as - New Forge Company, Hi-Tech Industries, Shankala Industries, SAE Hydrolics Pvt. Ltd, M/S Flash Infotech (P) limited and many more for testing.

### *TECHNICAL*

The technical team at Diginet has a combined IT experienced of more than 15 years in tools as follows
C / C ++ / VB / VC++ / Java / Developer 2000
Access / MS SQL Server / Oracle
HTML / DHTML / ASCOM / COM+ / D-COM / MTS

# 2.0 System Study and Analysis

## 2.1 Software Requirement Specification

System requirement specification (SRS) specifies the requirements and features to be provided in the proposed system. The requirements for this VOIP system are:

**Point –to- point communication**

This VoIP system has been developing for a real time point –to-point voice communication over an IP network such as LAN or internet. Here, the communication can be established between two systems. The voice signals are digitized and compressed and converted to IP packets and then transmitted over an IP network using UDP protocol. At the other end, IP packets are uncompressed, quantized and converted to voice signals.

It includes a daemon which notifies the user when a client someone@somewhere.net is trying to connect the user. *Start Talk* will be providing real time voice communication over the Internet. This software is tested on two machines that will be placed side by side. The targeted time difference between the moments the user in machine A will Start Talking with the user on machine B will hear on the user on machine A, should be less than 100msec.

**Virtual Full Duplex System**

This VOIP system is using virtual full duplex system with the use of half duplex sound card. The virtual full duplex system will give the user the feeling that he is using a full duplex card system.

This software will automatically open a channel for either recording or playback by detecting whether the user is in listening mode or in talking mode.

**Fast transmission**

The voice signals would be transmitted in such a way that it should be fast. To achieve this purpose, consider the following points:

- **Protocol** – which should be fast and consumes less bandwidth.
- **Compression** – the compression of the signals will give a high data transfer rate and which utilizes less bandwidth. By using the IMA-ADPCM compression technique, PCM samples can be compressed in the ratio 4:1.

**No Loss of Data**

The proposed VoIP system guarantee that the data loss is maintained at its minimum. This can be accomplished by

- **Reducing noise** – Noise may result in the unwanted data or loss of data. It is required to take measures for reducing the noise by the **Squelch** process technique.
- **Compression algorithm** – The compression algorithm can effectively decompress and produce the original signals from the compressed data such that there should be a minimum data loss. The *Start Talk* includes an "*in house*" silence suppression or **Voice Activation Detector (VAD)** to conserve bandwidth and avoid transmission of background noise".

## 2.2 Existing System

- **ANALYSIS between Existing and Proposed System**

This phase is for understanding the existing system, which is usually just the starting activity of this phase and is relatively simple. . Analysis involves getting data from the people who are currently working in the existing system.

**Existing System**

- The client has to wait for a long time.
- The client cannot directly connect to the other client.
- Voice is associated with noise.
- Transmission process is slow.

## 2.3 Proposed System

- Noise is reduced.
- Compression of voice for fast and better communication.

Simple GUI is developed under X Windows.

# 3.0 Programming Environment

## 3.1 Hardware Configuration

- **Domain** : Networking

**Hardware Used**

| | | |
|---|---|---|
| Processor | : | Pentium 1 onwards |
| RAM | : | 64 MB |
| Hard Disk | : | 5MB(approx) |
| Monitor | : | VGA Color Monitor |
| Network Card (Speed) | : | 100mbps |

**Software Used**

Operating System : Linux

- Language Used : "C" language
- Scripting language : TCL/TK

- **For network communications**

⇒ Berkeley Sockets interface under the Linux OS .
⇒ The GUI for the Linux version will be developed for X Windows.

## 3.2 Description of Softwares and Tools used

Programming in C
Linux is distributed with a wide range of software-development tools. Many of these tools support the development of C and C++ applications. This module describes the tools that can be used to develop and debug C applications under Linux.We will see
What C is

- The GNU C compiler Debugging GCC applications with gdb

## What Is C?

C is a general-purpose programming language that has been around since the early days of the UNIX operating system. It was originally created by Dennis Ritchie at Bell Laboratories to aid in the development of UNIX. The first versions of UNIX were written using assembly language and a language called B. C was developed to overcome some of the shortcomings of B. Since that time, C has become one of the most widely used computer languages in the world.
Why did C gain so much support in the programming world? Some of the reasons that C is so commonly used include the following:

- It is a very portable language. Almost any computer that you can think of has at least one C compiler available for it, and the language syntax and function libraries are standardized across platforms. This is a very attractive feature for developers.

- Executable programs written in C are fast.

- C is the system language with all versions of UNIX/LINUX.

C has evolved quite a bit over the last 20 years. In the late 1980s, the American National Standards Institute published a standard for the C language known as ANSI C. This further helped to secure C's future by making it even more consistent between platforms. The 1980s also saw an object-oriented extension to C called C++.
The C compiler that is available for Linux is the GNU C compiler, abbreviated GCC. This compiler was created under the Free Software Foundation's programming license and is therefore freely distributable.

## The GNU C Compiler

The GNU C Compiler (GCC) that is packaged with the Red Hat Linux distribution is a fully functional, ANSI C compatible compiler.

## Invoking GCC

The GCC compiler is invoked by passing it a number of options and a number of filenames. The basic syntax for invoking gcc is this:
gcc [options] [filenames]
The operations specified by the command-line options will be performed on each of the files that are specified on the command line.

## GCC Options

There are more than 100 compiler options that can be passed to GCC. You will probably never use many of these options, but you will use some of them on a regular basis. Many of the GCC options consist of more than one character. For this reason you must specify each option with its own hyphen, and you cannot group options after a single hyphen as

you can with most Linux commands. For example, the following two commands are not the same:
The first command tells GCC to compile test.c with profile information for the prof command and also to store debugging information within the executable. The second command just tells GCC to compile test.c with profile information for the gprof command.
When you compile a program using gcc without any command-line options, it will create an executable file (assuming that the compile was successful) and call it a.out. For example, the following command would create a file named a.out in the current directory.
gcc test.c
To specify a name other than a.out for the executable file, you can use the -o compiler option. For example, to compile a C program file named count.c into an executable file named count, you would type the following command.

gcc -o count count.c
There are also compiler options that allow you to specify how far you want the compile to proceed. The -c option tells GCC to compile the code into object code and to skip the assembly and linking stages of the compile. This option is used quite often because it makes the compilation of multifile C programs faster and easier to manage. Object code files that are created by GCC have a .o extension by default.
The -s compiler option tells GCC to stop the compile after it has generated the assembler files for the C code. Assembler files that are generated by GCC have a .s extension by default. The -E option instructs the compiler to perform only the preprocessing compiler stage on the input files. When this option is used, the output from the preprocessor is sent to the standard output rather than being stored in a file.
The following file extensions are assumed to be used when using the language compilers, including gcc:

| Extension | Type of File |
|---|---|
| .a | Archive file |
| .c | C program file |
| .C, .cc, or .cxx | C++ program file |
| .h | A preprocessor (include) file |
| .i | An already preprocessed C file only needing compiling and assembling |
| .ii | An already preprocessed C++ file only needing compiling and assembling |
| .m | Objective-C program file |
| .o | Compiled object file |
| .s | Assembler source that had been preprocessed |
| .S | Assembler source which requires preprocessing |

## ▪ Optimization Options

When you compile C code with GCC, it tries to compile the code in the least amount of time and also tries to create compiled code that is easy to debug. Making the code easy to debug means that the sequence of the compiled code is the same as the sequence of the source code, and no code gets optimized out of the compile. There are many options that you can use to tell GCC to create smaller, faster executable programs at the cost of compile time and ease of debugging. Of these options the two that you will typically use are the -o and the -O2 options.
The -o option tells GCC to perform basic optimizations on the source code. These optimizations will in most cases make the code run faster. The -O2 option tells GCC to make the code as fast and small as it can. The -O2 option will cause the compilation speed to be slower than it is when using the -o option, but will typically result in code that executes more quickly.
In addition to the -o and -O2 optimization options, there are a number of lower-level options that can be used to make the code faster. These options are very specific and should only be used if you fully understand the consequences that using these options will have on the compiled code. For a detailed description of these options, refer to the GCC manual page by typing man gcc on the command line.

## ▪ Debugging and Profiling Options

GCC supports several debugging and profiling options. Of these options, the two that you are most likely to use are the -g option and the -pg option.

The -g option tells GCC to produce debugging information that the GNU debugger (gdb) can use to help you to debug your program. GCC provides a feature that many other C compilers do not have. With GCC you can use the -g option in conjunction with the -o option (which generates optimized code). This can be very useful if you are trying to debug code that is as close as possible to what will exist in the final product. When you are using these two options together you should be aware that some of the code that you have written will probably be changed by GCC when it optimizes it.

The -pg option tells GCC to add extra code to your program that will, when executed, generate profile information that can be used by the gprof program to display timing information about your program.

## ▪ Networking in Linux

Networking and Linux are terms that are almost synonymous. In a very real sense Linux is a product of the Internet or World Wide Web (WWW). Its developers and users use the web to exchange information ideas, code, and Linux itself is often used to support the networking needs of organizations. This chapter describes how Linux supports the network protocols known collectively as TCP/IP.

The TCP/IP protocols were designed to support communications between computers connected to the ARPANET, an American research network funded by the US government. The ARPANET pioneered networking concepts such as packet switching and protocol layering where one protocol uses the services of another. ARPANET was retired in 1988 but its successors (NSF[1] NET and the Internet) have grown even larger. What is now known as the World Wide Web grew from the ARPANET and is itself supported by the TCP/IP protocols. Unix ™ was extensively used on the ARPANET and the first released networking version of Unix ™ was 4.3 BSD. Linux's networking implementation is modeled on 4.3 BSD in that it supports BSD sockets (with some extensions) and the full range of TCP/IP networking. This programming interface was chosen because of its popularity and to help applications be portable between Linux and other Unix ™ platforms.

## ▪ A Data Communications Model

To discuss computer networking, it is necessary to use terms that have special meaning. Even other computer professionals may not be familiar with all the terms in the networking alphabet soup. As is always the case, English and computer-speak are not equivalent (or even necessarily compatible)
languages. Although descriptions and examples should make the meaning of the networking jargon more apparent, sometimes terms are ambiguous. A common frame of reference is necessary for understanding data communications terminology.An architectural model developed by the International Standards Organization (ISO) is frequently used to describe the structure and function of data communications protocols. This architectural model, which is called the *Open Systems Interconnect Reference Model* (OSI), provides a common
reference for discussing communications. The terms defined by this model are well understood and widely used in the data communications community - so widely used, in fact, that it is difficult to discuss data communications without using OSI's terminology. The OSI Reference Model contains seven *layers* that define the functions of data communications protocols. Each layer of the OSI model represents a function performed when data is transferred between cooperating applications across an intervening network. Figure  identifies each layer by name and provides a short functional description for it. Looking at this figure, the protocols are like a pile of building blocks stacked one upon another. Because of this appearance, the structure is often called a *stack* or *protocol stack.*

## ▪ The OSI Reference Model

A layer does not define a single protocol - it defines a data communications function that may be performed by any number of protocols. Therefore, each layer may contain multiple protocols, each providing a service suitable to the function of that layer. For example, a file transfer protocol and an electronic mail protocol both provide user services, and both are part of the Application Layer.
Every protocol communicates with its peer. A *peer* is an implementation of the same protocol in the equivalent layer on a remote system; i.e., the local file transfer protocol is the peer of a remote file transfer protocol. Peer-level communications must be standardized for successful communications to
take place. In the abstract, each protocol is concerned only with communicating to its peer; it does not care about the layer above or below it. However, there must also be agreement on how to pass data between the layers on a single computer,because every layer is involved in sending data from a local application to an equivalent remote application. The upper layers rely on the lower layers to transfer the data over the underlying network.Data is passed down the stack from one layer to the next, until it is transmitted over the network by the Physical Layer protocols. At the remote end, the data is passed up the stack to the receiving
application. The individual layers do not need to know how the layers above and below them function; they only need to know how to pass data to them. Isolating network communications functions in different layers minimizes the impact of technological change on the entire protocol suite. New applications can be added without changing the physical network, and new network hardware can be installed without rewriting the application software.

Although the OSI model is useful, the TCP/IP protocols don't match its structure exactly. Therefore,in our discussions of TCP/IP, we use the layers of the OSI model in the following way:

*Application Layer*
The Application Layer is the level of the protocol hierarchy where user-accessed network processes reside. In this text, a TCP/IP application is any network process that occurs above the Transport Layer. This includes all of the processes that users directly interact with, as well as other processes at this level that users are not necessarily aware of.

*Presentation Layer*
For cooperating applications to exchange data, they must agree about how data is represented.In OSI, this layer provides standard data presentation routines. This function is frequently handled within the applications in TCP/IP, though increasingly TCP/IP protocols such as XDR and MIME perform this function.

Session Layer
As with the Presentation Layer, the Session Layer is not identifiable as a separate layer in the TCP/IP protocol hierarchy. The OSI Session Layer manages the sessions (connection) between cooperating applications. In TCP/IP, this function largely occurs in the Transport Layer, and the term "session" is not used. For TCP/IP, the terms "socket" and "port" are used to describe the path over which cooperating applications communicate.

