An Analysis of the Skype Peer-to-Peer Internet Telephony Protocol

Salman A. Baset and Henning G. Schulzrinne
Department of Computer Science
Columbia University, New York NY 10027
{salman,hgs}@cs.columbia.edu

Abstract—Skype is a peer-to-peer VoIP client developed in 2003 by the organization that created Kazaa. Skype claims that it can work almost seamlessly across NATs and firewalls and has better voice quality than other VoIP clients. It encrypts calls end-to-end, and stores user information in a decentralized fashion. Skype also supports instant messaging and conferencing.
This paper analyzes key Skype functions such as login, NAT and firewall traversal, call establishment, media transfer, codecs, and conferencing under three different network setups. Analysis is performed by careful study of the Skype network traffic and by intercepting the shared library and system calls of Skype. We draw a map of super nodes to which Skype establishes a TCP connection at login.

I. INTRODUCTION

Skype [1] is a peer-to-peer (p2p) VoIP client developed by the organization that created Kazaa [2]. Skype allows its users to place voice calls and send text messages to other users of Skype clients. In essence, it is very similar to the MSN and Yahoo IM applications, as it has capabilities for voice-calls, instant messaging, audio conferencing, and buddy lists. However, the underlying protocols and techniques it employs are quite different.

Like its file sharing predecessor Kazaa, Skype uses an overlay peer-to-peer network. There are two types of nodes in this overlay network, ordinary hosts and super nodes (SN). An ordinary host is a Skype application that can be used to place voice calls and send text messages. A super node is an ordinary host's end-point on the Skype network. Any node with a public IP address having sufficient CPU, memory, and network bandwidth is a candidate to become a super node. An ordinary host must connect to a super node and must authenticate itself with the Skype login server. Although not a Skype node itself, the Skype login server is an important entity in the Skype network as user names and passwords are stored at the login server. This server ensures that Skype login names are unique across the Skype name space. Starting with Skype version 1.2, the buddy list is also stored on the login server. Figure 1 illustrates the relationship between ordinary hosts, super nodes and the login server.

Apart from the login server, there are SkypeOut [3] and SkypeIn [4] servers which provide PC-to-PSTN and PSTN-to-PC bridging. SkypeOut and SkypeIn servers do not play a role in PC-to-PC call establishment and hence we do not consider them to be a part of the Skype peer-to-peer network. Thus, we consider the login server to be the only central component in the Skype p2p network. Online and offline user information is stored and propagated in a decentralized fashion.

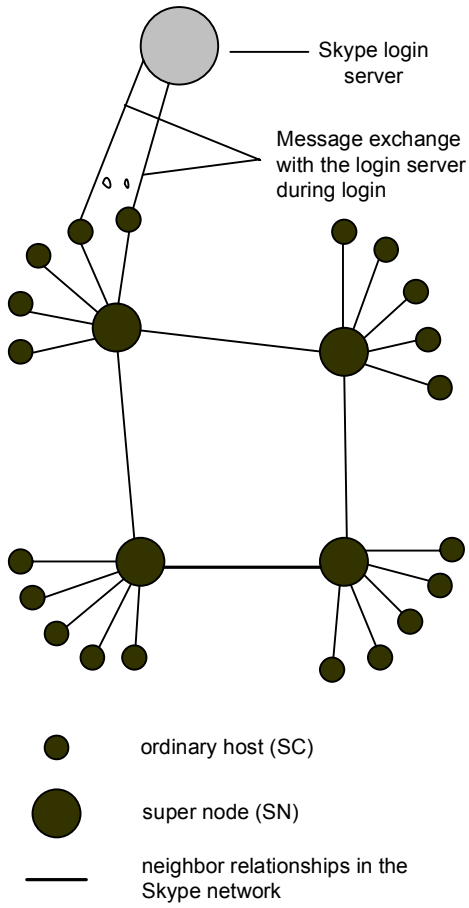

Figure 1. Skype Network. There are three main entities: supernodes, ordinary nodes, and the login server.

We believe that each Skype node uses a variant of the STUN [5] protocol to determine the type of NAT and firewall it is behind. We also believe that there is no global NAT and firewall traversal server because if there was one, the Skype node would have exchanged traffic with it during the login and call establishment phases in the many experiments we performed.

The Skype network is an overlay network and thus each Skype client (SC) needs to build and refresh a table of reachable nodes. In Skype, this table is called host cache (HC) and it contains IP address and port number of super nodes. Starting with Skype v1.0, the HC is stored in an XML file.

Skype claims to have implemented a '3G P2P' or 'Global Index' [6] technology, which is guaranteed to find a user if that user has logged in the Skype network in the last 72 hours.

Skype uses wideband codecs which allows it to maintain reasonable call quality at an available bandwidth of 32 kb/s. It uses TCP for signaling, and both UDP and TCP for transporting media traffic.

The rest of this paper is organized as follows. Section II describes key components of the Skype software and the Skype network. Section III describes the experimental setup we used for reverse-engineering the Skype protocol. Section IV discusses key Skype functions like startup, login, user search, call establishment, media transfer and codecs, and presence timers. Flow diagrams based on actual network traffic have been included to elaborate on the details. Section V discusses conferencing. Section VI discusses other experiments and compares aspects of Skype with Yahoo, MSN and Google Talk IM applications. A world map of SNs to which a SC establishes a TCP connection at login is also drawn.

II. Key Components of the Skype Software

A Skype client listens on particular ports for incoming calls, maintains a table of other Skype nodes called a host cache, uses wideband codecs, maintains a buddy list, encrypts messages end-to-end, and determines if it is behind a NAT or a firewall. This section discusses these components and functionalities in detail.

A. Ports

A Skype client (SC) opens a TCP and a UDP listening port at the port number configured in its connection dialog box. SC randomly chooses the port number upon installation. In addition, SC also opens TCP listening ports at port number 80 and 443 which, otherwise, are used to listen for incoming HTTP and HTTP-over-TLS requests. Unlike many Internet protocols like SIP [9] and HTTP [10], there is no default TCP or UDP listening port. Figure 14 shows a snapshot of the Skype (v1.4) connection dialog box. This figure shows the ports on which a SC listens for incoming connections.

B. Host Cache

The host cache (HC) is a list of super node IP address and port pairs that SC builds and refreshes regularly. It is a critical part to the Skype operation. In SC v0.97, at least one valid entry must be present in the HC. A valid entry is an IP address and port number of an online Skype node. At login time, a SC v0.97 tried to establish a TCP connection and exchange information with any HC entry. If it was unable to do so, it reported a login failure. In Skype v1.2 and onwards, if a SC is unable to establish a TCP connection with any HC entry, it tries to establish a TCP connection and exchange information with one of the seven bootstrap IP address and port pairs hard-coded in the Skype executable. A SC for Windows XP stores the host cache as a XML file 'shared.xml' in C:\Documents and Settings\<XP User>\Application Data\Skype. A SC for Linux stores the HC as a XML file 'shared.xml' at $(HOMEDIR)/.Skype. After running a SC for two days, we observed that HC contained a maximum of 200 entries. Host and peer caches are not new to Skype. Chord [20], another peer-to-peer protocol, has a finger table, which it uses to quickly find a node.

C. Codecs

During our experiments, we observed that Skype uses the iLBC [12], iSAC [13], and iPCM [14] codecs. These codecs have been developed by GlobalIPSound [15]. For SC v1.4 we measured that the Skype codecs allow frequencies between 50-8,000 Hz to pass through. This frequency range is the characteristic of a wideband codec.

