SIP, NAT, and Firewalls

Master's Thesis

By

Fredrik Thernelius

May 2000

DEPARTMENT
OF
TELEINFORMATICS

KUNGL
TEKNISKA
HÖGSKOLAN

Email:
Private: fredrik_thernelius@hotmail .com
Work: fredrik.thernelius@uab.ericsson.se

SIP:
Private: sip:fredrik_thernelius@hotsip.com
Work: sip:fredrik.thernelius@uab.ericsson.se

1 Table of contents

2 Abstract

The work presented in this Master's Thesis is an examination of how the SIP signaling, which occurs when a so called IP Telephony session is set up, will be able to traverse firewalls. It is necessary to solve the problems/issues that SIP brings about when the SIP messages traverse firewalls if this protocol ever will gain popularity.

In order to set up those data streams needed for transporting the sound in an IP telephony session the client enters his IP address and a port number in the SDP part of the SIP message to tell the other party where he should sent his audio data. Here is where problems occurs with the firewall. It needs to understand and interpret what the SIP message says to be able to set up rules for allowing traffic to pass through the firewall to these addresses. The problem is extended by the fact that it is common today to use "private addresses" on the LAN. These addresses are not allowed to exist on the Internet and thus the firewall software must remove this address and replace it with an address that is allowed on the Internet. A Network Address Translator (NAT) in the firewall normally does this together with Application Level Gateways (ALGs).

The work of this Master's Thesis has been focused around analyzing the above mentioned problems with SIP and Firewalls and then using this as input designing a prototype of an Application Level Gateway for SIP, which could be used together with perhaps a Linux firewall.

3 Sammanfattning

Det här examensarbetet är en undersökning av hur den SIP signalering som sker då ett så kallat IP-telefonisamtal ska sättas upp ska kunna ta sig genom brandväggar. Att lösa de problem som SIP för med sig då det gäller att ta sig förbi brandväggar är ett måste om detta protokoll någon gång ska kunna få stor spridning.

För att kunna sätta upp de dataströmmar som behövs för att transportera ljudet i ett IP-telefonisamtal sätter klienterna in sin IP-adress och ett portnummer i SIP meddelandena för att tala om för den andra parten vart hon ska skicka det audiodata hon sänder. Det är här problem uppstår med brandväggen. Den behöver förstå och tolka det som står i SIP meddelandet för att kunna sätta upp regler som tillåter att det passerar trafik genom brandväggen till dessa adresser. Ytterligare problem uppstår vid brandväggen då det idag är vanligt att många LAN använder "privata adresser". Dessa får inte förekomma ute på Internet och därför måste programvaran i brandväggen även kunna filtrera bort de privata adresserna och ersätta dem med IP-adress och portnummer som kan tillåtas att släppas ut på Internet. En adressöversättare (NAT) gör normalt detta tillsammans med "applikations-gateways" i brandväggen.

Tyngdpunkten på detta arbete har legat på att analysera de ovannämnda problemen med SIP och brandväggar och med utgång från detta sedan implementera en prototyp till den mjukvara, en "applikation-gateway" för SIP, som skulle kunna användas tillsammans med t.ex. en Linux-brandvägg.

4 Introduction

This is a Master's thesis project carried out at the Department of Teleinformatics at the Royal Institute of Technology in Stockholm. The work corresponds to twenty weeks (≈one semester) of full time studies and should demonstrate that the student has the ability to solve a problem independently with knowledge gained both from previous studies at the school and during the project.

4.1 Acknowledgements

I would like to thank my supervisor at the Department of Teleinformatics, Professor Gerald Q. "Chip" Maguire Jr., who has put up with me for such a long time. I would also like to thank everyone at Ericsson Utvecklings AB who have encouraged me to finally finish my Master's Thesis.

4.2 The goal of the project

The goals of this Master's Thesis are to study SIP, understand how the protocol interacts with firewalls which may use Network Address Translation (NAT) and then implement a prototype of an Application Level Gateway (ALG) for SIP.

4.3 Understanding the problems with SIP and Firewalls

The problems with SIP and firewalls are easily understood and I will outline them in the following paragraphs.

Firewalls are essentially made to prevent the community on the Internet from accessing hosts on the enterprise LAN, but still letting the employees on the LAN have access to the benefits of the Internet. This is done by putting rules in the firewall saying what traffic is allowed through from each direction, i.e. from the inside and from the outside. As can been seen from this, the rules in the firewall need to be applied asymmetrically. The need for asymmetric rules is seen from the following statement. *"Just because you are allowed to surf on the Internet does not automatically mean that someone one the Internet is allowed to surf on your Intranet."*

Services like HTTP and Telnet work on well-know ports (80 and 23 [RFC1700] respectively) and don't give firewall administrators much problem. SIP is another service, which also works on a well-know port, 5060. So why should the SIP signaling be any harder for the firewalls to handle? In answering this question one has to remember that SIP is *only* a way for setting up sessions, i.e. media streams between clients. It is not SIP in it self that makes it hard (not entirely true!), it is the information that describes the session that SIP helps to set up that is the hard part. SIP in itself is just a tool for helping users to find each other and to distribute information between them. The information that is distributed, the session description, is the IP address and the port number of the telephone application in the caller's computer. SIP is described in Section 6. The ports that are used for the media streams are emphermal (dynamic and > 1023). Thus the firewall will not know that an audio stream destined for a certain address and port should be let inside/outside or not. This is a problem that can only be solved by having some software analyzing all the SIP messages that pass the firewall on the well-known SIP port and then letting that software tell the firewall what it should do with packets to/from a certain address, e.g. let it through or deny and drop it. A few different kinds of firewalls are discussed in Section 7.

To make things more complicated the addresses that are used on the LAN can be from a range, which is not allowed on the Internet. These so-called "private addresses" are mostly used due to the fact that is it hard to get official IPv4 addresses, see Section 8. The private addresses can be used by anyone with the understanding that they are not allowed to let them outside their own domain. By using private addresses it is possible for a company to only have one global IP address, used by the outside interface of the firewall (or router if no firewall is used) while still having the possibility to use many thousands of private addresses in the LAN. The problem with SIP is now extended from having some software analyze the SIP messages and telling the firewall what ports to open to providing software that removes all of the private addresses in the outgoing SIP messages and replaces it with the firewall's own IP address and telling the firewall about its usage. When messages come back the software has to take out the global address it entered before and put the private address back. How this works is outlined in more

detail in Section 8.4. The exchange of addresses actually limits the security in SIP since end-to-end encryption and authentication is impossible. Section 9 gives an introduction to SIP security.

By definition in [RFC2663] a program doing the work described above, i.e. analyze messages and exchange addresses, would be called an Application Level Gateway (ALG) as it must understand the application data, and for SIP specifically it would be called a SIP ALG. As a part of the work in this Master's Thesis a prototype SIP ALG was implemented, see Section 10.

To use an Application Level Gateway is not the only way to get SIP to work through a firewall. Section 11 shows how a protocol called Real Specific IP (RSIP) can be used instead.

5 Earlier studies

The report begins with a review of what others already have done in this area: IP telephony call control used together with Firewalls and private addresses. The most important parts of the literature study was thus to look for and examine documents in this area.

No studies on the subject SIP and firewalls existed at the time of the start (1998) of this Master's Thesis. However, there has been at least one study [Intel] in this area on a similar protocol for session initiation and IP telephony, namely ITU's H.323. [Intel] outlines in great detail the problem with sending IP addresses and emphermal port numbers for the media streams in the set up messages. The paper also touches upon the subject of hiding IP addresses and performing network address translation (NAT) in the gateway between the LAN and the Internet, which also happened to be one of the objectives with this paper.

As to the subjects of private IP addresses and NAT numerous documents have been written. Many of these [Hasenstein97 and RFC1631] outline a similar problem as that which this Thesis addresses, in that FTP sends the emphermal ports for file transfers in its messages.

In February, two papers [SIPNAT and SIPALG] were published. Both of them address the issues discussed in this paper. These papers have not been considered in this project but they provide some interesting reading.

