Insight into the Gtalk Protocol

Riyad Alshammari and A. Nur Zincir-Heywood Dalhousie University, Faculty of Computer Science

Halifax NS B3H 1W5, Canada

(riyad,zincir)@cs.dal.ca

Abstract

Google talk (Gtalk) is an instant messeger and voice over IP client developed by Google in 2005. Gtalk can work across firewalls and has very good voice quality. It encrypts calls end-to-end, and stores user information in a centralized fashion. This paper analyzes fundamental Gtalk functions such as login, firewall traversal, and call establishment under different network scenarios. The analysis is performed by both active and passive measurement techniques to understand the Gtalk network traffic. Based on our analysis, we devised algorithms for login and call establishment processes as well as data flow models.

I. Introduction

Voice over IP (VoIP) is becoming a major communication service for enterprises and individuals since the cost of VoIP calls is low and the voice quality is getting better. To date, there are many VoIP products that are able to provide high call quality such as Skype [1], Google Talk (Gtalk) [2], Microsoft Messenger (MSN) [3], and Yahoo! Messenger (YMSG) [4]. However, few information has been documented about protocol design used by these VoIP clients since the protocols are not standardized, many of them are proprietary, and the creators of these softwares want to keep their information confidential and protect their protocol design form competitors. In this paper, we have generated and captured network traffic in order to determine the broad characteristics of one of the most recent and fast growing VoIP applications, Google Talk.

Google talk (Gtalk) is an instance messenger developed by Google that allows its users to place voice calls, send text messages, check emails and transfer files. Gtalk provides very similar services as of MSN, YMSG and Skype since it has abilities for voice call, instant messaging and buddy lists. Practically, it has resemblance with Skype application since Gtalk encrypts its traffic; however the fundamental protocols and techniques it employs are relatively distinctive.

Gtalk application provides end users many services: i) voice communication, ii) video communication, iii) file transfer, and iv) chat services. The communication between users is established using a traditional end-to-end IP paradigm, but Gtalk routes calls through a relay node to ease the traversal of symmetric NATs and firewalls. Though Gtalk may relay on TCP and UDP at the transport layer, communication data are favorably carried over UDP. User's authentication is performed by a client-server architecture, using public key mechanisms. After the user (client) has been authenticated, all further communication is carried out with the 'nearest' Google server relay node. This way not only the quality of service can be guaranteed by Google but also both the scaling issues and the control issues can be solved by Google in a more seamless way. The main difference between Gtalk and other VoIP clients is that Gtalk can benefit from the vast amount of Google servers, which is being used as relay nodes (super nodes) to ensure the quality of service. On the other hand, other Peer-to-Peer (P2P) architectures like Skype can choose any suitable machine on the P2P network as a relay node[5].

The main contributions of this paper are the following. First, we characterize and explain the Gtalk communications at login and call establishments. Second, we investigate how Gtalk reacts to different and changing network conditions. In order to understand Gtalk protocol, we performed several experiments in a controlled environment: the testbed involved several PCs connected through the Internet and several network scenarios were emulated to observe how Gtalk reacts to different network restrictions. Moreover, the effects (if any) of different types of access technology (i.e. WiFi and Ethernet) were also investigated, as well as their combination. Overall, we have conducted over 100 experiments equivalent to more than 25 hours of Gtalk traffic. In these experiments we generated and captured more than 6 GB of traffic at both ends, where approximately 34 million packets were transmitted. Finally, we also make this testbed traffic available to the research community at this site [6]. In the following, we discuss the similar applications to Gtalk in Section II. Section III discusses the Gtalk main components, whereas the experiment set up is explained in Section IV. The Gtalk functions are discussed in Section V. Finally, conclusions are drawn and the future work is discussed in Section VI.

II. Similar applications

In this section we present an overview of three popular VoIP applications that provide similar functionality as Gtalk. These applications are: Skype, YMSG, and MSN. They provide similar services such as chat, transfer files, voice communication, privacy controls, buddy list and other options even though each clients has been designed individually. Moreover, the underlying protocols of theses applications is similar in one aspect that is authenticating is done by a central server.

Skype is based on a P2P architecture except the user authentication component, which is performed based on a central architecture (client-server model via public key mechanisms). After authentication is completed, all communication is performed on the P2P network. Therefore, user information and search queries is stored and broadcasted in a decentralized approach. On the P2P network, there are two types of nodes, ordinary nodes (hosts) and supernodes. An ordinary node is a Skype client that can be used to make communication through the service provided by Skype. On the other hand, any node on the P2P network with sufficient CPU power, memory and network bandwidth is a suitable candidate for a supernode. A supernode is part of the decentralized Skype network that can ease the routing of Skype traffic to bypass NATs and firewalls. Moreover, ordinary hosts have to connect to a supernode and register with the Skype login server in order to join the P2P network. Skype uses TCP or UDP at the transport layer to provide services to users such as voice and video calls, file transfer, chat and conference calls. For network communications, Skype mostly prefers UDP protocol. The communication among peers (users) on the P2P network is established via Internet Protocol (IP) paradigm. However, Skype has the ability to route traffic via supernodes to circumvent the NATs and firewall restrictions. A more detailed description of Skype protocol can be found in [5], [7].

On the other hand, YMSG uses a client-sever architecture for normal operations and voice-chat service [8], [9]. The client needs to contact one Yahoo server and then route all communications and subsequent communications through that server. It uses Session Initiation Protocol (SIP) [10] protocol for voice communication and transmit the data via Real-Time Transport Protocol (RTP) [11]. SIP is an application layer text based protocol for establishing multimedia session and adapts client-server architecture. Several major companies have chosen SIP for signaling VoIP traffic such as Microsoft and Yahoo. SIP message can be transmitted over UDP, TCP or SSL. MSN and YMSG VoIP applications use SIP for creating, modifying and terminating voice session. However in SIP, The voice and video communication is carried over RTP protocol, which job is to carry on voice data from caller to callee.

MSN has a similar approach as YMSG [8], [9]. It used a client-server architecture too for normal operations. MSN uses SIP for voice communication and transmits the data through RTP protocol. The major difference between YMSG and MSN is that YMSG needs only to contact one server and provide all services via that server whereas MSN has several types of servers for each service such as login, user search and voice that it provides. Moreover, its traffic is routed to the appropriate server based on the services requested by the client. Finally, YMSG and MSN do not have any encryption capabilities.

III. Gtalk Components

Gtalk is a P2P VoIP application based on Jabber/XMPP protcol. A Gtalk client opens random ports and exchanges traffic with Google server using a secure channel. Gtalk client does not store any information on the Windows registry compared to Skype, which maintains a list of Skype super nodes on the windows registry. Moreover, Gtalk receives information such as buddy lists and relay nodes from the Google server. Gtalk encrypts its messages from end-to-end and determines whether it is behind an NAT or a firewall. This section discusses these components and functionalities in detail.

