國立暨南國際大學資訊工程學系

碩士論文

SIP 協作系統的可攜式使用者介面設計與架構

Design and Architecture of a Portable User Agent in SIP Collaboration Systems

指導教授：吳坤熹博士

研究生：張兢真

中華民國九十七年 九月

致謝

這研究所的這兩年中，在各方面都有許多成長，在吳坤熹老師的指導下也學到許多，像是文件的撰寫、做研究應有的態度、解決問題的能力以及清楚的思路等等，對我在未來的職場或人生十分有幫助。另外，也要感謝胡大雄博士，教導我許多程式撰寫的技巧以及增加效能的方法等等，讓我受益良多。發現原來一個小小的程式，也是要經過許多的測試才算是一個完整的程式。

這個研究的部分經費是由行政院國家科學委員會所贊助，計畫編號為 NSC96-2622-E-260-002-CC3 。我們也很感謝國尊科技股份有限公司(http://www.kinghood.com/)提供的 Cyberhood 系統，讓我們成功地完成了這篇論文研究結果。此外，Cyberhood 系統的分散式架構也提供了一個具有延展性的環境，有助於未來本系統進一步發展。

論文名稱 ：

SIP 協作系統的可攜式使用者介面設計與架構

校院系 ： 國立暨南國際大學資訊工程研究所　　頁數:71 頁

畢業時間 ： 97 年 9 月　　學位別 ： 碩士

研究生 ： 張兢真　　指導教授 ： 吳坤熹博士

中文摘要

企業資源規劃(Enterprise Resource Planning，簡稱 ERP)系統對大企業來說，是一個功能強大的工具。它可以將一個企業裡所有資源有效地整合起來，使得企業可以有效地進行運作。可預期的，一個發展良好的企業需要有足夠的客源，因此客戶對企業來說，也是十分重要的一項「資源」。客戶關係管理(Customer Relationship Management，簡稱 CRM) 與 ERP 整合的系統可以幫助企業鞏固與客戶良好的互動。此外，網路電話(Voice over Internet Protocol，簡稱 VoIP)已成為目前主流的通訊技術；相較於傳統的電話系統，它可以讓使用者以較低的成本在網路上進行語音溝通。

在本篇論文中，提出了將一個現有的 CRM 系統與 VoIP 服務做結合的架構，客戶與員工之間可以透過語音直接溝通。一方面讓客戶對產品有疑問或顧慮的時候，可以直接與公司溝通。另一方面，員工與員工之間也可以透過網路直接討論工作細節或是通知臨時的訊息。為了實作這個架構，我們設計了一個可攜式的 Session Initiation Protocol(簡稱 SIP)網路電話，與 Cyberhood 的 ERP 系統做結合。

關鍵字 ： 協作系統、CRM、ERP、可攜式、SIP、VoIP

Title of Thesis :

Design and Architecture of a Portable User Agent in SIP Collaboration Systems

Name of Institute : Department of Computer Science and Information Engineering,

National Chi Nan University Page : 71 pages

Graduation Time : 9/2008 Degree Conferred : Master

Student Name : Ching –Chen Chang Advisor Name : Quincy Wu

Abstract

Enterprise Resource Planning (ERP) systems have become a basic and powerful tool in large enterprises. It provides the function to compactly integrate all kinds of resource in an organization, which is crucial to enterprise operation. Naturally, customers are important to enterprises. Integrating Customer Relationship Management (CRM) systems with ERP systems can help enterprises to maintain strong relationship with customers. In addition, a popular technology, Voice over Internet Protocol (VoIP), allows users to have real-time communication through the Internet.

In this thesis, we propose an architectural solution to integrate the VoIP service to an existing CRM system. On the one hand, customers can freely communicate with the enterprise when they have some problems with the products. Furthermore, employees can discuss with each other directly through the Internet. To verify the architecture, a portable SIP phone (PSP) was designed to be integrated to an existing ERP system – Cyberhood , which also provides the CRM system service.

Keyword: Collaboration, CRM, ERP, Portable, SIP, VoIP

Lists of Figures

Chapter 1. Introduction

In a running enterprise, there are lots of procedures including manufacturing, cooperating with other corporations, selling products to customers, etc. There is a lot of information that has to be managed, stored, shared, and protected during the procedures. The longer time the enterprises operate, the more information they have to maintain. One century ago, these records had to be written on paper so that it wasted much time and cost. Nowadays with the support of computer systems, each record can be stored in a database which makes it a simpler task to store and retrieve data. However, computerization may cause a new problem that the information in different computer systems is difficult to be integrated. In an enterprise, every department may develop its own system with different functions, different databases and different interfaces to manage its own information. For example, the payroll department needs a system to calculate the compensation of each personnel to prepare the paycheck, while the finance department stores each financial business transaction by its own system. Thus, we need a system to combine all the functions in currently independent applications. What we require is an ERP (Enterprising Resource Planning) [2] system which efficiently integrates and manages all the resources in an enterprise.

In addition to the traditional resources which ERP systems can manage, it will be even better if the real-time communication service can be included to support a good communication between customers and employees. The function of CRM (customer relationship management) [10] is often included in an ERP system to allow employers to keep in touch with co-workers and customers via email, fax, and telephony systems. Today, VoIP(Voice over IP) plays a more and more important role as a real-time communication service, and a multimedia communication protocol SIP (Session

Initiation Protocol) was defined by IETF (Internet Engineering Task Force) to manage VoIP sessions. Lots of VoIP devices and applications are implemented based on SIP now. SIP is a powerful and stable protocol which will be discussed later. VoIP provides the real-time audio communication function which allows two or more parties to communicate with each other in different places. Therefore, in this thesis, we propose to integrate the SIP VoIP with CRM systems.

1.1. ERP systems

The origin of ERP system could be traced back to MRP (Material Requirement Planning) and MRPII (Manufacturing Resource Planning). MRP was created in 1970s because manufacturing companies needs an automatic mechanism to calculate which material they need, when they need and how many they need. With further improvement, MRPII (which was evolved from MRP in 1980s) supports a centralized database to maintain all the data, integrates all the various processes which contain materials, finance, accounting, etc, and intends to keep the lowest inventory.

Nowadays enterprises need systems to help them integrating all the business data and processes of an organization into a unified system to support purchasing, inventories, tracking orders, interacting with suppliers, providing customer support, and many related activities. Therefore, an ERP system which standardizes the whole system with lower maintenance effort becomes critical to enterprise competence. In an ERP-centric environment, workflow is the delivery mechanism by which the data required for transactional processing are provided. A common business process can be established across these platforms by deploying a common workflow scheme that incorporates the ERP platforms as different processing activities [2].

1.2. CRM system

Nowadays, customers are the most important assets of an enterprise. However, it is difficult to maintain the loyalty of customers because their needs and expectations change rapidly. In order to reach customers easily, CRM systems are usually integrated to ERP systems. A CRM system helps an enterprise managing customer relationships in an organized way and aims at improving customer satisfaction so that business profits can be maximized. Generally a CRM system contains three major areas as shown in Figure 1. It includes *Marketing* which personalizes customer preference, *Sales* which analyze the real-time marketing data to increase the revenue, and *Services* which let customers directly communicate with employees.

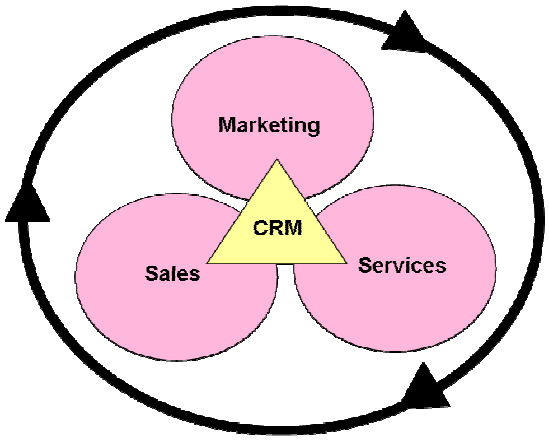

Figure 1. The simple architecture of a CRM system

1.3. Popularity of VoIP

Recently, VoIP [1] is rapidly growing and becoming a mainstream telecommunication service. Today, there are a few famous VoIP services, such as

Skype [23], X-Lite [22], etc. It reduces the waste of time and cost in transportation between two or more parties. The real-time communication capability also shortens the distance between employees and customers. Compared with traditional PSTN (Public Switched Telephone Network) systems, VoIP takes lower cost in both deployment and operation. As a matter of fact, it is possible for a company to save the cost of buying all the PSTN phones, wiring, and the call charges. By building a VoIP system, it only requires the network cables and the hubs to connect computers to the Internet. Since many companies already have the networking facility, they just have to install appropriate VoIP software in each computer, and then they can utilize it to talk with other people via VoIP instead of PSTN. It also allows users to make a call between VoIP systems and legacy PSTN phones through an appropriate gateway.

1.4. Advantage of VoIP to be integrated with CRM

Customers need supports frequently when they are using some products. Sometimes even before they are using the products, when they are filling in an on-line registration form of a product, they often have questions which need to be clarified and answered. Moreover, sometimes customers will send an e-mail to the company, and wait a long time for the response. However, some problems and answers are difficult to be described clearly by emails. Real-time communication can solve the customer's problem quickly. They can just make a call by PSTN or VoIP, and get the answers from the customer service personnel directly [12].

Sometimes employees may want to discuss some details with other departments. If they use PSTN phone to make a call, it incurs the call charge. On the contrary, if they use VoIP to make a call, lots of money will be saved chronically. Currently, many CRM systems basically provide features such as shared calendar, email, bulletin board,

etc. Employees can use it to announce some events, check important information, and discuss with group members. Since VoIP provides the audio real-time communication service which allows employees to communicate with each other freely and quickly, if we can further integrate VoIP with CRM, we can help enterprises reducing the cost and time. With the VoIP support, enterprises can operate more efficiently, and get strong relationship with customers. This thesis aimed to use a case study methodology to identify what ways (if at all) the implementation of VoIP can benefit the ERP system in an enterprise.

Chapter 2. Case Study: An Existing ERP System

To illustrate the architectural approach in integrating VoIP to an existing ERP system, we took a commercial ERP system Cyberhood [13] as an example.

2.1. The architecture of Cyberhood

Cyberhood is a powerful web-based collaborative software program developed by Kinghood Corporation. To provide better performance, independent servers will be allocated to run services like authentication and dispatching, application services, streaming and downloading, and message pushing. This distributed design also provides fault-tolerance, so that in case a single server crashes, it does not hinder the operation of the whole system. By ubiquitous access through the Internet, Cyberhood allows users to control and manage the business processes anywhere and anytime.

Figure 2 shows a variety of servers in current Cyberhood system which includes AD Servers that manage authentication and dispatching of tasks, AP Servers that support lots of application services, Communication Servers that support real-time communication like instant messaging, Download Servers that manage the files to be uploaded and downloaded, Mail Servers that transfer electronic mail messages, and DB Servers that store all kinds of data.

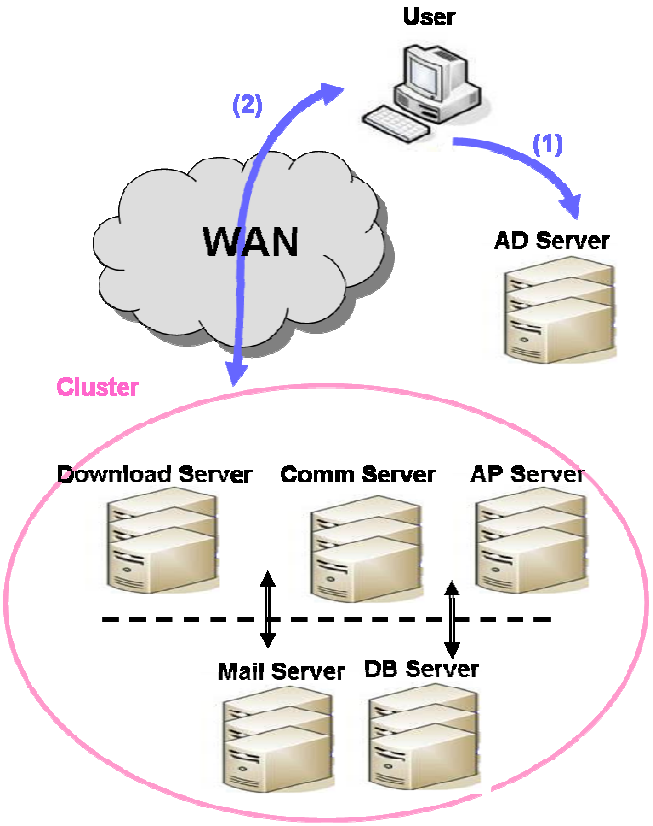

Figure 2. The architecture of the Cyberhood system.

