A Comprehensive VoIP System with PSTN Connectivity

YUEN Ka-nang

A Thesis Submitted in Partial Fulfillment

of the Requirements for the Degree of

Master of Philosophy

in

Computer Science and Engineering

June 2001

A Comprehensive VoIP System with PSTN Connectivity

submitted by

YUEN Ka-nang

for the degree of M.Phil in CSE
at The Chinese University of Hong Kong
in June 2001

Abstract

Voice over IP (VoIP) is the transmission of analog voice signal over packet-based data network. It is commonly used in an Internet point-to-point environment for voice communication. Instead of researching VoIP in an Internet environment, this thesis focuses research on voice over IP in an Intranet environment.

This thesis covers three aspects of VoIP for Intranet. The first aspect is the feasibility of VoIP in an Intranet environment. To do so, a feasibility study is done to show how well an Intranet can support a reasonable number of VoIP connections. The second aspect is the exploration of methods and techniques in building functional and robust Intranet VoIP systems. In this research, ways to construct a VoIP client program and other VoIP components are discussed. It shows that, in order to construct a well-performed VoIP system, suitable multimedia and networking technologies have to be used. The third aspect of this research is to investigate some innovative and useful computer applications that can be created using VoIP. In our work, two prototype VoIP applications have been created. Furthermore, a comprehensive VoIP project called Graduate Second Phone (GSP) which is described implements based on the developed VoIP technology. Finally, we give a brief conclusion on the experience of our work.

公共電話網絡與內聯網互接環境下的ＩＰ網絡語音傳送（ＶｏＩＰ）系統

submitted by

YUEN Ka-nang

for the degree of M.Phil in CSE
at The Chinese University of Hong Kong
in June 2001

Abstract

ＩＰ網絡語音傳送技術 Voice over IP (VoIP) 是利用封包形式的數據網絡傳送類比語音訊號，這種技術普遍應用於互聯網上的點對點語音通話。與其針對於互聯網上的ＩＰ網絡語音傳送技術進行研究，本論文將焦點放於內聯網上的ＩＰ網絡語音傳送研究。

本論文覆蓋內聯網ＶｏＩＰ的三個範疇。第一個範疇是關於內聯網ＶｏＩＰ的可行性，透過可行性研究可以得知內聯網能支持多少個合理質素的ＩＰ網絡語音連線。第二個範疇是關於探索具體和可靠的內聯網ＶｏＩＰ系統的建設方法與技術；透過了對ＶｏＩＰ客戶端軟件及其他元件的開發方法的研究，我們認爲要建立表現良好的ＶｏＩＰ系統，必須要運用適當的網絡及多媒體技術。第三個範疇是關於研究透過ＶｏＩＰ技術如何能創造出有創意及有用的電腦應用系統，我們建立了兩個ＶｏＩＰ應用系統的原形。除此之外，本論文記述了一個建基於已發展ＶｏＩＰ技術的專案計劃—研究生第二電話系統 Graduate Second Phone (GSP)。最後，論文針對整個研究中所得出的經驗作出簡短的總結。

Acknowledgement

I would like to thank my supervisor, Dr. Y. S. Moon, who suggests this exciting and innovative project to me as my M.Phil research project and supports me with advices and guidance throughout the research.

Moreover, I would also like to thank the other two members in my research group, Mr. Ho Ho Ching and Mr. Leung Cheung Chi, who have worked closely with me on the voice over IP research project. Without their support and encouragement, I would not be able to achieve what I have done in my research.

Last but not least, I would like to thank my parents, especially my mother, for welcome me home with bowls of delicious soup after I work hard on my research every day.

Table of Contents

List of Tables

List of Figures

1. INTRODUCTION

1.1. Background

Traditionally, our world of communication is divided into two kinds of network: telephone line network and data network. Telephone line network such as POTS (Plain Old Telephone Service) is used to carry analog voice signal between telephones; data network is used to carry digital data between computers. Due to the recent advancements of technologies in both kinds of network, the boundary between these two types of network starts to blur. With the advancement in computer modem technology, telephone lines no longer only carry voice but also data. The telephone line data rate has advanced from early days 2400 bps to 14400 bps (ITU V.32 bis), to current state-of-art 56000 bps X2, K56Flex and ITU V.90 modem technologies. More and more data can be transmitted through telephone lines. On the side of data network, with the advancement of CPU computation speed, digitization of analog voice signals such that these signals may be transmitted on data network is available. As a result, computer networks can no longer only carry data but also voice.

While these two networks have the same capability to carry both voice and data, they differ in terms of cost, stability and mode of transmission. For these reasons, several attempts have been made to connect these two kinds of network together for transmission of data or voice across them.

1.2. Objectives

There are three main objectives in this research. The first objective is to investigate how well current computer network speed and bandwidth can support VoIP. To do so, a feasibility study is done to examine different aspects of VoIP. It shows that a current computer network can support a reasonable number of VoIP connections. The second objective of this research is to explore methods and techniques in building functional and robust VoIP systems. In this research, ways to construct VoIP client and other VoIP components are discussed. It shows that in

order to construct a well-performed VoIP, suitable multimedia and networking technologies have to be used. Finally, the third objective of this research is to find out some innovative and useful computer applications that can be constructed using VoIP. In this research, three prototype VoIP applications are created which shows that useful VoIP application can be created that allows computer users and telephone users to communicate with each other easily and conveniently.

1.3. Overview of Thesis

This thesis is organized as follows. In Chapter 2, an overview on the VoIP technology will be given. The overview will mention various issues in VoIP. Then in Chapter 3, some experimental studies on the implementation of VoIP will be presented. The studies include a number of VoIP clients and a prototype voice stream mixing server. Experiments are done to evaluate the performance of them. In Chapter 4, two prototype VoIP applications will be introduced: pure-IP call-center and prototype PBX. These two prototype applications serve as cornerstones of a comprehensive VoIP project called Graduate Second Phone (GSP) that attempts to solve a telephone problem in our computer laboratory. The details of the Graduate Second Phone (GSP) project will be mentioned in Chapter 5, followed by conclusions in Chapter 6.

2. NETWORK ASPECT OF THE VOIP TECHNOLOGY

In this chapter, a general overview of VoIP technology is presented. First, an overview of VoIP is given. Then the four main elements in VoIP are discussed, followed by the analysis of factors affecting the performance of VoIP. Next, there is a section that describes the differences between Internet and Intranet VoIP – how VoIP technology is used differently in an Intranet hybrid-network environment as opposed to a general pure IP Internet environment. At the end of this chapter, a feasibility investigation on using VoIP in today's computer network technology is presented.

2.1. VoIP Overview

VoIP is the transmission of voice over IP network. Voice is transported in the form of digital voice packets over IP network. The packet transportation can be one-to-one or one-to-many depends on the type of voice communication. Different kinds of computer technology are used in VoIP such as signal processing and computer networking.

In summary, VoIP is a technology that enables users to have voice communication with each other without using conventional phone and public-switched telephone network.

2.2. Elements in VoIP

There are four main elements in VoIP: Call Setup, Media Capture/Playback, Media Encoding/Decoding and Media Transportation.

2.2.1. Call Setup

Call Setup is the first step in making a VoIP connection between participants. It is a mechanism that allows the participants to prepare for the voice connections. This includes finding the other side in the network, determining the bandwidth and codec

capabilities of the other side of the connection, request/acknowledge the connections, etc.

There are many protocols developed by various parties in making call setup for VoIP connections. The two most common protocols are H.323 [1] and SIP [2]. [3] presents more details on these two protocols and the differences between them.

2.2.2. Media Capture/Playback

Media Capture/Playback is the main interaction between VoIP and VoIP users. Media Capture is the process to record a VoIP user's analog voice in digital form by using a microphone. Media Playback is the process to present a VoIP user with the voice of the other users in the voice connection by using headphones or speakers.

2.2.3. Media Encoding/Decoding

Media Encoding is a process that converts continuous raw digital voice data captured from the Media Capture process into encoded digital voice data. Media Decoding is the reverse process of Media Encoding: it converts encoded digital voice data into continuous raw digital voice data. Media Encoding/Decoding is needed because the bandwidth for transporting the voice data is small, or the bandwidth occupied by voice data is required to be kept at a minimum for network utilization efficiency. By Media Encoding/Decoding, fewer bytes are needed to represent the same voice information as raw digital voice data.

Media Encoding/Decoding is done by codec (COder/DECoder). According to M. Goncalves [4], there are three types of codec: waveform codec, source codec and hybrid codec. Different codec have different voice quality performance and bandwidth requirement. Which codec to be used depends on the network environments and user requirements.

Some of the codec used in VoIP are G.711 [5], G.721 [6], G.723.1 [7], G.727 [8], G.729A [9] and GSM [10].

2.2.4. Media Transportation

Media Transportation is a process that transports the digital voice data over a computer network. The digital voice data is transported in the form of network packets on the computer network. The computer network must try its best to transmit the packets from the source to the destination of the voice connection.

As VoIP is a real-time application, many network protocols are developed specifically to transport real-time data such as VoIP. The most commonly used protocol for such an application is Real-time Transport Protocol (RTP) developed by IETF [11]. A similar protocol that is used to transfer real-time multimedia data is Real-time Streaming Protocol [12].

Other Internet protocols such as Transmission Control Protocol (TCP) [13], User Datagram Protocol (UDP) [14] that are used commonly in many network applications are not used to transport voice data.

For TCP, the sliding window and lost packet retransmission properties in TCP makes TCP unsuitable for voice transportation because even if a tiny interval of voice (such as 20 - 40 ms) is lost in the transmission, a retransmission of that interval of voice is attempted. This is not necessary for VoIP because if the lost interval of voice is small, human hardly notice the lost. Moreover, there are algorithms that can conceal the lost of voice packets without reducing the overall quality of the voice.

For UDP, there is no ordering information in the UDP packets. It is possible that two consecutive packets arrive out of order -- the first sent packet arrives later than the second sent packet. It is impossible to determine if the packet received is the first packet or the second packet in UDP. Moreover, UDP does not have information on the media type of the RTP packets.

2.3. Performance Factors Affecting VoIP

There are a number of factors that affect the performance of VoIP ranging from computer equipment to network conditions. In this section, several important factors are identified and discussed.

2.3.1. Network Bandwidth

Network bandwidth is the amount of data that can be transmitted in a fixed amount of time over the network. As the digitized voice is transported over a computer network, the bandwidth of the network determines how many VoIP connections can be supported simultaneously. The bandwidth consumed by a VoIP connection depends on the type of codec used, the amount of voice data in each RTP packet, and if a silence suppression algorithm is used.

2.3.2. Latency

In a VoIP voice connection, the delay time between voice traveled from one participant to the other participant(s) is called latency. It determines the users' perception of the responsiveness of VoIP voice communication. Latency comprises record delay, packetization delay, network delay, depacketization delay and playback delay. Latency can be represented by the following equation.

$$\text{delay (latency)} = t_r + t_p + t_n + t_d + t_k$$

t_r: Record delay
t_p: Packetization delay
t_n: Network delay
t_d: Depacketization delay
t_k: Playback delay

High latency in voice creates confusion that affects users' response to each other. The latency should be kept less than 150 ms if a good voice quality is preferred [15][16].

2.3.3. Packet Loss

Packet loss is another factor that affects the quality of the system. The more packets are lost in transportation between the participants, the more voice data is lost in the voice communication. As a result, the voice conversation is interrupted with silence intervals that affect the overall quality of voice communication.

There are mainly two reasons for packet loss. First, if there is congestion in the network, it will cause overflow in the router's packet queue. As a result, incoming voice packets to the router are forced to be dropped. Second, the bad quality of network cabling causes too much error in transmission that forces erroneous voice packets to be dropped.

Algorithms such as FEC (Forward Error Correction) are devised to alleviate this problem by adding redundant information to voice packets for reconstructing lost packets [17].

2.3.4. Voice Quality

Voice quality is another factor that affects users' perception of the quality of VoIP. The encoding, transmission and decoding processes in VoIP affects the voice quality in a VoIP voice connection. It is important to make sure the voice quality in VoIP meet the level users required.

2.3.5. Quality of Service (QoS)

Quality of Service (QoS) is one way to ensure the performance of VoIP meets a certain level. QoS ensures the packetized voice data is transported in the computer network with higher priority than other network traffic. This ensures that voice data is not affected by varying network traffic conditions.

One of the most commonly used QoS is Resource Reservation Protocol (RSVP) [18]. It ensures quality of service of voice packet transmission.

2.4. Different Requirements of Intranet VoIP and Internet VoIP

This section presents the different requirements of Intranet VoIP and Internet VoIP. It is very important to clarify the differences between them because they affect the way VoIP is implemented.

The following table shows the differences between Intranet VoIP and Internet VoIP.

	Internet VoIP	Intranet VoIP
Packet Loss/Delay/Jitter	Higher	Lower
Interoperability	Lower	Higher
Bandwidth Requirement	Stricter	Looser
Security Requirement	Higher	Lower

Table 2-1: Different properties between Internet and Intranet VoIP

2.4.1. Packet Loss/Delay/Jitter

The first difference is Packet Loss/Delay/Jitter. In Internet VoIP, the path between the participants usually has to go through the Internet. As the Internet is a commonly shared computer network, there is a lot of other traffic going through the Internet at the same time. The variation of the amount of traffic in the Internet is therefore high. Moreover, the path between the participants usually goes through a number of non-homogeneous routers and communication links. As a result, data packets that go through the Internet are more likely to be lost or delayed. In contrast, the voice communication path in Intranet VoIP usually lies only between routers in the Intranet. Therefore, the number of hops for packets to go from one participant to another participant is small. Moreover, as most of the Intranet is used exclusively by an organization, the variation of the traffic is small and under-controlled. In such an environment, the packet loss and delay is much lower.

The amount of packet loss determines if a packet recovery technique is needed in VoIP. For the higher packet loss rate in Internet, it may be necessary to use a packet recovery technique such as Forward Error Correction (FEC) code [17] to

recover the packet lost in the transmission. In contrast, lower packet loss rate in Intranet VoIP means it may not be necessary to use a packet recovery technique. As a result, the VoIP transport mechanism in Intranet VoIP is much simpler.

The amount of packet delay and jitter determines the size of packet buffer and packet prefetch length. As the delay and jitter in Internet is higher, therefore the size of buffer used in Internet VoIP is higher than that in Intranet VoIP in order to prefetch more voice data and prepare for variations in packet arrival time. As a result, Internet VoIP client has to allocate more memory for more buffers.

2.4.2. Interoperability

The second difference is interoperability. In Internet VoIP, it is likely that participants in a voice connection use different types of operating system, computer, codec and VoIP software. Although there are standards such as H.323 [1] to ensure the interoperability between different VoIP software and hardware, it is possible that the two participants cannot communicate with each other due to the interoperability problem. In contrast to Internet VoIP, the operating system, computer, codec and VoIP software in Intranet VoIP are more likely to be the same or have been tested extensively for interoperability. Moreover, due to similar machine configuration, the initialization of voice connection is much simpler as there is no need to negotiate the codec to be used. As a result, the call setup in Intranet VoIP is simpler than Internet VoIP.

2.4.3. Available Bandwidth

The third difference is available bandwidth. As the Internet network is shared among many Internet users, the bandwidth requirement between VoIP participants is stricter. Internet VoIP must be able to cope with situations where the available bandwidth is small. As a result, high compression CPU-intensive codec such as G.723.1 is usually used in Internet VoIP to minimize bandwidth consumption. In contrast, the available bandwidth in an Intranet is higher. As a result, less CPU-intensive code (such as GSM) may be used by paying a small penalty of using more bandwidth.

2.4.4. Security Requirement

The security requirement of the Internet is higher than that of Intranet VoIP. This is because in Internet VoIP, the voice packets have to go through many routers and midpoints in the Internet that are vulnerable to eavesdropping. In order to prevent eavesdropping, it may be necessary to use encryption in voice packets that travel through the Internet [19]. In contrast, the voice packets in an Intranet are protected from outsiders by a firewall. Therefore, the security requirement is looser and it may not be necessary to do encryption on voice packets.

2.5. Some Feasibility Investigations

In this section, the maximum number of VoIP connections that can be supported in a network is investigated from a theoretical point of view.

2.5.1. Bandwidth Calculation

First, the bandwidth consumed by a VoIP connection is calculated.

For the amount of voice data in each RTP packet, the voice data can be transmitted in a large number of small packets or a small number of large packets. A RTP packet typically contains 20 to 40 ms voice data [20]. From the figure below, each voice packet has 12 bytes of RTP header, 8 bytes of UDP header and 20 bytes of IP header. The header in each RTP packet adds bandwidth overhead to the overall voice transmission. Given a fixed amount of voice data, the bandwidth required to transmit it depends on how many data are put into each RTP packet. In this calculation, we assume there is only one voice frame in each voice packet to provide lowest latency for the voice connection. Moreover, we assume the codec used in the voice connection is GSM. The frame size of GSM codec is 33 bytes.

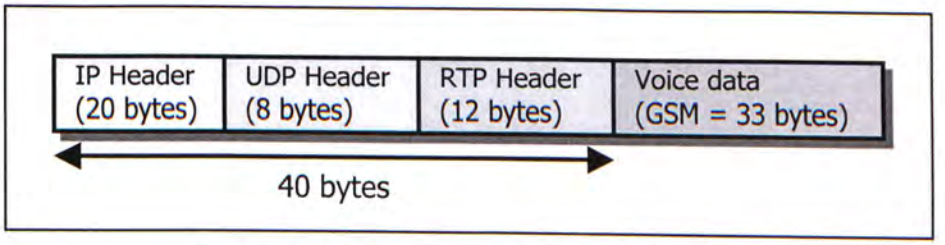


Figure 2-1: The structure of a voice packet

As a result, the size of a voice packet is:

Packet size = 33 bytes + 40 bytes
= 73 bytes

(33 bytes = GSM voice data, 40 bytes = IP + UDP + RTP headers)

Voice packets are sent at 20ms interval, therefore the bandwidth of a one-way connection is:

Bandwidth of one-way connection = 73 bytes / ($20x10^{-3}$)
= 3650 byte/s
= 3.56 kB/s

As VoIP is a two-way connection, that is, voice is sent from both sides of a connection:

Bandwidth of two-way connection = 3.56 kB/s x 2
= 7.12 kB/s

As a result, each voice connection consumes 7.12 kB/s.

If we assume the efficiency of Fast Ethernet is only 10%, then:

100 Mb/s = 12.5 MB/s
= 12800 kB/s

Therefore,

Calculated bandwidth = 12800 kB/s * 10%
= 1280 kB/s

Number of VoIP connections = 1280 kB/s / 7.12 kB/s

= **180 connections**

As a result, it shows that even using a pessimistic assumption that the efficiency of the Fast Ethernet is 10%, the Fast Ethernet can support **180** connections. This concludes that the Fast Ethernet bandwidth is sufficient to support a large number of VoIP connections based on the bandwidth calculation.

2.5.2. Simulation

The above calculation only calculates bandwidth without considering the latency of voice packets. In this section, a network simulation is created to simulate the packet flow of a number of voice connections in a network. The objective of this simulation is to determine whether a large number of VoIP connections may cause the latency of VoIP connection to be too high such that VoIP communication is impossible. In the experiment, the latency of voice packets in the simulated environment is measured.

The simulation is done using a network simulator called Network Simulator 2 (ns-2) [21]. The network simulator program runs on a Solaris machine. The process of the simulation is shown in the figure 2-2.

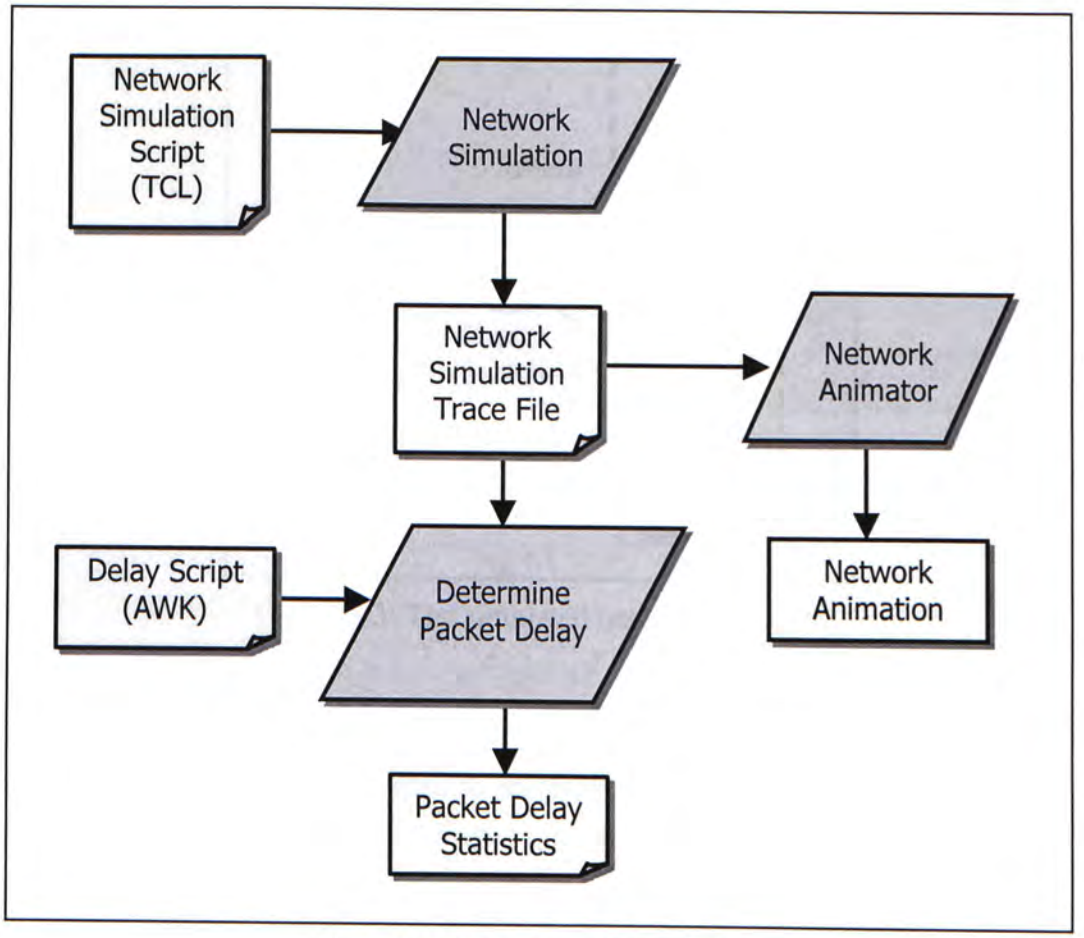


Figure 2-2: The flow of simulation

First, a network simulation script is written using Object TCL (OTCL) [22] to specify the Intranet network setup. The computer network consists of an Intranet gateway, a router (switch) and a number of PCs. The simulation script is then passed to the network simulator for simulation. The network simulation produces a simulation trace file that contains the simulation results. The simulation trace file is passed to the Network Animator (NAM) [23] to visualize the simulation result. On the other hand, the simulation trace file is used to determine the delay time for each packet in each voice connection. To do so, a delay determination script written in AWK scripting language [24] is used to parse the simulation trace file to find the transmission and arrival time of each packet. The delay statistics are then plotted to show the variation of delay.

The computer network constructed in the network simulation script is shown in the figure 2-3.

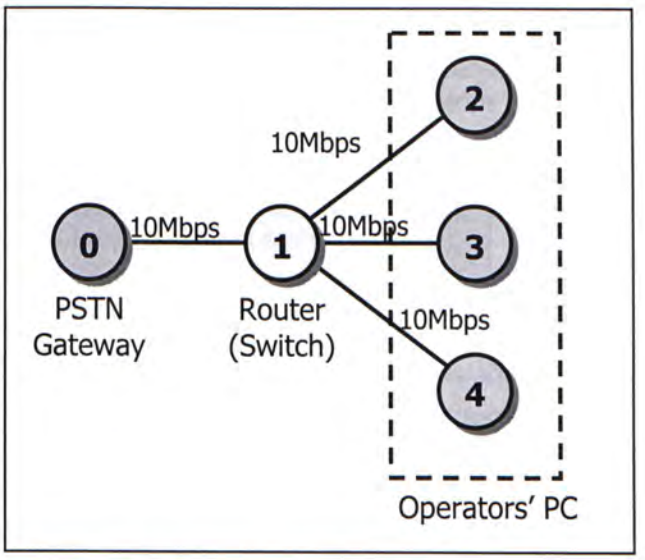


Figure 2-3: The simulated network structure

In the script, nodes are used to represent network endpoints such as PCs and the gateway. Moreover, a node is used to represent a switch or hub that routes packets between computers in the network. For each PC node (such as node 2 in the figure), a duplex VoIP voice connection is made between it and the gateway (represented by node 0). Voice packets are sent from the gateway to the PC and vice versa at regular intervals. The transmission and arrival time of the voice packets are recorded to calculate the packet latency. Moreover, the same experiment is repeated using a different number of PCs, from 1, 2 to even 50 to determine the distribution of packet latency under different number of PCs.

The duration of the simulation is 6 seconds. The voice transmission between the gateway and PCs starts at 0.5s and stops at 5.5s. They are not started at the start of the simulation and stopped at the end of the simulation to ensure that the start and end of the simulation environment is stable, e.g. without packets in transmission.

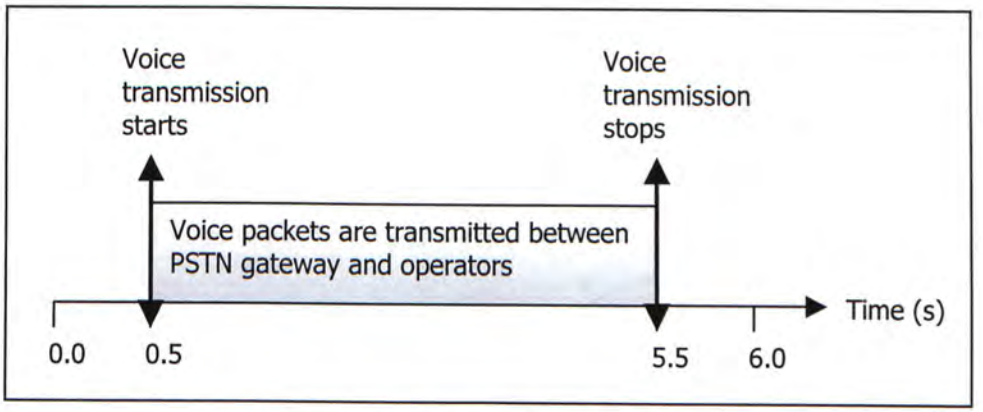


Figure 2-4: The simulation timeline

The size of voice packets is 73 bytes as in the calculation in the last section. Packets are sent every 20 ms. The period between voice transmission starts and stops is 5 seconds, therefore there are 250 packets sent from the source to the destination in a VoIP communication.

The result of the experiment is described as follows.

The following graph shows that the mean of the packet latency increases as the number of connection increases. It should be noted that the mean of the packet latency increases at a slow linear rate. Even though there are 50 VoIP connections, the mean of the packet latency is about 22 ms, which is quite small. As a result, the number of voice connections does not have much affect on the mean of the packet latency.

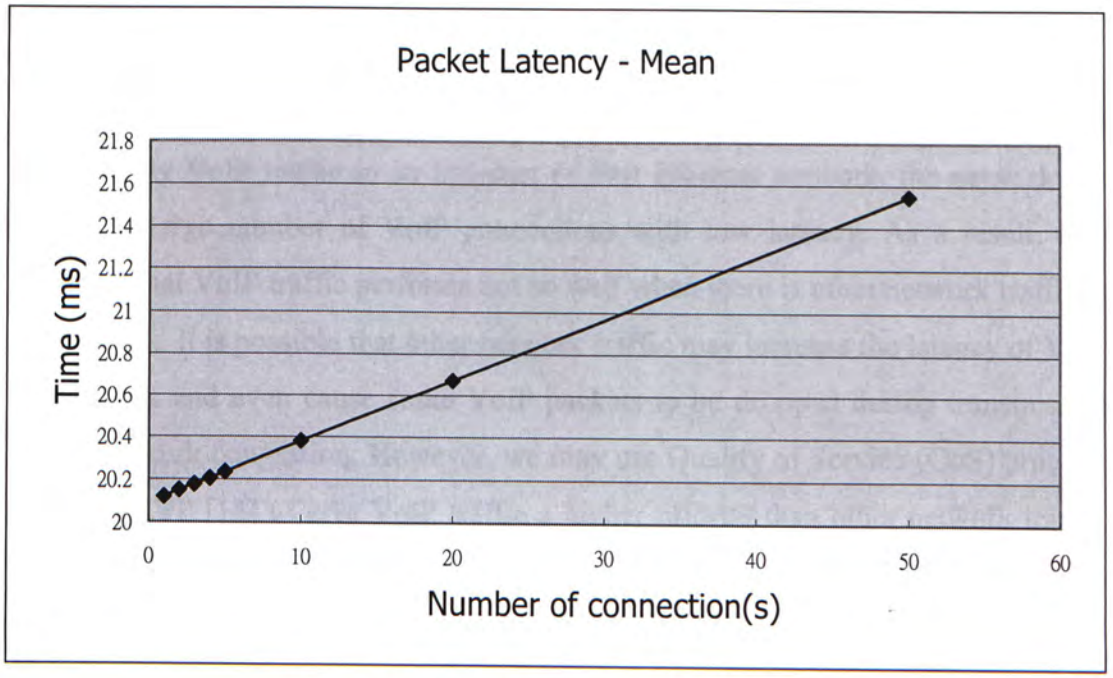


Figure 2-5: The mean of packet latency

The following graph plots the standard deviation of packet latency against number of connections. It shows that the standard deviation increases slowly as the number of voice connections increases.

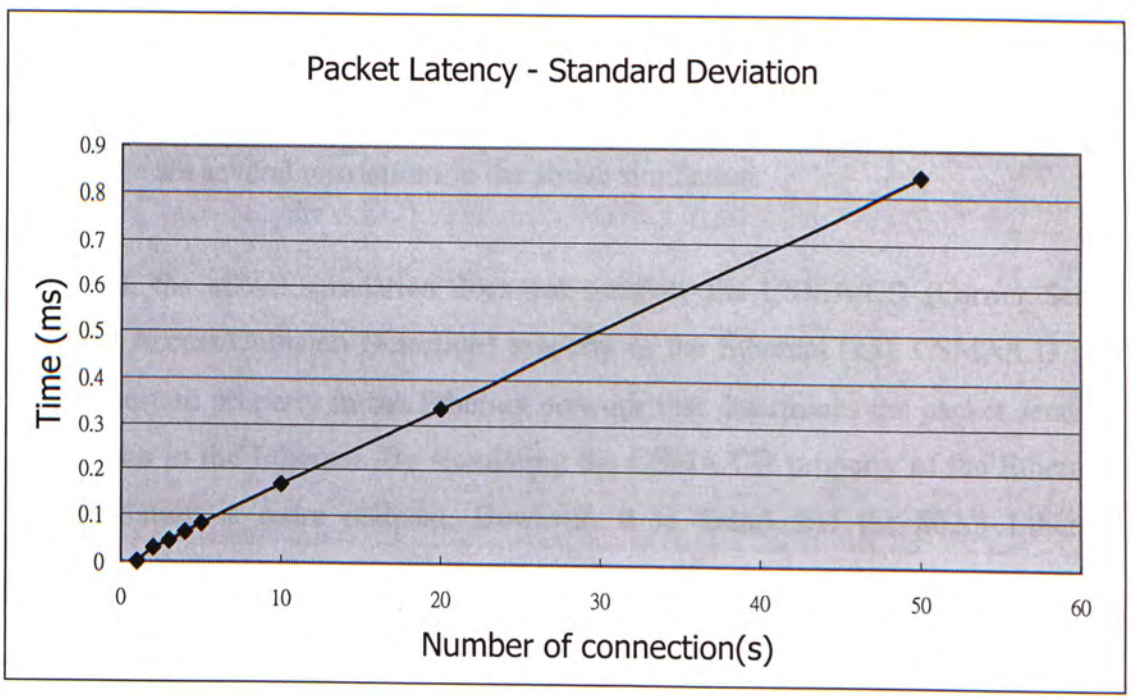


Figure 2-6: The standard deviation of packet latency

2.5.3. Conclusion

To conclude, based on the above calculation and simulation, it shows that if there is only VoIP traffic in an Ethernet or Fast Ethernet network, the network can support a large number of VoIP connections with low latency. As a result, it is predicted that VoIP traffic performs not so well when there is other network traffic in the network. It is possible that other network traffic may increase the latency of VoIP connections and even cause some VoIP packets to be dropped during transmission due to network congestion. However, we may use Quality of Service (QoS) protocol such as RSVP [18] to give VoIP traffic a higher priority than other network traffic. Therefore, the quality of VoIP can be ensured to be above a certain level even when there is other network traffics in the network.

Moreover, it should be noted that even when it is shown from experiment and calculation that a router can support a large number of VoIP connections, it is better to distribute the VoIP connections over a number of gateway and routers for reliability. Otherwise, all the VoIP connections will fail if there is any problem in the gateway or the router.

2.5.4. Simulation Restrictions

There are several restrictions in the above simulation.

First, the above simulation does not simulate the CSMA/CD (Carrier Sense Multiple Access/Collision Detection) property of the Ethernet [25]. CSMA/CD is a very important property in the Ethernet network that determines the packet sending mechanism in the Ethernet. By simulating the CSMA/CD property of the Ethernet, the simulation is more realistic. However, it is found that the 802.3 Ethernet simulation facilities in Network Simulator 2 contains bugs and does not function properly.

Moreover, this simulation assumes the node "1" is a switch instead of a hub. In the above Intranet scenario, there is no advantage in using a switch instead of a hub in the network, as there is no non-overlapping VoIP path for packet transmission. All

VoIP connections are setup between the gateway and PCs. Therefore, all VoIP paths have the edge between node 0 and node 1 in common. As mentioned in the last paragraph, the Ethernet simulation does not function properly, therefore it is not possible to simulate a hub. As a result, a switch simulation is used instead.

Different types of routers use different routing algorithms and sizes of packet buffers. As a result, it is difficult to set the routing algorithm and size of packet buffers. In this simulation, the default routing algorithm and packet buffer size are used in the router. The default routing algorithm is static routing and the packet buffer is a 100 packets FIFO Drop Tail queue.

3. SOFTWARE ASPECT OF THE VOIP TECHNOLOGY

This chapter presents the technologies used in implementing VoIP on a computer platform. The technologies discussed in this chapter focus on point-to-point VoIP communication. Multicast or broadcast VoIP is not mentioned.

First, the structure of VoIP clients is presented. These VoIP clients serve as voice terminals for computer users to participate in a voice communication. Then a prototype mixing server that can mix voice channels is briefly mentioned.