6 Introduction to Internet Telephony and Voice over IP

This Section will discuss the building blocks needed for IP telephony to work. The approach will be from bottom up. We start in Section 6.1 by looking at a protocol, the Real-Time Protocol, which enables real-time audio transfer on the Internet. In Section 6.2, we move on to look at a protocol (the Session Description Protocol) that lets the senders of real-time data know how to encode the audio in such a way that the recipient can decode the audio and listen to it. Finally, in Section 6.3, we dig into a protocol (the Session Initiation Protocol) that enables users to find and contact each other in an organized way.

Now, we can summarize the whole chain of events that will lead to an IP telephony session:

1. Adam, who wants to talk to Bob, sends a Session Initiation message to a server where he knows that Bob is registered. This server then forwards the message to Bob.
2. The initiation message contains a session description part that in turn contains the information about the media Adam wishes to use for the session.
3. When Bob receives the message he issues a response message with a similar session description.
4. When Adam receives the response he issues an acknowledgement message to Bob.
5. At this point both parties in the call can send real-time data to each other.

6.1 Streaming Audio – the Real-Time Protocol

The Real-Time Protocol (RTP) [RFC1889] delivers real-time data between two end systems. This is done in such a way that the receiving end system is able to reconstruct the original data stream sent by the other end system, even if the packets are delayed or arrive out of order. If packets are lost on the way, the protocol will be able to detect this but it does not support requests for retransmissions of any data. The sequence number in the RTP header, see Figure 1, is used to detect lost and out of order packets.

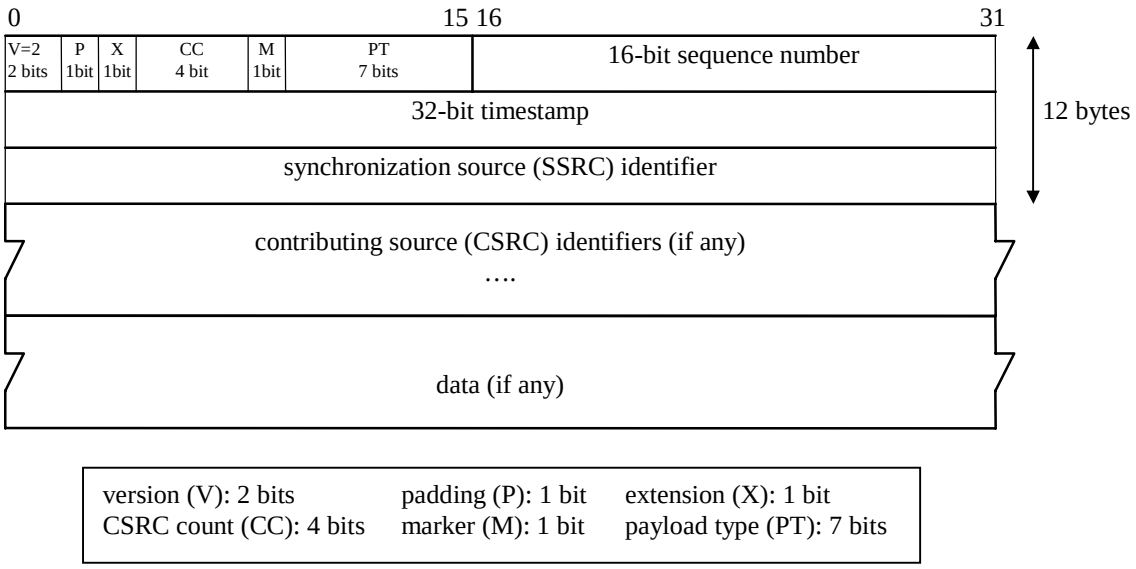

Figure 1. The RTP header

The reason for not supporting retransmission in the protocol is that it would most likely take too long (at least one RTT, which could be several hundred milliseconds) to request that the source resend the lost RTP packet and for this copy to arrive. A better solution, for the case of audio at least, is to extrapolate sound from previous audio samples to make up for the lost ones. Another algorithm to solve the problem due to lost packets is to just ignore the lost data and go on as if nothing has happened. Simply ignoring lost packets works, because the duration of the audio in one packet is relatively short, ranging from maybe 20 ms to about 60 ms. The loss of sound for that short period of time will not have a major influence of the quality. It is likely that it is not even noticed at all.

The topic of retransmission is a major reason for not using TCP [RFC793]. TCP, which is a reliable connection oriented protocol, uses retransmissions as a way to guarantee the delivery of the data handed to the TCP layer from the application layer. According to [Stevens97], TCP in BSD implementations uses a retransmission timer with a lower bound of 500ms. The reason that it cannot be lower is that an OS dependent timer with a resolution of this value is used to initiate the retransmissions. Thus the timer can have values of N*500 ms, where $N \geq 1$. A value of zero for N would cause an immediate retransmission. What resolution and what lower bounds other OS manufacturers use for their implementations of the TCP retransmission timer is outside the scope of this thesis to consider.

Instead of TCP, RTP normally uses UDP [RFC768] as the default transmission protocol. UDP does not provide any reliability features. UDP in turn uses IP, with best effort delivery to encapsulate its data. Any application level protocol that depends on UDP for transmission and still has the desire to be sure that any data sent is also received must implement its own retransmission algorithm.

Now we summarize the processing and encapsulation, see Figure 2, of the audio for an IP telephony session before it is sent from a host:

1. The sound from the microphone will be sampled at certain times. A number of samples are bundled together by the application to be the data encapsulated in a RTP packet. Typically 20 ms of sound is encapsulated into one RTP packet.
2. On the transportation level the RTP packet is encapsulated into a UDP datagram.
3. On the network layer the UDP datagram is encapsulated into an IP packet. For reference The UDP and TCP headers are shown in Figure 3 and Figure 4.
4. The IP packet is encapsulated into an Ethernet (or any other link layer protocol) frame and then the frame is sent off.