D. Buddy List[1]

In Windows XP, Skype stores its buddy information in an XML file 'config.xml' at C:\Documents and Settings\<XP user>\Application Data\Skype\<skype user id>. In Linux, Skype stores the 'config.xml' file in $(HOMEDIR)/.Skype/<skype user id>. Starting with Skype v1.2 for Windows XP, the buddy list is also stored on a central Skype server whose IP address is 212.72.49.142. The buddy list is stored unencrypted on a computer. Figure 2 shows a fragment of the config.xml file.

```
<CentralStorage>
  <LastBackoff>0</LastBackoff>
  <LastFailure>0</LastFailure>
  <LastSync>1135714076</LastSync>
  <NeedSync>0</NeedSync>
  <SyncSet>
    <u>
    <skypebuddy1>2f1b8360:2</skypebuddy1>
    <skypebuddy2>d0450f12:2</skypebuddy2
```

Figure 2. A fragment of the config.xml file for a SC. It shows two Skype buddies and an four-byte number for each buddy. If two SCs have the same buddy, their corresponding config.xml files have a different four-byte number for the same buddy.

E. Encryption

The Skype website [18] explains: "Skype uses AES (Advanced Encryption Standard), also known as Rijndael, which is used by U.S. Government organizations to protect sensitive, information. Skype uses 256-bit encryption, which has a total of 1.1×10^{77} possible keys, in order to actively encrypt the data in each Skype call or instant message. Skype uses 1024 bit RSA to negotiate symmetric AES keys. User public keys are certified by the Skype server at login using 1536 or 2048-bit RSA certificates."

F. NAT and Firewall

We conjecture that SC uses a variation of the STUN [5] and TURN [19] protocols to determine the type of NAT and firewall it is behind. We also conjecture that SC refreshes this information periodically. This information is also stored in the shared.xml file.

Unlike its file sharing counter part Kazaa, a Skype client cannot prevent itself from becoming a super node.

III. Experimental Setup

Experiments were performed for the Windows Skype version 1.4.0.84 and for the Linux Skype version 1.2.0.18. We used traffic analysis, shared library and system call interception techniques to analyze various aspects of the Skype protocol. Tools like memgrp [21] can be used to perform a runtime analysis of the Skype memory. We have used this tool sparingly as it requires an extensive effort and trial and error to 'decipher' the memory dumps. Therefore, we do not present any results from using that tool. Tools by MaxMind [26] were

[1] Buddy list is an AOL trademark.

used to perform reverse country, city, and ISP lookups for an IP address when dig failed to return a DNS PTR record.

Below, we explain the experimental setup for experiments performed on different versions of the Skype client.

A. *Skype version 1.4.0.84.*

This version was available for Windows. Traffic analysis was the primary mechanism for experiments performed for this version. A SC was installed on two Windows XP machines. Each machine had a 3 GHz Pentium 4 CPU with 1 GB of RAM. Each machine had a 10/100 Mb/s Ethernet card and was connected to a 100 Mb/s network.

We performed experiments under three different network setups. In the first setup, both Skype users were on machines with public IP addresses; in the second setup, one Skype user was behind a port-restricted[1] NAT; in the third setup, both Skype users were behind a port-restricted NAT and UDP-restricted firewall. The NAT and firewall machines ran Mandriva Linux 10.2 and were connected to 100 Mb/s Ethernet network. The NAT was configured using Linux 'iptables'.

Ethereal [7] and NetPeeker [8] were used to monitor and control network traffic, respectively. NetPeeker was used to tune the bandwidth so as to analyze the Skype operation under network congestion.

B. *Skype version 1.2.0.18*

This version is available for Linux. We used shared library and system call redirection techniques to gain more insights into the Skype protocol. In Linux, at program startup, dynamic linking allows to load a shared library pointed by LD_PRELOAD environment variable before any other shared library. This makes it possible to overload a library function such as strcpy() or send(). When LD_PRELOAD is set to a library containing an overloaded strcpy() function, and the program which contains strcpy() calls is executed, the overloaded strcpy() is called. The parameters passed to this overloaded strcpy() function can be displayed or any appropriate action can be taken. Also, the overloaded strcpy() function can then call the libc strcpy() function. Austin Godber [22] provides a nice tutorial on this technique and Linux function interception.

In our experiments, we exported the display of two Linux machines using X-Win32 [23]. Thus, we were able to run different instances of a Skype client on the same host machine. However, the sound device cannot be accessed when the display is exported. To overcome this problem, we overrode the open(), close(), select(), and ioctl() calls using the technique described above. Each of these calls called the namesake libc function from within. In Skype, the socket and sound descriptors are polled by a select() system call. When Skype requests to open a sound device, our overloaded open() system call returns a fake descriptor. Skype then requests this descriptor to be polled by select(); however since this is a fake descriptor, we must not pass this descriptor to the actual select() system call. Therefore, our overloaded select() clears this fake descriptor from the read descriptor list before calling the actual select() function.

[1] A port-restricted NAT allows an external host, with source IP address X and source port P, to send a packet to the internal host only if the internal host had previously sent a packet to IP address X and port P.

An actual sound device (microphone) will have periodic data to read after it is open. However, since ours is a dummy sound device, select() will not return periodically on this device. To solve this issue, we created a select() timer in the actual select() system call with an interval of 20 ms. When the select() returns on a timer event, we add to the select() read descriptor list which is passed to the overloaded select() the fake sound device descriptor. Skype then issues a read() on this fake descriptor. Since read() is overloaded, our read() function is called. The overloaded read() then returns a dummy sound buffer to the Skype. We observed that Skype requested to read 960 bytes from the sound device on each read request.

All experiments were performed between November and December, 2005.

In the subsequent sections, any reference to function overloading in experiments implies that Linux Skype version 1.2.0.18 was used. Otherwise, Windows Skype version 1.4.0.84 was under test.

IV. SKYPE FUNCTIONS

Skype functions can be classified into startup, login, user search, call establishment and tear down, media transfer, and presence messages. This section discusses each of them in detail.

A. *Startup*

When SC v1.4 was run for the first time after installation, it sent a HTTP 1.1 GET request to the Skype server (skype.com). The first line of this request contained the keyword 'installed'.

The complete startup messages for Skype v0.97 are reported in the technical report [11].

B. *Login*

Login is perhaps the most critical function to the Skype operation. It is during this process a SC authenticates its user name and password with the login server, advertises its presence to other peers and its buddies, determines the type of NAT and firewall it is behind, discovers online Skype nodes with public IP addresses, and checks the availability of latest Skype version.

1) *Login Process*

Using the library function call overloading technique described in section III.B, we overrode the connect(), and sendto() calls such that these calls always returned with a failure. However, we permitted a TCP connection to localhost since Skype refuses to run if cannot establish this connection. The system time was printed whenever the connect() and sendto() functions were called to accurately profile the time at which Skype sends its login messages. Also, before running the Skype we deleted the HC XML file. Then we ran the SC, and made a login attempt. We observed that the SC first sent a UDP packet of length 18 bytes to each of the seven bootstrap SN IP address and port 33033. If there was no response after five seconds, SC tried to establish a TCP connection with each of these seven default SNs IP address on port 33033. If the connection attempts failed, it repeated the whole process after six seconds. We ran this experiment for 15 minutes, and strangely Skype never reported a login failure. Figure 3 shows these login attempts as a flow chart.

In the same experiment conducted in July 2005 for Skype Linux v1.0, we had observed that Skype tried to establish a

connection with each of the SN IP address on port 80 and port 443. Most firewalls are configured to allow outgoing TCP traffic to port 80 (HTTP port) and port 443 (HTTP-over-TLS port). However, we did not observe such attempts for Skype Linux v1.2.