As most of these prototype software are written in Java language, theoretically they should be platform-independent, i.e. they can run on any OS platform that support Java such as a Unix, Linux or Windows OS platform. The truth is that these prototype programs are not totally platform-independent because many elements such as multi-threading, accessing low-level multimedia devices in VoIP are highly platform-dependent. Therefore, unless other specified, these programs are preferred to run on Windows OS platform instead of Unix or Linux.

Moreover, many toolkits and API used in the experiments are supported better in Windows 9X/NT platform than in other platforms. For example, an API called Java Media Framework (JMF) is more updated in the Windows version than in the Linux version. Therefore, the Windows 9X/NT platform is preferred.

3.1. VoIP Client in JMF

This section introduces our first implemented IP side VoIP client called JMFPhone. Our first IP side VoIP client is implemented in Java with the help of the Java Media Framework (JMF) package [26].

The Java Media Framework (JMF) is a Java API for multimedia application [26]. JMF adds advanced multimedia functionality to the core Java. It works as building blocks for constructing multimedia programs. The JMF package is not in the core Java language and must be downloaded separately. There are three distributions

of Java Media Framework in its latest version 2.1: All-Java, Windows-platform performance pack and Solaris-platform performance pack. The All-Java distribution can only perform multimedia playback but not multimedia capturing. Therefore, All-Java distribution is not suitable for implementing VoIP as VoIP needs both multimedia playback and multimedia capture. As mentioned earlier the preferred OS platform is Windows, therefore the Windows-platform performance pack is used.

The figure 3-1 shows how this JMF VoIP client uses Java and JMF technology to conduct VoIP.

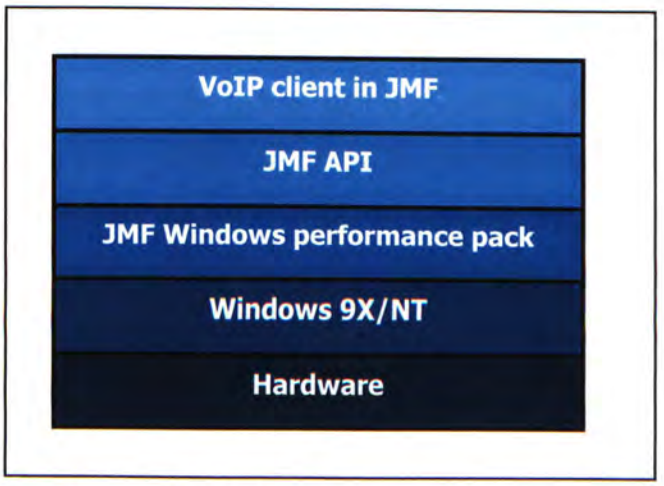


Figure 3-1: VoIP client application layers

3.1.1. Architecture

The JMF VoIP client consists of three components: the JMF VoIP main program, the user interface module and the VoIP core module. The figure 3-2 shows the JMF VoIP client main program and the components inside it.

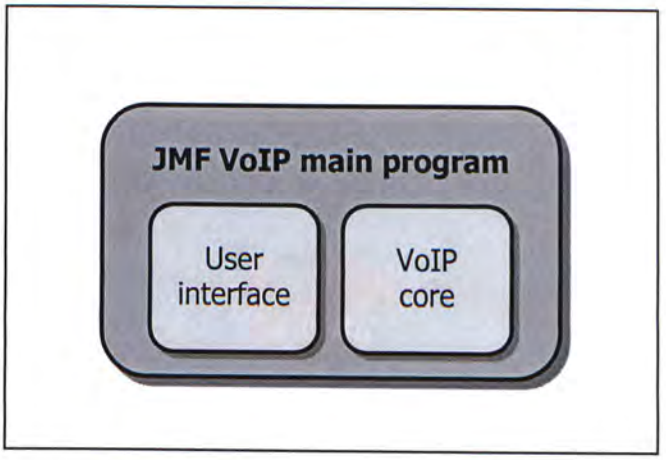


Figure 3-2: Components in VoIP client

The user interface module provides an interface for the user to initiate or breakdown a voice connection. It provides a simple call-setup mechanism in this VoIP client – peer-to-peer manual call setup. The participants of both sides of a voice connection have to activate the VoIP function manually. This differs from traditional client-server phone call setup: at first, one participant initiates a phone call, and then the other side hears the phone ring and answers the phone call. It is possible to modify the call setup to the one used in traditional phone call setup.

The VoIP core provides the main VoIP functionality of the VoIP client. There are a number of JMF objects inside the VoIP core. The JMF objects are interconnected to form paths for incoming and outgoing voice streams between network and multimedia hardware devices such as a sound card.

The user interface and VoIP core are separated by putting them into different modules such that VoIP functionality can be easily integrated into any application by adding the VoIP core module.

The structure of the VoIP core is shown below.

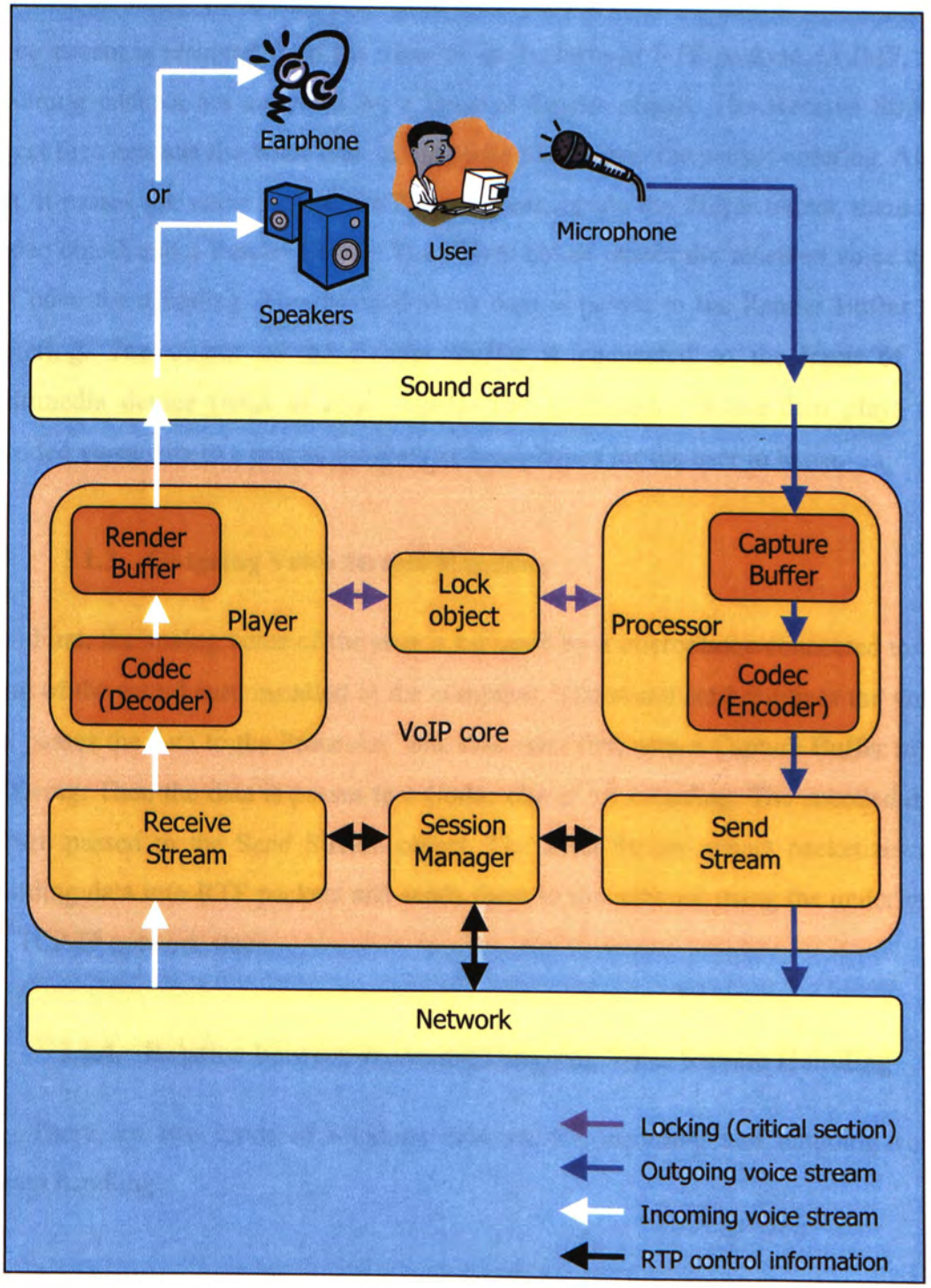


Figure 3-3: The structure of VoIP client in JMF

3.1.2. Incoming Voice Stream Handling

The incoming voice stream is handled as follows. First, the data of the incoming voice stream is received from the network in the form of RTP packets. In JMF, the incoming packets are collected by a Receive Stream object. The Receive Stream object then extracts the voice data in the packets and performs packet ordering. After that, it passes the voice data to the Player object. Inside the Player object, there are Codec object and a Render Buffer. The Player object passes the received voice data to Codec for decoding. The decoded voice data is passes to the Render Buffer for buffering. The output of the Render Buffer is connected to the input of the multimedia device (such as sound card). The multimedia device then plays the decoded voice data to a pair of speakers or headphones for the user to listen.

3.1.3. Outgoing Voice Stream Handling

First, the analog voice of the user is captured by a microphone connected to the input of the sound card installed in the computer. The sound card digitizes the voice and passes the data to the Processor. The Processor first uses a Capture Buffer to do buffering. Then the data is passes to a Codec object for encoding. The encoded data is then passed to the Send Stream object. The Send Stream object packetizes the encoding data into RTP packets and sends them to the network using the underlying OS TCP/IP network stack.

3.1.4. Relation between Incoming/Outgoing Voice Stream Handling

There are two kinds of relations between the incoming and outgoing voice stream handling.

The first relation is the one between the Player object and the Processor object. These two objects are both controlled by a Lock object. The Lock object is used to ensure the code fragments that initialize Player object and Processor object do not run concurrently, because the Player object and Processor object initialization fail if they are initialized at the same time. The initialization methods of the Player object and the Process object are asynchronous, that means the methods complete execution even the objects are not completely initialized. As a result, it is possible for these two

code fragments to run concurrently by running them in either one or two threads. To prevent them from running concurrently, a lock object is used. The locking mechanism is shown in the following figure.

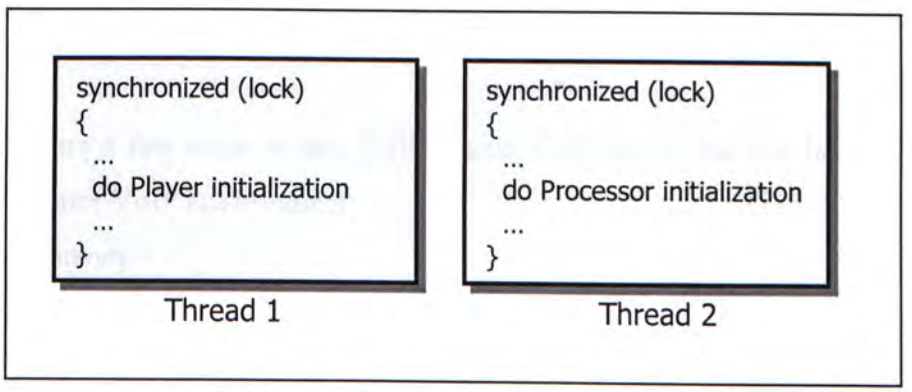


Figure 3-4: The code fragments of the locking mechanism

The code fragments of Player initialization and Processor initialization are encapsulated into a "synchronized (lock)" statement. As a result, in order that the code fragment be executed, the thread must acquire the lock object. If the thread cannot acquire the lock object, it will wait until the other thread exits the scope of the "synchronized (lock)" statement and releases the lock object. As a result, by using this Lock object, the Player initialization and Processor initialization will not interfere with each other.

The second relation is the one between the incoming/outgoing voice stream handling. Both the Receive Stream object and the Send Stream object communicate with the same Session Manager. The Session Manager controls the sending and receiving of RTP packets between the application program and the network. The Session Managers of JMF VoIP clients in the voice connection communicate with each other using Real-time Control Protocol (RTCP). The statistics for packet sending and receiving are exchanged between Session Managers. In order to collect the statistics, the Session Manager queries the packet statistics from the Send Stream and the Receive Stream objects.

3.1.5. Areas for Further Improvement

The performance of the JMF IP side VoIP client is good. By using the clients on the computers, two computer users can communicate with each other using VoIP. The voice quality of the VoIP client is acceptable upon test.

There are a few areas in this JMF IP side VoIP client that can be improved to provide a better VoIP environment:

1. Latency
2. Loading time

Improving the Media Capture/Playback can further lower the latency of the JMF side VoIP client. It is because the capture/playback of current JMF Player and Processor objects are not optimal. From various emails in the official JMF mailing list [27], the Player and Processor objects are not performing capture/playback at their best. Moreover, a forthcoming new version of JMF includes two new objects called DirectAudioRender and DirectAudioCapturer for better capture/playback in Windows [28]. These two pieces of evidence prove that the capture/playback of current JMF Player and Processor is not optimal. As a result, it is possible to lower the latency by building our own capture/playback components using OS-level multimedia capture/playback.

The loading time of the JMF side VoIP client can be reduced. It is because in JMF a process called flow graph building is performed on Media Capture/Playback. Flow graph building is performed as JMF assumes that the media format of incoming audio stream from soundcard or network is not same and fixed. Therefore the flow graph building process computes an optimal set of codec that should be used to convert the format of incoming audio stream into a format that the user specified. As a particular audio format for all audio streams is fixed in our Intranet VoIP, it is not necessary to do flow graph building. As a result, the loading time of the client can be shortened.

Moreover, the loading time can be further improved by preloading the Java classes. It is because in JMF VoIP client, the codes for conducting VoIP is not loaded

until the call has been setup. The class loading causes a delay between the call setup and the time the user starts hearing the other side speaks. This gives the user an impression that the call setup takes a lot of time. If the classes can be preloaded or pre-cached when the application is initialized, then the delay caused by class loading can be avoided. One way to do so is to initiate a dummy voice connection to force the Java classes that conduct VoIP to be loaded from the hard disk into the computer memory.

These few areas of improvement are implemented in the following sections by using Java or C++ code accessing native OS services.

3.2. Capture/Playback Enhanced VoIP Client

The JMF Capture/Playback Enhanced VoIP client (called JMFPhone2) is an improvement over JMFPhone. JMFPhone2 improves voice capture and playback by using native C++ code for capture/playback through Windows multimedia API. As a result, Media Encoding/Decoding and Media Transportation are done in Java while Media Capture/Playback is done in C++.

The following diagram shows how the enhanced VoIP client uses the underlying API for the three major VoIP operations: Media Transportation, Media Encoding/Decoding and Media Capture/Playback.

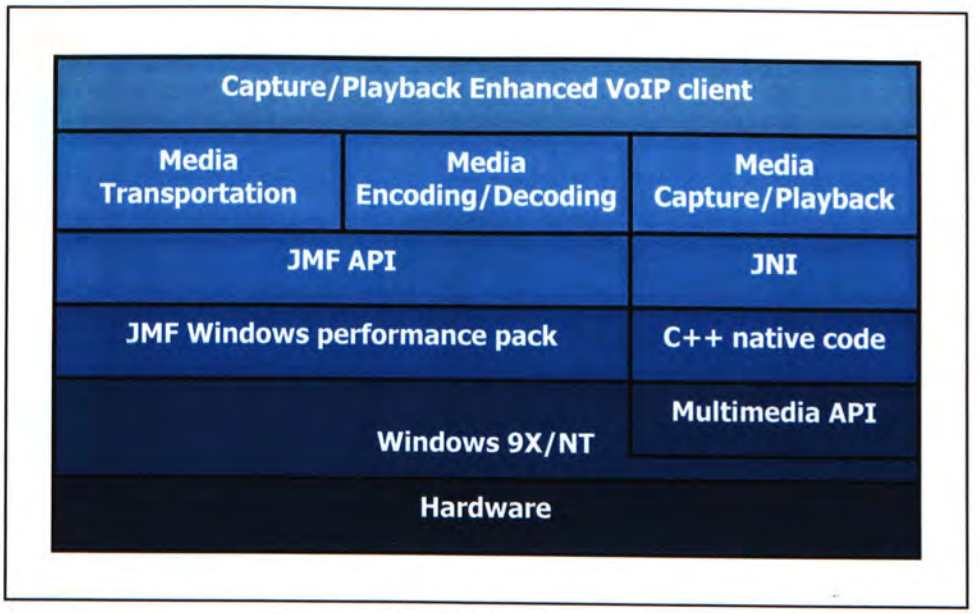


Figure 3-5: VoIP application layers

It shows that the client uses native C++ code to access OS level multimedia API for Media Capture/Playback. In order for Java to interface with C++, Java Native Interface (JNI) is used. By using JNI, Java methods can call native C++ functions that in turn access native Windows Multimedia API.

3.2.1. Architecture

The enhanced VoIP client consists of the same components as the original VoIP client.

The difference is that enhanced VoIP client uses a different VoIP core. This VoIP core accesses Windows Multimedia API for Media Capture/Playback [29]

The architecture of the enhanced VoIP core is shown as follows.

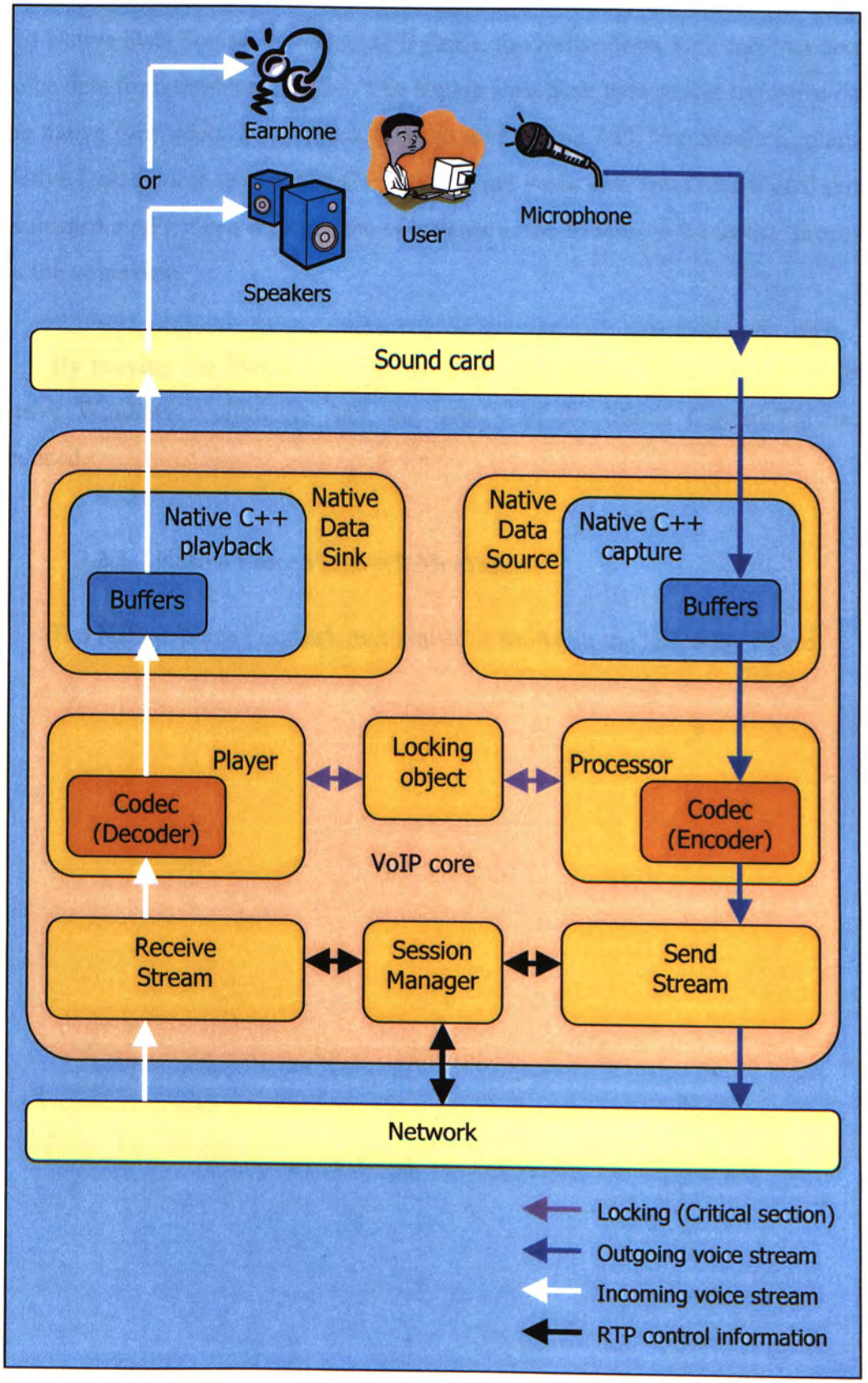


Figure 3-6: The structure of the Capture/Playback enhanced VoIP client

The main difference between this enhanced VoIP core and the original VoIP core is the Media Capture/Playback is done by two new Java objects: Native Data Sink and Native Data Source. For Media Playback, the Native Data Sink receives decoded voice data from the Player object. The Native Data Sink then passes the voice data to the native C++ code for playback through multimedia API. For Media Capture, the Native Data Source uses native C++ code to get voice data from sound card through Multimedia API. Then it passes the voice data to the Processor for further processing on the voice data.

By moving the Media Capture/Playback functionality from the Java object to native Windows multimedia API, the latency of the Media Capture/Playback is reduced.

3.2.2. Native Voice Playback Mechanism

The Native Voice Playback mechanism is shown in the following figure.

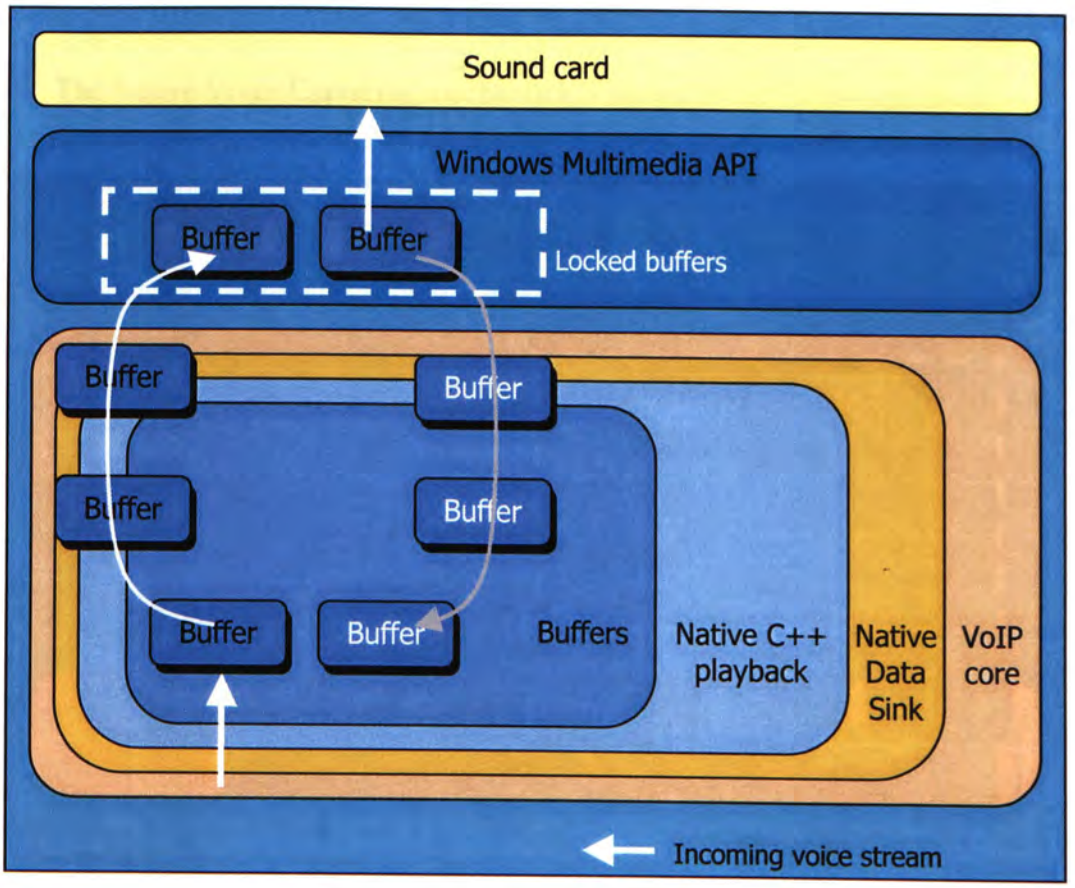


Figure 3-7: Native Voice Playback Mechanism

There is a pool of buffers for playback of the voice data. For each incoming frame of voice data, the frame is put into one buffer and passes to the Windows Multimedia API. The buffers filled with voice data are those with the word "Buffer" in black. The Windows Multimedia API then locks the buffer for playback. If a buffer is already under playback in Windows Multimedia API, then other buffers entered into the Windows Multimedia API will be queued and wait until the current buffer finishes playback. Buffers that have been played are returned to the buffer pool. The buffers that finished playback and emptied are those with the word "Buffer" in white.

By using a buffer pool technique, we can ensure that there is always voice data in the Windows Multimedia API ready for playback. As a result, there is not a situation that there is no audio data in Windows Multimedia API. Otherwise a short period of silence is created that will disturb the user.

3.2.3. Native Voice Capturing Mechanism

The Native Voice Capturing mechanism is shown in the following figure.

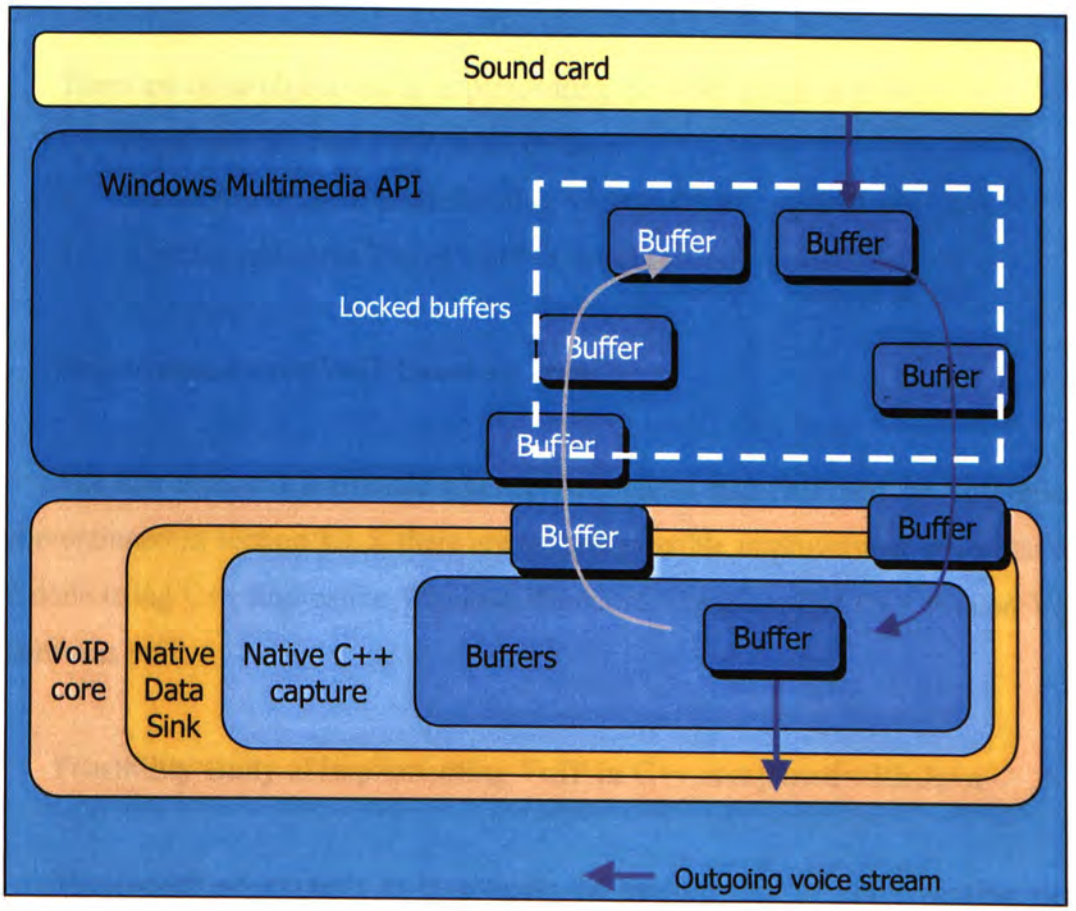


Figure 3-8: Native Voice Capture Mechanism

The mechanism is similar to the one in voice playback. However, it should be noted that in voice capture, most of the empty buffers are locked in the multimedia API ready for filling with voice data. The reason for this is that voice data is continuously generated from the sound card. By using the above buffering technique, there is always empty buffer ready for filling the voice data from the sound card. Otherwise, voice clipping occurs and affect the voice quality of VoIP.

3.3. Win32 C++ VoIP Client

As mentioned in the section 3.1.5, it is possible to further improve the performance of VoIP by implementing the VoIP client in C++. Therefore a C++

version VoIP client was built to test the performance of VoIP in C++ native OS environment.

3.3.1. Objectives

There are three objectives in implementing the VoIP client in C++:

1. Improvement over VoIP based on Java
2. Feasibility study of implementing VoIP in C++ compared with Java
3. A better understanding of VoIP in details

Improvement over VoIP based on Java

The first objective is to build a better VoIP client than JMF version VoIP client. As mentioned in section 3.1.5, there are several possible improvement areas that can be done using C++ and native Windows functions, therefore this C++ version VoIP client was built.

Feasibility study of implementing VoIP in C++ compared with Java

The second objective is to investigate the performance of implementing VoIP client in C++ instead of Java. A common belief is that the advantages of Java are quick development time and platform-independence. The disadvantage of Java is its runtime performance is not as good as Java. In contrast, the advantage of C++ is better performance than Java, while the disadvantages are longer development time and possibly platform-dependence (depends on what APIs C++ accesses). As VoIP is a real-time application that depends heavily on good performance, C++ is probably a better choice for implementing VoIP. The implementation of this VoIP client in C++ serves as a practical feasibility study of the approach.

A better understanding of VoIP in details

While it is easy to implement VoIP client in JMF, most of the underlying details are hidden from the programmer. By implementing a VoIP client in C++, we can get

a better understanding of the underlying details such as multi-threading, buffering, device control, RTP management, etc.

3.3.2. Architecture

JMF provides all RTP network transmission, codec for encoding and decoding, objects for controlling sound card capture/playback. However, there is no such integrated environment in C++ when implementing VoIP. As a result, the three major VoIP components: Media Capture/Playback, Media Encoding/Decoding and Media Transportation are done using different components.

There are two approaches for Media Capture/Playback in Windows 9X/NT environment: Multimedia API [29] and DirectSound [30]. The first approach, Multimedia API, is used in the enhanced version VoIP client. The other approach, DirectSound, is a lower-level API than Multimedia API. DirectSound controls the sound card directly for better performance.

This C++ version VoIP client uses Multimedia API as it is easier to implement. It is possible to switch from Multimedia API to DirectSound due to the modular architecture of the VoIP client.

For Media Encoding/Decoding, there is not much C++ codec library available. This C++ version VoIP client uses a GSM codec library [31] that allows encoding linear voice data into GSM voice data and decoding GSM voice data into linear voice data.

For Media Transportation, a C++ RTP library called JRTPLib is used [32]. This C++ RTP library supports unicast and multicast RTP transmission. As a result, it is possible to modify the C++ VoIP client to work under multi-participant multicast environment.

The following diagram shows how the C++ VoIP client uses various API and libraries.

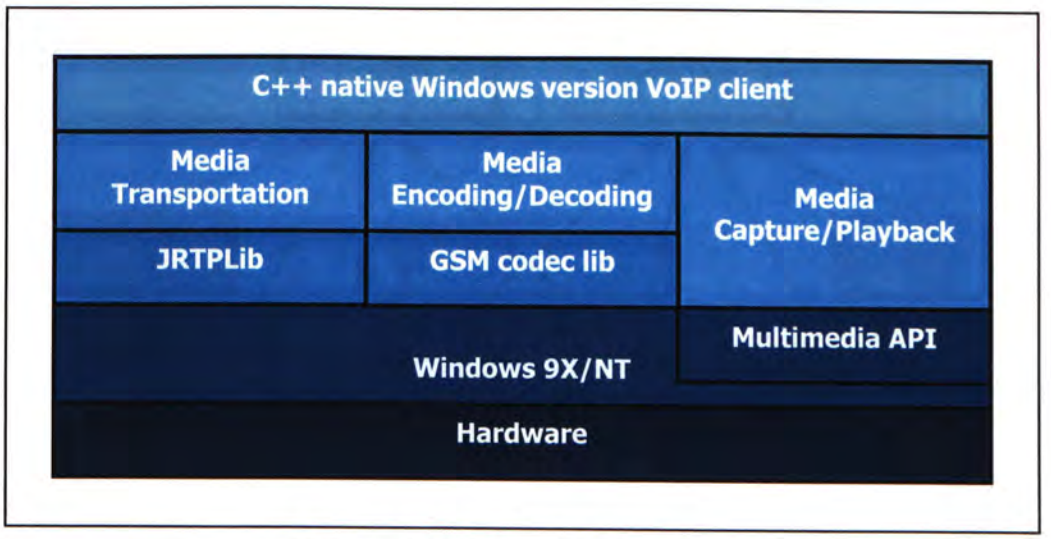


Figure 3-9: VoIP application layers

The C++ version VoIP client is built on six major objects: Capturer, Renderer, GSM Encoder, GSM Decoder, RTP Receiver and RTP Sender.

Capturer object captures audio data from the sound card using Windows Multimedia API. Renderer object playbacks audio data to sound card using Windows Multimedia API. GSM Encoder object encodes linear audio data into GSM audio data. GSM Decoder object decodes GSM audio data into linear audio data. RTP Receiver object receives RTP data from the network. RTP Sender object sends RTP data to the network.

As voice data is pushed instead of pulled from the source to the destination, some of the above objects are the destination for pushing voice data while some objects are the source for pushing voice data. As a result, two classes PushSrc and PushDest are created to define common properties for source and destination of audio data respectively.

The following class hierarchy diagram shows the six major classes inherited from PushSrc and PushDest.