According to its distributed design, Cyberhood allows enterprises to deploy a cluster of servers to support any single service. The dynamic task dispatching supports load balancing among servers to avoid any individual server being overburdened.

On the client side, Cyberhood provides a software program Kiwi [13] to support the text-based real-time communication, whose graphical user interface (GUI) is shown in Figure 3. Kiwi supports text-based instant messages, messaging log, email access and presence service. It allows a user to send text messages to a single user or multi-users at the same time. It is also a *portable* application that does not need to be installed onto the permanent storage device. Instead, it can be stored on a USB flash

drive. Users can easily connect the USB flash drive to a computer and begin running the application without any installation procedure. However, this convenient utility of Cyberhood in instant communication only delivers text messages. In this thesis, we shall illustrate how to integrate the real-time audio conversation capability with the Cyberhood ERP system, to make it more powerful and efficient. Certainly, taking into account the convenience of portability, we require this newly developed VoIP software to be portable as aforementioned as Kiwi.

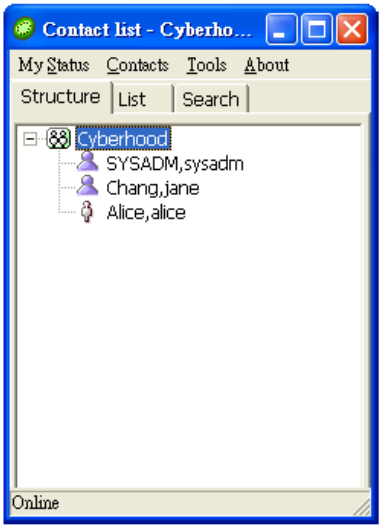

Figure 3. The graphical user interface of Kiwi.

2.2. How to enable VoIP service

2.2.1. Protocol overview

2.2.1.1. SIP

SIP (Session Initiation Protocol) [5], which is a multimedia conferencing

standard developed by IETF (Internet Engineering Task Force), enables a VoIP application to achieve comparable high quality and reliability as traditional telephony services. SIP also has the advantage of being scalable, flexible, and easy to implement. It supports lots of services like 3PCC (Third Party Call Control) [6], NAT (Network Address Translation) [7] handling, etc. SIP defines six basic methods including INVITE, ACK, OPTIONS, BYE, CANCEL and REGISTER to accomplish the procedure of call setup and teardown, location registration, and so on. In a SIP-based VoIP system, we usually need a SIP Proxy Server to handle SIP signaling, where all the SIP signaling must be sent to the SIP Proxy Server first. A simple SIP flow of an INVITE request is shown in Figure 4.

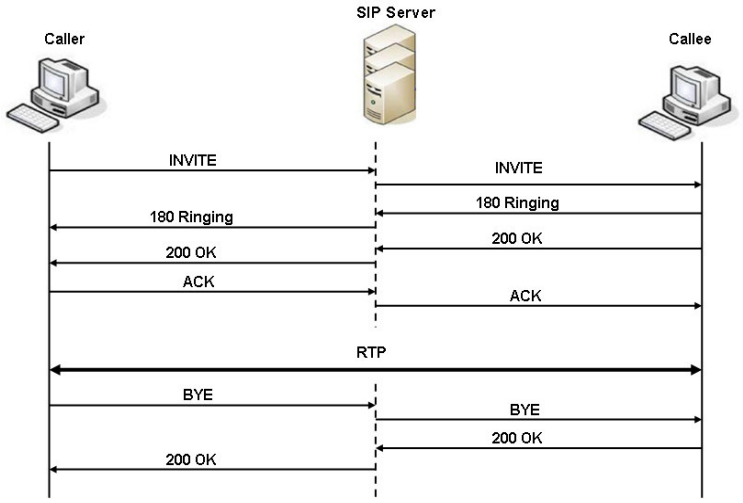

Figure 4. SIP call set up and tear down.

Figure 5 shows the contents of a SIP message. We can see that a SIP message consists of three parts: a status line (SIP/2.0 200 OK), the message header (as shown in the upper red rectangle), and the message body (as shown in the lower blue rectangle). In the message header, there are lots of fields about the route handling (where the packet comes from and where the packet is sent to). The message body in

SIP uses SDP (Session Description Protocol) [8] which we will discuss in the next section.

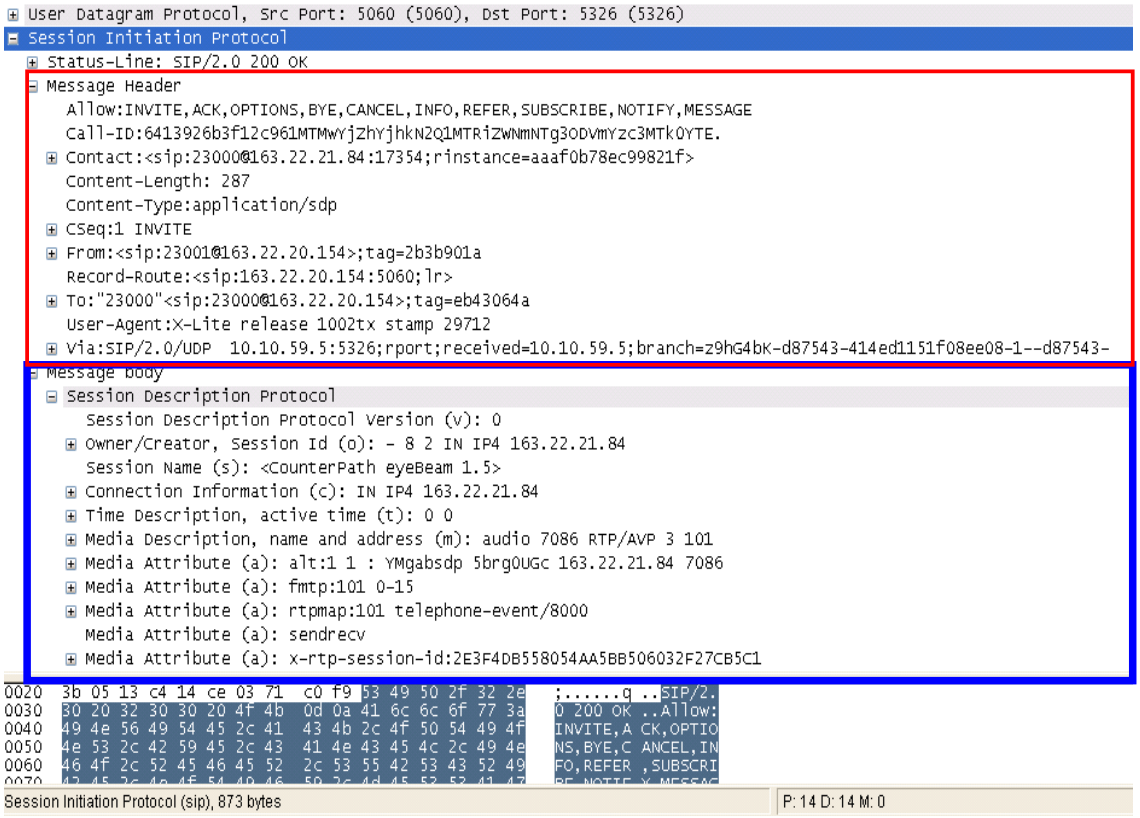

Figure 5. A SIP message.

2.2.1.2. SDP

The Session Description Protocol (SDP) [8] embedded in SIP is a format of multimedia session description for the purpose of session announcement, session invitation and other forms of multimedia session initiation. An SDP session description includes the following information: the media type, the supporting codec, the network port and the IP address for media transport, etc. It provides information for two parties to negotiate the media type and encoding format for further communication. We show some important SDP fields in the following:

v (protocol version)

o (originator and session identifier)

s (session name)

t (time the session is active)

c (connection information)

m (media descriptions)

a (media attribute lines)

The c field contains the connection information including network type, address type and connection address. The m field contains supporting codec, media type and media port. A complete example of SDP is shown in the lower blue rectangle of Figure 5.

2.2.1.3. RTP

The Real-time Transport Protocol defines a standardized packet format for delivering audio and video data over the Internet. It was developed by the Audio-Video Transport Working Group of the IETF and defined in RFC 3550 [4]. The size of the RTP header is 12 bytes. The RTP header contains the version of RTP, the payload type, the timestamp, etc, as shown in Figure 6. In this section, we shall describe some fields which are relevant to our system. The initial value of the *sequence number* is random, and it increments by 1 after a RTP packet is sent. With the sequence number, the receiver can check whether any packet is lost. The value of *synchronization source* (SSRC) identifies the source of a stream of RTP packets. We can see an example in Figure 7. The media type is audio, and format is G.711. The SSRC identifiers of the RTP packets which are sent from Caller to Callee

and from Callee to Caller are 2328960936 and 1693736445, respectively.

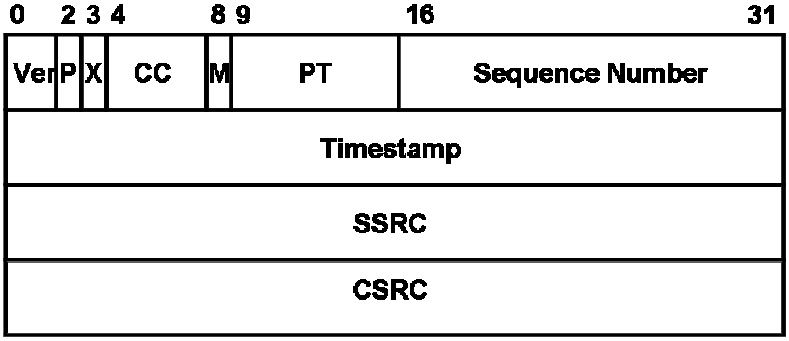

Figure 6. RTP header.

No. .	Time	Source	Destination	Protocol	Info
1	0.000000	10.10.20.199	163.22.20.151	RTP	Payload type=ITU-T G.711 PCMA, SSRC=2328960936, Seq=5476, T
2	0.002075	163.22.20.151	10.10.20.199	RTP	Payload type=ITU-T G.711 PCMA, SSRC=1693736445, Seq=32302,
3	0.020466	10.10.20.199	163.22.20.151	RTP	Payload type=ITU-T G.711 PCMA, SSRC=2328960936, Seq=5477, T
4	0.021629	163.22.20.151	10.10.20.199	RTP	Payload type=ITU-T G.711 PCMA, SSRC=1693736445, Seq=32303,
5	0.040062	10.10.20.199	163.22.20.151	RTP	Payload type=ITU-T G.711 PCMA, SSRC=2328960936, Seq=5478, T
6	0.043706	163.22.20.151	10.10.20.199	RTP	Payload type=ITU-T G.711 PCMA, SSRC=1693736445, Seq=32304,
7	0.059704	10.10.20.199	163.22.20.151	RTP	Payload type=ITU-T G.711 PCMA, SSRC=2328960936, Seq=5479, T
8	0.062635	163.22.20.151	10.10.20.199	RTP	Payload type=ITU-T G.711 PCMA, SSRC=1693736445, Seq=32305,
9	0.080122	10.10.20.199	163.22.20.151	RTP	Payload type=ITU-T G.711 PCMA, SSRC=2328960936, Seq=5480, T
10	0.082165	163.22.20.151	10.10.20.199	RTP	Payload type=ITU-T G.711 PCMA, SSRC=1693736445, Seq=32306,
11	0.099722	10.10.20.199	163.22.20.151	RTP	Payload type=ITU-T G.711 PCMA, SSRC=2328960936, Seq=5481, T
12	0.102683	163.22.20.151	10.10.20.199	RTP	Payload type=ITU-T G.711 PCMA, SSRC=1693736445, Seq=32307,
13	0.120171	10.10.20.199	163.22.20.151	RTP	Payload type=ITU-T G.711 PCMA, SSRC=2328960936, Seq=5482, T
14	0.122213	163.22.20.151	10.10.20.199	RTP	Payload type=ITU-T G.711 PCMA, SSRC=1693736445, Seq=32308,
15	0.139683	10.10.20.199	163.22.20.151	RTP	Payload type=ITU-T G.711 PCMA, SSRC=2328960936, Seq=5483, T
16	0.144276	163.22.20.151	10.10.20.199	RTP	Payload type=ITU-T G.711 PCMA, SSRC=1693736445, Seq=32309,
17	0.160015	10.10.20.199	163.22.20.151	RTP	Payload type=ITU-T G.711 PCMA, SSRC=2328960936, Seq=5484, T
18	0.161757	163.22.20.151	10.10.20.199	RTP	Payload type=ITU-T G.711 PCMA, SSRC=1693736445, Seq=32310,
19	0.179598	10.10.20.199	163.22.20.151	RTP	Payload type=ITU-T G.711 PCMA, SSRC=2328960936, Seq=5485, T

Figure 7. The example of SSRC.

