Design and Implementation of VOIP Based PABX Exchange through Existing Network

A Thesis submitted

by

D. M. S. Sultan (ID: 02-03169-2)
Chowdhury Tawhudul Islam (ID: 02-02905-1)
Dawen Md.Sohal (ID: 01-02652-3)
Yusuf Zahida Binta (ID: 02-03093-2)

under the supervision of

Mr. Tareq Bin Ali
Lecturer,
Faculty of Engineering
AIUB

Computer Engineering Department
Faculty of Engineering
American International University- Bangladesh

Fall Semester 2005-2006

December 2005

Design and Implementation of VOIP Based PABX Exchange through Existing Network

A Thesis submitted to the Computer Engineering Department of the Engineering Faculty, American International University - Bangladesh (AIUB) in partial fulfillment of the requirements for the degree of Bachelor of Science in Computer Engineering.

D. M. S. Sultan	(ID: 02-03169-2)
Chowdhury Md. Tawhudul Islam	(ID: 02-02905-1)
Dawen Md.Sohal	(ID: 01-02652-3)
Yusuf Zahida Binta	(ID: 02-03093-2)

Computer Engineering Department
Faculty of Engineering
American International University- Bangladesh

Fall Semester 2005-2006

December 2005

Declaration

This is to certify that this project is our original work. No part of this work has been submitted elsewhere partially or fully for the award of any other degree or diploma. Any material reproduced in this project has been properly acknowledged.

D. M. S. Sultan
ID: 02-03169-2
Dept.: COE

Chowdhury Md. Tawhidul Islam
ID: 02-02905-1
Dept.: COE

Dawen Md. Sohal
ID: 01-02652-3
Dept.: COE

Yusuf Zahida Binta
ID: 02-03093-2
Dept.: COE

Approval

The Design Project titled "Design and Implementation of VOIP Based PABX Exchange through Existing Network" has been submitted to the following respected members of the Board of Examiners of the Faculty of Engineering in partial fulfillment of the requirements for the degree of Bachelor of Science in Computer Engineering on 28 December 2005 by the following students and has been accepted as satisfactory.

1. D. M. S. Sultan (ID: 02-03169-2)

2. Chowdhury Md. Tawhidul Islam (ID: 02-02905-1)

3. Dawen Md. Sohal (ID: 01-02652-3)

4. Yusuf Zahida Binta (ID: 02-03093-2)

_____

Mr. Tareq Bin Ali
Lecturer
Faculty of Engineering
AIUB
Supervisor

_____

Mr. Hasan Tareq Imam
Lecturer
Faculty of Engineering
AIUB
External

_____

Dr. Carmen Z. Lamagna
Vice Chancellor
American International University - Bangladesh

Acknowledgements

This project has been done under the supervision of Mr. Tareq Bin Ali. We would like to express our heartiest gratefulness to him and also we would like to thank him for supervising us in an enormous way. He always guide us how to carry on a project work and also provide us information and necessary assistance during this project.

We would also like to thank our honorable Vice Chancellor Dr. Carmen Z. Lamagna for continuous encouragement.

We would also like to show our heartiest gratitude to other faculty members who always support us to do the best. Our special thanks to our university IT Department who has given us the opportunity of using Internet free. So it provided us valuable information and materials, needed for preparing this report.

At first, we express our heartiest gratefulness to the almighty Allah. It would not be possible for us to complete this project successfully without his divine blessing.

Contents

Abstract

With the advancement of telecommunication technology Voice Over IP is nothing but the demand of technical world. Using this VOIP technology, people are talking to each other through internet as well intranet (Local Network). VOIP requires about a null cost for such communication. Besides this, PABX or Private Automatic Branch Exchange is another telecommunication technology that is known as private switching system. Through PABX exchange device people can call each other in intranet and can call end user of PSTN network. The communication between these two individuals is done by circuit switching in most of the exchange. This PABX system is widely used today in different companies but the matter of regret that a PABX exchange PANASONIC KX-TD232 of 32 subscribers costs about 2 laks. This is so costly.

For switching, packet switching technology was used and no additional hardware was required. A Soft Router with C++ (MFC API) was built that will act as a soft PABX exchange. This can be implemented in any gateway server or any Always On PC with IPS (Instant Power System) within a computer network interface. To communicate with PSTN network only MODEM will be needed.

Though in developed countries a lot of researches were conduct with VOIP but in respect of Bangladesh, such project on PABX industry is totally new. Hopefully this will bring a revolution in telecommunication industry.

Chapter – 1
Introduction

1.1 Introduction

Private Automatic Branch Exchange (PABX) is the integral part of today's technical world to provide a dynamic telephony solution among organizations or institutions. Several researches have been conducted in the different fields of technology but no remarkable research has been conducted in the field of PABX especially in Bangladesh to simplfy its design or to reduce the cost of implementation and maintenance. This Voice Over Inter Protocol (VOIP) based PABX exchange through existing Intranet (LAN, WAN) considers those facts in very effective manner. In addition, VOIP is such an invention in the communication world that makes the cost of voice communication about to null. The design and implementation of PABX with VOIP technology would bring revolution in currently implemented whole PABX system.

1.2 Historical Background

In developed countries like Europe and Japan, several researches have been done and are being conducted regarding VOIP. Some of the researches have already evaluated some real application regarding voice communication. For instance, IP Private Branch Exchange (IP-PBX) and Computer Branch Exchange (CBX) devices are such technical devices that support VOIP based communication through existing network. This project has implemented VOIP in soft PABX exchange that has reduced the dependency on CBX type hardware as well as high cost of it.

1.3 Objective of the work

The objective of this project was to implement a PABX system with VOIP technology to minimize the cost of current PABX system in large extent. It is widely seen that most of the companies as well organizations or institutions have already implemented or implementing LAN (Local Area Network) or WAN (Wide Area Network) with the need of advanced communication world. Therefore, VOIP that requires a minimum null cost for private switching system (PABX) can easily be installed on such open networks using packet switching.

1.4 Introduction to this thesis

The work has been divided in couple of chapters. Each chapter consists of detailed information according to the topics.

Chapter 2 introduces VOIP and its detail information. Besides this, VOIP protocol IP header format as well as its benefits is also described there.

Chapter 3 introduces the PABX system. Here, the existing services of it are vividly described. PABX exchanges are classified into several categories on the basis of providing services. Among those, the services of PABX 1 and PABX5 have been described.

Chapter 4 describes a detailed description of design and implementation of this VOIP based soft PABX exchange.

Chapter 5 describes the analytical cost effectiveness between existing PABX exchanges and the project work. The cost minimization of PABX system which was one of the objectives of this project has been shown through this chapter.

Chapter 6 discusses the problems as well as solutions that was made during project work. Future Plan of further development has been discussed here also.

Chapter – 2
Introduction to VOIP

2.1 Introduction

Civilization started to make human life much easier. On the way of civilization, different types of technologies are invented. The current world is the world of technology. All the technologies of the world have made the human life easy and comfortable in which communication system plays the most important role.

Recently communication system has been developed widely. But there is no end of development. Again, people's needs have no boundary. Before 30 years, people cannot even think of mobile. But now most of us cannot think of a day without mobile phone. Again, in communication field, internet is also a major issue. Moreover, people needs have no boundary. So, now the demand for telephone system in internet is required. To support this, Net2Phone is established. But in consideration of costing effects a new technology called "VOIP" is started.

Internet Protocol technology treats services like voice as an application. With VoIP, telephony signals are digitized and transmitted as packets to their destination, just as with an email, streaming video or any other kind of IP transaction. In short, IP destroys the old distinctions between "voice" and "data" that are a standard part of PSTN thinking. Indeed, because the information associated with any particular application is broken down into bits and does not take its analog form (i.e. sound, text or pictures) until it is reassembled at the terminating end, it is impossible to assign it to any particular at any point other than origin or destination. An IP network provider, for example, can be carrying real- time two-way voice bits without itself offering voice service to any end- user customer.

Today with headphones that allow real-time voice communication by players in different locations, through broadband Internet connections to the home.

Each of these embeds IP voice into the offering and therefore may be labeled Voice over IP. But none resembles traditional phone service – or fits into the old classifications of the circuit-switched world.

2.2 History of VOIP

Voice over Internet Protocol, VoIP or Broadband phone service as it is often referred to, is changing the telephony world. Traditional phone lines are slowly being phased out as businesses and households around the world embrace the benefits and features that VoIP technology has to offer. As this evolution accelerates, it is worthwhile to stop and take a look at the history of VoIP. People will find that as interesting as the history of VoIP may be, the future of VoIP is even more intriguing and exciting.

The concept of VoIP (Voice over Internet Protocol) originated in about 1995, when hobbyists began to recognize the potential of sending voice data packets over the Internet rather than communicating through standard telephone service. This concept allowed PC users to avoid long distance charges, and it was in 1995 that the first Internet Phone Software appeared. While contemporary VoIP uses a standard telephone hooked up to an Internet connection, early efforts in the history of VoIP required both callers to have a computer equipped with the same software, as well as a sound card and microphone. The software was used to compress the voice signal, convert it into voice packets, and then finally to ship it out over the Internet. This particular technology worked as long as both the caller and the receiver had the same tools and software. However, the sound quality was not even close to that of the standard equipment in use at that point of time. This attempt can be termed as the first IP phone that came into existence. These early applications of VoIP were marked by poor sound quality and connectivity, but it was a sign that VoIP technology was useful and promising.

Vocaltec released the first internet phone software. This software was designed to run on a home PC and much like the PC phones used today, it utilized sound cards, microphones and speakers. The software was called "Internet Phone" and used the H.323 protocol instead of the SIP protocol that is more prevalent today. Vocaltec had initial success with Internet Phone, and had a successful IPO in 1996. It was the Skype of the mid 90s

VoIP evolved gradually over the next few years. At that stage, only PC-to-PC communication was available. The potential of the technology was soon seen, and by 1998 VoIP gradually reaching the point where some small companies were able to offer PC to phone service in about 1998. VoIP had reached “respectability”. A number of entrepreneurs set up gateways to allow first PC-to-Phone and later Phone-to-Phone connections. Some of these operated a “free to the customer” marketing model, typically paid for by advertising. These (SIP based) services often required the services of a PC to originate the call, although the actual communication was from ‘phone to ‘phone. At this stage, VoIP traffic represented rather less than 1% of voice traffic. Three IP switch manufacturers introduced equipment capable of switching VoIP traffic in 1998. Three IP switch manufacturers launched equipment, during the year 1998, which was capable of being used for switching. Currently, the majority of IP switching and routing equipment suppliers offer VoIP on their mid-range and up equipment, either as standard equipment or as an option.

Networking manufacturers such as Cisco and Lucent introduced equipment that could route and switch the VoIP traffic and as a result by the year 2000, VoIP traffic exceeded 3% of voice traffic by 2000, the breakthrough in VoIP history came when hardware manufacturers such as Cisco Systems and Nortel started producing VoIP equipment that was capable of switching. What that meant was that functions that previously had to be handled by a computer's CPU, such as "switching" a voice data packet into something that could be read by the PSTN (and vice versa) could now be done by another device, thus making VoIP hardware less computer dependent. Once hardware started becoming more affordable, larger companies were able to implement VoIP on their internal IP networks, and long distance providers even began routing some of the calls on their networks over the Internet. Since 2000, VoIP usage has expanded dramatically. There are several different technical standards for VoIP data packet transfer and switching and each is supported by at least one major manufacturer - no clear "winner" has yet emerged to adopt the role of a universal standard.

While companies often switch to VoIP to save on both long distance and infrastructure costs, VoIP service has also been extended to residential users. In just a few short years, VoIP has gone from being a fringe development to a mainstream alternative to standard telephone service VoIP as either a standard or as an option on their mid-range and up equipment is forecast to grow rapidly to between 25% and 40% of all international voice traffic by 2004.

Now, in 2005, major voice quality issues have long since been addressed and VoIP traffic can be prioritized over data traffic to ensure reliable, clear sounding, unbroken telephone calls. Revenue from VoIP equipment sales alone are projected to reach around $3 billion this year and are being forecast to be over $8.5 billion by the end of 2008. This is primarily being driven by low cost unlimited calling plans and the abundance of enhanced and useful telephony features associated with VoIP technology.

This is a phenomenal growth rate and with the rapid introduction of Video over IP

2.3 VOIP Users

i. Telecommunications companies.
ii. Enterprises with multiple site offices.
iii. Home workers.
iv. Individuals through software–
    a) Net2Phone
    b) Microsoft's NetMeeting

2.4 Frequency calculation for VOIP

An uncompressed voice conversation across a traditional analog telephone network requires only 64 Kbps of bandwidth, and about 60 percent of that bandwidth is wasted on silence. On IP networks, digital technologies recapture the silent time and compress the actual sound down to as little as 5 Kbps, not including the IP packet overhead. Figuring on 14 Kbps to 24Kbps per voice session is considered adequate when planning VoIP networks.

Bandwidth

The amount of bandwidth required to carry voice over an IP network is dependent upon a number of factors. Among the most important are: Codec (*co*der/*dec*oder) and sample period IP header Transmission medium Silence suppression. The codec determines the actual amount of bandwidth that the voice data will occupy. It also determines the rate at which the voice is sampled. The IP/UDP/RTP header can generally be thought of as a fixed overhead of 40 octets per packet, though on point-to-point links RTP header compression can reduce this to 2 to 4 octets (RFC 2508). The transmission medium, such as Ethernet, will add its own headers, checksums and spacers to the packet. Finally, some codecs employ silence suppression, which can reduce the required bandwidth by as much as 50 percent. The Codec The conversion of the analogue waveform to a digital form is carried out by a codec. The codec samples the waveform at regular intervals and generates a value for each sample. These samples are typically taken 8,000 times a second. These individual values are accumulated for a fixed period to create a frame of data. A sample period of 20 ms is common. Some codecs use longer sample periods, such as 30 ms employed by G.723.1.

Others use shorter periods, such as 10 ms employed by G.729a.The important characteristics of the codec are: The number of bits produced per second. The sample period - this defines how often the samples are transmitted.

Together, these give us the size of the frame. For example, take a G.711 codec sampling at 20 ms. This generates 50 frames of data per second. G.711 transmits 64,000 bits per second so each frame will contain 64,000¸ 50 = 1,280 bits or 160 octets.

2.5 Frames and Packets

Many IP phones simply place one frame of data in each packet. However, some place more than one frame in each packet. For example, the G.729a codec works with a 10 ms sample period and produces a very small frame (10 bytes). It is more efficient to place two frames in each packet. This decreases the packet transmission overhead without increasing the latency excessively.

2.5.1 Voice Packet Moduling

The Voice Packet Module contains a number of functions. The Pulse Code Modulation (PCM) Interface interacts with the voice samples and includes a tone generator to generate the Dual Tone Multifrequency (DTMF) tones as necessary. The Echo Canceller is compliant with the ITU-T G.165/G.168 echo cancellation standards, which improve the clarity of the received signal. The Voice Activity Detector monitors the received signal for voice activity. If no activity is detected, the silence is suppressed, yielding additional savings in transport bandwidth. The Tone Detector receives the DTMF tones and reports them to the host system. The Voice Codes software includes algorithms for many of the codecs which compress the voice signal. A Fax Interface Unit is also included; it allows facsimile information to be transmitted and received. The Adaptive Playout Unit provides timing information in both transmit and receive directions, thus controlling packet jitter and packet loss. The Packet Protocol function encapsulates the compressed voice or fax information into a packet for transmission over the network. Finally, the Message Processing Unit controls the exchange of monitoring and control information between this software module and the host equipment that it resides in.

2.5.2 Voice Packet Moduling Functions

The Voice Packet Module contains a number of functions

a) The Pulse Code Modulation (PCM)

The Pulse Code Modulation (PCM) Interface interacts with the voice samples and includes a tone generator to generate the Dual Tone Multifrequency (DTMF) tones as necessary.

b) Echo Canceller

The Echo Canceller is compliant with the ITU-T G.165/G.168 echo cancellation standards, which improve the clarity of the received signal.