Since the HC file had been deleted, and since we saw the same bootstrap IP address and port pairs in subsequent failed login attempts, we conclude that these IP address and port pairs are hard-coded in the Skype executable.

We have observed that a SC must establish a TCP connection with a SN in order to connect to the Skype network. If it cannot connect to a super node, it will report a login failure.

In another experiment, we filled the SC HC with an invalid IP address and port pair. Initially, SC was unable to establish a TCP connection with this invalid entry; however, after some time, it established a TCP connection with one of the bootstrap SNs. Since IP address and port number of any bootstrap SN was not present in the HC, it gives more credence to our belief that some SN IP address and port number pairs are hard-coded in the Skype executable.

In order to see the minimal set of messages a SC exchanges with other entities for a successful login, we performed the following experiment. We deleted the HC and permitted inbound and outbound UDP and TCP traffic. A SC was started and a login attempt was made. The login attempt succeeded. We then repeated this experiment for the same Skype user id two more times. Figure 4 shows the set of messages exchanged between SC, bootstrap SN, SN, and the login server in a condensed form.

In these experiments we observed that the first and the second messages exchanged with the login server were always the same across multiple login attempts even for different skype user ids. The decimal representation of message (1) is 22 3 1 0 0 and decimal representation of message (2) is 23 3 1 0 0. In most of our experiments, only four messages were exchanged between SC and the login server. The length of these messages was almost the same in subsequent experiments. Messages (3) and (4) were different for each login attempt. However, message (3) and (4) shared a four byte common header across different experiments. The decimal equivalent of first four bytes of these common headers is the same as message (1) and (2), respectively. The decimal equivalent of the fifth byte in message (3) was 205. On inspection, we found the header '23 3 1 0' at that location [header+205] and another length field after that header whose value was 198. The decimal equivalent of the fifth byte in message (4) was '217' which appears to be the length of the message.

Note that in SSL messages, the first byte indicates the message type and the next two bytes indicate the SSL version. The value 22 (0x16) corresponds to the SSL message type client_key_exchange and the value 3 0 corresponds to the SSL version 3.0. Since the messages a SC sends to the login server contains the header 22 3 1 0, it indicates that Skype is using part of SSL header for its login messages.

Using the same setup that was used for the experiment described in the above paragraph, a login attempt was made with an invalid password. The length of the messages (1), (2), and (3) exchanged with the login server remained the same. The length of the message (4) returned by the login server was 18 bytes indicating a login failure. The decimal equivalent of the fifth byte in message (4) was 13 which indicated the length of this message after a four byte header.

To see if it is possible to block Skype, we performed the following experiment. A successful login attempt was made. Then, the SC was shut down. We overrode connect() such that it returned with an error when a connection attempt was made with the login server IP addresses. SC was then started and a

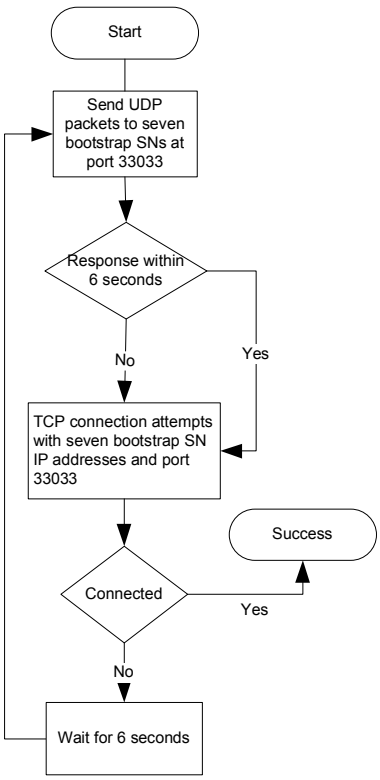

Figure 3. Skype login process. SC sends UDP packets of length 18 bytes to all bootstrap SNs. After 5s, it attempts TCP connections with the seven bootstrap SN IP address and ports 33033. Authentication with the login server is not shown.

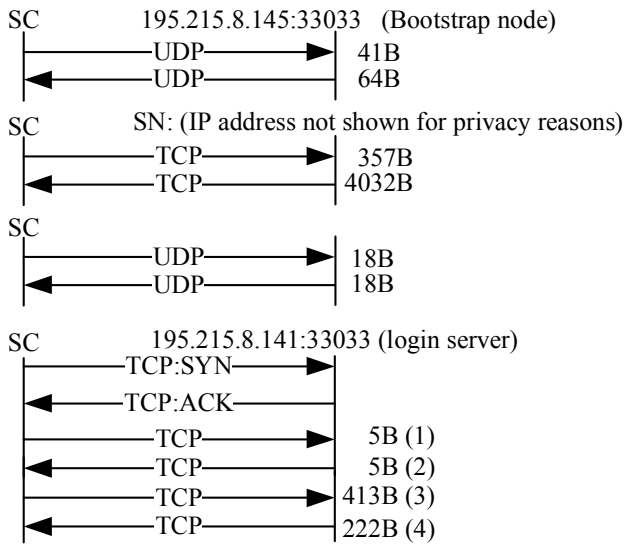

Figure 4. Minimal set of messages exchanged with the bootstrap SN, the SN, and the login server. Messages exchanged with the bootstrap SN, and SN have been aggregated. Message size for messages exchanged with the bootstrap node and SN correspond to the cumulative size. Messages sent after the exchange with the login server is completed are not shown.

TABLE I
SKYPE (VER 1.4) LOGIN EXPERIMENT SUMMARY

Skype on a Machine with/behind	Data Exchanged	Time to Login
Public IP address	10 KB	3-7 s
Port-restricted NAT	11 KB	3-7 s
UDP-restricted firewall	7 KB	35 s

login attempt was made. Strangely, the login attempt succeeded. We noted the IP address of the node to which the initial login message having decimal representation 22 3 1 0 0 was sent. In our overloaded connect(), we blocked connection attempts to this IP address. We then started Skype, and attempted a login. However, Skype was still able to login successfully. We then kept on blocking IP addresses in connect() to which login messages were sent in the previous login attempt. In all, we ended up blocking six IP addresses in connect(). However, Skype was still able to login successfully. From this experiment, we conclude that Skype routes login messages through SNs. This is a change from version 0.97 where it was possible to block Skype by simply blocking the login server IP address.

Next, we overrode the send() call such that it always returned with an error when it saw a message whose first four bytes were 22 3 1 0. Note that these are the first four bytes of message (1) and (3) shown in Figure 4. Skype was then started and a login attempt was made. Skype was unable to login despite multiple login attempts for different Skype user ids. Thus, it is possible to block Skype by dropping all the packets whose first four bytes of payload are 22 3 1 0. However, care should be taken to ensure that any such rule at the firewall does not result in blocking legitimate traffic.

For Skype v1.4, we performed experiments to understand the Skype login behavior for the three network setups described in section III.A. For these experiments, a previous copy of SC was uninstalled and Windows registry was cleared of old Skype entries. Then, a new copy of SC was installed. Table I summarizes the results of these experiments. Detailed message flows for these login attempts for v0.97 are available in the technical report [11].

In most of the login attempts, we observed that a SC sent ICMP messages to the following IP addresses: 204.152.* (USA), 130.244.* (Sweden), 202.139.* (Australia), 202.232.* (Japan). The reason for sending these messages is not clear. The reverse lookup done using MaxMind [26] suggests that each of these IP addresses is in countries located in different continents.