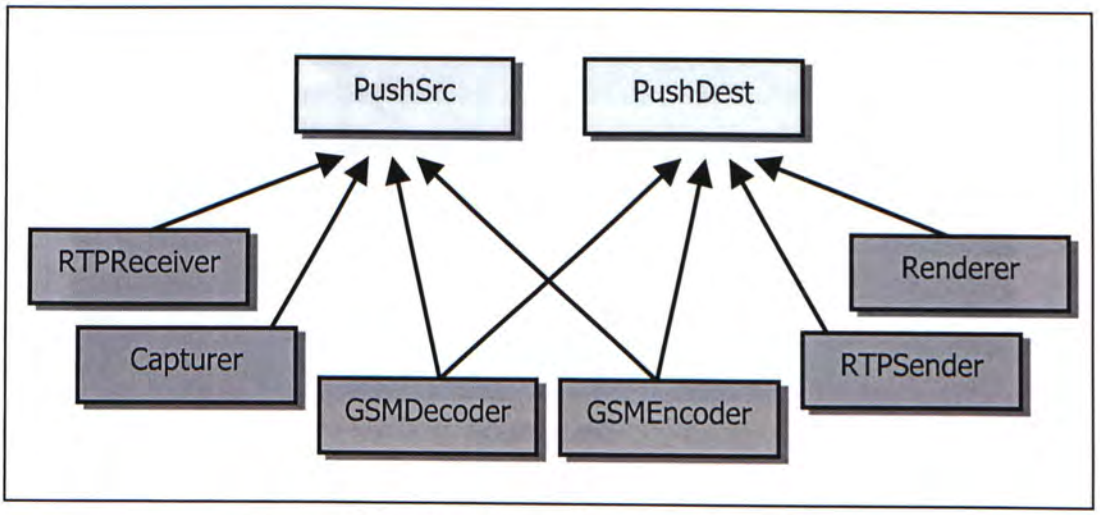


Figure 3-10: Class hierarchy of the C++ objects

It should be noted that GSMEncoder and GSMDecoder inherit from both PushSrc and PushDest. This is because they are both the source and destination of pushing voice data.

Based on these six major objects, the C++ version VoIP client is constructed as follows:

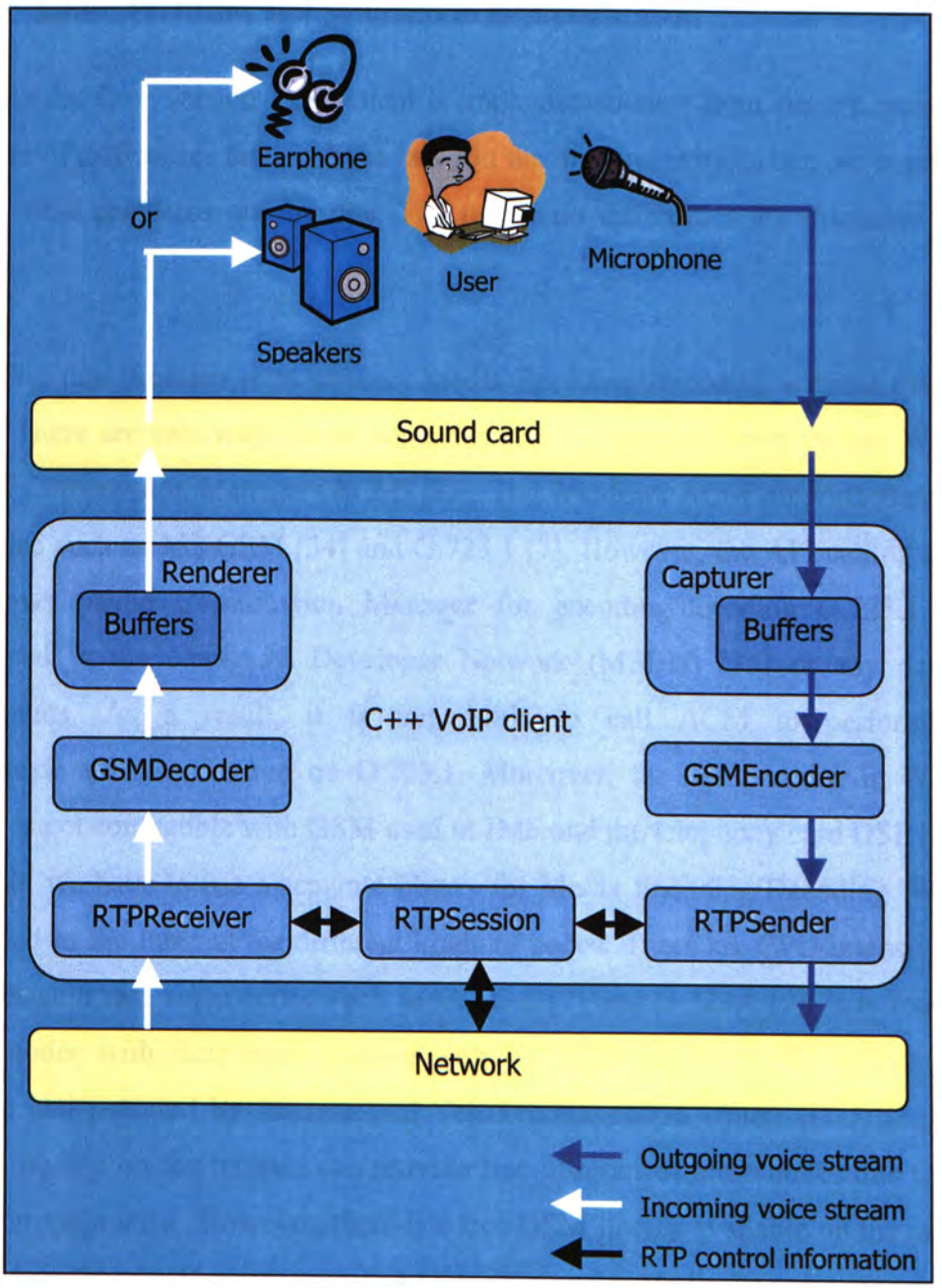


Figure 3-11: The structure of Win32 C++ VoIP client

The function of the six objects is similar to the original JMF VoIP client.

It should be noted that in the C++ VoIP client, the program starts faster than the program ends. This is because when the program starts, all the threads can be started immediately without waiting for others. However, when the program ends, the program main thread must wait for other threads to exit first. Otherwise, the main thread may release some memory and hardware resources that some threads are still using. This explains why the program starts faster than the program ends.

3.3.3. Problems and Solutions in Implementation

As the C++ version VoIP client is implemented start from scratch, there are a number of difficulties that we have faced in the implementation, but, we managed to solve these problems one by one. The two main difficulties are discussed in this section.

The first problem is to conduct Media Encoding/Decoding without the use of JMF. There are two ways to do so: either use a separate library or use Windows Audio Compression Manager (ACM) [33]. The Windows ACM supports many kinds of codec such as MS-GSM [34] and G.723.1 [7]. However, the API calling steps of Windows Audio Compression Manager for encoding/decoding G.723.1 is not disclosed in the Microsoft Developer Network (MSDN) [35] or any developer documents. As a result, it is impossible to call ACM to perform audio compression/decompression on G.723.1. Moreover, the GSM codec in Windows ACM is not compatible with GSM used in JMF and the telephony card DSP [10]. As a result, we have to use a separate library for Media Encoding/Decoding. We have searched in the Internet for different kinds of codec. There are CPU-intensive high-compression rate voice codec such as G.723.1 [7] and G.729A [9]. It is found that these codec with their name starting with "G" followed by an ID with digits are owned and patented by International Telecommunication Union (ITU) [36]. As a result, no one on the Internet can provide free libraries of these codec due to patent and copyright issue. However, there is a free GSM library available on the net. As a result, our C++ VoIP client uses this GSM library for Media Encoding/Decoding. It should be noted that although our VoIP client is based on the GSM library currently, it is easy to switch from GSM codec to any other codec due to our object-oriented structure of the VoIP client. The GSM codec object can be replaced by other codec objects easily.

The second problem that we have faced is that the performance of the VoIP client is not good initially. The voice processed by the client is broken and the quality is bad. After investigation, we have found that the reason is that the threads responsible for processing incoming/outgoing streams are not working in regular intervals as they should be. This is because voice in VoIP is processed frame by

frame. As a result, the threads for processing it must be woken up by the OS scheduler and do processing in regular intervals. To solve this problem, the thread priority of these two threads is increased to real-time priority. As a result, the threads are scheduled to wake up more precisely and do processing in regular intervals.

By solving the second problem mentioned above, we discovered one important requirement in VoIP application. The requirement is that the underlying operating system must provide very precise thread scheduling. That is, the operating system must provide a way to ensure threads wake up in a regular interval to process voice frame even when there are other OS activities such as page swapping, hard disk loading occuring at the same time. It is impossible to wake up the thread in a precise manner if the OS thread scheduling does not provide good real-time support. For example, the time interval is 20 ms, next time the thread should be woken up at 1000ms. However, due to page swapping in the OS, the thread has to delay its wake up to 1002ms. As a result, the wakeup time of the thread is not precise. This affects the continuity of voice capture and playback and causes interruption in the voice stream. Although a buffering technique can alleviate this problem, a precise thread scheduling is required to solve this problem and provide good VoIP performance.

As the thread scheduling in Windows NT is better than that in Windows 95/98 due to the Windows NT OS design, it is predicted that the threads in NT VoIP client run more precisely such that the overall performance of VoIP client in Windows NT is better than Windows 95/98. We may later test-run our VoIP clients on Windows NT platform to check if this is true or not.

3.4. Win32 DirectSound C++ VoIP Client

The last VoIP client is the Win32 DirectSound C++ VoIP client. This VoIP client is similar to the Win32 C++ VoIP Client. The difference is that DirectSound VoIP client uses the DirectSound in DirectX for media capture and playback, while the Win32 C++ VoIP client uses the Multimedia API for media capture and playback.

The main objective of implementing this VoIP client is to determine if it is possible to further improve VoIP client implemented in native Win32 C++. As mentioned earlier, it is believed that the DirectSound API is better than the Multimedia API for low latency audio manipulation. Because VoIP is a low latency audio manipulation application, it is expected that DirectSound API will perform better in such application.

3.4.1. Architecture

The architecture of the Win32 DirectSound VoIP client is shown in the following figure.

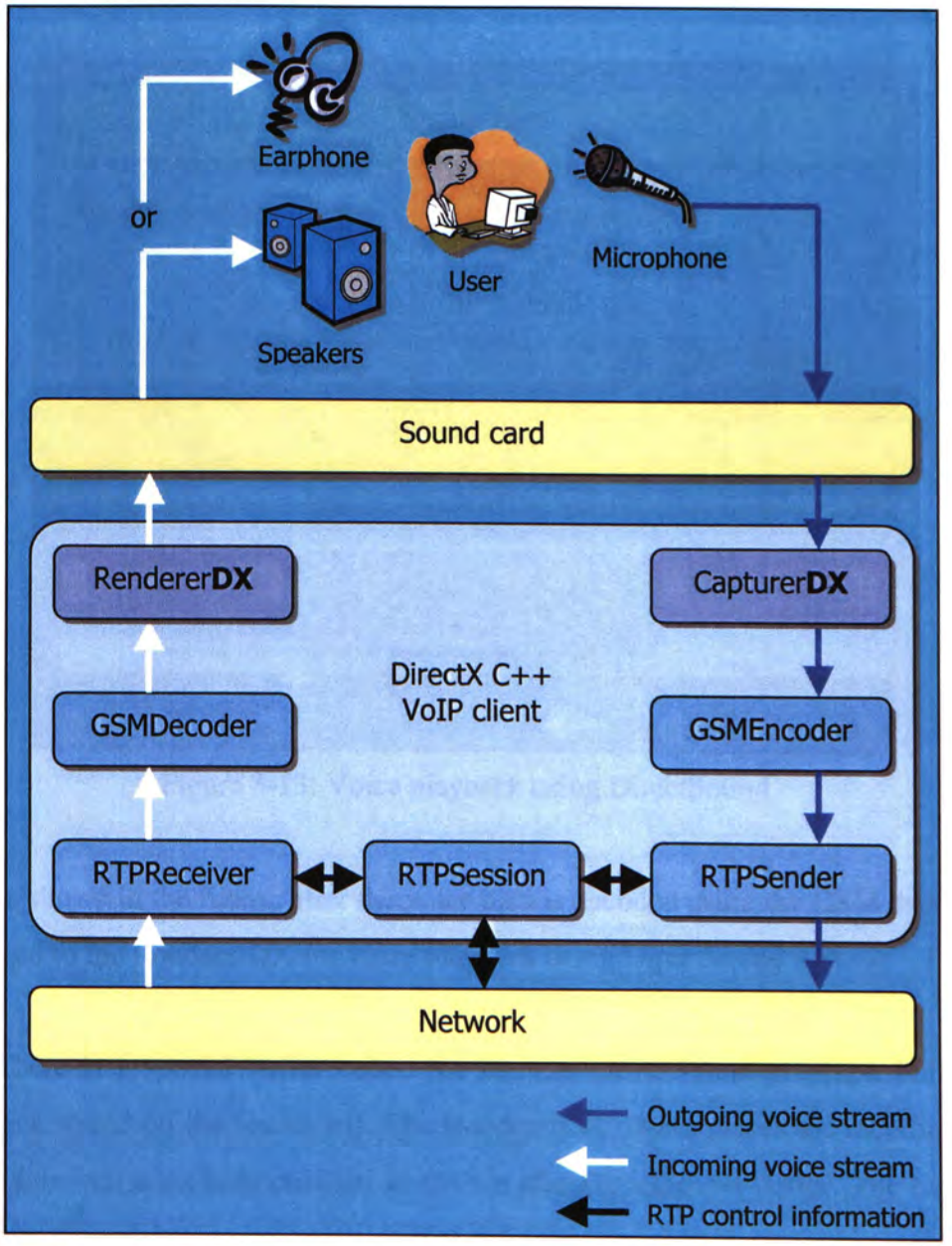


Figure 3-12: The structure of Win32 DirectSound C++ VoIP client

As shown in the figure, the Renderer object is replaced with a RendererDX object for media playback. Similarly, the Capturer object is replaced with a CapturerDX object for media capturing. Both RendererDX and CapturerDX objects utilize the DirectSound API for media capture and playback. The details of the media capture and playback are discussed in the following two sections.

3.4.2. DirectSound Voice Playback Mechanism

The mechanism of DirectSound voice playback is shown in the following figure.

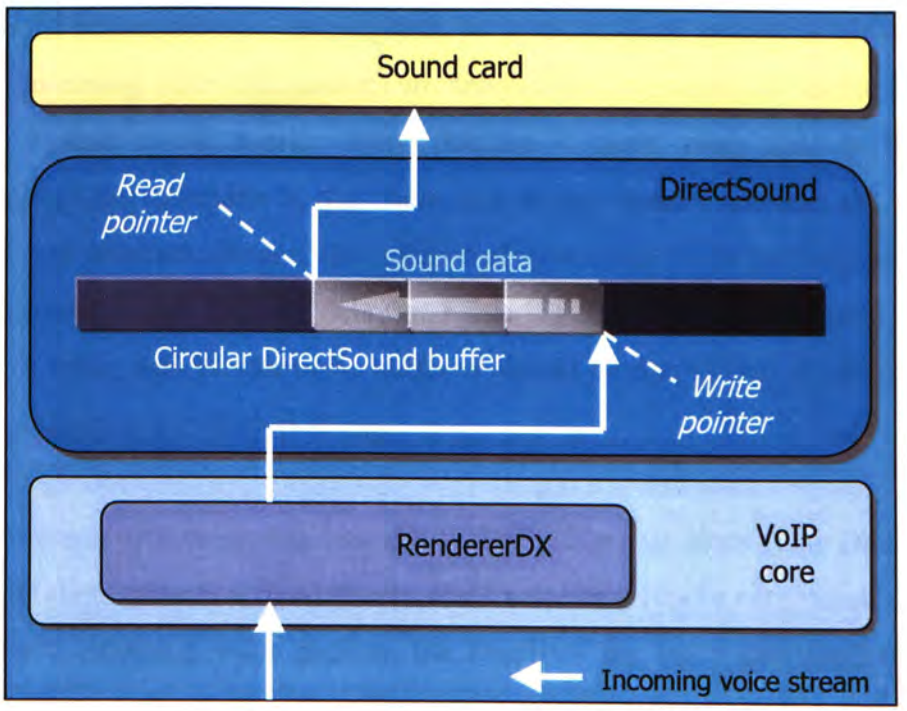


Figure 3-13: Voice playback using DirectSound

As shown in the figure, after the voice data is decoded using the GSM decoder, it is passed to the RendererDX for voice playback using DirectSound API.

There is a special buffer called the circular DirectSound playback buffer for playback sound on the soundcard. The RendererDX object places the incoming raw voice data into a suitable position in the circular DirectSound buffer. The buffer in the DirectSound is circular because there are two special pointers moving circularly in the buffer. The two pointers are the read pointer and the write pointer. The sound card reads voice data pointed by the read pointer and advances the read pointer to the right at the same time. The RendererDX object places incoming voice data to the position pointed by the write pointer and advances the pointer to the right at the same time. As a result, if the playback is smooth, there is always sound data to the right of the read pointer ready for playback. However due to jittering and delay in VoIP voice transmission, it is possible that voice data in the buffer is depleted such that no voice data is available for playback. In order to cope with this problem, a buffering technique is used. Instead of starting playback after the first block of voice data is received, the block of voice data is buffered. The voice playback is started later until more blocks of voice data are received and buffered. After a considerable amount of

voice data blocks are buffered, all the buffered voice data are placed in the DirectSound buffer and start the playback. As a result, there are blocks of voice data even the incoming voice data packets are delayed due to jittering. It should be noted that the system could tolerate more jittering if more voice data is buffered. Nevertheless, the drawback is that there is a higher latency between the voice is captured and playback. Hence, the number of voice data to be buffered should be chosen carefully such that the delay and jittering is balanced. We found that the number of voice data blocks to be buffered should be one to two in our computer network as there is not much jittering.

The length of a voice data block is typically 20ms or 40ms. This DirectSound C++ VoIP client transmits voice data in 40ms voice data blocks encapsulated in RTP packets. For incoming voice packets, the length of the voice data blocks may be 20ms or 40ms. Although the size of the incoming voice data blocks is different, the handling is similar in the sense that the blocks are placed on the circular DirectSound buffer for playback to the sound card.

The following diagram shows an example of the buffering technique used to tackle jittering problem.

Figure 3-14: Buffering example, time = 3

In the example, the prefetch setting for the playback is two, and the rate of incoming voice packet is assumed to be one packet/time unit. Therefore, a voice packet arrives at t = 1. Another voice packet arrives at t = 2. Yet another voice packet arrives at t = 3 as shown in the figure. For t = 1 or t = 2, there is no voice playback due to buffering. After the third voice packet arrived at t = 3 as shown in the figure, the playback is started such that the voice packet at t = 1 is played to the soundcard.

The effect of this buffering is that there is a delay of two time units between receiving and playback. In spite of this, the advantage is that the playback can tolerate jittering as long as three time units.

Assume the packets for t = 4, t = 5 and t = 6 arrive late. The following figure shows that there is no packets 4, 5 and 6 in the buffer because they are late.

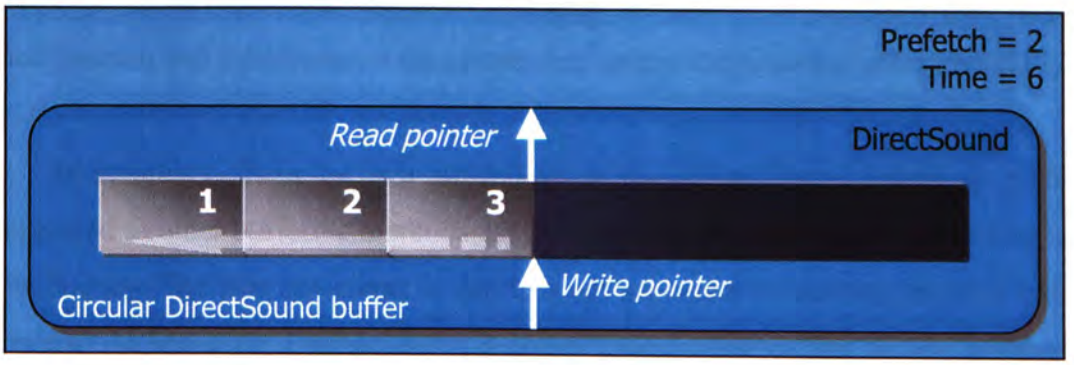


Figure 3-15: Buffering example, time = 6

Between time unit 3 and 6, the read pointer advances from that in figure 3-14 to that in figure 3-15. It is obvious that there is still voice data for playback even the packets 4, 5 and 6 are late. This shows the buffering technique in effect that ensures the smooth playback even some packets are arriving late.

Figure 3-16 shows the effect if the packets 4, 5 and 6 arrive just in time at time unit 6.

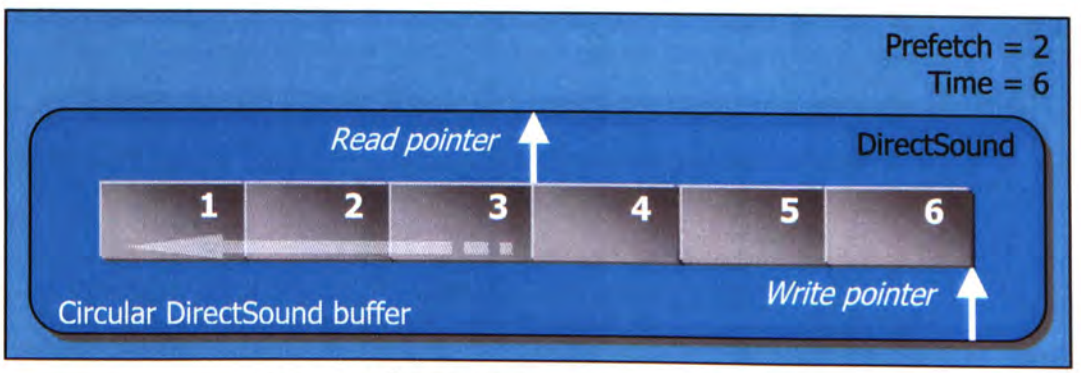


Figure 3-16: Buffering example, time = 6

From the figure, it shows that there is voice data for time = 4 just in time for playback. The playback now functions smoothly. As a result, it shows the advantage of the buffering technique in handling jittering problem.

On the other hand, if the packets 4, 5 and 6 are still late, then there is no voice data for playback and the user will hear silence. There are many techniques developed to alleviate such problem. One technique is to playback a simulated random background noise such that the user feels comfortable [37]. The other technique is to playback the content of the last voice data packet [37]. In our prototype implementation, the user will hear silence. It is expected that further modification and refinement to the system can handle this situation more gracefully.

In summary, there are two advantages of this DirectSound buffering method compared to other buffering methods used in previous VoIP clients: space saving and low latency. The first advantage is that it is not necessary to allocate an extra buffer space to do buffering. Consequently, memory space is saved. The second advantage is that the buffering is done in DirectSound buffer such that the buffering is closer to the playback such that the buffering cannot be interfere by thread scheduling which is possible in other buffering methods.

3.4.3. DirectSound Voice Capturing Mechanism

The mechanism of DirectSound voice capturing is shown in the following figure.

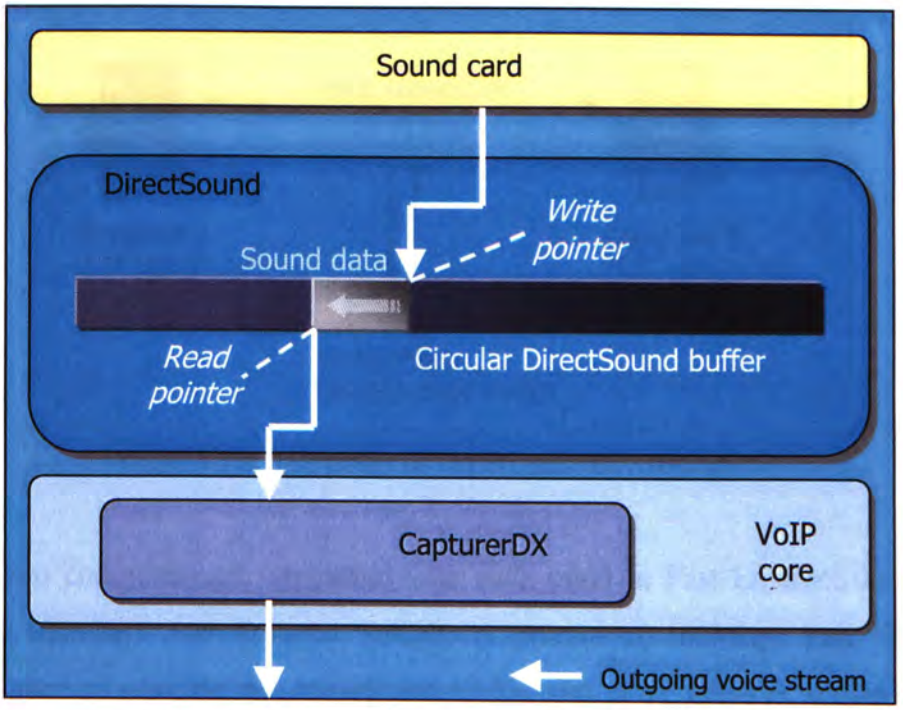


Figure 3-17: Voice capturing using DirectSound

As shown in the figure, the sound card places the captured voice data periodically to the circular DirectSound buffer and adjusts the write pointer to the right. At the same time, the DirectSound generates an event to the CapturerDX component. On receiving the event, the CapturerDX component extracts the sound data from the circular DirectSound buffer and transfers it to the GSM encoder connected with it.

The mechanism of capturing voice data using DirectSound is similar to the one using Multimedia API. The DirectSound voice capturing mechanism is used because it is not possible to use multimedia API for voice capturing when DirectSound voice playback is used.

3.5. Testing VoIP Clients

An experiment is done to evaluate the performance of these VoIP clients.

3.5.1. Setup of Experiment

The setup of the experiment is shown in the following figure.

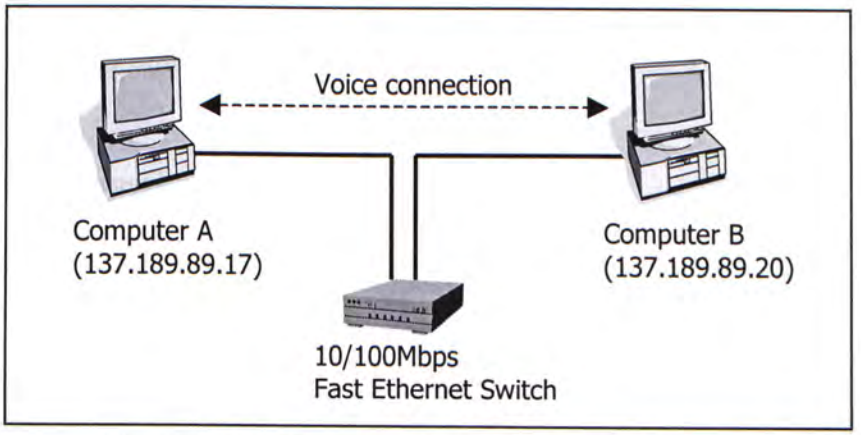


Figure 3-18: The setup of experiment environment

The two computers are connected with each other in Fast Ethernet through a private 10/100Mbps Fast Ethernet Switch. A private 10/100Mbps Fast Ethernet switch is used to separate the experiment traffic and other network traffic in our department. This avoids our department's network traffic from interfering with the experiment.

Two computers A and B are loaded with each VoIP client in turn to setup a voice communication. The latency, bandwidth and the startup time of the voice communication are recorded.

The configuration of the computers is shown in the following table.

Computer	A	B
IP	137.189.89.17	137.189.89.20
CPU	Celeron 300Mhz	PII 266Mhz
OS	Windows 98	Windows 98
RTP port (Receiving)	1400	1400
RTCP port (Receiving)	1401	1401
RTP port (Sending)	1402	1402
RTCP port (Sending)	1403	1403

Table 3-1: Configuration of the computers

After booting into the Windows 98 operating system, the amount of background processes is kept to a minimum to prevent other programs interfering with the experiment.

A program called DU meter [38] is used to trace the bandwidth usage. The latency and startup time is measured manually with a stopwatch.

3.5.2. Experiment Results

The result of the experiment is shown in the following table.

VoIP client	Latency (second)	Bandwidth (kB/s)	Startup time (second)
JMF (*JMFPhone*)	0.9 – 1	2.5	8(*), 2
Enhanced Capture/Playback (*JMFPhone2*)	0.6 – 0.7	2.5	6(*), 2
Native Win32 C++ (*AICPhone*)	0.3	2.5	1
Native Win32 DirectSound C++ (*DirectX*)	0.3	2.5	1

(Notes: * = For the first time only)

Table 3-2: VoIP clients experiment results

For latency, the original JMF VoIP client is the highest and VoIP clients in C++ are the lowest.

The bandwidth consumption of all the VoIP clients is the same because they use the same codec, the same frame size and the same frame to RTP packet ratio.

The startup time for Java VoIP programs is higher than the startup time for C++ VoIP clients. This is because the Java class loading takes time. Once the classes are loaded into memory, the startup time of VoIP connection is greatly reduced to 2 seconds that is comparable to C++ VoIP clients.

The quality of the voice in the enhanced capture/playback VoIP client and C++ VoIP client is better than that of the original JMF VoIP client.

The Win32 C++ client seems to perform the same as the Win32 DirectSound C++ client. The fact is that the DirectSound C++ client is more tolerant to network

jittering. If the two VoIP clients run for a long time, the delay for the Win32 C++ client will be higher because the delay is accumulated.

3.5.3. Experiment Conclusion

From the experimental results, it is shown that the latency of VoIP client is improved by using customized playback/capture component. The startup of the Java client is slower than C++ client.

By using Windows Multimedia or DirectSound API to capture/playback the voice, the voice quality is better.

Based on the latency and startup time results, it shows that C++ VoIP client performs better than their Java VoIP counterparts in latency and startup time.

3.6. Real-time Voice Stream Mixing Server

The VoIP technology used in the Intranet is mainly for point-to-point voice communication. That is, there are only two participants in a voice connection. However, it is sometimes necessary to transmit the mixed voice data from these two participants to a server for archive. As a result, a prototype real-time voice stream mixing server is created.

By using this prototype real-time voice stream mixing server, the voices of two participants in a voice connection are mixing into one voice stream. The voice stream is sent to an archive server for archive. Later if a user needs to retrieve the content of a previous conversation, he/she may search or browser in the archive server.

3.6.1. Structure Overview

The structure of the prototype real-time voice stream mixing server is implemented using Java Media Framework (JMF). The number of channels to be mixed can be changed due to the flexible design of the object interactions in the server. Hence, the server is capable of mixing more than two or more voice channels

together. The mixed voice can either be played locally or transmitted over the network to an archive server.

The structure of the prototype real-time voice stream mixing server is shown in the following diagram.

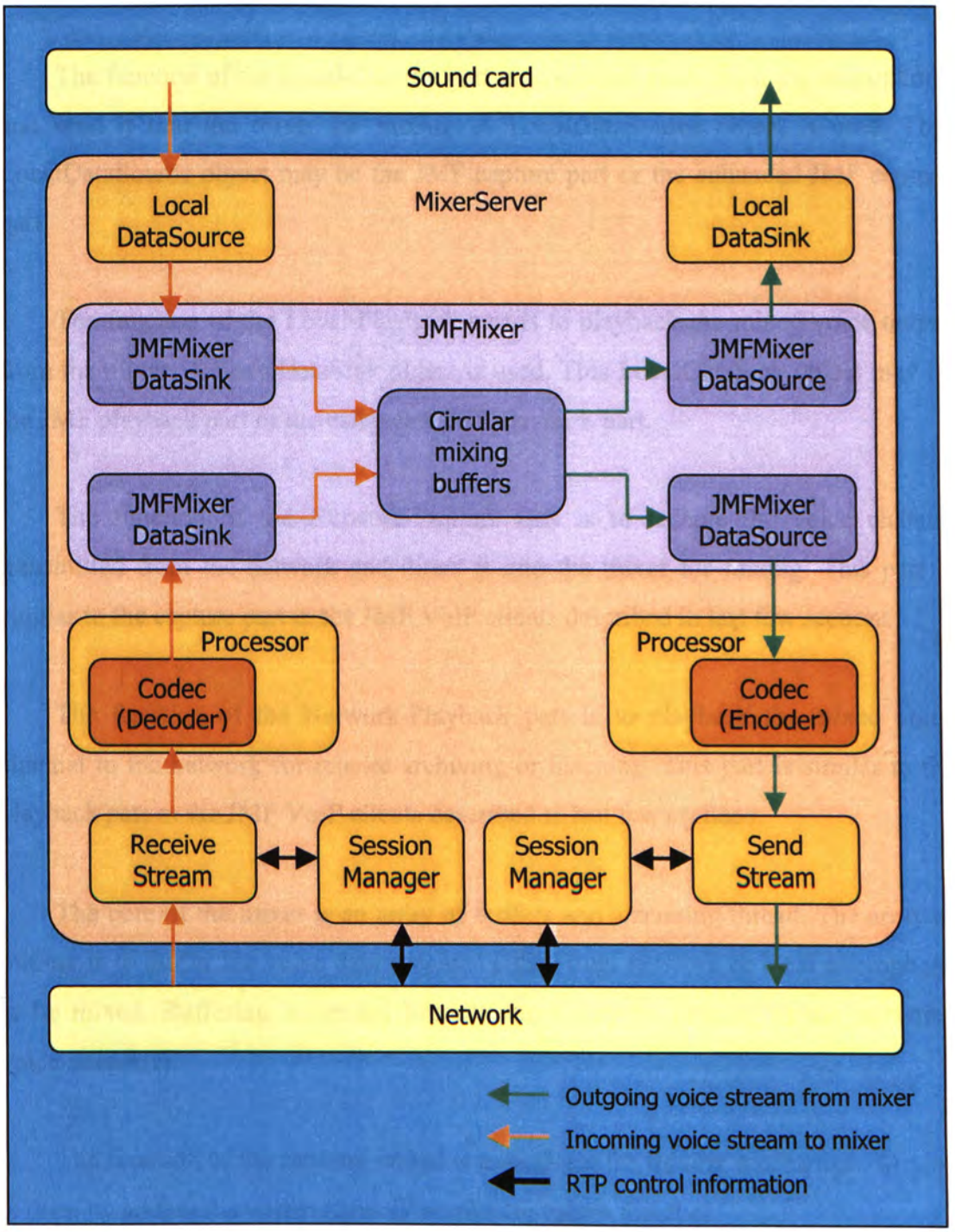


Figure 3-19: The structure of the prototype mixer

In the center of the diagram is a block called circular mixing buffers. Based on this block, the diagram can be divided into four main parts: upper-left, upper-right, lower-left and lower-right parts. These four parts are Local-Capture, Local-Playback,

Network-Capture and Network-Playback. The Local-Capture and Local-Playback parts are optional, they provide local capture/playback capabilities that may be not be necessary.

The function of the Local-Capture part is to capture voice from the microphone and send it into the mixer for mixing. A LocalDataSource object is used. This LocalDataSource object may be the JMF capture part or the enhanced JMF capture part.