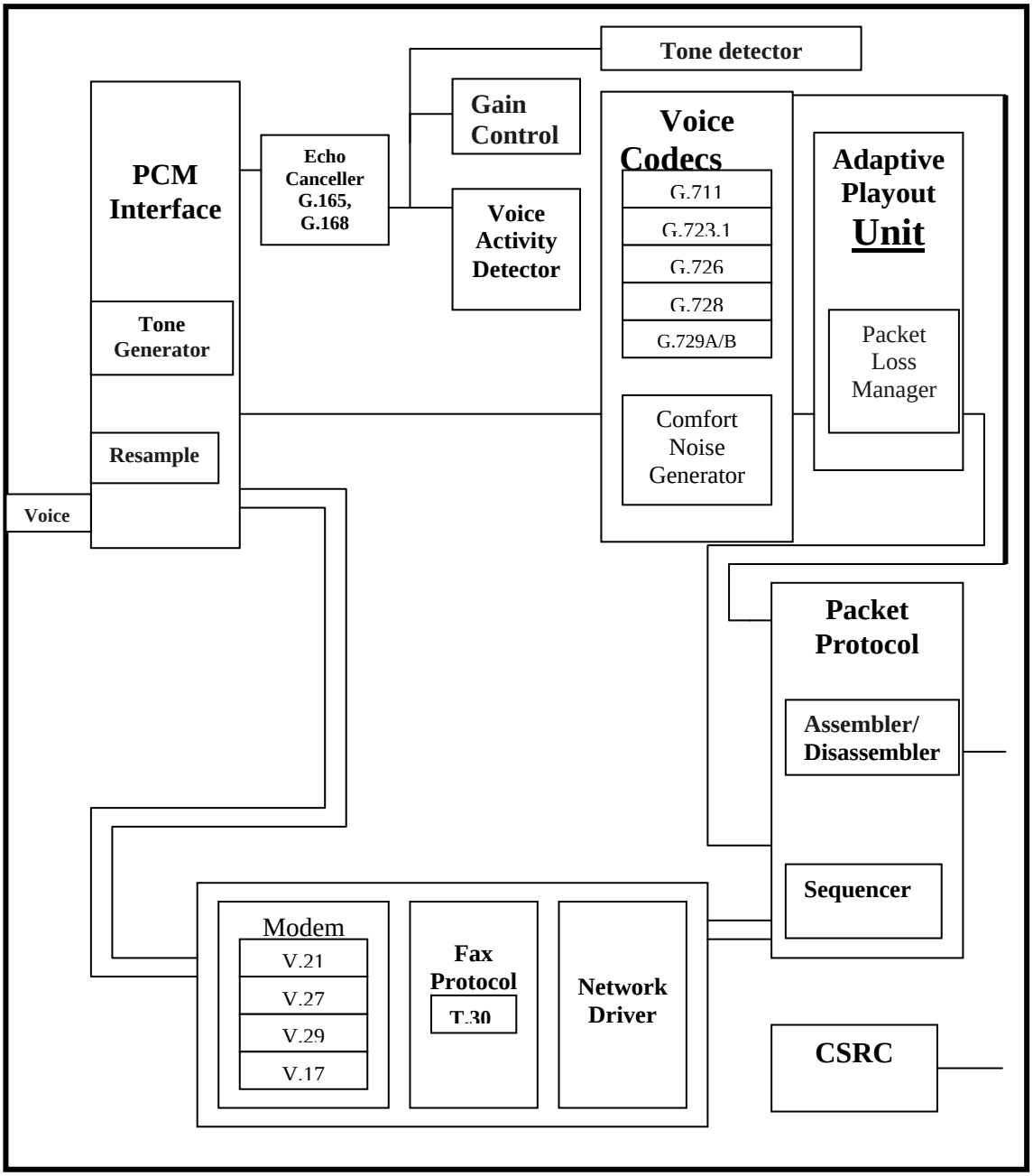

Figure 2.1 Diagram of Voice Packet Module

c) Voice Activity Detector

The Voice Activity Detector monitors the received signal for voice activity. If no activity is detected, the silence is suppressed, yielding additional savings in transport bandwidth.

d) Tone Detector

The Tone Detector receives the DTMF tones and reports them to the host system.

e) Voice Codes software

The Voice Codes software includes algorithms for many of the codecs which compress the voice signal.

f) Fax Interface Unit

A Fax Interface Unit is also included; it allows facsimile information to be transmitted and received.

g) Adaptive Playout Unit

The Adaptive Playout Unit provides timing information in both transmit and receive directions, thus controlling packet jitter and packet loss.

h) Message Processing

Finally, the Message Processing Unit controls the exchange of monitoring and control information between this software module and the host equipment that it resides in.

2.6 Layered model

In common with many communications systems, the protocols involved in Voice over IP (VOIP) follow a layered hierarchy which can be compared with the theoretical model developed by the International Standards Organization (*OSI seven layer model*). Breaking a system into defined layers can make that system more manageable and flexible. Each layer has its job, and does not need a detailed understanding of the layers around it.

For example, IP datagram can be transported across a variety of link layer systems including serial lines (using PPP), Ethernet and Token Ring. The link layer protocol is for the most part irrelevant to IP (unless that protocol limits the size of its datagram), and need not be the same for the first link of a Voice over IP call and the final link of a VOIP call.

As always there are exceptions (such as IP over ATM), but the simple discreet layered model will be considered in this document.

The effect of each layer's contribution the communication process is an additional header preceding the information being transmitted. The complete packet which a layer creates (header and data) becomes the data passed to the next level for processing. That layer will then add a header portion, and so on. Each layer, started at the Network (or Internet) Layer are considered in the sections which follow.

2.6.1 IP (Internet Protocol)

The Internet Protocol is responsible for the delivery of packets *(or* datagram) between host computers. IP is a connectionless protocol, that is, it does not establish a virtual connection through a network prior to commencing transmission; this is the job for higher level protocols.

IP makes no guarantees concerning reliability, flow control, error detection or error correction. The result is that datagram could arrive at the destination computer out of sequence, with errors or not even arrive at all. Nevertheless, IP succeeds in making the network transparent to the upper layers involved in voice transmission through an IP based network.

Any Voice over IP transmission must use IP. IP is not well suited to voice transmission. Real time applications such as voice and video require guaranteed connection with consistent delay characteristics. Higher layer protocols address these issues (to a certain extent).

The diagram below shows the header that precedes the data payload to be transmitted. In its most basic form, the header comprises 20 octets. There are optional fields which can be appended to the basic header, but these offer additional capabilities which are not necessary for VOIP transmission as described in this document.

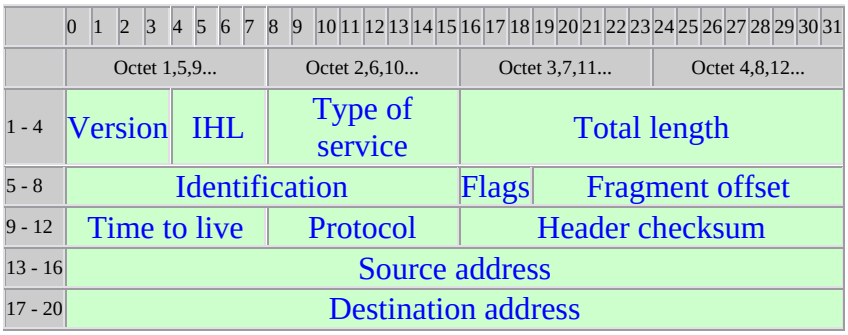

Figure 2.2 Internet Protocol (IP) Packet Format

The fields shown are briefly described below:

Version
The version of IP being used. For this format header, the version would be 4.

IHL
The length of the IP header in units of four octets (32 bits). For the basic header shown in this diagram, the value would be 5 (each line in the diagram represents four octets).

Type of service
Specifies the quality of service requested by the host computer send the datagram. This is not always effectively supported by routers or Internet Service Providers.

Total length
The length of the datagram, measured in octets, including the header and payload.

Identification
As well as handling the addressing of datagrams between two computers (or hosts), IP needs to handle the splitting of data payloads into smaller packages. This process, known as fragmentation, is required because, although a single IP datagram can handle a theoretical

maximum length of 65,515 octets, lower link layer protocols such as Ethernet cannot always handle these large packet sizes. This field is a unique reference number assigned by the sending host to aid in the reassembly of a fragmented datagram.

Flags
These flags indicate whether the datagram may be fragmented, and, if it has been fragmented, whether further fragments follow this one.

Fragment offset
This field indicates where in the datagram this fragment belongs. It is measured in units of 8 octets (64 bits).

Time to live
This field indicates the maximum time the datagram is permitted to remain in the internet system. This parameter ensures that a datagram which cannot reach its destination host is given a finite lifetime.

Protocol
This indicates the higher level protocol in use for this datagram. Numbers have been assigned for use with this field to represent such transport layer protocols as TCP and UDP.

Header checksum
This is a checksum covering the header only.

Source address
The IP address of the host which generated this datagram. IPv4 addresses are 32 bits in length and, when written or spoken, a dotted decimal notation is used (e.g.: 192.168.0.1).

Destination address
The IP address of the destination host.

2.6.2 UDP (User Datagram Protocol)

Generally, there are two protocols available at the transport layer when transmitting information through an IP network. These are TCP (Transmission Control Protocol) and UDP (User Datagram Protocol). Both protocols enable the transmission of information between the correct processes (or applications) on host computers. These processes are associated with unique port numbers (for example, the HTTP application is usually associated with port 80).
TCP is a connection oriented protocol; that is, it establishes a communications path prior to transmitting data. It handles sequencing and error detection, ensuring that a reliable stream of data is received by the destination application.

Voice is a real-time application, and mechanisms must be in place with ensure that information is received in the correct sequence, reliably and with predictable delay characteristics. Although TCP would address these requirements to a certain extent, there are some functions which are reserved for the layer above TCP. Therefore, for the transport layer, TCP is not used, and the alternative protocol, UDP, is commonly used.

In common with IP, UDP is a connectionless protocol. UDP routes data to it's correct destination port, but does not attempt to perform any sequencing, or to ensure data reliability.

	0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31	
	Octet 1,5	Octet 2,6	Octet 3,7	Octet 4,8
1 - 4	Source port	Destination port
5 - 8	Length	Checksum

Figure 2.3 User Datagram Protocol (UDP) Packet Format

The fields shown are briefly described below:

Source port
Identifies the higher layer process which originated the data.

Destination port
Identifies with higher layer process to which this data is being transmitted.

Length
The length in octets of the UDP data and payload (minimum 8).

Checksum
Optional field supporting error detection.

2.6.3 RTP (Real-time Transport Protocol)

Real time applications require mechanisms to be in place to ensure that a stream of data can be reconstructed accurately. Datagrams must be reconstructed in the correct order, and a means of detecting network delays must be in place.

Jitter is the variation in delay times experienced by the individual packets making up the data stream. In order to reduce the effects of jitter, data must be buffered at the receiving end of the link so that it can be played out at a constant rate. To support this requirement, two protocols have been developed. These are RTP (Real-time Transport Protocol) and RTCP (RTP Control Protocol).
RTCP provides feedback on the quality of the transmission link. RTP transports the digitized samples of real time information. RTP and RTCP do not reduce the overall delay of the real time information. Nor do they make any guarantees concerning quality of service.
The RTP header, which precedes the data payload, is shown in the diagram below:

	0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31			
	Octet 1,5,9	Octet 2,6,10	Octet 3,7,11	Octet 4,8,12
1 - 4	V=2 P X CC	M PT	Sequence number
5 - 8	Timestamp
9 - 12	Synchronization source (SSRC) number

Figure 2.4 Real-time Transport Protocol (RTP) Packet Format

Version
Identifies the version of RTP (currently 2).

Padding
A flag which indicates whether the packet has been appended with padding octets after the payload data.

X (Header extension)
Indicates whether an optional fixed length extension has been added to the RTP header.

CC (CSRC count)
Although not shown on this header diagram, the 12 octet header can optionally be expanded to include a list of up to contributing sources. Contributing sources are added by mixers, and are only relevant for conferencing application where elements of the data payload have originated from different computers. For point to point communications, CSRCs are not required.
M (Marker)
Allows significant events such as frame boundaries to be marked in the packet stream.

PT (Payload type)
This field identifies the format of the RTP payload and determines its interpretation by the application

Sequence number
A unique reference number which increments by one for each RTP packet sent. It allows the receiver to reconstruct the sender's packet sequence.

Timestamp
The time that this packet was transmitted. This field allows the received to buffer and play out the data in a continuous stream.

Synchronization source (SSRC) number
A randomly chosen number which identifies the source of the data stream.

2.7 The IP Header

The term ‘IP header’ is used to refer to the combined IP, UDP and RTP information placed in the packet. The payload generated by the codec is wrapped in successive layers of information in order to deliver it to its destination.
These layers are:

1) IP – Internet Protocol
2) UDP – User Datagram Protocol
3) RTP- Real-time Transport Protocol

RTP is the first, or innermost, layer added. This is 12 octets. RTP allows the samples to be reconstructed in the correct order and provides a mechanism for measuring delay and jitter.

UDP adds 8 octets, and routes the data to the correct destination port. It is a connectionless protocol and does not provide any sequence information or guarantee of delivery.

IP adds 20 octets, and is responsible for delivering the data to the destination host. It is connectionless and does not guarantee delivery or that packets will arrive in the same order they were sent.

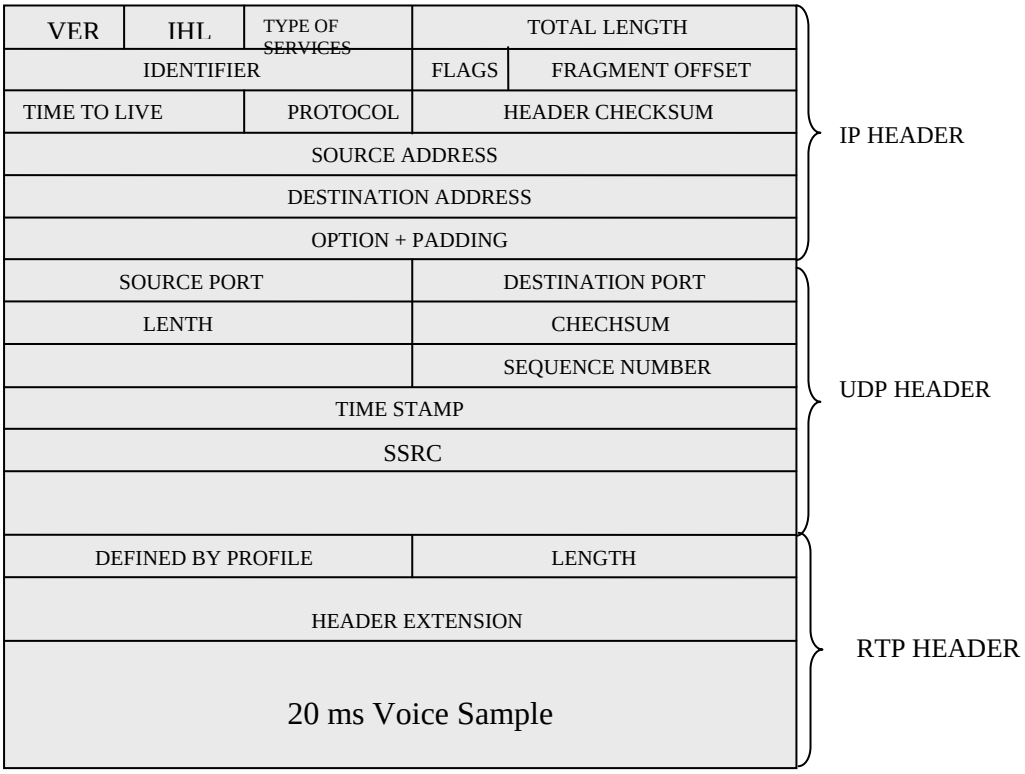

Figure 2.5 Diagram of Voice Packet Module

In total, the IP/UDP/RTP headers add a fixed 40 octets to the payload. With a sample period of 20 ms, the IP headers will generate an additional fixed 16 kbps to whatever codec is being used. The payload for the G.711 codec and 20 ms sample period calculated above is 160 octets; the IP header adds 40 octets. This means 200 octets, or 1,600 bits sent 50 times a second – result 80,000 bits per second. This is the bandwidth needed to transport the Voice over IP only; it does not take into account the physical transmission medium.

There are other factors, which can reduce the overhead incurred by the IP headers, such as compressed RTP (CRTP). This can be implemented on point-to-point links and reduces the IP header from 40 to just 2 or 4 octets. Though this is not that common today, its use will become more widespread with it being implemented with 3G mobile networks.

2.8 The Transmission Medium

In order to travel through the IP network, the IP packet is wrapped in another layer by the physical transmission medium. Most Voice over IP transmissions will probably start their journey over Ethernet, and parts of the core transmission network are also likely to be Ethernet. Ethernet has a minimum payload size of 46 octets. Carrying IP packets with a fixed IP header of 40 means that the codec data must be at least 6 octets long. The Ethernet packet starts with an 8 octet preamble followed by a header made up of 14 octets defining the source and destination MAC addresses and the length. The payload is followed by a 4 octet CRC. Finally, the packets must be separated by a minimum 12 octet gap. The result is an additional Ethernet overhead of 38 octets. Ethernet adds a further 38 octets to our 200 octets of G.711 codec frame and IP header. Sent 50 times a second – result 95,200 bits per second, see example 1 below. This is the bandwidth needed to transmit Voice over IP over Ethernet. Transmission of IP over other mediums will result in different overhead calculations.

2.9 Degrade Quality of Service (QoS)

Quality of Service (QoS) refers to the speed and clarity expected of a VOIP conversation. QoS makes attacks easier and defense harder. Implementing proper security measures such as firewalls and encryption introduces latency, jitter and packet lost.

2.9.1 Latency and Packet Overhead

The time from when words are spoken until they are heard at the other and Latency greater than 150 milliseconds is unacceptable in most cases

Unlike many data applications, voice cannot tolerate high levels of latency or delay. Toll-quality voice requires that sounds take 100 ms or less to travel from the speaker’s lips to the receiver’s ear. Delays exceeding 150 ms can start to irritate callers. When delays approach 500 ms, voice communication becomes impossible without reverting to the old practice of saying “over” to signal the end of a transmission. In addition to the latency physically present on the WAN link, VoIP protocols such as SIP, MGCP, and H.323 can introduce additional latency

The choice of the number of frames per packet is a trade-off between two characteristics: latency and packet overhead. Long sample periods produce high latency, which can affect the perceived quality of the call. Long delays make interactive conversations awkward, with the two parties often talking over each other. On the basis of this fact, the shorter the sample period, the better the perceived quality of the call. However, there is a price to pay. The shorter the sample period, the smaller the frames and the more significant the packet headers become. For the smallest packets, well over half of the bandwidth used is taken up by the packet headers; clearly an undesirable case. See figure next page

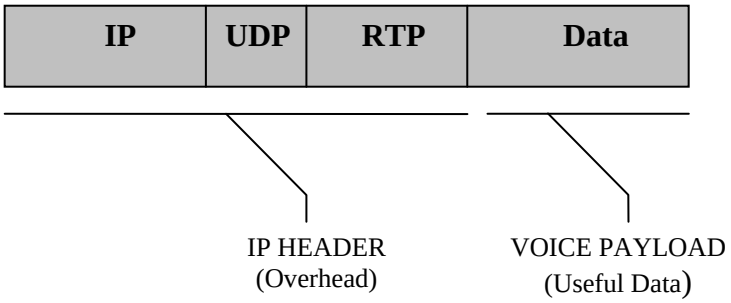

Figure 2.6 IP header forms a significant part of small Voice Over IP packets

2.9.2 Jitter

Non-uniform delays Requires buffering at the endpoints and application level reordering (more latency). Increased jitter makes it harder to tell when a packet is missing or just late. The irritation that latency creates for listeners is further compounded by jitter, which is variation in the delay of sequentially transmitted packets. When the packets are carrying voice, this variable delay should not exceed 50 ms for toll-quality voice. Buffers and queuing mechanisms are often used on VoIP equipment and routers to reduce jitter, but these approaches can introduce their own delay.