Payload type (PT) specifies the format of RTP payload. RTP supports lots of payload types, such as PCM (A-law and μ-law), G.729, GSM. Figure 8 lists a few codecs and the corresponding payload types. For example, the GSM codec has payload type 3, and G.729 codec has payload type 18. The audio data are carried as payload following the 12-byte RTP header. For the destination to successfully decode the audio data using the correct codec, the receiver must be notified of the payload type in advance, as described in the SDP m field mentioned in the previous section.

PT	Codec
0	PCMU
2	G.721
3	GSM
8	PCMA
18	G.729

Figure 8. RTP payload types with corresponding codecs.

2.2.2. NAT

2.2.2.1. Network Address Translation

NAT is a device which rewrites the source/destination IP address and port number of IP packets as they pass through the NAT router. The local network behind NAT uses the designated private IP addresses [11]. Using private IP address alleviates the shortage of IPv4 address, because the private IP address can be re-used again and again. The Internet Assigned Numbers Authority (IANA) reserved three blocks of IP address space for private networks. The range of private IP addresses is defined in RFC 1918 [11] and shown in Figure 9. Although using private IP addresses mitigates the depletion of IP addresses, the hosts and devices with private IP addresses can not be reached from the public Internet. The private address can only be reached by hosts within the local network, and routers on the public Internet are normally configured to discard any traffic destined to private IP addresses.

Address Range	Mask
10.0.0.0 – 10.255.255.255	10.0.0.0/255.0.0.0
172.16.0.0 – 172.31.255.255	172.16.0.0/255.240.0.0
192.168.0.0 – 192.168.255.255	192.168.0.0/255.255.0.0

Figure 9. The range of private IP addresses

As shown in Figure 10, the NAT device has two interfaces (and thus two IP addresses). One is 163.2.2.1 which is public and the other is 10.0.0.1 which is private. Several clients under the NAT can use the public IP address to connect to the Internet from different ports. The NAT maintains an internal mapping table and can re-write the source and/or destination addresses of IP packets as they traverse the NAT. Using NAT enables multiple hosts on a private network to access the Internet by a single public IP address. For example, if a user (IP address 10.0.0.2; port 1052) under NAT wants to connect to the Internet, NAT will create an entry in its mapping table as shown in Figure 11. In this example, the entry maps IP address 10.0.0.2 with port 1052 to 163.2.2.1 with port 2000.

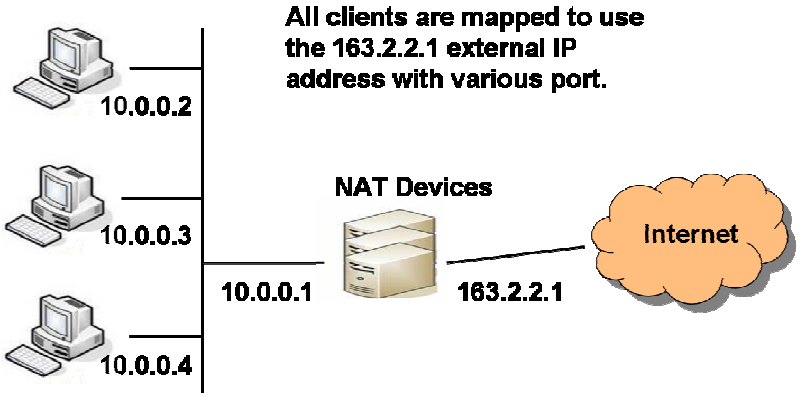

Figure 10. An example of NAT

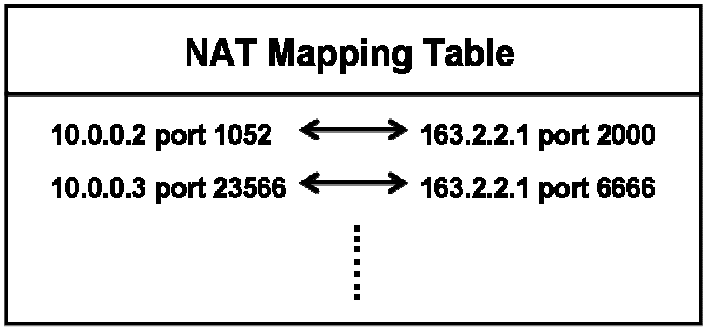

Figure 11. NAT mapping table

For applications running with the client/server model, NAT is a good mechanism which allows more people to connect to the Internet by occupying only a single public IP address. However, there is a problem when VoIP applications are deployed in a NAT environment. We shall use the following two figures to illustrate the problem of NAT introduced to VoIP applications. The normal scenario when users are not behind NAT is shown in Figure 12 and the scenario when a user agent (UA) is behind NAT is shown in Figure 13. In Figure 12, there is a SIP session created between UA1 and UA2. During media negotiation, UA1 is told that it should send RTP packets to 163.22.2.3 (which is the IP address of UA2), and UA2 is told that it should send RTP packets to 163.22.2.2 (which is the IP address of UA1). Both UA1 and UA2 have public IP addresses, so they can communicate with each other directly. On the other hand, in Figure 13, UA1 is behind NAT and its IP address is private. In this scenario after the session is established, UA1 could send the RTP packets to 163.22.2.3 (which is the IP address of UA2). However, when UA2 sends the RTP packets back to 192.168.0.2 (which is the IP address of UA1), the packets will be discarded by routers on the Internet because of its private destination address. Thus, UA2 cannot send RTP packets to UA1 which is behind the NAT.

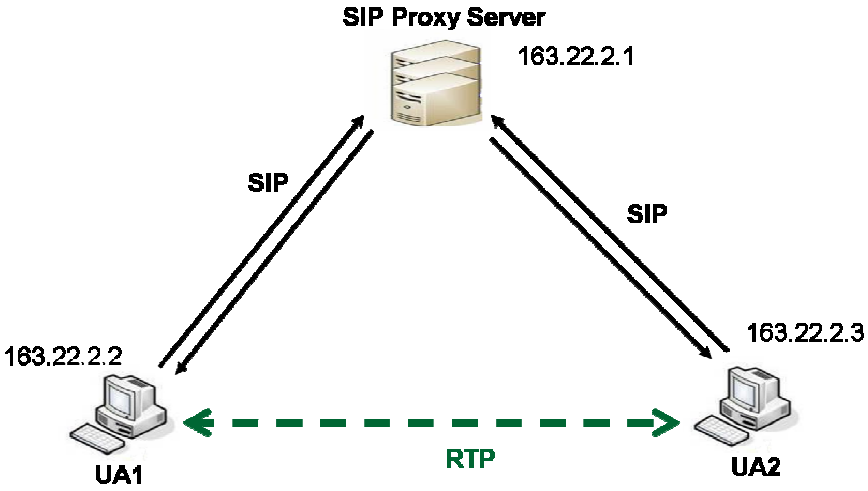

Figure 12. The situation when users are not behind NAT

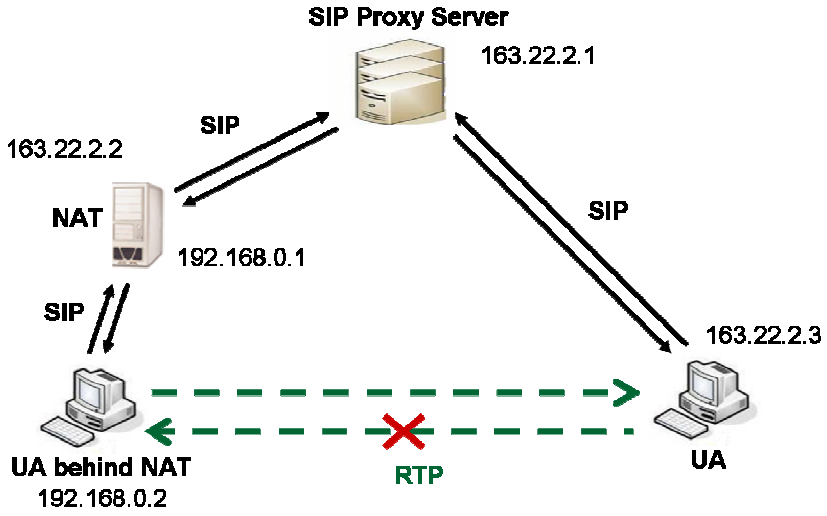

Figure 13. The situation when a user is behind NAT

2.2.2.2. NAT traversal

Today there are a few methods to handle NAT traversal [9], such as STUN (Simple Traversal of UDP through NAT), TURN (Traversal Using Relay NAT), ICE (Interactive Connectivity Establishment), RTP Proxy Server, etc. The solution we choose is RTP Proxy Server [20] because it is a solution which does not require client-side support and can traverse all kinds of NAT types [9] . The process of NAT traversing by RTP Proxy Server is illustrated in Figure 14. To successfully deliver the RTP packets through the NAT, two parties do not send RTP packets to each other directly. Instead, they send to the RTP Proxy Server which has a public address. In this way, they can communicate through the RTP Proxy Server. Therefore, to extend a CRM system with VoIP support, we need a SIP Proxy Server and (a few) RTP Proxy Servers for NAT traversal. All these servers need to possess public IP addresses.

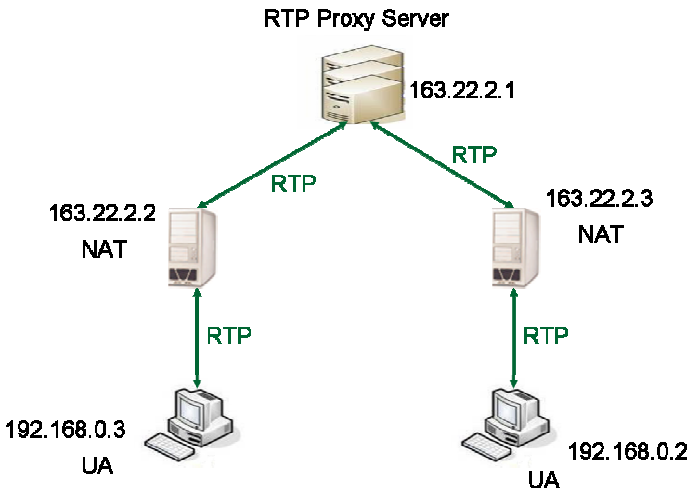

Figure 14. Traversing NAT by RTP Proxy Server.

First, we need to modify the configuration of the SIP Proxy Server and let it support NAT handling. We shall use two situations to illustrate how the SIP Proxy Server helps in handling NAT traversal. The first situation is registration. When a user

agent behind NAT registers at a SIP Proxy Server, the user agent sends a SIP request (which only contains its private IP address) to the SIP Proxy Server. The SIP Proxy Server inspects whether the user is behind NAT. If it is, the SIP Proxy Server marks it in the user location database. After that, the SIP Proxy Server sends back a "200 OK" response by adding the public IP address and port of NAT in the Via header field which records the routing path. The simple SIP flow of registration with NAT handling is shown in Figure 15.

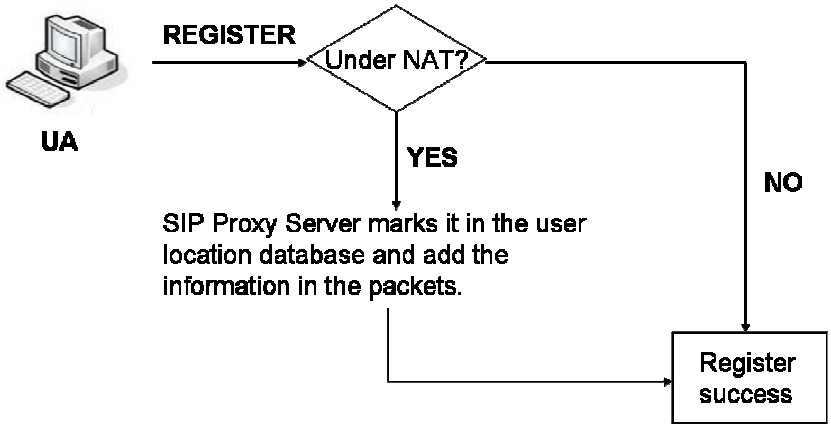

Figure 15. Registration flow with NAT.