Transport Layer
Much of our discussion of TCP/IP is directed to the protocols that occur in the Transport Layer. The Transport Layer in the OSI reference model guarantees that the receiver gets the data exactly as it was sent. In TCP/IP this function is performed by the Transmission Control Protocol (TCP). However, TCP/IP offers a second Transport Layer service, User Datagram Protocol (UDP), that does not perform the end-to-end reliability checks.

Network Layer
The Network Layer manages connections across the network and isolates the upper layer protocols from the details of the underlying network. The Internet Protocol (IP), which isolates the upper layers from the underlying network and handles the addressing and delivery of data,is usually described as TCP/IP's Network Layer.

Data Link Layer
The reliable delivery of data across the underlying physical network is handled by the Data Link Layer. TCP/IP rarely creates protocols in the Data Link Layer. Most RFCs that relate to the Data Link Layer discuss how IP can make use of existing data link protocols.

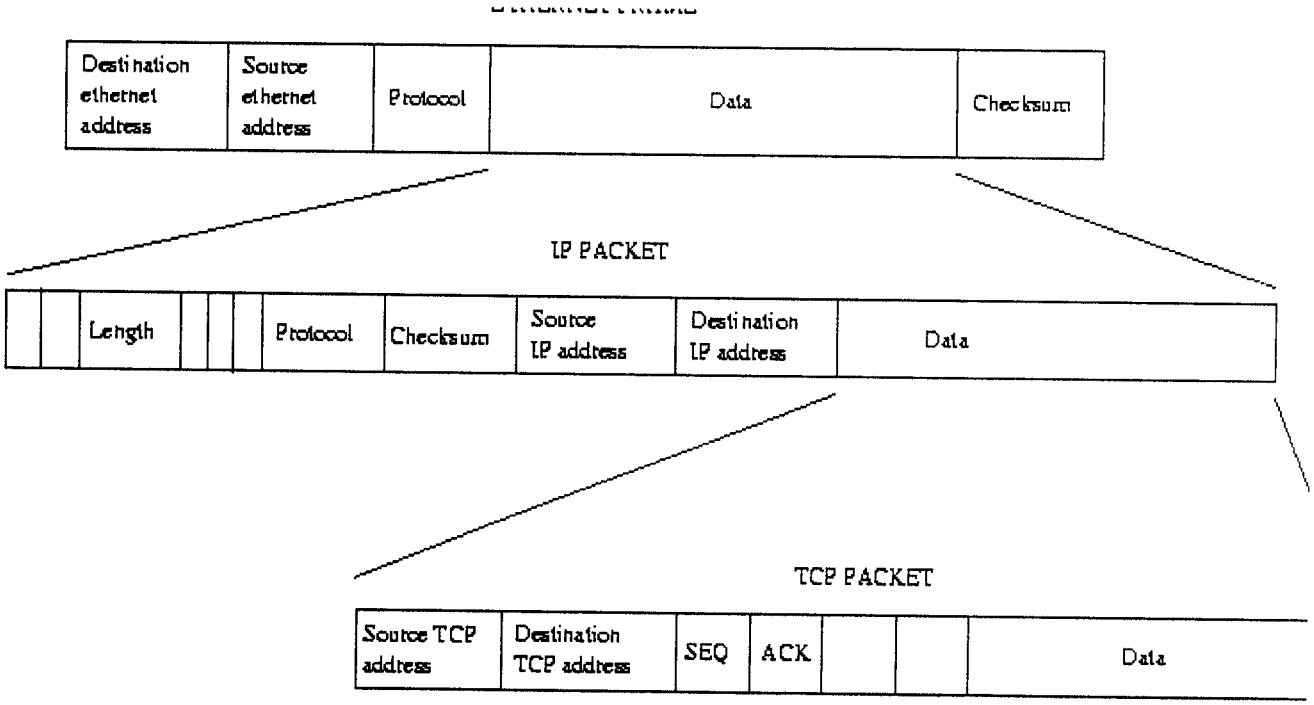
Physical Layer
The Physical Layer defines the characteristics of the hardware needed to carry the data transmission signal. Features such as voltage levels, and the number and location of interface pins, are defined in this layer. Examples of standards at the Physical Layer are interface connectors such as RS232C and V.35, and standards for local area network wiring such as IEEE 802.3. TCP/IP does not define physical standards - it makes use of existing standards.

The terminology of the OSI reference model helps us describe TCP/IP, but to fully understand it, we must use an architectural model that more closely matches the structure of TCP/IP.

## An Overview of TCP/IP Networking

This section gives an overview of the main principles of TCP/IP networking. It is not meant to be an exhaustive description, for that I suggest that you read . In an IP network every machine is assigned an IP address, this is a 32 bit number that uniquely identifies the machine. The WWW is a very large, and growing, IP network and every machine that is connected to it has to have a unique IP address assigned to it. IP addresses are represented by four numbers separated by dots, for example, 16.42.0.9. This IP address is actually in two parts, the *network* address and the *host* address. The sizes of these parts may vary (there are several classes of IP addresses) but using 16.42.0.9 as an example, the network address would be 16.42 and the host address 0.9. The host address is further subdivided into a *subnetwork* and a *host* address. Again, using 16.42.0.9 as an example, the subnetwork address would be 16.42.0 and the host address 16.42.0.9. This subdivision of the IP address allows organizations to subdivide their networks. For example, 16.42 could be the network address of the ACME Computer Company; 16.42.0 would be subnet 0 and 16.42.1 would be subnet 1. These subnets might be in separate buildings, perhaps connected by leased telephone lines or even microwave links. IP addresses are assigned by the network administrator and having IP subnetworks is a good way of distributing the administration of the network. IP subnet administrators are free to allocate IP addresses within their IP subnetworks.
Generally though, IP addresses are somewhat hard to remember. Names are much easier. linux.acme.com is much easier to remember than 16.42.0.9 but there must be some mechanism to convert the network names into an IP address. These names can be statically specified in the /etc/hosts file or Linux can ask a Distributed Name Server (DNS server) to resolve the name for it. In this case the local host must know the IP address of one or more DNS servers and these are specified in /etc/resolv.conf.

Whenever you connect to another machine, say when reading a web page, its IP address is used to exchange data with that machine. This data is contained in IP packets each of which have an IP header containing the IP addresses of the source and destination machine's IP addresses, a checksum and other useful information. The checksum is derived from the data in the IP packet and allows the receiver of IP packets to tell if the IP packet was corrupted during transmission, perhaps by a noisy telephone line. The data transmitted by an application may have been broken down into smaller packets which are easier to handle. The size of the IP data packets varies depending on the connection media; ethernet packets are generally bigger than PPP packets. The destination host must reassemble the data packets before giving the data to the receiving application. You can see this fragmentation and reassembly of data graphically if you access a web page containing a lot of graphical images via a moderately slow serial link.
Hosts connected to the same IP subnet can send IP packets directly to each other, all other IP packets will be sent to a special host, a gateway. Gateways (or routers) are connected to more than one IP subnet and they will resend IP packets received on one subnet, but destined for another onwards. For example, if subnets 16.42.1.0 and 16.42.0.0 are connected together by a gateway then any packets sent from subnet 0 to subnet 1 would have to be directed to the gateway so that it could route them. The local host builds up routing tables which allow it to route IP packets to the correct machine. For every IP




Figure

Figure: TCP/IP Protocol Layers

The IP protocol is a transport layer that is used by other protocols to carry their data. The Transmission Control Protocol (TCP) is a reliable end to end protocol that uses IP to transmit and receive its own packets. Just as IP packets have their own header, TCP has its own header. TCP is a connection based protocol where two networking applications are connected by a single, virtual connection even though there may be many subnetworks, gateways and routers between them. TCP reliably transmits and receives data between the two applications and guarantees that there will be no lost or duplicated data. When TCP transmits its packet using IP, the data contained within the IP packet is the TCP packet itself. The IP layer on each communicating host is responsible for transmitting and receiving IP packets. User Datagram Protocol (UDP) also uses the IP layer to transport its packets, unlike TCP, UDP is not a reliable protocol but offers a datagram service. This use of IP by other protocols means that when IP packets are received the receiving IP layer must know which upper protocol layer to give the data contained in this IP packet to. To facilitate this every IP packet header has a byte containing a protocol identifier. When TCP asks the IP layer to transmit an IP packet , that IP packet's header states that it contains a TCP packet. The receiving IP layer uses that protocol identifier to decide which layer to pass the received data up to, in this case the TCP layer. When applications communicate via TCP/IP they must specify not only the target's IP The IP protocol is a transport layer that is used by other protocols to carry

their data. The Transmission Control Protocol (TCP) is a reliable end to end protocol that uses IP to transmit and receive its own packets. Just as IP packets have their own header, TCP has its own header. TCP is a connection based protocol where two networking applications are connected by a single, virtual connection even though there may be many subnetworks, gateways and routers between them. TCP reliably transmits and receives data between the two applications and guarantees that there will be no lost or duplicated data. When TCP transmits its packet using IP, the data contained within the IP packet is the TCP packet itself. The IP layer on each communicating host is responsible for transmitting and receiving IP packets. User Datagram Protocol (UDP) also uses the IP layer to transport its packets, unlike TCP, UDP is not a reliable protocol but offers a datagram service. This use of IP by other protocols means that when IP packets are received the receiving IP layer must know which upper protocol layer to give the data contained in this IP packet to. To facilitate this every IP packet header has a byte containing a protocol identifier. When TCP asks the IP layer to transmit an IP packet , that IP packet's header states that it contains a TCP packet. The receiving IP layer uses that protocol identifier to decide which layer to pass the received data up to, in this case the TCP layer. When applications communicate via TCP/IP they must specify not only the address but also the *port* address of the application. A port address uniquely identifies an application and standard network applications use standard port addresses; for example, web servers use port 80. These registered port addresses can be seen in /etc/services.

This layering of protocols does not stop with TCP, UDP and IP. The IP protocol layer itself uses many different physical media to transport IP packets to other IP hosts. These media may themselves add their own protocol headers. One such example is the ethernet layer, but PPP and SLIP are others. An ethernet network allows many hosts to be simultaneously connected to a single physical cable. Every transmitted ethernet frame can be seen by all connected hosts and so every ethernet device has a unique address. Any ethernet frame transmitted to that address will be received by the addressed host but ignored by all the other hosts connected to the network. These unique addresses are built into each ethernet device when they are manufactured and it is usually kept in an SROM[2] on the ethernet card. Ethernet addresses are 6 bytes long, an example would be 08-00-2b-00-49-A4. Some ethernet addresses are reserved for multicast purposes and ethernet frames sent with these destination addresses will be received by all hosts on the network. As ethernet frames can carry many different protocols (as data) they, like IP packets, contain a protocol identifier in their headers. This allows the ethernet layer to correctly receive IP packets and to pass them onto the IP layer.

In order to send an IP packet via a multi-connection protocol such as ethernet, the IP layer must find the ethernet address of the IP host. This is because IP addresses are simply an addressing concept, the ethernet devices themselves have their own physical addresses. IP addresses on the other hand can be assigned and reassigned by network administrators at will but the network hardware responds only to ethernet frames with its own physical address or to special multicast addresses which all machines must receive. Linux uses the Address Resolution Protocol (or ARP) to allow machines to translate IP addresses into real hardware addresses such as ethernet addresses.

A host wishing to know the hardware address associated with an IP address sends an ARP request packet containing the IP address that it wishes translating to all nodes on the network by sending it to a multicast address. The target host that owns the IP address, responds with an ARP reply that contains its physical hardware address. ARP is not just restricted to ethernet devices, it can resolve IP addresses for other physical media, for example FDDI. Those network devices that cannot ARP are marked so that Linux does not attempt to ARP. There is also the reverse function, Reverse ARP or RARP, which translates phsyical network addresses into IP addresses. This is used by gateways, which respond to ARP requests on behalf of IP addresses that are in the remote network.