For the first two experimental setups, the SC sent messages to about 22 nodes and received responses from them after authenticating itself with the login server.

2) Login Server

After a SC is connected to a SN, the SC must authenticate the user name and password with the Skype login server. The login server is the only central component in the Skype p2p network. It stores Skype user names and passwords and ensures that Skype user names are unique across the Skype name space. SC must authenticate itself with the login server for a successful login. During our experiments we observed that SC always exchanged data over TCP with a node whose IP address was either 212.72.49.141 or 195.215.8.141. We believe that these nodes are the login servers. A reverse lookup of these two IP addresses did not retrieve a NS record. The first hostname returned in the authority section of the reverse lookup query (dig) was ns07.customer.eu.level3.net and ns3.DK.net respectively. Country lookup done using MaxMind tools suggests that 212.72.49.141 is in Netherlands and 195.215.8.141 is in Denmark. The buddy list is hosted on a server whose IP address is 212.72.49.142. We consider it to be a part of the login server.

TABLE II
BOOTSTRAP SN IP ADDRESS AND HOSTNAMES OBTAINED BY A REVERSE LOOKUP

IP address:port	Reverse Lookup Result	Authority Section
66.235.180.9:33033	sss1.skype.net	ns1.hopone.net
66.235.181.9:33033	No PTR result	ns1.hopone.net
212.72.49.143:33033	No PTR result	ns07.customer.eu.level3.net
195.215.8.145:33033	No PTR result	ns3.DK.net
64.246.49.60:33033	rs-64-246-49-60.ev1.net	ns2.ev1.net
64.246.49.61:33033	rs-64-246-49-61.ev1.net	ns2.ev1.net
64.246.48.23:33033	ev1s-64-246-48-23.ev1servers.net	ns1.ev1.net

3) Bootstrap Super Nodes

We list the IP address and port numbers of the seven default SNs observed during a failed login attempt. The corresponding hostnames and the first entry of the authority section returned by reverse lookup query (dig) are given in Table II.

From the reverse lookup, it appears that one SN is maintained by Skype itself.

4) NAT and Firewall Determination

We conjecture that a SC is able to determine at login if it is behind a NAT and a firewall. We guess that there are at least two ways in which a SC can determine this information. One possibility is that it can determine this information by exchanging messages with its SN using a variant of the STUN [5] protocol. The other possibility is that during login, a SC sends and possibly receives data from some nodes after it has established a TCP connection with the SN. We conjecture that at this point, SC uses its variation of STUN [5] protocol to determine the type of NAT or firewall it is behind. Once determined, the SC stores this information in the shared.xml file. We also conjecture that SC refreshes this information periodically. We are not clear on how often a SC refreshes this information since Skype messages are encrypted.

5) Skype Latest Version

During login, a SC sent a HTTP 1.1 GET request to the Skype server (skype.com) to determine if a new version was available. The first line of this request contained the keyword 'getlatestversion'. Along with the HTTP request sent at first time startup, these are the only text-based messages sent by Skype.

6) Login Process Time

We measured the time to login on the Skype network for the three different network setups described in section III. For this experiment, the HC already contained the maximum of two hundred entries. The SC with a public IP address and the SC behind a port-restricted NAT took about 3-7 seconds to complete the login procedures. The SC behind a UDP-restricted firewall took about 35 seconds to complete the login process. For SC behind a UDP-restricted firewall, we observed that it sent UDP packets to its twenty HC entries. At that point it concluded that it is behind UDP-restricted firewall. It then tried to establish a TCP connection with the HC entries and

was ultimately able to connect to a SN. Also, a SC behind a UDP-restricted firewall and port-restricted NAT took 5-10 seconds for immediate subsequent logins. This shows that a SC stores its last connectivity information in a file.

C. User Search

Skype uses its Global Index (GI) [6] technology to search for a user. Skype claims that search is distributed and is guaranteed to find a user if it exists and has logged in during the last 72 hours. Extensive testing suggests that Skype was always able to locate users who logged in using a public or private IP address in the last 72 hours.

Skype is a not an open protocol and its messages are encrypted. Whereas for login, we were able to form a reasonably precise opinion about the different entities involved, it is not possible to do so in search, since we cannot trace the Skype messages beyond a SN. Also, we were unable to force a SC to connect to a particular SN. Nevertheless, we have observed and present search message flows for the three different network setups.

A SC has a search dialog box. After entering the Skype user id and pressing the find button, SC starts its search for a particular user. For SC on a public IP address, SC sent a TCP packet to its SN. It appears that the SN gave SC the IP address and port number of eight nodes to query, since after that exchange with SN, SC sent UDP packets to eight nodes. If it could not find the user, it informed the SN over TCP. It appears that the SN now asked it to contact sixteen different nodes, since the SC then sent UDP packets to sixteen different nodes. This process continued until the SC found the user or it determined that the user did not exist. On average, SC contacted more than 24 nodes. The search took three to four seconds.

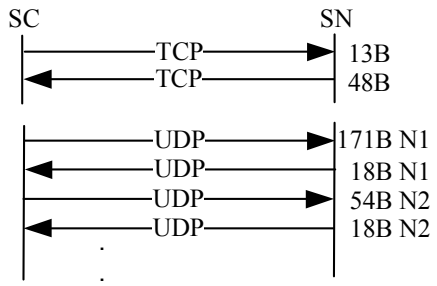

Figure 5. Message flow for a successful user search when SC v1.4 has a public IP address. 'B' stands for bytes and 'N' stands for node. Message sizes correspond to payload size of TCP or UDP packets. Not all messages are shown.

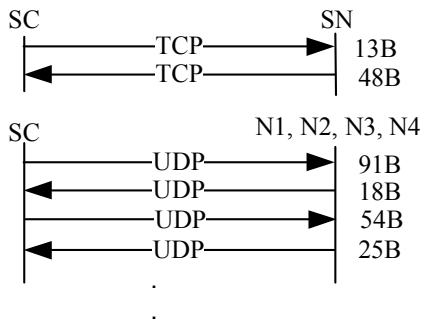

Figure 6. Message flow for a successful user search when SC v1.4 is behind a port-restricted NAT. 'B' stands for bytes and 'N' stands for node. UDP packets were sent to N1, N2, N3, and N4 during login process and responses were received from them. Message size corresponds to payload size of TCP or UDP packets. Not all messages are shown.

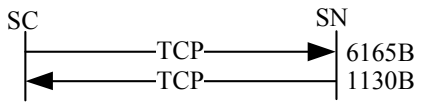

Figure 7. User search by a SC v1.4 behind a UDP-restricted firewall. 'B' stands for bytes. Messages have been aggregated for space. Data is exchanged with SN only. Message size corresponds to the approximate cumulative size of messages exchanged between SC and a SN and vice versa.

A SC behind a port-restricted NAT exchanged data between SN and some of the nodes which responded to its UDP request during login process. The message flow is shown in Figure 6. The search took about five to six seconds.

A SC behind a port-restricted NAT and UDP-restricted firewall sent the search request over TCP to its SN. We believe that SN then performed the search query and informed SC of the search results. Unlike a user search by SC on a public IP address, SC did not contact any other nodes. This suggests that SC knew that it was behind a UDP-restricted firewall. The aggregated message flow is shown Figure 7. The search took about 10-15 seconds.

In some successful searches, we saw the SC exchanging information with the login server. It appears that Skype is using the login server as a fall back option in case the search is unsuccessful. For a non-existent Skype name, a SC always contacted the login server.

We are not clear on how SC terminates the search if it is unable to find a user.