2.9.3 Packet loss

Voice can tolerate some packet loss, because the human brain automatically and unconsciously fills in small sound gaps. However, packet loss has to be kept to 1 percent or less for toll quality voice, and conversations start to break up when losses exceed three to five percent.

Because voice is so time-sensitive, the connection-oriented Transmission Control Protocol (TCP) is not used for transmitting VoIP packets. Instead, VoIP relies on the lighter, connectionless User Datagram Protocol (UDP). Since UDP cannot control the order in which packets are received, the actual voice content is encapsulated via Real-time Transport Protocol (RTP). Voice transmission architecture is thus RTP over UDP over IP. VOIP is highly sensitive to packet loss–Loss Rates as low as 1% can garble communications

2.10 Protocols Supported by VOIP

Two types of protocol have been established to maintain the activities of VOIP. These are as following.
i).SIP
ii).H.323.

i) SIP

The Session Initial Protocol (SIP) was designed by IETF It is an application layer protocol that establishes manages and terminates a multimedia session (call). It can be used to create two-party multiparty or multicast session. SIP is designed to be independent of the underlying transport layer it can run on either UDP or TCP Because of the open standards approach SIP devices are generally interoperable between different vendors. In a SIP based VoIP (voice over IP) system, most of the intelligence is in the phones rather than centrally in the PABX. This results in a much more scalable system. Installing additional phones only requires there to be a suitable network connection, apart from licensing issues, the system cannot become "full" unlike a PABX.

Open Standard - SIP based equipment is compatible with all other SIP equipment. Because SIP is flexible, it is in the manufacturers' interest to stick to the standard. With H.323 (the existing VoIP standard), many vendors have broken the standard to suit their needs, and in doing so produced non standard solutions. The flexibility of SIP should stop this happening.

It dissolves the concept of having a phone number for a phone, more an identifier for a person or workgroup. Where necessary this SIP address will also be reachable from the PSTN using a standard telephone number.

SIP is not like having a PABX system where the PABX and endpoints are linked together by IP. It is a true internet phone system. While it can be used to replicate the functionality in an PABX system it can potentially do much more.

Messages of SIP

SIP is a text-based protocol like HTTP SIP like HTTP uses messages Six messages are defined as
i. INVITE.
ii. ACK.
iii. BYE.
iv OPTIONS.
v. CANCEL.
vi. REGISTER.

Each message has a header and a body The header consists of several lines that describe the structure of each message Then we show their application in the sample session.

The caller initializes a session with the INVITE message after the callee answers the call the caller send an ACK message for confirmation. The BYE message terminates a session .The OPTIONS message queries a machine about its capabilities The CANCEL message cancels an already stared initialization process .The REGISTER message makes a connection when the callee is not available .

Addresses

In a regular telephone communication a telephone number identifies the sender ,and another telephone number identifies the receiver .SIP is very flexible .In SIP an email address an IP address a telephone number, and other types of addresses can be used to identify the sender and receiver . However, the address needs to be in SIP format (also called scheme

SIP provides the following functions:

Name Translation and User Location

to ensure that a call reaches the called party regardless of location. SIP addresses users by an email-like address. Each user is identified through a hierarchical URL built around elements such as a user's telephone number or host name (for example, SIP:user@company.com). Because of this similarity, SIP URLs are easy to associate with a user's e-mail address.

Feature Negotiation

SIP allows all parties involved in a call to negotiate and agree on the features supported, recognizing that all participants may not be able to support the same kind of features. For example, a session between a mobile voice-only telephone user and two video-enabled device users would agree to support voice features only. When the mobile telephone user leaves the call, remaining participants may renegotiate session features to activate video communications.

Call Participant Management

During a session, a participant can bring other users into the call or transfer, hold, or cancel connections.

Simple Session

A simple session using SIP consists of three modules: establishing, communicating, and terminating.

Establishing a Session

Establishing a session in SIP requires a three-way handshake. The caller sends an INVITE message using UDP or TCP to being the communication .If the callee is willing to start the session, it sends a reply message To confirm that a reply code has been received , the caller sends an ACK message

Communicating After the session has been established the caller and collee can communicate using two temporary ports

Terminating the Session The session can be terminated with a BYE message sent by either party

Tracking the Callee

What happens if the callee is not is not sitting at her terminal? She may be away from her system or at another terminal She may not even have a fixed IP address if DHCP is being used SIP has a mechanism (similar to one in DNS) that finds the IP address of the terminal at

which the callee is sitting. To perform this tracking SIP uses the concept of registration SIP defines some servers as registrars. At any moment a user is registered with at least one registrar server this server knows the IP address of the callee. When a caller needs to communicate with the callee the caller can use the email address instead of the IP address in the INVITE message. The message goes to a proxy server The proxy server sends a lookup message (not part of SIP) to some registrar server that has registered the callee. When the proxy server receives a reply message from the registrar server the proxy server takes the caller's INVITE message and inserts the newly discovered IP address of the callee. This message is then sent to the callee.

Performance of SIP

Have the same phone identifier follow you around. Just log on to whichever phone you are nearest to. Work from home and be on the office phone system. You would need a suitable ADSL line at home. Breakout onto the PSTN using the office gateway, thus eliminating the need for a second phone line or to claim for work calls.

If you are a multisite organization, then have the phone system bridged across all of your sites. Have free calls between offices. Many companies already have data links between offices.

Supporting Components to establish a SIP network

If user wanted to completely replace his existing PABX system by SIP, he must require the following components.

a) SIP Proxy/Registrar software
b) SIP telephones (such as the Snom 190)
c) A gateway (e.g. the vegastream)
d) SIP Media Server software (if voice mail or auto attendant are required)

Sip Registrar

Also know as a SIP proxy. The software forms a soft directory of users on the system.

The phones register with the proxy to allow dialing by methods other than IP address. The proxy provides hold and transfer services.

Ideally your proxy would be connected to a globally routable IP address which (with suitable firewall holes) would enable calls to phones registered on other proxies (to make IP calls outside your organization.)

A registration can be roamed from one phone to another. For example, a user could register either at home or at work. A user could even log on from home and work at the same time.

The software licenses for the SIP registrar are sold based on the number of registrations that they will accept. Users need one registration license per phone per registration.

So if a user had 2 phones in 2 offices, and only registered in one office at a time, then they would only need 1 server registration. If they registered in both offices then they would need 2 server registrations.

PSTN Gateway

This provides a gateway between the IP phone system and the traditional telephone system or an existing PABX system. Physically it is a box with Ethernet on one side and either ISDN or analogue phone connections on the other. Most gateways also contain some logging and billing software so you can extract log information to produce usage reports or billing information.

ProVu has found that ISDN gateways work a lot better than analogue FXO gateways.

Media Gateway

A media gateway terminates voice calls on inter-switch trunks from the public switched telephone network, compresses and pocketsize the voice data, and delivers compressed voice packets to the IP network. For voice calls originating in an IP network, the media gateway performs these functions in reverse order. For ISDN calls from the PSTN, Q.931 signaling information is transported from the media gateway to the media gateway controller (described below) for call processing.

Media Gateway Controller

A media gateway controller handles the registration and management of resources at the media gateway(s). A media gateway controller exchanges ISUP messages with central office switches via a signaling gateway (described below). Because vendors of media gateway controllers often use off-the-shelf computer platforms, a media gateway controller is sometimes called a softswitch.

Media Server

Although SIP phones have significant intelligence themselves, sometimes a server is required to help out. A media server is a software product that provides features such as Voice Mail, music on hold, conference calling, fault messages and auto attendant.

Key Advantages of SIP protocol

i. Call SIP addresses over the internet for free.
ii. Open Standard - compatible with all other SIP devices.
iii. More flexible than a traditional PABX - not limited to where you can run phone cable, but can still provide all the features you expect from a traditional PABX.
iv. Completely scalable - the intelligence is in the phone so never full to capacity.
v. Calls are made to a Personal identifier. Log on to your identifier from any location.
vi. No need for phone number just name, eg - tim@provu.co.uk

ii) H.323

H 323 is a standard designed by ITU to allow telephones on the public telephone network to talk to computers (called terminals in H 323) connected to the Internet.
A gateway connects the Internet to the telephone network. In general, a gateway is a five – layer device that can translate a massage from one protocol stack to another. The gateway here does exactly the same thing. It transforms a telephone network message to an Internet message. The gatekeeper server on the local area network plays the role of the registrar server, as we discussed in the SIP protocol.

Protocols used in H.323

H.323 uses a number of protocols to establish and maintain voice (or video) communication.
H.323 uses G.71 or G.723.1 for compression. It uses a protocol named H.245 which allows the parties to negotiate the compression method. Protocol Q.931 is used for establishing and terminating connection. Another protocol called H.225, or RAS (Registration/Administration/Status), is used for registration whit the gatekeeper.

Operation of H.323

Let us show the operation of a telephone communication using H.323 whit a simple example that shows the steps used by a terminal to communicate whit a telephone.
i. The terminal sends a broadcast message to the gatekeeper. The gatekeeper responds whit its IP address.
ii. The terminal and gatekeeper communicate, using H.225 to negotiate bandwidth.
iii. The terminal, the gatekeeper, gateway, and the telephone communicate using Q.931 to set up a connection.
iv. The terminal, the gatekeeper, gateway, and the telephone communicate using H.245 to negotiate the compression method.
v. The terminal, gateway, and the telephone exchange audio using RTP under the management of RTCP.
vi. The terminal, the gatekeeper, gateway, and the telephone communicate using q.931 to terminate the communication.

2.11 VOIP Billing System

Billing system VoIP BS 0.2 is designed for middle and small sized VoIP carriers with up to 1 mln subscribers. VoIP BS can handle about 1 mln minutes per day in real-time mode and no less than 150 authorization requests per second VoIP BS has been successfully tested for compatibility with the Cisco equipment (Cisco 26xx, 36xx, 53xx, 54xx, 38xx with digital or analog voice modules), Quintum gateways and MERA VoIP Transit Softswitch.

Billing system can run under UNIX operating systems: Linux and FreeBSD. The database is running under MySQL. The client part requires any operating systems from the Microsoft Windows family as well as any from UNIX family with web-browser installed (Microsoft

Internet Explorer, Netscape/Mozilla/Firefox, Konqueror). It's recommended that any E-mail client and Unzip-utility have been installed. Minimal hardware requirements: CPU Intel Pentium III–700 / RAM 256 Mb / HDD 20 Gb IDE.

VoIP BS includes multilanguage Interactive Voice Response (IVR). At this moment there are five languages:
i. russian;
ii. ukrainian;
iii. english;
iv. chinese;
v. vietnamese.
Billing supports PIN authorization, ANI (Caller ID) authorization and Callback services based on the ANI. IVR used Cisco TCL IVR version 1.0 or 2.0.

System Architecture

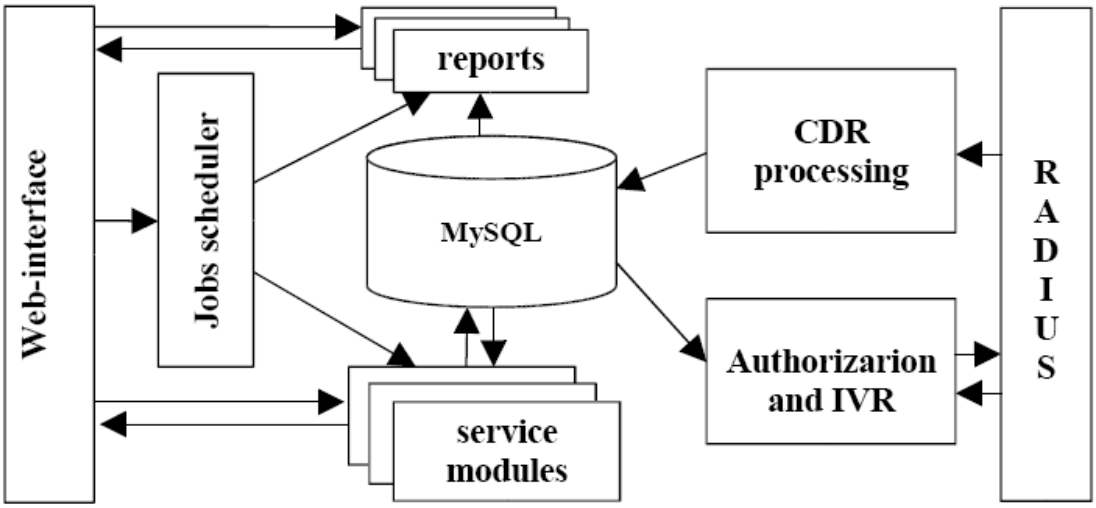

Figure 2.7 VOIP Billing System Architecture

VOIP BS consists of several modules which interact with each other. Job scheduler controls major part of modules. It takes query from web-interface and then decides when and which module should handle the query. Job scheduler looks after the number simultaneously running modules and the queue queries for each module and tries to balance load on the system. Job scheduler starts recurring tasks also. VoIP BS works with equipment by RADIUS protocol. Call Detail Records (CDR) are sent to CDR processing, which tries to handle calls and puts the handled information into DBMS. Incoming IVR queries are sent into Authorization module. First of all the module takes incoming call's data and customer's information and decides if this call is permitted.

2.12 Benefits of VOIP

VOIP today makes up a relatively small percentage of overall communications revenue. But the technology is growing rapidly. *Telephony Magazine* recently published figures stating that more than a third of all business phones shipped during the second quarter of 2003 were

IP-enabled. Last year, Cisco announced it had sold its 2 millionth IP handset. It is impossible to enumerate all the advantages Voice over IP, in large part because the technology is so new, and many of its features have yet to be developed. But here is a sampling of the benefits:

Lower costs

VOIP leverages the inherent economic advantages of IP and allows voice services to be delivered at lower cost to end- users. Tone service enables enterprises to forgo investment in expensive premise-based switching equipment. It also eliminates long-distance charges between branch offices and cuts costs associated with IT and is administration. It is estimated that with VOIP, businesses installing new phone systems can save as much as 40 percent on communications costs compared with traditional PBX systems, taking both capital and operational costs into account.

Convergence

VOIP allows the seamless fusing of voice and data applications onto a single platform – a concept known as "convergence." A converged network offers businesses numerous advantages over the old, two-network voice and data system, including lower costs, simpler network management and the ability to deploy new applications that heighten both productivity and customer service.

Innovative tele-working

With Voice over IP, employees are less tied to bricks-and-mortar offices. For instance, a stay-at-home parent who works in technical support could use VOIP to direct incoming calls to his home office between the hours of 8:00 a.m. and 3:00 p.m., while his children were at school. During that "on" period, he would use his broadband connection to receive tech support calls at home, with full access to customer and product data stored on a remote company server. Such flexibility allows telecommunications- intensive companies to use part-time employees spread out in areas across the country.

Multimedia conferencing

VOIP enables multiple users to communicate with one another via voice and video while drawing on data sources like spreadsheets and financial statements. For example, it would allow members of an engineering team located in different parts of the world to work together on the design of a building. They could collaborate via voice and share data and documents in real time to revise design specifications for the project.

High-power call centers

VOIP communications will enable call-center operators to provide more focused assistance to customers. IP technology allows the operator to receive calls and relevant customer data

simultaneously. The operator can access case histories, account and credit information, inventory and shipping data – at the exact moment the customer makes contact.

Unified messaging

VOIP allows a user to have a single message platform for all types of communications. Rather than receive e- mail on a computer, voicemail on the phone, faxes on fax machines, and pages on a pager, Voice over IP can route all to a single unified mailbox. A voicemail can be converted into text using voice recognition software, and an e- mail can be converted into a voice message. Users can organize, store, and prioritize these messages in the manner that suits them best.

Location scheduling

VOIP users can create a daily location schedule (and update it anytime from anywhere) indicating where communications should be forwarded. In other words, a user could direct communications (of any form) to a mobile device during her commute, to her office during the day, to her brother's house during the holidays, and to a unified messaging center when she is eating dinner.

Simplified relocation

VOIP makes moves and changes much less painful and less expensive. In a circuit-switched system, a company moving an employee to a different office has to map extensions, re-program special call-handling features, activate new phone sets, and re-customize the employee's phone configurations. VOIP simplifies this process. Employees moving to an office in another country take their customized features with them automatically because configuration data is tied to the user rather than a physical extension.

2.13 Summary

It's worth noting that, with traditional phone service, the set of features available to businesses and consumers today – call waiting, caller ID, etc. – represent the pinnacle of technological achievement for the PSTN. With IP voice communications, by contrast, sophisticated features of the kind listed above are first-generation building blocks from which untold future applications will evolve and make a rebellion to the telecommunication industry.