- The Linux TCP/IP Networking Layers

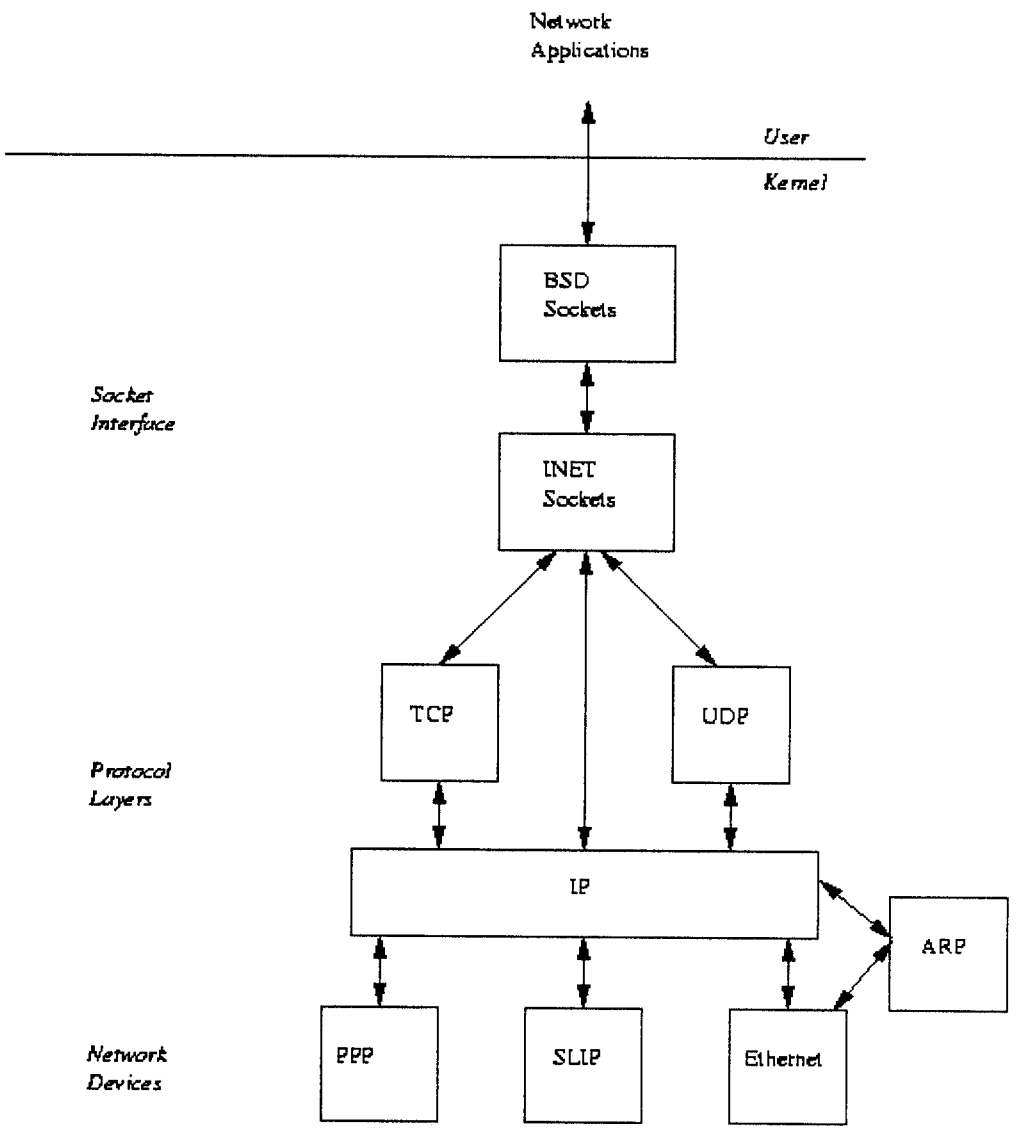



Figure: Linux Networking Layers

Just like the network protocols themselves, Figure shows that Linux implements the internet protocol address family as a series of connected layers of software. BSD sockets are supported by a generic socket management software concerned only with BSD sockets. Supporting this is the INET socket layer, this manages the communication end points for the IP based protocols TCP and UDP. UDP (User Datagram Protocol) is a connectionless protocol whereas TCP (Transmission Control Protocol) is a reliable end to end protocol. When UDP packets are transmitted, Linux neither knows nor cares if they arrive safely at their destination. TCP packets are numbered and both ends of the TCP connection make sure that transmitted data is received correctly. The IP layer contains code implementing the Internet Protocol. This code prepends IP headers to transmitted data and understands how to route incoming IP packets to either the TCP or UDP layers. Underneath the IP layer, supporting all of Linux's networking are the network devices, for example PPP and ethernet. Network devices do not always represent physical devices; some like the loopback device are purely software devices. Unlike standard Linux devices that are created via the mknod command, network devices appear only if the underlying software has found and initialized them. You will only see /dev/eth0 when you have built a kernel with the appropriate ethernet device driver in it. The ARP protocol sits between the IP layer and the protocols that support ARPing for addresses.

- What we Need Before You Start Networking

Before you start modifying system files, take a few minutes to determine a few basic pieces of information you'll need. It is advisable to write these down somewhere so that they will be handy when you need them, and also so that you won't enter two different values in two files, thereby causing major problems for the system.

## IP Address

First you need an IP address, a unique number for your machine. Every machine on the network has to be identified uniquely to allow proper routing. TCP/IP-based networks use 32-bit addresses to uniquely identify networks and all the devices that reside within that network. These addresses are called Internet addresses or IP addresses.
The 32 bits of the IP address are broken into four 8-bit parts. Each 8-bit part can then have valid numbers ranging from 0 to 255. In IP addresses, the four 8-bit numbers are separated by a period, a notation called dotted quad. Examples of dotted quad IP addresses are 255.255.255.255 and 147.14.123.8.

For convenience, IP addresses are divided into two parts: the network number and the device number within that network. This separation into two components allows devices on different networks to have the same host number. However, because the network number is different, the devices are still uniquely identified.
For connection to the Internet, IP addresses are assigned by the Internet Network Information Center (NIC) based on the size of the network. Anyone who wants to connect to the Internet must register with the NIC to avoid duplication of network addresses. If you don't plan to connect to the Internet, you are free to create your own numbering scheme, although future expansion and integration with Internet-using networks can cause serious problems.

For maximum flexibility, IP addresses are assigned according to network size. Networks are divided into three categories: Class A, Class B, and Class C. The three network classes break the 32-bit IP addresses into different sizes for the network and host identifiers.

A Class A address uses one byte for the network address and three bytes for the device address, allowing more than 16 million different host addresses. Class B networks use two bytes for the network and two bytes for the host. Because 16 bits allows more than 65,000 hosts, only a few large companies will be limited by this type of class. Class C addresses have three bytes for the network and one for the number of hosts. This provides for a maximum of 254 hosts (the numbers 0 and 255 are reserved) but many different network IDs. The majority of networks are Class B and Class C.
You do have a limitation as to the first value. A Class A network's first number must be between 0 and 127, Class B addresses are between 128 and 191, and Class C addresses are between 192 and 223. This is because of the way the first byte is broken up, with a few of the bits at the front saved to identify the class of the network. Also, you can't use the values 0 and 255 for any part, because they are reserved for special purposes.
Messages sent using TCP/IP use the IP address to identify sending and receiving devices, as well as any routing information put within the message headers. If you are going to connect to an existing network, you should find out what their IP addresses are and what numbers you can use. If you are setting up a network for your own use but plan to connect to the Internet at some point, contact the Network Information Center for an IP address. On the other hand, if you are setting up a network for your own use and don't plan to have more than a telephone connection to other networks (including the Internet), you can make up your own IP addresses.
If you are only setting up a loopback driver, you don't even need an IP address. The default value for a loopback driver is 127.0.0.1.

Network Mask
Next, you need a network mask. This is pretty easy if you have picked out an IP address. The network mask is the network portion of the IP address set to the value 255, and it's used to blank out the network portion to determine routing.
If you have a Class C IP address (three bytes for network and one for devices), your network mask is 255.255.255.0. A Class B network has a network mask of 255.255.0.0, and a Class A network mask is 255.0.0.0.

If you are configuring only a loopback driver, your network mask is 255.0.0.0 (Class A).
If you have a complex network setup with subnets or share addresses, you should consult your network administrator for more information about setting your network mask and IP address.

- Network Address

The network address is, strictly speaking, the IP address bitwise-ANDed to the netmask. In English, what this means is that it's the network portion of your IP address, so if your IP address is 147.120.46.7 and it's a Class B network, the network address is 147.120.0.0. To get your own network address, just drop the device-specific part of the IP address and set it to zero. A Class C network with an IP address of 201.12.5.23 has a network address of 201.12.5.0.
If you're only working with a loopback address, you don't need a network mask.

- Broadcast Address

The broadcast address is used when a machine wants to send the same packet to all devices on the network. To get your broadcast address, you set the device portion of the IP address to 255. Therefore, if you have the IP address 129.23.123.2, your broadcast address will be 129.23.123.255. Your network address will be 129.23.123.0.
If you are configuring only a loopback driver, you needn't worry about the broadcast address.

- Gateway Address

The gateway address is the IP address of the machine that is the network's gateway out to other networks (such as the Internet). You need a gateway address only if you have a network that has a dedicated gateway out. If you are configuring a small network for your own use and don't have a dedicated Internet connection, you don't need a gateway address.
Normally, gateways have the same IP address as your machines, but they have the digit 1 as the device number. For example, if your IP address is 129.23.123.36, chances are that the gateway address is 129.23.123.1. This convention has been used since the early days of TCP/IP.
Loopback drivers do not require a gateway address, so if you are configuring your system only for loopback, ignore this address.

- Nameserver Address

Many larger networks have a machine whose purpose is to translate IP addresses into English-like names, and vice versa. It is a lot easier to call a machine bobs_pc instead of 123.23.124.23. This translation is done with a system called the Domain Name System (DNS). If your network has a name server, that's the address you need. If you want to have your own machine act as a name server (which requires some extra configuration not mentioned here), use the loopback address 127.0.0.1.
Loopback drivers don't need a name server because the machine only talks to itself. Therefore, you can ignore the nameserver address if you are only configuring a loopback driver.

## ▪ Setting Up the Dummy Interface

Dummy interface is a bit of a trick to give your machine an IP address to work with when it uses only SLIP and PPP interfaces. A dummy interface solves the problem of a stand-alone machine (no network cards connecting it to other machines) whose only valid IP address to send data to is the loopback driver (127.0.0.1). While SLIP and PPP may be used for connecting your machine to the outside world, when the interface is not active you have no internal IP address that applications can use.
The problem arises with some applications that require a valid IP address to work. Some word processors and desktop layout tools, for example, require the TCP/IP system to be operational with an IP address for the target machine. The dummy interface essentially sets an IP address for your local machine that is valid as far as TCP/IP is concerned, but doesn't really get used except to fool applications.
Creating a dummy interface is very simple. If your machine has an IP address already assigned for it in the /etc/hosts file, all you need to do is set up the interface and create a route. The two commands required are

ifconfig dummy machine_name

route add machine_name
where machine_name is your local machine's name (such as darkstar). This will create a link to your own IP address. If you do not have an IP address for your machine in the /etc/hosts file, add one before you create the dummy interface.

## ▪ Configuration Files

Configuring Linux for TCP/IP is not difficult because only a few configuration files need to have the information about IP address and such added to them. You can do this with any editor as long as it saves the files in ASCII format. It is advisable to make copies of the configuration files before you modify them, just in case you damage the format in some way.
Many of these files are similar in every version of UNIX, including most versions of Linux, except for one or two slight naming variations. If you've ever set up a UNIX system (or snooped around one in detail), these files and steps might seem familiar. If you haven't done anything with Linux or UNIX before, just take it one step at a time and follow the instructions!

## ▪ rc Files

Linux reads the rc (run command) files when the system boots. The init program initiates the reading of these files, and they usually serve to start processes such as mail, printers, cron, and so on. They are also used to initiate TCP/IP connections. Most Linux systems have the rc command files in the directory /etc/rc.d.
The files of interest to TCP/IP, at least as far as Red Hat Linux is concerned, are under the /etc/rc.d/rc2.d directory. The files in this directory start different TCP/IP services. The names of most of the files identify their purposes. For example, the file K20nfs deals with starting the NFS service.

If you want to change services that are started when Red Hat Linux boots, you can edit these files with as ASCII editor. Identify the file that is involved with the service you need to modify (you can often use grep to find the right file, or you can look through them all with more). Whichever file is involved, look for a line that refers to the service. In some cases, this line or lines will be commented out (have a pound sign as the first character) to prevent the system from trying to run them. If the lines are commented out, remove the comment symbol.

- /etc/hosts

The /etc/hosts file is a simple list of IP addresses and the hostnames to which they correspond. This is a good location to list all your favorite machines so that you can use the name and have the system look up the IP address. On very small networks, you can add all the machines in the network here and avoid the need to run the nameserver.
Every /etc/hosts file will have an entry for localhost (also called loopback, IP address 127.0.0.1) and probably one for your machine if you named it when you installed the software. If you didn't supply a name and there is no line other than localhost, you can add it now. Use an editor and set your IP address and machine name. Don't bother adding too many other machines until you're sure the network works properly! Here's a sample /etc/hosts file:

127.0.0.1 localhost

147.12.2.42 merlin.tpci merlin
You will notice that the format is quite simple: an IP address in one column and the name in another column, separated by tabs. If the machine may have more than one name, supply them all. In the example, which uses random numbers for the IP address, the machine 147.12.2.42 has the name merlin. Since it is also part of a larger network called tpci, the machine can be addressed as merlin.tpci. Both names on the line ensure that the system can resolve either name to the same address.
You can expand the file a little if you want by adding other machines on your local network, or those you will communicate with regularly:

127.0.0.1 localhost

147.12.2.42 merlin.tpci merlin

147.12.2.43 wizard.tpci wizard

147.12.2.44 arthur.tpci arthur bobs_machine

147.12.2.46 lancelot.tpci lancelot
In this example, there are several machines from the same network (the same network mask). One has three different names.