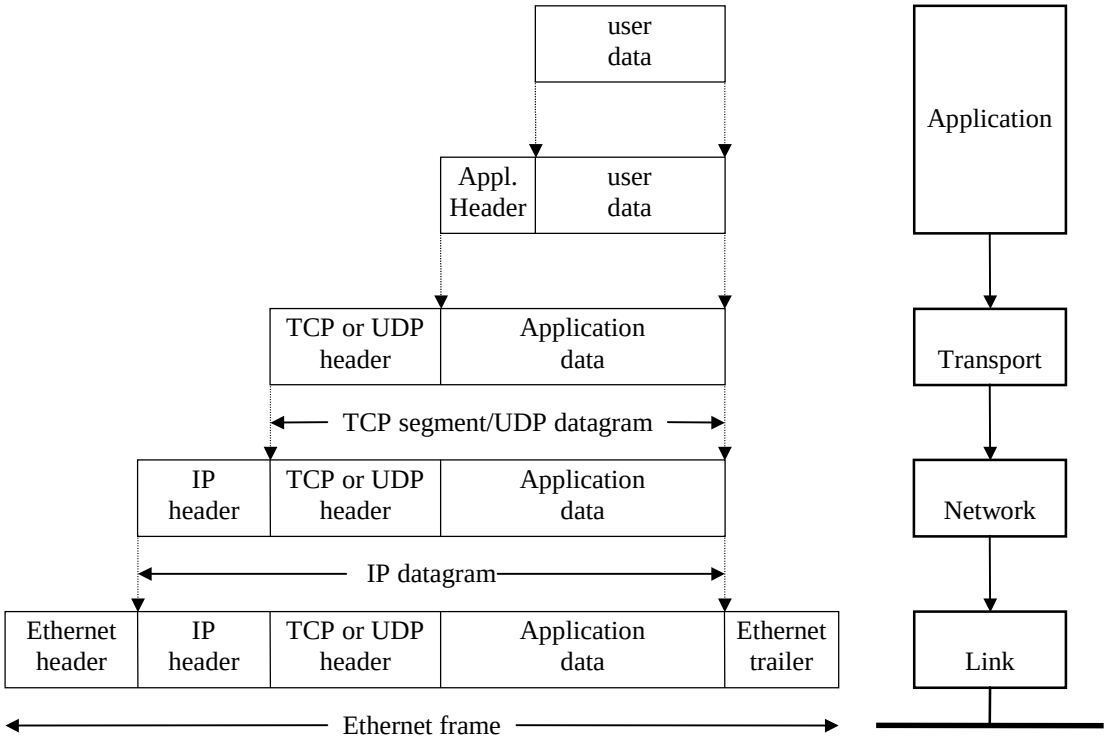

Figure 2. Encapsulation and layer distribution

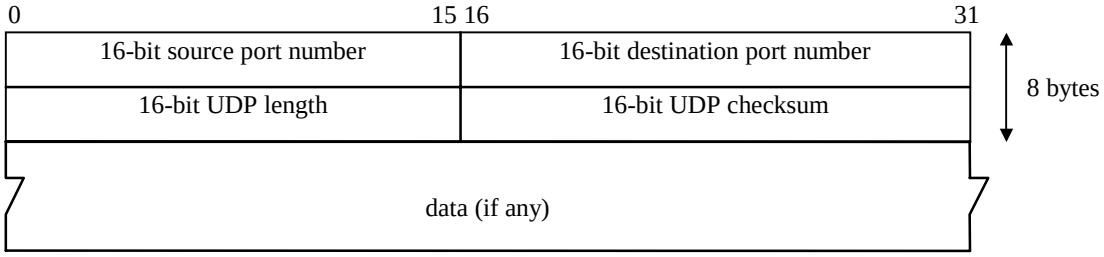

Figure 3. The UDP header

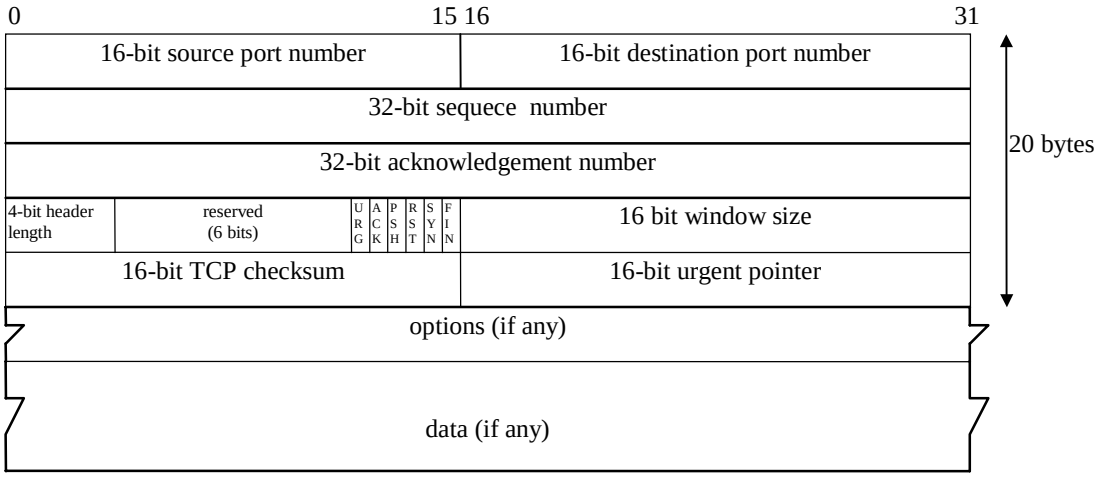

Figure 4. The TCP header

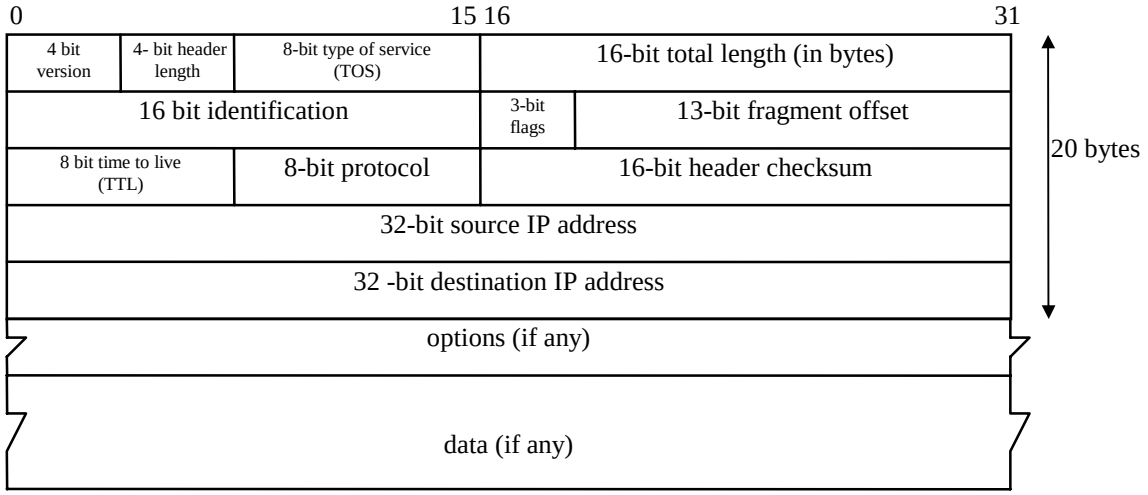

Figure 5. The IPv4 header

RTP has its own control protocol, the Real-Time Control Protocol (RTCP) [RFC1889]. The main purpose for RTCP is to give feedback on the quality of the delivery of data from the recipients of the data, to those who send it. The feedback is given in reports sent with RTCP and can for example contain the number of lost packets in the session and information about delays in the intermediate networks. RTCP is not streamed continuously in the same way as RTP, instead it is sent periodically with a typical period of a few seconds between the reports.

Any given RTP media stream is always assigned to an even port number while the associated RTCP reports are sent on the next higher, and thus odd, port number.

6.2 The Session Description Protocol

The Session Description Protocol (SDP) [RFC2327] has three main objectives that need to be achieved before an IP telephony session can begin. All of these there objectives have to do with letting the one whom you want to speak with know what you want from the session. First, you need to tell the other party what kind of media you want to receive: audio, video, or both[1]. The second thing is how you want the media to be coded by him so that you can understand what is being sent. The third thing you need is to inform the other party about is the address and port you want the media to be delivered to. For this to work the person on the other side will also have to send

[1] For multicast sessions, the session description work opposite to unicast sessions. Here the session descriptions tell what the client is willing to send. Multicast sessions are not considered in this thesis.

you a session description with his information to you, or else you will not be able to send any media data to him. A typical session description looks like the one in Figure 6.

```
v=0
o=uabfrth 955720785594 955720785594 IN IP4 134.138.242.7
s=Basic Session
c=IN IP4 134.138.242.7
t=955720785594 0
m=audio 2328 RTP/AVP 8 0 96 98 99 97
a=rtpmap:96 SC6/6000
a=rtpmap:98 SC6/3000
a=rtpmap:99 RT24/2400
a=rtpmap:97 VR15/1500
```

Figure 6. Example of a session description

6.2.1 Text based

As already seen in Figure 6 in the previous section, SDP is entirely textual. This means that the information conveyed by SDP is not coded into a more compact form, unlike H.323, which uses ASN.1. The recommended character set for SDP is ISO 10646, in UTF-8 encoding [RFC2279]. The specification for SDP allows for the use of other character sets if they are needed.

6.2.2 The Structure of SDP

The grammar for SDP is very structured and strict when it tells what a session description should look like. SDP does not allow for headers to come in any other order than the one shown in Figure 7. All the lines, i.e. header fields, follow the same pattern, <type> = <value>. The type-field is always only one character long and this character is always in lower case.

```
Session description
   v=  (protocol version)
   o=  (owner/creator and session identifier).
   s=  (session name)
   i=* (session information)
   u=* (URI of description)
   e=* (email address)
   p=* (phone number)
   c=* (connection information - not required if included in all media)
   b=* (bandwidth information)
   One or more time descriptions
   z=* (time zone adjustments)
   k=* (encryption key)
   a=* (zero or more session attribute lines)
   Zero or more media descriptions

Time description
   t=  (time the session is active)
   r=* (zero or more repeat times)

Media description
   m=  (media name and transport address)
   i=* (media title)
   c=* (connection information - optional if included at session-level)
   b=* (bandwidth information)
   k=* (encryption key)
   a=* (zero or more media attribute lines)

* optional item
```

Figure 7. SDP header fields

Some header fields are of special/major importance to this thesis will be discussed a little bit further, as for the rest the listing above is sufficient.