The second situation is call setup. When a user sends an INVITE request to the SIP Proxy Server, the SIP Proxy Server checks whether both Caller and Callee are behind NAT in the location database. If one of them is or both of them are, the SIP Proxy Server adds the public IP address and port of NAT in the Via header field of the response and modifies the connection information and the port of media descriptions in the SDP. The connection information describes the IP address where the audio packets are sent to, and the port of media descriptions is the UDP port to which the media stream is sent. The simple SIP flow of an INVITE request with NAT is shown in Figure 16.

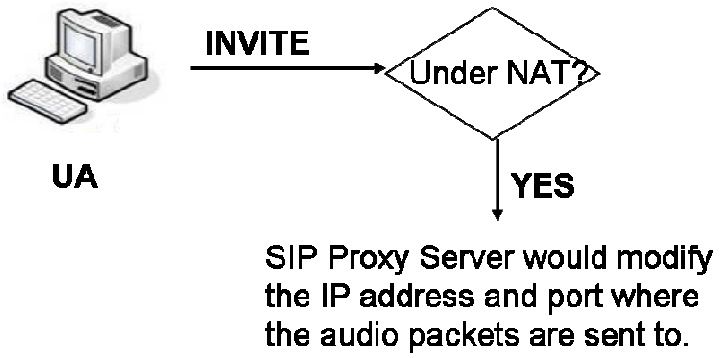

Figure 16. A flow of an INVITE request with NAT.

After both the SIP Proxy Server and the RTP Proxy server are set up, we can traverse NAT by those servers. The configuration file is discussed in detail in the appendix.

2.2.3. The architecture of SIP real-time communication service

In the following, we shall illustrate the architecture of the Cyberhood ERP system, and how to add the SIP real-time communication feature into it [3]. In Figure 17, the oval represents an existing Cyberhood system, while the inner rectangle (SIP Proxy Server and RTP Proxy Servers) represents the new components to be added to this system, and the rectangle at the bottom shows the components in a UA. In order to add the audio-based real-time communication services into this ERP system, we need a SIP Proxy Server that supports registration, authentication and NAT handling, and we also need a few RTP Proxy Servers that support load balancing. Each UA needs Kiwi, and a portable SIP-based program which we call it PSP (portable SIP phone) [19]. The reason for logical separation of the SIP Proxy Server and RTP Proxy

Servers is that, if a RTP Proxy Server is overburdened with many RTP streams, we can bring up more RTP Proxy Servers to share the load. Moreover, the SIP Proxy Server would not be affected by the RTP traffic, so it can always respond quickly to SIP requests. For small systems without heavy RTP traffic, it is also possible to collocate the SIP Proxy Server and the RTP Proxy Server on a single host to save cost.

In order to reduce the risk that adding the new audio conversation function may accidentally interfere with the original ERP system, a loosely-coupled approach was adopted in the design. Therefore, we used two independent servers to support the original Cyberhood service and the new VoIP service, respectively. For users to communicate with each other, they need to register both at the AD Server and SIP Proxy Server. The registration steps are described as below.

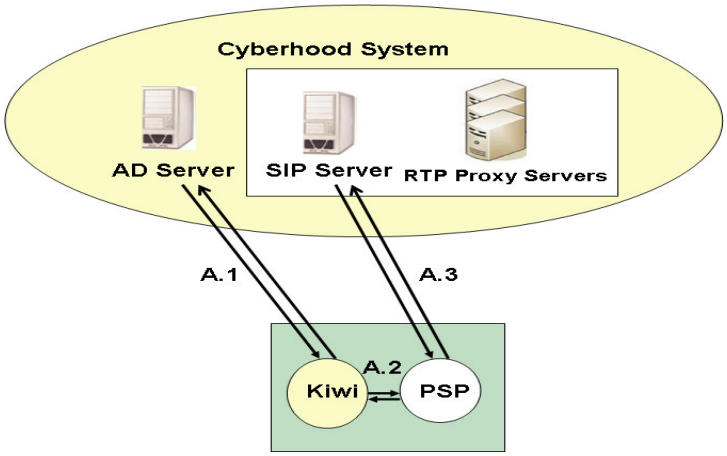

Figure 17. The flow of registration.

Step A.1. When a user logins by Kiwi, it would access the AD server and get information that contains the corresponding SIP account, password, IP address of the SIP server and the corresponding account of Callee.

Step A.2. Kiwi would invoke PSP. Suppose the account of the user is “Bob”, and the

password is “PASSWD”, IP address of the SIP server is “163.2.2.1” and the account of the called party is “Alice”. Then, Kiwi calls ShellExecute() function which is a win32 API function to trigger PSP with a command “`C://…../psp.exe Bob PASSWD 163.2.2.1 Alice`”.

Step A.3. PSP registers at the SIP Proxy Server with the username “Bob” and password “PASSWD” (obtained from Kiwi), and the SIP Proxy Server stores the current location of the user agent into its database.

After users register at the SIP Proxy Server, a caller can simply send a request to the SIP Proxy Server which would find out the IP address of the called party. Figure 18 shows the flow that UA1 makes a call to UA2. The procedure is described as follows.

Step B.1 UA1 and UA2 both registered at the SIP Proxy Server. UA1 sends an INVITE request to the SIP server, and then the SIP Proxy Server would look up the location of UA2 from the location database and send this request to UA2. Because the message sent from UA1 to UA2 contains the location of UA1, and later the response from UA2 to UA1 contains the location of UA2, this process allows UA1 and UA2 to know the contact address of each other. After that, they can communicate with each other directly without the intervening of the SIP Proxy Server.

Step B.2 UA1 and UA2 can communicate with each other by sending and receiving RTP packets directly.

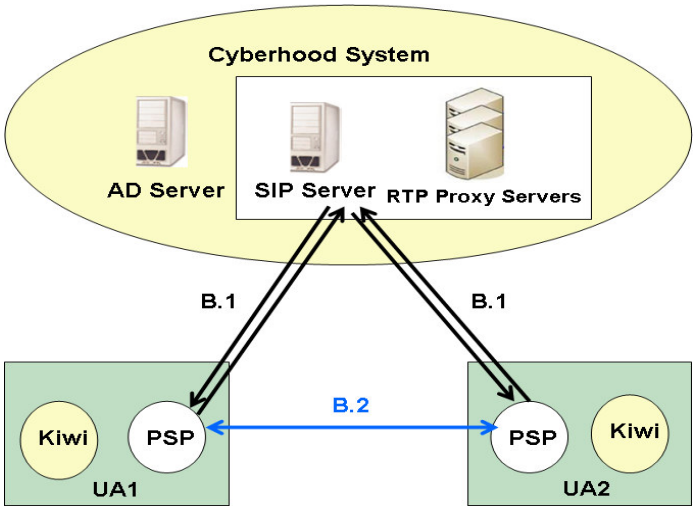

Figure 18. The flow of an INVITE request.

Chapter 3. Survey of VoIP Clients

Today there are many VoIP software programs developed on the Internet, like Skype, MSN, etc. To support the VoIP function in the CRM system, we surveyed a few popular software programs and compare their features.

3.1. Microsoft Windows Messenger

Microsoft Windows Messenger 5.1 provides the ability to communicate with all of the Microsoft's Instant Messaging (IM) services. It allows users to make calls using either the Session Initiation Protocol (SIP) or its proprietary MSN protocol.

3.2. Skype

The graphical user interface of Skype is shown in Figure 19**.**

Figure 19. The interface of Skype

Skype is a peer-to-peer VoIP client developed by Kazaa. Skype employs an overlay peer-to-peer network shown in Figure 20. Everyone who uses Skype application has to authenticate itself to the login server. There are two types of nodes in the network - ordinary nodes and super nodes. Any node who has a public IP address, sufficient CPU capacity, memory storage, and network bandwidth could be an ideal candidate to become a super node. Each ordinary nodes needs to connect to a super node to get information. Skype allows ordinary nodes under NAT to transmit and receive audio streams through the super nodes (which always have public IP addresses).

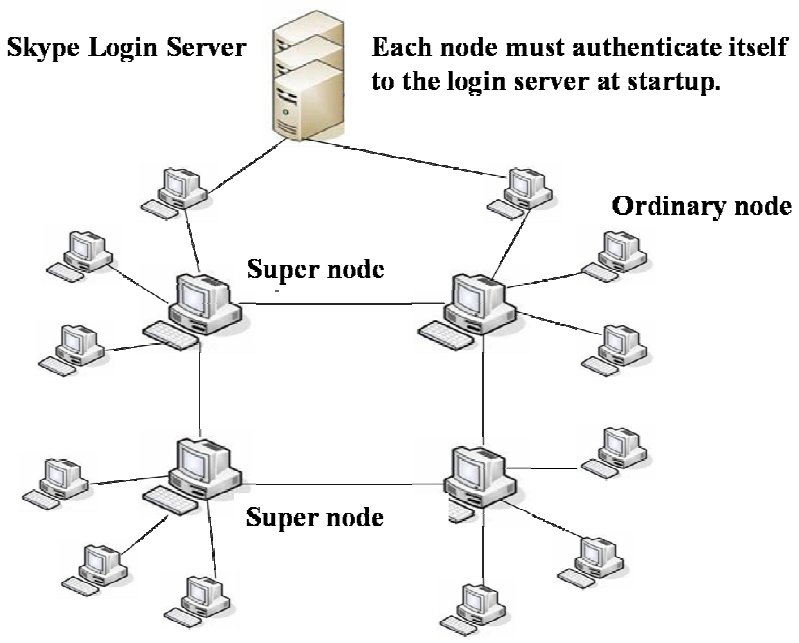

Figure 20. The overlay peer-to-peer network.

In Windows operating system, Skype can be run from a USB flash drive without being installed on the target computer. It is portable.

3.3. X-Lite

X-Lite is a free SIP-based softphone developed by CounterPath Solutions Inc [22]. It supports SIP-based signaling for all interactive media sessions, enhanced Quality of Service (QoS) for voice and video calls, IM and presence management, comprehensive personal address book including detailed call lists and history, multi-party and ad hoc voice and video conferencing, detachable and sliding drawers for quick video and contact information, and toast pop-ups allowing the management of incoming calls. The graphical user interface of X-Lite is shown in Figure 21. X-Lite can also be started (or triggered to make a call) by command lines. For example, if we want to start X-Lite, we can type the command, "`C:\Program Files\CounterPath\x-lite\x-lite.exe`" (default path).

Figure 21. The interface of X-Lite

3.4. P2P VoIP

3.4.1. The function of P2P VoIP

P2P VoIP beta 1.1 [18] is a portable P2P Internet phone. The interface and function of P2P VoIP are shown in Figure 22. The PSP software which we created is based on P2P VoIP.

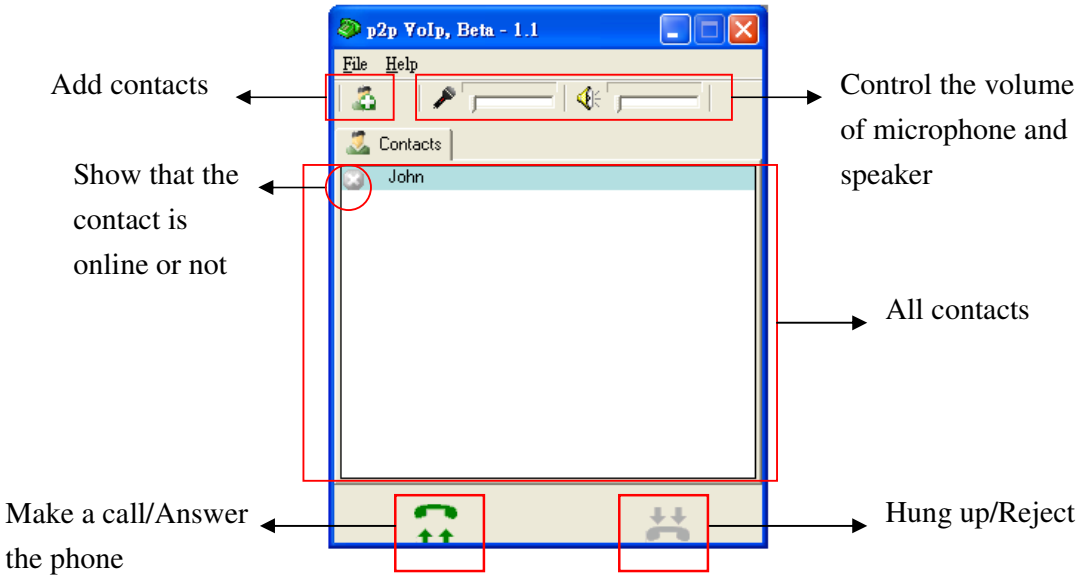

Figure 22. The interface and function of P2P VoIP beta 1.1.