The function of the Local-Playback part is to playback the mixed voice output from the mixer. A LocalDataSink object is used. This LocalDataSink object may be the JMF playback part or the enhanced JMF playback part.

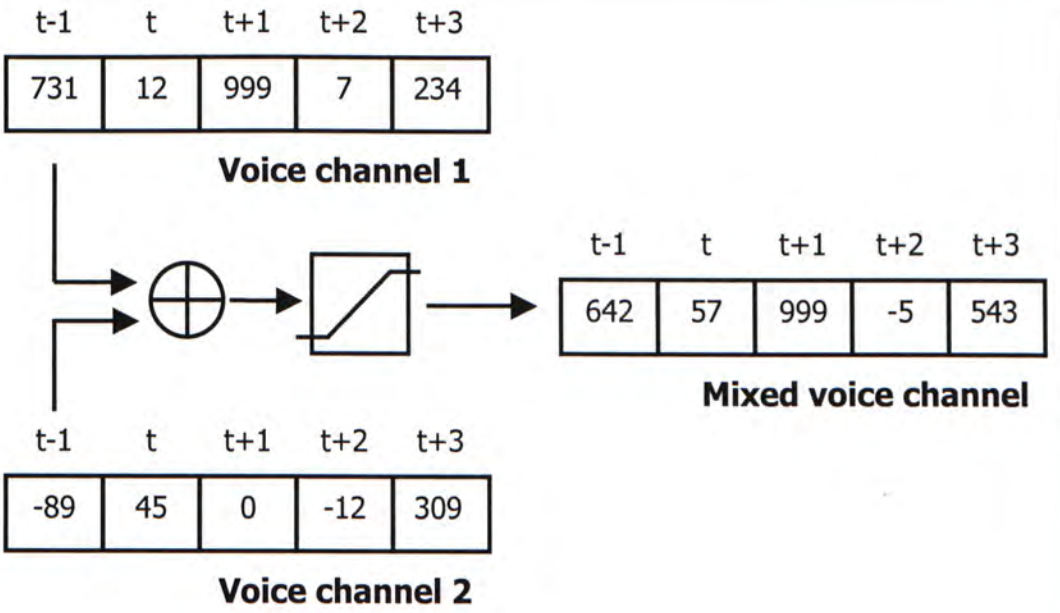
The function of the Network-Capture part is to receive the voice channel transmitted from the network and direct it into the mixer for mixing. This part is similar to the capture part in the JMF VoIP clients described in last few sections.

The function of the Network-Playback part is to playback the mixed voice channel to the network for remote archiving or listening. This part is similar to the playback part in the JMF VoIP clients described in last few sections.

The core of the mixer is an array of buffers and a running thread. The array of buffers is to buffer the voice data received either from network or local microphone to be mixed. Buffering is needed because there may be jittering in the incoming voice stream(s).

The function of the running thread is to perform the mixing mechanism. Mixing is done by adding the voice channels' amplitude values together.

The mixing mechanism is shown below.



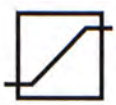
Example

There are two steps in calculating the mixed value for time t-1.

Step 1: ⊕ Addition

Add the value of channel 1 in time t-1 and the value of channel 2 in time t-1

731 + (-89) = 642

Step 2:  Bounding

For 8-bit, bound the sum between [0, 255]
For 16-bit, bound the sum between [32767, -32768]

Assume we are mixing in 16-bit.

As 642 is between 32767, -32768, therefore there is no need to adjust the value.
The mixed value is 642 for t-1.

If the sum is 39000, then the value is bounded to 32767.

Figure 3-20: The voice mixing mechanism

Bounding is needed in order to store the sum to the possible range that can be represented by each data unit in voice buffers. In 8-bit mixing, the size of each voice

data unit is one unsigned byte, as a result the possible range is [0, 255]. In 16-bit mixing, the size of each voice amplitude data unit is two signed bytes, as a result the possible range is [32767, -32768]. The mixing in the prototype mixing server uses 16-bit mixing as the raw voice data is captured and transmitted in 16-bit.

If there is too frequent adjustment of the sum in the bounding step for the incoming voice channels, then an effect called clipping occurred. Clipping reduces the quality of the mixed voice due to the too large volume of the input voice channel or too many voice channels to be mixed. To solve this problem, the volume of the incoming voice channels should be reduced to avoid the sum to be underflow or overflow the possible range. Or the number of channels to be mixed should be reduced.

3.6.2. Experiment

A simple experiment is created to evaluate the performance of the mixing server. The setup of the experiment is shown in the following diagram.

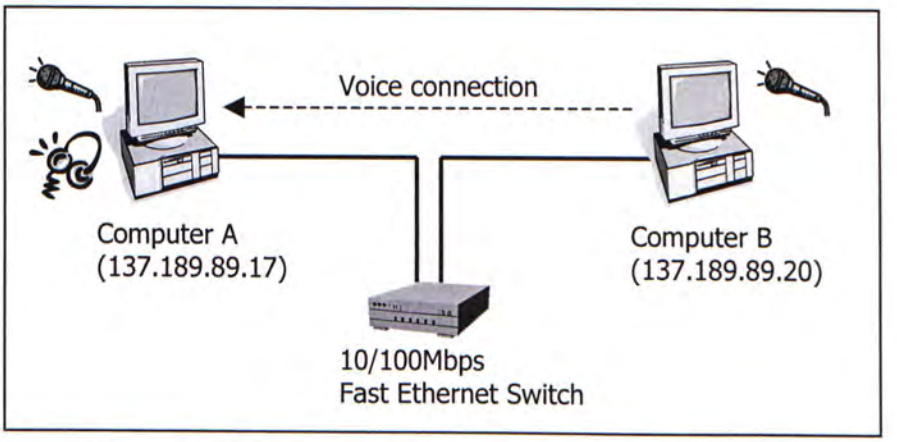


Figure 3-21: The setup of prototype mixer experiment

The mixing server is run in computer A with local capture and local playback enabled. A voice stream captured from microphone in computer B is sent to computer A for mixing. The mixing server mixed the voice from local microphone and remote voice stream. After mixing, the mixed voice is played at the local speaker. The mixing voice latency is measured.

The latency from the voice input at Computer A microphone to the mixed voice output at Computer A Speakers is 0.2 second. The latency from the voice input at Computer B microphone to the mixed voice output at Computer A Speakers is about 1 second. The latency for the latter case is higher because the voice has to go through Media Encoding, Media Transportation and Media Decoding processes before the voice is mixed at the server. In contrast, the voice inputted in Computer A is not encoded/decoded or transported over the network, therefore the latency is lower.

3.6.3. Conclusion

To conclude, a prototype real-time voice stream mixing server is presented. The experiment shows that it is feasible to perform voice stream mixing using software instead of hardware. However, it should be noted that each incoming voice from the network for mixing needs a decoder to decode the encoded voice. As a result, if there is a lot of incoming voice from network to be mixed, the CPU demand of the mixing server will be high because a lot of decoders are needed.

4. EXPERIMENTAL STUDIES

In this chapter, two implemented prototype VoIP systems are introduced. These two VoIP systems are both simple VoIP-based call centers. They use the VoIP core components discussed in last chapter for voice communication. These two prototype call center examples show that the idea of VoIP call center is feasible.

4.1. Pure IP-side VoIP-based Call Center – VoIP in Education

The Pure IP-side VoIP-based call center prototype, VoIP in Education, is an IP-side only VoIP call center. The main purpose of this prototype call center is to allow students to communicate with their tutors/lecturers using VoIP. As a result, students can study at their own pace. When they have any problems, they can ask their tutors/lecturers through VoIP. As a result, this scenario is a call center that is specifically used in an education environment.

There are two important elements in this call-center: VoIP and collaborative browser. VoIP provides two-way voice communication. Collaborative browser provides the same browser window to both sides for working collaboratively. The collaborative browser is constructed by another member in the VoIP group.

4.1.1. Architecture

In this call center scenario, there are four main components, namely the students with their computers, the lecturer/tutors with their computers, an Intranet and a Voice Connection Server.

The connections between these components are shown in the following figure.

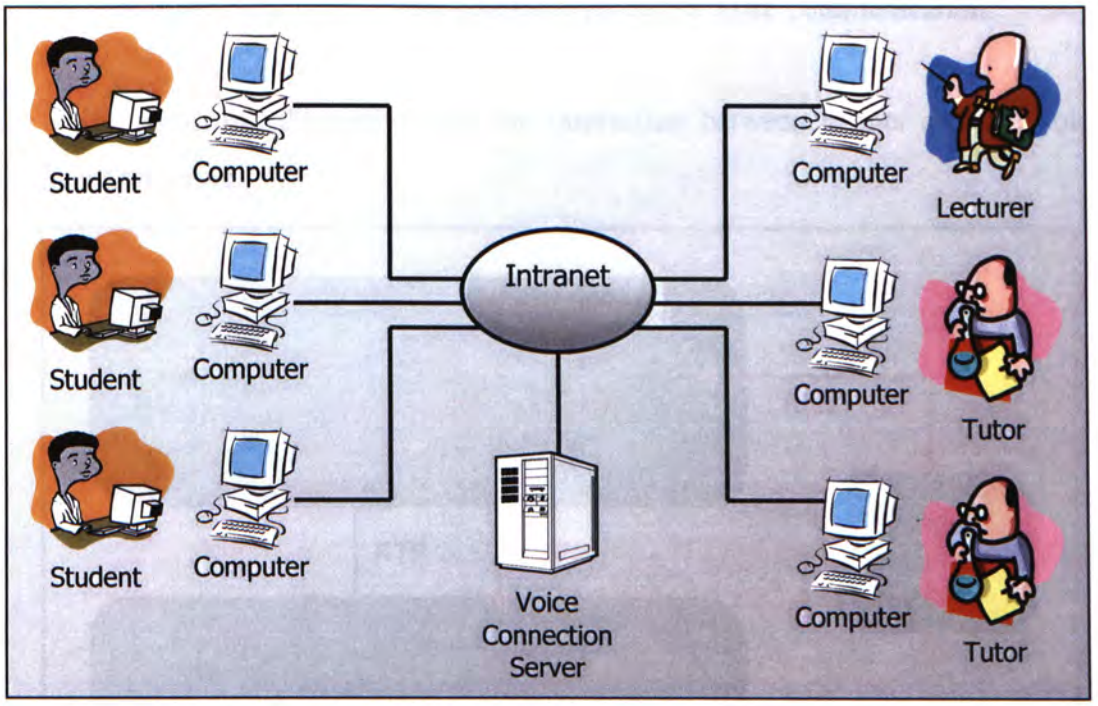


Figure 4-1: The VoIP in Education application model

In the following section, a brief introduction to the client programs is given. Client programs are programs that run on students' computer or tutors'/lecturers' computer.

4.1.2. Client Structure

There are two types of client program in this call-center application: the student client program, the tutor/teacher client program. Students use the student client program and the tutor/teacher uses tutor/teacher client program. These two client programs are similar in nature although there are some minor differences. By using Java class inheritance, the two client programs can be constructed quickly without duplication of code.

The clients are Java applets running in one of the three frames in a browser window. One of the other browser frames provides control to collaborative browser constructed by another group member in our VoIP group [39]. The remaining browser frame allows the user to view web information.

The client program applets provide simple queue control and status reporting. VoIP core is included in the client program applets for VoIP communication.

The following diagram shows the interaction between clients and the voice connection server.

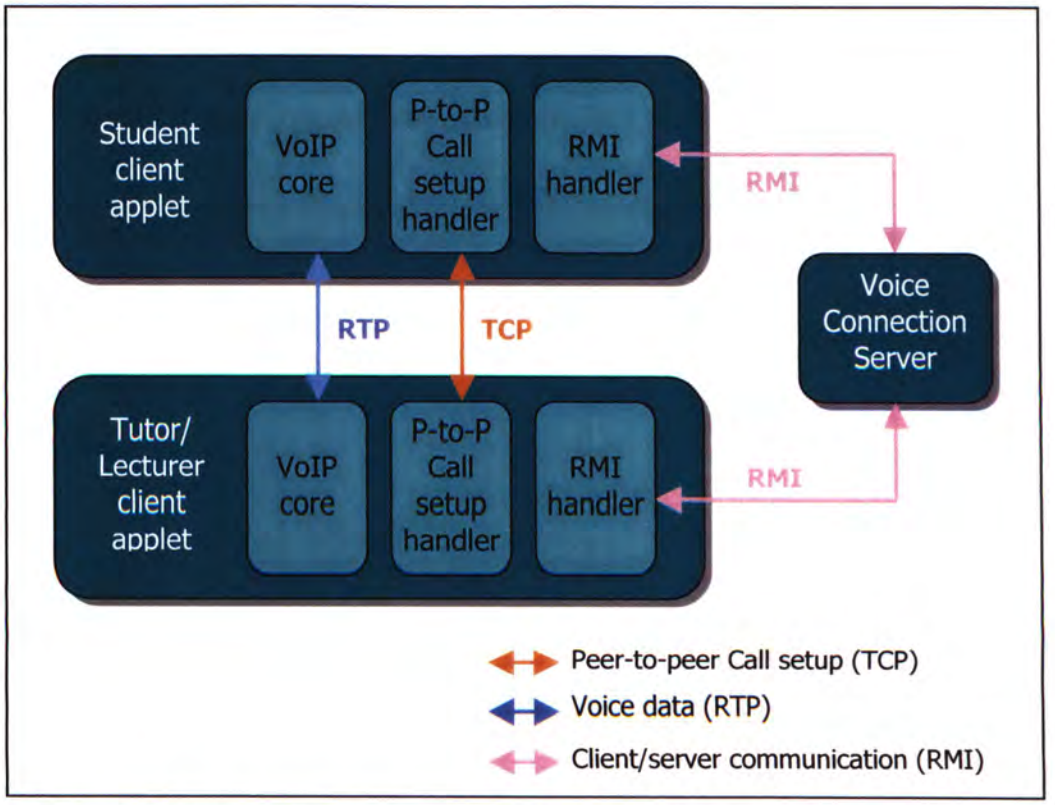


Figure 4-2: The interaction between clients and the voice connection server

First, either the student or the tutor/teacher uses the client applet to login to the system. The client applet uses Java Remote Method Invocation (RMI) to send the information to the voice connection server. After that, students may contact the tutor/teacher to ask questions. Their client applets first setup a voice connection by communicate with each other using simple TCP packets. After that, a VoIP communication channel is setup using RTP.

The reason for using TCP instead of RMI in making peer-to-peer call setup is because it is simpler to use TCP in this scenario. If we use RMI to do peer-to-peer call setup, each side has to execute one more Java utility called RMIRegistry for registering the call setup remote methods that are available to other machines. In any

case, it is not difficult to change the peer-to-peer call setup to use RMI instead of TCP.

The mechanism of the voice connection server and how to schedule the connections between student and tutor/teacher is discussed in detail in [40] that is published by another member in the VoIP group.

4.1.3. Client Applet User Interface

The following two figures show the user interface of the student client program.

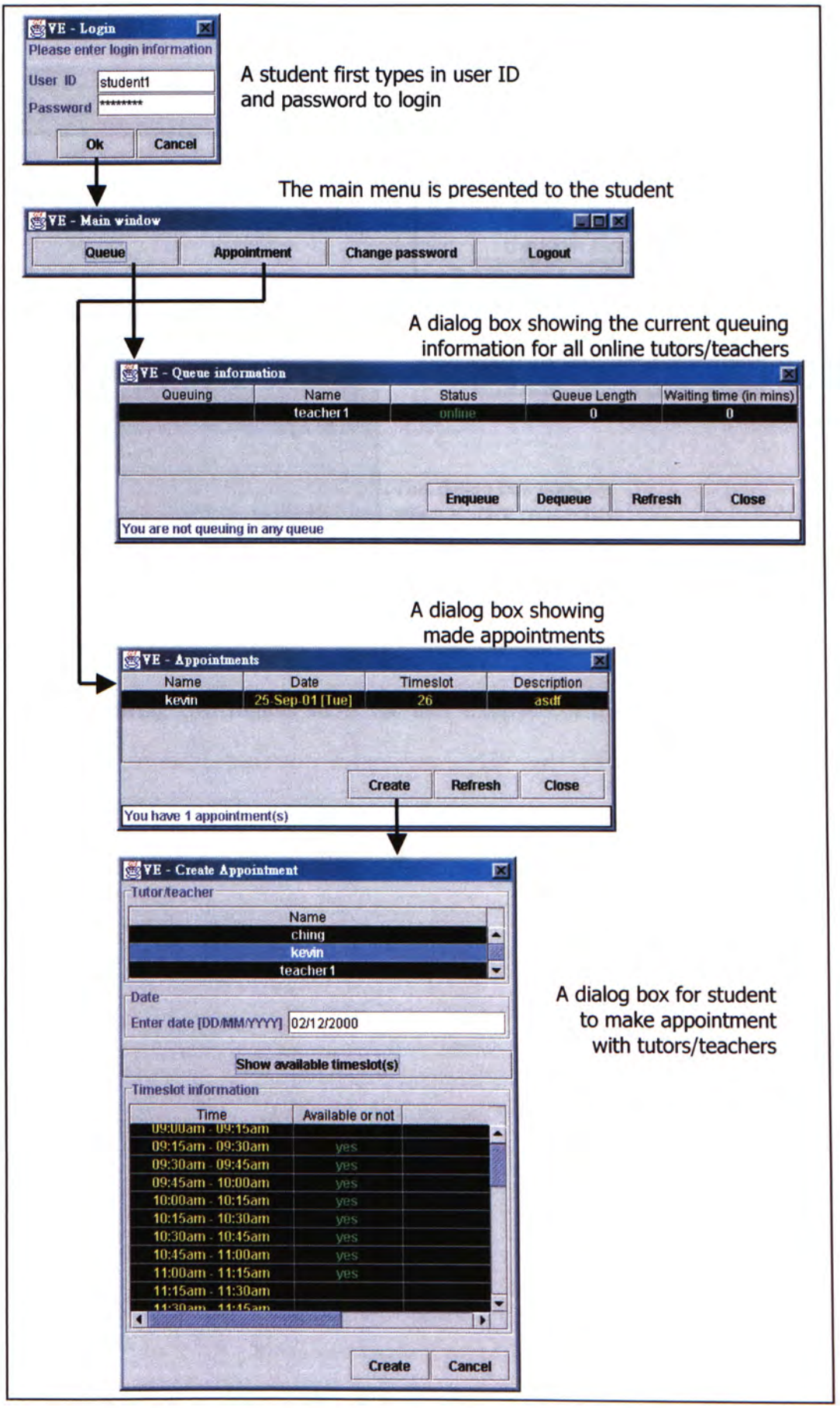


Figure 4-3: Client applet user interface (Student)

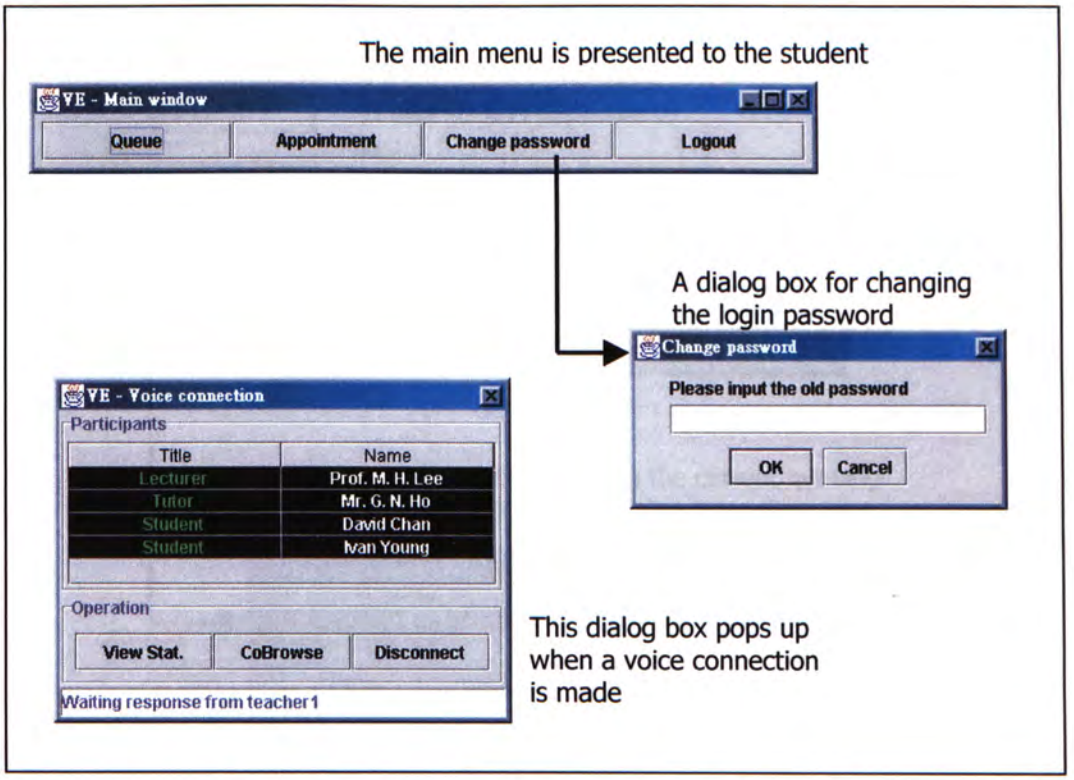


Figure 4-4: Client applet user interface (Student)

The following two figures show the user interface of the tutor/teacher client program.

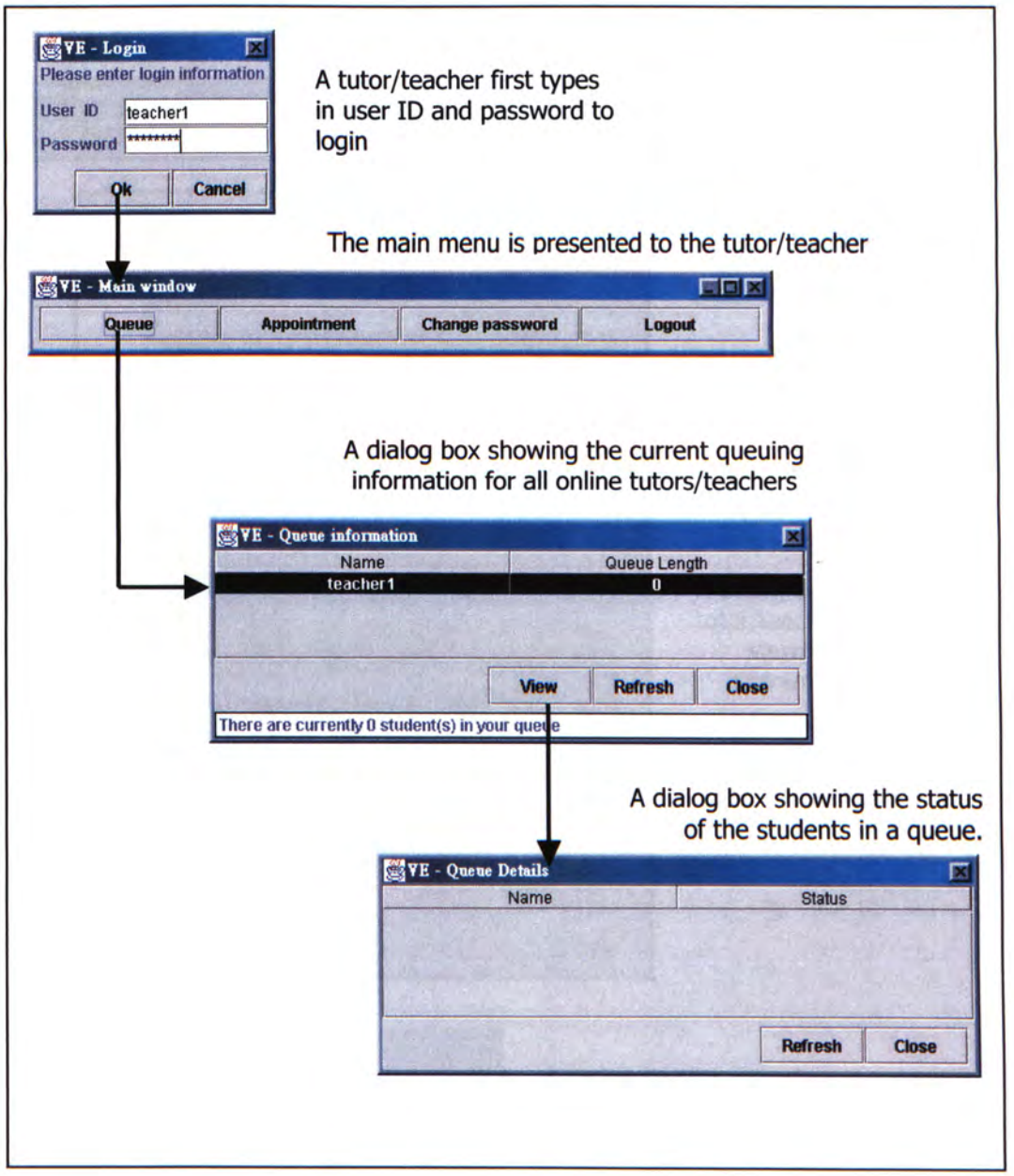


Figure 4-5: Client applet user interface (Tutor/Teacher)

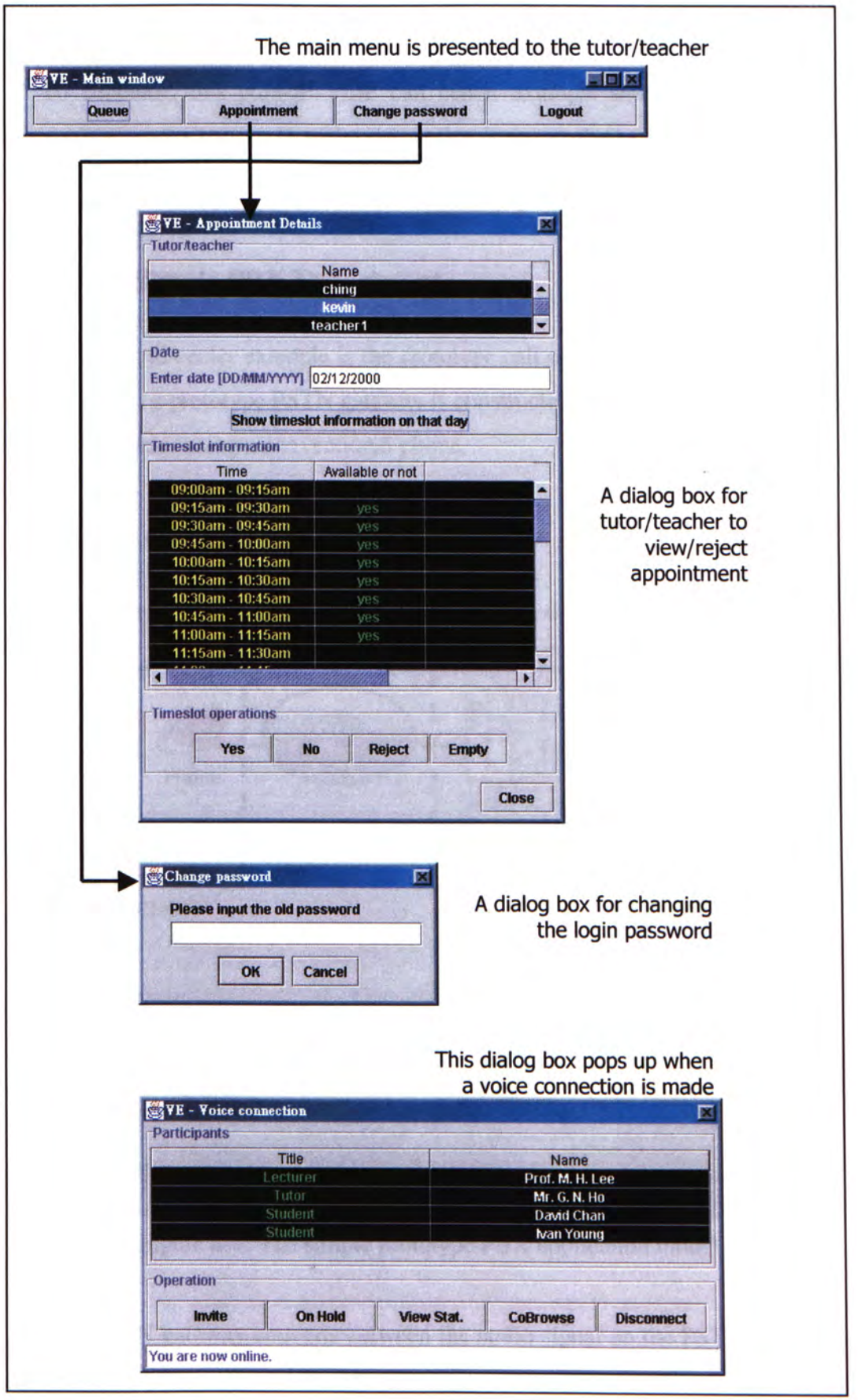


Figure 4-6: Client applet user interface (Tutor/Teacher)

4.1.4. Observations

To conclude, this pure-IP VoIP call center example shows that VoIP voice communication can cooperate with other technologies such as collaborative browser to provide useful and innovative application.

4.2. A Simple PBX Experiment

Another call center example is the prototype call center with PSTN gateway. In this call center, a prototype PSTN gateway is constructed that allows customer to dial into the call center using a conventional phone.

4.2.1. Structural Overview

The structure of the prototype VoIP call center is shown as follows.

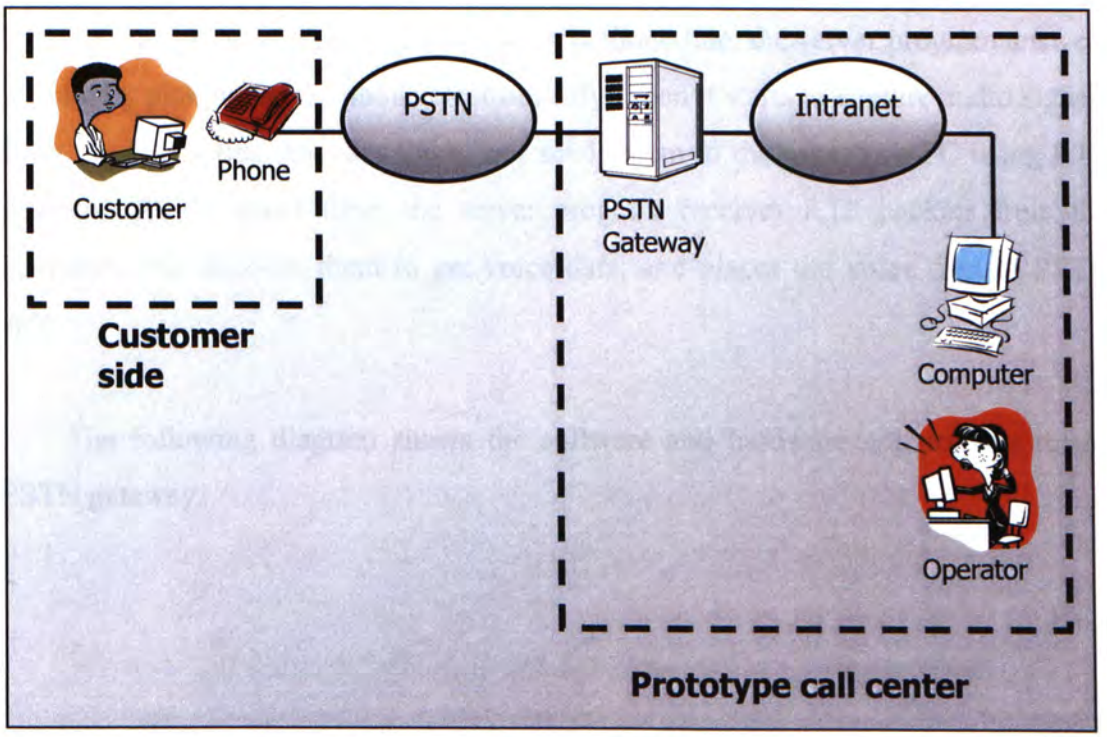


Figure 4-7: The simple prototype PBX application model

The PSTN gateway converts between the audio signal on the PSTN side and IP voice packets on the IP side. In this prototype call center, no call setup mechanism such as H.323 is used. All incoming phone calls from the PSTN side are directed to the same operator. The PSTN gateway is a PIII computer equipped with a Natural

Microsystem telephony card AG2000 with at most four analog telephone lines connected to it. In the prototype call center scenario, the telephony card has only one analog telephone line connected to it.

By replacing the four-port analog telephony card with a more powerful one such as those telephony cards that have E1/T1 interface, more simultaneous incoming connections can be made.

The analog telephony card supports both public analog telephone line and PBX analog telephone line in our university telephone network.

4.2.2. PSTN Gateway Server Program

In the PSTN gateway, a server program is executed to wait for an incoming phone connection from PSTN network. If an incoming PSTN connection is detected by loop start protocol signaling in analog telephone line, the server program answers the call by picking up the phone electronically. Then it starts to capture audio signals from the PSTN line, encodes them, and sends them to the operator's PC using RTP protocol. At the same time, the server program receives RTP packets from the operator's PC, decodes them to get voice data, and places the voice data to PSTN side.

The following diagram shows the software and hardware components in the PSTN gateway.

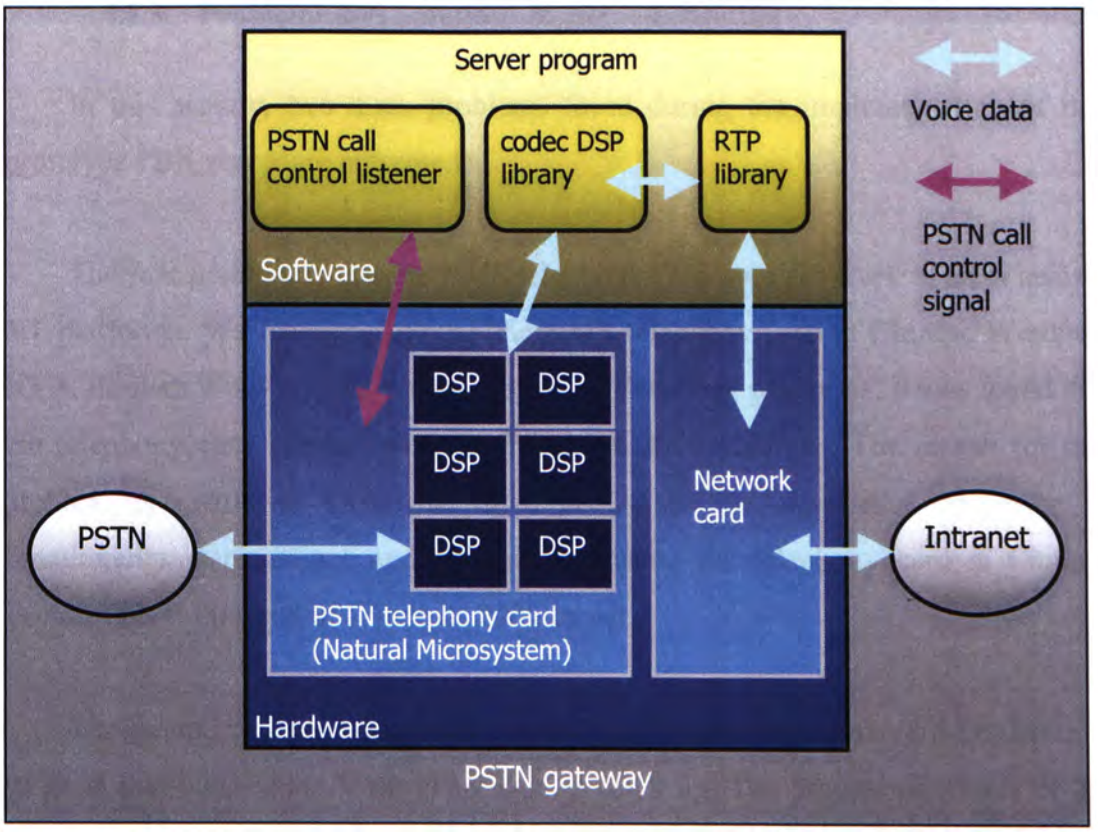


Figure 4-8: The software and hardware components of the PSTN gateway

For incoming voice data from PSTN, the voice data is encoded by using telephony card on-board hardware DSP. The encoded voice data is then transferred to network card by the gateway software. The network card then sends the voice data to the Intranet.