1) Search Result Caching

To observe if search results are cached at intermediate nodes, we performed the following experiment for SC v1.4. User A was behind a port-restricted NAT and UDP-restricted firewall and logged on the Skype network. User B logged in using a SC running on machine B, which was on a public IP address. User B (on a machine with a public IP address) searched for user A, who was behind a port-restricted NAT and UDP-restricted firewall. We observed that search took about 10-11 seconds. Next, SC on machine B was uninstalled, and the Skype registry cleared so as to remove any local caches. SC was reinstalled on machine B and user B searched for user A. The search took about 3-4 seconds. This experiment was repeated four times on different days and similar results were obtained.

From the above discussion we infer that the SC performs user information caching at intermediate nodes.

Skype allows the user to perform wildcard searches of different Skype user ids. To see if the same wildcard search query executed on two instances of SC retrieved the same result, we performed the following experiment. We started two instances of a Skype client on two different machines and executed the same wildcard search query on them. The retrieved results were not completely identical. In all the wildcard searches we performed, the retrieved results were never completely identical.

D. Call Establishment and Teardown

We consider call establishment for SC v1.4 for the three network setups described in section III. Further, for each setup,

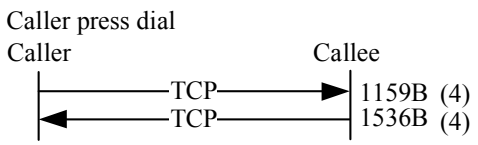

Figure 8. Message flow for call establishment when caller and callee SC v1.4 are on machines with public IP addresses and the callee is present in the buddy list of the caller. 'B' stands for bytes. Messages have been aggregated for space. Message size corresponds to the approximate cumulative size of messages exchanged between caller and a callee and vice versa. The number in paranthesis shows the total number of messages sent in that direction.

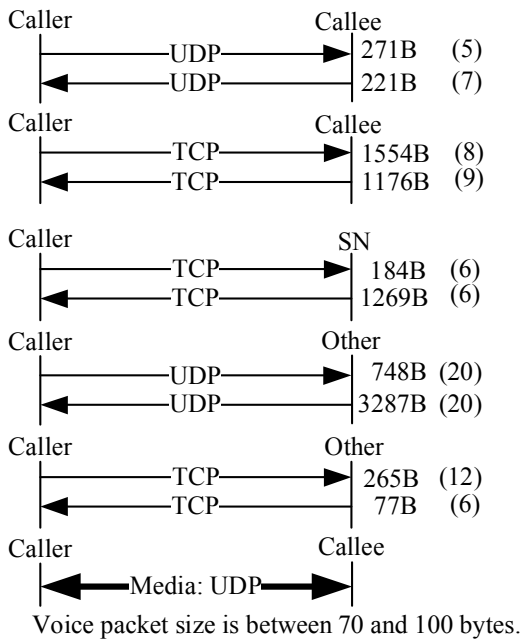

Figure 9. Message flow for call establishment when caller SC is behind a port-restricted NAT and callee SC is on a public IP address. 'B' stands for bytes. Not all messages are shown. Messages have been aggregated for space. Message size corresponds to the approximate cumulative size of messages exchanged between caller, callee, SN, other nodes and vice versa. The number in paranthesis shows the total number of messages sent in that direction.

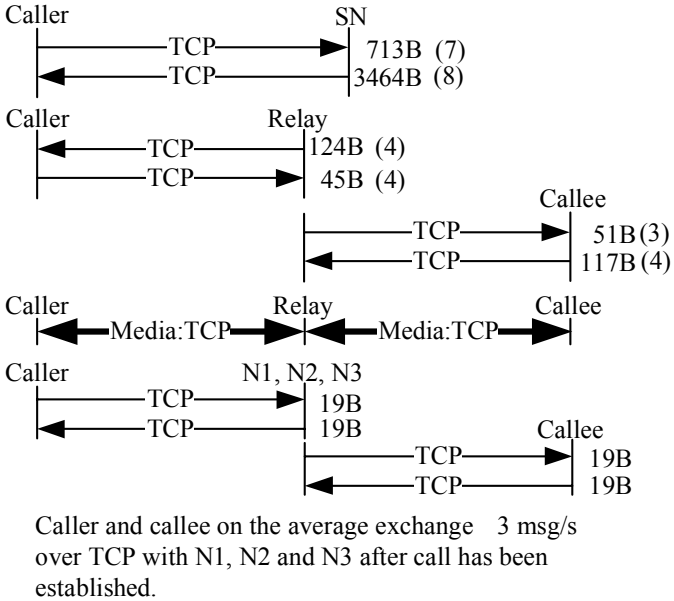

Figure 10. Message flow for call establishment when caller and callee SC are behind a port-restricted NAT and UDP-restricted firewall. 'B' stands for bytes and 'N' stands for a node. Not all messages are shown. Messages have been aggregated for space. Message size corresponds to the approximate cumulative size of messages exchanged between caller, callee, other nodes and vice versa. Voice traffic flows over TCP. The number in paranthesis shows the total number of messages sent in that direction.

we consider call establishment for users that are in the buddy list of caller and for users that are not present in the buddy list. It is important to note that call signaling is always carried over TCP.

For users that are not present in the buddy list, call placement is equal to user search plus call signaling. Thus, we discuss call establishment for the case where the callee is in the buddy list of the caller.

If both users were on machines with public IP addresses, online and were in the buddy list of each other, then upon pressing the call button, the caller SC established a TCP connection with the callee SC. Signaling information was exchanged over TCP. The aggregated message flow between caller and callee is shown in Figure 8.

The initial exchange of messages between caller and callee indicates the existence of a challenge-response mechanism. The caller also sent some messages (not shown in Figure 8) over UDP to alternate Skype nodes. For this scenario, approximately six kilobytes of data was exchanged.

In the second network setup, where the caller was behind a port-restricted NAT and the callee was on a public IP address, signaling information did not flow directly between caller and callee initially. Instead, the caller sent signaling information over TCP to an online Skype node which forwarded it to callee over TCP. After a call had been established, the media flowed directly between caller and callee over UDP. The message flow is shown in Figure 9. For this scenario, approximately eight kilobytes of data was exchanged.

For the third setup, in which both users were behind port-restricted NAT and UDP-restricted firewall, both caller and callee SC exchanged signaling information over TCP with another online Skype node. Caller SC sent media over TCP to an online node, which forwarded it to callee SC over TCP and vice versa. The message flow is shown in Figure 10. For this scenario, approximately eight kilobytes of data was exchanged.

There are certain advantages of having a node route the voice packets from caller to callee and vice versa. First, it provides a mechanism for users behind NATs and firewalls to talk to each other. Second, if users behind NATs or firewalls want to participate in a conference, and some users on public IP address also want to join the conference, this node serves as a mixer and broadcasts the conferencing traffic to the participants. The negative side is that there will be a lot of traffic flowing across this node. Users generally do not like the fact that arbitrary traffic could flow across their machines.

During call tear-down, signaling information is exchanged over TCP between caller and callee if they are both on public IP addresses, or between caller, callee and their respective SNs. The messages observed for call tear down between caller and callee on public IP addresses are shown in Figure 11.

For the second and third network setups, call tear down signaling is also sent over TCP. We, however, do not present these message flows, as they do not provide any interesting information.