This software requires each user to be identified by an IP address. It has its proprietary signaling protocols, uses GSM 6.10 codec, and transmits signaling and voice data at the same UDP port 4444. The software program inspects the length of an incoming packet to distinguish signaling packets from voice data. The components of P2P VoIP are shown in Figure 23.

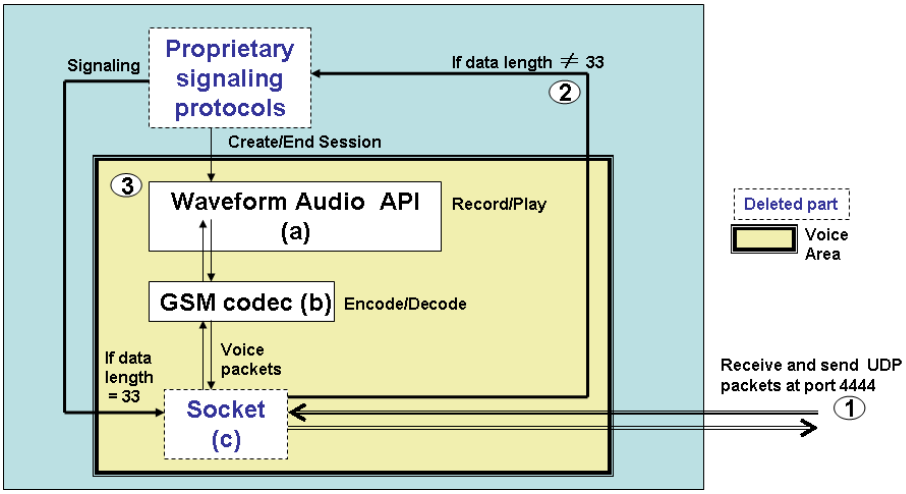

Figure 23. The architecture of P2P VoIP.

3.4.2. The proprietary signaling protocols of P2P VoIP

The author of P2P VoIP defined its own proprietary protocols in Figure 24. P2P VoIP uses the proprietary protocols to make a call, log out, etc. All of the signaling packets are 4 bytes, except the first packet of an INVITE request which we would discuss later.

```
enum _PROTOCOL
{
    _NULL_ = 0,_PROGRAMM_ABORT,
    _CONTACT_REFRESH,_CONTACT_ERASE,
    _CONTACT_ADD,_CONTACT_PING,
    _CONTACT_PING_IN,_CONTACT_PING_OUT,
    _CONTACT_PONG,_CONTACT_PONG_IN,
    _CONTACT_PONG_OUT,
    _CONTACT_OFFLINE,_CONTACT_OFFLINE_IN,
    _CONTACT_OFFLINE_OUT,_CLIENT_SHEET_SHOW,
    _CLIENT_SHEET_HIDE,_CLIENT_SHEET_NAME,
    _CLIENT_SHEET_IMAGEINDEX,_CLIENT_CONNECT_DURATION,
    _CLIENT_PROTOCOL,_TIME_OUT,
    CALL_IN_,CALL_OUT_,_I_,_I_180_,_I_200_,
    _I_480_,_A_,_G_,_G_200_,_G_486_,
    _C_,_C_200_,_C_480_,_LA_,_LG_,
    _ERROR_PLAY,
};
```

Figure 24. The proprietary protocols of P2P VoIP

3.4.2.1. Ping

When a user logs in, he will try to notify his contacts and check whether his contacts are online or not. P2P VoIP would send a CONTACT_PING signal to each contact on the list. If the contact is online, a_CONTACT_PONG signal will be sent back to the user shown in Figure 25. If the contact is offline, the user will resend the CONTACT_PING signal every minute shown in Figure 26. The texts upon the arrows are the signaling messages which are sent. The texts on the left or right indicate the

status of the users. The software program will act depend on its own status and the signal which it received.

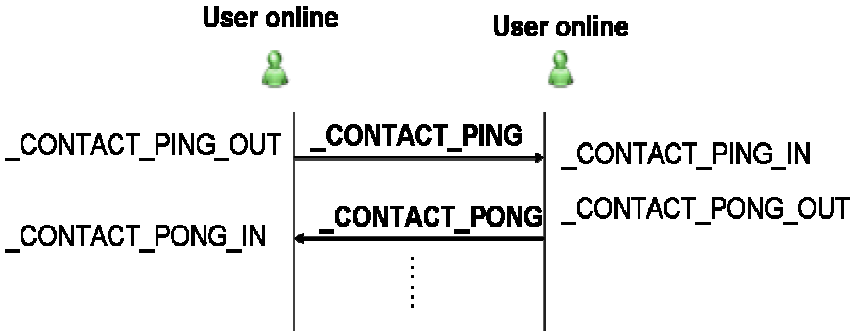

Figure 25. The flow of PING when the contact is online

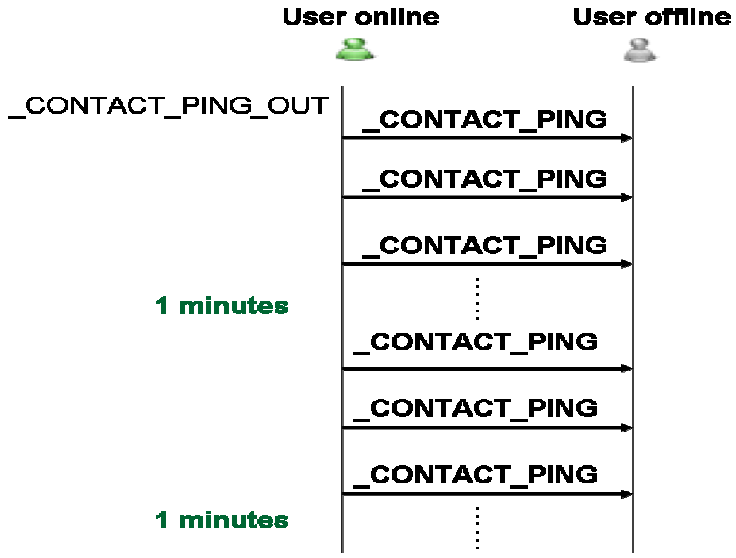

Figure 26. The flow of PING when the contact is offline

3.4.2.2. Call

The flow of INVITE request is shown in Figure 27. The proprietary signaling protocol deployed by P2P VoIP is similar to SIP. The SIP flow of an INVITE request is `INVITE`, `180 Ringing`, and `200 OK`. The flow of an INVITE request in P2P VoIP is `_I_`, `_I_180_`, and `_I_200_`.

The first packet of an INVITE request contains the 4-byte signaling data which contains one-byte signaling data and three-byte null characters, plus the display name of the user (whose length is variable) and one-byte null character. For example, if the display name of the user is ”MICKY”, then the contents of packet would be “17 00 00 00 4D 49 43 4B 59 00”. In order to guarantee that only the length of voice data would be 33 bytes, it limits the display name to be shorter than 28 letters.

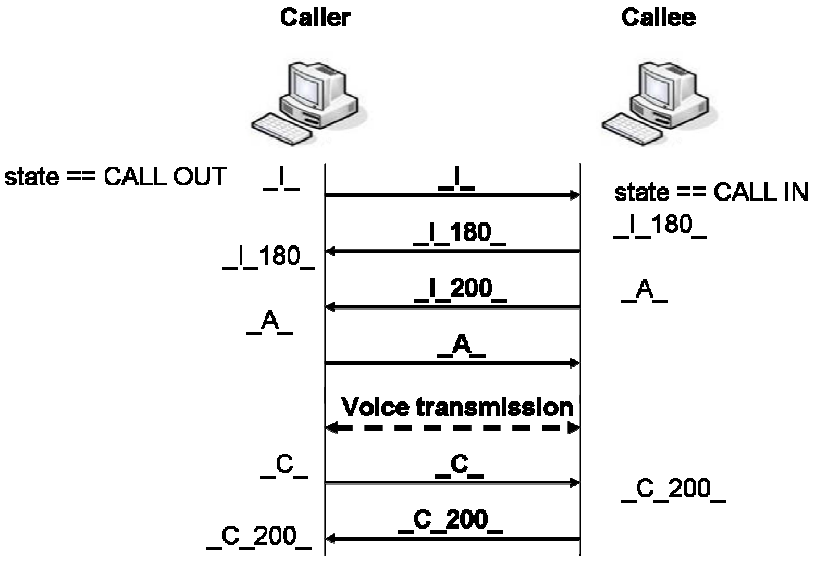

Figure 27. The flow of an INVITE request.

3.4.2.3. Cancel

If the called party does not want to answer the phone, he may reject it with a _C_ signal. The flow is shown in Figure 28.

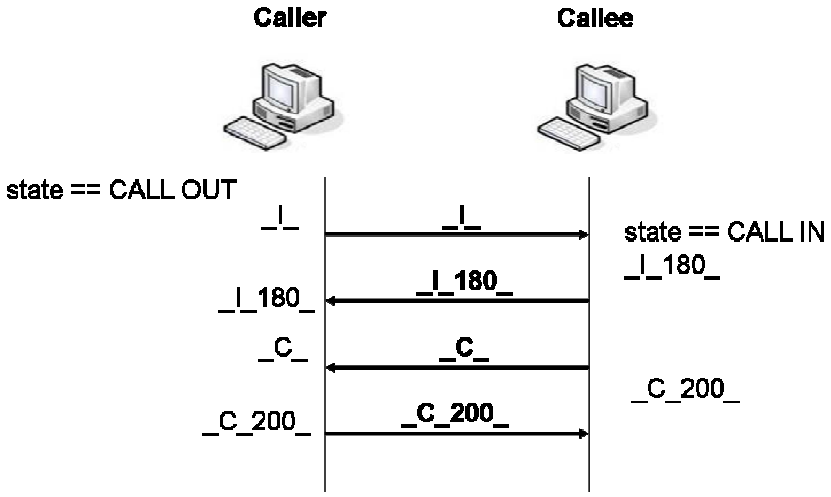

Figure 28. The flow of the Cancel operation.

3.4.2.4. No answer

If Callee does not answer the phone, P2P VoIP sends the _LA_ signal every 6 seconds to check whether the called party is online. The flow is shown in Figure 29.

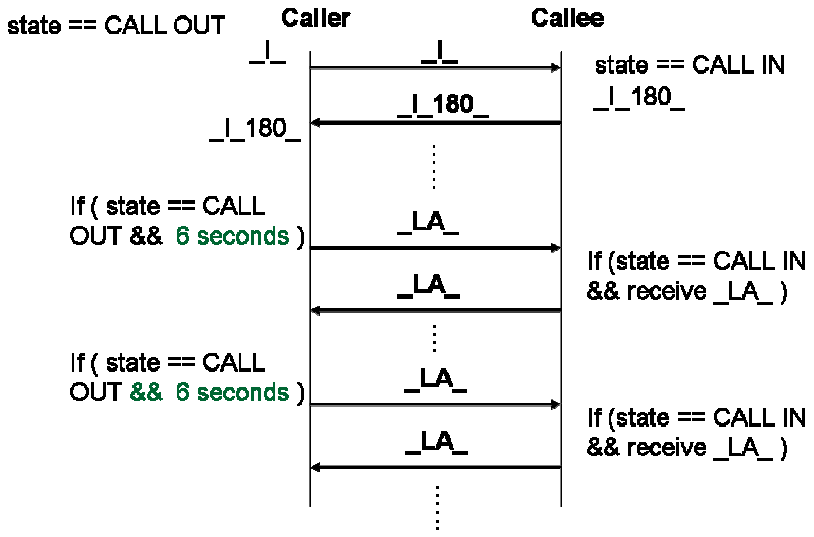

Figure 29. The flow of an INVITE request while no answer.

3.4.2.5. Logout

If a user wants to log out, P2P VoIP sends a signal to every contact and let them know that he is going to be offline. The flow is shown in Figure 30.