Chapter – 3
Introduction to PABX

3.1 Introduction:

A private automatic branch exchange (PABX) is an automatic telephone switching system within a private enterprise. Originally, such systems - called private branch exchanges (PBX) - required the use of a live operator. Since almost all private branch exchanges today are automatic, the abbreviation "PBX" usually implies a "PABX."

Some manufacturers of PABX (PBX) systems are distinguished their products from others by creating new kinds of private branch exchanges. Rolm offers a Computerized Branch Exchange (CABX) and Usha Informatics offers an Electronic Private Automatic Branch Exchange (EPABX).

In today market, there are different types PABX exchanges. These PABX exchanges are differentiated according to given services into 7 categories. A couple of them are discussed below:-

3.2 Private Automatic Branch Exchange 1 (PABX 1)

The PABX 1 is a modern, automatic telephone system which effectively meets the communication needs of small or medium sized businesses and commercial organizations. The system accommodates up to 10 exchange lines and 49 automatic extensions and provides for automatic connection to the public telephone network and full internal dialing between extensions. Incoming exchange calls are received on a small, compact, key-operated switchboard on a table or desk, and are dealt with by the PABX operator. A small number of manual extensions and lines to other PBXs can be connected to the system. Note: PBX means either a Private Automatic Branch Exchange (PABX) or a Private Manual Branch Exchange (PMBX). The automatic equipment is contained in a steel cabinet which has front and rear doors to let the maintenance engineer get at both sides of the racks. Power to operate the equipment is supplied from a secondary-cell battery linked with an automatically controlled charging unit connected to the mains. The PABX 1 is available in four sizes. Details are shown with the general information.

3.2.1 General facilities

Extensions call each other simply by dialing the number. Extensions make outgoing exchange calls by first dialing 9 for connection to an exchange line. Any extension can be connected in such a way that it cannot dial 9 for an exchange line but can get through to the public system only by way of the operator's switchboard. Incoming exchange calls are first received at the switchboard and then extended to the required extension by the operation of press-buttons. A few manual extensions can be provided. Automatic extensions can call them direct but they receive incoming exchange calls and make all outgoing calls through the switchboard. Lines to other PBXs can be provided. Extensions dial 7 for connection to an

inter-PBX line and in some instances it is possible over these lines to dial direct to extensions on a distant PABX. On an outgoing exchange call, timing of the call stops and the exchange line is released immediately the extension handset is put back. Extensions dial 0 to call the switchboard operator.

3.2.2 Switchboard facilities

The PABX 1 could be fitted with one of two types of switchboard. The early, SA8120 version was made of wood, whilst the latter, SA 8133 type was of a modern design with the usual grey plastic cover. Extensions are called from the switchboard by press-button. Ringing is automatically applied to extensions on calls from the switchboard. Calls to or from extensions are automatically released when the extension handset is put back. If on an incoming call the called extension is engaged the caller can wait and be automatically connected immediately the extension is free. The PABX operator can interrupt an engaged extension to ask if another call can be accepted. A ticking sound indicates that the operator is on the line. An incoming or outgoing exchange call can be held on the switchboard by connecting it to a special holding number. After holding, the call can be connected to an extension by press-button.

3.2.3 Extension facilities

Calls between extensions are released when either extension handset is put back. On calls to or from the exchange or another PBX, pressing the button on the telephone twice, calls in the PABX operator, who can then hold the call or transfer it as required. An automatic extension can, by pressing the button on the telephone once, hold an exchange call and make an enquiry of another extension. The original call can be returned to by pressing the button again; or if the handset is simply replaced the calls transferred to the other extension. This operation can be repeated as often as required on the call. Manual extensions do not have this arrangement. For night service there is a special key on the switchboard. When this is in the `night service' position incoming exchange calls cause suitably sited bells to ring continuously. Any automatic extension can then answer the call by dialing 8, and can, if necessary, transfer it to any other extension. A different kind of night service can be given by putting individual exchange lines through to selected extensions.

3.3 PABX 5

The PABX5 was initially available in two sizes, namely for three exchange lines and ten extensions or five exchange lines and 20 extensions. It is now only available in the larger size. A Manual switchboard is not provided. All calls are answered by designated extensions and transferred as required.

The automatic equipment (Equipment PABX SA8450) comprises a single self contained unit which employs PO Type 2 Uniselectors and relays, mainly 3000-type, for all switching purposes. The unit is floor-mounted and is supported at the top by two metal brackets which are bolted to a wall. A detachable metal dust cover is fitted over the unit. A power supply at a nominal 50V D.C. is obtained from a Power-unit No. 69A. To aid installation a version has

been introduced with more uniselector mechanisms fitted. This is coded Equipment PABX5A.

3.3.1 Facilities

The following general facilities are provided as standard and a more detailed description is given in the relevant diagram notes.

3.3.2 Public Exchange Service

This is given over bothway working exchange line circuits terminated at the PABX with Relay-sets SA8455. The calling condition to the public exchange may be either a loop or an earth on the B wire followed by a loop.

a) Outgoing Exchange Calls

Access by an extension to a public exchange line, as with other standard PABX's, is obtained by dialing the digit 9. It is possible to prohibit this facility from any particular extension as desired.

b) Incoming Exchange Calls

Incoming exchange call bells are sounded and the calls are answered by extensions designated for that purpose. There may be a number of designated extensions (up to 10 or 20 according to the size of the equipment installed) any one of which may answer an incoming call on any exchange line merely by lifting the receiver. An incoming exchange call, having been answered by a designated extension, may be transferred by the designated extension to any other extension allowed connection to incoming calls by the use of the enquiry and automatic transfer facilities. If required any particular extension may be prohibited from receiving incoming exchange calls.

c) Extension-to-Extension Calls

These are dialed direct on connecting circuits. The connecting circuit is held for the duration of the call.

d) Enquiry calls

An extension engaged on an exchange call may make an enquiry call to any other extension or inter-PBX extension. An extension engaged on an inter-PBX call (either inter-PBX extension or inter-PBX private circuit) may make an enquiry call to any other extension or inter-PBX circuit.

The extension wishing to make the enquiry depresses the button on the extension instrument once to receive dial tone and then dials the required extension number. The public exchange or inter-PBX circuit connection is held for the duration of the enquiry call. When the enquiry call is ended the instrument button is again depressed and the extension is reconnected to the

exchange line or inter-PBX circuit. Non receipt of dial tone after the first depression of the instrument button is an indication that all connecting circuits are engaged. The extension then should return to the exchange line in the normal manner, i.e. by a second depression of the button, and either abandon or attempt the enquiry call later.

e) Automatic transfer

An extension having made an enquiry to a second extension while holding an exchange line or inter-PBX line may also transfer the call to the second extension. This is done by asking the second extension to hold on while the handset of the originating extension is replaced. This operation may be repeated as often as required.

f) Exchange line transfer guard

If transfer be attempted to an extension normally barred from receiving exchange calls, the incoming exchange call bells are again sounded after a short timing period and a designated extension would get access to the exchange line relay-set as for a normal incoming call. Similarly, if transfer is not successfully accomplished the incoming exchange call bells will be sounded.

g) Intrusion (or trunk offering)

A designated extension wishing to transfer an incoming exchange call to a second extension may obtain busy tone on dialing the required extension. To trunk offer, or intrude, the designated extension dials the digit 1 and obtains access to the busy extension to offer the exchange call. During the time that the designated extension is intruding in this way on an established call a 'warn' pulse is automatically applied to that call to indicate the intrusion.

h) Night service

The normal operating conditions are retained for night service but one extension arranged as designated and/or limited facility during the day can be converted to full facility and become designated when the night key is operated. Also additional or different call bells can be operated to indicate an incoming call. The night service key will be fitted on a selected extension telephone or the attendant's telephone if provided.

3.3.3 Inter-PBX circuits

Each inter-PBX circuit provided reduces by one the number of exchange line circuits available. The types of inter-PBX circuit available are as follows:-

Bothway dialing, loop signaling (Relay-sets SA8457) for working to other PABX's.

Bothway earth signaling with earth dialling inward (Relay-set SA8456) for working to PMBX's.

Bothway dialling for working to other PABX's or dialling in from PMBX's, 1 VF (SSAC13) signaling (Relay-set SA8460). This system is required when the signalling limits for 3.8.1 and 3.8.2 are exceeded or when hf plant is used. (It is necessary to provide Equipment, Signalling, No. 24/ ... when vf signalling is used.)

These circuits may be connected as inter-PBX extensions in which case incoming exchange calls may be extended over them, or as inter-PBX private circuits when access to public exchange lines is prohibited. Inter-PBX circuits may be provided over up to three different routes, direct access being obtained by a single routing digit (7, 8 or 0) in each case.

3.3.4 Alarms

A fuse alarm lamp is provided on the night service control telephone or the attendant's telephone if provided.

3.3.5 Attendant's telephone

This consists of a modified Telephone No. 710 mounted on an Answering Unit No 1a (grey). Two designated extensions are terminated on the Answering Unit No 1A which has SPEAK, RELEASE, and HOLD buttons associated with each extension. An exchange line calling lamp and fuse alarm lamp are provided on the telephone in addition to the night service key.

3.3.6 Numbering scheme

The extension numbering ranges for the two sizes are:-

Size	**Number range**
3 + 10	20-29
5 + 20	20-39

The single digits 7, 8, 9 and 0 are reserved for routing of calls to public exchange lines or inter-PBX circuits. A maximum of three routes for exchange and inter-PBX lines is available on the 3 + 10 and a maximum of four routes on the 5 + 20.

3.3.7 Description of circuits

Dgm SA8450 shows the trunking scheme for a PABX5 and a brief description of the features of the circuits follows.

Extension line circuits, linefinders, connecting switch and exchange (EF) finders (SA8451). All finders and connecting switches are PO Type 2 25-point Uniselectors. The extension circuits are in two groups of ten and are multiplied to the banks of the line and EF finders and to the banks of the connecting switches. In addition each exchange line or inter-PBX circuit

has an associated auxiliary extension line circuit and these circuits are also multipled to the banks of the-line and EF finders but not to the banks of the connecting switches.

Each linefinder and connecting switch is directly connected to a connecting circuit and the controlling relays for these switches are accommodated in the connecting circuit. The EF finders are correctly connected to the exchange line or inter-PBX circuits and the controlling relays for these finders are within the exchange line or inter-PBX circuits as appropriate. The connecting switches are arranged to 'home'; the linefinders and EF finders are not.

Connecting circuits (SA8454) Each connecting circuit is arranged as a strip-mounted set, a new feature being that the sets may be jacked in and out.

Up to two or four connecting circuits may be provided on the 3+10 and 5+20 sizes respectively. Access to the circuits is via a start chain, and to ensure that only one linefinder will search for the calling line the start circuits of the linefinders not in use are held open (or busy) until the calling line is found. Thus only one linefinder can hunt at any one time. When a calling extension has been found dial tone is returned from the connecting circuit and a period of approximately 15 seconds is allowed for dialling to take place. Failure to dial within this period causes the connecting circuit to be released and the extension made PG.

3.3.8 Local calls

If an extension number in the range 20-29 is dialled the CS Uniselector is stepped directly by the dialled pulses and the CS wipers are positioned accordingly. The connecting circuit then behaves in the same way as a final selector. If an extension in the range 30-39,is dialled, during the inter-train pause following the initial digit 3 the CS wipers are automatically driven past the contacts used for extensions 20-29 and a second. digit 1-0 will allow connection to extensions 30-39 as appropriate. On clear down at the end of a call the CS Uniselector homes.

3.3.9 Outgoing exchange line or inter-PBX calls

For an outgoing exchange line call the CS Uniselector steps according to the digit dialled and a start signal is applied to the start chain of the exchange line relay-sets via a CS are and wiper. The EF finder associated with the exchange line relay-set to be used rotates and finds the marking condition set up by the connecting circuit and its linefinder (this is possible since some arcs of the linefinders and EF finders are multipled together) so that both the linefinder and EF finder stand on the calling extension's position. The connecting circuit is then released, the calling extension is connected to the exchange line relay-set, and the CS Uniselector homes. A similar sequence of events takes place on outgoing inter- PBX calls.

3.3.10 Exchange line relay-sets (SA8455)

A maximum of three or five exchange line relay-sets may be fitted in the 3+10 and 5+20 sizes respectively. As with the connecting circuits these are strip-mounted sets and may be jacked in or out. It should be noted that the maximum number of exchange lines is reduced by

one for each inter-PBX circuit fitted. An EF finder is associated with each relay-set.

Outgoing calls progress as described above, i.e. a start signal is received from the connecting circuit which causes the EF finder to rotate, find the extension, release the connecting circuit and connect the extension to the exchange. The calling signal to the public exchange will normally be earth on the B wire although the relay-set can be arranged to work 'loop calling' if required.

On incoming exchange calls ringing current operates a relay in the exchange line relay-set and the incoming call bells are sounded. A designated extension, to answer the incoming call, lifts his receiver and a start signal is applied to the calling exchange line relay-set. The EF finder of that relay-set rotates to find the answering extension, the incoming exchange line call is connected to this extension and the incoming call bells are disconnected.

If two incoming calls arrive simultaneously, the exchange line start chain is arranged so that only one EF finder can hunt at anytime. In these circumstances a designated extension would answer one of the incoming calls, the incoming call bells would continue to ring and a second designated extension would answer the remaining call. The incoming call bells would then be disconnected.

On an outgoing or incoming exchange call it is possible during the course of the call for the extension user to role an enquiry call to a second extension and to do this use is made of an auxiliary line circuit associated with the exchange line relay-set. Depression of the button on the extension telephone gives access to the auxiliary line circuit which is used in the same manner as an extension's line circuit to obtain connection to a connecting circuit. At the same time a loop is presented to the exchange line to hold the outgoing connection or maintain the correct answering condition. The enquiry call now proceeds in the same way as any other local call and when the enquiry has been made a second depression of the telephone button will re-connect the extension to the exchange line and release the connecting circuit.

If required it is possible for the exchange connection to be transferred from the enquiring extension to the second extension (except where this facility has been prohibited). During the enquiry call the connecting circuit is being used with its linefinder standing on the auxiliary line circuit of the line relay-set and its CS switch standing on the extension number dialed. Thus it is possible to mark this second extension via the linefinder and CS switch, and to affect transfer the calling extension's receiver is replaced. The following sequence of events then takes place. The connecting circuit begins clear down, the EF finder of the exchange line rotates until it finds the marked extension, the complete clear down on the connecting circuit is then permitted, the required extension is connected to the exchange line relay-set and then finally through to the exchange line. The holding loop is disconnected since the exchange line is now under the control of the extension. This process can be repeated as often as required.

3.3.11 Inter-PBX circuits (SA8456, SA8457, SA8460)

These can only be fitted where less than the maximum capacity of exchange lines is required, each inter- PBX circuit fitted replacing one exchange line circuit. An EF finder is associated with each relay-set and outgoing calls follow closely the pattern of outgoing exchange line

calls, a single routing digit being necessary to gain access to a relay-set. If the inter-PBX circuit is to a PMBX no further dialling is required by the extension after the routing digit, but for circuits to another PABX a routing digit followed by the number of the required extension will be dialed.

On an incoming call seizure of the relay-set by the distant PABX or PMBX results in the seizure, via the auxiliary line circuit, of a connecting circuit. The required number is dialed into the connecting circuit and if it is free ringing is applied to the called extension's line. When the called extension answers, a reversal is repeated from the connecting circuit to the inter-PBX line relay-set and, at the same time. the position of the called extension is marked on an arc of the EF finders via the linefinder and CS switch of the connecting circuit. Receipt of the reversal from the connecting circuit causes the EF finder of the inter-PBX circuit to rotate and find the marked extension, the connecting circuit is released, and the extension connected through to the inter-PBX circuit.

It is not possible for the inter-PBX circuit to obtain access to exchange line routes.

3.3.12 Tones

Dial, busy, ring, number unobtainable and warn tones are obtained from a ringing, tone and miscellaneous circuits relay-set (SA8452) although the actual tone generators are detachable units within this relay-set. The diagram relating to the ringing and tone generators is SA8453.

3.3.13 Fuse alarms (SA8452)

If a fuse is blown an alarm lamp is lit on a particular extension telephone to give a fault indication.

3.3.14 Mains fail

In the event of a mains failure each exchange line is connected to a predetermined extension and incoming exchange calls are received directly on those extensions. The extensions are also able to make outgoing exchange calls. On the restoration of the mains supply the exchange lines are disconnected from the extension lines and reconnected to the exchange line relay-sets. However, should an extension be engaged on an exchange line call when the mains supply is restored the reconnection of the exchange line to the exchange line relay-set is delayed until the clear down of that call.

3.4 Summary

PABX is important technical part of business world. It was developed to minimize telecommunication cost as well as to reduce local PSTN traffic. No company, organization even institution cannot think a day without this PABX

Chapter – 4
Design and Implementation

4.1 Introduction

In developed countries a lot of researches and applications were conduct with packet switching based communication system to eliminate the ancient circuit-switching communication, but in Bangladesh, circuit-switching based PABX communication systems are still used. In this regard this concept of VOIP based PABX system is absolutely new. This idea integrates the advantages of circuit switching and packet switching and eliminates some long-awaiting problems of typical PABX system.

4.2 Design

The VOIP PABX System replicates the idea of Computer Branch Exchange, which also requires an additional hardware, but this system does not need any additional hardware.