A. Gtalk Client

A Gtalk client does not use a default port number. It opens a random port number and listens on it. However, random TCP port numbers are opened and closed as a Gtalk client authenticates and communicates with a Google server.

B. Google Server

Gtalk client needs to allocate a Google server to authenticate users and login. Firstly, a Gtalk client sends a Domain Name System (DNS) query to obtain the address of a Google server. Then, it opens a random TCP port and connects to the Google server using secure socket layer (SSL) protocol. Gtalk client does not store any information about the Google server in Windows registry. Every time a Gtalk client tries to login, it issues a DNS query to obtain the address of a Google server. We believe such an approach ensures that the load is balanced in terms of the usage of Google servers.

C. Relay Node

A Relay Node is a list of IP address - Port number pairs that are necessary for a client to use in order to connect to the network. In some ways, these are analogous to Skype supernodes. A Gtalk client sends a DNS query to obtain the list of a Google relay node, Figure 5. Comparing Gtalk to Skype, Skype maintains a list of super nodes in the windows registry in order to connect to the Skype network. On the other hand, Gtalk client is required to connect to a Google server (relay node) and all the traffic (between any two peers communicating) has to traverse through a Google relay node in order for end-to-end users to communicate. Unlike Skype client who cannot prevent itself from being a super node, Gtalk client does not become a relay node since all the relay nodes are on the Google network (a relay node has to be a Google server).

D. Buddy List

A Gtalk client obtains its buddy list from a Google server using Jabber/XMPP protocol [12] after logging in successfully. On the other hand, Skype stores the buddy list on the Windows registry. Extensible Messaging and Presence Protocol (XMPP) is a client server protocol. It is used to be called Jabber [13]. The Jabber/XMPP protocol can be used to carry on voice traffic over IP, get buddy list and other features. The Jabber/XMPP does not require a central server unlike MSN. Moreover, the clients on the Jabber/XMPP network do not connect to each other directly but they connect through the server

E. Encryption

Gtalk uses SSL [14] to authenticate users and to encrypt all traffic. As soon as a Gtalk client finds a Google server, it sends a Hello message using the SSL protocol and lists all cipher specs available by the client machine. Then, the Google sever chooses the encryption and starts setting up an SSL encryption session. It prefers the Rivest, Shamir and Adleman (RSA) algorithm with 128-bit encryption to exchange keys, Fig. 3.

F. NAT and Firewall

A Gtalk client uses the Simple Traversal of UDP over NAT (STUN) [15] protocol to reveal the type of NAT and/or firewall it comes across. It exchanges STUN messages with the Google server after the login to determine if it is behind an NAT and a firewall during the call establishment. During the experiments, we observed that a Gtalk client updates this information regularly by exchanging variants of STUN protocol messages with the Google server during the entire time of a call. Moreover, Gtalk uses STUN protocol to start and end calls. It sends a STUN Binding Request message to the Google server. Upon receiving the STUN response message, it establishes the connection with the Google server using Jabber XML protocol to transmit voice traffic. On the other hand, if Gtalk does not get the STUN response message, it connects to the server using SSL protocol and starts to transmit voice traffic.

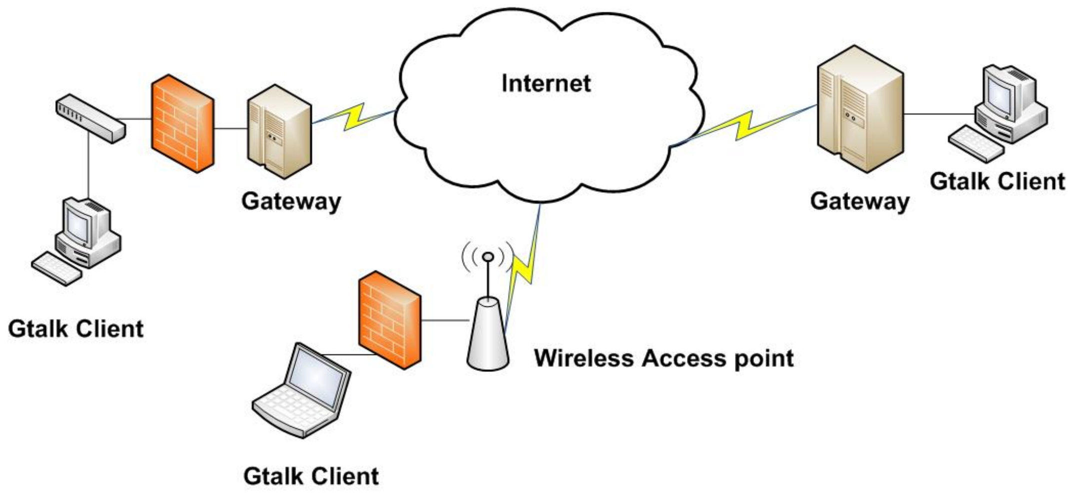


Fig. 1: Network Setup with restrictions.

IV. Testbed Setup and Traffic Generation

For this work, a Gtalk client was installed on each of the three windows XP machines. The first machine was a Pentium 4 2.4 GHz Core 2 Duo with 2 GB RAM, the second machine was a Pentium 4 2 MHz Core 2 Duo with 2 GB RAM, and the third machine was a MacBook 2 GHz Intel Core 2 Duo with 2 GB RAM. Two machines had a 10/100 Mv/s Ethernet and the third machine had a wireless 10/100 Mv/s card. Furthermore, one was connected to 1 GB/s network while the others were connected to a 10/100 Mb/s network. All three machines had Windows XP Service Packet 2 and all experiments were done using the Gtalk client version 1.0.0.104.

In all experiments, we have observed the Gtalk behavior from both ends. In all cases, we have performed experiments under several different network scenarios, Figures 1 and 2. These scenarios include: i) Firewall resritiction on one user end and no restriction at the other end; ii) Firewall restrictions at both users ends; iii) No restrictions at both users ends; iiii) Use of wireless and wire-line connections; iv) Blocking of all UDP connections, and v) Blocking of all TCP connections.

It should be noted here that during these experiments all the Internet communications went through our network's firewall. The firewall was configured to permit access to the aforementioned restrictions e.g. do not permit anything or permit limited well known port numbers such as port 22, 53, 80 and 443. Moreover, we have observed a Gtalk client through out the installation period as well as the first time login. Wireshark [16] and NetPeeker [17] were used to monitor and control network traffic. NetPeeker was used to block ports and to allow either both TCP and UDP ttraffic or only UDP or TCP traffic in order to analyze the behavior of a Gtalk client. On the other hand, Wireshark was used to capture traffic from both users ends. All Experiments were performed from October to December 2009.