For SC v1.4, we performed experiments to determine if the call signaling goes end-to-end when caller and callee SC are on

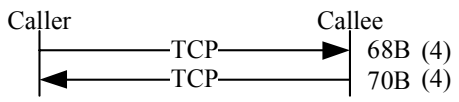

Figure 11. Call tear down message flow for caller and callee with public IP addresses. Messages have been aggregated for space. The number in paranthesis shows the total number of messages sent in that direction.

machines with public IP addresses and are in the buddy list of each other. Two instances of SC were started on two Windows machines with pubic IP addresses. Each instance had the other Skype user in its buddy list. After successful login, we waited

until each instance was aware of the presence of other instance. This was shown by a buddy item changing color to green in the buddy list. We ensured that there was no TCP connection between the two machines. Using NetPeeker, we blocked all outgoing and incoming connections except for the one destined for the callee machine and vice versa. A call attempt was made which succeeded.

We then blocked all TCP connection attempts between these two machines and attempted to make a call. The call attempt failed.

This experiment shows that for the scenario described above, the call signaling does go end-to-end. Also, call signaling is carried over TCP.

E. Media Transfer and Codecs

If both Skype clients (v1.4) were on machines with public IP addresses, then media traffic flowed directly between them over UDP. The media traffic flowed to and from the UDP port configured in the options dialog box. The voice packet size varied between 40 and 120 bytes. For two users connected to Internet over 100 Mb/s Ethernet with almost no congestion in the network, roughly 85 voice packets were exchanged both ways in one second. The total uplink and downlink bandwidth used for voice traffic was 5 kilobytes/s. This bandwidth usage agrees with the Skype claim of 3-16 kilobytes/s.

If either caller or callee or both were behind port-restricted NAT, they sent voice traffic to each other. The voice packet size varied between 40 and 110 bytes, which is the size of UDP payload. The bandwidth used was about 5 kilobytes/s.

If both users were behind port-restricted NAT and UDP-restricted firewall, then caller and callee sent and received voice traffic over TCP from another online Skype node. The TCP packet payload size for voice traffic varied between 30 and 90 bytes. The total uplink and downlink bandwidth used for voice traffic was about 5.5 kilobytes/s. For media traffic, SC used TCP with retransmissions.

In all three cases, the codec used was iSAC [13].

The Skype protocol seems to prefer the use of UDP for voice transmission. The SC will use UDP for voice transmission if it is behind a NAT or firewall that allows UDP packets to flow across.

1) Silence Suppression

No silence suppression is supported in Skype. We observed that when neither caller nor callee was speaking, voice packets were still flowing between them. While this increases the bandwidth usage, transmitting these silence packets has two advantages. First, it maintains the UDP bindings at NAT and second, these packets can be used to play some background noise at the peer. In the case where media traffic flowed over TCP between caller and callee, silence packets were still sent. The purpose is to avoid the drop in TCP congestion window size, which takes some RTT to reach the maximum level again.

2) Putting a Call on Hold

Skype allows peers to hold a call. Since a SC can operate behind NATs, it must ensure that UDP bindings are valid at a NAT box. On average, a SC sent one UDP packet every three seconds to the call peer, SN, or the online Skype node acting as a media proxy when a call is put on hold. We also observed that in addition to UDP messages, the SC also sent periodic messages over TCP to the peer, SN, or online Skype node acting as a media proxy during a call hold.

3) Codec Frequency Range

We performed experiments to determine the range of frequencies Skype codecs allow to pass through. A call was established between two Skype clients (v1.4). Tones of different frequencies were generated using the NCH Tone Generator [16] on the caller SC and output was observed on the callee SC and vice versa. We observed that the minimum and maximum audible frequency Skype codecs allowed to pass through were 50 Hz and 8,000 Hz respectively.

Using Net Peeker [8], we reduced the uplink and downlink bandwidth available to Skype application to 1,500 bytes/s, respectively. We observed that the minimum and maximum audible frequencies Skype codecs allowed to pass through remained unchanged, i.e., 50 Hz and 8,000 Hz, respectively.

4) Congestion

We checked Skype call quality in a low bandwidth environment by using Net Peeker [8] to tune the upload and download bandwidth available for a call. We observed that uplink and downlink bandwidth of 2 kilobytes/s each was necessary for bare minimum audible call quality. The voice was almost unintelligible at an uplink and downlink bandwidth of 1.5 kilobytes/s.

F. Keep-alive Messages

We observed for three different network setups that the SC v1.4 sent a refresh message to its SN over TCP. When SC was on a machine with a public IP address, a refresh message was sent every 120s.

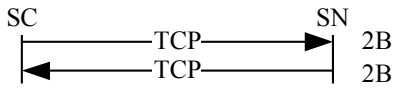

Figure 12. Skype refresh message to SN when SC was on a machine with public IP address.

V. CONFERENCING

We observed the Skype conferencing features for a three-user conference for the three network setups discussed in section III.A for Skype v1.4. We use the term user and machine interchangeably. Let us name the three users or machines as A, B, and C. Machine A was a 1.6 GHz Pentium 4 laptop with 512 MB RAM while machine B and C had a 3 GHz Pentium 4 CPU with 1 GB of RAM. In the first setup, the three machines had public IP addresses. A call was established between A and B. Then B decided to include C in the conference. From the ethereal dump, we observed that B and C were sending their voice traffic over UDP to SC on machine A, which was acting as a mixer. It mixed its own packets with those of B and sent them to C over UDP and vice versa as shown in Figure 13.

TABLE III
SKYPE, YAHOO, MSN AND GOOGLE TALK COMPARISON

	Application version	Memory Usage before call (caller, callee)	Memory Usage during call (caller, callee)	Process priority before call	Process priority during call	Mouth-to-ear latency	Latency Standard Deviation
Skype	1.4.0.84	19 MB, 19 MB	21 MB, 27 MB	Normal	High	96 ms	4
Yahoo	7.0.0.437	38 MB, 34 MB	43 MB, 42 MB	Normal	Normal	152 ms	12
MSN	7.5	25 MB, 22 MB	34 MB, 31 MB	Normal	Normal	184 ms	16
G-Talk	1.0.0.80	9 MB, 9 MB	13 MB, 13 MB	Normal	Normal	109 ms	10

In the second setup, B and C were behind port-restricted NAT, and A was on the public Internet. Initially, user A and B established the call. Both A and B were sending media to each other over UDP. User A then put B on hold and established a call with C. It then started a conference with B and C. We observed that both B and C were now sending their packets to A over UDP, which mixed its own packets with those coming from B and C, and forwarded it to them appropriately.

In the third setup, B and C were behind port-restricted NAT and UDP-restricted firewall and A was on the public Internet. User A started the conference with B and C. We observed that both B and C were sending their voice packets to A over TCP. A mixed its own voice packets with those coming from B and C and forwarded them to B and C appropriately.

If user B was in a call with user C using a relay D and if user B initiated a conference with user A, relay D was still being used between user B and C.

In the same experiments for Skype v0.97, we had observed that the most powerful machine always got elected as a conference host.

For a three party conference, Skype does not do full mesh conferencing [17].

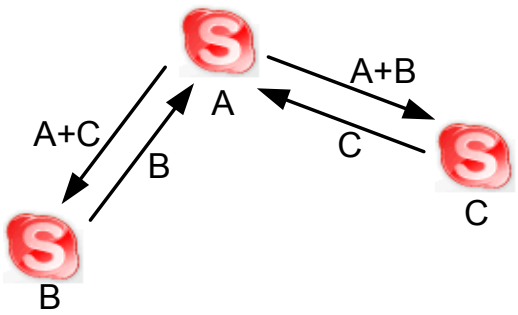

Figure 13. Skype three user conferencing

VI. OTHER EXPERIMENTS

Unlike MSN and Yahoo Messenger, which signs out a user if that user logs in on another machine, Skype allows a user to log in from multiple machines simultaneously. The calls intended for that user are routed to all locations. Upon picking a call at one location, the call is immediately cancelled at other locations. Similarly, instant messages for a user who is logged in at multiple machines are delivered to all the locations.