- the **o**rigin field o=<username> <session id> <version> <network type> <address type> <address>

The parameters of the origin field will together form a unique identifier for the current session. The last parameter contains an address, which could be an IPv4 address expressed either in dotted-decimal form or as a fully qualified domain name. None of the parameters in the origin field are intended to be used for routing media between clients.

- the **c**onnection field c=<network type> <address type> <connection address>

The connection field can exist in both the session description and the media description part of the SDP. Its purpose is to give the port number given in the media field an address to be associated with. If there is a connection field in the media description part then the address in this field is prioritized over the address that was possibly given in a connection field in the session description part.

- the **m**edia field. m=<media> <port> <transport> <fmt list>

The purpose of the media field is to let the other party in the session know what kind of media, audio or video, or maybe something entirely different, the recipient of the SDP should deliver, to what port on the associated connection address (see above) the media should be delivered to, and in what way the media should be coded. The example in Figure 6 uses two standard codecs [RFC1890] denoted 8 and 0 in the media field. In the same media field are four non-standard codecs, denoted 96, 97, 98 and 99, declared. The non-standard codecs are defined in the following attribute fields, one for each codec number. The following table shows the standardized payload types [RFC1890].

PT	encoding name	audio/video (A/V)	clock rate (Hz)	channels (audio)
0	PCMU	A	8000	1
1	1016	A	8000	1
2	G721	A	8000	1
3	GSM	A	8000	1
4	unassigned	A	8000	1
5	DVI4	A	8000	1
6	DVI4	A	16000	1
7	LPC	A	8000	1
8	PCMA	A	8000	1
9	G722	A	8000	1
10	L16	A	44100	2
11	L16	A	44100	1
12	unassigned	A		
13	unassigned	A		
14	MPA	A	90000	
15	G728	A	8000	1
16--23	unassigned	A		
24	unassigned	V		
25	CelB	V	90000	
26	JPEG	V	90000	
27	unassigned	V		
28	nv	V	90000	
29	unassigned	V		
30	unassigned	V		
31	H261	V	90000	
32	MPV	V	90000	
33	MP2T	AV	90000	
34--71	unassigned	?		
72--76	reserved	N/A	N/A	N/A
77--95	unassigned	?		
96--127	dynamic	?		

Table 1. Payload types (PT) for standard audio and video encodings

6.3 SIP overview

The session initiation protocol (SIP) [RFC2543] is a signaling protocol for setting up sessions between clients over a network, i.e. the Internet. These sessions do not necessarily have to be Internet telephony sessions. SIP could just as well be used for setting up gaming sessions or for distance learning where a lecture is streamed out to the participants. M. Handley, H. Schulzrinne, E. Schooler, and J. Rosenberg, authors of the SIP protocol give a more comprehensive description, as shown in Figure 8. The official SIP web site can be found at [SIPSite].

"The Session Initiation Protocol (SIP) is an application-layer control (signaling) protocol for creating, modifying and terminating sessions with one or more participants. These sessions include Internet multimedia conferences, Internet telephone calls and multimedia distribution. Members in a session can communicate via multicast or via a mesh of unicast relations, or a combination of these. SIP invitations used to create sessions carry session descriptions which allow participants to agree on a set of compatible media types. SIP supports user mobility by proxying and redirecting requests to the user's current location. Users can register their current location. SIP is not tied to any particular conference control protocol. SIP is designed to be independent of the lower-layer transport protocol and can be extended with additional capabilities." [RFC2543]

Figure 8. SIP description

It is outside the scope of this paper to discuss all the features of the protocol, but I will however discuss at least a few basic ones. See Figure 9 for an example of a SIP message. The different parts of a SIP message will be discussed in the following Sections.

```
INVITE sip:uabfrth@134.138.228.159 SIP/2.0
via: SIP/2.0/UDP 134.138.242.7:5062
from: sip:Fredrik.Thernelius@uab.ericsson.se
to: sip:uabfrth@134.138.242.7
call-ID: 955720785564@134.138.242.7
cseq: 1444 INVITE
user-agent: Ellemtel-PICo/R2H
contact: sip:Fredrik.Thernelius@134.138.242.7:5062
content-type: application/sdp
content-length: 250

v=0
o=uabfrth 955720785594 955720785594 IN IP4 134.138.242.7
s=Basic Session
c=IN IP4 134.138.242.7
t=955720785594 0
m=audio 2328 RTP/AVP 8 0 96 98 99 97
a=rtpmap:96 SC6/6000
a=rtpmap:98 SC6/3000
a=rtpmap:99 RT24/2400
a=rtpmap:97 VR15/1500
```

Figure 9. Example of a SIP message

6.3.1 Call setup - 1

The SIP sessions are set up by using a three-way handshake procedure (much like TCP), see Figure 10. When client A wants to set up an IP telephony session with client B, A sends an INVITE (1) request to B. The INVITE message contains a payload with a description of the session he/she wants to set up with B. If it is an IP telephony session that is about to be set up, then the session description contains information about which audio encoding types (see Table 1). A can understand and it also specifies on which ports A wants the RTP audio data sent to. The protocol to convey session descriptions is called the Session Description Protocol (SDP), see Section 6.2 for a summary. It is not mandatory for SIP to use SDP, but it is the only one defined so far.

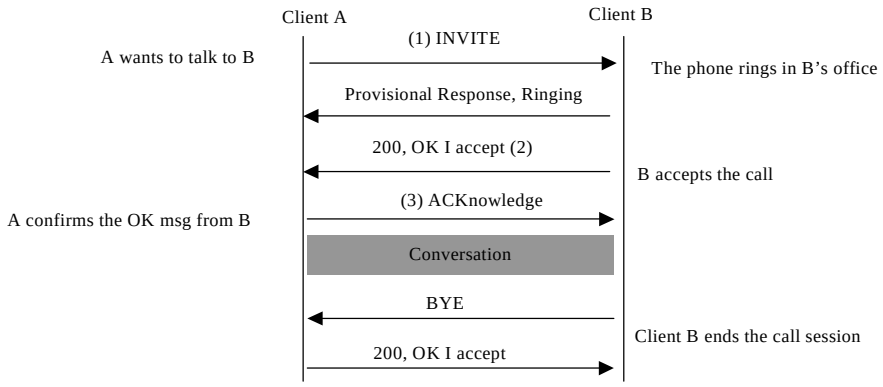

Figure 10. Setting up a SIP session

When B accepts the call his user agent sends a message (2) with a response code of 200. Any 2xx response means that the message was successfully received, understood, and accepted. See Section 6.3.9 for a summary of response codes. In the response client B adds his codec capabilities and the port numbers where he wants A to send his RTP data to. The final part (3) of the three-way handshake occurs when A sends an acknowledgement to B. By sending an ACK the caller confirms that it has received the response from the callee. After the setup procedure is completed the conversation can begin.

Figure 10 also shows an optional message, a provisional response. The provisional response here is an informational message that provides feedback to the caller that the phone is ringing on the receiving side.

6.3.2 Call setup – 2

The previous section on call setup showed what types of messages are involved in setting up a SIP connection between two users, but it did not say how the caller found out where the callee was. In order to find out where to send the INVITE message in that example, it is first necessary to find out which SIP server is responsible for a particular user. This can be quite tricky and is best explained in [RFC2543 - Section 1.4.2]. After receiving information about where the SIP server is located the caller will be able to send the INVITE to that location or locations (if several alternatives were received).

The SIP server will read the To field in the message, see figures 9, 11 and 12, and initiate a search for the particular user, who is pointed out by the SIP URL in this field. The user is located via a query to the location server, which could be an LDAP server [RFC1777] as suggested in [RFC2543]. When the SIP server receives a location or locations for that user it will react in one of two ways. It will either forward the packet to its destination, see Figure 11, or it will send a response back to the caller, see Figure 12, which contains the location(s) of the callee and thus lets the caller contact the callee himself. The server is said to be working in either proxy- or redirect mode.