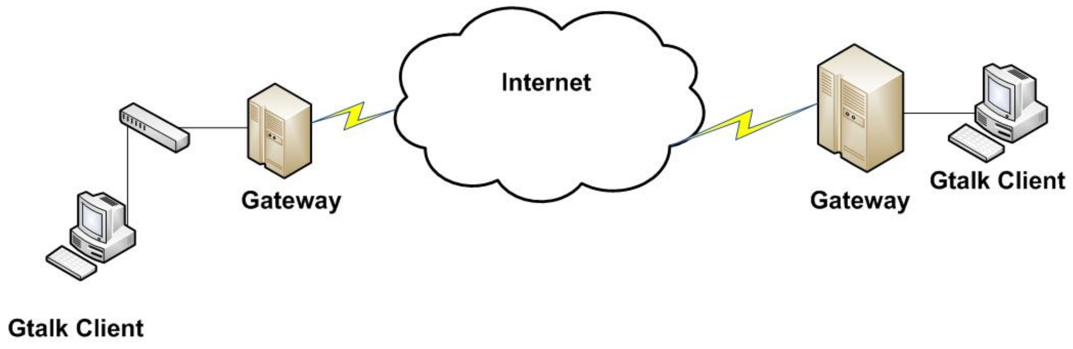


Fig. 2: Network Setup without restrictions.

V. Gtalk Functions

In our experiments, a Gtalk client can be found in one of the three phases, namely startup, login, and call establishment. This section explains each of these phases in detail.

A. Startup

When the Gtalk client runs for the first time after installation, it sents an “HTTP 1.1 Get” request to the Google server. The first line of this request contains the words ‘google-talk-versioncheck’. This is what we refer as the Startup phase.

B. Login procedure

To log in, the user needs to have an account at Google Gmail. Moreover, the contact list that the Gmail account has is automatically added to the buddy list. It should be noted here that we gained useful insights in the Gtalk login process by observing the message flow between Gtalk client and Google server, Figure 3. The experimental setup and observations for the loging process are described as the following. Figure 4 summarizes the login algorithm for Gtalk. To study the different steps of the login component of Gtalk client, we first blocked all UDP traffic. A Gtalk client was then started and a login attemp was made. Gtalk client kept trying untill the machine lost its internet connection since the DHCP [18] protocol was block too (no UDP traffic at all). In short, in this first experiment, Gtalk client was not successful in order to establish a call. In the next experiment, we blocked only the DNS traffic. In order for Gtalk client to login to the Gtalk network and to work properly it needs the IP address of the authenticating Google server “talk.google.com”. In this case, we observed that a Gtalk client always sends a DNS query for “gtalk.google.com” as its first step (we notice by blocking gtalk.google.com query and Gtlak client would report login failure). Moreover, we observe that it kept incrementing the waiting time to get the DNS reply. This shows that Gtalk needs to obtain the Google server address in order to work unlike Skype, which keeps the address of the super node in the Windows registry. When the Gtalk client gets the IP address of the Google server, it switches to using TCP protocol to complete the authentication.