The PABX Server acts as a bridge between the internal user and the PSTN phone users. When a subscriber or user of PABX Server dials a number, the number is sent to the PABX Server. PABX Server checks whether the dialed number is a user of internal network or the call is destined to PSTN phone user. If the call is for an internal user then PABX Server tries to connect to the appropriate user, if it fails to acquire a connection then it sends the caller a connection failure message. This scenario occurs only if the receiver computer is currently

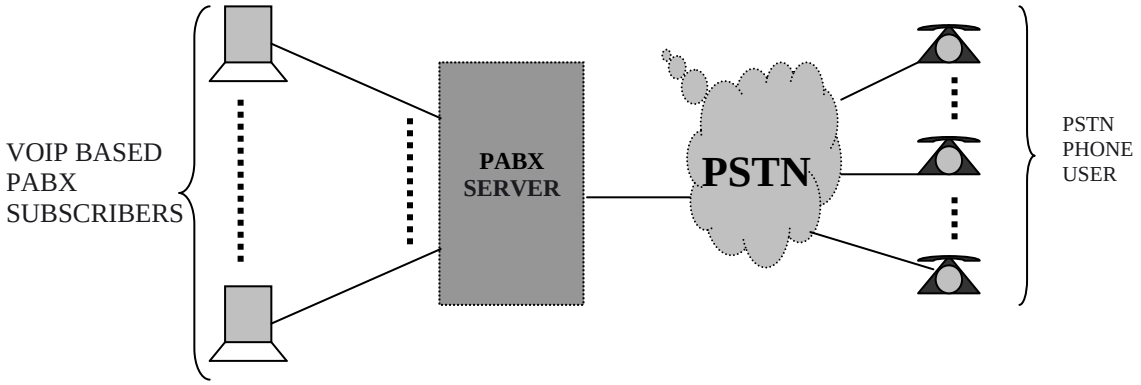

Figure 3.1 Basic VOIP Based PABX System

off or the client software currently not running on the client Computer. If connection is established successfully then the client is notified that a call has arrived. This time the client can either receive or hang-up the call. If Client receives the call then a call reception message is sent to the caller and PABX Server starts a clock for billing purpose. If Client decides to hang-up on the call then a call hang-up message is sent to the caller and PABX Server closes the connection.

If the dialed number is a PSTN number, then the server starts dialing using a modem. When the receiver receives the call PABX Server just then establishes a connection with the caller and starts a clock for billing.

When either of the party hang-up or disconnects the call, PABX Server stops the clock to get the talk-time.

When a call from PSTN arrives to PABX Server, it asks the caller for an extension number. PABX Server tries to connect to the desired extension user; if it fails to acquire a connection then it sends the caller a connection failure message. This scenario occurs only if the receiver computer is currently off or the client software currently not running on the client Computer. If connection is established successfully then the client is notified that a call has arrived. This time the receiver can either receive or hang-up the call. If Client receives the call then a call reception message is sent to the caller. If receiver decides to hang-up on the call then a call hang-up message is sent to the caller and PABX Server closes the connection.

4.3 Flowchart

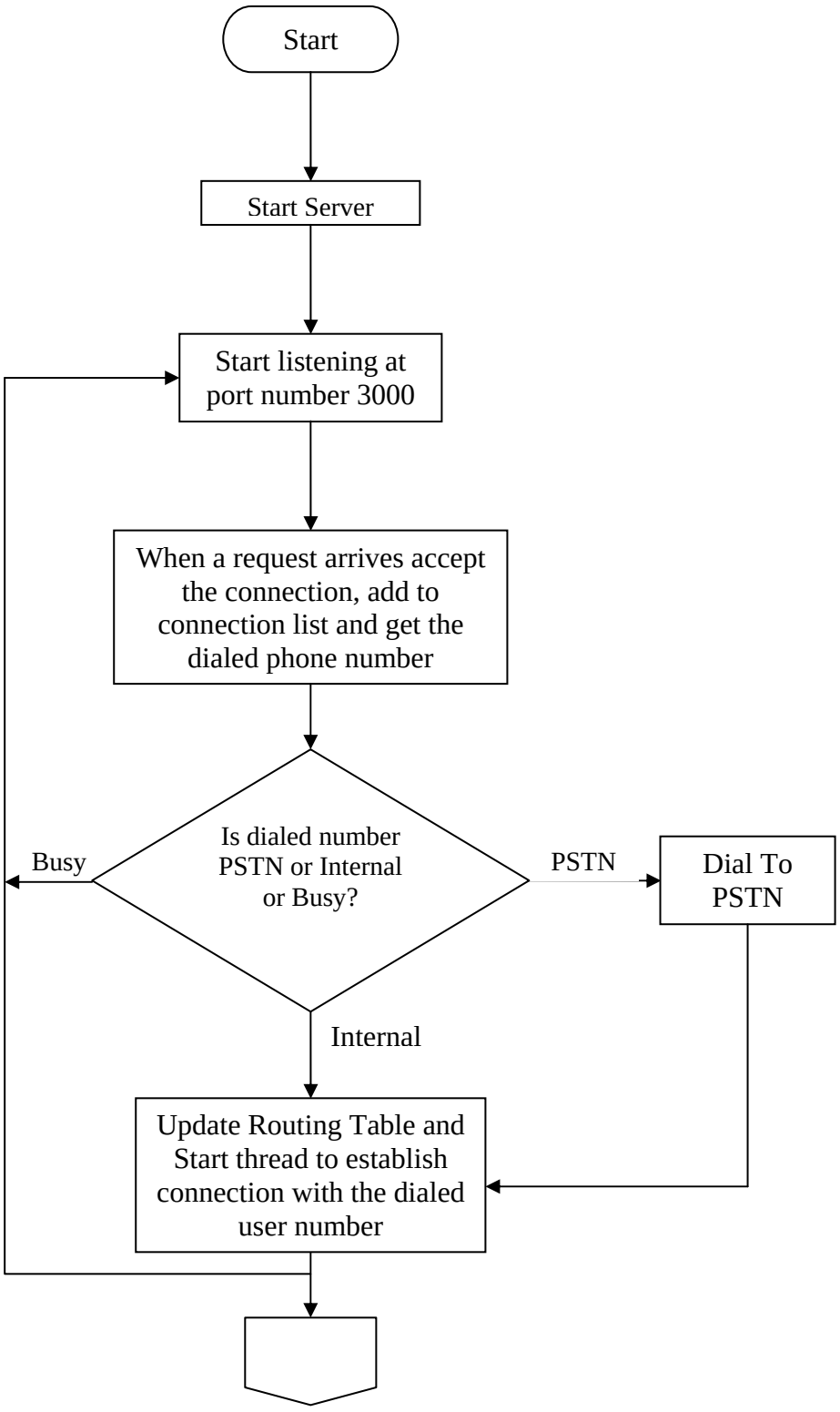

Figure 4.2 Flow chart of PABX server work (Continued)

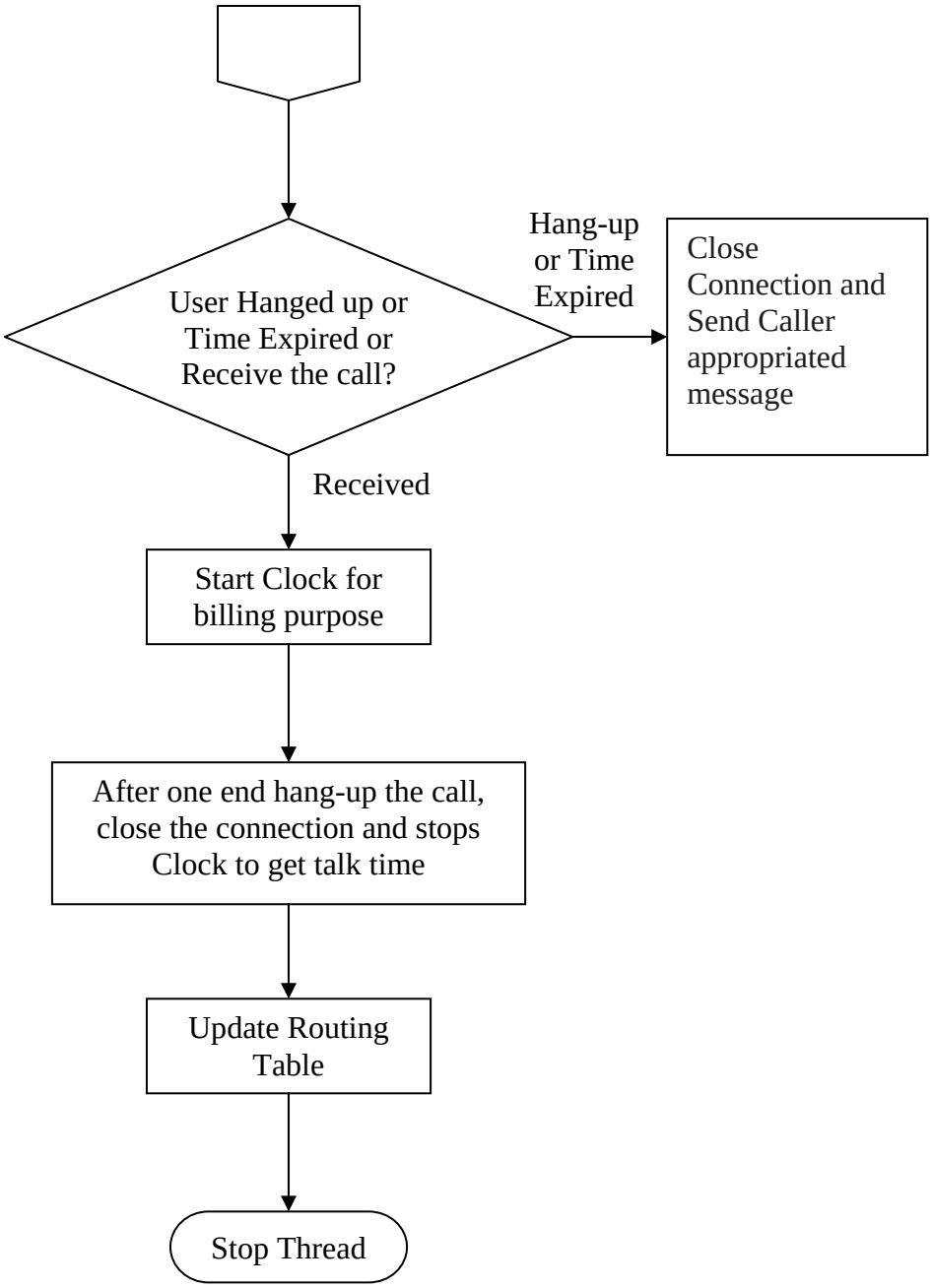

Figure 4.2 Flow chart of PABX server work

4.4 Implementation

This VOIP based system PABX was implemented being more oriented to software in comparison to hardware keeping in mind to reduce the PABX cost. This soft PABX exchange was designed using concurrent server concept. Though this PABX can be implemented in existing network on server or Always on PC, it is also made in compatible with Embedded Device to be installed. Then this embedded system will act as stand alone PABX exchange.

In this VOIP based PABX, when the Administrator of PABX Server starts the server, the application at first initializes the connection list. Then the server socket opens a port at port number 3000. After that, it start listening to that port for incoming connection request to the server.

Whenever, a request for call arrives from a subscriber of the VOIP Based PABX System to

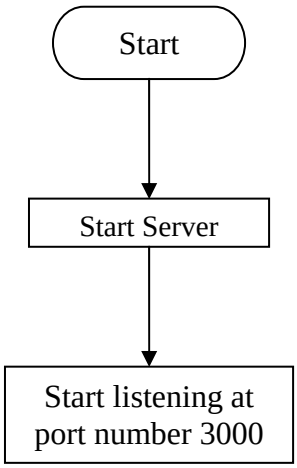

Figure 4.3 Initial Steps for Starting PABX Server

the server it inserts a connection socket to the connection list and accepts the connection. After that Server initializes some attributes of that connection. Then Server gets the dialed number from the caller. At this time the Server checks that whether the dialed number is for another subscriber of the PABX System or for a user connected with the PSTN. If Server finds that the number is for another subscriber of the PABX System, then at first it checks whether the receiver is busy or not. If the receiver is busy then Server sends a _USER_BUSY_ message to the caller and closes the connection. After that Server cleans up that connection from the connection list. If the receiver is not busy then Server spawns a thread and start listening for the next incoming call.

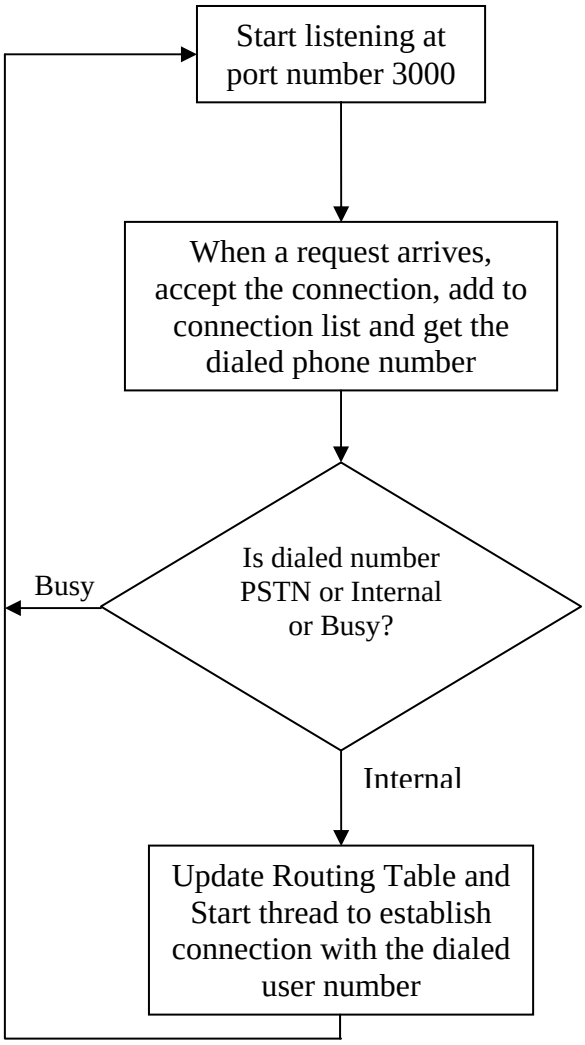

Figure 4.4 Steps of Request Arrival and Call Routing

The thread spawned by Server initializes another socket for the receiver. Afterwards that thread creates a child-parent relationship between the receiver and caller sockets. Next, the thread tries to establish a connection with the receiver. If connection with the receiver fails, then the thread sends an appropriate error message explaining the cause for which the connection has failed, then it closes the connection and cleans up that connection from the connection list. If the thread successfully establishes a connection with the receiver then the thread handover the control to the connection socket.

The connection socket then waits for a confirmation signal which tells it whether the receiver hanged-up the call or received the connection or time has expired for the user to receive the call. The Expiration time is set to eight seconds that means if the receiver does not receive a call within eight seconds then the call will be counted as a missed call. If time expires for the receiver then a _USER_TIME_EXPIRED_ message arrives from the receiver side to the

Server side. This message is then passed to the caller end. After that Server closes the connection and cleans up that connection from the connection list.

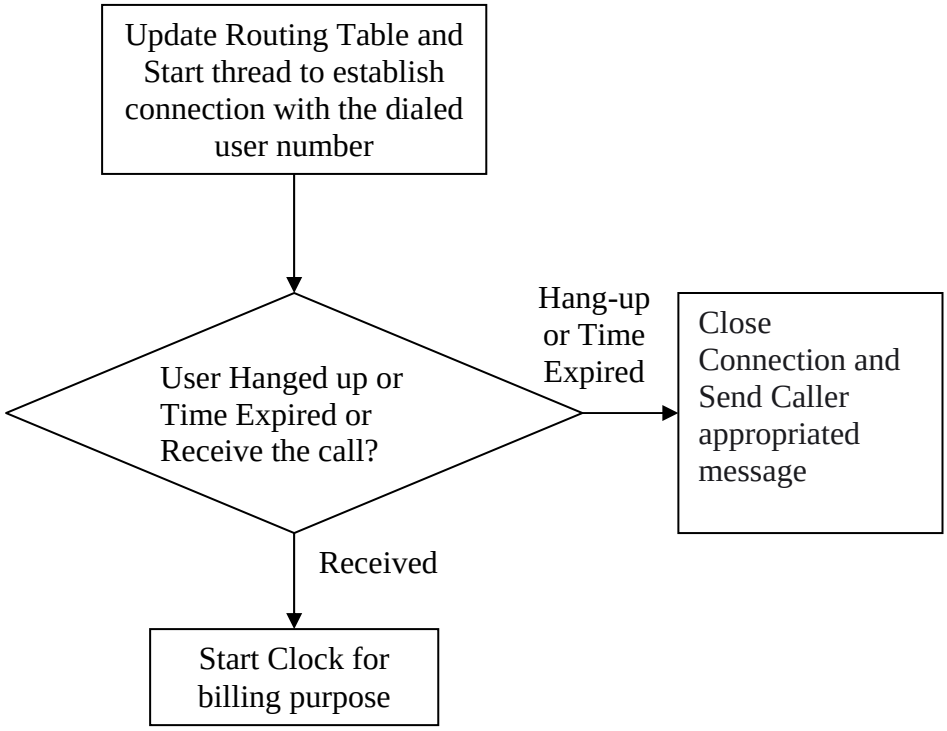

Figure 4.5 Steps of Call Reception & Declination

If receiver receives the call then a _USER_RECEIVED_ message is passed to the receiver. This ensures that the receiver has accepted the call and there is no problem in the connection and the receiver and caller can start their conversation. When the _USER_RECEIVED_ message arrives at the server end application gets the timestamp as the starting time of conversation. When either of the party, that means either the receiver or caller disconnects the call, the application at the Server end gets another timestamp as the end of conversation time. The difference between the starting time and end of conversation time yields the talktime, which can be used for billing purpose. After that Server closes the connection and cleans up that connection from the connection list.