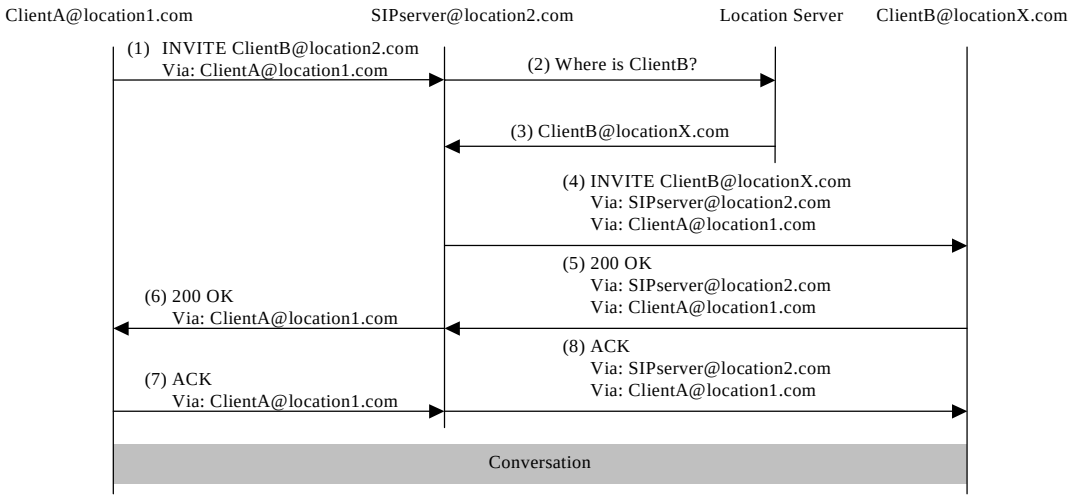

Figure 11. SIP Server in proxy mode

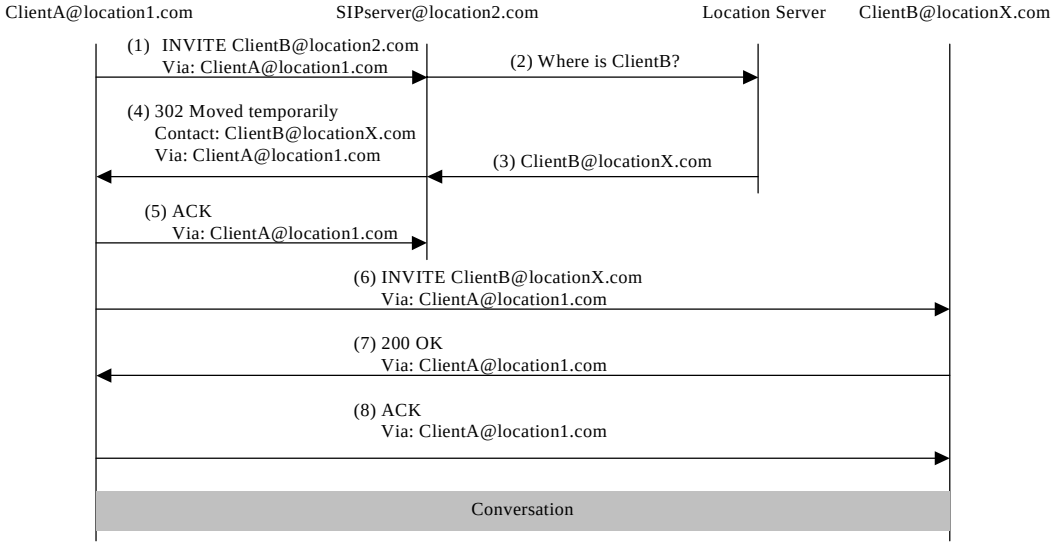

Figure 12. SIP server in redirect mode

6.3.3 Routing SIP messages

Each SIP entity that is sending out SIP requests is required to add one Via header field with its own address. When the request has arrived at responding client the list might be very long if there were many SIP servers in between the caller and the callee. The Via list tells how responses should be routed back to the caller. The responding client will thus take the address from the top-most Via header and send the response to that address. A SIP entity that receives a response will remove the top-most Via header from the message and forward the response to the address given in the next Via header in the list. This procedure can for instance be in Figure 11, in items 5 and 6. In item 6 one via field has been removed. In all the other SIP messages showed in Figure 11 and Figure 12, the scenario is easier since there were no additional responses going across any servers.

6.3.4 SIP addresses

In SIP users are identified by SIP addresses, the so-called SIP URLs. SIP URLs work in the same way as email addresses do. They use the format “sip:user@host”. The user part could be a user name or a telephone number, while the host part can be a hostname or an IP address.

As has already been described briefly earlier, the private IP addresses that are used in an outgoing SIP message must be replaced by the globally routable IP address of the NAT Firewall. This fact makes it is important to recognize which header fields can contain IP addresses and which ones do not. This subject is described further in Sections 9 and 10.

In general IP addresses can occur in all the places, which can hold a SIP URL, i.e. a SIP address. SIP URLs can be found in (1) the Request line, (2) the To field, (3) the From field, (4) the Via field, (5) the Contact filed, (6) the Record-route field, and (7) the Route field. It is also common that the last part of the Call-ID field contains the IP address or hostname of the calling host in order to help ensure uniqueness of the session identification.

Examples of SIP URLs can be seen in Figure 9. They are easily recognized, they all start with "sip:".and the host part is either a domain name or a numeric network address. There are several examples of URLs of this type in Figure 9: **sip:uabfrth@134.138.228.159** and **sip:Fredrik.Thernelius@uab.ericsson.se** are two of them.

6.3.5 Transportation protocol for SIP

SIP is not dependent of the service of any specific transport protocol. In the SIP RFC, TCP and UDP are suggested but it also states that SIP may also be used with protocols such as ATM AAL5 [RFC 1483], IPX, frame relay, or X.25.

6.3.6 SIP Terminology

Location Server: The location server contains the information about user locations.

User Agent: The User Agent (UA) is an application program running at the endpoints of the call, at the users. The UA consists of two parts, the User Agent Client (UAC) and the User Agent Server (UAS). The UAC sends SIP requests on behalf of the user and the UAS listen for responses and notify the user when they arrive.

SIP server: The SIP server is responsible for the users within a domain. It can work in proxy mode or redirect mode. The redirect mode it relays information to caller about the callees' location, see Figure 11. In proxy mode it relays all messages between the caller and the callee, see Figure 12.

SIP Proxy server: See SIP server.

SIP Redirect Server: See SIP server.

Call leg: The 3-tuple of the (1) Call-ID, (2) the From field, and (3) the To field (including any tags) defines a call leg, i.e. identifies of a session between two SIP clients.

6.3.7 SIP message structure

This section gives the main building block for describing SIP messages. Further details have to be read in [RFC2543] or if one is only interested in the grammar then this can be found in [Grammar], appendix 16.2.

The structure and the syntax of SIP messages are, as many IETF developed text based protocols such as HTTP and FTP, based upon the definitions made in the "Standard for ARPA Internet Text Messages"[RFC822]. The specific character set used by SIP is ISO 10646 with UTF-8 encoding [RFC 2044], just as for SDP. [RFC822] bases its syntax on the augmented Backus-Naur Form (ABNF) [RFC2234]. The syntax of augmented BNF is described in appendix 16.3.

The SIP messages are always either a request from a client to a server or a response to a request from a server to a client. With augmented BNF this is written as:

SIP-message = Request | Response

Both the request message and the response message share a common structure, i.e. the generic message structure. The generic message structure is defined as:

generic-message = start-line *message-header CRLF [message-body]

The generic message syntax gives the structure of the entire SIP msg. Please compare this syntax to the sample SIP message provided in Section 6.3. After the first line, the start-line follows several message headers and after the empty line in the middle follows the message body, i.e. the session description.

All parts have their own syntax to follow:

start-line = Request-Line | Status-Line
message-header = (general-header | request-header | response-header | entity-header)

A summary of the different types of headers can be found in Section 6.3.10. All of the message headers follow the same syntax and structure.

message-header = field-name ":" [field-value] CRLF

A request, like the one showed in Section 6.3, have the following main building blocks.