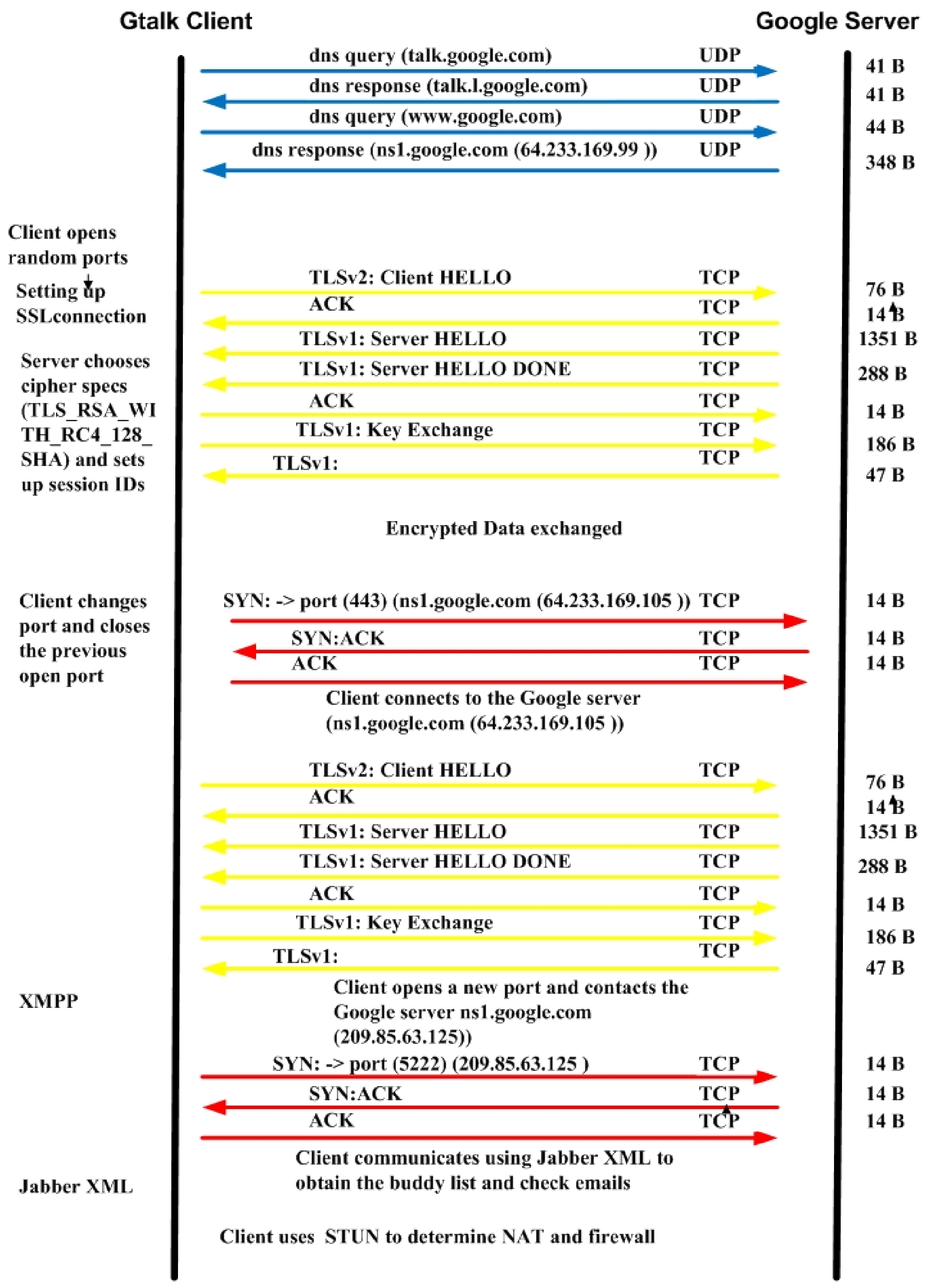

Gtalk Client
Google Server
dns query (talk.google.com) UDP 41 B
dns response (talk.l.google.com) UDP 41 B
dns query (www.google.com) UDP 44 B
dns response (ns1.google.com (64.233.169.99)) UDP 348 B
Client opens random ports
Setting up SSLconnection
TLSv2: Client HELLO TCP 76 B
ACK TCP 14 B
TLSv1: Server HELLO TCP 1351 B
TLSv1: Server HELLO DONE TCP 288 B
ACK TCP 14 B
TLSv1: Key Exchange TCP 186 B
TLSv1: TCP 47 B
Server chooses cipher specs (TLS_RSA_WITH_RC4_128_SHA) and sets up session IDs
Encrypted Data exchanged
Client changes port and closes the previous open port
SYN: -> port (443) (ns1.google.com (64.233.169.105)) TCP 14 B
SYN:ACK TCP 14 B
ACK TCP 14 B
Client connects to the Google server (ns1.google.com (64.233.169.105))
TLSv2: Client HELLO TCP 76 B
ACK 14 B
TLSv1: Server HELLO TCP 1351 B
TLSv1: Server HELLO DONE TCP 288 B
ACK TCP 14 B
TLSv1: Key Exchange TCP 186 B
TLSv1: TCP 47 B
XMPP
Client opens a new port and contacts the Google server ns1.google.com (209.85.63.125))
SYN: -> port (5222) (209.85.63.125) TCP 14 B
SYN:ACK TCP 14 B
ACK TCP 14 B
Jabber XML
Client communicates using Jabber XML to obtain the buddy list and check emails
Client uses STUN to determine NAT and firewall

Fig. 3: An overview of the packet flow for the login process. 'B' stands for bytes. SYN and ACK are shown to indicate who initiated TCP connection. Packet flows are not strictly according to time and packet size might change. The packet size corresponds to size of TCP/UDP payload. Not all packets are shown.

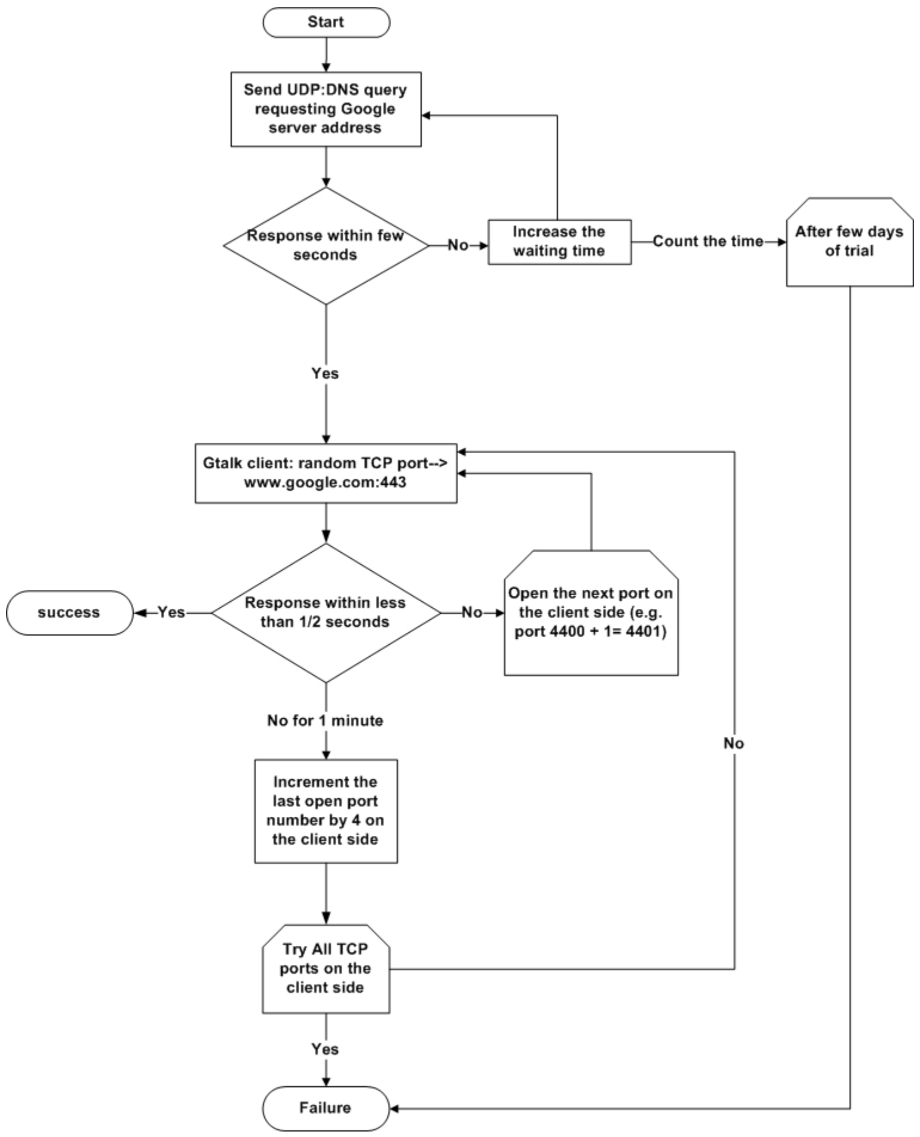
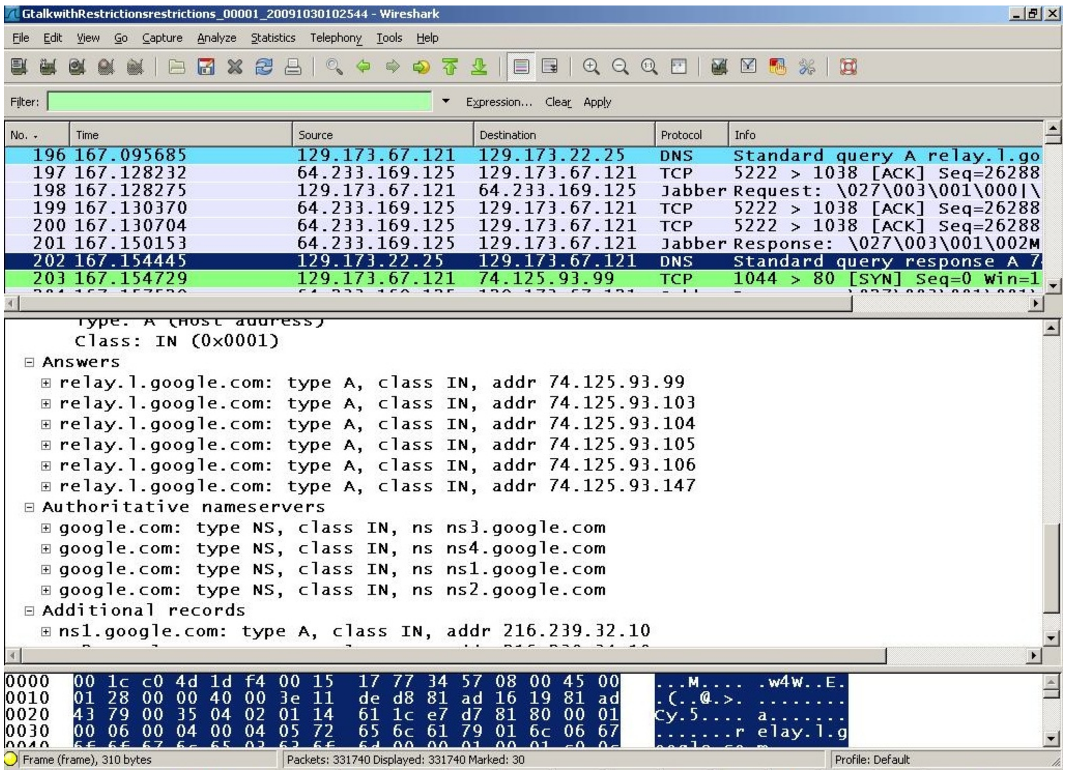