If receiver hang-up the call when the call arrives, then a _USER_HANGUP_ message arrives to the Server end and then this message is passed to the caller end. After that Server closes the connection and cleans up that connection from the connection list.

During this negotiation for establishing a call if any error occurs and any different messages from those explained above arrives at the server side then Server sends an ‘invalid command’ error message to the caller and closes the connection and cleans up that connection from the connection list.

If the number dialed by the caller is destined to a receiver who is a subscriber of PSTN phone line, then the server spawns a thread which starts dialing the number and waits for the reply from the receiver. If the receiver is busy then a ‘user busy’ message is passed to the caller. After that Server closes the connection and cleans up that connection from the connection list.

If receiver receives the call then the Server redirects the call to the caller. When either of the party that means either the receiver or caller disconnects the call, the application at the Server end closes the connection and cleans up that connection from the connection list.

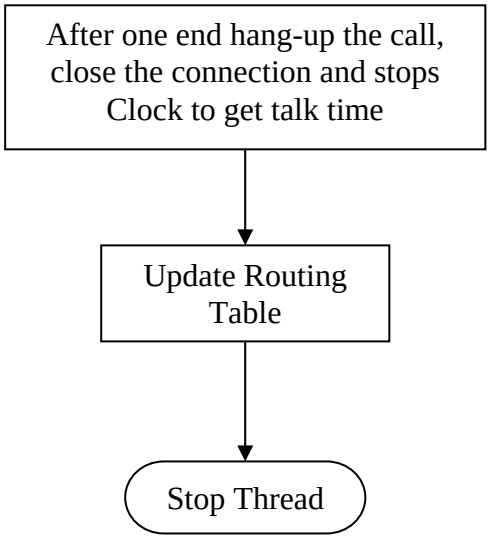

Figure 4.6 Step of Ending a Call Session

If time expires for the receiver then a _USER_TIME_EXPIRED_ message arrives from the Server side to the caller side. After that Server closes the connection and cleans up that connection from the connection list.

If a call from PSTN arrives to the PABX Server then PABX Server asks for the extension number. When receiver dials the extension number then Server checks that whether the receiver is busy or not. If the receiver is busy then Server sends a user busy message to the caller and closes the connection. After that Server cleans up that connection from the connection list. If the receiver is not busy then Server spawns a thread and start listening for the next incoming call.

The thread spawned by Server initializes another socket for the receiver. Afterwards that thread creates a child-parent relationship between the receiver and caller sockets. Next, the thread tries to establish a connection with the receiver. If connection with the receiver fails, then the thread sends an appropriate error message explaining the cause for which the connection has failed, then it closes the connection and cleans up that connection from the connection list. If the thread successfully establishes a connection with the receiver then the thread handover the control to the connection socket.

The connection socket then waits for a confirmation signal which tells it whether the receiver hanged-up the call or received the connection or time has expired for the user to receive the call. The Expiration time is set to eight seconds that means if the receiver does not receive a call within eight seconds then the call will be counted as a missed call. If time expires for the receiver then a 'receiver time expired' message arrives from the receiver side to the Server side. This message is then passed to the caller end. After that Server closes the connection and cleans up that connection from the connection list.

If receiver receives the call then the receiver and caller can start their conversation. When either of the party, that means either the receiver or caller disconnects the call, the conversation session ends. After that Server closes the connection and cleans up that connection from the connection list.

If receiver hang-up the call when the call arrives, then a 'user hang-up' message arrives to the Server end and then this message is passed to the caller end. After that Server closes the connection and cleans up that connection from the connection list.

4.5 Summary

The whole system was design and implemented considering all features of existing PABX system in reality. The PABX Server acts as a Bridge between the PABX subscribers and the PSTN subscribers. When an internal VOIP based PABX System subscriber dials a number this number is sent to the PABX Server. PABX Server then recognizes whether the call is destined for a PSTN subscriber or for another PABX subscriber. Then a multithreaded approach is taken by the PABX Server to provide multiple users the same services. The way, the server has been implemented it works as a Concurrent Server rather than an Iterative Server. The whole system serves its subscribers more elegant and efficient services than ever.

Chapter – 5
Cost Analysis

5.1 Introduction

From the beginning of each technology, technologists are trying to minimize the cost as far as possible so that every people can get the benefit of it. Though PABX system is integral part of business as well as to official organization, it is a matter of regret is that the investors have to invest a huge amount for setting up a fully workable PABX workstation. The target of this project was to minimize the cost as far as possible and it has successfully reached to its goal implementing VOIP in existing network.

5.2 Cost analysis

Since this project's vision was about to minimize the current local market cost of PABX exchange, it has successfully reached to its goal. Before going to the analysis part of cost consumption, some overview of PABX exchange on local market are given below:

Table 5.1 Four different types of PABX exchage's Installation expendture

PABX		16 line	24 line	32 line	500line
Input/Extension		6+16	8+24	8+32	35~40 Usually 80
Initial Cost		Tk – 32000	Tk – 45000	Tk – 110,000	16 lac
Extension Card	Tk. 25000 (additional 16 lines)				
Disa Card	Tk. 15000~60000 (at least one)	15000	20000~25000	35000~40000	60000
Surge Protector	Tk. 2000 (per 10 ports)	4000	6000	8000	1 lac
MDF	Tk. 1000 (per10 ports)	2000	3000	4000	50000
CO for 500 line M/C	Tk. 25000 (Only 500 lines)				
Total Initial Cost		53,000	74000~79000	1 lac 57000 ~ 1 lac 62000	18 lac 10000 + cost of COs

There are several types of PABX exchange are now in local market based on the number of local user under that particular exchange. On the table, there four different types of PABX exchange's expenditures are shown.

For 16 lines PABX system like PANASONIC KX-TA616, it is shown that initially it costs 32000 tk, Later it requires about 53000 tk in total to build a full workable PABX system excluding wiring cost, only to serve 16 people. To extend the user capacity, an Extension Slot

is needed to be installed which costs 25000 tk each. In comparison to this, VOIP based PABX exchange costs so little. For implementing this project package, it has to be installed in the server of the existing network. VOIP based PABX system can support up to 10000 users to remain PC operating system stable. There will be no additional cost for extension card, surge protector or MDF. The facilities of Disa Card are already impacted to this project. Therefore, no cost for Disa Card will be required.

24 lines PABX system as PANASONIC KX-TA308 initially costs 45000 tk, To make it fully capable to work, it requires about 74000 to 79000 tk for serving 24 users. To extend the user capacity an Extension Slot is needed to be installed which costs a lot. VOIP based PABX exchange costs so little comparatively. Since this VOIP based PABX system can support up to 10000 users to remain PC operating system stable, there would be no additional cost for extension card as well as surge protector or MDF. As the facilities of Disa Card are already impacted to this project, no cost for Disa Card will be required.

32 lines PABX system such as PANASONIC KX-TD232 initially cost 1,10,000 tk and to build a fully workable PABX work station, it requires about 1 lac 57000 tk to 1 lac 62000 tk only to provide services into 32 peoples only. An Extension Slot is needed to be installed to provide among more than 32 users. VOIP based PABX exchange is the right choice in this regard since this project package only has to be installed in the server of the existing network. no additional cost for extension card, surge protector or MDF will be needed since VOIP based PABX system can support up to 10000 users to remain PC operating system stable in the same time. No cost for Disa Card will also be required since Disa Card's facilities are already impacted to this project.

500 users supported PABX system is also in local market like PANASONIC KX-TD500 which costs 16 lacs primarily. It requires totally 18 lacs 10000 tk to work as a fully performed workstation. For extending the number of user more than 80, A CO card is needed to be installed which costs 25000 tk each. CO card basically adds only one input line.

VOIP based PABX will be a revolution in PABX local market as it costs near about to none for existing network. It requires no additional hardware and this is done by computer interfacing only. If there is no gateway or server on existing network, only a PC will be needed with IPS (Instant Power System) facility which will cost at most 30,000 thousands tk. One of the most important facts is that this VOIP based PABX system can be easily upgraded with CDR (Cost Domain Routing) server facilities for which a VOIP based PABX exchange owner need not to provide addition 10000 tk to 20000 tk monthly to third party CDR owner for calculating bills.

5.3 Summary

Network topology is now a demand of advanced communication world and most of the organization has already implemented LAN or WAN system. Utilizing such network facility, VOIP based PABX can easily be implemented in so little amount of cost. Users of such PABX system do not have to worry since it is capable to provide services among 10000 users and number of user can be extended in need without any cost. In a word, VOIP based PABX exchange is a revolution in terms of cost effectiveness.

Chapter – 6
Discussions and Conclusions

6.1 Discussions

It requires a huge task to implement the whole project. During implementation a number of remarkable problems have been faced and have solved as well. Though these debugging sessions requires patience, it made a real joy after solution. Since there was shortage of time, it was done only for windows platform only. Hence, it became more depended on DirectX. Because of this dependency, the project has to configure very carefully in accordance with DirectX tolls requirements. Besides this, there is also some lacking which will be overcome in near future. Instead of this, VOIP based PABX system has reached a satisfactory state.

6.2 Problems and Solutions

The problems those were faced during the testing period as well as the solution of those are given below:

At the inaugural state of this project, it has to overcome some hurdles to take the decision of development considering the lowest consumption of implementation.

Since VOIP is recent technology in the advanced communication world, there are not available information resources regarding the fact. The work that has been done in this project is totally new and Different than other VOIP related works. Besides this, it was thought firstly that the switching technology would be implemented in VHDL supported FPGA chips with circuit switching technology to communicate with PSTN network. Later this idea was eliminated since circuit switching is not as efficient as packet switching. In addition to this, to communicate with PSTN technology with packet switching, only modem will be required. Hence, it eliminated a costly device like FPGA chips as well as implementation technology. To reach this final solution, a couple days have to be waited.

In local market, routers like CISCO, Quintum etc provide VOIP services installing CBX card but the matter of great regret that router providing companies do not provide the software for billing system management. As a result, they have to be connected with a CDR (Cost Domain Router) server to compute bills. Hence, they have to pay a TK.10000 to TK.20000 monthly to the CDR provider which is the extravaganza for VOIP related business. From the point of view of providing service, cost minimization is necessary for a profitable business.

This research work can solve such problem with efficient manner because the billing management system has been implemented with this VOIP based PABX system package. Therefore, our local VOIP service providers do not need to contact with CDR server owner anymore and then the extravaganza is eliminated.

At the very beginning of the design project a problem was occurred while a testing was occurred in PC to PC voice communication. Voice was transmitted in a simplex way that

means voice just can be sent from one side to another side, but from the other end PC can not reply.

Several checking system was applied to solve this problem. At first networked connectivity was checked to find out the fault, but after checking no error had been detected. Everything was perfect for network connection. Later on the logical error was checked by analyzing throughout the code. Still percentage of error was zero which implies the criteria that defined code was not responsible for the described problem. To find out the real problem, debugging session took around 10 days. After analyzing over software part, analysis on hardware part has been started. Then it is detected that the problem was hardware related. Basically, the problem was regarding DirectX. All the DirectX tools were checked properly and it was found that the input device such as microphone was not tested on another end. Whenever the microphone was tested with DirectX, it worked successfully as full duplex. Finally, it was found that the input device for voice sending must be DirectX tested.

At first iterative server implemented to design VOIP based PABX system and results a lot of problems in supporting multi users. Such as when one user send any packet to another user will be idle. Iterative server works maintaining TDMA system. That means at the same time, more then one user can not send their packets. As a PABX system is a one kind of telecommunication technology that deals with multi-users, TDMA suffers a lot because makes system idle as it has maintain time slot for each individual. That was a great problem in implementation field as the whole system will have to be inactive while TDMA gives time to dedicated link that has not any active call.

As initially implemented iterative server was not efficient enough, later on concurrent server was established. Because concurrent server is able to provide service for multi-user. More than one user can send packet at the same time. This concurrent server use multi-thread, for which, when any user try to connect through the server, server forward them to thread instantly. So, no user sits idle.

At first this research was decided to implement a universal server through which any operating system platform user can communicate with each other using VOIP based PABX software. To implement such server a standard message passing system was required. Unfortunately there are no standards regarding this. Organizations that provide VOIP services or softwares are not ready to disclose their own internal message passing procedures used in their applications.

As still universal server for such case is not found yet, this problem will not be solved until IEEE (Institute of Electrical and Electronics Engineers) and ISO (International Organization for standardization) do not standardized this fact.

Each telecommunication service providing company is using their own internal message packet passing system maintaining unique standard. As there is no universal message passing system, different companies are using different massages. That why this VOIP based exchange has already implemented a new custom made massaging system. As a matter of fact it is not possible that two different softwares can be used by caller and receiver at the

same time. For instance, a caller can not make call using SKYPE to a receiver who is using NET2PHONE.

Several hurdles have been faced at the implementation stage of PSTN connectivity. One of those was that only gateway could communicate with the PSTN network but the rest PABX subscribers could not communicate.

Initially, the connectivity was checked but it was found they were absolutely in good condition. Then the code portion of the project was checked through debugging process. It was tiresome but after a long while, it was found the voice data size to encapsulate in VOIP was of 20 milliseconds according to standard VOIP packet. The size was reshaped to 60 milliseconds. The problem became solved after doing that.

There was another problem that has occurred throughout PSTN support implementation. It was that some connection request was refused automatically.

It also was of coding related problem. A precise scanning process was done throughout the whole coding part. A logical problem was found out then and the problem is overcome.

At the stage of coding several thing was considered like more focus was on easy hardware interfacing rather than optimum coding. The soft PABX exchange eliminates large portion of hardware from existing PABX solution.

One lacking of this project is that this system currently supports windows platform only. It does not support Cross platform that means this system has not implemented yet to run in Windows platform as well as UNIX, LINUX because of shortage of time. Having been dependant only on Windows platform, it most of the voice communication is done using DirectX. One of the important facts is that DirectX 8.1 or higher version of it should have to be installed at least in client side.

This PABX exchange's PSTN input line numbers are depends on the number of PCI slots available on the motherboard.

6.3 Suggestion for Future Work

This PABX system should have to be implemented in Cross Platform, so that any operating system user can get the benefit of it. Besides, this developer can take necessary steps to make this exchange more independent on DirectX.

The limitation of PABX input line can also be solved by using upcoming multi RJ11supported modem.

Besides this, in future online billing database system can be implemented by which any VOIP extension user can be registered and check his personal account through internet.

6.4 Conclusion

Technology advances forward in every key of time. Different types of research are conducting in different types of technology but there is no remarkable progress in the field of PABX system. IP based PBX system is in reality in the advanced world and several types of researches regarding the fact is going on. On contrary, in respect of Bangladesh, there are no noteworthy researches have conducted. This VOIP based PABX system has added a new horizon in this vicinity.

VOIP is now rapidly immerging as a business transforming technology for enterprises because VOIP provide a simpler network design and supports several multimedia features. Enterprises already using VOIP are realizing the benefit from lower capital cost, reduced operating expenses and value added functionality compared with Legacy voice system. This research analyzes the current market rate of VOIP provider of Bangladesh. It also analyses the cost effectiveness of VOIP service in Bangladesh. Considering the cost analysis sections it can be said that this research work not only reduce the cost for technical support, it also includes a built-in billing system which is still not available for the VOIP provider of Bangladesh. On the design section this paper depicted a complete design format for a VOIP based PABX system.

Through there are some problems during implementing project works, those were solved step by step but there are also lacking due to have very short period of time. Instead of these, it is matter of relief that this project has reached to its desired goal according to vision of the project.

Billing management system gives this project a new dimension through which a number of problems regarding the fact have been solved and users are getting more reliable and user friendly billing environment throughout this package.

In essence, the unique and exiting features of this VOIP based PABX system will be revolution in local PABX industry in terms of cost and effectiveness.