The SN is selected by the Skype protocol based on a number of factors like CPU and available bandwidth. It is not possible to arbitrarily select a SN by filling the HC with the IP address of an online SC. This conclusion was drawn from the following experiment. Consider two online Skype nodes A and B. A is connected to the Skype network and has only one entry in its HC. We call super node of A as SN_A. Now we modify the HC of SC on machine B, such that it only contains the IP address and port number of SC running at A. When B logged onto the Skype network, we observed that it connected to another super node rather than connecting to A.

To see if a different encryption key is embedded within each Skype executable, we compared the Skype setup files downloaded at randomly chosen times during a week. There were no differences between the setup files.

If two Skype users were behind the same NAT and established a call with each other, voice traffic flowed directly between them over the private network. If two Skype users were behind different NATs, then some ARP messages were seen on the wire. This seems to indicate that Skype was trying to determine network connectivity.

A. *Comparison with Yahoo, MSN and Google Talk IM/Voice Applications*

We measured memory usage and process priority before and during calls, and mouth-to-ear latency for the Skype, Yahoo, MSN, and Google Talk applications.

For our experiments, mouth-to-ear latency is defined as the difference between the time the words are spoken on one voice client, and the time they are heard at the other voice client given the two voice clients are already in a voice session. If both the original voice signal and the signal that traveled over the network can be recorded in a stereo format, then the delay or relative shift between these two signals can be calculated by computing a correlation between these two signals using a fast fourier transform (FFT). adelay [24] is a tool developed by Hao Huang in our IRT Lab that computes the mouth-to-ear latency using the technique described above.

For this experiment, the input signal was a pre-recorded Sun .au file of 24s. Skype, MSN, Yahoo and Google Talk applications were started on two separate laptops running Windows XP service pack 2 (SP2) and having identical hardware configuration. Each laptop had a Pentium (M) 1.7 GHz processor with 1 GB of RAM. Both machines had public IP addresses and were connected to a 100 Mb/s LAN. A voice session was established between respective voice clients. Using Ethereal we checked that both caller and callee Skype, Yahoo, MSN, and Google Talk clients were sending audio packets directly to each other. The pre-recorded audio file was played on a separate machine and the audio signal was provided as an input to the caller machine. The original signal and the signal received over the network were given as an input to the LineIn jack of another machine which ran the Cool Edit Pro version 2.1 [25] software. Using Cool Edit Pro, the two signals were recorded in Sun .au stereo format sampled at 8,000 Hz with 16 bit signed linear encoding. The sampling time for adelay was two seconds. For each voice client, the experiment was repeated four times. The results of these experiments are summarized in Table III. The mouth-to-ear latency is an average of four experiments for each IM client. The round-trip delay between the caller and callee machines, measured using ping, was less than one second.

We compared the memory usage and process priority for the three clients under test. Unlike Yahoo, MSN and Google Talk clients, Skype changes its priority to High priority, when a call is established.

TABLE IV
SN DISTRIBUTION PER DAY FOR 894 UNIQUE SN'S

	Unique SNs per day	Cumulative Unique SNs	Common SNs between previous and current day
Day1	224	224	
Day2	371	553	42
Day3	202	699	98
Day4	246	898	103

TABLE V
SN DISTRIBUTION PER COUNTRY (NON-US

Countries	Cumulative SNs
Taiwan	256
Israel	194
Japan	168
France	75
Switzerland	55

TABLE VI
SN DISTRIBUTION PER UNIVERSITY

Countries	Cumulative SNs
Harvard	367
Columbia	366
UNL	350
UPenn	245
BU	179

B. *Skype Super Node Map*

We performed experiments to get insights into the Skype super node selection mechanism. We know that at login, a SC must always connect to a SN. We take advantage of the fact that if a SC is behind a NAT, it will never become a SN and it will most likely connect to only one super node. Also, in a subsequent immediate login, a SC does not always reconnect to the same super node it connected last time. By having Skype repeatedly login onto the Skype network for an extended period of time, we can get a partial snapshot of the Skype super nodes.

For this experiment, we used AutoIt [27] to automatically start the Skype application, have it login on the Skype network, and then terminate it. AutoIt is a scripting tool for automating Windows tasks such as GUI input. There was a gap of 30 seconds between Skype application startup and termination and a gap of 10 seconds between the termination and the next startup. We ran the experiment for 96 hours (four days). Using netstat, we noted the IP address and port number to which the test machine had established a TCP connection. Since there were no applications running on the machine which established TCP connections except Skype, and since Skype must establish a TCP connection with a SN at login, the IP address and port number of the established TCP connection reported by netstat are indeed the IP address and port number of the SN to which Skype connects.

Theoretically, over a period of four days 8,640 login attempts are possible since the runtime of one iteration is 40 seconds. However, we recorded 8,175 login attempts and the lesser number is attributed to the script execution and Windows overhead. After removing the datasets that contained multiple TCP connections in ESTABLISHED state in the netstat data, we found 898 unique super nodes in 8,163 successful login attempts.

Using MaxMind tools, we determined the latitude, longitude, country and city of each IP address. They have been plotted on the map shown in Figure 16. MaxMind tools were unable to determine the country for four IP addresses, which were only used for 10 connections. Thus, our data set size was reduced to 8,153 successful login attempts and 894 unique super nodes.

The processing of data revealed the following:

- SN IP addresses distribution: US 83.7%, Asia 8.9%, Europe 7.1%.
- In 8153 login attempts, 2855 (35%) hostnames had a '.edu' suffix and belonged to 102 universities.
- Out of the 894 unique SNs, the top 20 nodes received 43.8% of the total connections and top 100 nodes received 70.5% connections.

Table IV shows the number of unique super nodes per 24 hours. It also shows the number of unique SNs that were common between the last day and the current day. Note that there were a total of 894 unique SNs. Table V shows the top five countries excluding US that received the most number of connections. Table VI shows the top five universities that received the highest number of connections.

VII. CONCLUSION

In this paper, we have tried to analyze various aspects of the Skype protocol by analyzing the Skype network traffic and by intercepting the shared library and system calls of Skype. We observed that Skype can work almost seamlessly behind NATs and firewalls. We believe that Skype client uses its version of STUN [5] protocol to determine the type of NAT or firewall it is behind. The NAT and firewall traversal techniques of Skype are similar to many existing applications such as network games. It is by the random selection of sender and listener ports, the use of TCP as voice streaming protocol, and the peer-to-peer nature of the Skype network, that not only a SC traverses NATs and firewalls but it does so without any explicit NAT or firewall traversal server. Skype uses TCP for signaling. It uses wide band codecs and has licensed them from GlobalIPSound [15]. Skype communication is encrypted.

The underlying search technique that Skype uses for user search is still not clear. Our guess is that it uses a combination of hashing and periodic controlled flooding to gain information about the online Skype users. Skype search mechanism falls back to the login server for all unsuccessful and some successful searches.

Skype has a central login server which stores the login name, password and buddy list of each user. Since Skype packets are encrypted, it is not possible to say with certainty what other information is stored on the login server. However, during our experiments we did not observe any subsequent exchange of information with the login server after a user logged onto the Skype network.

Compared to Yahoo, MSN, and Google Talk applications, Skype reported the best mouth-to-ear latency.