For incoming voice data from the Intranet, the data is captured by the network card on the PSTN gateway. The data is then passed to DSP for decoding into linear PCM voice data. The linear PCM voice data is then put to the PSTN analog line.

The gateway software uses call control API ADI [41] or NCC (Natural Call Control) [42] to monitor incoming PSTN call. When an incoming PSTN call is detected, the gateway software initiate a voice channel between PSTN and Intranet.

The DSP on the telephony card provides voice encoding/decoding in hardware that is much faster and less CPU-intensive. The TRAU API [43] is needed to control the DSP in the telephony card.

4.2.3. Problems and Solutions in Implementation

In this section, two main problems faced during the implementation of this prototype PBX server are presented.

The first problem is that the NMS telephony card does not work in all Windows NT platforms. We have tried to use the NMS telephony card in Chinese Windows NT 4, English Windows NT 4 and Windows 2000 server platforms. It was found that the telephony card couldn't work in Windows 2000 platform. The reason for this problem is unknown. However, there should be no significant difference in functionality, performance and reliability in using the telephony card in Chinese Windows NT 4 instead of Windows 2000 server.

The second problem is that the performance of the telephony card hardware is so good that our IP-side VoIP client cannot catch up. The processing power of the telephony card is good enough to conduct VoIP in 20 ms voice frame rate for both incoming and outgoing voice stream. However, the fastest voice frame rate for all VoIP clients are 20 ms and 40 ms for incoming and outgoing voice stream respectively. The reason for the fastest rate for outgoing voice stream is only 40 ms but not 20 ms may be due to the fact that Media Capture process is not fast enough using Windows Multimedia API. The rate may be increased by using DirectSound because DirectSound accesses multimedia hardware directly. However, the voice frame rate of the telephony card for incoming and outgoing voice stream must be the same. As a result, we have developed one solution to allow the telephony card to work in 20 ms voice frame rate while the IP side can work with the card. The solution is to use hybrid frequency for incoming and outgoing voice stream. The C++ VoIP client is modified to support hybrid frequency. The C++ VoIP client is able to output voice stream in its best 40 ms voice frame rate while accept incoming voice stream in 20 ms voice frame rate sent by the telephony card from the network.

4.2.4. Experiment 1

An experiment is done to evaluate the performance of the PSTN gateway. The gateway is loaded with the server program. A phone call is made to the PSTN

gateway to setup a voice connection between a phone and one operator's PC. The configuration of the computers and network is shown in the following table.

IP address	137.189.89.16	137.189.89.17
Platform	WinNT 4	Win98
Encoding codec	GSM 8000 Hz, 8 bit, mono	GSM 8000 Hz, 8 bit, mono
RTP frame size	20 ms	40 ms
Program	PSTN gateway program	C++ VoIP client

Table 4-1: Configuration of the experiment

The following graph shows the bandwidth consumption from the operator's PC.

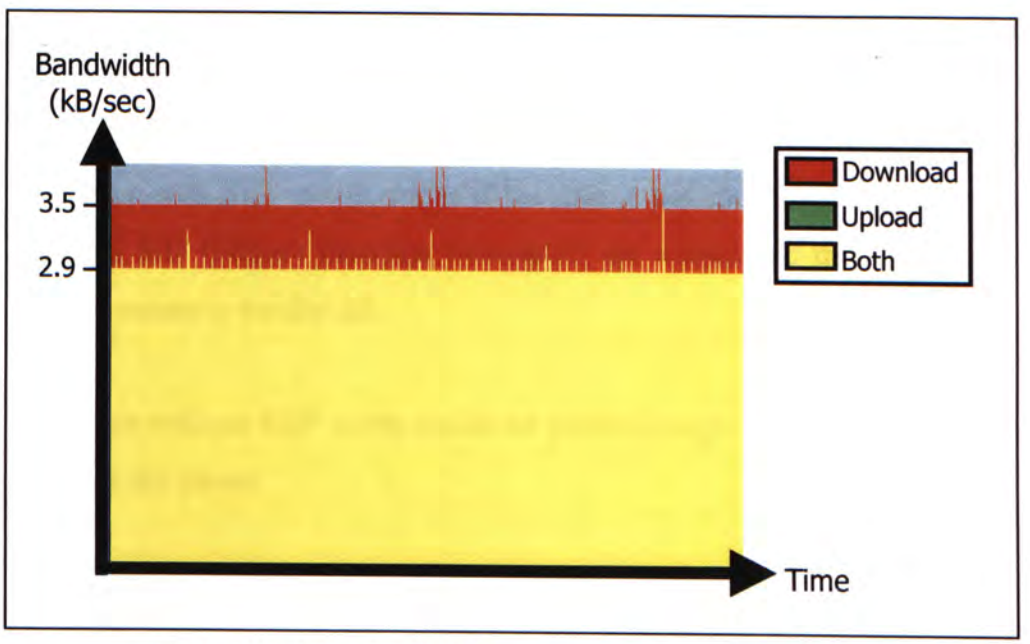


Figure 4-9: The bandwidth consumption of the gateway

It shows that the bandwidth from the PSTN gateway to the operator's PC is 3.5 kB/sec. The bandwidth from operator's PC to PSTN gateway is 2.9 kB/sec. The bandwidth from operator's PC to PSTN gateway is lower because the call operator's PC C++ VoIP client is operated in 40ms instead of 20ms.

For the latency, the latency of both sides is shown in the following table.

	Latency
From PSTN gateway	< 0.2 s
To PSTN gateway	< 0.2 s

Table 4-2: Latency of voice transmission

It shows that the latency of the voice connection is very low. As a result, the responsiveness of the voice connection is comparable with conventional PSTN voice connection.

4.2.5. Experiment 2

The second experiment is to test the silence suppression function in the PSTN gateway. There are three RTP packet transmission modes in the gateway server: "All packets", "Silence with SID", "Silence without SID".

"All packets" mode means that RTP packets are transmitted from the server even if the user is not speaking.

"Silence with SID" mode means if the user is not speaking, only some special packets call SID (Silence Insertion Descriptor) are transmitted in a regular interval (120ms) to conserve bandwidth.

"Silence without SID" mode means no packet is sent at all when the user is not speaking on the phone.

Our programs can work with "All packets" mode only. In order for our programs to work with "Silence with SID" and "Silence without SID" mode, we have to modify our programs. The most important modification is to transmit the voice packet to headphone based on timestamp instead of sequence number.

The experiment setup is the same as the one in experiment 1. The main difference is that the silence suppression function is switched to different modes in this experiment.

The following diagram shows the bandwidth consumption measured in the operator's computer. The RTP packet transmission mode is "Silence without SID" mode.

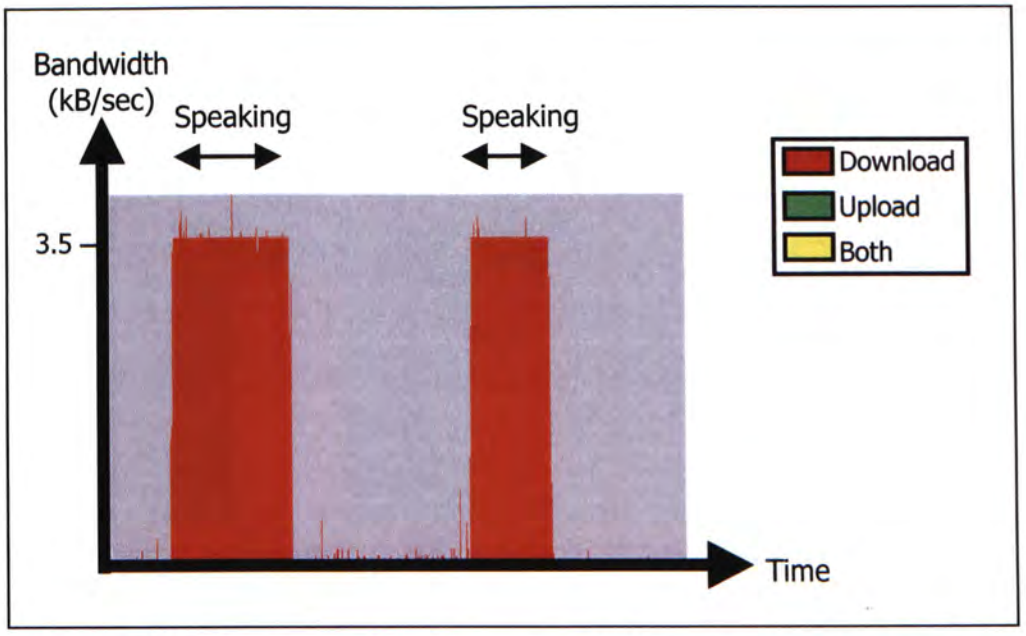


Figure 4-10: The bandwidth consumption of "Silence without SID" mode

The red blocks correspond to the bandwidth consumed by the sent RTP packets as the user is speaking at that time. When the user is speaking, the bandwidth consumed is 3.5 kB/sec. When the user is not speaking, no SID packet is sent, therefore the bandwidth consumption is zero. In the diagram, there is still some tiny bandwidth consumption even when the user is not speaking. The tiny bandwidth consumption is made by other programs or the OS in the system. As a result, by using this packet transmission mode, a lot of bandwidth is saved when the user is not speaking.

The following diagram shows the bandwidth consumption measured in the operator's computer. The RTP packet transmission mode is "Silence with SID" mode.

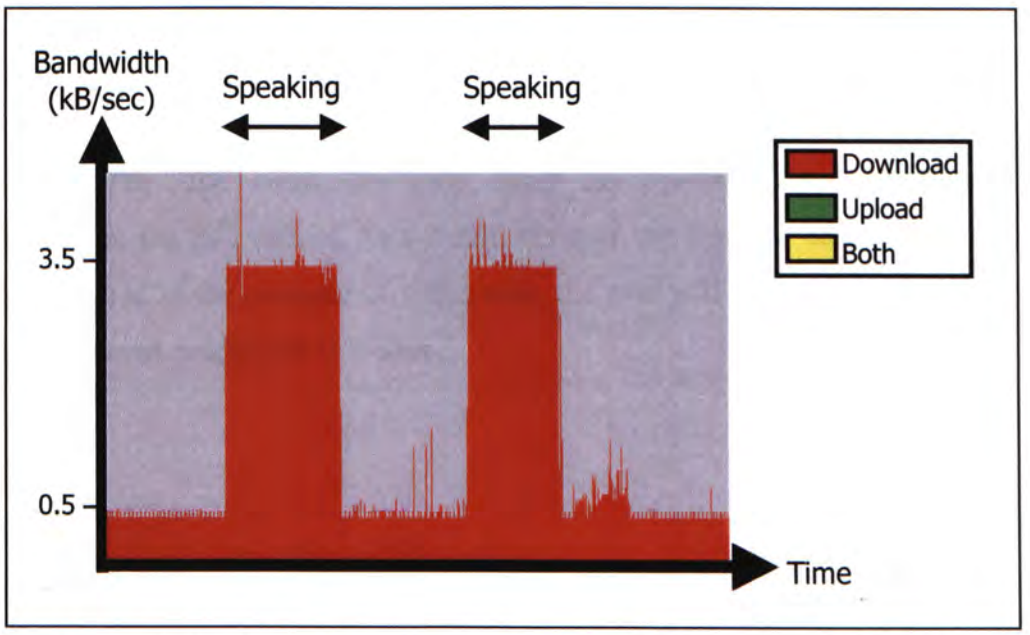


Figure 4-11: The bandwidth consumption of "Silence with SID" mode

It shows that when the user is not speaking, SID packets are transmitted. To measure how frequently SID packets are sent, a program is written to monitor the incoming RTP connection. It shows that the size of the SID packet is the same as normal RTP packet. The SID packet is transmitted in 120 ms interval. As a result, the packet sending rate is six times smaller than when the user is speaking. The normal bandwidth is 3.5 kB/sec. Divide 3.5 kB/s by 6 gives 0.58 kB/s. As a result, the bandwidth monitor program DU Meter accurately shows that the bandwidth consumed by SID packets is around 0.5 kB/s to 0.6 kB/s.

From the experiment, it shows that it is feasible to use a PSTN gateway to connect PSTN network and IP network to allow voice connection between them. The latency of the PSTN gateway is low.

Moreover, if Voice Activation Detection (VAD) is used, the bandwidth consumption of the voice connection may be further reduced.

To improve the VoIP performance, we should modify our VoIP client to work in "Silence with SID" mode or "Silence without SID" mode.

The main difference between "Silence with SID" and "Silence without SID" mode is that the bandwidth consumption of "Silence without SID" mode is smaller than the "Silence with SID" mode. However, by receiving an SID packet in the "Silence with SID" mode, the VoIP client can playback the background noise described in the SID packet. As a result, the user can hear the background noise of the other side of the connection. Otherwise, the user will hear complete silence and think the voice connection is broken.

5. A COMPREHENSIVE VOIP PROJECT – GRADUATE SECOND PHONE (GSP)

The two prototype VoIP applications mentioned in last chapter serve as cornerstones for the prototype VoIP application presented in this chapter. The prototype VoIP application mentioned in this chapter is called Graduate Second Phone (GSP). The GSP project will be discussed in details in this chapter.

5.1. Overview

Before going into depth to discuss the inner workings of the GSP project, this section gives an overview of different aspects of the project. First, we will look at the background of this project, including the rationales justifying this project, the reasons why this project is proposed, and the problems to be solved by this project, etc. Then, an overall architecture of the GSP project is presented in section 5.1.2. As there are various technologies used in this project, an overview of these is presented in section 5.1.3. Finally, the major functions of the GSP project are outlined in section 5.1.4 to give readers a brief understanding of what functions are available in this GSP project.

5.1.1. Background

The telephone is one of the most prevalent means of communication nowadays. We have mobile phones, phones in the home and phones in the office. In our department office, the telephone plays an important role of communication between undergraduate students, graduate students and lecturers. By using telephones, students can contact their tutors to ask questions about course work; supervisors can contact graduate students to discuss about research matters. However, it is common that there are not enough telephones for the graduate students in our department office.

For example, the following figure depicts one of the rooms in our department office -- Room 1005.

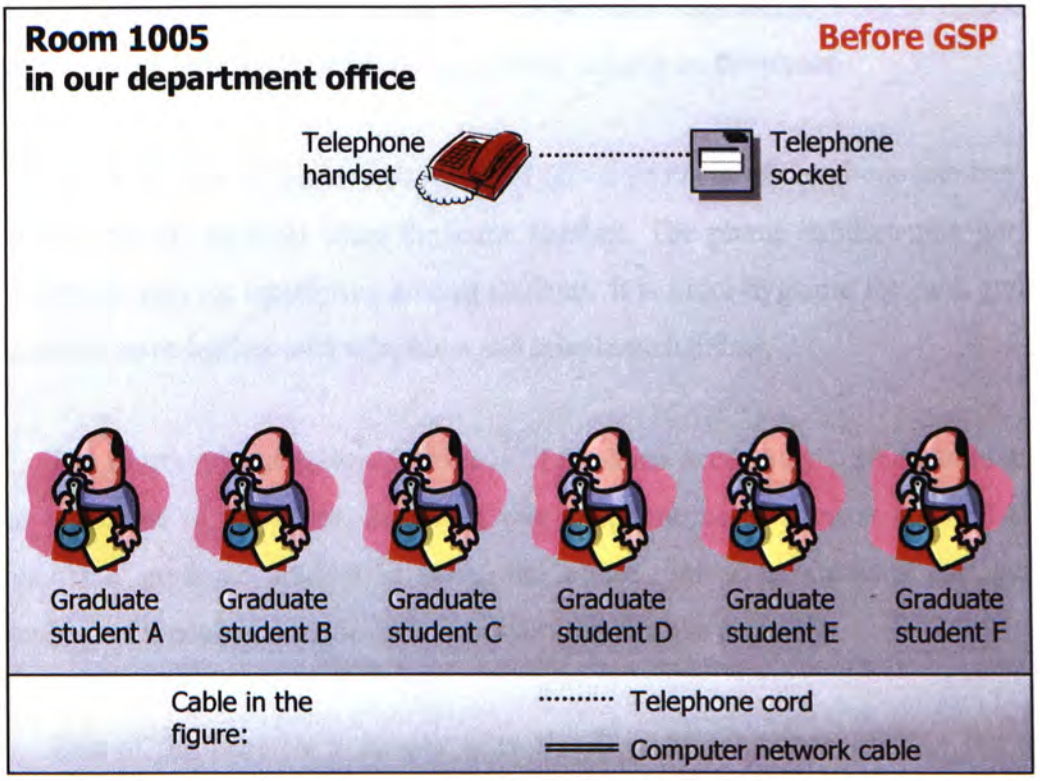


Figure 5-1: The phone and graduate students in Room 1005

As shown in the figure, there are six graduate students in Room 1005. However, there is only one telephone in that room. As a result, it is not possible for each graduate student to have his/her own telephone on the desk.

This raises four types of inconvenience for the graduate students in Room 1005: "Forced call operator", "Remoteness of the telephone from desk", "Hygiene problem of the telephone headset" and "Telephone occupancy".

The first type of inconvenience is "Forced call operator". When there is an incoming phone call, one of the graduate students has to answer the phone call in person. The graduate student who sits nearest to the phone usually answers the phone call. Occasionally, this graduate student will have to answer many phone calls that disturbs his/her own work. The graduate student may not willing to be the call operator of the room.

The second type of inconvenience is "Remoteness of the phone from desk". Most of the graduate students do not sit near to the telephone. As a result, if they would like to access their desks when talking on the phone, they have to travel

between their desks and the phone. It is not possible for them to work in front of the desks (for example, operate computers) while talking on the phone.

The third type of inconvenience is "Hygiene problem of the phone handset". All the six graduate students share the same handset. The phone handset may get dirty and disease may be transferred among students. It is more hygienic for each graduate student to have his/her own telephone and telephone handset.

The fourth type of inconvenience is "Telephone occupancy". Since there is only one telephone in the room, thus only one telephone connection is allowed at one time. If a graduate student is using the phone, no other students can use the telephone. Of course, outside callers cannot dial into the room too.

One of the possible solutions is to install more telephone lines in the room. Nevertheless, there are two shortcomings. The first shortcoming is that budget is needed to install more telephone lines and telephone handsets. If the budget is tight, then it is not possible to support the installation and maintenance cost. Moreover, if there are more telephone lines, it may be necessary to have private branch exchange (PBX) facilities to provide functions such as call forward and call transfer. However, the price of PBX facilities is expensive. The second shortcoming is that sometimes it is not possible to install new telephone lines due to reasons such as office policies, geographical restrictions, etc. There are many similar situations such as teacher room and editorial room.

As all the graduate students in the room have computers on their desks and these computers are interconnected by an Intranet, a better solution is to employ voice over IP (VoIP) and the Intranet to solve the above problem. Therefore, the Graduate Second Phone (GSP) project is proposed. By using the GSP, every graduate student will have a "Second Phone" on the desk in addition to the one in the office.

The Graduate Second Phone project works as follows. Originally, the telephone line is connected to a telephony handset. Now, the telephone line is connected to a computer called Gateway. The Gateway computer is connected with other graduate

students' computers through Intranet. At the same time, special software called Client program is installed in all graduate students' computers. With the aid of the software, the computers can communicate with the public switch telephone network (PSTN) through the Gateway computer. As a result, the computers become virtual telephones that allow graduate students to make and receive phone calls without using the telephone handset in the room.

The following figure shows the telephone line connection after installing GSP system.

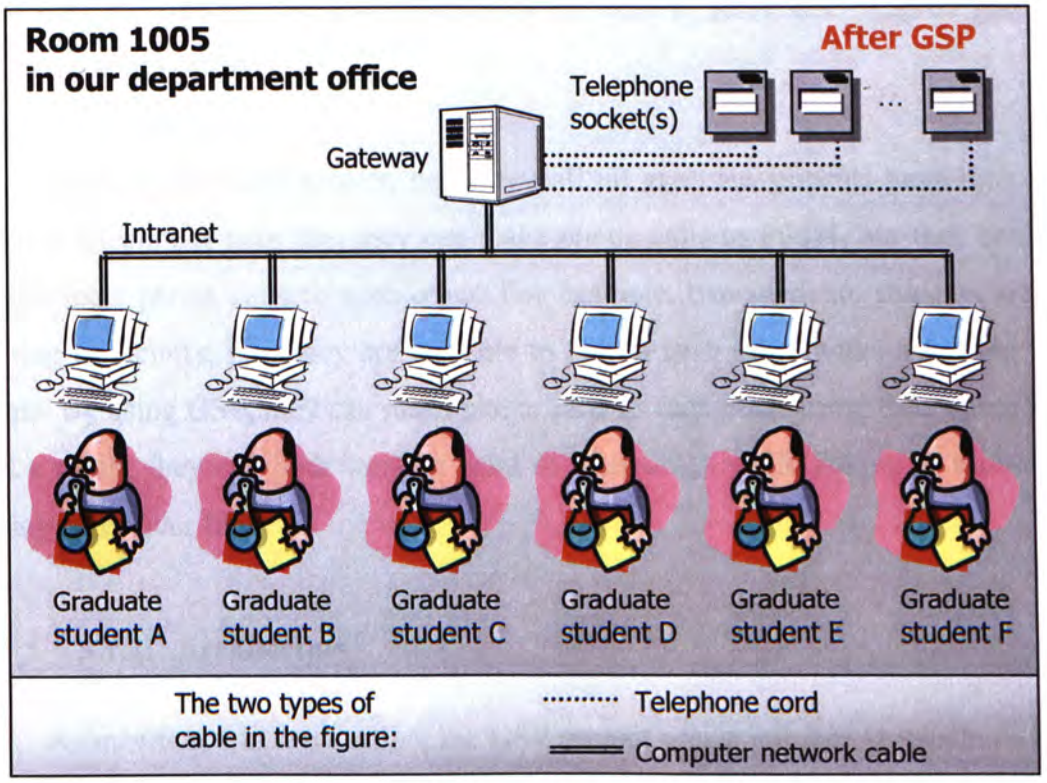


Figure 5-2: After GSP is installed

As shown in the figure, more than one telephone line can be connected to the Gateway. Instead of connecting telephone lines to the Gateway, E1/T1 digital trunk can also be connected to the Gateway. By using the GSP project, there are now six virtual telephones located in each graduate student's desk.

The GSP project solves the first three type of inconvenience. However, the fourth type of inconvenience – that is "Telephone occupancy", cannot be solved. In spite of this, the GSP project can be expanded to include more telephone lines. For

example, assume the number of telephone lines in Room 1005 has been increased to four. The four telephone lines can be all connected to the GSP system. As a result, four telephone lines are used to provide the six virtual telephones. By using more telephone lines in the GSP system, the users can dial in or dial out more easily.

The maximum number of telephone lines that can be connected to the GSP system depends on the hardware in the Gateway computer. By using suitable hardware, 24 or even 32 telephone lines may be connected to the system. This large number of telephone lines can serve rooms with a large number of graduate students. For example, the 32 telephone lines may be used to serve one hundred graduate students in the office.

By using the GSP project, not only will all graduate students have their own virtual telephones such that they can make phone calls to PSTN, but they can also make local phone calls to each other. For example, two graduate students are not sitting in vicinity, thus they are not able to talk to each other without leaving their seats. By using GSP, they can make phone calls to each other using their computers. As a result, they can both work in front of their desks while talking to each other using voice over IP.

5.1.2. Architecture

As mentioned in last section, the GSP project uses a number of telephone lines to provide virtual telephones to the graduate students. In this section, the GSP project components and the interconnection between them are outlined.

There are three main entities in the GSP project: Client, Gateway and Server. All the three entities contain both software and hardware components.

The following figure shows the architecture of the GSP system and the location of the three main entities.

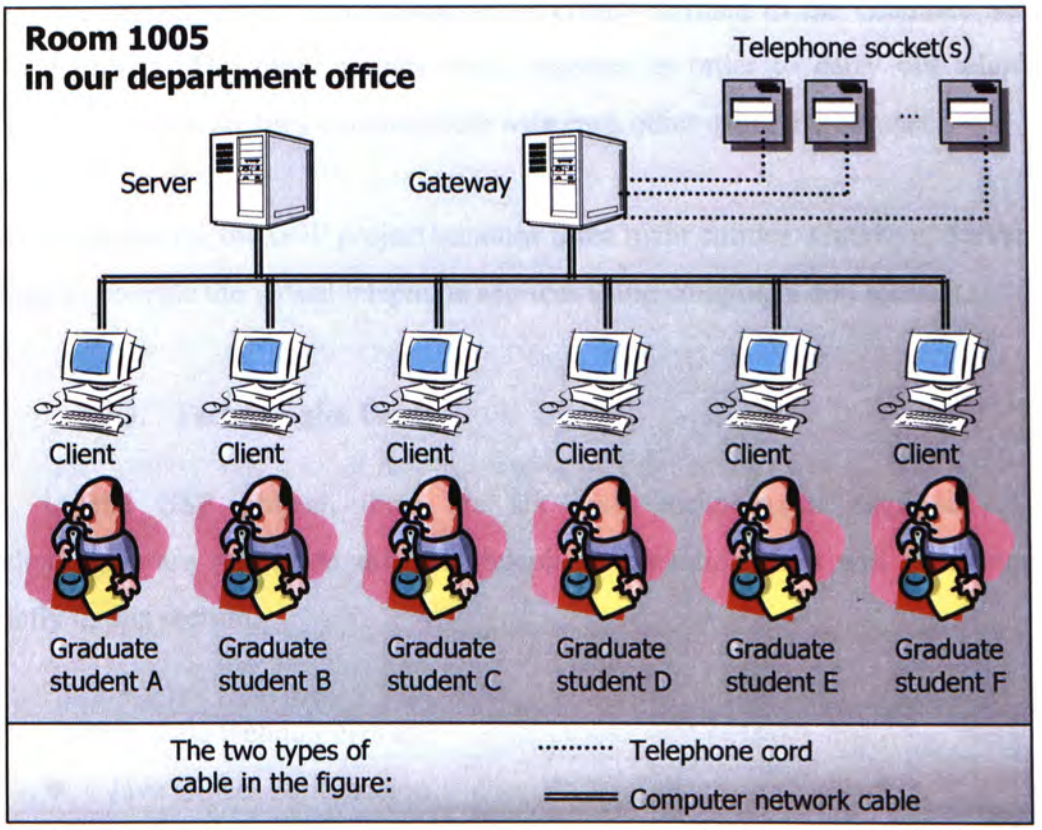


Figure 5-3: The architecture of the GSP system

The Gateway is used to connect the Intranet with the PSTN. The Gateway is equipped with computer telephony card for sending and receiving signals and data from the PSTN analog telephone lines or E1/T1 digital telephone lines. The voice signals flow through the Gateway when traveling between the graduate student and the PSTN network. Moreover, the Gateway asks the incoming caller from PSTN for which person he/she wishes to connect to.

The Client acts as a virtual telephone for a graduate student to carry out telephone operation and conversation. The Client uses the sound card in the computer to simulate the telephone. Moreover, it provides a user interface that resembles a typical telephone panel. Furthermore, the user interface provides additional functions such as call forwarding and call transfer.

The Server acts as a coordinator for the whole GSP system. It stores various information such as status, IP address of all Clients.

The role of the three main entities is clearly defined in the Graduate Second Phone project. The three entities work together in order to carry out telephone operations. To do so, they communicate with each other using the Intranet.

To conclude, the GSP project contains three main entities: Gateway, Server and Client to provide the virtual telephone services using computers and Intranet.

5.1.3. Technologies Used

In the GSP project, there are six main technologies employed. These technologies are related to different telephone operations and will be mentioned briefly in this section.

The six main technologies are:

- Java Object Serialization
- Remote Method Invocation (RMI)
- Java Native Interface (JNI)
- VoIP Core
- Microsoft Windows OS Infrastructure
- Natural Microsystem Telephony API

Java Object Serialization

Java Object Serialization is a method to store Java objects from main memory to hard disk or retrieve Java objects from hard disk to the main memory. This is because the high level components in GSP are implemented in Java, therefore it is more convenient to save the state of objects to hard disk by using Java Object Serialization. Otherwise, a large amount of code has to be written to perform data storing and retrieving between memory and hard disk.

The Java Object Serialization is responsible for storing persistent user and caller information such that the automatic call distributor can use these information and caller ID to direct incoming calls. Therefore, the proper functioning of automatic call distributor relies on the Java Object Serialization technology.

Remote Method Invocation (RMI)

Remote Method Invocation is a convenient way for distributed Java objects to communicate with each other through a computer network. It encapsulates low-level network communication into high-level Java method calls (invocation). As a result, there is no need to write low-level communication code when specifying the interaction between the three entities in the system. Instead, the interaction is specified as Java method calls between objects, which is easier to understand.

In summary, Remote Method Invocation is the communication protocol for the three entities, Client, Gateway and Server, such that they can provide telephone operations such as dialing, call forward and call transfer.

Java Native Interface (JNI)

Not all software components in GSP project are implemented in Java, some of the components are implemented in C++. Take Gateway entity as an example; the high-level call control is implemented in Java while the low-level interaction with the telephony card is implemented in C++. As a result, a way is needed to allow the software components written in Java to communicate with the software components written in C++. Java Native Interface is chosen for such communication because the communication is also method call based. Such communication method is similar to RMI, which is easy to use and understand. The main difference between JNI and RMI is that the communication in JNI is across language boundary while the communication in RMI is across Java Virtual Machine boundary.

The Java Native Interface enables telephone operations to be carried out by establishing a communication channel between C++ and Java layers such that high level telephone operation commands can be passed to the hardware telephony card.

VoIP Core

In order for the Client entity in GSP project to perform VoIP, the VoIP core software components mentioned in previous chapters are used. In this project, the DirectSound VoIP core software component is used. As a result, it shows that it is convenient to develop VoIP application just by integrating the modularized VoIP core into the application.

The VoIP core enables voice communication between the participants in the system.

Microsoft Windows OS Infrastructure

Another important technology that is used in the GSP project is the group of various APIs and infrastructure provided by the Microsoft Windows OS. For example, Windows inter-process messaging is used in order for processes to communicate with each other. Windows API is also used to set the priority of the threads.

The Microsoft Windows OS infrastructure provides the communication channel between VoIP core and other components in the Client such that the Client can perform telephone operations. Besides, it ensures the stability of the voice communication in the system by increasing the process scheduling priority of VoIP core.

Natural Microsystem Telephony API

In order for the GSP to control the telephony card in the Gateway entities, the telephony API provided by the card vendor is used. The card vendor of the telephony card is called Natural Microsystem. Natural Microsystem provides a number of APIs for controlling various functions in the telephony card. The Gateway entity uses the APIs to control the telephony card to carry out the function of the Gateway.

The Natural Microsystem Telephony API allows the high-level telephone operation commands to be passed to the hardware telephony card. Moreover, it allows the low-level telephone events to be passed to the above software layers.

5.1.4. Major Functions

There are five major functions in the GSP project.

1. Voice Connection between PSTN and IP Network
2. Intelligent Automatic Call Distribution
3. Dynamic Incoming Call Forwarding

4. Call Transfer
5. Support of Secure Local Voice Message Box

Voice Connection between PSTN and IP Network

The most important function in GSP is to enable voice connection between PSTN and the IP network. This is because computers and VoIP are used to create virtual telephones, therefore the most important function is to ensure conversation can be conducted between virtual telephones and conventional PSTN telephones. To do so, voice must be allowed to flow between PSTN and the Intranet IP network.

There are two types of voice connection: "Client to Client" and "Client to PSTN". The voice signals in "Client to Client" phone connection flows through the Intranet only. The signals do not need to traverse through the Gateway. However, the voice in "Client to PSTN" phone connection flows through both Intranet and PSTN. In such a phone connection, Gateway is needed to move the voice signals between the IP network and PSTN boundary. The GSP project must make sure that both types of voice connection are supported such that graduate students can talk to both IP and PSTN sides.

Intelligent Automatic Call Distribution

Whenever a person dials in, he/she would like to contact one of the graduate students in the office. Before using GSP, the one who answers the call would ask the caller whom he/she would like to contact. After using GSP, it is the Gateway, not a person that answers all incoming calls. Therefore, a way is needed for the Gateway to ask the person whom he/she would like to contact. The intelligent automatic call distributor in Gateway is responsible for such job. After an incoming call is connected to the Gateway, the automatic call distributor will ask the caller whom he/she likes to contact.

Dynamic Incoming Call Forwarding

It is possible for the graduate student that the caller tries to contact cannot answer the call. Possible reasons are that the graduate student may not be in the office, that he/she is talking to someone else, or even that the graduate student refuses to answer any phone call. As a result, there should be a way to forward the

incoming call directed to that student to other destinations. The destinations may be another graduate student in the office, a mobile phone number or even the graduate student's voice message box.

The incoming call forwarding should be dynamic. The destination of the call forwarding depends on the user's setting and the status of the user. Moreover, the forwarding process should be fast enough to avoid caller having the wait for a long time.

Call Transfer

Besides call forwarding, another useful telephone operation is call transfer. Call transfer allows the participants in a phone connection to transfer the call to another destination. The destination may be another graduate student or a PSTN phone number.

For example, as described in the following figure, the graduate student A and graduate student B are having a conversation with each other. During the conversation, graduate student B decides that it is more appropriate for graduate student A to talk to graduate student C. Instead of disconnecting the phone call between graduate student A and graduate B, and let this graduate student A contact graduate student C, graduate student B can use call transfer function to transfer the call to graduate student C. As a result, the graduate student A can talk to graduate student C instantly without go through the disconnection and re-connection process.