Start
Send UDP:DNS query requesting Google server address
Response within few seconds
No
Increase the waiting time
Count the time
After few days of trial
Yes
Gtalk client: random TCP port--> www.google.com:443
success
Yes
Response within less than 1/2 seconds
No
Open the next port on the client side (e.g. port 4400 + 1= 4401)
No for 1 minute
No
Increment the last open port number by 4 on the client side
Try All TCP ports on the client side
Yes
Failure

Fig. 4: An overview of the Gtalk Client login Algorithm. Please note that the authentication component with the login server is not shown here.

In the next experiment, we blocked all TCP traffic. In this case, Gtalk opened random port numbers on the client side and connected to port 443 on the server side. In this experiment, the Gtalk client kept trying to connect every half a second with a different port number from the client side (it opens the next available port on the client machine). After 1 minute without successfully connecting to the server, Gtalk incremented the port number by 4 and tried to communicate with that port (e.g. the last port tried was 44500, then it tried port 44504). In this experiment, we observe that a Gtalk client must esablish UDP and TCP connectons with Google server in order to work properly. If a Gtalk client cannot connect to a Google server, it will report a login failure.


GtalkwithRestrictionsrestrictions_00001_20091030102544 - Wireshark
File Edit View Go Capture Analyze Statistics Telephony Tools Help
Filter:
Expression... Clear Apply

```
No. -  Time        Source          Destination     Protocol Info
  196 167.095685  129.173.67.121  129.173.22.25   DNS      Standard query A relay.l.go
  197 167.128232  64.233.169.125  129.173.67.121  TCP      5222 > 1038 [ACK] Seq=26288
  198 167.128275  129.173.67.121  64.233.169.125  Jabber   Request: \027\003\001\000|\
  199 167.130370  64.233.169.125  129.173.67.121  TCP      5222 > 1038 [ACK] Seq=26288
  200 167.130704  64.233.169.125  129.173.67.121  TCP      5222 > 1038 [ACK] Seq=26288
  201 167.150153  64.233.169.125  129.173.67.121  Jabber   Response: \027\003\001\002M
  202 167.154445  129.173.22.25   129.173.67.121  DNS      Standard query response A 7
  203 167.154729  129.173.67.121  74.125.93.99    TCP      1044 > 80 [SYN] Seq=0 Win=1
```

```
        Class: IN (0x0001)
  ⊟ Answers
    ⊞ relay.l.google.com: type A, class IN, addr 74.125.93.99
    ⊞ relay.l.google.com: type A, class IN, addr 74.125.93.103
    ⊞ relay.l.google.com: type A, class IN, addr 74.125.93.104
    ⊞ relay.l.google.com: type A, class IN, addr 74.125.93.105
    ⊞ relay.l.google.com: type A, class IN, addr 74.125.93.106
    ⊞ relay.l.google.com: type A, class IN, addr 74.125.93.147
  ⊟ Authoritative nameservers
    ⊞ google.com: type NS, class IN, ns ns3.google.com
    ⊞ google.com: type NS, class IN, ns ns4.google.com
    ⊞ google.com: type NS, class IN, ns ns1.google.com
    ⊞ google.com: type NS, class IN, ns ns2.google.com
  ⊟ Additional records
    ⊞ ns1.google.com: type A, class IN, addr 216.239.32.10
```

```
0000  00 1c c0 4d 1d f4 00 15  17 77 34 57 08 00 45 00   ...M.... .w4W..E.
0010  01 28 00 00 40 00 3e 11  de d8 81 ad 16 19 81 ad   .(..@.>. ........
0020  43 79 00 35 04 02 01 14  61 1c e7 d7 81 80 00 01   Cy.5.... a.......
0030  00 06 00 04 00 04 05 72  65 6c 61 79 01 6c 06 67   .......r elay.l.g
```

Fig. 5: Google Server relay nodes.

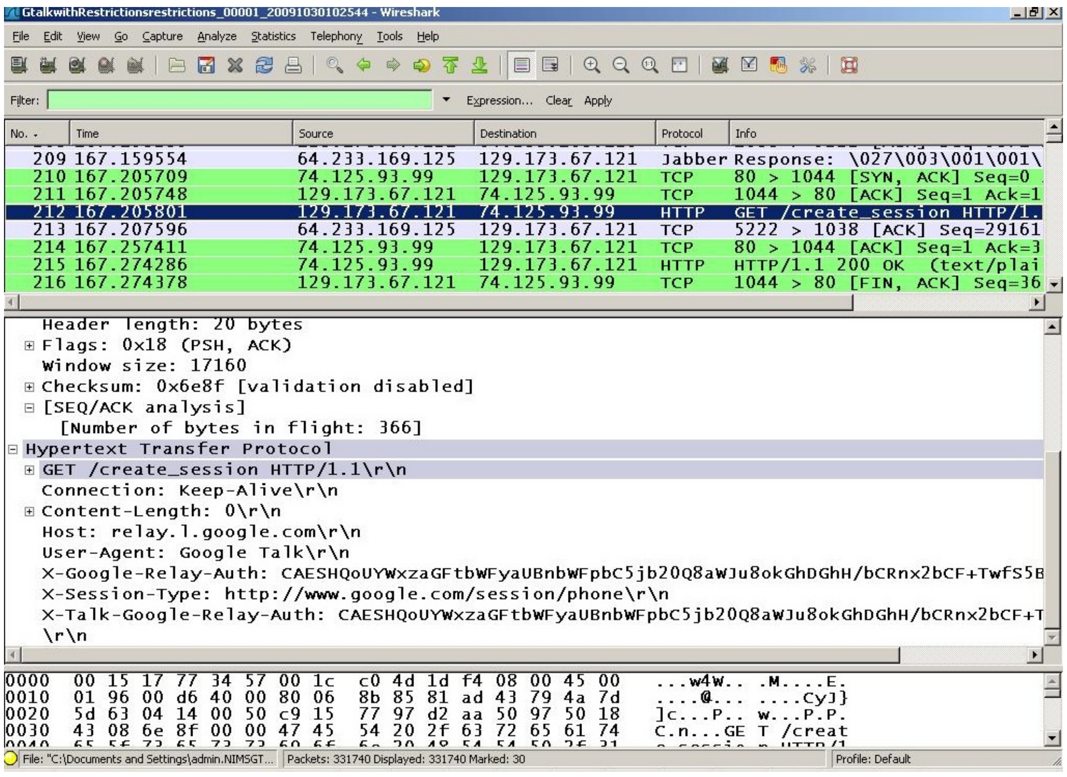
In terms of firewall restrictions, Gtalk creators take into account that most firewalls are configured to allow outgoing DNS traffic and TCP traffic to port 80 (HTTP) and port 443 (HTTPS). We observe that a Gtalk Client can take advantages of these facts to establish successful connections with the Google server. However, if these are blocked then Gtalk application cannot work.