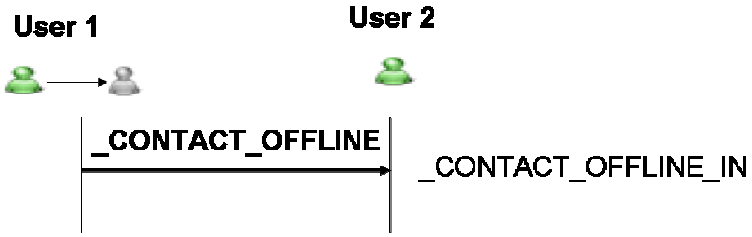

Figure 30. The flow of the logout operation

3.5. Disadvantages of existing VoIP clients

In this session, we describe the disadvantages of existing VoIP clients and

explain why they are inappropriate for our SIP collaboration design.

3.5.1. The disadvantages of Windows Messenger

The disadvantage of Windows Messenger 5.1 is that it is not portable, because it must be installed in the computer before execution.

3.5.2. The disadvantages of Skype

The disadvantages of Skype are listed as follows.

- Use proprietary signaling protocol.

  Skype did not open either its source code or its protocol. It can only communicate with itself. Therefore, it would be difficult to integrate Skype with existing CRM systems.

- Super nodes

  Users who have a powerful CPU and lots of bandwidth would be chosen to be the super nodes. Super nodes have to sacrifice part of their bandwidth for other ordinary nodes, sometimes without the cognizant agreement of the super nodes. Many organizations would prohibit the usage of Skype for this reason.

3.5.3. The disadvantages of X-Lite

Although X-Lite is a powerful IP telephony client based on open standards (SIP and SDP), it is not portable. The biggest problem is that it stores the SIP account of the user and the IP address of the SIP Proxy Server in the local hard disk. The

interface to input the registration information is shown in Figure 31. X-Lite does not support users to input the information of registration by the system command. Therefore, X-Lite does not meet our requirements.

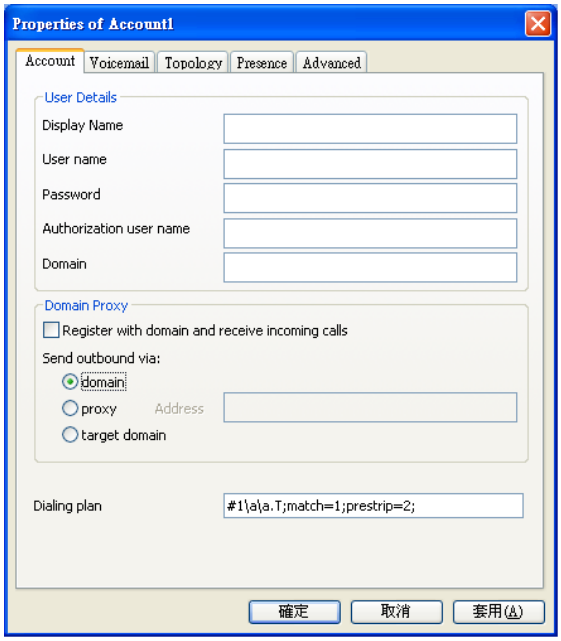

Figure 31. The registration information needs to be filled.

3.5.4. The disadvantages of P2P VoIP

The major advantage of P2P VoIP is its portability. However, it has a few disadvantages listed as follows.

- It cannot handle NAT.

  It is a P2P phone, and if one user is behind NAT, the other user can not send packets to him. It can only be used when both users are in the same subnet behind the same NAT or at least one of them has a public IP address.

- It uses its proprietary signaling protocol.

  P2P VoIP only communicates with itself but not other VoIP software/hardware.

- It uses only one transport port.

  Receiving all packets by the same port is not logically a good approach. Multiplexing must be conducted to separate audio streams from call setup signals.

After surveying the above existing solutions, we modified the P2P VoIP source code and created new VoIP software which we call PSP (Portable SIP Phone), as described in the next section.

Chapter 4. The Design of UA

4.1. Portable SIP Phone

The graphical user interface of Portable SIP Phone is shown in Figure 32. In this section, we will describe some important features of PSP and how to implement it.

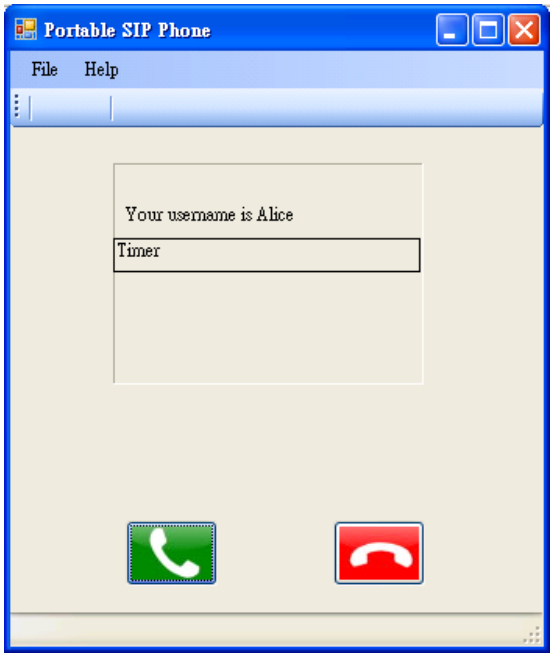

Figure 32. The interface of Portable SIP Phone.

- ***Portable*:** There are lots of software programs supporting SIP-based VoIP, like Windows Messenger, X-Lite, etc. However, they are not convenient for

nomadic users if users are not using dedicated computers. These software programs need to be installed and use the Windows registry to store configuration information. Let us consider a scenario that a user is using a public computer which disallows people to install software arbitrarily. It would be impossible to get Windows Messenger or X-Lite running on such a computer without the extra support of the system administrator. Additionally, it is not safe to store data (e.g. username, password, contact list) in a public computer, because it is easy to be stolen or misapplied. To overcome this problem, PSP is designed with the philosophy of being "portable". People can simply store PSP into a USB flash drive and connect the drive to a computer. Without any installation process, PSP can start running and supporting VoIP service right away. Moreover, people can store their own data and records in their own USB drive. Thus, users can run PSP on any computer with his/her USB drive and prevent their data from misuse.

- ***NAT Handling*:** Generally, two parties transmit RTP packets directly to communicate with each other. However, there is a special case that when users are behind NAT, they cannot transmit voice packets directly without proper assistance. As shown in Figure 33, we use a SIP Proxy Server and RTP Proxy Servers to handle it. We describe the procedure of traversing NAT as follows.
  - **Step C.1** First, the SIP Proxy Server would check whether users are behind NAT or not when they register. If they are, the SIP Proxy Server would mark it with a flag. When users behind NAT try to establish sessions with others, the SIP Proxy Server would modify the connection information in the SDP message which describes the media

information. The SIP Proxy Server replaces the IP address of the UA with the IP address of RTP Proxy Server.

- **Step C.2** Two parties send RTP packets to the RTP Proxy Server whose IP address is public. They transmit RTP packets through the RTP Proxy Server. Through the relay of RTP Proxy Server, users can communicate with each other easily, even though their IP addresses are private.

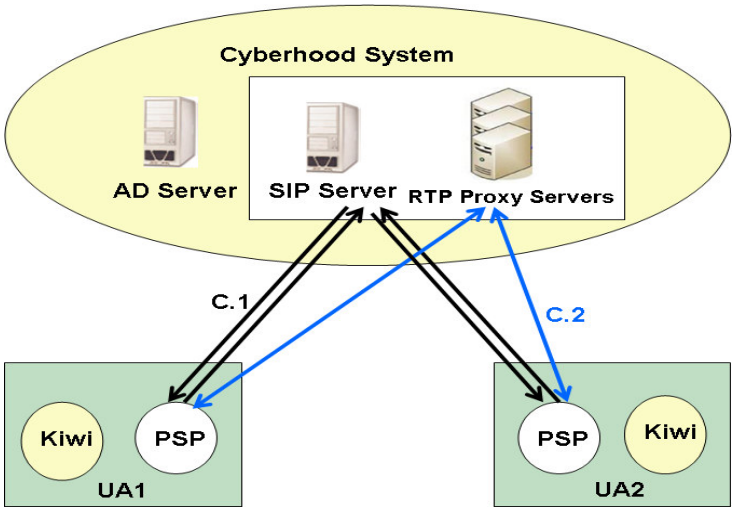

Figure 33. The flow of NAT handling.

4.2. Software components

In order to overcome the disadvantages of P2P VoIP, its proprietary signaling protocol is replaced with SIP, and voice data will be transported by RTP, to make it compatible with commercial SIP phones. The architecture of the new SIP-based VoIP is shown in Figure 35. We use the open library eXosip2 and oRTP to implement SIP and RTP, respectively.

eXosip2 [14] is a GPL library based on the GNU oSIP [15] library which is an

implementation complying with SIP. eXosip2 implements a higher layer API to control the SIP protocol stack for session establishment in an easier way. Figure 34 shows how eXosip2 supports Proxy Authentication in a SIP flow of INVITE request as defined in RFC 3261 [5].

oRTP [17] is a library implementing the Real-time Transport Protocol (RTP). It provides a packet scheduler for sending and receiving packets on time, and adaptive jitter compensation, etc. We adopt this library to encapsulate voice data into RTP packets by adding RTP headers

.

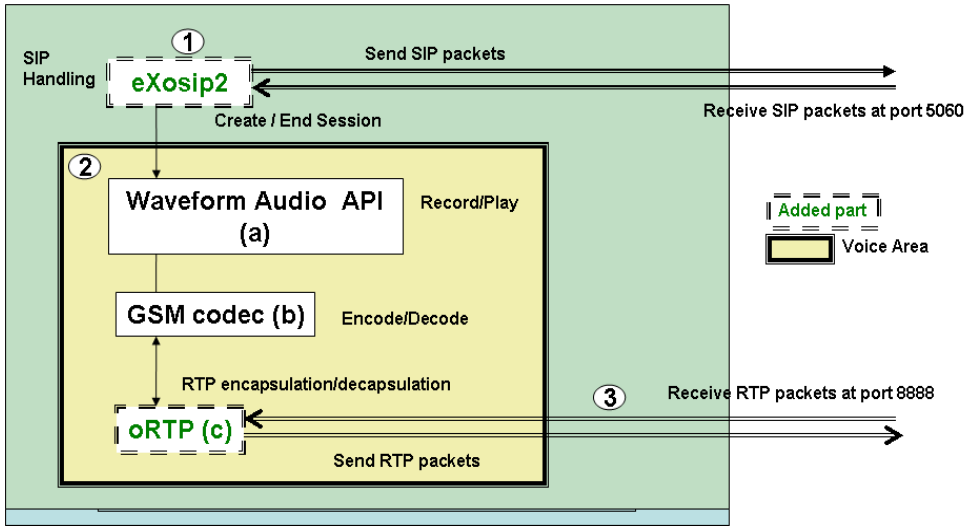

Figure 35. The architecture of PSP.

In order to overcome the drawback that P2P VoIP uses the same port to receive all packets, we separate signaling packets from voice data. The SIP signaling packets are received at port 5060 and RTP packets are received at port 8888.

We reuse the GSM RPE/LTP speech compression component in P2P VoIP. GSM 06.10 compresses frames of 160 13-bit samples into 260 bits. Our implementation

turns frames of 160 16-bit linear samples into 33-byte frames. The quality of the algorithm is good enough for audio conversation. The length of every RTP packets is 45 bytes, including the header of RTP (12 bytes) and the payload (33 bytes).