- /etc/networks

The /etc/networks file lists names and IP address of your own network and other networks you connect to frequently. This file is used by the route command. One advantage of this file is that it lets you call remote networks by name, so instead of typing 149.23.24, you can type eds_net.
The /etc/networks file should have an entry for every network that will be used with the route command if you plan on using the network name as an identifier. If there is no entry, errors will be generated, and the network won't work properly. On the other hand, if you don't need to use a network name instead of its IP address, then you can skip the /etc/networks file.
A sample /etc/networks file using random IP addresses is shown next. Remember that you need only the network mask and not the device portion of a remote machine's IP address, although you must fill in the rest with zeroes.

loopback 127.0.0.0

localnet 147.13.2.0

eds_net 197.32.1.0

big_net 12.0.0.0
At a minimum, you should have a loopback and localnet address in the file.

- /etc/host.conf

The system uses the host.conf file to resolve hostnames. It usually contains two lines that look like this:

order hosts, bind

multi on
These tell the system to first check the /etc/hosts file, then check the nameserver (if one exists) when trying to resolve a name. The multi entry lets you have multiple IP addresses for a machine in the /etc/hosts file (which happens with gateways and machines on more than one network).
If your /etc/host.conf file looks like these two lines, you don't need to make any changes at all.

- resolv.conf

The resolv.conf file is used by the name resolver program. It gives the address of your name server (if you have one) and your domain name (if you have one). You will have a domain name if you are on the Internet.

A sample resolv.conf file for the system merlin.tpci.com has an entry for the domain name, which is tpci.com (merlin is the name of an individual machine):

domain tpci.com
If a name server is used on your network, you should add a line that gives its IP address:

domain tpci.com

nameserver 182.23.12.4

If there are multiple name servers, which is not unusual on a larger network, each name server should be specified on its own line.
If you don't have a domain name for your system, you can safely ignore this file for the moment.

- /etc/protocols

LINUX systems use the /etc/protocols file to identify all the transport protocols available on the system and their respective protocol numbers. (Each protocol supported by TCP/IP has a special number, but that's not really important at this point.) Usually, this file is not modified but is maintained by the system and updated automatically as part of the installation procedure when new software is added.
The /etc/protocols file contains the protocol name, its number, and any alias that may be used for that protocol. A sample /etc/protocols file looks like this:

# Internet protocols (IP)

ip 0 IP

icmp 1 ICMP

ggp 3 GGP

tcp 6 TCP

egp 8 EGP

pup 12 PUP

udp 17 UDP

hello 63 HELLO

If your entries don't match this, don't worry. You shouldn't have to make any changes to this file at all, but you should know what it does.

- /etc/services

The /etc/services file identifies the existing network services. This file is maintained by software as it is installed or configured.
This file consists of the service name, a port number, and the protocol type. The port number and protocol type are separated by a slash, following the conventions mentioned in previous chapters. Any optional service alias names follow. Here's a short extract from a sample /etc/services file:

```
# network services

echo 7/tcp

echo 7/udp

discard 9/tcp sink null

discard 9/udp sink null

ftp 21/tcp

telnet 23/tcp

smtp 25/tcp mail mailx

tftp 69/udp

# specific services

login 513/tcp

who 513/udp whod
```

You shouldn't change this file at all, but you do need to know what it is and why it is there to help you understand TCP/IP a little better.

- /etc/hostname or /etc/HOSTNAME

The file /etc/HOSTNAME is used to store the name of the system you are on. (Red Hat Linux uses the uppercase version of the name.) This file should have your local machine's name in it:

```
merlin.tpci
```

That's all it needs. The host name is used by most protocols on the system and many applications, so it is important for proper system operation. The host name can be

changed by editing the system file and rebooting the machine, although many operating systems provide a utility program to ensure that this process is performed correctly.
Linux systems have a utility called hostname, which displays the current setting of the system name, as well as the uname program, which can give the node name with the command uname -n. When issued, the hostname and uname commands echo the local machine name, as the following sample session shows:

```
$ hostname

merlin.tpci.com

$ uname -n

merlin
```

All the configuration files necessary for TCP/IP to function have now been set properly, so you should be able to reboot the machine and see what happens.

## ▪ Network Programming

Here we look at the basic concepts we need for network programming:
Ports and sockets
Record and file locking
Interprocess communications

It is impossible to tell you how to program applications for a network in just a few pages. Indeed, the best reference to network programming available takes almost 800 pages in the first volume alone! If you really want to do network programming, you need a lot of experience with compilers, TCP/IP, network operating systems, and a great deal of patience.
For information on details of TCP/IP, check the book Teach Yourself TCP/IP in 14 Days by Tim Parker (Sams).

## ▪ Ports and Sockets

Network programming relies on the use of sockets to accept and transmit information. Although there is a lot of mystique about sockets, the concept is actually very simple to understand.
Most applications that use the two primary network protocols, Transmission Control Protocol (TCP) or User Datagram Protocol (UDP) have a port number that identifies the application. A port number is used for each different application the machine is handling, so it can keep track of them by numbers instead of names. The port number makes it easier for the operating system to know how many applications are using the system and which services are available.
In theory, port numbers can be assigned on individual machines by the system administrator, but some conventions have been adopted to allow better communications. This convention enables the port number to identify the type of service that one system is

requesting from another. For this reason, most systems maintain a file of port numbers and their corresponding services.
Port numbers are assigned starting from the number 1. Normally, port numbers above 255 are reserved for the private use of the local machine, but numbers between 1 and 255 are used for processes requested by remote applications or for networking services.
Each network communications circuit that goes into and out of the host computer's TCP application layer is uniquely identified by a combination of two numbers, together called the socket. The socket is composed of the IP address of the machine and the port number used by the TCP software.
Because there are at least two machines involved in network communications, there will be a socket on both the sending and receiving machine. The IP address of each machine is unique, and the port numbers are unique to each machine, so socket numbers will also be unique across the network. This enables an application to talk to another application across the network based entirely on the socket number.
The sending and receiving machines maintain a port table that lists all active port numbers. The two machines involved have reversed entries for each session between the two, a process called binding. In other words, if one machine has the source port number 23 and the destination port number set at 25, then the other machine will have its source port number set at 25 and the destination port number set at 23.

## ▪ The INET Socket Layer

The INET socket layer supports the internet address family which contains the TCP/IP protocols. As discussed above, these protocols are layered, one protocol using the services of another. Linux's TCP/IP code and data structures reflect this layering. Its interface with the BSD socket layer is through the set of Internet address family socket operations which it registers with the BSD socket layer during network initialization. These are kept in the pops vector along with the other registered address families. The BSD socket layer calls the INET layer socket support routines from the registered INET proto_ops data structure to perform work for it. For example a BSD socket create request that gives the address family as INET will use the underlying INET socket create function. The BSD socket layer passes the socket data structure representing the BSD socket to the INET layer in each of these operations. Rather than clutter the BSD socket wiht TCP/IP specific information, the INET socket layer uses its own data structure, the sock which it links to the BSD socket data structure. This linkage can be seen in Figure 10.3. It links the sock data structure to the BSD socket data structure using the data pointer in the BSD socket. This means that subsequent INET socket calls can easily retrieve the sock data structure. The sock data structure's protocol operations pointer is also set up at creation time and it depends on the protocol requested. If TCP is requested, then the sock data structure's protocol operations pointer will point to the set of TCP protocol operations needed for a TCP connection.

## ▪ Socket Programming

Linux supports BSD style socket programming. Both connection-oriented and connectionless types of sockets are supported. In connection-oriented communication, the server and client establish a connection before any data is exchanged. In connectionless communication, data is exchanged as part of a message. In either case, the server always starts up first, binds itself to a socket, and listens to messages. How the server attempts to listen depends on the type of connection for which you have programmed it.

## ▪ What is a socket?

Socket is a way to speak to other programs using standard Linux file descriptors.
It acts as communication endpoint.When Linux programs do any sort of I/O, they do it by reading or writing to a file descriptor. A file descriptor is simply an integer associated with an open file, but the file can be a network connection, a FIFO, a pipe, a terminal, a real on-the-disk file, or just about anything else.

## ▪ structs and Data Handling

In a network there are two byte orderings: most significant byte (sometimes called an "octet") first, or least significant byte first. The former is called **"Network Byte Order"**.( "Network Byte Order" is also know as "Big-Endian Byte Order".) and the later is called **"Host Byte Order"**.

The functions used for byte order conversions are:

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

The built-in structures used in socket programming are:

**1)struct sockaddr.** This structure holds socket address information for many types of sockets:

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

**2) struct sockaddr_in:**To deal with struct sockaddr, programmers created a parallel

structure: struct sockaddr_in ("in" for "Internet".)

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

This structure makes it easy to reference elements of the socket address. Note that *sin_zero* (which is included to pad the structure to the length of a struct sockaddr) should be set to all zeros with the function memset(). Also, and this is the *important* bit, a pointer to a struct sockaddr_in can be cast to a pointer to a struct sockaddr and vice-versa. So even though socket() wants a struct sockaddr*, you can still use a struct sockaddr_in and cast it at the last minute! Also, notice that *sin_family* corresponds to *sa_family* in a struct sockaddr
and should be set to "AF_INET". Finally, the *sin_port* and *sin_addr* must be in *Network Byte Order*!

3) struct in_addr : Internet address (a structure for historical reasons)

*struct in_addr {*
*unsigned long s_addr; // that's a 32-bit long, or 4 bytes*
*};*

## ▪ IP Addresses and How to Deal With Them

**1)inet_addr()**: converts an IP address in numbers-and-dots notation into an unsigned

long.