C. *Subsequent Login Processes*

During our experiments, the subsequent login processes were identical to the first-time login process. The Gtalk client looks for the address of the Google server and starts a TCP connection with the server. Furthermore, we measured the time to login on the Google server network for the different network scenarios described in Section IV. Both the Gtalk with a public IP address and the Gtalk behind a port-restricted firewall took on average about 3 to 27 seconds to complete the login procedures. After a week of trying to login, a Gtalk client that is behind a UDP/TCP-restricted firewall (block All UDP or block All TCP), could not login and kept trying till we switch off the machine and called it a failure.

D. *Call Establishment*

In this paper, we analyzed the call establishment for the first three network scenarios described in Section IV. The call establishment happens between a caller and a callee who are in the buddy list and running Gtalk clients on


GtalkwithRestrictionsrestrictions_00001_20091030102544 - Wireshark
File Edit View Go Capture Analyze Statistics Telephony Tools Help
Filter:
Expression... Clear Apply

```
No.  Time        Source           Destination      Protocol Info
209 167.159554   64.233.169.125   129.173.67.121   Jabber   Response: \027\003\001\001\
210 167.205709   74.125.93.99     129.173.67.121   TCP      80 > 1044 [SYN, ACK] Seq=0
211 167.205748   129.173.67.121   74.125.93.99     TCP      1044 > 80 [ACK] Seq=1 Ack=1
212 167.205801   129.173.67.121   74.125.93.99     HTTP     GET /create_session HTTP/1.
213 167.207596   64.233.169.125   129.173.67.121   TCP      5222 > 1038 [ACK] Seq=29161
214 167.257411   74.125.93.99     129.173.67.121   TCP      80 > 1044 [ACK] Seq=1 Ack=3
215 167.274286   74.125.93.99     129.173.67.121   HTTP     HTTP/1.1 200 OK  (text/plai
216 167.274378   129.173.67.121   74.125.93.99     TCP      1044 > 80 [FIN, ACK] Seq=36
```

```
    Header length: 20 bytes
  ⊞ Flags: 0x18 (PSH, ACK)
    Window size: 17160
  ⊞ Checksum: 0x6e8f [validation disabled]
  ⊟ [SEQ/ACK analysis]
      [Number of bytes in flight: 366]
⊟ Hypertext Transfer Protocol
  ⊞ GET /create_session HTTP/1.1\r\n
    Connection: Keep-Alive\r\n
  ⊞ Content-Length: 0\r\n
    Host: relay.l.google.com\r\n
    User-Agent: Google Talk\r\n
    X-Google-Relay-Auth: CAESHQoUYWxzaGFtbWFyaUBnbWFpbC5jb20Q8aWJu8okGhDGhH/bCRnx2bCF+TwfS5B
    X-Session-Type: http://www.google.com/session/phone\r\n
    X-Talk-Google-Relay-Auth: CAESHQoUYWxzaGFtbWFyaUBnbWFpbC5jb20Q8aWJu8okGhDGhH/bCRnx2bCF+T
    \r\n
```

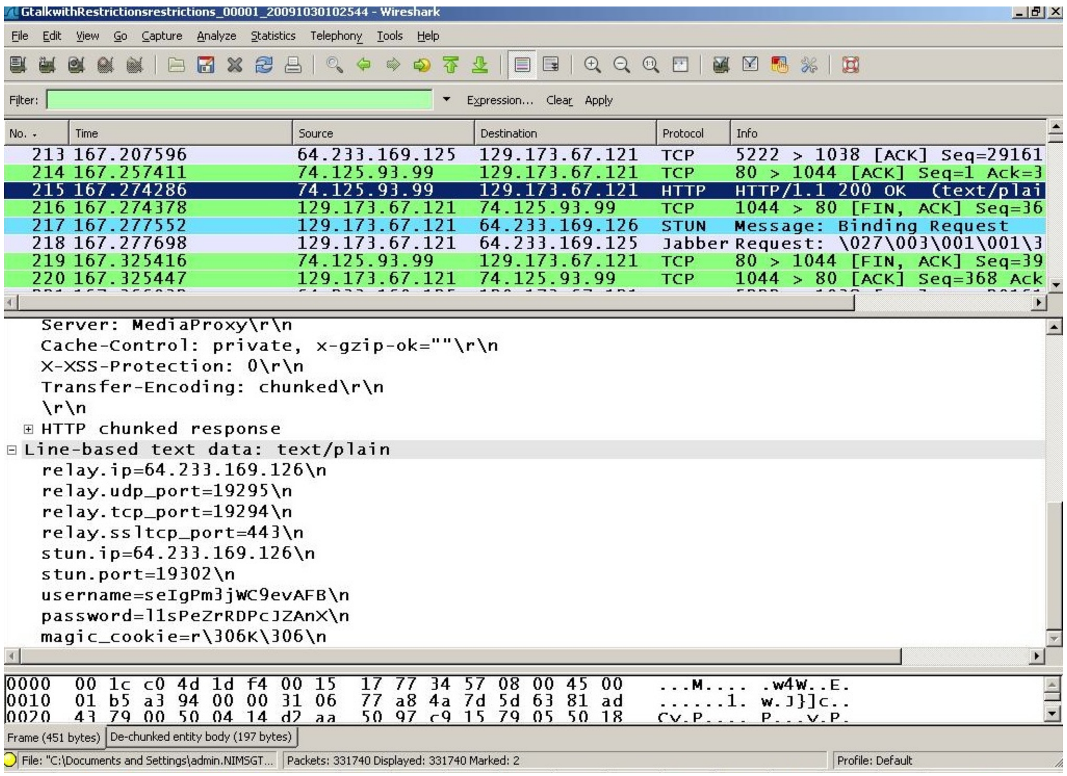
```
0000  00 15 17 77 34 57 00 1c  c0 4d 1d f4 08 00 45 00   ...w4W.. .M....E.
0010  01 96 00 d6 40 00 80 06  8b 85 81 ad 43 79 4a 7d   ....@... ....CyJ}
0020  5d 63 04 14 00 50 c9 15  77 97 d2 aa 50 97 50 18   ]c...P.. w...P.P.
0030  43 08 6e 8f 00 00 47 45  54 20 2f 63 72 65 61 74   C.n...GE T /creat
```

```
File: "C:\Documents and Settings\admin.NIMSGT...   Packets: 331740 Displayed: 331740 Marked: 30   Profile: Default
```

Fig. 6: Google Server determines the type of connection as 'phone' and sets up the session.

windows machines. If the users are online and are in the buddy list, upon pressing the call button, Gtalk establishes the connection as described below.