Appendices

Appendix A

Source Codes for PABX Server

A.1 Accepting Connection Requests

```
int num;//i=m_gvSocket.size();
    list<NConnectionSocket*>::iterator it;

    it=m_gvSocket.insert(m_gvSocket.end(), new NConnectionSocket());
    Accept(*(*it));

//  m_gvSocket[i]->m_bCompleted=true;
    (*it)->m_bClosed=false;
    (*it)->res=res;
    (*it)->m_bConnected=false;
    (*it)->server=this;

    CSingleLock sLock(&res->m_mutex);
    sLock.Lock();
    (*it)->clNo=++res->m_iClient;
    sprintf(buffer, "Client %d has Connected!!!", (*it)->clNo);
    res->m_lbStatus.AddString(buffer);

    //sprintf(buffer, "Connected To Server. You are Client No %d",m_gvSocket[i]->clNo);
    //m_gvSocket[i]->Send(buffer, strlen(buffer));

    //get receiver extenson number
    buffer[(*it)->Receive(buffer, BUFLEN)]=0;
    sscanf(buffer,"%d",&num);
    // check whether num > total usernumber. if yes send rcvr message and close
    if(res->m_stUserTable[num].status)//if receiver is busy
        {
            sprintf(buffer, "_USER_BUSY_");
            (*it)->Send(buffer, strlen(buffer));
            (*it)->Close();
            m_gvSocket.erase(it);
        }
    else
        {//if receiver is not busy
            res->m_stUserTable[num].status=true;
            CString addr;
            UINT prt;
            (*it)->GetPeerName(addr, prt);
```

```
            (*it)->m_iRId=num;
            //find caller's ip address
            for(prt=0; prt < res->m_iTotalUser; prt++)
                {
                    if(addr == res->m_stUserTable[prt].address)
                        {
                            (*it)->m_iId=prt;
                            res->m_stUserTable[prt].status=true;
                            break;
                        }
                }
            (*it)->address=res->m_stUserTable[num].address;
            AfxBeginThread(ConnectToReceiver, &it);
        }

    sLock.Unlock();
```

A.2 Connecting With The Receiver

```
list<NConnectionSocket*>::iterator ptr= *((list<NConnectionSocket*>::iterator *)pParam);
    (*ptr)->receiver=new NReceiverSocket();
    (*ptr)->receiver->m_bFirstCheck=true;
    (*ptr)->receiver->iter=ptr;
    if((*ptr)->Connect((*ptr)->address, 5500))
        (*ptr)->receiver->parent=*ptr;
    else
        {//ERROR : check connection status
            CString str;
            (*ptr)->GetErrorCode(GetLastError(), str);
            if(str != "")
                {
                    (*ptr)->Send(str, str.GetLength());
                    (*ptr)->CloseConnection();
                }
        }
```

A.3 Negotiation While Connecting With The Receiver

```
buffer[length=Receive(buffer, BUFLEN)]=0;
    if(m_bFirstCheck)//only check for the first recieve
        {
            if(strcmp(buffer, "_USER_HANGUP_"))
                {
                    parent->m_bConnected=false;
                    //sprintf(buffer, "ACK_CLOSE");
```

```
                parent->Send(buffer, length);
                parent->CloseConnection();
            }
        else if(strcmp(buffer, "_USER_RECEIVED_"))
            {
                parent->start=clock();
                parent->m_bConnected=true;
                parent->Send(buffer, length);
            }
        else
            {//invalid command received
                parent->m_bConnected=false;
                sprintf(buffer, "Invalid Command!!!");
                parent->Send(buffer, strlen(buffer));
                parent->CloseConnection();
            }
        m_bFirstCheck=false;
    }
else
    parent->Send(buffer, length);
```

A.4 Close And Cleaning-up Connections

```
if(!m_bClosed)
    {
        if(m_bConnected)
            {
                int duration = (clock() - start) / CLOCKS_PER_SEC;
                sprintf(buffer, "Client %d (talk-time): %d seconds\n", duration,clNo);

                CSingleLock sLock(&res->m_mutex);
                sLock.Lock();
                res->m_lbStatus.AddString(buffer);
                sLock.Unlock();
            }

        m_bClosed=true;
        list<NConnectionSocket*>::iterator it=receiver->iter;
        receiver->Close();
        Close();
        CSingleLock sLock(&res->m_mutex);
        sLock.Lock();
        res->m_stUserTable[m_iId].status=false;
        res->m_stUserTable[m_iRId].status=false;
        sLock.Unlock();

        CSingleLock nLock(&server->m_mutex);
```

```
            nLock.Lock();
            server->m_gvSocket.erase(it);
            nLock.Unlock();
```

A.4 Getting Error Messages

```
str="";
    switch( errorCode )
        {
            case WSAENETDOWN:
                str="The network is down.";
                break;
            case WSAEADDRINUSE:
                str="The specified address is already in use.";
                break;
            case WSAEADDRNOTAVAIL:
                str="The specified address is not available from the local machine.";
                break;
            case WSAEAFNOSUPPORT:
                str="Addresses in the specified family cannot be used with this socket.";
                break;
            case WSAECONNREFUSED:
                str="The attempt to connect was forcefully rejected.";
                break;
            case WSAEDESTADDRREQ:
                str="A destination address is required.";
                break;
            case WSAEFAULT:
                str="The lpSockAddrLen argument is incorrect.";
                break;
            case WSAEINVAL:
                str="The socket is already bound to an address.";
                break;
            case WSAEISCONN:
                str="The socket is already connected.";
                break;
            case WSAEMFILE:
                str="No more file descriptors are available.";
                break;
            case WSAENETUNREACH:
                str="The network cannot be reached from this host at this time.";
                break;
            case WSAENOBUFS:
                str="No buffer space is available. The socket cannot be connected.";
                break;
            case WSAENOTCONN:
```

```
            str="The socket is not connected.";
            break;
        case WSAENOTSOCK:
            "The descriptor is a file, not a socket.";
            break;
        case WSAETIMEDOUT:
            str="The attempt to connect timed out without establishing a connection.";
            break;
        case WSAEWOULDBLOCK :
            str="The socket is marked as nonblocking and the connection cannot be
completed immediately.";
            break;
        //default:
        //    str.Format("OnConnect error: %d", errorCode);
    }//switch( nErrorCode )
```