**2)inet_aton():** ("aton" means "ascii to network"):

```
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>

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

**3)inet_ntoa():** ("ntoa" means "network to ascii") like this:

- System Calls :

**1)socket()**– To get the File Descriptor

***synopsis***

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

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

*domain* should be set to "AF_INET",
*type* argument tells the kernel what kind of socket this is: SOCK_STREAM or SOCK_DGRAM.
*protocol* to "0" to have socket() choose the correct protocol based on the *type*

**2)bind()**–to associate socket with a port on the local machine

***synopsis***

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

int bind(int sockfd, struct sockaddr *my_addr, int addrlen);

*sockfd* is the socket file descriptor returned by socket(). *my_addr* is a pointer to a struct sockaddr that contains information about your address, namely, port and IP address. *addrlen* can be set to sizeof(struct sockaddr).

bind() also returns -1 on error and sets *errno* to the error's value.

**3)connect()**– to connect to a socket on remote host

synopsis

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

int connect(int sockfd, struct sockaddr *serv_addr, int addrlen);

sockfd is our friendly neighborhood socket file descriptor, as returned by the socket() call, serv_addr is a struct sockaddr containing the destination port and IP address, and addrlen can be set to sizeof(struct sockaddr).
return value from connect()–it'll return -1 on error and set the variable errno.

**4)listen()**– waits for an incoming connection

***synopsis***

int listen(int sockfd, int backlog);

*sockfd* is the usual socket file descriptor from the socket() system call. *backlog* is the number of connections allowed on the incoming queue.
incoming connections are going to wait in this queue until you accept() them

listen() returns -1 and sets *errno* on error.

**5)accept()**– gets the pending connection and returns new file descriptor tp be used for this single connection.The original one is still listening on your port and the newly created one is finally ready to send() and recv().

***synopsis***

#include <sys/socket.h>

int accept(int sockfd, void *addr, int *addrlen);

*sockfd* is the listen()ing socket descriptor.
*addr* will usually be a pointer to a local struct sockaddr_in. This is where the information about the incoming connection will go (and with it you can determine which host is calling you from which port). *addrlen* is a local integer variable that should be set to sizeof(struct sockaddr_in)

accept() returns -1 and sets *errno* if an error occurs.

**6)send() and recv()**–These two functions are used for communicating over stream sockets or connected datagram sockets.

***Synopsis***

int send(int sockfd, const void *msg, int len, int flags);

*sockfd* is the socket descriptor you want to send data to (whether it's the one returned by socket() or the one you got with accept().) *msg* is a pointer to the data you want to send, and *len* is the length of that data in bytes. Just
set *flags* to 0.

-1 is returned on error, and *errno* is set to the error number.

int recv(int sockfd, void *buf, int len, unsigned int flags);

*sockfd* is the socket descriptor to read from, *buf* is the buffer to read the information into, *len* is the maximum length of the buffer, and *flags* can again be set to 0.

recv() returns the number of bytes actually read into the buffer, or -1 on error (with *errno* set, accordingly.). recv() can return 0,this can means the remote side has closed the connection on you.

**7)sendto() and recvfrom()**–These two functions are used for communicating over

connectionless datagram sockets.

***Synopsis***

int sendto(int sockfd, const void *msg, int len, unsigned int flags, const struct sockaddr *to, int tolen);

This call is basically the same as the call to send() with the addition of two other pieces of information. *to* is a pointer to a struct sockaddr which contains the destination IP address and port. *tolen* can simply be set to sizeof(struct sockaddr).Just like with send(), sendto() returns the number of bytes actually sent (which, again, might be less than the number of bytes you told it to send!), or -1 on error.

int recvfrom(int sockfd, void *buf, int len, unsigned int flags, struct sockaddr *from, int *fromlen);

Again, this is just like recv() with the addition of a couple fields. *from* is a pointer to a local struct sockaddr that will be filled with the IP address and port of the originating machine. *fromlen* is a pointer to a local int that should be initialized to sizeof(struct sockaddr). When the function returns, *fromlen* will contain the length of the address actually stored in *from*. recvfrom() returns the number of bytes received, or -1 on error (with *errno* set accordingly.)

**8)close() and shutdown()**– These calls are used to close the connection on the socket

descriptor.

***Synopsis***

close(sockfd);This will prevent any more reads and writes to the socket. Anyone attempting to read or write the socket on the remote end will receive an error.

Synopsis

int shutdown(int sockfd, int how);

sockfd is the socket file descriptor you want to shutdown, and how is one of the following:
- 0 – Further receives are disallowed
- 1 – Further sends are disallowed
- 2 – Further sends and receives are disallowed (like close())

shutdown() returns 0 on success, and -1 on error (with errno set accordingly.)

9)getpeername()–The function getpeername() will tell you who is at the other end of a connected stream socket.

Synopsis

#include <sys/socket.h>

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

sockfd is the descriptor of the connected stream socket, addr is a pointer to a struct sockaddr (or a struct sockaddr_in) that will hold the information about the other side of the connection, and addrlen is a pointer to an int, that should be initialized to sizeof(struct sockaddr).

The function returns -1 on error and sets errno accordingly.

10)gethostname()– It returns the name of the computer that your program is running on.

The name can then be used by gethostbyname(), below, to determine the IP address of your local machine.

Synopsis

#include <unistd.h>
int gethostname(char *hostname, size_t size);

The arguments are simple: hostname is a pointer to an array of chars that will contain the hostname upon the function's return, and size is the length in bytes of the hostname array.

The function returns 0 on successful completion, and -1 on error, setting errno as usual.

11)gethostbyname()- returns IP address when hostname is given

Synopsis

#include <netdb.h>

struct hostent *gethostbyname(const char *name);

It returns a pointer to a struct hostent, the layout of which is as follows:

struct hostent {
char *h_name;
char **h_aliases;
***int h_addrtype;***
***int h_length;***
***char **h_addr_list;***
*};*

the descriptions of the fields in the struct hostent are:
• *h_name* – Official name of the host.
• *h_aliases* – A NULL-terminated array of alternate names for the host.
• *h_addrtype* – The type of address being returned; usually *AF_INET.*
• *h_length* – The length of the address in bytes.
• *h_addr_list* – A zero-terminated array of network addresses for the host. Host addresses are in Network Byte Order.
• *h_addr* – The first address in *h_addr_list.*

gethostbyname() returns a pointer to the filled struct hostent, or NULL on error. (But *errno* is *not* set–*h_errno* is set instead.)

12)select()–Synchronous I/O Multiplexing

select() gives you the power to monitor several sockets at the same time. It'll tell you which ones are ready for reading, which are ready for writing, and which sockets have raised exceptions,

*Synopsis*

#include <sys/time.h>
#include <sys/types.h>
#include <unistd.h>

int select(int numfds, fd_set *readfds, fd_set *writefds, fd_set *exceptfds, struct timeval *timeout);

The function monitors "sets" of file descriptors; in particular *readfds*, *writefds*, and *exceptfds*. If you want to see if you can read from standard input and some socket descriptor, *sockfd*, just add the file descriptors 0 and *sockfd* to the set *readfds*. The parameter *numfds* should be set to the values of the highest file descriptor plus one. In this example, it should be set to *sockfd+1*, since it is assuredly higher than standard input (0). When select() returns, *readfds* will be modified to reflect which of the file descriptors you selected which is ready for reading. You can test them with the macro FD_ISSET(), below.

Each set is of the type fd_set. The
following macros operate on this type:
- FD_ZERO(fd_set *set) – clears a file descriptor set
- FD_SET(int fd, fd_set *set) – adds *fd* to the set
- FD_CLR(int fd, fd_set *set) – removes *fd* from the set
- FD_ISSET(int fd, fd_set *set) – tests to see if *fd* is in the set

***struct timeval :*** This time structure allows you to specify a timeout period. If the time is exceeded and select() still hasn't found any ready file descriptors, it'll return so you can continue processing.

The struct timeval has the follow fields:

```
struct timeval {
int tv_sec; // seconds
int tv_usec; // microseconds
};
```

Just set *tv_sec* to the number of seconds to wait, and set *tv_usec* to the number of microseconds to wait. There are 1,000 microseconds in a millisecond, and 1,000 milliseconds in a second. Thus, there are 1,000,000 microseconds in a second.When the function returns, *timeout might* be updated to show the time still remaining. This depends on what flavor of Linux you're running.Standard Linux timeslice is around 100 milliseconds, so you might have to wait that long no matter how small you set your struct timeval. If you set the fields in your struct timeval to 0, select() will timeout immediately,effectively polling all the file descriptors in your sets. If you set the parameter *timeout* to NULL, it will never
timeout, and will wait until the first file descriptor is ready.

If you're on a line buffered terminal, the key you hit should be RETURN or it will time out anyway. Now, some of you might think this is a great way to wait for data on a datagram socket–and you are right: it *might* be. Some Unices can use select in this

manner, and some can't. You should see what your local man page says on the matter if you want to attempt it.Some Unices update the time in your struct timeval to reflect the amount of time still remaining before a timeout. But others do not. Don't rely on that occurring if you want to be portable.

## ▪ Client-server applications

The client-server application paradigm is besf understood when cast in the concept of *distributed system.*
A distributed system is a collection of applications, running on various (possibly heterogeneous) machines, whose distribution is transparent to the user so that the system appears as if running on the user's local machine. This is in contrast to a network, where the user is aware that there are several machines, and their location, storage replication, load balancing and functionality is not transparent.
The expression *client-server* refers to a software partitioning paradigm in which a distributed system is split between one or more server applications which accept requests, according to some protocol, from (distributed) client applications, asking for information or action. There may be either one centralised server or several distributed ones. This model allows clients and servers to be placed independently on nodes in a network.
Note that this paradigm places clients and servers asymmetrically with respect to each other: servers are there because they offer a certain computational/informational service, and clients make use of it to their advantage. This is in contrast with a peer-to-peer computation/communication model, where all applications in the distributed environment are more or less the same, and use/offer services from/to other ones.
Examples of client-server applications are the name-server/name-resolver relationship in DNS (Domain Name System), the file-server/file-client relationship in NFS (Network File System) and the screen server/client application split in the X Window System.
A *server*, generally speaking, is a program which provides some service to other (client) programs. The connection between client and server is normally by means of message passing, often over a network, and uses some protocol to encode the client's requests and the server's responses. The server may run continuously (as a daemon), waiting for requests to arrive or it may be invoked by some higher level daemon which controls a number of specific servers (e.g., inetd on Linux).

## ▪ Addressing

. The client would then pass this address to the As stated above, one of the goals of the client-server paradigm is to be able to arbitrarily place clients and servers on a network of interconnected machines. This means, for example, that a client application Cl running on a certain machine MCl must be (ideally) able to work properly - i.e. to exchange information with a server Sr - independently on the machine MSr the server runs on: the machine might be substituted with another one, or the server might be replaced with another program offering the same kind of service, without the client taking notice of it.
Let's see how this goal can be achieved, and let's proceed step by step, starting with the server application. The application will in general run as a process, or a set of related processes on a machine. Let's assume, since this is the most common case, that there's

only one process dedicated to the communication needs of the server application. The clients then need a way to find this process.

The simplest possible way is to hardwire the network address of the server's machine in the client programtransport/network services of the operating system of the its machine, so that the communication be established. This is clearly non-transparent from the client's perspective, since the location of the server is explicitly accounted. Moreover, in this scheme a machine could offer only one kind of service by means of one single server application: there's no way to distinguish between two servers running on the same machine.

Since the client is not interested in the machine *per se*, but in the server process, a possible way to solve the second problem might be to specify a process number along with the server machine address: something like 6234@132.206.72.34, where 6234 is the process id of a program running on the machine whose network address is 132.206.72.34 (which, btw, is a sample IP address). However, this would generally be too much of a constraint for the operating system of the server machine, since process id.s are generally assigned independently of the programs they run.
A better scheme is one in which the server registers itself with the OS at a fixed *port*, i.e. at a logical communication address which is guaranteed to be unique within the machine, so that the pair portnumber@machine be unique over the network. The client would then address the server at the machine through its port number in order to establish a connection. Furthermore, the server OS might assign logical names to the ports corresponding to service names (i.e. map service names to ports and vice versa), so that the client might request connections for a certain service at a given machine in the form servicename@machine, the server OS providing for the translation of the service name into the appropriate port number (thus freeing the server programmer or administrator from having to always use the same port number).

However this scheme is still far from the ideal one, since it still assumes that the client has explicit knowledge of the machine offering the service. Clearly it's needed a way to assign machine-independent identifiers to servers. A solution, useful for network that allow for broadcasting (i.e. sending messages to be received by all connected stations) is to let the servers choose their own identifier, say random number from a large space like the set of the 128-bit integers. Upon choosing an address, the server would broadcast a message to all the connected stations, asking whether that number is already taken by a previously set-up application, and if not that would be its number. The client's OS could then find the server for a certain service in two ways:
by always sending a broadcast message *asking for a certain service*, and containing the client's machine address to direct the reply (like a self-addressed envelope), and waiting for the server to reply;

By sending a similar broadcast message the first time, asking for the *address/port pair(s) of the machine(s)* where a server for a certain service is running, caching the reply(ies), and then directing any further service requests to that machine (or to one of them, in case of several replies - there may be several servers offering the same service)

There's a qualitative difference between the two methods: the former is connectionless (neither the client nor the client's OS ever know about the server's machine: they just wait for a reply to a service request to come out of the blue); the latter explicitly uses a connection after the server has been located.

This scheme solves he transparency problem, scales well with the number of connected stations, and it's simple to implement (Apple's Appletalk is an example of it). However the broadcasts may put a heavy load on the communication bandwidth and on the station's performances, since they all *must* listen to them.

An alternative way is to use a *name server* to provide mapping between service names and machine/port pairs. Initially the clients need to know only the machine/port address of the name server (and they can either have it hardwired in a configuration file, or get it by the above broadcasting method. Whenever they need another service they would first ask the name server for the port/address pair(s) of the machine(s) where the server(s) for that service run, cache this information (possibly updating by a new query every once in a while, to take into account changes network/machine configuration), and use it for all subsequent queries. Evidently this method always use connections.

This third scheme is widely used (e.g. the Domain Name Server on the Internet). However, it's not free from defects: since it relies upon a centralized facility (the name server) it's prone to malfunctions if it fails; replicating the name server is obviously feasible, but introduces the problem f maintaining consistent information among servers in a dynamically changing environment (workstations are rebooted, printers are switched off and on, etc). Moreover, the size of the addressing tables in the name server may grow so huge that some kind of hyerachical organization becomes mandatory (e.g. the DNS again).

## ▪ Threading models

Clients go through the trouble of locating servers for the only purpose of using their services, hopefully having their queries answered as efficiently as possible. For heavily queried services (e.g. network file systems, centralized or distributed databada systems), server design becomes a relevant issue in determining the overall performance of the distributed system. A key consideration in this respect is finding the optimal way for the server to handle the network I/O, given the kind of service it offers and the expected load of client queries. No single design scheme is best for every service, and selecting the right one depends on the constraints and the goals for the service itself. Questions that must be answered prior to chose a certain design path are, among the others: how long does it take to process a typical client's request? how many requests are likely to arrive during that time? how long a client may be acceptably left waiting? how complex the server's design can acceptably become, without being too difficult to verify? how critical is the service for the overall distributed system performance? how robust must it be?
Experience has shown that a good design for a server starts with the choice of an appropriate *threading model*, i.e. of an organization of the server application in parallel

cooperating processes or threads (in the following the term thread will be used in both meanings).

Four commonly used threading models are, in order of increasing complexity:

Single thread, single client.

Single thread, multiple clients with selection.

One thread per client.

Worker threads.

## ▪ Single thread, single client

It's the simplest of all threading models. A server sits in a single infinite loop listening to a port, accepts a connection with a forthcoming client, and then services it immediately in the same thread. Further clients must wait until the request is completely serviced.
This model is easy to implement and has low resource requirement: it uses only one thread and two units of network connection services (i.e. two sockets in LINUX: the listening socket and the connected one). However it can handle only one client at a time, hence it's inappropriate for all but the most basic services

## ▪ Single thread, multiple clients with selection

The server still uses only one thread, but it can handle multiple clients simultaneously by time-multiplexing among them. Within the network listening loop it repeatedly polls the network connections (in LINUX the select(2) call would be used for this purpose) and when it determines that one is ready to send/receive data it checks whether it's a new connection request (from the listening port) or an already connected one: in the former case it accepts a new connection with a new forthcoming client; in the latter it sends or receive data as appropriate.
This scheme can create a powerful services, because it take advantages of the time delays in the clients and in the network: a client may need to maintain a connection open for a long time (because of the delays involved in closing it and opening a new one when needed), but use it only for short busts. In between the server can select and serve or create other connections. However the selection itself does not come for free, and it's acceptable when CPU use is not an issue, but presents a problem if the service requires high performance.

## ▪ One thread per client

It's probably the most popular threading model, because it's reasonably simple to implement, and is the fastest model when the maximum number of ``simultaneous'' (at the time scale of the service completion) requests is in the order of a few tens. The server

sits in a loop accepting forthcoming connections, and as soon as they arrive it spawns a thread that is responsible for handling the client for all the duration of the connection. Using a separate thread for each client has the advantage of reducing complexity, because each code path needs to perform only a single operation: the main thread accepts connection, and the child threads service them.

The downside is that this model doesn't scale well with the number of clients, for two reasons: because of the demands of each service thread on system resources, and because the of time the system spends context-switching (or, even worse, process-switching) among threads. With current technologies, this last term significantly impacts the server's performance when there are more then a few tens of threads running. With increasing number of connections, both terms reduce the rate at which each client can be serviced up to a point when accepting new connections becomes impractical.

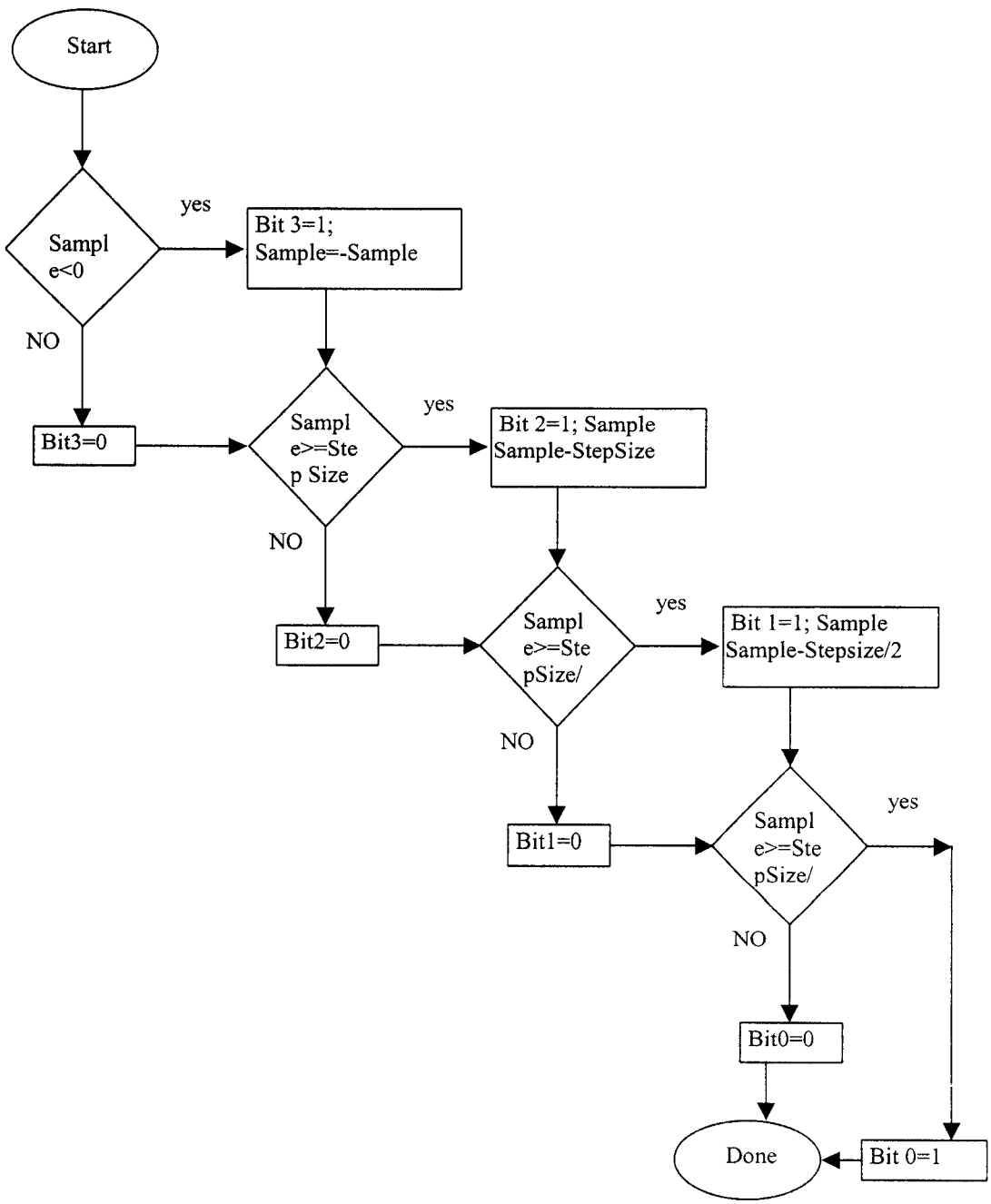