The call establishment algorithm between the caller and the callee is shown in Figure 8. For the three aforementioned network scenarios, Gtalk protocol behaves differently depending on the restrictions imposed on the network but not the connection type (WiFi, Ethernet). These behaviors are grouped into two: i) Firewall Restrictions; and ii) No firewall Restrictions. To establish the call, a Gtalk client requests the IP address of a Google relay node (server). In response to this, a list of server addresses is sent by DNS, Figure 5. Then, the Gtalk client chooses the first address on the list and associates an HTTP connection with it over TCP. Since Google servers provide many services such as mail and web search, the session setup with the relay server includes the 'phone' to define the Google service type, Figure 6. After that, the Google relay server sends the information to the Gtalk server responsible for establishing the voice talk between the caller and the callee, Figure 7. This information is send in clear text and includes the user name and the password for the server, the port numbers to connect (prefers UDP port 19295 and TCP 19294 on the server side) and alternative port numbers in the case of failing to connect via port 443 (HTTPS) on the server side. In all our experiments, the same information is sent (by the client) with all our voice connection attempts. Once the connection is established, the Gtalk client tries to determine the firewall restrictions (if any) by sending STUN messages over UDP. If the Gtalk client gets a response, it establishes the connection


GtalkwithRestrictionsrestrictions_00001_20091030102544 - Wireshark
File Edit View Go Capture Analyze Statistics Telephony Tools Help
Filter:
Expression... Clear Apply
Frame (451 bytes)
De-chunked entity body (197 bytes)
File: "C:\Documents and Settings\admin.NIMSGT...
Packets: 331740 Displayed: 331740 Marked: 2
Profile: Default

```
No.  Time        Source          Destination     Protocol Info
213 167.207596  64.233.169.125  129.173.67.121  TCP      5222 > 1038 [ACK] Seq=29161
214 167.257411  74.125.93.99    129.173.67.121  TCP      80 > 1044 [ACK] Seq=1 Ack=3
215 167.274286  74.125.93.99    129.173.67.121  HTTP     HTTP/1.1 200 OK  (text/plai
216 167.274378  129.173.67.121  74.125.93.99    TCP      1044 > 80 [FIN, ACK] Seq=36
217 167.277552  129.173.67.121  64.233.169.126  STUN     Message: Binding Request
218 167.277698  129.173.67.121  64.233.169.125  Jabber   Request: \027\003\001\001\3
219 167.325416  74.125.93.99    129.173.67.121  TCP      80 > 1044 [FIN, ACK] Seq=39
220 167.325447  129.173.67.121  74.125.93.99    TCP      1044 > 80 [ACK] Seq=368 Ack
```

```
    Server: MediaProxy\r\n
    Cache-Control: private, x-gzip-ok=""\r\n
    X-XSS-Protection: 0\r\n
    Transfer-Encoding: chunked\r\n
    \r\n
  ⊞ HTTP chunked response
⊟ Line-based text data: text/plain
    relay.ip=64.233.169.126\n
    relay.udp_port=19295\n
    relay.tcp_port=19294\n
    relay.ssltcp_port=443\n
    stun.ip=64.233.169.126\n
    stun.port=19302\n
    username=seIgPm3jWC9evAFB\n
    password=11sPeZrRDPcJZAnX\n
    magic_cookie=r\306K\306\n
```

```
0000   00 1c c0 4d 1d f4 00 15   17 77 34 57 08 00 45 00   ...M.... .w4W..E.
0010   01 b5 a3 94 00 00 31 06   77 a8 4a 7d 5d 63 81 ad   ......1. w.J}]c..
0020   43 79 00 50 04 14 d2 aa   50 97 c9 15 79 05 50 18   Cy.P.... P...y.P.
```

Fig. 7: Google Server sends the login information for the client.

over UDP using the port number provided by the server (preferably port 19295) on the server side and encrypts all its traffic. On the other hand, if Gtalk client does not get STUN response messages, then it keeps resending STUN messages. In this case, after 20 trial messages, the Gtalk client determines that it is behind a firewall and connects with the Google server using HTTPS (port 443) protocol and encrypts all its traffic. Moroever, Gtalk uses STUN to start and end the call as described in Section III-F.

We observe that in order for a Gtalk client to establish a call, it has to connect to the Google server, and the Google server is responsible to forward the traffic to the user. In our experiments, during the first network scenario where the caller was behind the firewall and the callee was with a public IP address without any restrictions, we observed that the Gtalk client connected to the Google server using TCP protocol from the caller side whereas the Gtalk client on the callee end connected to the Google server using UDP protocol. On the other hand, for the second scenario, where the firewall restriction was imposed on both the caller and the callee, the Gtalk client connected to the Google server using HTTPS protocol at both ends. Furthermore, in the third scenario where were no restrictions for both users, both Gtalk clients used UDP to connect to the Google server. In short, for the three scenarios studied, the Gtalk client exchanged more than 1605 kilobytes of data (payload size) with Google Servers.

In summary, in these experiments, the Gtalk client always exchanges traffic to establish the voice calls with the Google server whose IP address is 64.233.169.126. A reverse lookup of this IP address retrieved NS (name server)