Appendix B

Source Codes for Dialing and Receiving PSTN Calls

```
//  FUNCTION: BOOL Create()
//
//  PURPOSE: Initializes TAPI
//
BOOL CTapiConnection::Create(char *szPhoneNumber)
{
   long lReturn;

   // If we're already initialized, then initialization succeeds.
   if (m_hLineApp)
      return TRUE;

   // If we're in the middle of initializing, then fail, we're not done.
   if (m_bInitializing)
      return FALSE;

   m_bInitializing = TRUE;

   // Initialize TAPI
   do
   {
      lReturn = ::lineInitialize(&m_hLineApp,
         AfxGetInstanceHandle(),
         lineCallbackFunc,
         "DialIt",
         &m_dwNumDevs);

      if (m_dwNumDevs == 0)
      {
         AfxMessageBox("There are no telephony devices installed.");
         m_bInitializing = FALSE;
         return FALSE;
      }

      if (HandleLineErr(lReturn))
         continue;
      else
      {
         OutputDebugString("lineInitialize unhandled error\n");
         m_bInitializing = FALSE;
         return FALSE;
      }
```

```
    }
    while(lReturn != SUCCESS);

    OutputDebugString("Tapi initialized.\n");

    // If the user furnished a phone number copy it over.
    if (szPhoneNumber)
        strcpy(m_szPhoneNumber, szPhoneNumber);

    m_bInitializing = FALSE;
    return TRUE;
}

//
//  FUNCTION: DialCall()
//
//  PURPOSE: Get a number from the user and dial it.
//
BOOL CTapiConnection::DialCall(char *szPhoneNumber)
{
    long lReturn;
    LPLINEDEVCAPS lpLineDevCaps = NULL;

    if (m_bTapiInUse)
    {
        AfxMessageBox("A call is already being handled.");
        return FALSE;
    }

    // Make sure TAPI is initialized properly
    if (!m_hLineApp)
    {
        if (!Create(NULL))
            return FALSE;
    }

    // If there are no line devices installed on the machine, quit.
    if (m_dwNumDevs < 1)
        return FALSE;

    // We now have a call active.  Prevent future calls.
    m_bTapiInUse = TRUE;

    // Get a phone number from the user.
    if (szPhoneNumber == (char *)NULL)
    {
        if (m_szPhoneNumber == (char *)NULL)
```

```
        {
            HangupCall();
            goto DeleteBuffers;
        }
    }
    else
        strcpy(m_szPhoneNumber, szPhoneNumber);

    // Get the line to use
    lpLineDevCaps = GetDeviceLine(&m_dwAPIVersion, lpLineDevCaps);

    // Need to check the DevCaps to make sure this line is usable.
    if (lpLineDevCaps == NULL)
    {
        OutputDebugString("Error on Requested line\n");
        goto DeleteBuffers;
    }

    if (!(lpLineDevCaps->dwBearerModes & LINEBEARERMODE_VOICE ))
    {
        AfxMessageBox("The selected line doesn't support VOICE capabilities");
        goto DeleteBuffers;
    }

    // Does this line have the capability to make calls?
    if (!(lpLineDevCaps->dwLineFeatures & LINEFEATURE_MAKECALL))
    {
        AfxMessageBox("The selected line doesn't support MAKECALL capabilities");
        goto DeleteBuffers;
    }

    // Does this line have the capability for interactive voice?
    if (!(lpLineDevCaps->dwMediaModes & LINEMEDIAMODE_INTERACTIVEVOICE))
    {
        AfxMessageBox("The selected line doesn't support INTERACTIVE VOICE
capabilities");
        goto DeleteBuffers;
    }

    // Open the Line for an outgoing call.
    do
    {
        lReturn = ::lineOpen(m_hLineApp, m_dwDeviceID, &m_hLine,
            m_dwAPIVersion, 0, 0,
            LINECALLPRIVILEGE_NONE, 0, 0);
```

```
      if((lReturn == LINEERR_ALLOCATED)||(lReturn == LINEERR_RESOURCEUNAVAIL))
      {
         HangupCall();
         OutputDebugString("Line is already in use by a non-TAPI application "
            "or by another TAPI Service Provider.\n");
         goto DeleteBuffers;
      }

      if (HandleLineErr(lReturn))
         continue;
      else
      {
         OutputDebugString("Unable to Use Line\n");
         HangupCall();
         goto DeleteBuffers;
      }
   }
   while(lReturn != SUCCESS);

   // Start dialing the number
   if( MakeTheCall(lpLineDevCaps, m_szPhoneNumber))
      OutputDebugString("lineMakeCall succeeded.\n");
   else
   {
      OutputDebugString("lineMakeCall failed.\n");
      HangupCall();
   }

DeleteBuffers:

   if (lpLineDevCaps)
      LocalFree(lpLineDevCaps);

   return m_bTapiInUse;
}

//
//  FUNCTION: void GetDeviceLine()
//
//  PURPOSE: Gets the first available line device.
//
//
LPLINEDEVCAPS CTapiConnection::GetDeviceLine(DWORD *pdwAPIVersion,
LPLINEDEVCAPS lpLineDevCaps)
{
   DWORD dwDeviceID;
```

```
   char szLineUnavail[] = "Line Unavailable";
   char szLineUnnamed[] = "Line Unnamed";
   char szLineNameEmpty[] = "Line Name is Empty";
   LPSTR lpszLineName;
   long lReturn;
   char buf[MAX_PATH];
   LINEEXTENSIONID lineExtID;
   BOOL bDone = FALSE;

   for (dwDeviceID = 0; (dwDeviceID < m_dwNumDevs) && !bDone; dwDeviceID ++)
   {

      lReturn = ::lineNegotiateAPIVersion(m_hLineApp, dwDeviceID,
         EARLY_TAPI_VERSION, SAMPLE_TAPI_VERSION,
         pdwAPIVersion, &lineExtID);

      if ((HandleLineErr(lReturn))&&(*pdwAPIVersion))
      {
         lpLineDevCaps = MylineGetDevCaps(lpLineDevCaps, dwDeviceID,
*pdwAPIVersion);

         if ((lpLineDevCaps -> dwLineNameSize) &&
            (lpLineDevCaps -> dwLineNameOffset) &&
            (lpLineDevCaps -> dwStringFormat == STRINGFORMAT_ASCII))
         {
            // This is the name of the device.
            lpszLineName = ((char *) lpLineDevCaps) +
               lpLineDevCaps -> dwLineNameOffset;
            sprintf(buf, "Name of device is: %s\n", lpszLineName);
            OutputDebugString(buf);

         }
         else  // DevCaps doesn't have a valid line name.  Unnamed.
            lpszLineName = szLineUnnamed;
      }
      else  // Couldn't NegotiateAPIVersion.  Line is unavail.
         lpszLineName = szLineUnavail;

      // If this line is usable and we don't have a default initial
      // line yet, make this the initial line.
      if ((lpszLineName != szLineUnavail) &&
         (lReturn == SUCCESS ))
      {
         m_dwDeviceID = dwDeviceID;
         bDone = TRUE;
      }
      else
```

```
        m_dwDeviceID = MAXDWORD;
    }
    return (lpLineDevCaps);
}

//
//  FUNCTION: MylineGetDevCaps(LPLINEDEVCAPS, DWORD , DWORD)
//
//  PURPOSE: Gets a LINEDEVCAPS structure for the specified line.
//
//  COMMENTS:
//
//    This function is a wrapper around lineGetDevCaps to make it easy
//    to handle the variable sized structure and any errors received.
//
//    The returned structure has been allocated with LocalAlloc,
//    so LocalFree has to be called on it when you're finished with it,
//    or there will be a memory leak.
//
//    Similarly, if a lpLineDevCaps structure is passed in, it *must*
//    have been allocated with LocalAlloc and it could potentially be
//    LocalFree()d.
//
//    If lpLineDevCaps == NULL, then a new structure is allocated.  It is
//    normal to pass in NULL for this parameter unless you want to use a
//    lpLineDevCaps that has been returned by a previous I_lineGetDevCaps
//    call.
//
//

LPLINEDEVCAPS CTapiConnection::MylineGetDevCaps(
    LPLINEDEVCAPS lpLineDevCaps,
    DWORD dwDeviceID, DWORD dwAPIVersion)
{
    // Allocate enough space for the structure plus 1024.
    size_t sizeofLineDevCaps = sizeof(LINEDEVCAPS) + 1024;
    long lReturn;

    // Continue this loop until the structure is big enough.
    while(TRUE)
    {
        // Make sure the buffer exists, is valid and big enough.
        lpLineDevCaps =
            (LPLINEDEVCAPS) CheckAndReAllocBuffer(
                (LPVOID) lpLineDevCaps, // Pointer to existing buffer, if any
                sizeofLineDevCaps);      // Minimum size the buffer should be
```

```
      if (lpLineDevCaps == NULL)
         return NULL;

      // Make the call to fill the structure.
      do
      {
         lReturn =
            ::lineGetDevCaps(m_hLineApp,
               dwDeviceID, dwAPIVersion, 0, lpLineDevCaps);

         if (HandleLineErr(lReturn))
            continue;
         else
         {
            OutputDebugString("lineGetDevCaps unhandled error/n");
            LocalFree(lpLineDevCaps);
            return NULL;
         }
      }
      while (lReturn != SUCCESS);

      // If the buffer was big enough, then succeed.
      if ((lpLineDevCaps -> dwNeededSize) <= (lpLineDevCaps -> dwTotalSize))
         return lpLineDevCaps;

      // Buffer wasn't big enough.  Make it bigger and try again.
      sizeofLineDevCaps = lpLineDevCaps -> dwNeededSize;
   }
}

//
//  FUNCTION: LPVOID CheckAndReAllocBuffer(LPVOID, size_t, LPCSTR)
//
//  PURPOSE: Checks and ReAllocates a buffer if necessary.
//
LPVOID CTapiConnection::CheckAndReAllocBuffer(LPVOID lpBuffer, size_t
sizeBufferMinimum)
{
   size_t sizeBuffer;

   if (lpBuffer == NULL)  // Allocate the buffer if necessary.
   {
      sizeBuffer = sizeBufferMinimum;
      lpBuffer = (LPVOID) LocalAlloc (LPTR, sizeBuffer);

      if (lpBuffer == NULL)
      {
```

```
            OutputDebugString("LocalAlloc failed in CheckAndReAllocBuffer./n");
            return NULL;
        }
    }
    else // If the structure already exists, make sure its good.
    {
        sizeBuffer = LocalSize((HLOCAL) lpBuffer);

        if (sizeBuffer == 0) // Bad pointer?
        {
            OutputDebugString("LocalSize returned 0 in CheckAndReAllocBuffer/n");
            return NULL;
        }

        // Was the buffer big enough for the structure?
        if (sizeBuffer < sizeBufferMinimum)
        {
            OutputDebugString("Reallocating structure\n");
            LocalFree(lpBuffer);
            return CheckAndReAllocBuffer(NULL, sizeBufferMinimum);
        }
    }

    memset(lpBuffer, 0, sizeBuffer);
    ((LPVARSTRING) lpBuffer ) -> dwTotalSize = (DWORD) sizeBuffer;
    return lpBuffer;
}


//
//  FUNCTION: MylineGetAddressCaps(LPLINEADDRESSCAPS, ..)
//
//  PURPOSE: Return a LINEADDRESSCAPS structure for the specified line.
//
LPLINEADDRESSCAPS CTapiConnection::MylineGetAddressCaps (
    LPLINEADDRESSCAPS lpLineAddressCaps,
    DWORD dwDeviceID, DWORD dwAddressID,
    DWORD dwAPIVersion, DWORD dwExtVersion)
{
    size_t sizeofLineAddressCaps = sizeof(LINEADDRESSCAPS) + 1024;
    long lReturn;

    // Continue this loop until the structure is big enough.
    while(TRUE)
    {
        // Make sure the buffer exists, is valid and big enough.
        lpLineAddressCaps =
```

```
        (LPLINEADDRESSCAPS) CheckAndReAllocBuffer(
            (LPVOID) lpLineAddressCaps,
            sizeofLineAddressCaps);

        if (lpLineAddressCaps == NULL)
            return NULL;

        // Make the call to fill the structure.
        do
        {
            lReturn =
                ::lineGetAddressCaps(m_hLineApp,
                    dwDeviceID, dwAddressID, dwAPIVersion, dwExtVersion,
                    lpLineAddressCaps);

            if (HandleLineErr(lReturn))
                continue;
            else
            {
                OutputDebugString("lineGetAddressCaps unhandled error\n");
                LocalFree(lpLineAddressCaps);
                return NULL;
            }
        }
        while (lReturn != SUCCESS);

        // If the buffer was big enough, then succeed.
        if ((lpLineAddressCaps -> dwNeededSize) <=
            (lpLineAddressCaps -> dwTotalSize))
        {
            return lpLineAddressCaps;
        }

        // Buffer wasn't big enough.  Make it bigger and try again.
        sizeofLineAddressCaps = lpLineAddressCaps -> dwNeededSize;
    }
}

//
//  FUNCTION: MakeTheCall(LPLINEDEVCAPS, LPCSTR)
//
//  PURPOSE: Dials the call
//

BOOL CTapiConnection::MakeTheCall(LPLINEDEVCAPS lpLineDevCaps, LPCTSTR
lpszAddress)
{
```

```
LPLINECALLPARAMS  lpCallParams = NULL;
LPLINEADDRESSCAPS lpAddressCaps = NULL;
long lReturn;
BOOL bFirstDial = TRUE;

// Get the capabilities for the line device we're going to use.
lpAddressCaps = MylineGetAddressCaps(lpAddressCaps,
   m_dwDeviceID, 0, m_dwAPIVersion, 0);
if (lpAddressCaps == NULL)
   return FALSE;

// Setup our CallParams.
lpCallParams = CreateCallParams (lpCallParams, lpszAddress);
if (lpCallParams == NULL)
   return FALSE;

do
{
   if (bFirstDial)
      //lReturn = ::lineMakeCall(m_hLine, &m_hCall, lpszAddress, 0, lpCallParams);
      lReturn = WaitForReply( ::lineMakeCall(m_hLine, &m_hCall, lpszAddress,
               0, lpCallParams) );
   else
      lReturn = WaitForReply(::lineDial(m_hCall, lpszAddress, 0));

   if (lReturn == WAITERR_WAITABORTED)
   {
       OutputDebugString("While Dialing, WaitForReply aborted.\n");
       goto errExit;
   }

   if (HandleLineErr(lReturn))
      continue;
   else
   {
      if (bFirstDial)
         OutputDebugString("lineMakeCall unhandled error\n");
      else
         OutputDebugString("lineDial unhandled error\n");

      goto errExit;
   }
}
while (lReturn != SUCCESS);

bFirstDial = FALSE;
```

```
    if (lpCallParams)
        LocalFree(lpCallParams);
    if (lpAddressCaps)
        LocalFree(lpAddressCaps);

    return TRUE;

  errExit:
    if (lpCallParams)
        LocalFree(lpCallParams);
    if (lpAddressCaps)
        LocalFree(lpAddressCaps);

    AfxMessageBox("Dial failed.");

    return FALSE;
}


//
//  FUNCTION: CreateCallParams(LPLINECALLPARAMS, LPCSTR)
//
//  PURPOSE: Allocates and fills a LINECALLPARAMS structure
//
//
LPLINECALLPARAMS CTapiConnection::CreateCallParams (
    LPLINECALLPARAMS lpCallParams, LPCSTR lpszDisplayableAddress)
{
    size_t sizeDisplayableAddress;

    if (lpszDisplayableAddress == NULL)
        lpszDisplayableAddress = "";

    sizeDisplayableAddress = strlen(lpszDisplayableAddress) + 1;

    lpCallParams = (LPLINECALLPARAMS) CheckAndReAllocBuffer(
        (LPVOID) lpCallParams,
        sizeof(LINECALLPARAMS) + sizeDisplayableAddress);

    if (lpCallParams == NULL)
        return NULL;

    // This is where we configure the line.
    lpCallParams -> dwBearerMode = LINEBEARERMODE_VOICE;
    lpCallParams -> dwMediaMode  = LINEMEDIAMODE_INTERACTIVEVOICE;

    // This specifies that we want to use only IDLE calls and
```

```
    // don't want to cut into a call that might not be IDLE (ie, in use).
    lpCallParams -> dwCallParamFlags = LINECALLPARAMFLAGS_IDLE;

    // if there are multiple addresses on line, use first anyway.
    // It will take a more complex application than a simple tty app
    // to use multiple addresses on a line anyway.
    lpCallParams -> dwAddressMode = LINEADDRESSMODE_ADDRESSID;

    // Address we are dialing.
    lpCallParams -> dwDisplayableAddressOffset = sizeof(LINECALLPARAMS);
    lpCallParams -> dwDisplayableAddressSize = sizeDisplayableAddress;
    strcpy((LPSTR)lpCallParams + sizeof(LINECALLPARAMS),
          lpszDisplayableAddress);

    return lpCallParams;
}

//
//  FUNCTION: long WaitForReply(long)
//
//  PURPOSE: Resynchronize by waiting for a LINE_REPLY
//
//  PARAMETERS:
//    lRequestID - The asynchronous request ID that we're
//                 on a LINE_REPLY for.
//
//  RETURN VALUE:
//    - 0 if LINE_REPLY responded with a success.
//    - LINEERR constant if LINE_REPLY responded with a LINEERR
//    - 1 if the line was shut down before LINE_REPLY is received.
//
//  COMMENTS:
//
//    This function allows us to resynchronize an asynchronous
//    TAPI line call by waiting for the LINE_REPLY message.  It
//    waits until a LINE_REPLY is received or the line is shut down.
//
//    Note that this could cause re-entrancy problems as
//    well as mess with any message preprocessing that might
//    occur on this thread (such as TranslateAccelerator).
//
//    This function should to be called from the thread that did
//    lineInitialize, or the PeekMessage is on the wrong thread
//    and the synchronization is not guaranteed to work.  Also note
//    that if another PeekMessage loop is entered while waiting,
//    this could also cause synchronization problems.
//
```

```
//    One more note.  This function can potentially be re-entered
//    if the call is dropped for any reason while waiting.  If this
//    happens, just drop out and assume the wait has been canceled.
//    This is signaled by setting b Reentered to FALSE when the function
//    is entered and TRUE when it is left.  If b Reentered is ever TRUE
//    during the function, then the function was re-entered.
//
//    This function times out and returns WAITERR_WAITTIMEDOUT
//    after WAITTIMEOUT milliseconds have elapsed.
//
//


long CTapiConnection::WaitForReply (long lRequestID)
{
   static BOOL bReentered;
   bReentered = FALSE;

   if (lRequestID > SUCCESS)
   {
      MSG msg;
      DWORD dwTimeStarted;

      m_bReplyReceived = FALSE;
      m_dwRequestedID = (DWORD) lRequestID;

      // Initializing this just in case there is a bug
      // that sets m_bReplyRecieved without setting the reply value.
      m_lAsyncReply = LINEERR_OPERATIONFAILED;

      dwTimeStarted = GetTickCount();

      while(!m_bReplyReceived)
      {
         if (PeekMessage(&msg, 0, 0, 0, PM_REMOVE))
         {
            TranslateMessage(&msg);
            DispatchMessage(&msg);
         }

         // This should only occur if the line is shut down while waiting.
         if ((m_hCall != NULL) &&(!m_bTapiInUse || bReentered))
         {
            bReentered = TRUE;
            return WAITERR_WAITABORTED;
         }
```

```
            // Its a really bad idea to timeout a wait for a LINE_REPLY.
            // If we are execting a LINE_REPLY, we should wait till we get
            // it; it might take a long time to dial (for example).

            // If 5 seconds go by without a reply, it might be a good idea
            // to display a dialog box to tell the user that a
            // wait is in progress and to give the user the capability to
            // abort the wait.
        }

        bReentered = TRUE;
        return m_lAsyncReply;
    }

    bReentered = TRUE;
    return lRequestID;
}

//
//  FUNCTION: lineCallbackFunc(..)
//
//  PURPOSE: Receive asynchronous TAPI events
//
void CALLBACK CTapiConnection::lineCallbackFunc(
    DWORD dwDevice, DWORD dwMsg, DWORD dwCallbackInstance,
    DWORD dwParam1, DWORD dwParam2, DWORD dwParam3)
{

    // Handle the line messages.
    switch(dwMsg)
    {
        case LINE_CALLSTATE:
            MyThis->HandleLineCallState(dwDevice, dwMsg, dwCallbackInstance, dwParam1,
dwParam2,
                dwParam3);
            break;

        case LINE_CLOSE:
            // Line has been shut down.
            ASSERT(MyThis);
            MyThis->m_hLine = NULL;
            MyThis->m_hCall = NULL;
            MyThis->HangupCall(); // all handles invalidated by this time
            break;

        case LINE_REPLY:
            if ((long) dwParam2 != SUCCESS)
```

```
            OutputDebugString("LINE_REPLY error\n");
         else
            OutputDebugString("LINE_REPLY: successfully replied.\n");
         break;

      case LINE_CREATE:
         ASSERT(MyThis);
         if (MyThis->m_dwNumDevs <= dwParam1)
            MyThis->m_dwNumDevs = dwParam1+1;
         break;

      default:
         OutputDebugString("lineCallbackFunc message ignored\n");
         break;
   }
   return;
}

//
//  FUNCTION: HandleLineCallState(..)
//
//  PURPOSE: Handle LINE_CALLSTATE asynchronous messages.
//

void CTapiConnection::HandleLineCallState(
   DWORD dwDevice, DWORD dwMessage, DWORD dwCallbackInstance,
   DWORD dwParam1, DWORD dwParam2, DWORD dwParam3)
{

   // Error if this CALLSTATE doesn't apply to our call in progress.
   if ((HCALL) dwDevice != m_hCall)
   {
      OutputDebugString("LINE_CALLSTATE: Unknown device ID ");
      return;
   }


   // dwParam1 is the specific CALLSTATE change that is occurring.
   switch (dwParam1)
   {
      case LINECALLSTATE_DIALTONE:
         OutputDebugString("Dial Tone\n");
         break;

      case LINECALLSTATE_DIALING:
         OutputDebugString("Dialing\n");
         break;
```

```
case LINECALLSTATE_PROCEEDING:
   OutputDebugString("Proceeding\n");
   break;

case LINECALLSTATE_RINGBACK:
   OutputDebugString("RingBack\n");
   break;

case LINECALLSTATE_BUSY:
   OutputDebugString("Line busy, shutting down\n");
   HangupCall();
   break;

case LINECALLSTATE_IDLE:
   OutputDebugString("Line idle\n");
   HangupCall();
   break;

case LINECALLSTATE_SPECIALINFO:
   OutputDebugString("Special Info, probably couldn't dial number\n");
   HangupCall();
   break;

case LINECALLSTATE_DISCONNECTED:
{
   LPSTR pszReasonDisconnected;

   switch (dwParam2)
   {
      case LINEDISCONNECTMODE_NORMAL:
         pszReasonDisconnected = "Remote Party Disconnected";
         break;

      case LINEDISCONNECTMODE_UNKNOWN:
         pszReasonDisconnected = "Disconnected: Unknown reason";
         break;

      case LINEDISCONNECTMODE_REJECT:
         pszReasonDisconnected = "Remote Party rejected call";
         break;

      case LINEDISCONNECTMODE_PICKUP:
         pszReasonDisconnected =
            "Disconnected: Local phone picked up";
         break;
```

```
    case LINEDISCONNECTMODE_FORWARDED:
       pszReasonDisconnected = "Disconnected: Forwarded";
       break;

    case LINEDISCONNECTMODE_BUSY:
       pszReasonDisconnected = "Disconnected: Busy";
       break;

    case LINEDISCONNECTMODE_NOANSWER:
       pszReasonDisconnected = "Disconnected: No Answer";
       break;

    case LINEDISCONNECTMODE_BADADDRESS:
       pszReasonDisconnected = "Disconnected: Bad Address";
       break;

    case LINEDISCONNECTMODE_UNREACHABLE:
       pszReasonDisconnected = "Disconnected: Unreachable";
       break;

    case LINEDISCONNECTMODE_CONGESTION:
       pszReasonDisconnected = "Disconnected: Congestion";
       break;

    case LINEDISCONNECTMODE_INCOMPATIBLE:
       pszReasonDisconnected = "Disconnected: Incompatible";
       break;

    case LINEDISCONNECTMODE_UNAVAIL:
       pszReasonDisconnected = "Disconnected: Unavail";
       break;

    case LINEDISCONNECTMODE_NODIALTONE:
       pszReasonDisconnected = "Disconnected: No Dial Tone";
       break;

    default:
       pszReasonDisconnected =
          "Disconnected: LINECALLSTATE; Bad Reason";
       break;

}

OutputDebugString(pszReasonDisconnected);
OutputDebugString("\n");
HangupCall();
break;
```

```
        }

        case LINECALLSTATE_CONNECTED:  // CONNECTED!!!
            OutputDebugString("Connected!\n");
            break;

        default:
            OutputDebugString("Unhandled LINECALLSTATE message\n");
            break;
    }
}

//
//  FUNCTION: HandleLineErr(long)
//
//  PURPOSE: Handle several of the standard LINEERR errors
//
BOOL CTapiConnection::HandleLineErr(long lLineErr)
{
    BOOL bRet = FALSE;

    // lLineErr is really an async request ID, not an error.
    if (lLineErr > SUCCESS)
        return bRet;

    // All we do is dispatch the correct error handler.
    switch(lLineErr)
    {
        case SUCCESS:
            bRet = TRUE;
            break;

        case LINEERR_INVALCARD:
        case LINEERR_INVALLOCATION:
        case LINEERR_INIFILECORRUPT:
            OutputDebugString("The values in the INI file are invalid.\n");
            break;

        case LINEERR_NODRIVER:
            OutputDebugString("There is a problem with your Telephony device driver.\n");
            break;

        case LINEERR_REINIT:
            ShutdownTAPI();
            break;

        case LINEERR_NOMULTIPLEINSTANCE:
```

```
            OutputDebugString("Remove one of your copies of your Telephony driver.\n");
            break;

        case LINEERR_NOMEM:
            OutputDebugString("Out of memory. Cancelling action.\n");
            break;

        case LINEERR_OPERATIONFAILED:
            OutputDebugString("The TAPI operation failed.\n");
            break;

        case LINEERR_RESOURCEUNAVAIL:
            OutputDebugString("A TAPI resource is unavailable at this time.\n");
            break;

        // Unhandled errors fail.
        default:
            break;
    }
    return bRet;
}


//
//  FUNCTION: BOOL HangupCall()
//
//  PURPOSE: Hangup the call in progress if it exists.
//
BOOL CTapiConnection::HangupCall()
{
    LPLINECALLSTATUS pLineCallStatus = NULL;
    long lReturn;

    // Prevent HangupCall re-entrancy problems.
    if (m_bStoppingCall)
        return TRUE;

    // If Tapi is not being used right now, then the call is hung up.
    if (!m_bTapiInUse)
        return TRUE;

    m_bStoppingCall = TRUE;

    OutputDebugString("Stopping Call in progress\n");

    // If there is a call in progress, drop and deallocate it.
    if (m_hCall)
```

```
{
   pLineCallStatus = (LPLINECALLSTATUS)malloc(sizeof(LINECALLSTATUS));

   if (!pLineCallStatus)
   {
      ShutdownTAPI();
      m_bStoppingCall = FALSE;
      return FALSE;
   }

   lReturn = ::lineGetCallStatus(m_hCall, pLineCallStatus);

   // Only drop the call when the line is not IDLE.
   if (!((pLineCallStatus -> dwCallState) & LINECALLSTATE_IDLE))
   {
      WaitForReply(lineDrop(m_hCall, NULL, 0));
      OutputDebugString("Call Dropped.\n");
   }

   // The call is now idle.  Deallocate it!
   do
   {
      lReturn = ::lineDeallocateCall(m_hCall);
      if (HandleLineErr(lReturn))
         continue;
      else
      {
         OutputDebugString("lineDeallocateCall unhandled error\n");
         break;
      }
   }
   while(lReturn != SUCCESS);

   OutputDebugString("Call Deallocated.\n");

}

// if we have a line open, close it.
if (m_hLine)
{
  lReturn = ::lineClose(m_hLine);
  if (!HandleLineErr(lReturn))
     OutputDebugString("lineClose unhandled error\n");

   OutputDebugString("Line Closed.\n");
}
```

```
    // Clean up.
    m_hCall = NULL;
    m_hLine = NULL;
    m_bTapiInUse = FALSE;
    m_bStoppingCall = FALSE; // allow HangupCall to be called again.

    // Need to free buffer returned from lineGetCallStatus
    if (pLineCallStatus)
        free(pLineCallStatus);

    OutputDebugString("Call stopped\n");
    return TRUE;
}

//
//  FUNCTION: BOOL ShutdownTAPI()
//
//  PURPOSE: Shuts down all use of TAPI
//
BOOL CTapiConnection::ShutdownTAPI()
{
    long lReturn;

    // If we aren't initialized, then Shutdown is unnecessary.
    if (m_hLineApp == NULL)
        return TRUE;

    // Prevent ShutdownTAPI re-entrancy problems.
    if (m_bShuttingDown)
        return TRUE;

    m_bShuttingDown = TRUE;

    HangupCall();

    do
    {
        lReturn = ::lineShutdown(m_hLineApp);
        if (HandleLineErr(lReturn))
            continue;
        else
        {
            OutputDebugString("lineShutdown unhandled error\n");
            break;
        }
    }
    while(lReturn != SUCCESS);
```

```
m_bTapiInUse = FALSE;
m_hLineApp = NULL;
m_hCall = NULL;
m_hLine = NULL;
m_bShuttingDown = FALSE;

OutputDebugString("TAPI uninitialized.\n");

return TRUE;
                                                    }
```

References

[1] Mark A. Miller, P.E. “Voice over IP Technologies Building the Converged Network”, Introduction to VOIP, U.S.A., 2005, pp. 152-167, 215-223.

[2] Paul J. Fong, Eric Knipp, Davis Gray, “Voice over IP”, Different types of switching technology, Cisco Academy, USA 2003, pp. 145-159.

[3] John C. Bellamy, “Digital Telephony”, Digital Switching, Third Edition USA 2000, pp. 162-272

[4] Behrouz A. Forouzan, “Data Communication and Networking”, Circuit Switching and Telephone Network, TATA McGraw hill third edition 2004, pp. 197 – 210

[5] Craig Hunt , “TCP/IP Network Administration”, TCP/IP security, Third Edition 2003, pp. 123-156

[6] Hersent Grule and Petit, “IP Telephony”, Second Edition 2001, pp. 145-172

[7] Vishawnathan, “Telecommunication Switching System and Network”, Second Edition 2002, pp. 127-181

[8]Microsoft MSDN library, October 2000.

[9] http://www.trolltech.com

[10] http://www.voip-info.org

[11] http://www.tidp.org/HOWTO/VoIP-HOWTO.html