- Worker threads

This method improves on the scalability, with respect to the previous one, but increases complexity and hence slows the service rate when used for a small number of connections. A primary thread is used to select among ready connections, and dispatch tasks to a fixed number of worker threads. There are a number of ways in which to break down the work between the primary thread and a worker thread. The former can simply indicate to the latter that a certain connection must be serviced: the worker would wake up and do the job, or enqueue it for later processing if it's already busy. Or the main thread can give a first look at the connection request, and tell the worker thread what to do with it.

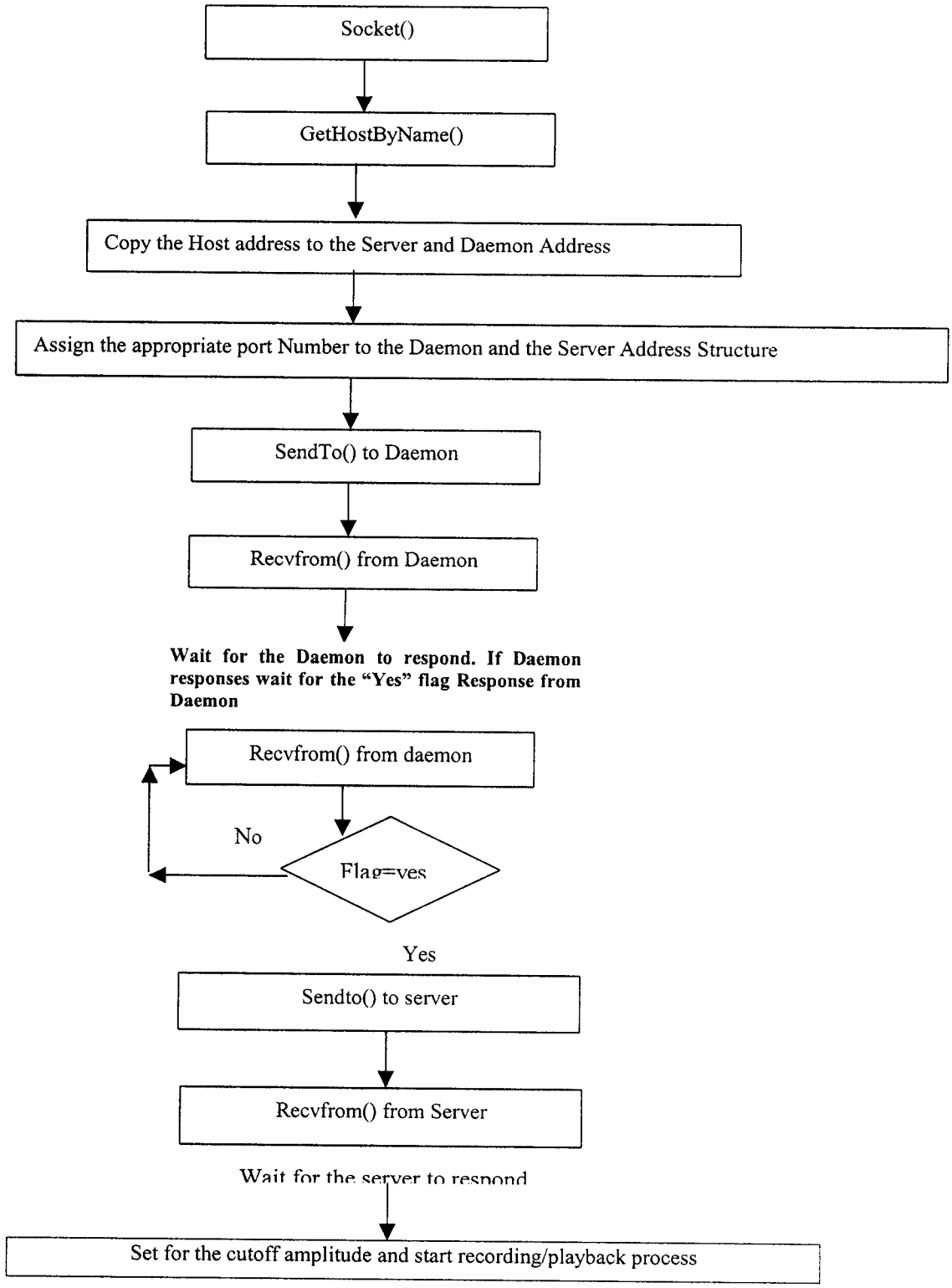
A problem with this scheme is the choice of the number of workers to use: if too many, the system will thrash, if too few, queues can get longer and longer. In general, a fine tuning of the application is needed on the filed. A commonly used rule of thumb, for multiprocessor machines, is to use at least as many threads as processors. More threads should be added if any of them spends a significant amount of time waiting for disk or network I/O to complete.

# 4.0 System Design

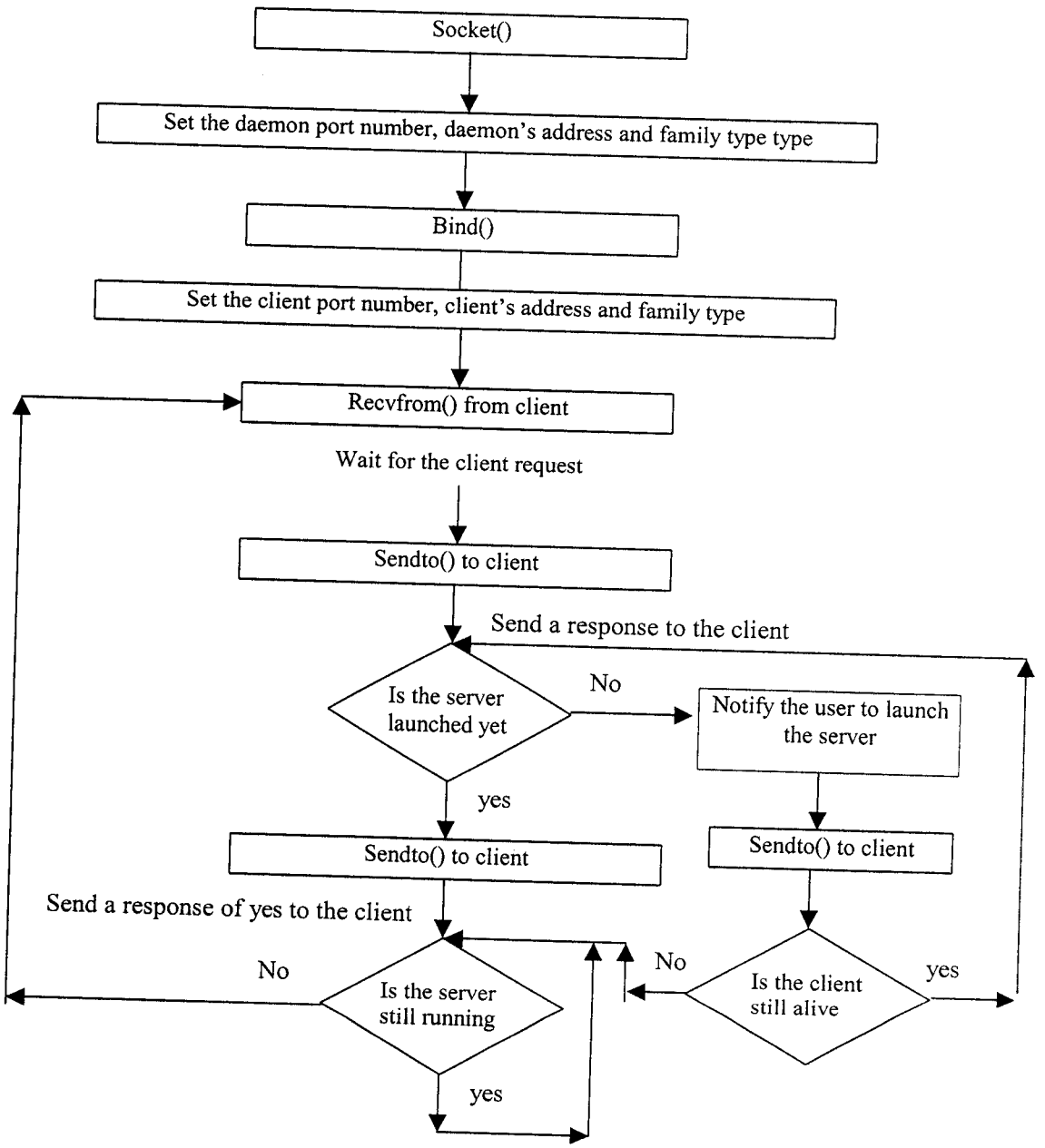
## 4.1 ADPCM Flow Chart



# Flow of Client Program



# Flow of Daemon Program



# 5.0 System implementation and Testing

## 5.1 Implementation

A total of three programs were written for the Linux version of the *Start Talk* . These are the *Start Talk* client, the *Start Talk* server and the *Start Talk* daemon. The client and the server work similarly except that the client makes the connection while the server waits to receive a connection request. The *Start Talk* daemon runs in the background waiting for a remote client connection request. If it receives the connection request, it will request the user to launch the server and at the same time, it will notify the remote client of the status of the server (whether it has been launched or not). Table summarizes the size of the *Start Talk* code and its executables.

| | *Start Talk* Daemon | *Start Talk* Server | *Start Talk* Client | *Start Talk* Header | *Start Talk* GUI | ADPCM Code | ADPCM Header |
|---|---|---|---|---|---|---|---|
| Code (bytes) | 3375 | 11202 | 12001 | 3556 | 12369 | 7304 | 411 |
| Executable (bytes) | 7256 | 13896 | 14285 | --------- | 1433269 | ---------- | ---------- |

Table : Summary of the code and executable size

The application begins execution with the detection of the sound card. Next, the network connectivity operation begins. If it is the server, it will wait for the connection from the remote host and if it is a client, it will attempts to make a connection to the remote host. Once the connection has been established, both the client and the server calculate the cut-off amplitude. The cut-off amplitude is the level of background noise in the room where *Start Talk* is running.

The program to determine whether the user is listening or he/she is talking uses the cut-off amplitude. If the user was listening then the program will start the playback process else it will start the recording process. Both users can begin the conversation once the cut-off amplitude has been calculated. To terminate the conversation, the user has to click on the Exit buttons from the Interface window or hit <RETURN> if he/she was running *Start Talk* outside X Windows.

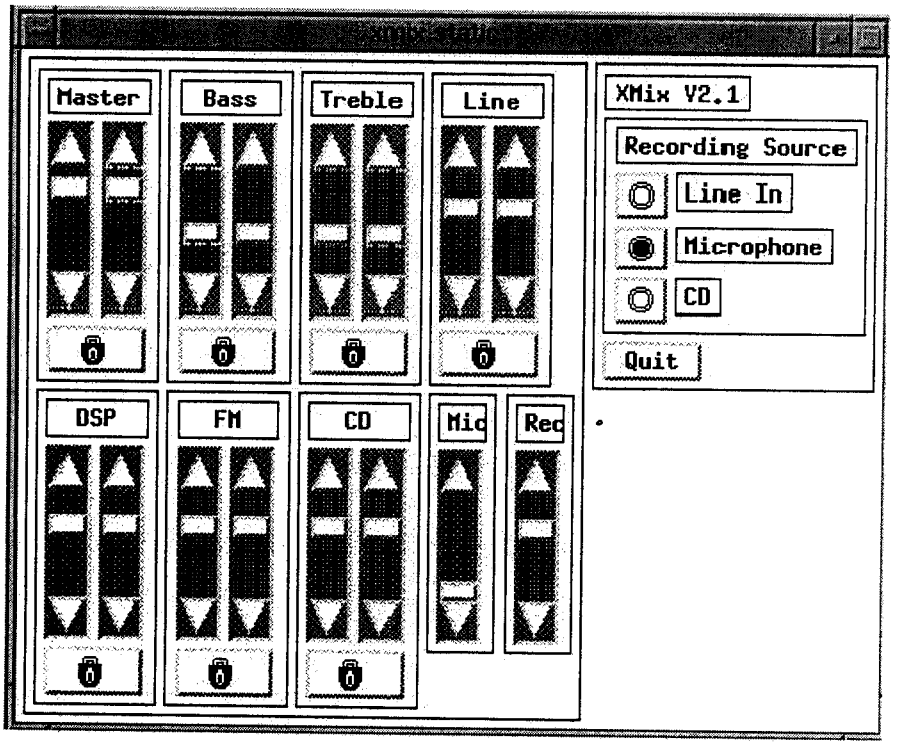
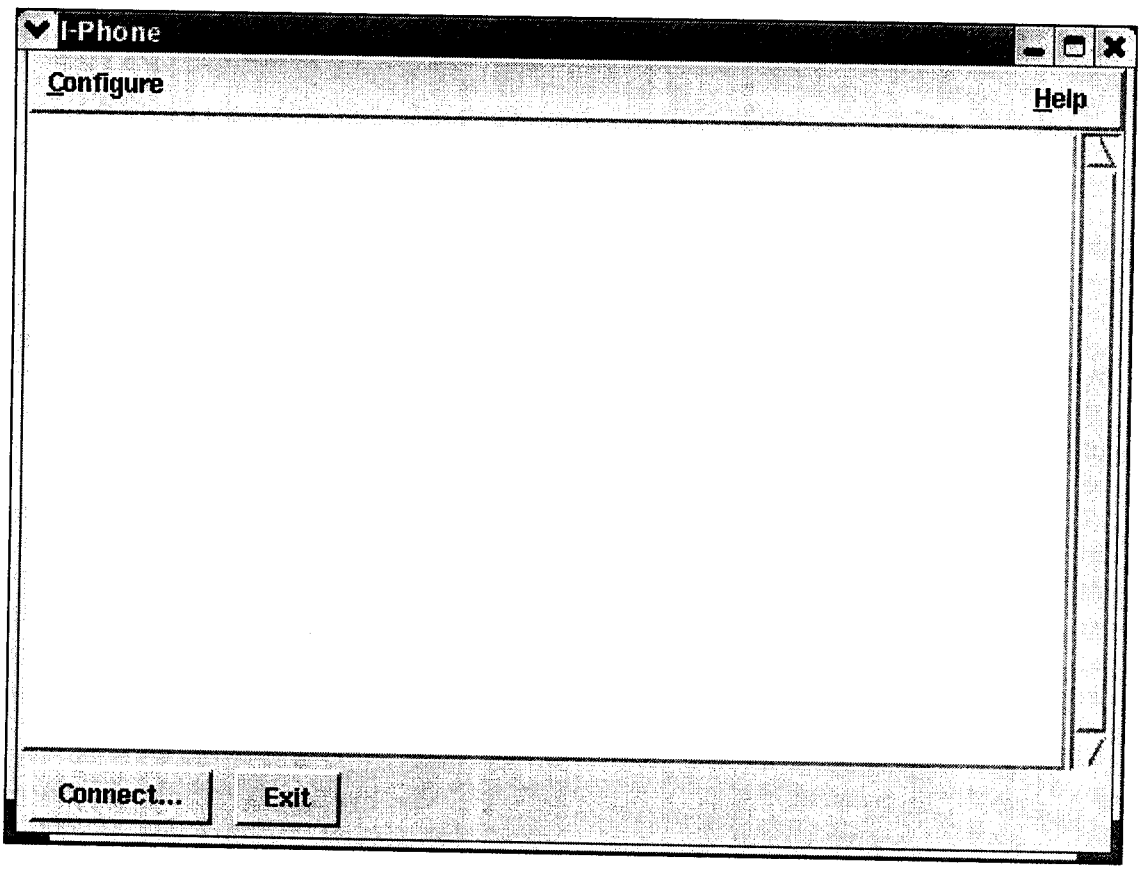



**Fig- The Xmix sound mixer for the Linux OS.**

For an optimum performance, the microphone level in the Xmix mixer should be set to 60% of the height, the recording level should be set to 100%, the DSP level should be set to 100%, and the master volume set to 80% of the height. These settings guarantee an intelligible real time voice communication between two Linux PCs. When tested between two rooms that are 30 meters apart, the speech quality was extremely bad at first because the cut-off amplitude was chopping off all the silence buffers that were less than 2000 samples in length. 2000 samples corresponds to 250 msec of speech at 8 Khz sampling frequency. Discarding speech that was less than 250 msec long resulted in choppy speech at the receiver. This problem was solved by increasing the number of consecutive silence buffers that are considered as part of the "talking" speech. The speech quality improved a lot once this problem was fixed.

The interface for the Linux version of *Start Talk* was developed for X Windows.

Fig- shows the main window of the Interface.

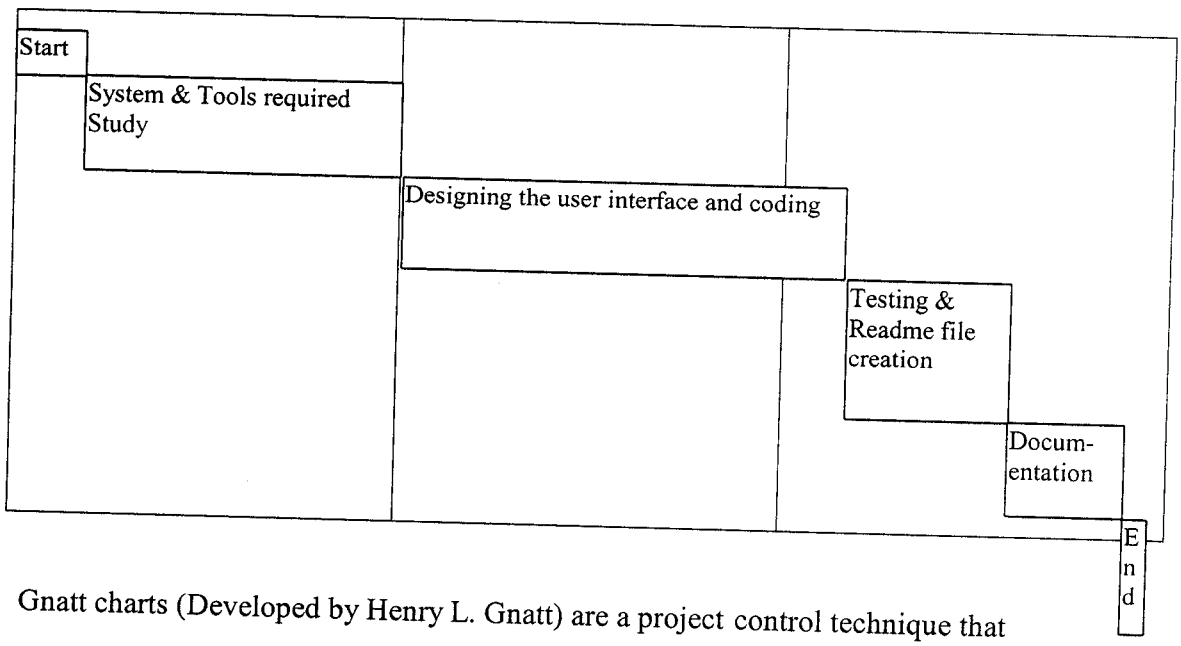



- Fig- The *Start Talk* Main Window

The interface is written to read standard outputs from the program every 250 milliseconds and display the output on the text widget. Among the features included in the interface is the menu bar with Configure and Help buttons. The Configure button lists the Xmix sound mixer, which can be used to set the volume, the microphone level, etc. The Help buttons list the *Start Talk* Frequently asked question (FAQ), the Linux Sound HOWTO document and the About *Start Talk* . The *Start Talk* FAQ document was written to assist the user with the installation and the operation of this application. The Linux Sound HOWTO document is added to the *Start Talk* package to provide the user with the help on sound card related problems. The about *Start Talk* window displays the version number and the *Start Talk* programmer's address.

*Start Talk* for the Linux OS can run both under and outside X Windows. When the user clicks on the OK button within the GUI connect window, the TCL script "kills" the *Start Talk* server and launches the *Start Talk* client by supplying the host name as the command line argument. The client program can also be launched outside X if the user supplies the host name at the command line.

- Gnatt Chart




Gnatt charts (Developed by Henry L. Gnatt) are a project control technique that can be used for several purposes, including scheduling, budgeting and resource planning.

A Gnatt chart is bar chart, with each bar representing an activity. The bars are drawn against a time line. The length of each bar is proportional to the length of time planned for activity.

### 5.1.1Validation Checks

In network communication it is always required to check the validity of each function. To validate the functions we have to start checking the return value of each function and if any function is not giving the desired value it should be reported to the user. Here first validity check we did at the time of launching of the package. Since this package starts the server program by itself so first we are checking that server is started or not.

In server program first operation what we are doing is socket creation there also validation check is done. If socket function is returning –1 value then in that case we informing the user that socket can't be created and closing the application on user interface. After that if any user is calling us we are replying him by text to start the conversion. Here we are checking the number of transferred bytes by the length of message if both are same it means message is transferred otherwise sending error will be reported to the user. Next we are going for recording and playing process for which we checking that sound card is working fine or not. For this purpose /dev/dsp file will be opened in read mode and if it is returning any positive number means that we can use sound card for our purpose.

Now sound card will be configured on required value by the program by calling the ioctl function if this function is returning any positive value then ser will progress otherwise it will stop there only by giving the message to the user.

## 5.2 Testing

Software testing is a critical element of software quality assurance and represents the ultimate review of specification, design and coding. The main objective of testing is
Testing is a process of executing a program with the intent of finding an error.
A good test case is one that has high probability of finding an as-yet undiscovered error.
A successful test is one that uncovers an as-yet undiscovered error.
This software is tested with loop testing, which is a white-box testing technique that focuses exclusively on the validity of loops constructs. In this software mainly we are using simple loops, nested loops and unstructured loops. Here we tried to skip the loops entirely and test that application is running or not for a single iteration. This test was successful. Next we tried to removed the nested loops to test the condition within the nested loops are running without this and returning the appropriate value, means that this test is also successful. The last but least we did testing for unstructured loops i.e. all the infinite loops is tested in this module, which was successful. Because without this loop our client and server both went off after finishing just one procedural execution. So as whole testing of this software was done in successful environment and suggested that before playing back the sound it should be stored temporally in queue. In this way we will not loose the sound and it can be played back in proper sequence.

Apart from this, we did the testing of the four modules as a modular testing.

### 5.2.1 Daemon Module

The demon will try to find the server's status. According to this status, the daemon will send The module is used to create a process named as
Daemon, which runs in background for continuously check the server and client for uninterrupted communication.

Here we are trying to accept the connection of client on
daemon process then checking the availability of the server. When client try to connect a particular server. The client will be connected to daemon first. Daemon will extract the client address and reply a message to the client that client is connected by daemon. After that message to client and server.

## Compile and run this daemon module,

cc –o iphone_daemon iphone_daemon.c

./iphone_daemon &

## Output of the daemon module,

### i. When the server is not running:

Client wants to connect the server. It will connect to daemon firstly and daemon send message "You have connected to daemon". Daemon will also extract the client address.

After that, daemon will check the server, if it is not running. The Daemon will send "NO" message to client.

### ii. When the server is running.

Firstly, client will connect to daemon and receive message "You have connected to Daemon". Daemon will extract the client address and check the server. If server is running then daemon will send message to server "The host ..... Is trying to contact you by iphone". And one message to client "Yes", means server is running.

## 5.2.2 Recording And Playing:

These two functions are the part of the client and server also.

By the recording function, user records the voice. And by the playing function, user plays the recorded voice. In Linux, two-device file for handle the recording and playing-

i. /dev/audio

ii. /dev/dsp

In /dev/audio, we have to handle the char* means 8 bits. But, in this software for recording and playing the sample size is 16 bits. We can handle the integer in /dev/dsp, so we are using the/dev/dsp.

## 5.2.3Squelch Values

This squelch_buffer function is the part of the client and severs

also. Here we are checking the adaptive squelch value and comparing the previous squelch value with the new squelch value. On our machine, the squelch testing values are given below:

i. 8409764.0000
ii. 8409598.0000
iii. 8451109.0000
iv. 8422544.0000
v. 8431555.0000
vi. 8415932.0000
vii. 8420778.0000
viii. 8447196.0000
ix. 8416817.0000
x. 8410426.0000

## 5.2.4Compression Module:

The module is used to create a process named as ADPCM
(Adaptive Pulse Code Modulation) compression and decompression. Which is compressing the 16 bits to 4 bits and again decompressing 4 bits to 16 bits. Both client and server are using this compression module. Two part of this compression module are-
adpcm_coder ()
adpcm_decoder ()

The argument of both function are same. These are-

i. short indata [],
ii. char outdata [],
iii. int len,
iv struct adpcm_state *state

# 6.0 Conclusion

A total of three programs will be written for the Linux version of the I-Phone. These will be the I-Phone client, the I-Phone server and the I-Phone daemon. The client and the server will work similarly except that the client makes the connection while the server waits to receive a connection request. The I-Phone daemon will run in the background and will be waiting for a remote client connection request. If it will receive the connection request, it will request the user to launch the server and at the same time, it will notify the remote client of the status of the server (whether it has been launched or not).

# 7.0 Scope for future development

A complete rewrite to include a better Voice Activation Detector, a better voice codec with high compression ratio, a multi-threaded application in which recording, playback, and network transmission is done on separate threads.
Support for H.323. This will allow I-Phone to talk to Windows based voice applications, which support H.323 such as NetMeeting and Internet Phone.
Use of H.323 based codec's such as G711, G723, and/or G729a, etc. Using these vocoders will ensure interoperatibility with H.323 compliant software and also provide us with better voice compression than what can be achieved with the IMA ADPCM codec.
Better intuitive graphical interface using Java, GTK, QT, or anything else.
And anything else, which one can think off...

## 8.0 Bibliography


Author Name: "W. Richard Stevens"
Title of The Book: "Unix NetworkProgramming"
Publisher Name: "BPB Publications"
Year of Publication:" 1990"

Author Name:" John K. Ousterhout"
Title of The Book:" Developing Linux Applications"
Publisher Name:"Tata Mc GrawHill"
Year of Publication:"1992"

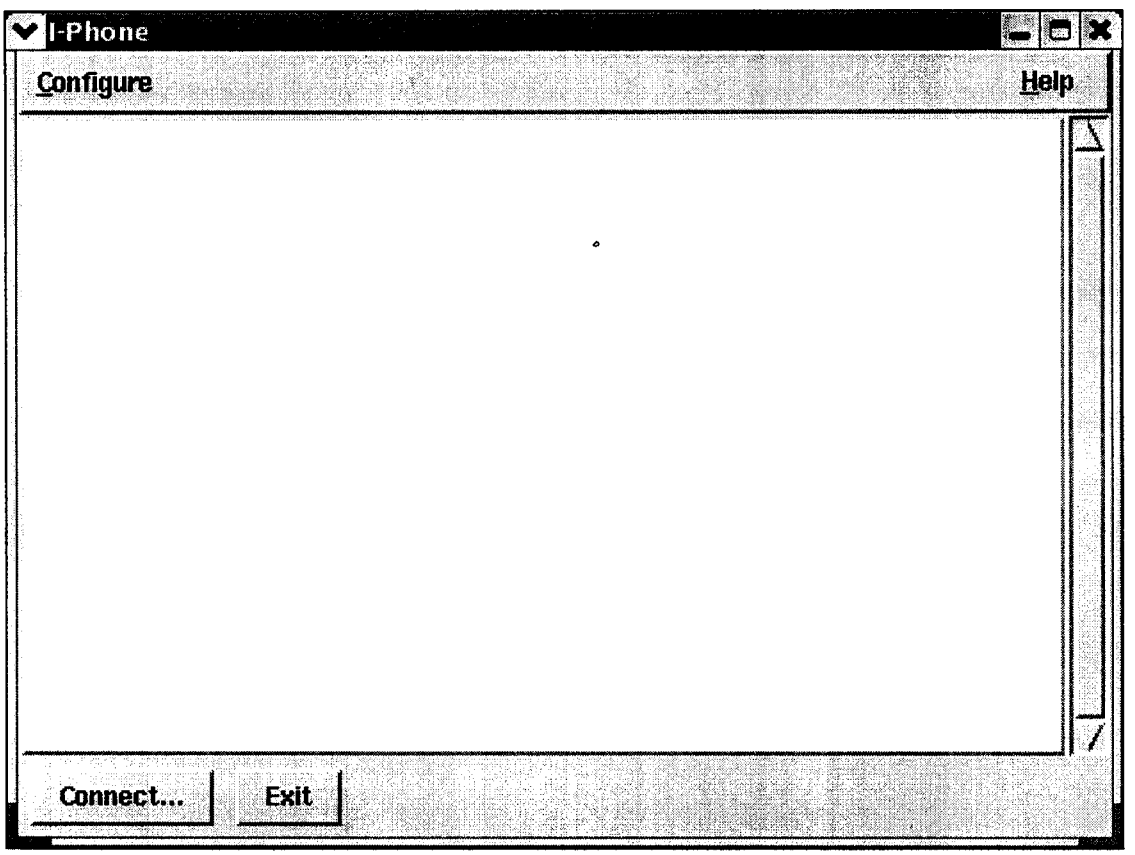
Author Name:" Eric Harlow"
Title of The Book:" Tcl and Tk Toolkit"
Publisher Name:"s.s Chand"
Year of Publication:"1988"

Author Name:" Carlos Ghezzi"
Title of The Book:" Fundamentals of software Engineering"
Publisher Name:"Tata Mc Graw Hill"
Year of Publication:"1996"

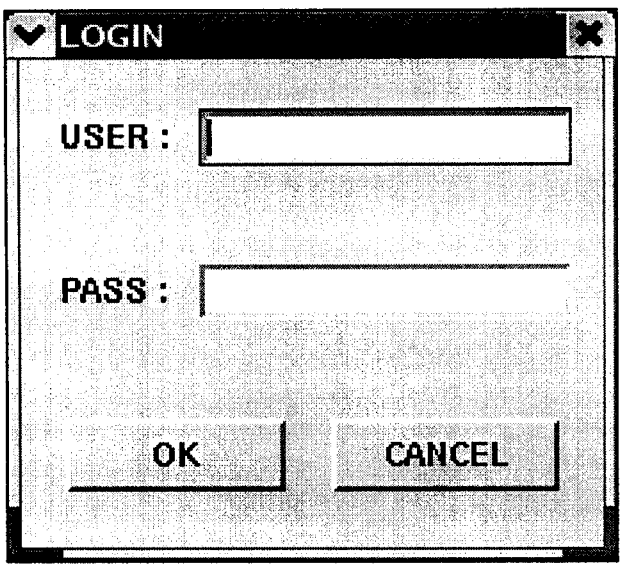
# 9.0 Appendix

## 9.1 Sample screens

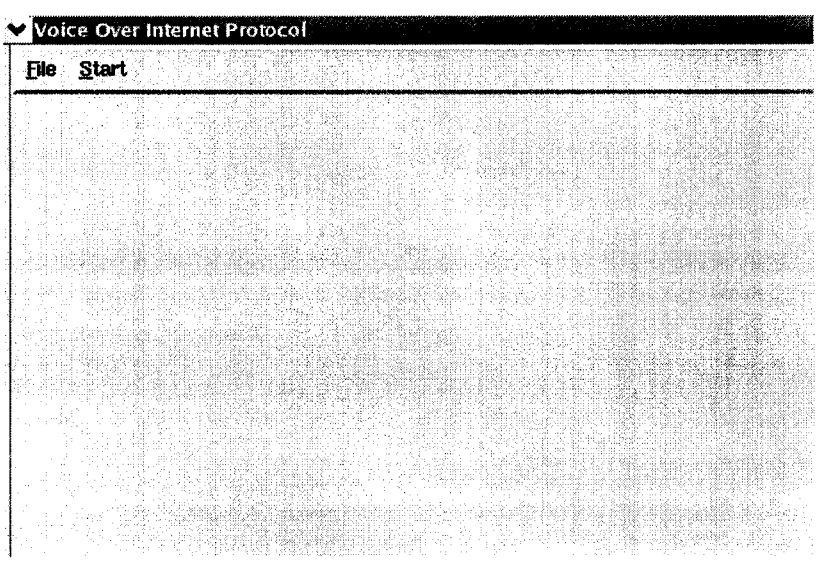
Fig- The *Start Talk* Main Window



## Login Window




## Main Window



## 9.2 Sample code.