Skype is a selfish application and it tries to obtain the best available network and CPU resources for its execution. It changes its application priority to high priority in Windows during the time call is established. It evades blocking by routing its login messages over SNs. This also implies that Skype is relying on SNs, who can misbehave, to route login messages to the login server. Skype does not allow a user to prevent its machine from becoming a SN although it is possible to prevent Skype from becoming a SN by putting a bandwidth limiter on the Skype application when no call is in progress. Theoretically speaking, if all Skype users decided to put bandwidth limiter on their application, the Skype network can possibly collapse since the SNs hosted by Skype may not have enough bandwidth to relay all calls.

From our experience of analyzing the Skype protocol, we gather that packet intercept and blocking can be used for protocol reverse engineering. Classical packet sniffing tools

such as Ethereal are less useful when packet content is encrypted. Shared library and system call interception techniques can be used to manipulate the network traffic of a black box executable.

ACKNOWLEDGMENTS

The authors would like to thank Faisal Ghias Mir, Antonio Altamare, and Ricardo Barrato for the insightful discussions on Linux function call interception and various aspects of Skype. The authors would also like to thank David Chaum, Alok Agrawal, Eunsoo Shim, Sarah Baker, Balwant Rathore and numerous others who gave comments on our Skype technical report.

REFERENCES

[1] Skype. http://www.skype.com

[2] Kazaa. http://www.kazaa.com

[3] SkypeOut. http://www.skype.com/products/skypeout/

[4] SkypeIn. http://www.skype.com/products/skypein/

[5] J. Rosenberg, J. Weinberger, C. Huitema, and R. Mahy. STUN: simple traversal of user datagram protocol (UDP) through network address translators (NATs). RFC 3489, IETF, Mar. 2003.

[6] Global Index (GI): http://www.skype.com/skype_p2pexplained.html

[7] Ethereal. http://www.ethereal.com

[8] Net Peeker. http://www.net-peeker.com

[9] J. Rosenberg, H. Schulzrinne, G. Camarillo, A. R. Johnston, J. Peterson, R. Sparks, M. Handley, and E. Schooler. SIP: session initiation protocol. RFC 3261, IETF, June 2002.

[10] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, T. Berners-Lee. HTTP: hyper text transfer protocol. RFC 2616, IETF, June 1999.

[11] S. Baset and H. Schulzrinne. An analysis of the skype peer-to-peer Internet telephony protocol. Columbia University Technical Report CUCS-039-04, September 2004.

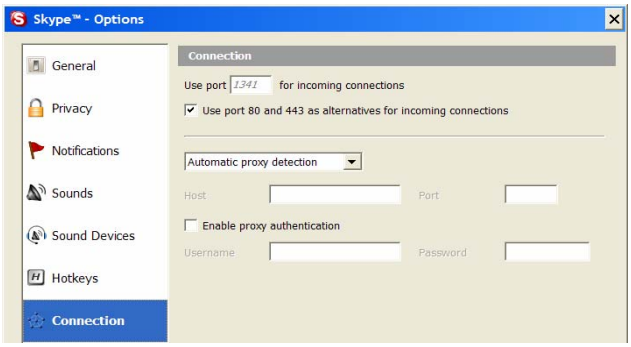

Figure 14. Skype (v1.4) connection tab. It shows the port on which Skype listens for incoming connections.

[12] iLBC codec. http://www.globalipsound.com/datasheets/iLBC.pdf

[13] iSAC codec. http://www.globalipsound.com/datasheets/iSAC.pdf

[14] iPCM codec. http://www.globalipsound.com/datasheets/iPCM-wb.pdf

[15] Global IP Sound. http://www.globalipsound.com/

[16] NCH Tone Generator. http://www.nch.com.au/tonegen/

[17] J. Lennox and H. Schulzrinne. A protocol for reliable decentralized conferencing. ACM International Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV), Monterrey, California, June 2003.

[18] Skype FAQ. http://support.skype.com/index.php?_a=knowledgebase&_j=questiondetails&_i=145

[19] J. Rosenberg, R. Mahy, C. Huitema. TURN: traversal using relay NAT. Internet draft, Internet Engineering Task Force, September 2005. Work in progress.

[20] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In Proc. ACM SIGCOMM (San Diego, 2001).

[21] memgrp. http://www.hick.org/code/skape/memgrep/

[22] Linux Function Interception. http://uberhip.com/godber/interception/

[23] X-Win32. http://www.xwin32.com/

[24] adelay. Measure the delay between two audio channels. http://www1.cs.columbia.edu/IRT/software/adelay/adelay.html

[25] Cool Edit Pro v2.1. http://www.softpedia.com/progDownload/Cool-Edit-Pro-Download-2076.html

[26] MaxMind. http://www.maxmind.com

[27] AutoIt. http://www.hiddensoft.com


```
<?xml version="1.0" ?>
- <config version="1.0" serial="6625" timestamp="1135714201.11">
  - <Lib>
    + <Account>
    + <BCM>
    - <Connection>
      - <Bandwidth>
          <CurSlotLength>6008</CurSlotLength>
          <LastRtTestTime>1135714068</LastRtTestTime>
          <OutHistory>7974</OutHistory>
        </Bandwidth>
        <DisablePort80>0</DisablePort80>
      + <EventServers>
      - <Firewall>
          <TcpInHistory>-1431655768</TcpInHistory>
          <UdpInHistory>-1431655768</UdpInHistory>
          <UdpOutHistory>1431655807</UdpOutHistory>
        </Firewall>
      - <HostCache>
          <_1>140.115.23.23:62601</_1>
          <_10>87.69.48.254:1586</_10>
          <_100>140.121.135.224:3256</_100>
          <_101>217.199.108.68:35749</_101>
          <_102>217.199.108.67:59107</_102>
```


Figure 15. Skype (v1.4) host cache list (shared.xml)

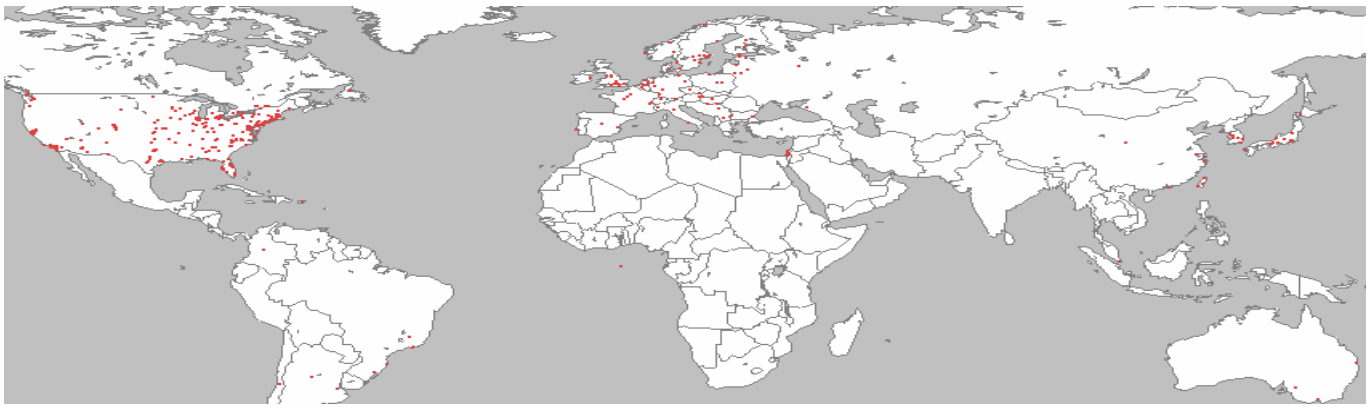

Figure 16. Worldmap of super nodes to which Skype establishes a TCP connection at login.