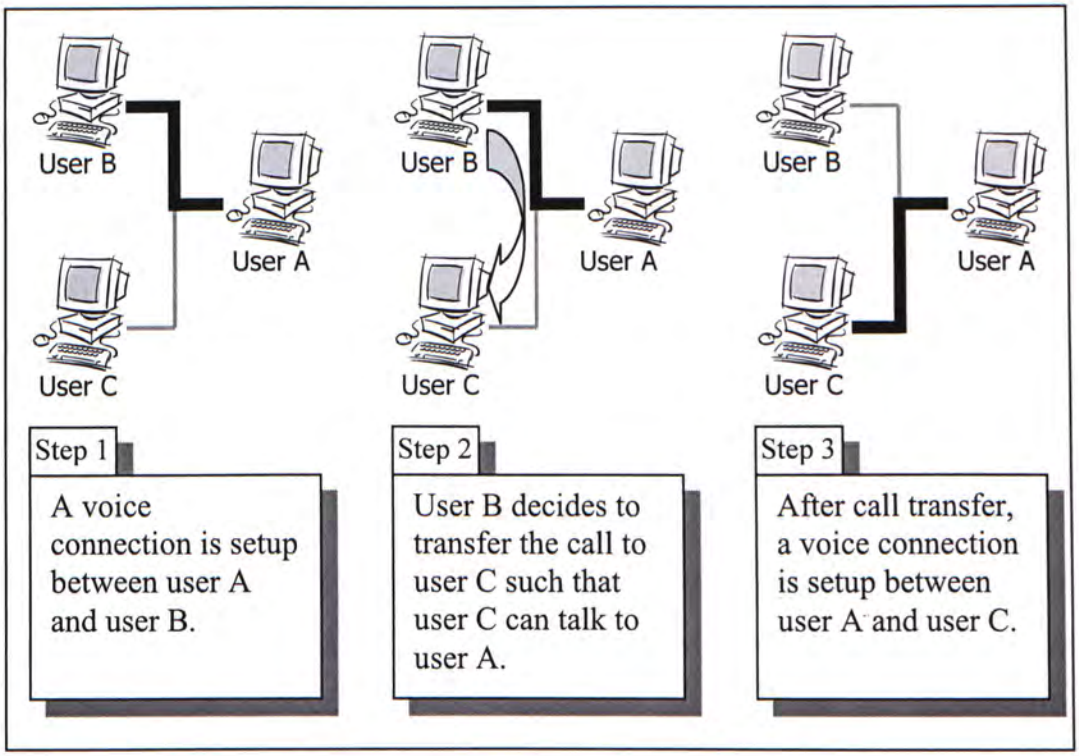


Figure 5-4: The mechanism of call transfer

Support of Secure Local Voice Message Box

The fifth major function of GSP is the support of secure local voice message box. Sometimes it is possible that there are phone calls directed to the graduate student and he/she is not in the room at the moment. Instead of forwarding the call to another graduate student, the system may ask the caller to leave a voice message to the student.

There are three possible locations to store voice messages: Gateway, a computer specialized for voice message storage and Client. For the first two options, all the voice messages for all graduate students are stored in the same computer. Hence, there is a risk that the voice messages will be lost if the computer crashes. Another risk is that as the voice messages are stored in a public area that all graduate students can access, it is possible that unauthorized users can read their voice messages. As a result, the best place to store voice messages is graduate students' own local computers. For example, voice messages for graduate student A are stored in graduate student A's computer, voice messages for graduate student B are stored in graduate student B's computer. By placing users' voice messages in their own

computers, the users of GSP can be guaranteed that other users cannot access their voice messages without authorization.

5.2. Client

The Client entity is discussed in details in this section.

The function of the Client entity is to provide virtual telephone using computer. To do so, the Client entity contains software code for interacting with Server and Gateway. In addition, it contains code to control multimedia facilities in the local computer. The Client entity is the main software component that interacts with GSP user. As a result, it gives GSP users the impression of how robust the system is and what functions the system provides.

The Client program is written using both Java and C++. The Client program runs on Win9X platform only. It is because the Client program contains a platform dependent VoIP core and code for communicating with VoIP core. It is possible to port the Client program to a Linux platform by performing modifications to the VoIP core and the code for communicating with it. The changes are minor and can be done within two to three days. As most of the machines in the department run on Win9X platform, the priority is therefore higher for the Client program to be able to run on Win9X platform.

There are three main external software components needed for the Client program to run on the computer: Java Development Kit (JDK) [44] or Java Runtime Environment (JRE) [45], DirectX (for the DirectX VoIP core) and a media playing utility that is capable to play sound file in Sun Microsystem's audio format (extension .au) [46]. JDK v1.3 or JRE v1.3 is needed to run the main Client program. In order for the DirectSound VoIP core to run, the Microsoft DirectX package hs to be installed into the computer before executing the Client program. Finally, a media playing utility is needed to play voice message file in Sun Microsystem's audio format. The common choices for such media playing utility are WinAmp [47] and Windows Media Player [48].

The Client program has to be installed before it can be executed. The installation process places the Client program files to suitable directories in the computer. An alternative approach is to place the Client program on a web server. To execute the Client program, the Client program files are loaded from the remote web server on demand. The advantage of this approach is that there is no need to redo installation in every computer when the Client program is updated or modified. However, in order for the program to be downloaded and run from the web server on demand automatically, some settings and additional Java coding have to be done. As the GSP system is now a prototype only, the former approach (installation required) is adopted.

5.2.1. Structure Overview

The Client consists of several software components that are responsible for different functions. The software components inside the Client are shown in the following diagram.

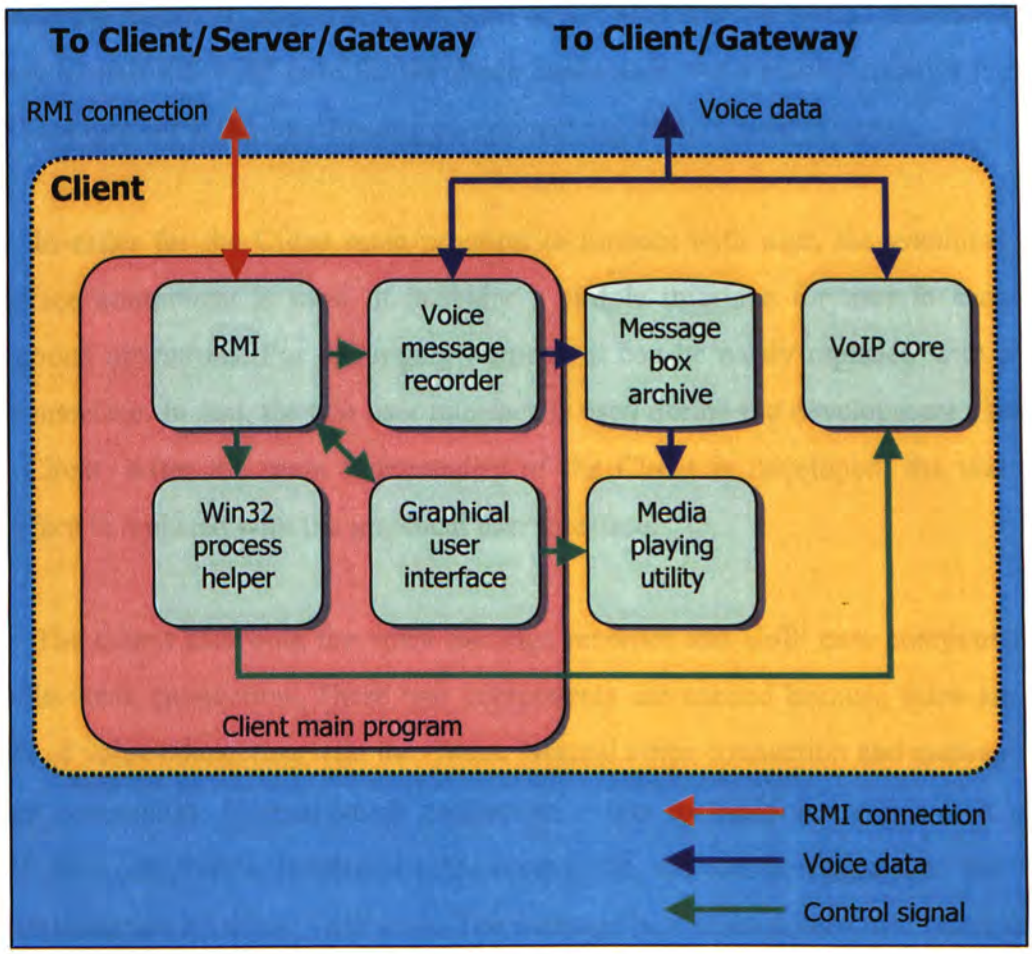


Figure 5-5: Client

The Client program consists of the Client main program, message box archive, media playing utility and a VoIP core. The Client main program consists of RMI component, voice message recorder, Win32 process helper and graphical user interface.

The Client main program contains the main functionality of the Client. In order to communicate with Server and Gateway through RMI, a RMI component is used. The RMI component uses Java RMI objects to communicate with Server and Gateway. By using RMI objects for the communication, no low-level network code is needed in the RMI component for communication. The RMI component interacts with the voice message recorder, Win32 process helper and the graphical user interface using Java event handling. Upon receiving a request from the network, the RMI component activates other components to carry out required operations. For example, when the voice connection request is received, it activates the GUI component to ask whether the user accepts an incoming connection. If the user

accepts the incoming connection, the RMI object activates the Win32 process helper object to start the VoIP core for the voice connection. As a result, it shows that the RMI component is the coordinating component inside the Client program.

In order for the Client main program to interact with user, the graphical user interface component is used. It provides a simple interface for user to carry out telephone operations. For debugging purpose, it can be easily replaced with a text user interface. In fact, the text user interface is used during the development phase of the Client. After the main functionality of the Client is developed, the text user interface is replaced with the graphical user interface.

The Client uses both the voice message recorder and VoIP core components to handle voice connection. These two components are needed because there are two types of voice connection with the Client: Normal voice connection and message box voice connection. Normal voice connection refers to the voice connection using VoIP core component. In normal voice connection, the user participates in the voice communication by using VoIP core. The message box voice connection refers to the voice connection using voice message recorder. In the message box voice connection, the user does not participate in the voice connection. The voice connection in the message box voice connection is one-way only and flows from Gateway or another Client into the Client. The incoming voice signals are recorded by the voice message recorder and stored as voice message for later retrieval.

The VoIP core component is not placed inside the Client main program because the VoIP core component is run in separate process space for performance reasons. If the VoIP core component runs as a thread inside the client main program, then it will compete for CPU resource with other threads inside the client main program. However, if the VoIP core is run in separate process space, then it avoids competing for CPU resources in the thread level. Instead, it can be assured of more CPU resources due to the process scheduling in the operating system.

Because the VoIP core is separated from the Client main program, a way is needed in order for the Client main program to start and stop the VoIP core. To do so, the Win32 process helper component is used in the Client main program. The

Win32 process helper starts the VoIP core by using Windows OS system call to start a new process. To stop the VoIP core, the Win32 process helper sends a WM_QUIT system message to the VoIP core component using Windows process messaging. The VoIP core component stops voice processing upon receiving this system message. The mechanism is shown in the following figure.

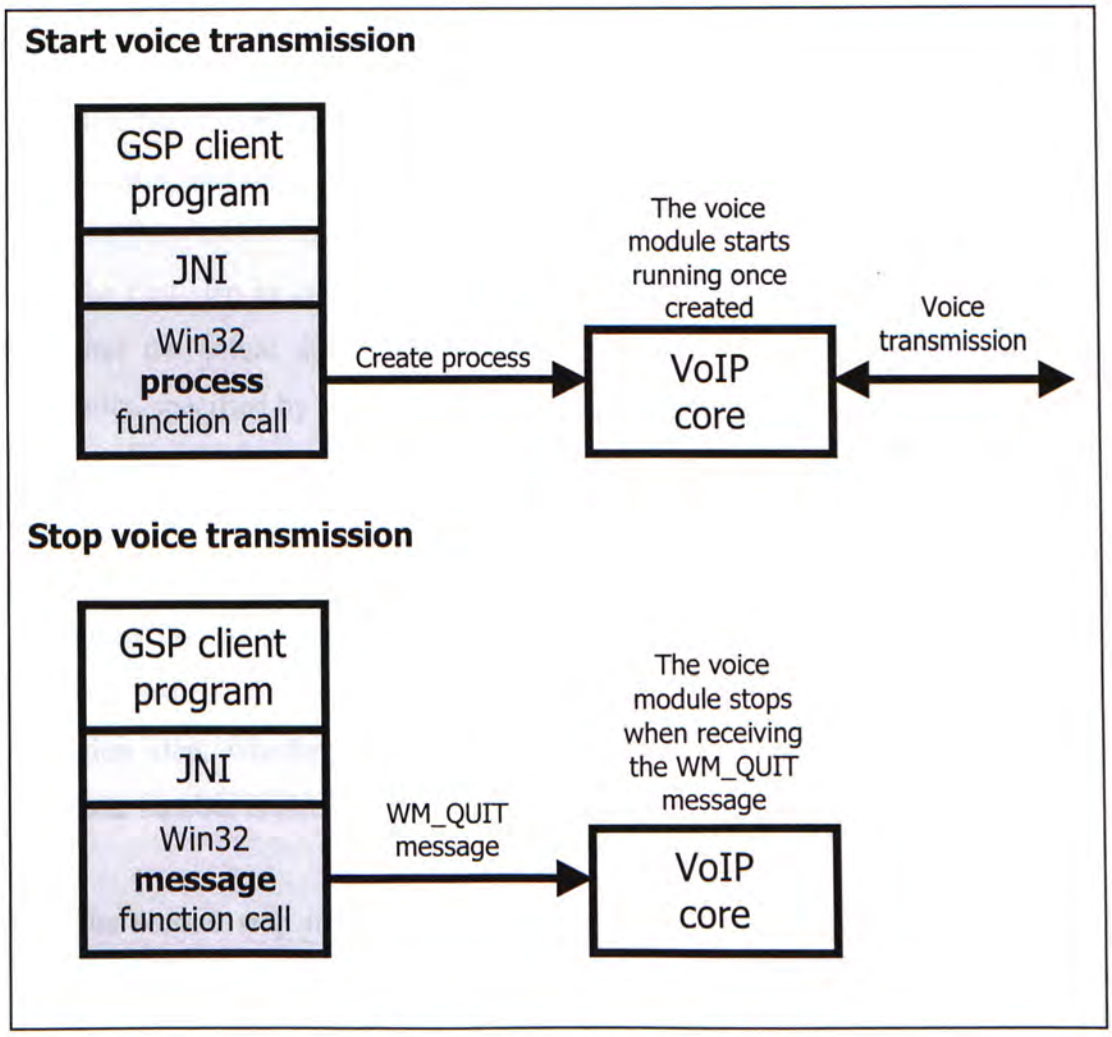


Figure 5-6: Starting and stopping VoIP core

As mentioned earlier, the Client main program uses the voice message recorder to record incoming voice data into voice message. The voice message recorder saves incoming voice messages into the message box archive. The message box archive is a special directory to hold all voice messages that belongs to the user of this computer. In order for the user to retrieve and listen the recorded voice messages, the user interacts with the graphical user interface that in turn activates an

external media playing utility such as Winamp to retrieve the voice message from the message box archive and playback to the user.

5.2.2. Connection Procedure

There is a core procedure in the GSP Client called "Connection Procedure". This core procedure defines the steps that need to be taken in order to initiate a voice connection with a particular participant. There are three main steps in the Connection Procedure (in chronological order): Destination Resolution, Connection Attempt, and Connect Messagebox.

The first step in the connection procedure is Destination Resolution. This step finds out the actual destination of the connection. It should be noted that the destination specified by the user might not be the actual destination of the connection due to call forwarding. For example, the user may specify to connect to user "kevin". However, the user "kevin" has specified to forward all his incoming calls to the PSTN telephone number "97861707" because he is not in the office at the moment. Therefore, the actual destination of the connection should be "97861707" via Gateway, instead of the computer that user "kevin" logged on. During the destination resolution step, whether the type of the destination is an IP client or a PSTN telephone number is also determined.

The second step in the connection procedure is Connection Attempt. In this step, the client program tries to determine if the state of the opposite side of the connection is "Online". If the actual destination is "Online", then the client program asks for connection permission. In this step, the client cooperates with the Server and the other side of the connection to determine if it is possible to setup a voice connection and setup a connection if the answer is affirmative.

The third step in the connection procedure is Connect Messagebox. This step is executed if the second step Connection Attempt failed. That is, it is not possible to make a voice connection to the callee. As a result, the user is connected to the voice message box of the callee such that the user may leave a voice message.

The pseudo code of the connection procedure is shown in the following figure.

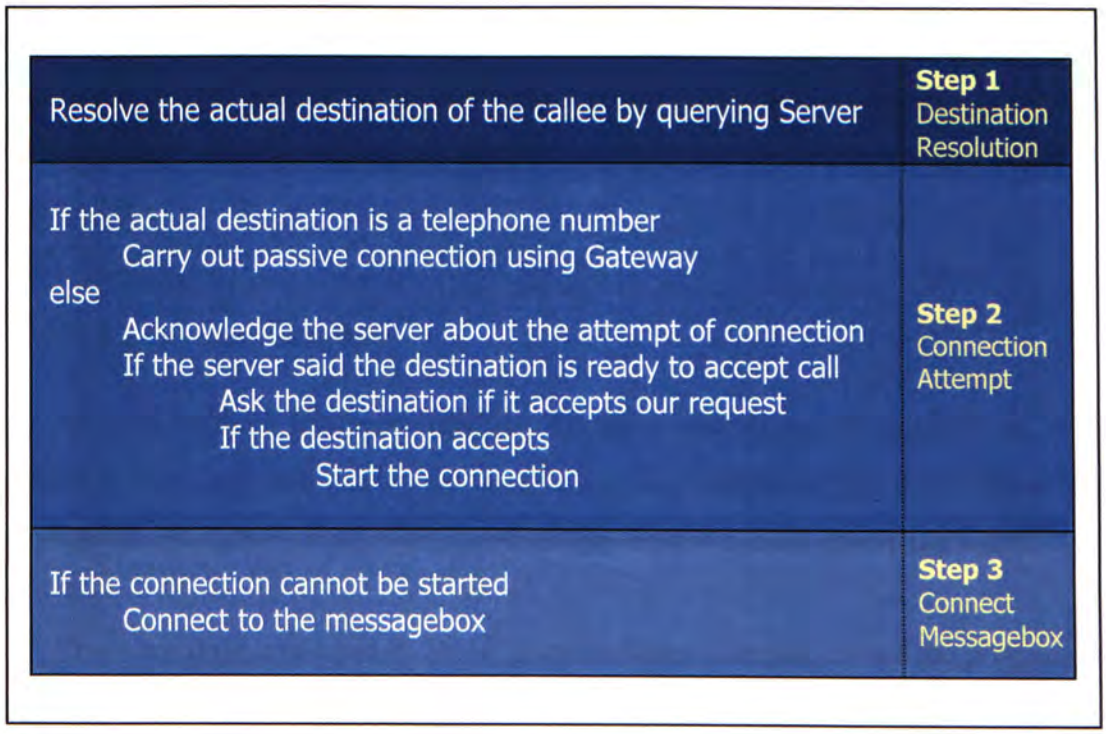


Figure 5-7: Pseudo code of the connection procedure

In the figure, there are two cases for the sentence "If the connection cannot be started" to be true. The first case is the status of the actual destination is not "Online". Hence, the server answered that the actual destination is not ready to accept call. The other case is that the actual destination refuses to accept the incoming call, although its status is "Online". In both cases, the caller should be connected to the message box.

Another important point in the figure is the keyword "Passive connection". A passive connection is setup when the destination is a telephone number instead of username. As a result, it is required to setup a connection via Gateway. There are two approaches in making a connection with PSTN through Gateway: active connection and passive connection. In the active connection approach, the client establishes a voice connection with Gateway first. Then the Gateway tries to make a voice connection to the PSTN. In passive connection approach, the client informs Gateway about the request. Then the Gateway tries to make a voice connection to the PSTN. The Gateway makes a voice connection to the caller only if it connects to the PSTN successfully. Otherwise, the Gateway informed the caller that the connection failed.

The passive connection is preferred to the active connection because if the PSTN side is often busy, then the voice connection between Gateway and caller has to setup and teardown many times which is not efficient. Therefore, the better approach is use passive connection that setups the voice connection between Gateway and caller only after the voice connection between Gateway and PSTN is successful.

5.2.3. User Interface

In this section, the user interface of the Client is briefly introduced. The graphical user interface allows the user to activate telephone function with a few keystrokes and mouse clicks. The user interface is shown in the following figure.

There are three main panels in the graphical user interface. When the program is started, only the main panel is shown. A number pad is provided in this panel to allow user to dial telephone number or input username.

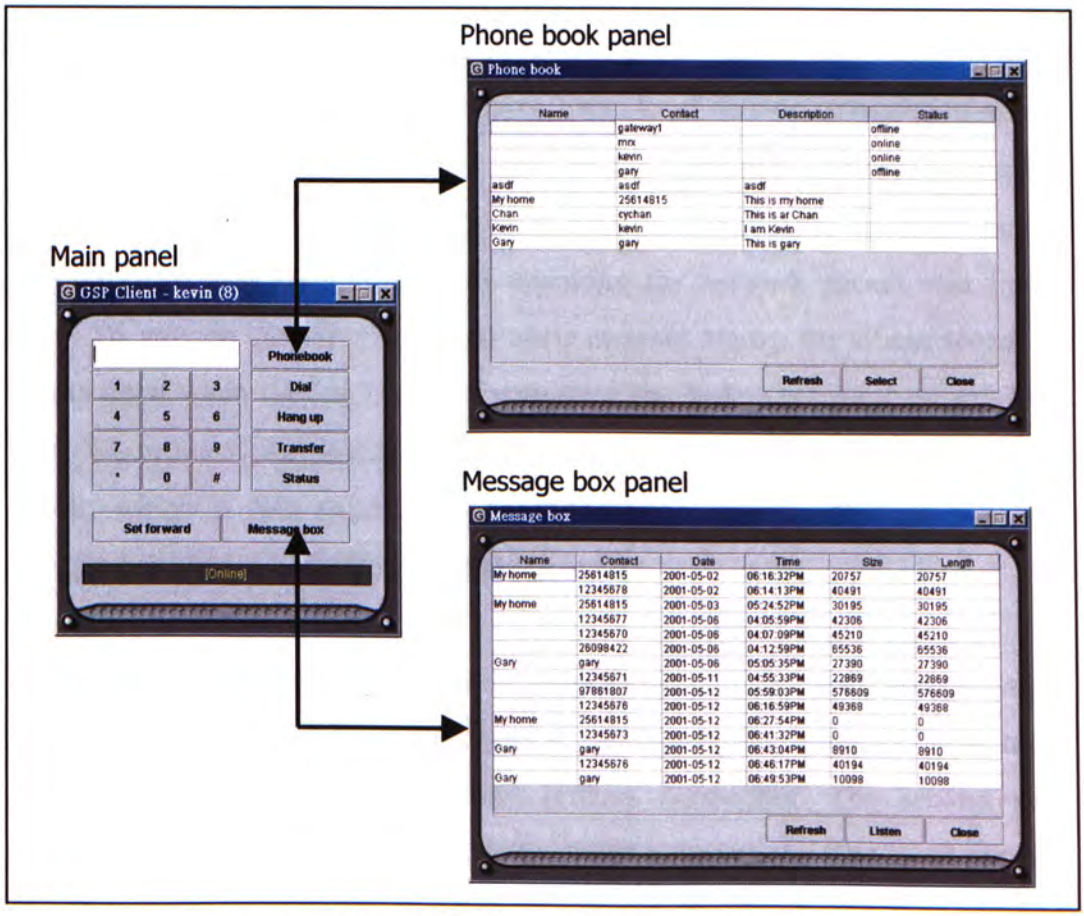


Figure 5-8: Graphical user interface of the Client

The phone book panel shows the entries in the user phonebook. The phonebook provides a convenient way to store frequently used phone number and username. As a result, the user may dial a phone number by selecting a particular entry in the phonebook.

The message panel shows all recorded voice messages in the local computer. The user may highlight a particular voice message and press the "Listen" button to activate the external media playing utility to listen to the voice message.

5.2.4. Observations

There are several observations when developing the Client.

The first observation is about the communication between the Client main program and the VoIP core. Since they are not located in the same process space, a way is needed to provide the communication channel between them. There are three ways to do so: local network connection, console piping and windows process messaging. To provide the communication using local network connection, the VoIP core listens to a particular network port in the computer. Whenever the Client program wants to notify the VoIP core, it sends a network packet to the port. The VoIP core can then be notified after receiving the network packet sent from the Client. To provide the communication using console piping, the Client specifies the console input when starting the VoIP core, then the VoIP core waits on any console input. When the Client wants to notify the VoIP core, it places the input on the console, which is then received by the VoIP core. Finally, the windows process messaging method is described in previous section.

The first method involves network programming. As a result, code for sending and receiving a network packet is needed. Moreover, the speed of local network transmission is slower than windows process messaging. The second method involves acquiring the console input of the VoIP core process. The VoIP core has to wait on the console input for command sent from the Client. Compared to console input waiting, process message waiting is more appropriate because it is a standard

way of process communication between processes in Windows. As a result, the last method is chosen as the method of the communication.

In spite of this, there is one disadvantage of using windows process messaging. It is not possible to use Windows process messaging in Linux. Consequently, other communication methods are needed if the Client is ported to the Linux platform. Among the three communication methods, the local network connection may be more suitable.

The second observation in implementing the Client is how to lay out the buttons and textboxes in the panels. Java user interface development guide [49] suggests that it is better to use layout manager for placing the buttons and textboxes in the panels dynamically such that the user interface can cope with different screen resolution if the size of the windows are resized. However, it is cumbersome to use layout manager to place a large number of graphics widgets in the windows. As a result, layout manager is not used in the implementation of the panels. Instead, the location of the graphics widgets in the panels is specified in absolute coordinates. As a result, the user interface can be built faster and the code is more understandable. The disadvantage is that the user interface cannot be resized because the location of the widgets in the panel is fixed. This should not pose any problem to the user interface of our system because it is not necessary to make the user interface of the Client resizable.

The last observation is that it is better to develop a text user interface before a graphical user interface when developing a complicated application. This is because it is easier to debug using a text user interface. A text user interface usually loads faster than a graphical user interface because graphics widgets have to be loaded in a graphical user interface. Moreover, it is easier to add temporary commands to show debugging information in the text user interface rather than the graphical user interface. After all the major functions in the program are tested thoroughly, the graphical user interface may be added to the program. However, it should be noted that some functions are closely related to the graphical user interface and therefore cannot be tested by text user interface. To conclude, implementing a simple text user interface aids software debugging and development.

5.3. Gateway

The second entity in the GSP system is the Gateway. The Gateway is responsible for transporting the voice data between the IP side and the PSTN side. Without the Gateway, it is impossible to make phone calls between the Intranet and the PSTN network.

Among the three entities (Client, Gateway and Server) in the GSP, the Gateway is the most complicated one. It is because it involves controlling both the computer network and telephone network. Moreover, it has to manipulate the telephone lines using a hardware telephony card.

The Gateway is similar to the PSTN gateway mentioned in section 4.2. They both use the API provided by the telephony card vendor to control the telephony card. However, the Gateway in this section is more complicated. It is because the PSTN gateway in section 4.2 only opens a voice connection between the IP side and PSTN side when an incoming call is detected. It does not handle situations such as call disconnection and dial out.

Similar to the Client program, the Gateway is also written using both Java and C++. It is because the Gateway contains C++ code to control the telephony card by using the API supported by the telephony card vendor. The API is implemented in Windows NT and Solaris platform only, therefore the Gateway can only be run on Windows NT and Solaris platform but not Windows 98 or Linux platform. Up to now, the Gateway is tested on Windows NT platform. It has not been tested in Solaris platform.

In the following section, the structure of the Gateway is presented.

5.3.1. Structure Overview

The Gateway consists of three layers: The Java layer, native C++ code layer and telephony card library layer. The three layers are shown in figure 5-12.

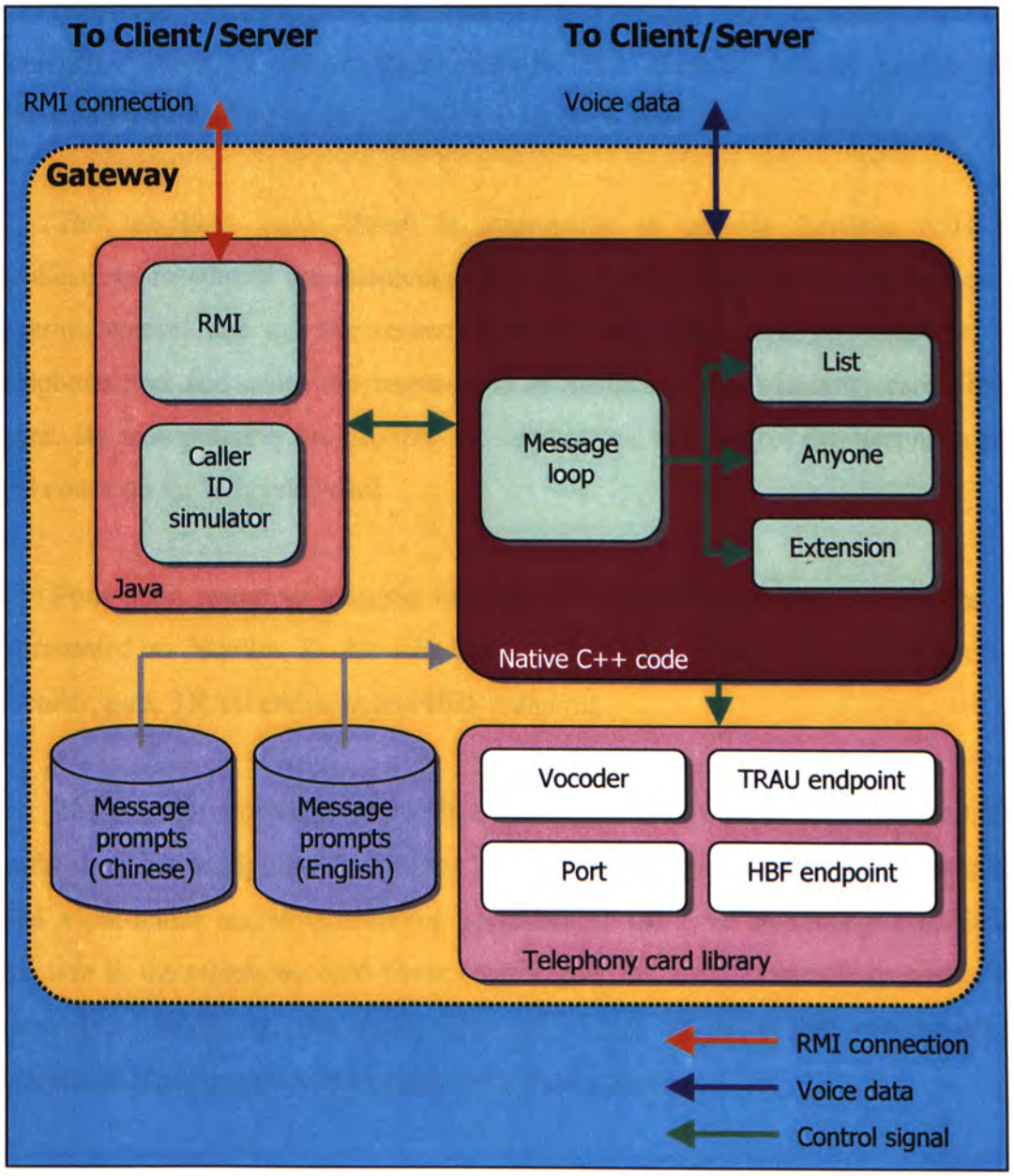


Figure 5-9: Gateway

The Java layer is responsible for communicating with the Client and Server through RMI. Moreover, it contains a caller ID simulator that simulates caller ID. The caller ID simulator will be mentioned later in section 5.3.3.

The native C++ code layer is responsible to control the hardware telephony card using the function calls and other facilities provided by the telephony card library. The core of the native C++ code layer is a message loop. The message loop waits for any event and activates the corresponding function to handle the event. For example, if there is a disconnect event, then the message loop will call the function to handle

the telephone disconnection. Moreover, the message loop dispatch incoming connection event to one of three modules: List module, Anyone module and Extension module. The function of these modules will be discussed later.

The telephony card library is responsible to provide function calls for applications to control the telephony card. It provides ways for the application to acquire, control and use the resources on the telephony card. For example, the telephone port and codec are represented as handles in the telephony card library layer. By manipulating the handles, the application can control the telephone port and codec on the telephony card.

Four main resources required to setup or teardown telephone connections are represented as handles in the telephony card library. These four resources are: vocoder, port, TRAU endpoint and HBF endpoint.

The vocoder represents the hardware DSP presented on the telephony card. The application can manipulate the DSP by using the vocoder handle. The application can load a particular encoding/decoding algorithm to the DSP by calling a particular function in the telephony card library and supply the vocoder handle as one of the parameters. Moreover, the application can switch on or switch off the Voice Activation Detection (VAD) by using the vocoder handle.

The port represents a particular socket at the back of the telephony card. There are four analog telephone sockets and each one is represented by a port object. Therefore, the application can control one of four telephone lines connected to the telephony card by manipulating the socket handle.

The two other resources are called TRAU endpoint and HBF endpoint. These two resources represent the vocoder and RTP socket as objects respectively. During a voice connection, voice data has to be moved between the codec and the IP network. To do so, the TRAU endpoint and HBF endpoint handles are acquired to represent the codec and the RTP port in the IP network respectively. By calling a particular function in the telephony card library, a voice channel is then setup between the

codec and the RTP port which enables the voice data flows between PSTN and IP network.

Besides the three layers in the Gateway, there are two special files in the Gateway: Chinese version message prompt and English version message prompt. These two files contain a number of voice files. These voice files are playback to the caller of incoming PSTN connection to guide he/she to select a particular callee. In the current implementation, the Gateway supports one language at one time. The gateway can only be startup with either Chinese version message prompt or English version message prompt, but not both. Therefore, this poses a problem when the caller to the Gateway does not understand the language that is currently loaded into the Gateway.

In order to allow the Java layer and the native C++ code layer to communicate with each other, JNI is used as the communication method. The communication between these two layers is bi-directional. The Java layer has to notify the native C++ code on how to control the telephony card. On the other hand, the native C++ code has to notify the Java layer of any changes in the telephone line signal. For example, in the presence of an incoming phone call, the participant on the PSTN side has to hang up the phone, etc. Fortunately, the JNI provides facilities for such bi-directional communication. From the Java side, the native C++ code is represented as Java methods. By calling the Java methods, the native C++ code can be executed. From the native C++ code side, the Java methods are denoted by method ID. By calling a JNI utility function and supplying a method ID as parameter, the Java method can be invoked.