```
            void adpcm_coder(indata, outdata, len, state)
    short indata[];
    char outdata[];
    int len;
    struct adpcm_state *state;
{
    short *inp;             /* Input buffer pointer */
    signed char *outp;      /* output buffer pointer */
    int val;                /* Current input sample value */
    int sign;               /* Current adpcm sign bit */
    int delta;              /* Current adpcm output value */
    int diff;               /* Difference between val and valprev */
    int step;               /* Stepsize */
    int valpred;        /* Predicted output value */
    int vpdiff;             /* Current change to valpred */
    int index;              /* Current step change index */
    int outputbuffer;       /* place to keep previous 4-bit value */
    int bufferstep;         /* toggle between outputbuffer/output */
    outp = (signed char *)outdata;
    inp = indata;
    valpred = state->valprev;
    index = state->index;
    step = stepsizeTable[index];
    bufferstep = 1;
    for ( ; len > 0 ; len-- ) {
      val = *inp++;

      /* Step 1 - compute difference with previous value */
      diff = val - valpred;
      sign = (diff < 0) ? 8 : 0;
      if ( sign ) diff = (-diff);

      /* Step 2 - Divide and clamp */
      delta = 0;
      vpdiff = (step >> 3);
      if ( diff >= step ) {
          delta = 4;
          diff -= step;
          vpdiff += step;
      }
      step >>= 1;
      if ( diff >= step  ) {
          delta |= 2;
          diff -= step;
          vpdiff += step;
      }
      step >>= 1;
      if ( diff >= step ) {
          delta |= 1;
          vpdiff += step;}
```

```
    /* Step 3 - Update previous value */
    if ( sign )
      valpred -= vpdiff;
    else
      valpred += vpdiff;

    /* Step 4 - Clamp previous value to 16 bits */
    if ( valpred > 32767 )
      valpred = 32767;
    else if ( valpred < -32768 )
      valpred = -32768;


      /* Step 5 - Assemble value, update index and step values */
    delta |= sign;
    index += indexTable[delta];
    if ( index < 0 ) index = 0;
    if ( index > 88 ) index = 88;
    step = stepsizeTable[index];

    /* Step 6 - Output value */
    if ( bufferstep ) {
        outputbuffer = (delta << 4) & 0xf0;
    } else {
        *outp++ = (delta & 0x0f) | outputbuffer;
    }
    bufferstep = !bufferstep;
  }

  /* Output last step, if needed */
  if ( !bufferstep )
    *outp++ = outputbuffer;
  state->valprev = valpred;
  state->index = index;
}
```