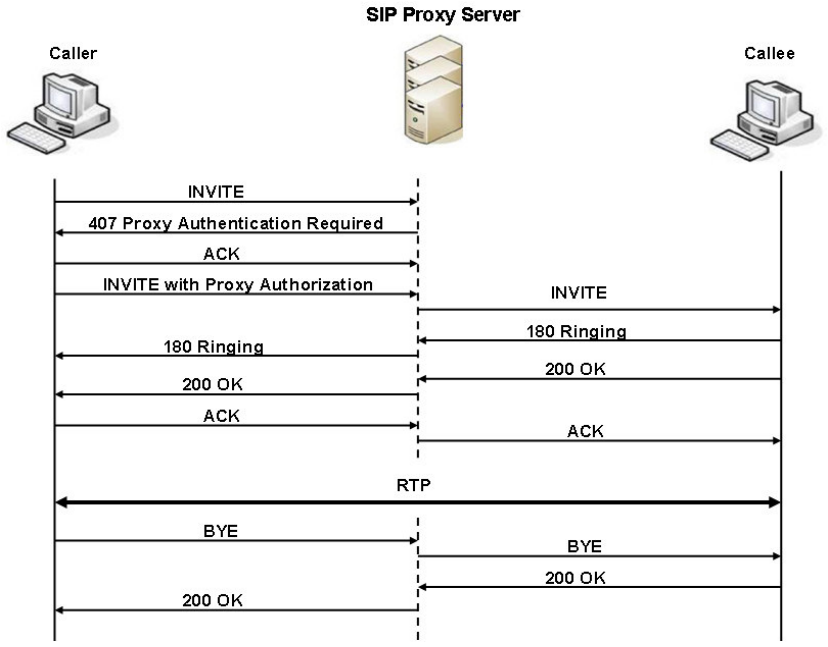

Figure 36. The flow of an INVITE request with authentication required.

Chapter 5. Implementation

The software applications that we use to implement this system are listed as follows.

5.1. Software platform

5.1.1. SIP Proxy Server

The SIP Proxy Server is a personal computer running FreeBSD 5.4, with some server applications installed. The database server is MySQL v 5.0.2 [16] used to store the location of users, and SIP Express Router (SER) v 0.9.6 [21] is the SIP Server handling SIP messages. There are also some RTP Proxy Servers which install rtpproxy v 0.3 [20]. Both SIP and RTP servers need public IP addresses in this scenario.

5.1.2. UA

PSP is developed by Microsoft Visual Studio .NET 2005. The UA has been programmed using Visual C++ supporting Common Language Runtime (CLR). It is linked with libraries eXosip2 2.2.3 and oRTP 0.13.1.

5.2. Threads

We use three threads in the design of PSP. We can see the flow in Figure 37. First, the program calls Mythread thread which handles all SIP signaling. If the

session is created, Mythread calls RTPsend thread which controls the audio input devices, encodes the audio data, and sends the RTP packets via the network interface. After RTPsend is called, it calls RTPrecv thread which receives the RTP packets, decodes the audio data, and plays it out via the audio output devices.

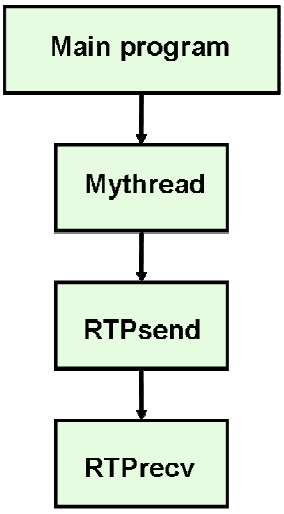

Figure 37. The threads.

5.3. Buffers

Both RTPsend and RTPrecv have a buffer which makes sure that the packets are not lost. Each buffer has two indexes which are writing and reading indexes. The writing index of the sending buffer in RTPsend thread increases when 320-byte data are written in and the reading index increases when the 33-byte encoded data are sent to the other party; the writing index of the receiving buffer in RTPrecv thread increases when an RTP packet is received and the reading index increases when the decoded data are sent to the buffer of the audio output device.

5.4. Interval between voice packets

P2P VoIP sends four voice packets in 20 ms, and sends next four voice packets after 80ms. On the contrary, RTP uses timestamps to control sending and receiving packets, so we send and receive the RTP packets every 20 ms in PSP. We can see the packet flow of P2P and PSP in Figure 38. In P2P, after one party sends four packets, the other party sends four packets. In PSP, both two parties send RTP packets in every 20ms, so the packets sent by both parties are interlaced.

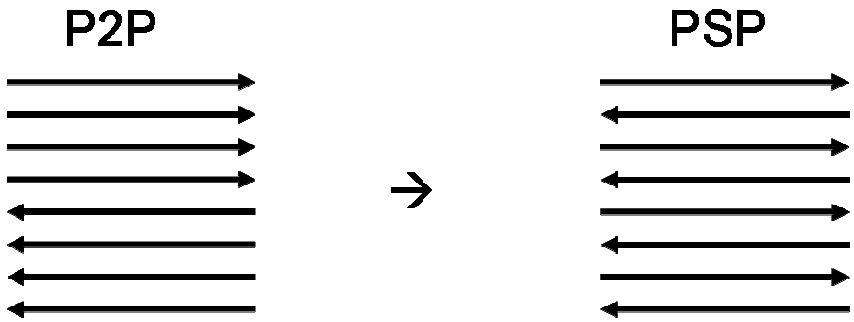

Figure 38. The packet delivery of P2P vs. PSP.

Chapter 6. Future Work & Conclusion

In this thesis, we presented a VoIP software design which is portable and can traverse the NAT successfully. We separate SIP and RTP traffic and receive them from different ports. Logically, it is a good approach, but using only one port is not really a disadvantage in the aspect of programming, because using one port only needs a thread to bind the port. On the contrary, using two ports requires two threads to receive the two types of packets (SIP and RTP), so more programming efforts are required.

This VoIP software application which we designed and implemented is based on SIP and RTP and can communicate with other commercial VoIP software applications which are also based on SIP and RTP, such as X-Lite softphone and Cisco hardphone.

So far, we deployed an RTP Proxy Server and a SIP Proxy Server on the same host to minimize the deployment cost. To get better performance, many hosts may be deployed to run RTP Proxy servers. We need a policy to choose an RTP Proxy Server dynamically to relay RTP streams for users. However, how to implement this idea and how to get good performance needs to be further studied.

Reference

[1] Daniel Collins, Carrier Grade Voice over IP, 2nd Ed., McGraw-Hill, September 2002.

[2] Robert J. Kearney, "Enhancing and Extending ERP Performance with an Automated Workflow System". In Layna Fischer (Ed.), *Workflow Handbook 2005*, pp.91-102, Future Strategies, April 2005.

[3] ChingChen Chang and Quincy Wu, "Design and Architecture of a Portable User Agent in SIP Collaboration Systems", in Proc. 3rd International Conference on Internet and Web Applications and Services (ICIW 2008), pp.273-278, Athens, Greece, June 8-13, 2008.

[4] H. Schulzrinne, S. Casner, R. Frederick, V. Jacobson, "RTP: A Transport Protocol for Real-Time Applications," IETF RFC 3550, July 2003.

[5] J. Rosenberg, J. Peterson, H. Schulzrinne, A. Johnston, J. Peterson, R. Sparks, M. Handley, E. Schooler, "SIP : Session Initiation Protocol," IETF RFC 3261, June 2002.

[6] J. Rosenberg, J. Peterson ,H. Schulzrinne, G. Camarillo, "Best Current Practices for Third Party Call Control (3PCC) in the Session Initiation Protocol (SIP)," IETF RFC 3725, April 2004.

[7] K. Egevang, "The IP Network Address Translator (NAT)," IETF RFC 1631, May 1994.

[8] M. Handley, V. Jacobson, C. Perkins, "SDP: Session Description Protocol," IETF RFC 4566, July 2006.

[9] P. Koski, J. Ylinen, P. Loula, "The SIP-Based System Used in Connection with a Firewall", IEEE Proceedings of the Advanced International Conference on Telecommunications and International Conference on Internet and Web Applications and Services (AICT-ICIW 2006), February 19-25, 2006.

[10] W. Fan, R. Luck, K. Manier, J. Pierce, L. Pool, S. D. Patek, “Customer Relationship Management for a Small Professional Technical Services Corporation”, IEEE Proceedings of 2004 Systems and Information Engineering Design Symposium, pp.243-248, April 16, 2004.

[11] Y. Rekhter, B. Moskowitz, D. Karrenberg, G. J. de Groot, E. Lear, “Address Allocation for Private Internets,” IETF RFC 1918, February 1996.

[12] Y.S. Moon, C.C. Leung, K.N. Yuen, H.C. Ho, X. Yu, “A CRM model based on voice over IP”, Proceedings of IEEE 2000 Canadian Conference on Electrical and Computer Engineering, Vol. 1, pp. 464-468 , March 7-10, 2000.

[13] Cyberhood [http://www.kinghood.com/]

[14] LibeXosip2 [http://sip.antisip.com/docu/eXosip2/]

[15] Libosip [http://sip.antisip.com/docu/osip2/]

[16] MySQL [http://www.mysql.org/]

[17] oRTP [http://freshmeat.net/projects/ortp/]

[18] PhoneSnd [http://phonesnd.com/]

[19] PSP [http://ms11.voip.edu.tw/~beautidays/PSP.html]

[20] Rtpproxy [http://ftp.iptel.org/pub/rtpproxy/]

[21] SER [http://iptel.org/ser]

[22] Counterpath [http://www.counterpath.com/]

[23] Skype [http://skype.com/]

Appendix

A. Install SER (SIP Express Router)

1. Environment

In this appendix, we shall only describe the installation steps for FreeBSD. If you are interested in the procedures of other operating systems, please refer to its web site (http://iptel.org/ser/).

2. Installation

FreeBSD supports installing SER by either “Packages” or “Ports”. To install packages, users can type the command “pkg_add ser-0.9.6_6.tbz”. To install ports, users can input the command “make install“ under the **/usr/ports/net/ser** directory.

#make install

This command will instruct the FreeBSD system to get the SER software and install it as specified in “Makefile” and “distinfo”.

After that, we have to create a database of SER which stores the information of users including the account, the location, etc. Suppose your MySQL server is installed and running, this can be done by the following command.

 # /usr/local/sbin/ser_mysql.sh create

The system will prompt you to type the MySQL password for root (please note that this is not the login password of root). A database ser will be created to store SER information. You may run the MySQL client “mysql” to login and run the command “SHOW DATABASES;” to verify that this database has been created successfully..

3. Using SER

After SER is installed, there is a sample configuration file **/usr/local/etc/ser/ser.cfg.sample**. You may copy this file to **/usr/local/etc/ser/ser.cfg** to start running SER. This default configuration supports some basic functions, such as registration and call setup between registered users. Before we start running SER, we have to set the environment variable, SIP_DOMAIN. It can be the IP address or the domain name of the SIP Proxy Server. For example:

#setenv SIP_DOMAIN sip.iptel.org (for C shell or tcsh users)

#export SIP_DOMAIN=sip.iptel.org (for Bourne shell or bash users)

Now you may start SER:

#serctl start

Or restart SER

#serctl restart

When you want to stop SER, you may type the following command

#serctl stop

4. The configuration file of SER

   In FreeBSD, the configuration file of SER is /usr/local/etc/ser/ser.cfg. A complete configuration file of SER may contain seven sections. However, we can also have a simple one with only four sections that cover the functions of a basic SIP Proxy Server. These are Global Definitions Section, Modules Section, Module Configuration Section, and Route Block。

   Global Definitions Section

   This block usually contains information like the debug level and the IP address and port number of the SIP server to receive requests. Settings in this section affect the SER daemon.

   Modules Section

There are lots of external files under the /usr/local/lib/ser/modules/ directory. The suffix ".so" indicates that they are shared objects (shared libraries). Those files are the modules of SER which supports functions not provided by the SER core. Every module has its own functionality. If we want to use a module, we have to load the module with the `loadmodule` command in this section. For example, if we want to load the registrar module which handles registration, we must modify the configuration file by specifying the following command:

#loadmodule "/usr/local/lib/ser/modules/registrar.so"

Module Configuration Section

Many of the external libraries specified in the Modules Section need to have parameters supplied properly. The parameters are specified by the `modparam` command in this section. For example, there is a parameter which controls the default expiration interval in the registrar module. We can set its value to 3600 by

 # modparam("registrar", "default_expires", 3600)

Route Block

Route Block is the main section to write the processing logic of SER. It is divided to main route block, secondary route blocks, reply route block and failure route block. Main route block is the entry point of processing a SIP message and controls how to handle each received message. Second route blocks are analogous to subroutines that would be called from the main route block. Optional reply route block handles the replies of SIP messages, and most of them are successful messages. Failure route block handles the failure condition, such as that the user is busy or a request is timeout.

After the configuration file of SER is prepared, we can start the SER

program. If there are syntax errors, SER will show which line has an error and what kind of error it is.

5. How to add users manually

We use the SER control console to add new users.

Add new users

#serctl add <username> <password> <email>

Ex. #serctl ul show Jane 0915 jane0915@yahoo.com.tw

You will be prompted to input the password of SER in MySQL (the default password is “heslo”).

Some useful commands are also described below.