5.3.2. Connection Procedure

The connection procedure for Gateway is the same as one in Client.

5.3.3. Caller ID Simulator

One of the challenges in the Gateway is to use the caller ID of the incoming call for dynamic presentation of information. Unfortunately, since Hong Kong Telecom

[50], that is, the telecom that provides the telephone line, does not follow the specification when implementing new telephone lines, it is impossible to use the standard telephony API function call to get the caller ID from the telephone line. As mentioned by the telephony card vendor Natural Microsystem technical support, there is another way to get caller ID by using other two function calls. Nonetheless, it is very difficult to uses these two function calls to get caller ID as lengthy trial-and-error is required. Therefore, the attempt to get caller ID from the telephone line is abandoned. Instead, a Java object called caller ID simulator is created to simulate the caller ID.

The mechanism of the caller ID simulator works as follows. Whenever there is an incoming phone call, the native C++ code will activate the Java caller ID simulator. Upon invocation, the Java caller ID simulator reads a list of telephone number from a telephone number file and randomly select one of the telephone number and returns it to the native C++ code. The returned telephone number is then assumed to be the telephone number for the incoming telephone call.

In the current implementation, the caller ID simulator only randomly picks one of the telephone numbers from the list. A better approach is to pick a telephone number using a probabilistic model such as normal distribution, etc. The higher the call count a phone number is assumed, the higher chance the phone number is chosen to be the phone number of the incoming call. Moreover, the caller ID simulator should also return a new telephone number to simulate that the incoming PSTN caller has never dialed in before.

5.3.4. Observations

There are three main problems encountered during the implementation.

The first problem is the retrieval of caller ID from telephone line. This problem is already discussed in section 5.3.3. It is hoped that this problem can be solved in later modification and enhancement to this project.

The second problem is the detection of a disconnect signal from the telephone line. Originally, it was not possible to detect a disconnect signal from the telephone line. As a result, the participant in the PSTN cannot notify the GSP system that he/she is disconnected. After consulting the technical support of Natural Microsystem, we are now able to detect the disconnect signal over the PSTN telephone line.

In order to detect the disconnect signal, we have to set the disconnect signal parameters in the Nature Microsystem telephony API. However, the parameters have to be measured manually. To do so, a special utility is used to record the waveform of the disconnect signal in the telephone line. After the disconnect signal waveform is recorded, the waveform is loaded into wave editor CoolEdit [51]. The screen in figure 5-12 shows what the waveform of PSTN disconnect signal looks like.

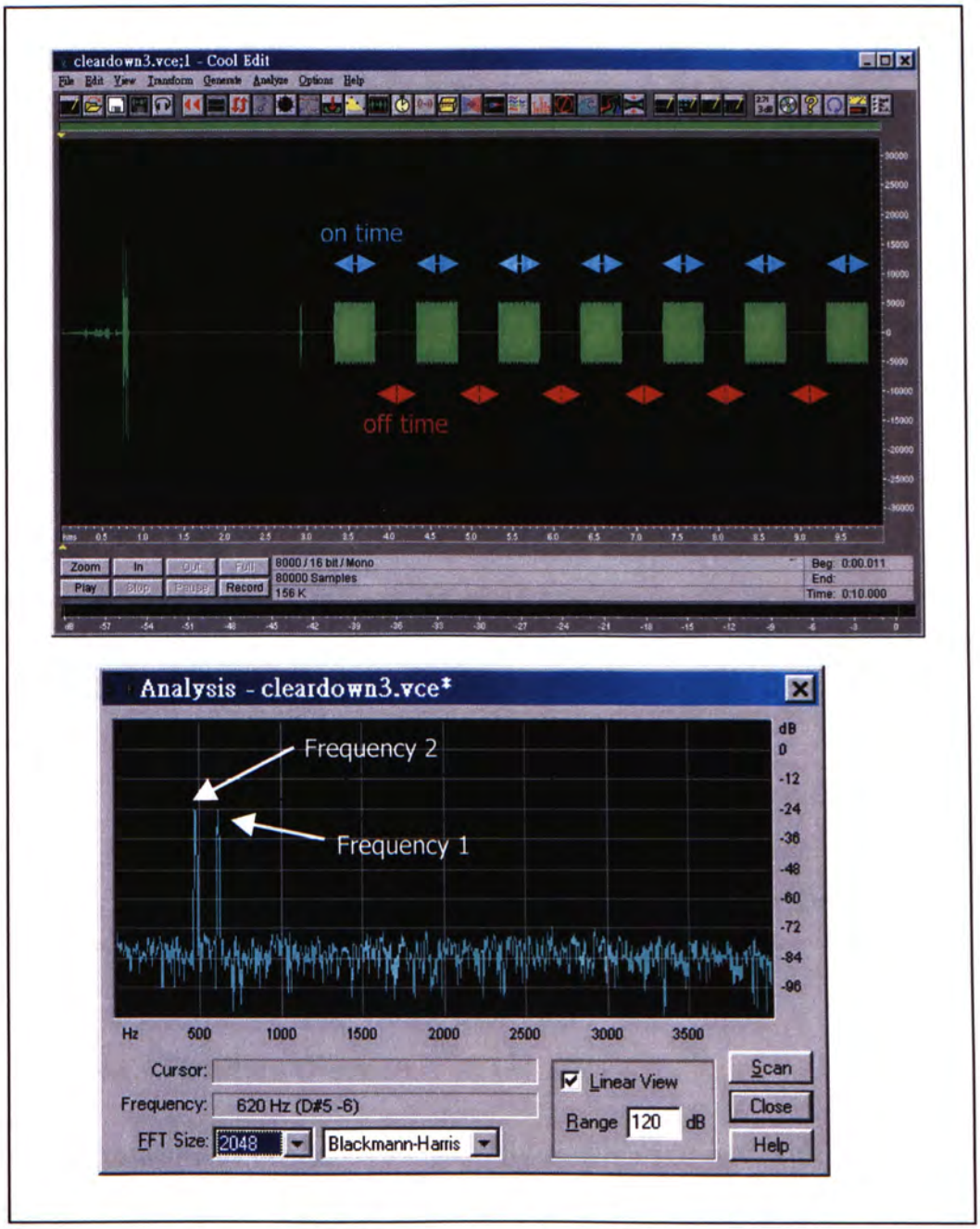


Figure 5-10: The waveform of the disconnect signal

Based on the shape of the waveform, the following parameters are determined from the waveform.

On time (min/max)	The length of the interval for the disconnect signal to appear in the line
Off time (min/max)	The length of the interval between disconnect signals
Frequency 1	The nominal frequency of the disconnect signal
Frequency 2	The second frequency of the disconnect signal

Table 5-1: Parameters of disconnect signal

The third problem that we have faced is the generation of the DTMF signals on the telephone line during the connection. As mentioned by the Natural Microsystem technical support, once the voice connection is created, the input of the telephone port is connected to the codec such as the encoded voice from the IP side can be transferred to the PSTN side. In order to generate DTMF digits on the telephone line, it is necessary to break such a connection and connect the input of the telephone port to the DSP such that the DSP can generate DTMF digits on the system. Due to the complexity, the generation of DTMF is currently not supported. It is expected later modification and refinement of the system will include this function.

5.4. Server

The third entity in the GSP system is the Server. The Server is responsible to keep track the status of Gateway and Client. Moreover, it stores information of users and callers in the system.

The Server is written in pure Java. Therefore, the Server can be run in Windows, Solaris or even Linux platform as long as the Java Development Kit supports that operating system platform.

5.4.1. Structure Overview

The structure of the Server is relatively simple compared with that Client and Gateway. The structure of the Server is shown in the following figure.

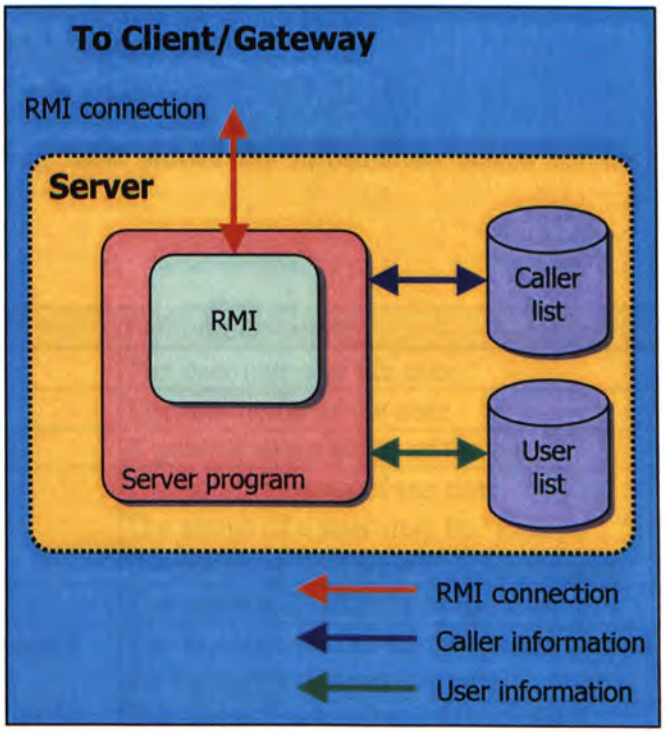


Figure 5-11: Server

The main part of the Server is the Server program. A RMI component is included in the Server program for communicating with Client and Gateway. Besides the Server program, the Server contains two database files for storing persistent information.

The two database files are called caller list and user list. The caller list stores the information for all incoming callers from PSTN. The user list stores information of all users in the system. The information in these two database files is loaded into the Server program when it is started.

The following table shows the information stored for each caller from PSTN.

Field name	Field description
Phone number	The phone number for this caller
Call count	How many times this caller dials into the system

Table 5-2: Caller information

The server has to keep track about the number of times a particular caller dials into the system because the Gateway will inquire the server for such information in

order to generate the dynamic order of the option list that presents the voice prompts to the incoming caller from PSTN.

The following table shows the information stored for each user in the GSP system.

Field name	Field description
User name	The user name of the user
Real name	The real name of the user
Description	A general description of the user
Status	The current status of the user. The status of a user may be "Online", "Offline", "Negotiating", "Talking", "Messagebox" or "N/A"
IP address	The current IP address of the user
Forward location	The location (either telephone number of user name) for forwarding incoming call
Extension number	The extension number of this user

Table 5-3: User information

Each user in the system is uniquely identified by a user name. For each user, the real name and description fields provide additional information to that user. Moreover, each user has an extension number that allows incoming PSTN callers to contact him/her quickly and directly.

The Server program provides various RMI method calls for Gateway and Client to inquire the information of caller and user. Based on the information, Gateway and Client can perform a number of telephony operations such as connect to a particular user, call transfer, etc.

5.5. Details of Major Functions

There are four main functions in the GSP system: secure local voice message box, call distribution, call forward and call transfer. These four main functions are the main differences between traditional VoIP point-to-point communication application (such as Netmeeting) and our intelligent GSP system. The mechanism of these four main functions will be presented one by one.

5.5.1. Secure Local Voice Message Box

As mentioned in section 5.1.4, sometimes it is possible that there are phone calls directed to a graduate student who is not in the room at the moment. Therefore, the incoming call is forwarded to the voice message box of the student. The voice message box is secure in the sense that the voice messages are stored in the user's local computer instead of a centralized computer dedicated for storing voice messages.

The secure voice message box is originally implemented in JMF. However, the performance of the implementation in JMF is not satisfactory. Therefore the secure voice message box is implemented in native Win32 C++ with the help of GSM library and tools.

The mechanism of storing and retrieving the voice messages is shown in the following figure.

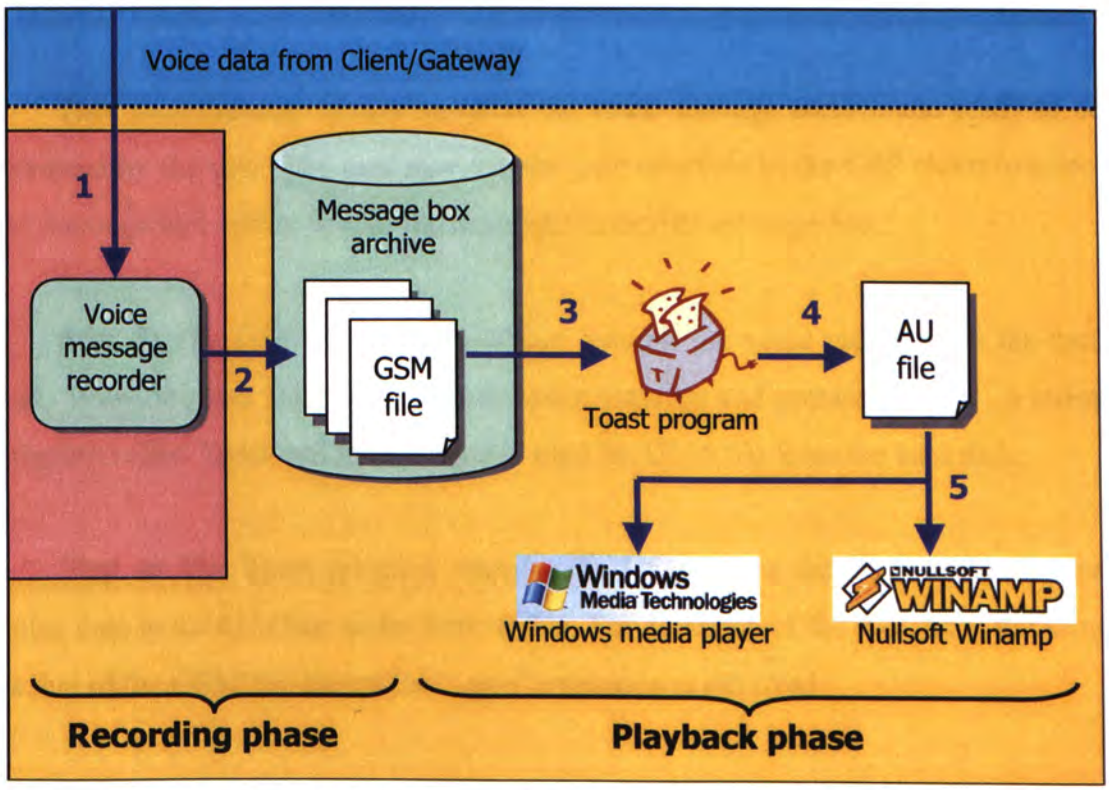


Figure 5-12: Mechanism of secure local voice message box

The mechanism consists of two phases: Recording phase and Playback phase. The Recording phase consists of steps 1 and 2. It refers to the steps in which the incoming voice message is saved to the voice message archive. The Playback phase consists of steps 3, 4 and 5. It refers to the steps in which the stored voice message is extracted and playback to the user.

The details of step 1 to step 5 are described below:

Step 1: The incoming voice stream comes from Client or Gateway in the form of GSM encoded RTP packets. The voice stream will be destined to one of the ports between port number 4000 and port number 4999 inclusive.

Step 2: A message box recorder object is created to save the GSM payload inside the RTP packets into a GSM voice data file. The naming of the voice data file depends on the telephone number/username, date and time:

[username/phone number(00000000)]-[date(yyyymmdd)]-[time(hhmmss)am/pm].au.gsm

The voice message is now saved to the voice message archive and ready to be retrieved by the user. The user may use the user interface in the GSP client to select the message box option to read the messages in his/her message box.

Step 3: The GSP client user interface lists all the voice messages in the hard disk. When the user selects a particular voice message and presses "Listen", a utility program called Toast will be activated to read the GSM file from the hard disk.

Step 4: The Toast program decodes the GSM voice data and saves the raw voice data to an AU (Sun audio format) file. The filename of the AU file is the same as that of the GSM file except the ".gsm" extension is removed.

Step 5: The AU file is read by a media playing utility such as Winamp or Windows Media Player for playback to the user. The user should be able to select the media playing utility for playing the AU file. However, it is required that the media playing utility must be able to playback audio file in AU format.

During the recording phase, the GSM encoded incoming voice stream is saved directly to a file without decoding. It is not necessary to do decoding when saving, and as a result, less CPU resource is required.

The following screenshot shows the windows media player that is used as the media playing utility to play a voice message in the voice message archive.

Figure 5-13: Playback voice message using Windows media player

It should be noted that it is possible that there are multiple message box connections to a Client. As a result, there may be multiple voice message recorder objects exist in a Client, listen to different voice message box ports and save the incoming voice streams into different files.

5.5.2. Call Distribution

The second major function of the GSP system is the call distribution.

In order to allow the incoming caller to select a particular callee, the Gateway will use voice prompts to ask the incoming caller. The call is distributed to a particular user based on the caller's choice. Voice prompts are pre-recorded voice messages that playback to the caller to ask him/her a question or to tell him/her some information.

For example, the Gateway asks the incoming caller to enter the extension number of the person to be called. The following voice message will be played to the caller in the phone line if the current Gateway supports English version voice prompt:

"Please enter extension number"

If the current Gateway supports Chinese version voice prompt, the following voice message will be played to the caller:

"請輸入內線號碼"

After that, the caller will hear a beep sound. The caller will then enter the extension number by pressing the buttons on his/her telephone panel. The Gateway will then detect the DTMF signals and informed of caller's input.

The above example shows a simple voice prompt. In order to generate voice prompt that is dynamic, multiple voice prompts are concatenated together to form a complete voice prompt.

For example, the Gateway would like to report the number of online users to the caller. The Gateway will play a voice prompt similar to the one shown below:

"There are two users online"

However, the number of online users is changing from time to time. Therefore, the voice prompt has to be generated dynamically based on the current number of online users. To do so, we have to analyze the complete voice prompt and separate it into constant and non-constant parts. The constant and non-constants parts are shown in the following diagram.

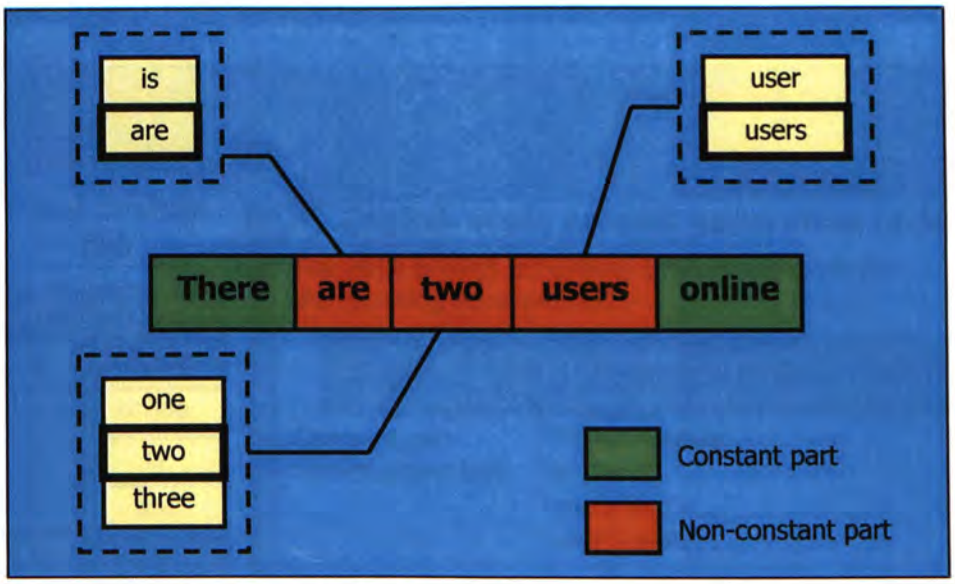


Figure 5-14: Generating dynamic voice prompt

As shown in the figure, there are five voice prompts in the complete voice prompts. Some of the voice prompts should be dynamic and some should not. For example, the number of users is dynamic. As a result, the Gateway may say "one", "two", "three" or others depends on the number of users.

As shown in the figure, not only the word "one"/"two" or others has to be changed depending on the number of users, but the grammar of the voice prompt has to be changed dynamically too. For example, if there is only one online user, the Gateway has to say "is"/"user" instead of "are"/"users" such that the grammar is correct.

To further complicate the problem, the Gateway should support at least two languages: Chinese & English. The grammar of Chinese and English language is different. It is not necessary to handle countable or uncountable situations in the Chinese language. As a result, the number of voice prompts in the voice prompt files is different for different languages.

The voice prompt generating mechanism is shown in the following figure.

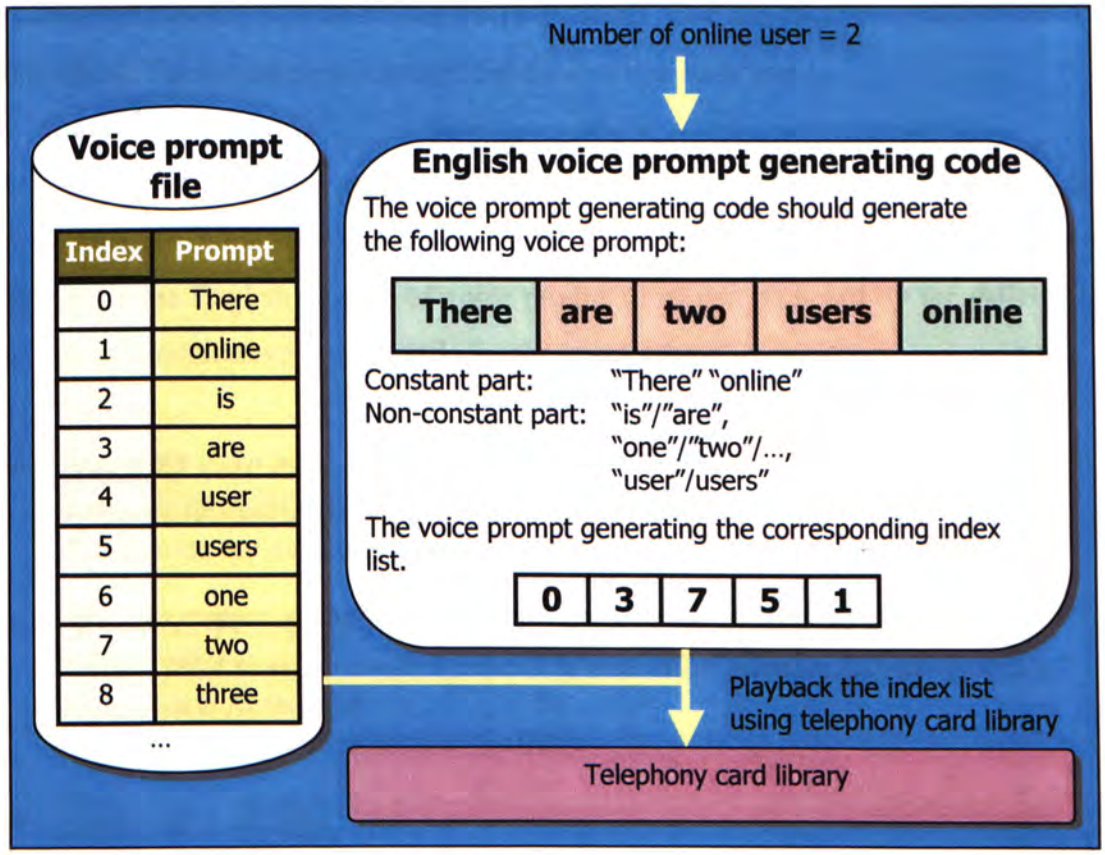


Figure 5-15: Voice prompt generating mechanism

All voice prompts for the same language version are stored in the same voice prompt file. As there are two languages supported by the Gateway, there are two voice prompt files, one for the Chinese language and the other for the English language. Each of the voice prompt has an index in the voice prompt file. Whenever a particular voice prompt has to be played during the Gateway execution, either the code for playback Chinese voice prompt or English voice prompt is executed. The code will then build the complete voice prompt from individual voice prompts by building a voice prompt index list. The voice prompt index list contains the voice prompt indexes in a complete voice prompt. The indexes may be hard coded if the parts in the voice prompt are constant. If the parts in the voice prompt are non-constant, then the indexes have to be computed to select the appropriate voice prompt index.

The above paragraphs described the steps in generating voice prompts. In following paragraphs, the actual voice prompts that the incoming caller will be described. The actual voice prompts form a tree like structure.

The structure of the voice prompt is shown as below. The words in ***italic***s are the Chinese version of voice prompts that can be heard by the caller. The words in ***italic***s are the English version of voice prompts that can be heard by the caller. The words in parentheses are the details for the voice prompt.

- ***歡迎使用 VoIP PBX 系統***
  Welcome to VoIP PBX system
  (A welcome message to the incoming caller.)

- ***按 一字/二字/三字/四字 選擇 任何一個人/內線/列表/系統資訊***
  Press 1/2/3/4 to select anyone/person by extension number/person by callee list/system information
  (The main menu presented to the caller.)

- (任何一個人 ***anyone***)
  (If the caller selects this option, the user will be connected to one of the online callee.)

- (內線 ***person by extension number***)
  請輸入內線號碼
  Please enter extension number
  (The caller has to input a valid extension number that represents a particular callee.)

- (列表 ***person by callee list***)
  按 一字/二字/三字 選擇 ...
  Press 1/2/3 to select...
  (A list of users' names is prompted to the caller. By selecting one of the users, the caller will be connected to that user.)

- (系統資訊 ***system information***)
  宜家系統共有 一/二/三 個人在線/ 宜家系統沒有人在線
  The callee cannot be reached at this moment. Please leave a message after the beep sound.
  (The caller can know how many users are currently online by pressing this option.)

Assume the caller selects a callee to be called by using either the "任何一個人 anyone ", "內線 extension number" or "列表 list" option, if the callee cannot be

connected at the moment, then the caller will be prompted with the following message. After that, the caller will be connected to the callee's messagebox.

- 電話暫時未能接通, 請 o 係 "beep" 一聲之後留底口訊
  The callee cannot be reached at this moment. Please leave a message after the beep sound.
  (After the caller hear this message, he/she may leave a voice message to the callee. The voice message is streamed to the callee's computer directly in real-time for storage.)

To conclude, there are four main options available to the incoming caller: "Anyone", "Extension", "List" and "System Information".

Option	Description
Anyone	If the caller selects this option, he/she will be connected to one of the online users. This option allows the incoming caller to ask for help if he/she does not know whom to connect.
Extension	If the caller selects this option, he/she will have to input the extension number of the user that he/she going to call. This option is useful for a frequent caller who always calls one of the users in the system.
List	If the caller selects this option, he/she will be presented with a list of users in the system. He/she can select one of the users in the list to call. This option is useful for a normal user who calls several persons in the room and cannot remember their extension numbers.
System Information	If the caller selects this option, he/she will be informed of how many users are currently online.

Table 5-4: Details of different option type

The incoming caller is classified into three types: First time caller, Usual caller, and Frequent caller. The order of the options presented to different caller types is different. If the caller dials in only once, he/she is classified as a first time caller. If the caller dials in less than five times, he/she is classified as a usual caller. If the caller dials in more than five times, he/she is classified as a frequent caller. The ranges in the classification are adjustable based on the system settings.

Caller type	Order of options (In chronological order)
First time caller	Anyone, List, Extension, System Information

Usual caller	List, Extension, Anyone, System Information
Frequent caller	Extension, List, Anyone, System Information

Table 5-5: Different caller types in the system

The "System Information" option is the least important option. Therefore it is placed at the end of the order list for all caller types.

For the first time caller, it is more likely that he/she does not know whom to call, therefore the "Anyone" option is prompted first. Even if the caller knows whom to call, it is more likely that he/she does not know the extension number. Therefore, the "List" option is placed in front of the "Extension" option.

For the usual caller, it is more likely that he/she has in mind whom to call, therefore the "Anyone" option is placed near the end of the option list. It is likely that although he/she has in mind whom to call, but he/she still does not remember the extension of that person because he/she does not call frequently. Therefore, the "List" option is placed in front of "Extension" option to allow he/she to select the person from the list.

For the frequent caller, it is more likely that he/she already remembers the extension number of the caller. Therefore, the "Extension" option is placed at the front of the option list. It is less likely that the caller will use "Anyone" option, therefore the option is placed nearer to the end of the option list.

5.5.3. Call Forward

The third major function of the GSP system is the call forwarding. By using call forwarding, the incoming call directed to a particular location can be forwarded to another location.

For each user in the system, there is a setting called forward that is stored in the server. During the connection procedure, the forward setting of the callee will be retrieved to see if the callee has set the forward field to a username or a telephone. If the forward field is not empty, the actual destination of the callee is updated by the

forward field. The above procedure is repeated until the forward field of the callee is empty.

An example is shown in the following diagram to show how call forwarding works. The following example assumes that a user does not like to answer any incoming call and forwards all the calls to another user. There are four users in the figure. The yellow boxes above the users are their forward setting. An empty forward setting box represents the user does not enable call forward.

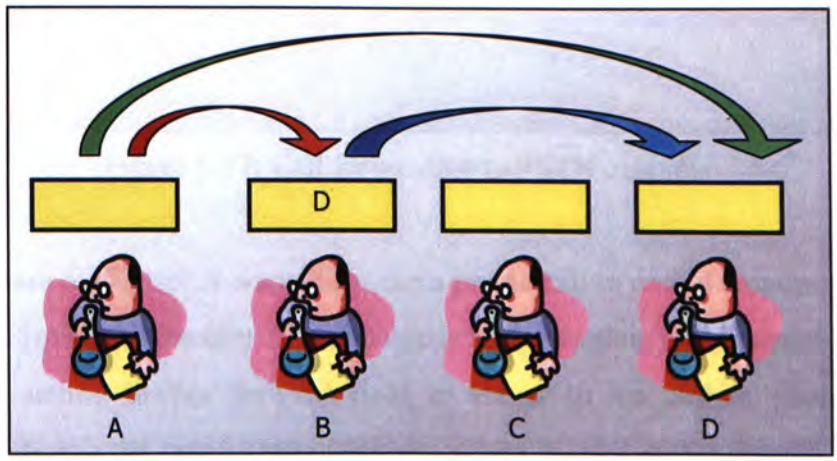


Figure 5-16: A call forwarding example

In the figure, user A wants to make a phone call to user B (represented by the red arrow). However, the user B has enabled call forwarding (represented by the blue arrow) by setting his/her forward field in server to username D. Therefore, during the voice connection procedure in user A, the system detects the user B forwards all incoming call to user D. As a result, the user A will make a phone call to user D instead of user B (represented by the green arrow).

The incoming call can be forwarded not only to another user, but also to PSTN network. In the following example, the user is leaving the office and likes to forward all the calls directed to him to his mobile phone number.

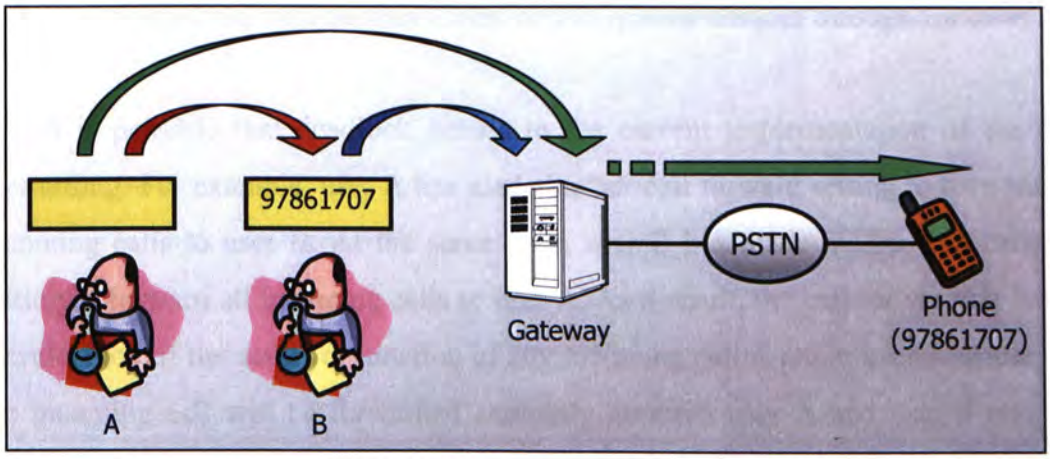


Figure 5-17: Call forwarding to PSTN example

In Figure 5-17, user A wants to make a phone call to user B (represented by the red arrow). However, the user B has enabled call forwarding (represented by the blue arrow) by setting his/her forward field in server to his mobile phone number. Therefore, during the voice connection procedure in user A, the system detects the user B forwards all incoming calls to a PSTN telephone number. As a result, the user A will make a voice connection with the Gateway (represented by the green arrow). The Gateway will in turn make a voice connection to the mobile phone number 97861707 through the PSTN network. After that, the user A is connected to the mobile phone number 97861707 through the Gateway.

A special scenario is that the caller is from the PSTN side and the forward location is a PSTN telephone number. In this scenario, the call forwarding has to rely on the call transfer function in the underlying telephone line provided by the telecom. If the underlying telephone line does not have call transfer enabled, then it is impossible to do such call forwarding.

In order for call forwarding to work, the Server stores the forwarding information for each user in the system. During the call connection procedure, the Client/Gateway sends a **getForwardRecursive** request to the Server and supply the username/phone number of the caller. When the Server receives such request, it will

recursively look up the forwarding information and follow the forwarding path to resolve the actual destination. The address of the actual destination is returned to the Client/Gateway. Based on the type of the actual destination, the Client/Gateway then initiates a connection with another Client or a telephone number through Gateway.

It is possible that deadlock occurs in the current implementation of the call forwarding. For example, user A has made his/her call forward setting to forward all incoming calls to user B. At the same time, user B has made his/her call forward setting to forward all incoming calls to user A. As a result, the call forwarding forms a cycle. Now if the actual destination of any incoming call is either user A or user B, the incoming call will be forwarded endlessly between user A and user B and the system will lock up. It is expected that future modification and enhancement to the system can prevent deadlock in call forward.

5.5.4. Call Transfer

The fourth major function of the GSP system is call transfer. The purpose of call transfer is to allow the swapping of one participant in a phone connection with another participant. The new participant may be one of the users in the system or in a PSTN telephone line. The process of call transfer is voluntary and must be activated by the participants in the voice connection.

There are four types of call transfer depends on the type of call participants in the original voice connection. There are two types of call participants in a voice connection: PC or PSTN. In the table, the word PC represents a Client in the Intranet; the word PSTN represents a PSTN connection endpoint through the Gateway. The subscript behind the word "PC" or the "PSTN" are used to differentiate different entities for the same type of call participant. The four types of call transfer are listed in the following table.

Case	Original connection	Transfer		New connection
		from	to	
1	$PC_1 \Leftrightarrow PC_2$	PC_2	PC_3	$PC_1 \Leftrightarrow PC_3$
2	$PC_1 \Leftrightarrow PC_2$	PC_2	PSTN	$PC_1 \Leftrightarrow$ PSTN
3	$PC_1 \Leftrightarrow$ PSTN	PC_1	PC_2	$PC_2 \Leftrightarrow$ PSTN

4	$PC_1 \Leftrightarrow PSTN_1$	PC_1	$PSTN_2$	$PSTN_2 \Leftrightarrow PSTN_1$

Table 5-6: The four types of call transfer

The mechanism in case 1 is simple. The voice connection between PC_1 and PC_2 is disconnected first. After that, the PC_1 initiates a voice connection with PC_3.