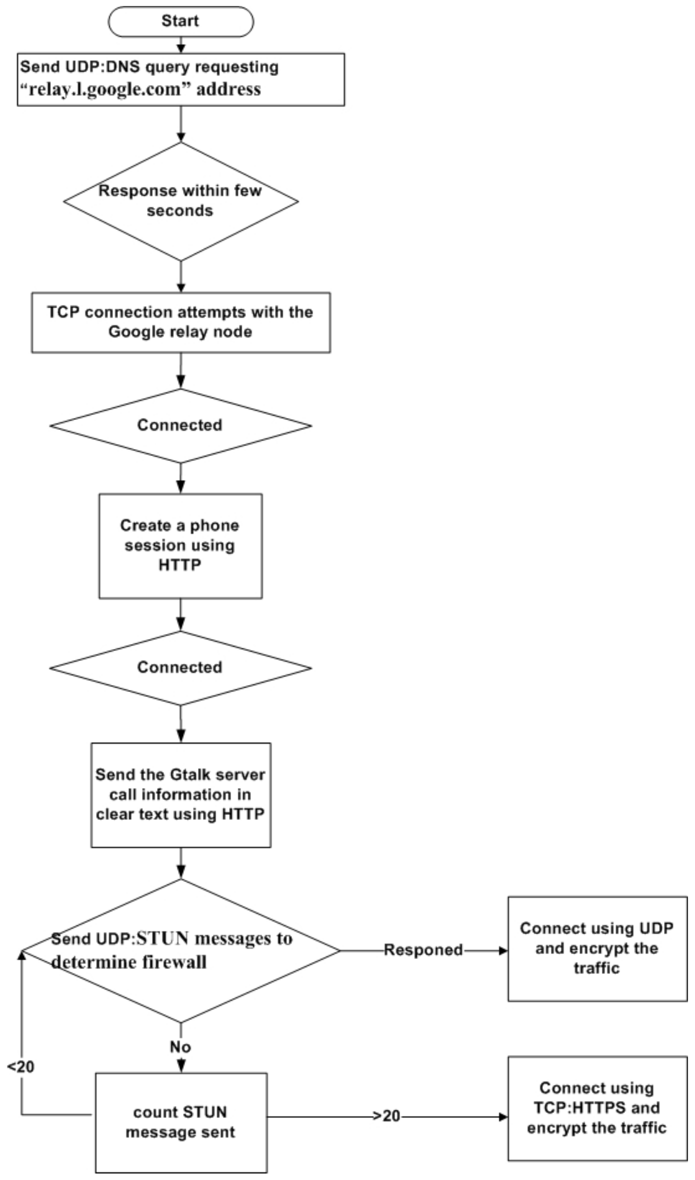

Start
Send UDP:DNS query requesting "relay.l.google.com" address
Response within few seconds
TCP connection attempts with the Google relay node
Connected
Create a phone session using HTTP
Connected
Send the Gtalk server call information in clear text using HTTP
Send UDP:STUN messages to determine firewall
Responed
Connect using UDP and encrypt the traffic
No
<20
count STUN message sent
>20
Connect using TCP:HTTPS and encrypt the traffic

Fig. 8: Call Establishment Algorithm.

records whose value is ns1.google.com. It seems from the reverse lookup that the server is hosted by Google based in USA. We speculate this Google server is responsible for our geographical area. There are many advantages of having a central server for a specific region to route packets from caller to callee and vice versa. Firstly, it ensures that a high quality of service can be achieved since the distances are minimized. Secondly, high performance productivity can be achieved by the server since it is responsible for managing only a certain area. Thirdly, it provides techniques for users behind an NAT and/or a firewall since the server can broadcast the traffic to all users. Fourthly, Google can have more control on traffic in terms of monitoring and managing. Last but not the least, such an approach preserves the users bandwidth by avoiding routing traffic on the client side unlike Skype, which employs the users network connection (bandwidth) to route traffic in the case of a super node.

VI. Conclusion

In this paper, we have analyzed and aimed to model the key components of Gtalk. We believe that this work will not only help the research community to understand how Gtalk works compared to other VoIP software on the Internet but also will shed more light into different types of encrypted traffic seen on today's networks. Based on our experiments, we observe that the Gtalk protocol takes advantages of the vast number of Google severs deployed around the world to ensure quality of services to its users. Gtalk provides very good voice quality and can work behind firewalls. Most noticeably, it is easy to install and use Gtalk. It uses both TCP and UDP protocols to establish the necessary communication. A Gtalk client stores all the user information on the server side and nothing on the client side. There is no direct communication with Gtalk clients to establish communication, the Gtalk clients on the caller side and callee side has to communicate through a Google server(s). Depending on the network restrictions, the communication between a Gtalk client and a Google server could use TCP or UDP protocol. However, Gtalk prefers UDP and if it is behind a firewall or NAT then it uses HTTPS (TCP port 443). Moreover, a Gtalk client uses STUN protocol to determine the type of NAT or firewall it is behind by sending messages to Google server.

Gtalk is similar to Skype, YMSG and MSN for using central server for user authentication. However, Gtalk is different than YMSG and MSN in the interior protocol design. YMSG and Yahoo depend on SIP and RTP protocols to start, establish, and end voice communications while Gtalk uses STUN and Jabber XML protocols to establish voice communications. Moreover, Gtalk encrypts its traffic while YMSG and MSN do not. On the other hand, Gtalk is different than Skype in terms of preserving the bandwidth of the client machine by not letting the client to be a super node. Furthermore, unlike Skype, Gtalk needs TCP and UDP protocols to provide its services. It is similar to Skype in terms of encrypting its traffic and preferring UDP protocol when it is not restricted.

References

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

[2] Google talk (Gtalk), last accessed October, 2009. http://www.google.com/talk/.

[3] MSN messenger. http://webmessenger.msn.com/

[4] Yahoo messenger. http://messenger.yahoo.com/

[5] S. A. Baset and H. G. Schulzrinne. An analysis of the skype peer-to-peer internet telephony protocol. In INFOCOM 2006. 25th IEEE International Conference on Computer Communications. Pages 1-11, April 2006.

[6] Gtalk Data set. http://www.cs.dal.ca/ riyad/DataSets

[7] D. Bonfiglio, M. Mellia, M. Meo, N. Ritacca, and D. Rossi. Tracking down skype traffic. In INFOCOM 2008. The 27th Conference on Computer Communications. IEEE, pages 261-265, April 2008.

[8] Jennings, R.B.; Nahum, E.M.; Olshefski, D.P.; Saha, D.; Zon-Yin Shae; Waters, C.; , "A study of Internet instant messaging and chat protocols," Network, IEEE , vol.20, no.4, pp.16-21, July-Aug. 2006

[9] de Menezes Filho, L.C.A.; da Costa, M.L.; Belem, R.L.; Arruda Filho, E.J.M.; , "Performance and Quality of Service on Free Softwares for VoIP," Optical Internet and Next Generation Network, 2006. COIN-NGNCON 2006. The Joint International Conference on , vol., no., pp.49-53, 9-13 July 2006.

[10] J. Rosenberg et al., SIP: Session Initiation Protocol, IETF RFC 3261, June 2002

[11] H. Schulzrinne, S. Casner, R. Frederick and V. Jacobsos, RTP: A Transport Protocol for Real-Time Applications , IETF RFC 3550. http://www.ietf.org/rfc/rfc3550.txt. July 2003.

[12] Jabber XML Protocol. http://xml.coverpages.org/jabber.html

[13] Jabber About us, last accessed March, 2010. http://www.jabber.com/CE/AboutUs

[14] The Secure Sockets Layer Protocol (SSL) . http://www.ietf.org/proceedings/32/sec/cat.elgamal.slides.html

[15] 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.

[16] Wireshark, Last accessed Sep, 2008. http://www.wireshark.org/.

[17] Net Peeker, last accessed October, 2009. http://www.net-peeker.com

[18] Dynamic Host Configuration Protocol (DHCP). http://www.ietf.org/rfc/rfc2131.txt