Request = Request-Line
***(general-header | request-header | entity-header)**
CRLF
[message-body]

Request-Line = Method SP Request-URI SP SIP-Version CRLF
Method = "INVITE" | "ACK" | "OPTIONS" | "BYE" | "CANCEL" | "REGISTER"

A summary and description of the request methods follow in the next section.

Like the request the responses is built up similarly:

Response = Status-Line
***(general-header | response-header | entity-header)**
CRLF
[message-body]

Status-Line = SIP-version SP Status-Code SP Reason-Phrase CRLF
Status-Code = Informational | Success | Redirection | Client-Error | Server-Error |
Global-Failure | extension-code

A summary and description of the status codes follow in Section 6.3.9.

6.3.8 Summary of SIP requests

INVITE
The INVITE method indicates that the user or service is being invited to participate in a session. The message body contains a description of the session to which the callee is being invited.

ACK
The ACK request confirms that the client has received a final response to an INVITE request.

OPTIONS
The server is being queried as to its capabilities. A server that believes it can contact the user, such as a user agent where the user is logged in and has been recently active, MAY respond to this request with a capability set.

BYE
The user agent client uses BYE to indicate to the server that it wishes to release the call. A BYE request is forwarded like an INVITE request and MAY be issued by either caller or callee.

CANCEL
The CANCEL request cancels a pending request with the same Call-ID, To, From and CSeq (sequence number only) header field values, but does not affect a completed request. (A request is considered completed if the server has returned a final status response, i.e. a 2xx response)

REGISTER
A client uses the REGISTER method to register the address listed in the To header field with a SIP server.

6.3.9 Summary of SIP response codes

The response codes in SIP are greatly influenced by the response codes given in HTTP/1.1 [RFC2068, RFC2616]. However, not all of them have been appropriate to reuse in SIP. SIP extends there with a new class of codes, called 6xx – Global Failure, see Table 2. The codes that extends HTTP/1.1 have number starting from x80 and above for each class. See Table 3 for a complete list of response codes.

1xx: Informational	request received, continuing to process the request
2xx: Success	the action was successfully received, understood, and accepted
3xx: Redirection	further action needs to be taken in order to complete the request
4xx: Client Error	the request contains bad syntax or cannot be fulfilled at this server
5xx: Server Error	the server failed to fulfill an apparently valid request
6xx: Global Failure	the request cannot be fulfilled at any server

Table 2. SIP Response Codes

1xx	181 Call Is Being Forwarded	182 Queued	
2xx	200 OK		
3xx	300 Multiple Choices	301 Moved Permanently	302 Moved Temporarily
	305 Use Proxy	380 Alternative Service	
4xx	400 Bad Request	401 Unauthorized	402 Payment Required
	403 Forbidden	404 Not Found	405 Method Not Allowed
	406 Not Acceptable	407 Proxy Authentication Required	408 Request Timeout
	409 Conflict	410 Gone	411 Length Required
	413 Request Entity Too Large	414 Request-URI Too Long	415 Unsupported Media Type
	420 Bad Extension	480 Temporarily Unavailable	481 Call Leg/Transaction Does Not Exist
	482 Loop Detected	483 Too Many Hops	484 Address Incomplete
	485 Ambiguous	486 Busy Here	
5xx	500 Server Internal Error	501 Not Implemented	502 Bad Gateway
	503 Service Unavailable	504 Gateway Time-out	505 Version Not Supported
6xx	600 Busy Everywhere	603 Decline	604 Does Not Exist Anywhere
	606 Not Acceptable		

Table 3. Complete list of SIP response codes

6.3.10 Summary of SIP headers

message-header = field-name ":" [field-value] CRLF

General-header	Accept	X	E	Cseq	N	E	Record-Route	X	H
	Accept-Encoding	X	E	Date	X	E	Timestamp	X	E
	Accept-Language	X	E	Encryption	N	E	To	N	E
	Call-ID	N	E	Expires	X	E	Via	N	E
	Contact	X	E	From	N	E			
Entity-header	Content-Encoding	X	E	Content-Length	X	E	Content-Type	X	E
Request-header	Authorization	X	E	Priority	C	E	Response-Key	C	E
	Contact	X	E	Proxy-Authorization	N	H	Subject	C	E
	Hide	N	H	Proxy-Require	N	H	User-Agent	C	E
	Max-Forwards	N	E	Route	X	H			
	Organization	C	H	Require	X	E			
Response-header	Allow	X	E	Server	C	E	WWW-Authenticate	C	E
	Proxy-Authenticate	N	H	Unsupported	X	E			
	Retry-After	C	E	Warning	X	E			

X = May be encrypted E = End-to-End header field
N = Must not be encrypted H= Hop-by-Hop header field
C = Should be encrypted

Table 4. Summary of SIP headers

7 Internet Firewalls

Firewalls are the main protection mechanism when it comes to keeping unwanted traffic away from ones own network. The concept can be easily described, see Figure 13 [Cheswick95]. Everything that is inbound (this is always seen from the internal networks point of view) or outbound will be filtered. Those packets that do not fulfill the rules set will be filtered out and dropped in order to enforce the security policy set in the filters.

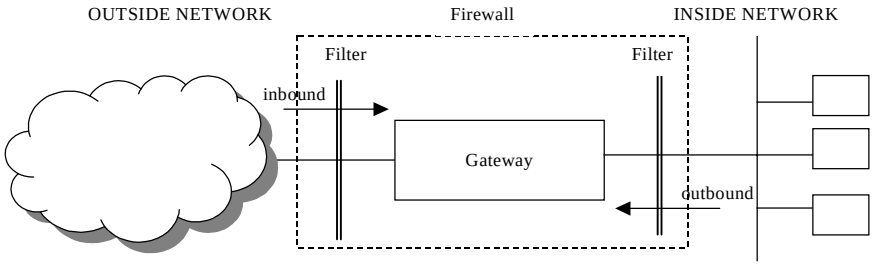

Figure 13. The principles of a firewall

In books like [Cheswick95, Chapman95, and Goncalves00] the subject of firewall architecture is described in great depth. This thesis will not attempt to cover this area other than the basic concepts. The model referred to in this text is the dual-homed i.e., the firewall has two network interfaces, host model, which also happens to match the figure shown above.

7.1 Packet filtering Gateways

Packet filtering gateways comes in two variations, those that do not remember what has happened previous in the session, and those that do. They are called stateless and stateful for obvious reasons. In both stateless and stateful filtering the main tools for filtering operate on the various headers of the IP datagrams, i.e. the IP header, the TCP header and the UDP header. The headers were shown in Section 6.1. The most important information when performing filtering are IP addresses (source and destination) and port numbers (source and destination). Other information such as the SYN and the ACK flag in the TCP header are also extremely important. They are used to

distinguish the start of the connection from an established connection. Figure 15 shows how a TCP connection is set up and then torn down after a connection. A three-way handshake is used to set up a TCP connection between two hosts while four TCP segments are needed to tear down the connection.

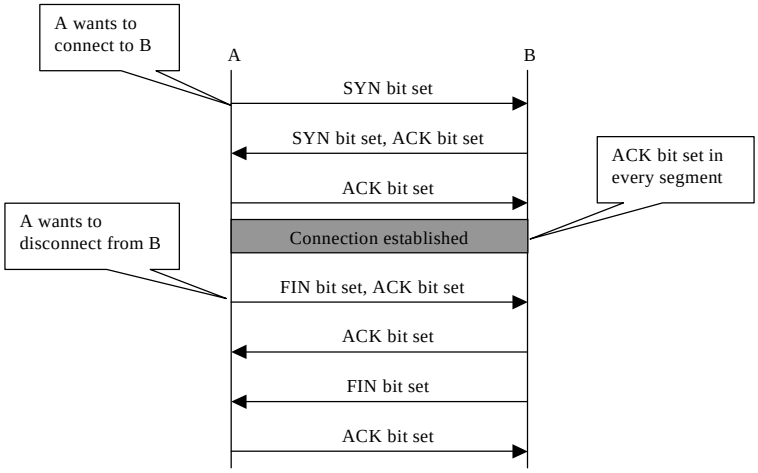

Figure 14. Setup and tear down of a TCP connection

[Course488] describes that stateful filters use information from the application part in the filtering process. This implies that stateful filters can recognize application protocols without having to base its decisions on whether a certain packet is destined for a service using a well-known port or not. Worth noting is that the packet filtering gateways do not have the ability to make changes to content of the application data of the IP datagrams.