The mechanism in case 2 is similar to case 1. The voice connection between PC_1 and PC_2 is disconnected first. After that, the PC_1 initiates a voice connection to PSTN by sending the request to the Gateway. The Gateway then dials out to the PSTN telephone number. After the PSTN call is connected, a voice connection is setup between PC_1 and the telephone number in PSTN through Gateway.

The mechanism in case 3 is similar to case 1. The voice connection between PC_1 and Gateway is disconnected first. However, the voice connection between the Gateway and the PSTN is not dropped. After the voice connection between PC_1 and Gateway is disconnected, a new voice connection is initiated from Gateway to PC_2. After the voice connection is setup between Gateway and PC_2, the PC_2 and PSTN are connected through the Gateway.

The mechanism in case 4 is similar to forwarding PSTN incoming call to another PSTN number that the telecom call transfer is needed. If the telephone line does not support the call transfer function, then it is not possible to do call transfer in this case. First, the voice connection between PC_1 and Gateway is disconnected. However, the voice connection between the Gateway and the PSTN is not dropped. Then the Gateway uses the telephone transfer function to transfer the call to another telephone line. In this call transfer scenario, the transfer function call in the telephony API is used to utilize the underlying telecom telephone transfer function. After that, a voice connection is setup between the two telephone lines. The Gateway is not involved in the voice connection between these two telephone lines.

5.6. Experiments

In order to evaluate the performance of the Graduate Second Phone, the following experiments are done to measure the performance of different areas of the

GSP. There are five experiments in total in this section: Secure Local Voice Message Box, Call Distribution, Call Forward, Call Transfer and Dial Out.

5.6.1. Secure Local Voice Message Box

The purpose of this experiment is to examine the relationship between the size and the length of (compressed and uncompressed) voice messages.

Experiment Setup

Six voice messages are created with different lengths ranging from ten seconds to three minutes. The content of the voice messages is the same that a person is speaking continuously without stopping. After the voice messages are created, the size of the voice messages before and after decoding are measured.

Experiment Results

The length and size of the voice messages are summarized in the following table and figure.

Voice message	Length	Size	
		GSM encoded (.au.gsm)	Sun audio format (.au)
1	15s	26kb	127kb
2	24s	41kb	197kb
3	36s	61kb	295kb
4	64s	106kb	516kb
5	126s	208kb	1,009kb
6	185s	305kb	1,481kb

Table 5-7: The length and size of voice messages

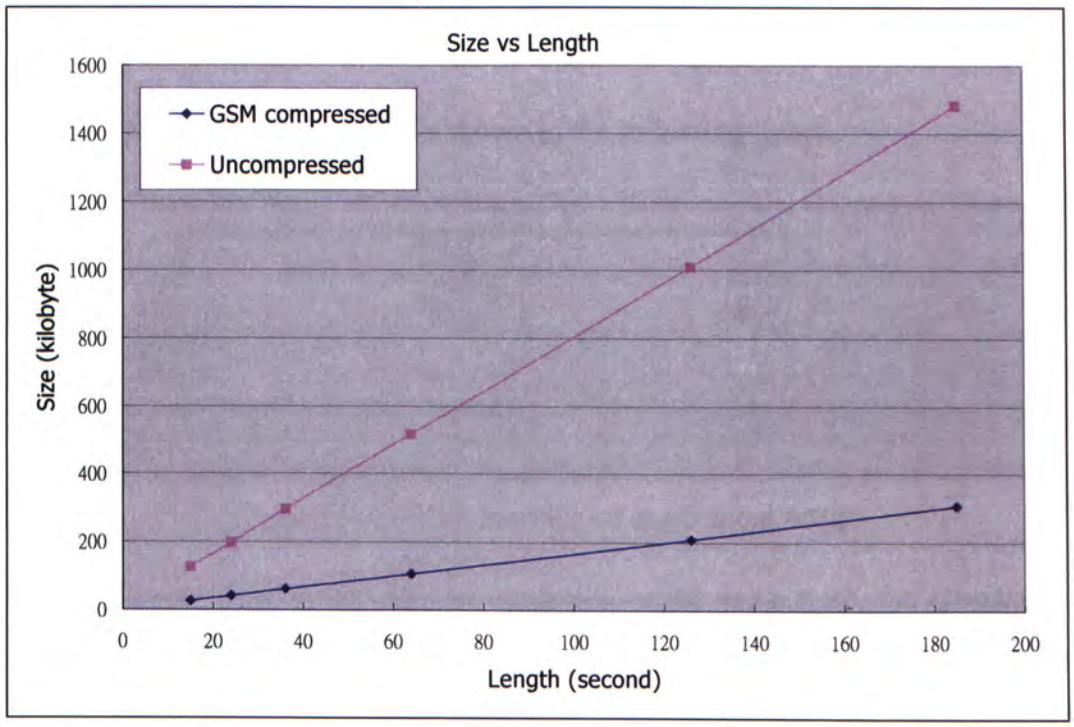


Figure 5-18: The length and size of voice messages

As shown in the experiment, the size of voice message is small. The size of GSM encoded voice message is much smaller than that of the decoded voice message. The size of a GSM encoded voice message file for three minutes of voice message is only about 300 kilobytes. As hard disk space nowadays is large, therefore a large number of voice messages can be stored in the computer hard disk.

It should be noted that time is needed to decode the voice data from a GSM encoded file before playback. However, the time needed for decoding is just one to two seconds for most of the above voice messages in the experiment. Moreover, the decoding process is only needed for first time playback. For later playbacks, the decoded file is used and no decoding process is necessary to save time. Therefore, the voice message system in the GSP project functions rapidly and compactly.

5.6.2. Call Distribution

The purpose of this experiment is to measure the time needed for the PSTN caller to connect to the callee successfully.

Experiment Setup

The setup of the experiment is shown in the following figure.

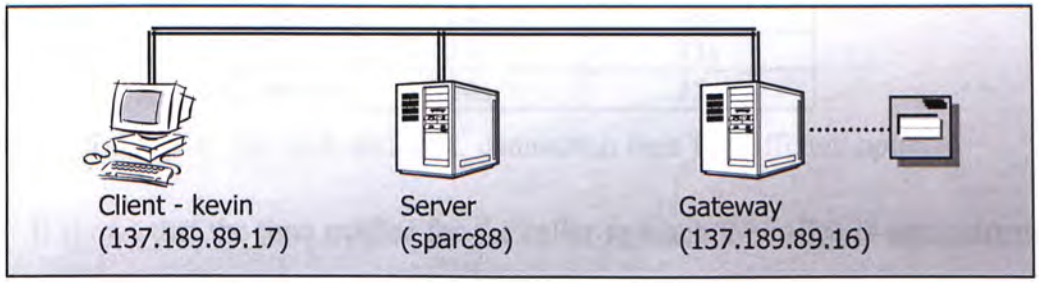


Figure 5-19: Call distribution experiment setup

The Server runs in the sparc88 machine. At the same time, the Gateway and Client are started in IP 137.189.89.16 and 137.189.89.17, respectively. The username of the Client is "kevin". Incoming calls are made from the PSTN network to connect to the client "kevin" using the three options: "Anyone", "Extension", "List". The Chinese version of the voice prompt is used in Gateway.

In this experiment, the minimum and maximum times to reach the callee are determined for the above three options. The time measured is the time between the incoming call connected to the Gateway and a dialog box is prompted to the callee to ask if he/she answers the call or not. The minimum time represents the time needed for a frequent caller to reach the callee. It is assumed that a frequent caller can make a choice after the first option is prompted to him/her. It is because the caller can choose his/her option after listening to the first option. In contrast, the maximum time represents the time needed for a first time caller to reach the callee. It is assumed that a first time caller can only make a choice after listening to all prompted options.

Moreover, it is assumed that there are three users in the system. As a result, only three usernames are prompted in the "List" option. If there are more users in the system, the length of the "List" option prompt will be longer.

Experiment Results

The result of the experiment is shown in the following table.

Option	Minimum time (Frequent caller)	Maximum time (First time caller)
Anyone	11s	24s
List	14s	33s
Extension	14s	25s

Table 5-8: The min. and max. connection time for different options

It shows that the time needed for the caller to reach the callee is approximately half a minute. Moreover, the time measured above included the system introduction voice message that lasts for eight seconds. If the system introduction voice message is removed, the time needed to reach the callee will be shorter. The following table shows the time needed if the system introduction voice message is removed:

Option	Minimum time (Frequent caller)	Maximum time (First time caller)
Anyone	3s	16s
List	6s	25s
Extension	6s	17s

Table 5-9: The min. and max. connection time for different options (exclude intro.)

As shown in the table, it shows that the frequent caller can reach the callee in three seconds. Even the maximum time for a first time caller to reach the caller is 25 seconds. Therefore, the average time needed for each caller to reach the callee is reasonable and acceptable. The table shows that it takes the longest time for a caller to select a callee using "List" option among the three options. This is because all the usernames will be prompted in "List" option and therefore it takes a lot of time if there are many users in the system. It should be noted that the real length of the time for a caller to reach callee may be longer because the caller may have to think before making a decision to press a button. Furthermore, the time needed varies and depends on the responsiveness of the caller.

In a call center application, the time needed to handle each call should be strictly kept to a minimum because even a second is precious and incurs cost in keeping the call center running. In contrast to a call center application, the most important point in the call distribution in our system is that the caller feels easy and

comfortable in listening to the voice prompt, makes his/her choice based on the prompts and connect to the callee. In the current system prototype, the voice prompts are recorded by the writer of this thesis, who is a male. Therefore, the quality of the voice prompts is not good. In order to improve the quality of voice prompts, a sweet female voice should be used for the voice prompts.

5.6.3. Call Forward

The purpose of this experiment is to measure the time needed to forward an incoming call to the destination.

Experiment Setup

Four user accounts are created for this experiment: "test", "test2", "test3" and "test4". The accounts are created such that the account "test" forwards all incoming calls to the account "test2". The account "test2" forwards all incoming calls to the account "test3". The account "test3" forwards all incoming calls to the account "test4". Therefore, the effect is that all incoming calls to the account "test" are forwarded to the account "test4". As a result, there are three levels of redirection in this call forward scenario:

test → test2 → test3 → test4

In the experiment, the account "kevin" will be used to connect to the account "test". Due to the call forward setting in the accounts "test", "test2" and "test3", the account "kevin" will be connected to "test4". The time needed for the call forwarding is measured by adding timing code to the Client and Server. Timing code is added to the Client to measure the time needed to invoke RMI call to server for resolve the actual location to be connected. Timing code is also added to the Server to measure the time needed to resolve the actual destination due to the call forward settings. The sum of the lengths of these two steps represents the time needed for call forwarding.

Experiment Results

The result of the experiment is shown in the following table:

Process	Time (second)
Client invokes call forward RMI call to Server	0.05s
Server performs destination resolution and returns the result to Client	0.001s
Total	0.051s

Table 5-10: Time needed for call forward

The time needed for the call forwarding process is negligible and the caller will not be aware of the call forwarding process. Moreover, it is rare that the depth of call forwarding is more than three. Therefore, the call forwarding process is transparent to the caller.

5.6.4. Call Transfer

In this experiment, the time needed to perform a call transfer is investigated. There are four types of call transfer in the system and the times needed for each of them are measured.

Experiment Setup

The setup of the experiment is shown in the following figure.

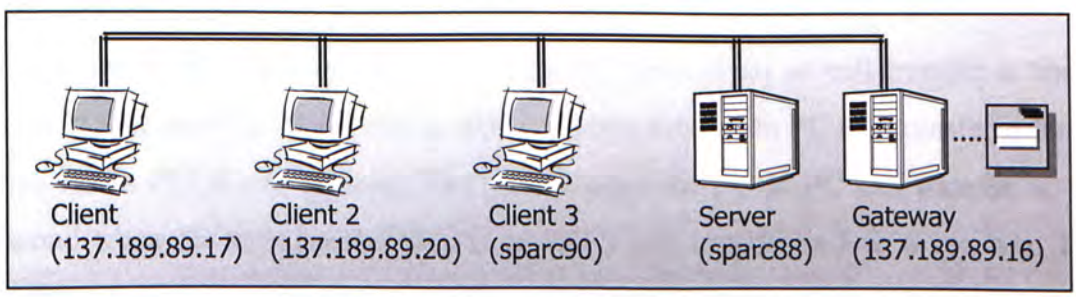


Figure 5-20: Call transfer experiment setup

The setup is similar to the previous experiment except that a new Client is added. The third client is executed in sparc90.

The four type of call transfer is show in the following table. The table is the same as the one in Section 5.5.4.

Case	Original connection	Transfer		New connection
		from	to	
1	$PC_1 \Leftrightarrow PC_2$	PC_2	PC_3	$PC_1 \Leftrightarrow PC_3$
2	$PC_1 \Leftrightarrow PC_2$	PC_2	PSTN	$PC_1 \Leftrightarrow$ PSTN
3	$PC_1 \Leftrightarrow$ PSTN	PC_1	PC_2	$PC_2 \Leftrightarrow$ PSTN
4	$PC_1 \Leftrightarrow PSTN_1$	PC_1	$PSTN_2$	$PSTN_2 \Leftrightarrow PSTN_1$

Table 5-11: The four types of call transfer to be tested

The time needed to perform the call transfer is defined as the time between when the call transfer is initiated and when a dialog box is popped in callee (for IP) or the ring sound is heard (for PSTN).

Experiment Results

The result of the experiment is shown in the following table:

Case	Original connection	Transfer		Time needed
		from	to	
1	$PC_1 \Leftrightarrow PC_2$	PC_2	PC_3	1.5s
2	$PC_1 \Leftrightarrow PC_2$	PC_2	PSTN	6s
3	$PC_1 \Leftrightarrow$ PSTN	PC_1	PC_2	1.5s
4	$PC_1 \Leftrightarrow PSTN_1$	PC_1	$PSTN_2$	5s

Table 5-12: The time needed for the four types of call transfer

It shows that the average time for the different types of call transfer is about three to four seconds. In the above table, it shows that PC to PC call transfer is faster than PC to PSTN call transfer. The time needed for PC to PC call transfer is 1.5 second while the time needed for PC to PSTN call transfer is 5 to 6 seconds. The reason for this is that for the latter case, either dialing or transfer operation has to be done in PSTN using telephone network signaling. As a result, the speed for the call transfer is slower. To conclude, the time needed for call transfer is reasonable.

5.6.5. Dial Out

In this experiment, the time needed to perform dial out is investigated. There are mainly two types of dial out and both of them are tested.

Experiment Setup

The setup of this experiment is similar to the last experiment except that the client 3 is not needed.

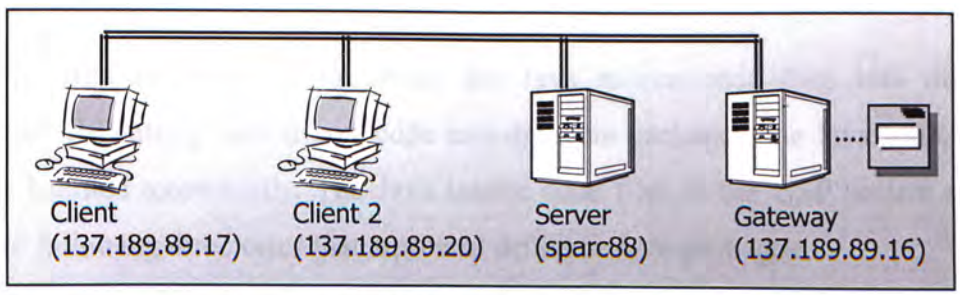


Figure 5-21: Dial out experiment setup

The two types of dial out are: PC to PC and PC to PSTN. For example, the type of connection from Client to Client 2 is PC to PC dial out while the connection from Client to PSTN phone number 18503 is PC to PSTN dial out. Both of them will be tested.

Similar to last experiment, the time needed to perform for the dial out is defined as the time between when the dial out is initiated and when a dialog box is popped in callee (for IP) or the ring sound is heard (for PSTN).

Experiment Results

The result of the experiment is shown in the following table.

Dial out type	Time needed
PC to PC	2s
PC to PSTN	6s

Table 5-13: Dial out experiment results

As a result, it is faster to make a voice connection between PC Clients. The reason for this is that PSTN telephone signaling and hardware telephony card are not involved. To conclude, the time needed for dial out is reasonable.

5.7. Observations

During the implementation of the GSP project, there are two more software engineering techniques observed that could improve the quality of the software development process.

The first technique is to divide the Java source code files into different packages. By putting Java source code into different packages, the Java source code can be handled more easily. The Java source code files in the GSP project are put into the following directories that represent different Java packages.

Package	Directory	Description
GSP	\GSP	This directory contains source code that does not belong to Client, Gateway or Server entity. An example is a Java class file storing all global constants
GSP.Client	\GSP\Client	This directory contains Client implementation
GSP.Client.Record	\GSP\Client\Record	This directory contains the voice message recorder implementation inside Client
GSP.Client.Win32Process	\GSP\Client\Win32\Process	This directory contains the Win32Process helper class implementation inside Client
GSP.Server	\GSP\Server	This directory contains the Server implementation
GSP.Gateway	\GSP\Gateway	This directory contains the Gateway implementation

Table 5-14: The Java source packages in the GSP system

By such ordering, the GSP source code files are more structured and facilitate the expansion of the system.

The second technique observed is the planning and specification using a finite state machine. In the GSP project, there are many different components that contain different number and types of states. For example, the Client may be in "Online", "Offline", "N/A", "Talking", "Negotiating" and other internal states. Therefore, the interaction of the different components in the system becomes quite complex. A good approach to handle the complexity is to construct a finite state machine to clarity the states and interactions of the components.

5.8. Outlook

The GSP project prototype is completed and shows the concept of Graduate Second Phone is feasible. However, there is a long way to go before the GSP project prototype becomes fully-fledged reliable working software. There are several outlooks on the structure and functions of the GSP project.

The user interface of the GSP project is still in prototype form. The user interface can be designed and implemented better. In order to allow the user to carry out telephone operations conveniently and intuitively, the interface should be clearly defined and well designed.

The second outlook is that we should investigate if it is possible to use Netmeeting as the VoIP core instead of the VoIP core built by ourselves. By allowing the system to use Netmeeting as the VoIP core, users of the system are free to choose the best VoIP core. As a result, the system is more flexible and can be tailored to customer's need.

The third outlook is to further enhance the function of the system. In the current system, there are four main functions: Secure Local Voice Message Box, Call Distribution, Call Forward and Call Transfer. There are many additional useful functions that should be added to the system. Two of the functions that yet have to be added to the system are call waiting and conditional call forwarding.

Call waiting is a useful telephony function. It allows the participant in a voice connection to temporary hold the connection and answer another incoming connection. The participant may later return to the original connection.

In the current GSP system, the call forwarding is partly conditional. A better approach is to change the call forwarding to fully conditional call forwarding that increases the flexibility of the system.

Finally, we may also enhance the security of the voice messages by using encryption algorithm such as DES [52], Blowfish [53], RSA [54] or even PGP [55] to encrypt the messages so that even if a hacker is able to copy the voice messages from the user's computer, the hacker cannot listen to the content inside the voice messages which is protected by encryption.

5.9. Alternatives

During the implementation of the GSP system, alternatives of approach and system to GSP system are searched. Up to now, there is no such system that cope with such specific problem. Two of the possible implementations to the GSP system are Netmeeting and OpenH323.

5.9.1. Netmeeting

Netmeeting is a voice over IP application developed by Microsoft. it supports point to point VoIP communication. Moreover, it provides a way for users to find other users on the Internet.

However, there are two shortcomings of Netmeeting compared with our GSP System in dealing with the discussed telephone problem.

First, Netmeeting depends heavily on Microsoft Windows operating system platform. Therefore, it is impossible to use Netmeeting on Unix or Linux. In contrast, although in our current GSP implementation, the Client can only run on Windows operating system platform too, it is expected in later version the GSP Client can also

be run on Unix and Linux platform. Therefore, the GSP system is more adaptable to different operating systems.

The second shortcoming, which is more serious, is that Netmeeting does not provide important telephone functions. For example, Netmeeting does not provide functions such as call forward and call transfer. Also, Netmeeting does not provide callers to record voice messages if the callee cannot be connected. These functions are very crucial in solving the "single telephone, multiple users" problem.

5.9.2. OpenH323

The second alternative implementation to our system is OpenH323 [56]. As mentioned on the OpenH323 homepage, "The OpenH323 project aims to create a full featured, interoperable, Open Source implementation of the ITU H.323 teleconferencing protocol that can be used by personal developers and commercial users without charge.", therefore OpenH323 is an open implementation of the H.323 protocol stack which is commonly used in voice over IP system. Moreover, a sample VoIP client is also included in the OpenH323 package that is called OpenPhone.

Although the OpenH323 project is an open source H.323 project, it is not quite suitable for our problem because the OpenH323 does not handle telephony operations such as call waiting, call forwarding, etc. Moreover, H.323 is a complex signaling protocol that is difficult to understand and use. Therefore, our GSP system is more suitable to handle the telephony problems similar to our office scenario.

6. CONCLUSIONS

Voice over IP is an innovative technology that allows voice to be transported over a computer network. Recent research in this area has brought many successful real life applications that change how we communicate with each other. VoIP clients like Microsoft Netmeeting and ICQ Phone [57] allow computer users to communicate with each other over the Internet. Popular free long distance call services such as DialPad [58] and Go2Call [59] utilize VoIP to provide free VoIP-based IP to PSTN long distance call services to US, Canada and other countries around the world. Instead of focusing VoIP in the Internet, this research is more Intranet-oriented. Our research focuses on three areas in voice over IP: how feasible is VoIP applied to an Intranet environment, how to implement various VoIP software components efficiently, and in what ways may VoIP technology be applied.

In searching the answers for the question "How feasible is VoIP applied to Intranet environment?", we have analyzed voice over IP thoroughly from different directions, such as the major components comprising VoIP, the performance factors affecting VoIP, the different requirements of Intranet VoIP and Internet VoIP, etc. In addition to a simple mathematical bandwidth estimation, a network simulation using network simulator 2 and network animator are conducted to find out the number of VoIP connections can be supported in some common Intranet configurations. The results of our calculation and simulation show that a reasonably fast Intranet environment should have no difficulty in supporting a reasonable number of VoIP connections.

In searching for the answers to the question "How to implement various VoIP software components efficiently?", various techniques for implementing VoIP technology are identified, presented and evaluated. We have studied different implementations of the VoIP client. They are VoIP client in JMF, Capture/Playback Enhanced VoIP client, Win32 C++ VoIP client and Win32 DirectSound C++ VoIP client. The study revealed the techniques for constructing VoIP client evolves, demonstrating how the choice of programming language and API can generate different performance levels. In summary, C++ is the preferred choice for

implementing performance related area in VoIP. In other areas, different programming languages and techniques may be used. Besides VoIP client, a simple real-time voice stream mixing server is also created to examine the mixing of multiple voice streams can be achieved in real time.

In searching for answers to the question "In what ways may VoIP technology be applied?", two prototype VoIP applications: pure IP-side VoIP-based call center and simple PBX are discussed and implemented. The first prototype VoIP application – pure IP-side VoIP based call center, combines the power of VoIP and collaborative browser technology to create a powerful and intuitive distributed computer learning environment. The second prototype VoIP application – Simple PBX, connects VoIP in Intranet with public switched telephone network and creates a PBX without using PBX hardware. These two applications prove the feasibility of creating that a diversity of useful VoIP applications in an Intranet environment. By using these two prototype VoIP applications as test beds, a comprehensive VoIP project called Graduate Second Phone (GSP) is implemented. GSP tries to solve a specific telephone problem. It combines the power of client/server architecture, C++ and Java hybrid programming, computer hardware telephony card and voice over IP technology to solve the problem successfully. Not only is the Graduate Second Phone project implemented, but extensive experiments have been done on it to test its performance on different areas.

The Linux operating system is becoming popular due to its open source nature and free development atmosphere. More and more applications based on Linux are being developed. It is possible to port our VoIP implementation to Linux platform. The Java based VoIP clients are platform independent. Therefore, they can be executed directly on Linux platform. For VoIP clients implemented in C++, they may be ported to Linux with slight modifications. Among the four main elements in these C++ VoIP clients: Call Setup, Media Capture/Playback, Media Encoding/Decoding and Media Transportation, only the Media Capture/Playback part is platform dependent. By modifying the Media Playing/Capture parts to utilize Linux platform audio drivers such as OSS/Free [60] and OSS [61], these C++ VoIP clients can be executed on Linux platform too. For the Graduate Second Phone project, the Server is platform independent. The Client can be ported to Linux by

using Linux VoIP core. For the Gateway, the telephony card is platform-dependent and depends whether the card vendor supports the card in Linux.

In fact, there are several reasons justifying why a Linux version of our VoIP implementations should be developed. The first reason is that Linux is now more mature in multimedia handling; resources such as Real Time Scheduling [62] and Low Latency Audio library [63] provide excellent low latency audio environment that VoIP highly needed. Moreover, a Linux operating system enables developers to fine-tune various features in the operating system and related software components to improve the smoothness and performance of VoIP. Such fine-tuning is not possible in Windows platform, where the source code of the operating system is not disclosed. Moreover, Linux can be run on embedded device in addition to desktop PC environment. Embedded Linux [64] refers to running Linux operating system in embedded device. Therefore, the porting of our implementations to embedded Linux creates embedded VoIP devices. However, embedded Linux poses several challenges to VoIP because there is less CPU computing power, less memory and limitations of audio capture/playback device.

For future research, there are three important software engineering key points that our implementations should try to achieve: OS independence, hardware independence and reusability. If our implementation is OS independent, then it can be run on any OS platform. If our implementation is hardware independent, then it can be run on any telephony card. If the reusability of our implementation is high, much of the code can be reused in porting for different OS/hardware even our implementation is not OS or hardware independent.

A natural question that arises from this research work is the comparison of our implementation with the H.323 implementation. There are two disadvantages for using H.323. The main disadvantage is that H.323 is a set of complicated signaling protocols. At least two to three protocols are needed in order to achieve the functionality of our work. The second disadvantage is that H.323 is not a free protocol. It is expensive to either buy protocol from a third party. On the other hand, it is time consuming to implement our own H.323 protocol stack based on the H.323 specification.

To conclude, this research has achieved its aim to consolidate the understanding of the building blocks for VoIP applications, especially at the network level. Our findings will enable future large-scale VoIP applications be constructed more efficiently and reliably.

Bibliography

[1] ITU-T H. Series, "H.323 – Line Transmission of Non-Telephone Signals – Visual Telephone Systems and Equipment for Local Area Networks which Provide a Non-Guaranteed Quality of Service," May 1996.

[2] M. Handley *et al.*, "SIP: Session Initiation Protocol," IETF RFC 2543, March 1999.

[3] Linden deCarmo, "Internet Telephony Protocols," *Dr. Dobb's Journal*, July 1999, pp. 30-39.

[4] M. Goncalves, "*Voice Over IP Networks*," McGraw-Hill, 1999.

[5] ITU Rec. G.711, "Pulse Code Modulation (PCM) of Voice Frequencies," 1972.

[6] ITU Rec. G.721, "32 kbit/s Adaptive Differential Pulse Code Modulation (ADPCM)," 1986.

[7] ITU Rec. G.723.1, "Dual Rate Speech Coder for Multimedia Communications Transmitting at 5.3 and 6.3 kbit/s," March 1996.

[8] ITU Rec. G.727, "5-, 4-, 3- and 2-bits/sample embedded Adaptive Differential Pulse Code Modulation (ADPCM)," 1990.

[9] ITU Rec. G.729 Annex A, "Reduced Complexity 8 kbit/s CS-ACELP Speech Codec," November 1996.

[10] E. Jorg, J. V. Hans, "*GSM Switching, Services and Protocols*," John Wiley & Sons, 1999, pp. 89.

[11] H. Schulzrinne, S. Casner, R. Frederick and V. Jacobson, "RTP: A Transport Protocol for Real-Time Applications," IETF Audio-Video Transport Working Group RFC 1889, January 1996.

[12] H. Schulzrinne, A. Rao, R. Lanphier, "Real Time Streaming Protocol (RTSP)," IETF RFC 2326, 1998.

[13] J. B. Postel, "Transmission Control Protocol," IETF RFC 793, 1981.

[14] J. B. Postel, "User Datagram Protocol," IETF RFC 768, August 1980.

[15] P. Goyal, A. Greenberg, C. R. Kalmanek, W. T. Marshall, P. Mishra, D. Nortz, K. K. Ramakrishnan, "Integration of call signaling and resource management for IP telephony," *IEEE Internet Computing*, Volume: 33, May-June 1999, pp. 44-52.

[16] A. M. Grilo, P. M. Carvalho, L. M. Medeiros, M. S. Nunes, "VTOA/VoIP/ISDN telephony gateway," *Proceedings of 2nd International Conference on ATM*, 1999, pp. 230-235.

[17] M. Podolsky, C. Romer, S. McCanne, "Simulation of FEC-Based Error Control for Packet Audio on the Internet," *Proceedings of IEEE INFOCOM*, 1998, pp. 505-515.

[18] L. Zhang, S. Deering, D. Estrin, S. Shenker, D. Zappala, "RSVP: a new resource ReSerVation Protocol," *IEEE Network*, Volume: 7 5, September 1993, pp. 8-18.

[19] R. Ramaswamy, "Design of a secure packet voice communication system in wide area networks," *Proceedings of IEEE International Carnahan Conference on Security Technology*, 1990, pp. 42-50.

[20] T. Yletyinen, R. Kantola, "Voice Packet Interarrival Jitter Over IP Switching," *Proceedings of Telecommunication Symposium*, 1998, pp. 16-21.

[21] "The Network Simulator – ns-2". Hyperlink at http://www.isi.edu/nsnam/ns/
[22] "OTcl – MIT Object Tcl (Hacked version for Tcl 8.0)". Hyperlink at http://www.jessikat.demon.co.uk/otcl.html
[23] "NAM: Network Animator". Hyperlink at http://www.isi.edu/nsnam/nam/index.html
[24] "comp.lang.awk FAQ". Hyperlink at http://www.landfield.com/faqs/computer-lang/awk/faq/
[25] "CSMA/CD". Hyperlink at http://www.erg.abdn.ac.uk/users/gorry/course/lan-pages/csma-cd.html
[26] "JMF Home Page". Hyperlink at http://java.sun.com/products/java-media/jmf/
[27] "Archives of JMF-INTEREST@JAVA.SUN.COM". Hyperlink at http://archives.java.sun.com/archives/jmf-interest.html
[28] "JMF 2.1.1 Beta 2 – Features". Hyperlink at http://developer.java.sun.com/developer/earlyAccess/jmf/jmf-features.html
[29] "Win32 API". Hyperlink at http://msdn.microsoft.com/library/psdk/portals/win32start_1n6t.htm
[30] "Microsoft DirectX – What's New". Hyperlink at http://www.microsoft.com/directx/
[31] "The GSM 06.10 lossy speech compression library and its applications". Hyperlink at http://kbs.cs.tu-berlin.de/~jutta/toast.html
[32] "Jori's RTP library". Hyperlink at http://lumumba.luc.ac.be/jori/jrtplib/jrtplib.html
[33] "Audio Compression Manager". Hyperlink at http://msdn.microsoft.com/library/psdk/multimed/audcomp_3kc2.htm
[34] "audio/msgsm Specification". Hyperlink at http://www.ietf.org/proceedings/99jul/slides/vpim-agenda-99jul/sld020.htm
[35] "MSDN Online". Hyperlink at http://www.ietf.org/proceedings/99jul/slides/vpim-agenda-99jul/sld020.htm
[36] "International Telecommunication Union (ITU) Home Page". Hyperlink at http://www.itu.int/
[37] V. Hardman, A. Sasse, M. Handley and A. Watson, "Reliable Audio for Use over the Internet," *Proceedings of INET*, June 1995.
[38] "DU Meter". Hyperlink at http://www.hageltech.com/dumeter/
[39] H. C. Ho, "Three-Tier Feature-based Collaborative Browsing for Computer Telephony Integration," MPhil Thesis, Department of Computer Science and Engineering, The Chinese University of Hong Kong, June 2001.
[40] C. C. Leung, "An Intelligent IP-based Call Center with Fault Tolerance Design," MPhil Thesis, Department of Computer Science and Engineering, The Chinese University of Hong Kong, June 2001.
[41] "ADI Service Developer's Manual," Natural Microsystem, 1999.
[42] "Natural Call Control Service Developer's Reference Manual," Natural Microsystem, 1999.
[43] "Fusion AG TRAU Developer's Reference Manual," Natural Microsystem, 1999.
[44] "Java 2 Platform, Standard Edition". Hyperlink at http://java.sun.com/j2se/1.3/

[45] "Java 2 Runtime Environment, Standard Edition". Hyperlink at http://java.sun.com/j2se/1.3/jre/
[46] "Sun/NeXT audio(AU) format". Hyperlink at http://ibis.nott.ac.uk/guidelines/ch62/chap6-2-F-2.html
[47] "Winamp". Hyperlink at http://www.winamp.com
[48] "Windows Media". Hyperlink at http://www.microsoft.com/windows/windowsmedia/EN/default.asp
[49] "Effective Layout Management". Hyperlink at http://developer.java.sun.com/developer/onlineTraining/GUI/AWTLayoutMgr/index.html
[50] "Hong Kong Telecom". Hyperlink at http://www.hkt.com
[51] "Syntrillium Software: Cool Edit Pro and Cool Edit 2000". Hyperlink at http://www.cooledit.com/
[52] "Data Encryption Standard (DES)". Hyperlink at http://www.itl.nist.gov/fipspubs/fip46-2.htm
[53] "The Blowfish Encryption Algorithm". Hyperlink at http://www.counterpane.com/blowfish.html
[54] "Welcome to RSA Security Inc.". Hyperlink at http://www.rsa.com
[55] "The International PGP Home Page". Hyperlink at http://www.pgpi.org/
[56] "Open H323 Project". Hyperlink at http://www.openh323.org/
[57] "ICQ Phone". Hyperlink at http://www.icq.com/icqphone/
[58] "Dialpad". Hyperlink at http://www.dialpad.com
[59] "Go2Call". Hyperlink at http://www.go2call.com
[60] "Linux Sound System". Hyperlink at http://www.linux.org.uk/OSS/
[61] "Open Sound System". Hyperlink at http://www.opensound.com/oss.html
[62] "Linux Real-Time Scheduler". Hyperlink at http://www1.mvista.com/realtime/rtsched.html
[63] "Linux high performance low latency audio". Hyperlink at http://www.gardena.net/benno/linux/audio/
[64] "LinuxDevices.com... the embedded Linux portal". Hyperlink at http://www.linuxdevices.com/