View all the online users

#serctl ul show

Because downloading all users may require a little time, this command will prompt you to confirm whether to view the information of all users. If we want to view the online information of only one user, we should input the following command.

#serctl ul show <username>

Ex. #serctl ul show Jane

Change the password

#serctl passwd <username> <newpassword>

Ex. #serctl passwd Jane jane0915

You will be prompted to input the password of SER in MySQL.

Delete users

#serctl rm <usrename>

Ex. #serctl rm Jane

You will be prompted to input the password of SER in MySQL to delete the user.

There is another way to show the user accounts in the database, login into the MySQL. The prompt would become mysql>. Choose the `ser` database. This database consists of many tables. One of the tables is `subscriber` and it contains the information of subscribers, such as the username, password, the time when the account is created, etc.

B. Install rtpproxy

Installing rtpproxy is simple. Because it is not a part of SER, we must download and install it independently. The latest version of this package can be found at http://ftp.iptel.org/pub/rtpproxy/. It contains the README and INSTALL documents. First, we connect to ftp.iptel.org

#ftp ftp.iptel.org

Log in as anonymous, and enter your own e-mail address as the password. After that, change to the /pub/rtpproxy/ directory, and download the files. For example, if we want to download rtpproxy v0.3, we issue the following command:

ftp> get rtpptoxy-0.3.tar.gz

Then we log out after downloading the file.

ftp> bye

From the file suffix ".tar.gz" you can see that this is a compressed file. We need to decompress it first.

#tar zxf rtpptoxy-0.3.tar.gz

A directory `rtpproxy-0.3/` will be created. After entering it, input the commands as follows in order.

#cd rtpproxy-0.3

#./configure

#make

#make install

We do not need extra parameters to start rtpproxy. Just type the rtpproxy command as follows.

#rtpproxy

If we want to make sure whether rtpproxy is running, we can type the following command to verify.

#ps aux | grep rtpproxy

C. How to set up the SIP Proxy Server support NAT handling

In this section, we introduce how to modify the configuration file and let the SIP Proxy Server handle NAT. A sample configuration file is shown below.

```
# ----------- global configuration parameters ------------------------

debug=3              # debug level (cmd line: -dddddddddd)
fork=yes
log_stderror=no    # (cmd line: -E)

listen=163.22.16.35
check_via=no       # (cmd. line: -v)
dns=no              # (cmd. line: -r)
rev_dns=no         # (cmd. line: -R)
port=5060
children=4
fifo="/tmp/ser_fifo"

# ------------------ module loading ----------------------------------

# Uncomment this if you want to use SQL database
loadmodule "/usr/local/lib/ser/modules/mysql.so"
loadmodule "/usr/local/lib/ser/modules/sl.so"
loadmodule "/usr/local/lib/ser/modules/tm.so"
loadmodule "/usr/local/lib/ser/modules/rr.so"
loadmodule "/usr/local/lib/ser/modules/maxfwd.so"
loadmodule "/usr/local/lib/ser/modules/usrloc.so"
loadmodule "/usr/local/lib/ser/modules/registrar.so"
loadmodule "/usr/local/lib/ser/modules/textops.so"
loadmodule "/usr/local/lib/ser/modules/uri.so"
loadmodule "/usr/local/lib/ser/modules/auth.so"
loadmodule "/usr/local/lib/ser/modules/auth_db.so"
loadmodule "/usr/local/lib/ser/modules/uri_db.so"
loadmodule "/usr/local/lib/ser/modules/nathelper.so"
=> (1)

# ----------------- setting module-specific parameters ---------------
# -- usrloc params --

modparam("auth_db|uri_db|usrloc", "db_url",
"mysql://ser:heslo@localhost/ser")
modparam("auth_db", "calculate_ha1", 1)
modparam("auth_db", "password_column", "password")
modparam("rr", "enable_full_lr", 1)
modparam("nathelper","natping_interval",30)                    =>
(2)
```

```
modparam("nathelper","ping_nated_only",1)                    =>
(3)
modparam("nathelper","rtpproxy_sock","unix:/var/run/rtpproxy.sock")
=> (4)
modparam("registrar","nat_flag",6)                           =>(5)

# ------------------------   request routing logic -------------------

# main routing logic

route{

      if (!mf_process_maxfwd_header("10")) {
            sl_send_reply("483","Too Many Hops");
            break;
      };
      if (msg:len >=   2048 ) {
            sl_send_reply("513", "Message too big");
            break;
      };
      if (!method=="REGISTER") record_route();
           if(method=="BYE" || method=="CANCEL"){
               unforce_rtp_proxy();
           };                                                    => (6)
      if (loose_route()) {
            if((method=="INVITE" || method=="REFER") && !has_totag()){
                  sl_send_reply("403","Forbidden");
                  break;
            }
            if(method=="INVITE"){
                  if(!proxy_authorize("","subscriber")){
                        proxy_challenge("","0");
                        break;
                  }
                  else if(!check_from()){
                        sl_send_reply("403","Use From=ID");
                        break;
                  };
            consume_credentials();

            if(nat_uac_test("19")){
                  setflag(6);
                  force_rport();
                  fix_nated_contact();
            };
            force_rtp_proxy("1");
            };
            route(1);
            break;

      };

      if (!uri==myself) {
            route(4);
            route(1);
            break;
      };
```

```
        if (uri!=myself) {
                route(4);
                route(1);
                break;
                };
        if (method=="ACK") {
                route(1);
                break;
        };
        if (method=="CANCEL") {
                route(1);
                break;
        }
        else if (method=="INVITE") {
                route(3);
                break;
        }
        else if (method=="REGISTER") {
                route(2);
                break;
        };
                lookup("aliases");
                if (uri!=myself) {
                        route(4);
                        route(1);
                        break;
                };

                if (!lookup("location")) {
                        sl_send_reply("404", "Not Found");
                        break;
                };
                route(1);
}

route[1]
{
        t_on_reply("1");
        if (!t_relay()) {                                                       =>(7)
                if (method=="INVITE" && isflagset(6)) {
                        unforce_rtp_proxy();
                };
                sl_reply_error();
        };
}
route[2] {
        #REGISTER
        if (!search("^Contact:[ ]*\*") && nat_uac_test("19")) {                 =>(8)
                setflag(6);
                fix_nated_register();
                force_rport();                                                  =>(9)
        };
        sl_send_reply("100", "Trying");
        if (!www_authorize("","subscriber")) {
                www_challenge("","0");
                break;
        };
        if (!check_to()) {
```

```
            sl_send_reply("401", "Unauthorized");
            break;
        };
        consume_credentials();
        if (!save("location")) {
            sl_reply_error();
        };
}
route[3] {
        #INVITE
        if (!proxy_authorize("","subscriber")) {
            proxy_challenge("","0");
            break;
        }
        else if (!check_from()) {
            sl_send_reply("403", "Use From=ID");
            break;
        };
        consume_credentials();
        if (nat_uac_test("19")) {
            setflag(6);                                          =>(10)
        }
        lookup("aliases");
        if (uri!=myself) {
            route(4);
            route(1);
            break;
        };
        if (!lookup("location")) {
            sl_send_reply("404", "User Not Found");
            break;
        };
        route(4);
        route(1);
}
route[4]                                                         =>(11)
{
        if(isflagset(6)){
            force_rport();
            fix_nated_contact();
            force_rtp_proxy();
        };
}
/*      onreply_route[1]{
        if(isflagset(6) && status=-"(180)|(183)|2[0-9][0-9]"){
            if(!search("^Content-Length:[ ]*0")){
                force_rtp_proxy();
            };
        };*/

onreply_route[1]
{                                                                =>(12)
        if (isflagset(6) && status=~"(180)|(183)|2[0-9][0-9]") {
            if (!search("^Content-Length:[ ]*0")) {
                force_rtp_proxy();
            };
        };
        if (nat_uac_test("1")) {
```

```
            fix_nated_contact();
        };                                                          =>(13)
    }
```

Next, we analyze the configuration file and explain its detail.

1. First, we load the `nathelper` module. This module supports NAT handling, and has the capability to re-write the SDP message. It also helps the communication between SER and rtpproxy.
2. There are three important parameters of nathelper module. The `natping_interval` parameter controls how long SIP Proxy Server pings each user who has registered. We set the value to be 30. This will make SIP Proxy Server sending a 4-byte UDP packet to each user every 30 seconds to ensure that the NAT ports allocated to these users will be kept alive.
3. The second parameter is `ping_nated_only`. It is related to the `natping_interval` parameter. When 1 is set, SIP Proxy Server only pings users under NAT.
4. The third parameter is `rtpproxy_sock`. SER and rtpproxy communicates through a Unix socket. This parameter tells SER where the socket file is.
5. If users under NAT register at a SIP Proxy Server, we need a method to tell the `registrar` module to store the related NAT information. The method we use is to set a flag, `nat_flag`. If a user under NAT registers, SER would set a flag with value 6 and store this information to the `flags` column in the MySQL `location` table.
6. If the request sent to the SIP Proxy Server is CANCEL or BYE, the call would be terminated, and it means that we must terminate the current rtpproxy session.
7. In order to handle users under NAT in particular, we need to handle the response properly. All the responses pass the reply route block. SER allows us to define many reply_route blocks. Here, t_on_reply("1") specifies that every reply packet has to pass through the onreply_route[1] block. The t_relay() function replies a

message statefully and returns a negative value if it fails.

8. We use the `nat_uac_test()` function in `nathelper` module to test whether the user is under NAT. When we use the function, we need to supply a parameter. Different parameter value will run different test to examine whether users are under NAT or not. There are five values specified as follows.

   1: Check whether the Contact URI contains a private IP address.

   2: Check whether the address in the topmost Via header differs from the source IP address of the request.

   4: Check whether the address in the topmost Via contains a private IP address.

   8: Check whether the SDP body of the request contains a private IP address.

   16: Check whether the port in the topmost Via differs from the source port of the request.

   All of them might be bitwise combined (which is equal to the sum of the values from the list above). For example, if we want to use the check method #1, #2, and #16, we may fill the parameter as 19 because 1+2+16=19. If one of the tests matched, the function returns true.

9. When the SIP Proxy Server detects that a REGISTER request is from a user under NAT, the request would be handled as follows. First, its flag is set as 6, and two functions `fixed_nated_register()` and `force_rport()` will be executed. `fixed_nated_register()` adds the receive parameter which contains public IP address and port of NAT in the Contact header in 200 OK packet, and stores this values into the database containing the information of user location. `force_rport()` adds the received port parameter in topmost Via header.

10. If users under NAT are detected, the flag of registrar module is set as 6.

11. Route[4] triggers rtpproxy. First, it checks whether the value of current flag is 6

or not. If it is, three functions would be called. First `force_rport()` adds a parameter received port in topmost Via header. Second `fixed_nated_contact()` modifies the Contact header and replaces the IP address and port with public IP address and port which is the public IP address of NAT. The third is `force_rtp_proxy()`, which modifies the c field (connection information) in SDP. This field contains the IP address where the RTP stream is sent to; usually this will be the IP address of RTP Proxy Server.

12. `t_on_reply()` function lets all replies pass the reply route block. Next, we introduce the `onreply_route` block. It checks whether users are under NAT and whether the response code is 180, 183 and 200.
13. If Contact header contains a private IP address, SER replaces it with a public IP address and port.

D. How to use eXosip2

Before we compile eXosip2, we have to compile osip2 first. We compile osip2 and eXosip2 by Microsoft Visual .NET 2005.

1. Compile osip2

First, download the file at the following link:

http://www.antisip.com/download/libosip2-2.2.3.tar.gz

After decompressing it, it would create a `libosip2-2.2.3` directory. Enter the `\platform\vsnet\` directory. We can find a file named `osip2.vcproj`. After we compile it, a Debug DLL directory will be created. There are two files `osip2.lib` and `osipparser2.lib` in this directory. The two files are very important because they will be utilized later.

2. Compile eXosip2

First, download the file at the following link:

http://www.antisip.com/download/libeXosip2-2.2.3.tar.gz

After decompressing it, it would create a `libeXosip2-2.2.3` directory. We create a new directory named `lib` in this directory and copy the `osip2.lib` and `eXosip2.lib` in this new directory. Now we can find a file `exosip.vcproj` in the `\platform\vsnet\` directory. Before we compile this file, we have to include the lib directory first. After we compile it, the `exosip.lib` will be created.

3. Use eXosip2

Before we use the eXosip2, we need to include all the lib files. After that, we can use it.