The filtering functions, rules, on packet filtering gateways are applied on a per interface and per direction (in or out) basis. The rules also depend on the direction of a certain packet, i.e., if it is going in to the gateway or if it is leaving the gateway. Figure 15 below shows how rules can be written in Cisco IOS and Linux. They are very similar. In Cisco IOS the rules are bundled in Access Control Lists (ACL) [Cisco]. The tool for writing rules in Linux is called ipchains [ipchains].

The figure shows an example where hosts on the internal network, 10.1.1.0, with a 24 bit net mask are allowed to use their web browsers (running an a port number above 1023) to access outside Web servers (running on port 80) and then download content from them.

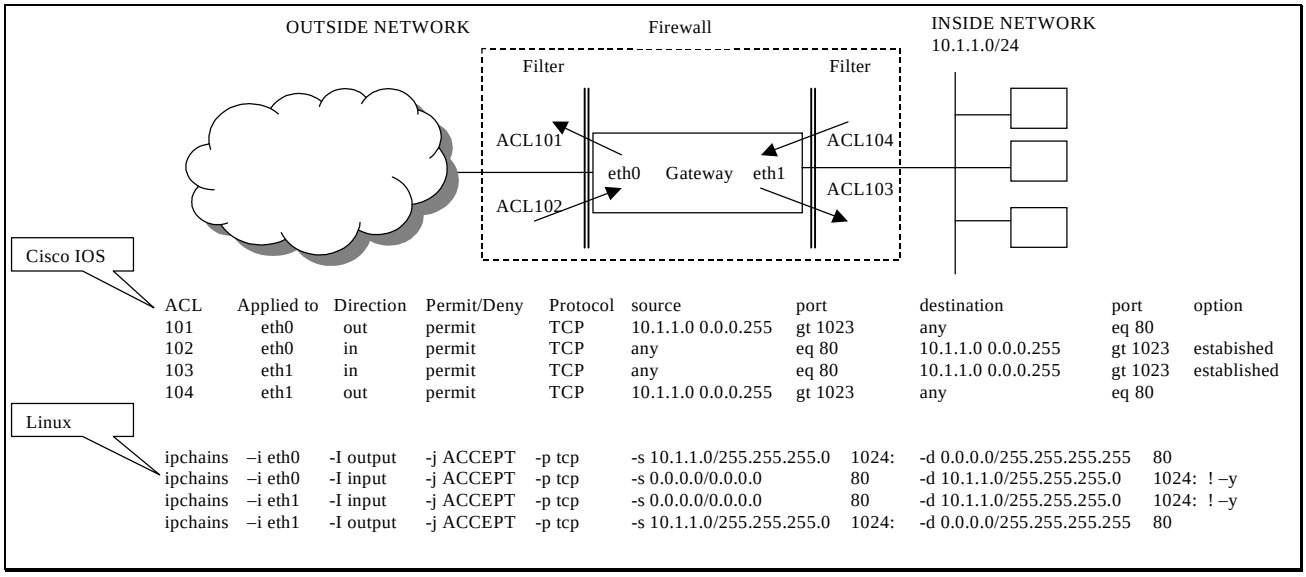

Figure 15. Cisco IOS Access Control Lists and Linux ipchains

The example in the figure shows how rules can be written to let hosts on the internal network 10.1.1.0 with a 24 bit net mask send requests to an outside Web server.

7.2 Circuit-Level gateways

As the name, circuit-level gateway, implies it deals with the virtual circuits that TCP provides (connections are reliable and packets arrive in sequence) and that is why this kind of gateway can only be used for TCP based application protocols, such as Telnet or FTP. Support for UDP currently is being developed.

The circuit level gateway (often referred to as circuit-level proxy) is a generic tool for relaying TCP connections from one side of the firewall to the other, see Figure 16. The internal client makes a connection to a TCP port on the gateway, which then opens a connection to the external server. The gateway does not try to interpret the content of the application part of the TCP segment, it only relays any information received from one side to the other. This procedure has been standardized in [RFC1928]. The SOCKS 5 protocol uses the well-known TCP port 1080. To use the SOCKS service the client software needs to be modified, as described in [Cheswick95], or replaced so that a SOCK-ified client can be used instead.

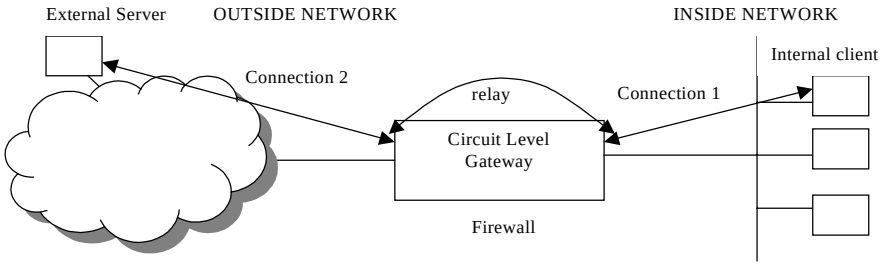

Figure 16. Circuit Level Gateway

The addresses that the clients use on the internal network will not be seen in the IP header of datagrams leaving the network since it will be replaced with the address of the second interface (on the public side). The port number in the TCP header will therefore also be replaced.

7.3 Application Level Gateways

The application-level gateway (ALG) is the most sophisticated kind of firewall gateway. It works in a similar fashion as the circuit level gateway but there are a number of important differences (shown later). Application level gateways are just like the circuit-level gateways often referred to as proxies. On other occasions, in [RFC2663] for example, an ALG is often referred to a program running in cooperation with a firewall performing Network Address Translation (NAT), as described in Section 8. The latter definition is the one that is used in this thesis. Here follows some features of an ALG:

1. The ALG is not generic like the circuit-level gateway, instead it uses special purpose code for each particular application/service it is supporting. Since it understands the application protocol it is relaying it achieves a higher security level.
2. The ALG does not have the shortcoming of only supporting TCP as the circuit-level gateway has. It also supports UDP based protocols, such as TFTP.
3. If the ALG is used together with a NAT (the main concern of this thesis), then the ALG will examine the application data for occurrences of internal addresses and replace them with the address of the firewalls external interface. Section 10 describes an implementation of an ALG for SIP.

ALGs with support for protocols like FTP, DNS, and SNMP are common components in NAT firewalls according to [RFC2663]. We hope soon it will be just as common to also support SIP in this fashion!

8 IP addressing issues

As the number of computers in the world increases, the limitations of the current Internet Protocol, version 4, are apparent. The topic discussed here is the rapid depletion of available IPv4 addresses. This is a fact, even though it seems strange when considering that IPv4 supports 32 bit addresses ($2^{32} \approx 4.3$ billion) and the numbers of computers in the word have not even reached (or has it?) a billion yet. The explanation is given in Section 8.1. Section 8.2 will show that the next version of the Internet Protocol, IPv6, solves this problem by introducing a larger address space. Section 8.3 introduces a special kind of addresses, called private addresses. The private address space is available for every one to use in their own network without having to request them from some authority. The private addresses are very useful, but since many hosts can use the same addresses at the same time, they are not allowed on the Internet. Section 8.4 explains how a Network Address Translator (NAT) can be used when hosts using private addresses want to communicate over the Internet. The use of private addresses and NATs are seen as an intermediate solution of the address depletion until the deployment of IPv6, which is expected to take several years.

8.1 Introduction to addressing with IPv4

In IPv4 the addresses are distributed in five different classes. Class A supports 128 different networks, each with almost 17 million possible host IDs. What organization or company would need that many IP addresses? Class B supports 16 thousand networks with approximately 65000 hosts. This class is very suitable for large corporations and organization. In the next step below, in Class C, the networks only have 8 bits ($2^8 = 256$) for the host part of the IP address, but instead it has a large amount network IDs ($2^{21} \approx 2$ million). The rest of the addresses, Class D and E, cannot be used for addressing specific hosts and are therefore ignored in this discussion. Figure 5 shows what the 20-byte long IPv4 header looks like.

To summarize: Class A has 50% of the number of possible addresses, Class B has a 25 % and Class C has 12.5%. Class A, B and C will thus have 87.5% together.

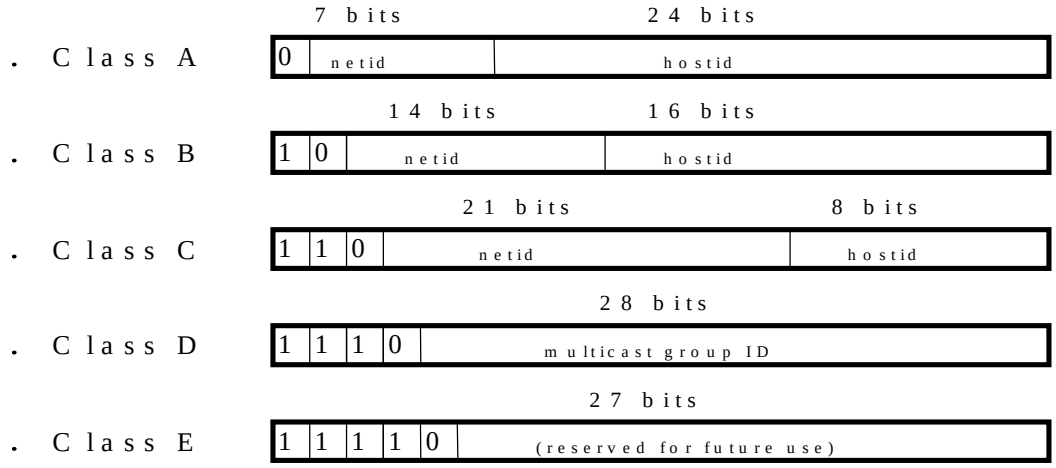

Figure 17. IPv4 Addressing

Class	Range
A	0.0.0.0 to 127.255.255.255
B	128.0.0.0 to 191.255.255.255
C	192.0.0.0 to 243.255.255.255
D	224.0.0.0 to 239.255.255.255
E	240.0.0.0 to 247.255.255.255

Figure 18. IPv4 Address range

Here is an example that shows how a great deal of the address space has been wasted:
Lets say that a fairly large corporation, with about 20 thousand employees and thus with 20 thousand computers all of a sudden needed to connect the computers to the Internet. They can either request a class B address from InterNIC and leave 45000 addresses unused or they can request 80 class C networks. As using a lot of class C networks creates a lot of administrative problems the natural choice would be to go with a Class B network. This was the usual way to do it in the *old days*, now days this would never be possible.

8.2 Introduction to addressing in IPv6

In the next version of the Internet Protocol, IPv6, the problem with the depletion of the Internet addresses has been dealt with in a grand fashion. The number of bits has been extended from 32 to 128, or if expressed in bytes, from 4 to 16 bytes. It is almost like comparing the number of stars in our galaxy, a very large number, to the number of stars in the entire Universe, an immensely large number. In [Goncalves00] the number of addresses per square meter of our planet (Earth) is calculated to be 665,570,793,348,866,943,898,599. In the calculation it is assumed that the surface of the Earth is 511,263,971,197,990 square meters.

An example of what an IPv6 address can look like in text representation is:

789A:0:0:0:3:4567:89AB:CDEF

The ABNF description of IPv6 text representation is shown in Figure 19 below. More information about the address architecture for IPv6 can be read in [RFC 2373].

```
IPv6address = hexpart [ ":" IPv4address ]
IPv4address = 1*3DIGIT "." 1*3DIGIT "." 1*3DIGIT "." 1*3DIGIT

IPv6prefix  = hexpart "/" 1*2DIGIT

hexpart = hexseq | hexseq "::" [ hexseq ] | "::" [ hexseq ]
hexseq  = hex4 *( ":" hex4)
hex4    = 1*4HEXDIG
```

Figure 19. ABNF for text representation of IPv6 addresses

As can be seen in the ABNF syntax, the IP version 6 has support for the use of IP version 4 addresses and this is good since the IPv6 and IPv4 are likely to co-exist during a transition period. Figure 20 shows the 40 byte long IPv6 header.

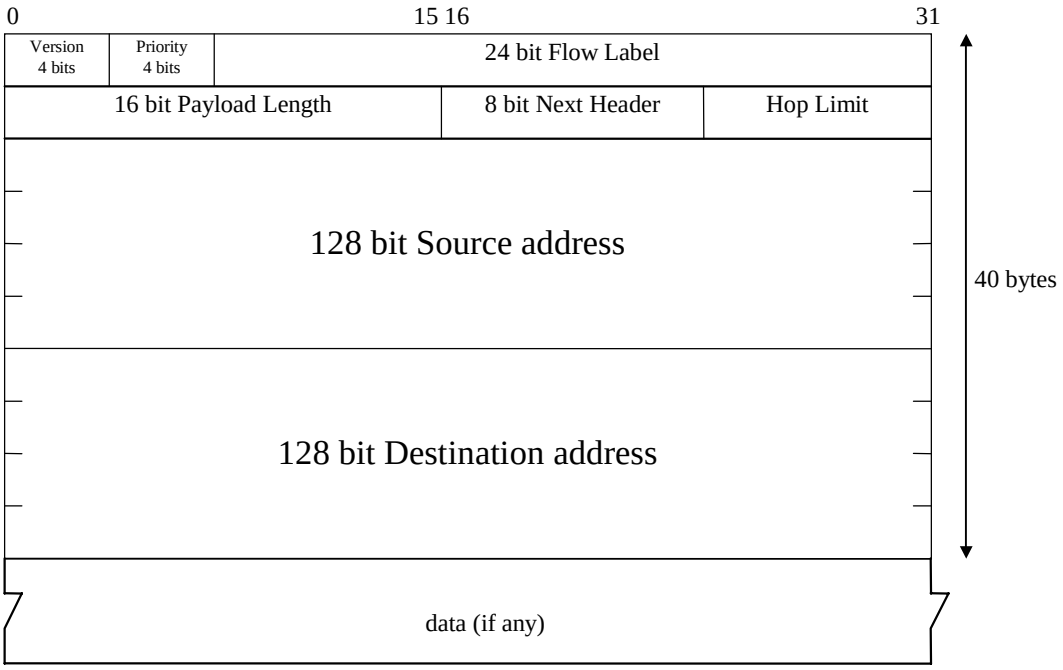

Figure 20. The IPv6 header

IP version 6 contains more features than just an extension of the address space. Authentication and Privacy capabilities are also included [RFC2401].

8.3 Private Addresses

The private address space defined in [RFC1918] enables network administrators to use certain special addresses in their own network without having to request them from any authority. In Figure 21 below the range of these special addresses are showed.

```
10.0.0.0      -   10.255.255.255      One class A network
172.16.0.0    -   172.31.255.255      16 continuous  class B networks
192.168.0.0   -   192.168.255.255     256 continuous  class C networks
```

Figure 21. The private address space

The private addresses are banned on the Internet to avoid ambiguity problems. There would be no way to route IP datagrams on the Internet to the right destination if two or more receiving hosts have the same IP address.

But why use private addresses if they can lead to this kind of inconsistencies? The reason is that the private addresses solve the big problem mentioned in the beginning of Section 8, i.e. the rapid depletion of the number of available IP addresses. The private addresses also enable network administrators to be more flexible in their design of networks, as more operational and administratively convenient addressing schemes can be used.

It is actually possible to use any kind of addresses, private or global, for private networks as long as it does leave the private network. In addition, packets leaving the private network must not contain those addresses. If your ISP catches you with sending out addresses not assigned to you they have the option to disconnect you in immediately. Making a mistake and sending out datagrams with private addresses could be forgivable, but sending out with someone else's addresses is an offence that is much worse.

In order to not "leak" addresses, it is necessary for the IP and the UDP/TCP- headers, see Figure 3, 4, and 5, of any outgoing packet to be changed, as it passes out onto the Internet and then changed back as responses are received from the Internet and passed on to the private LAN. This is how Network Address Translation (NAT) comes into the picture.

8.4 NAT

NAT [RFC1631] was originally thought of as a short-term solution until a long-term solution to the problem of the depletion addresses was developed. The long-term solution was expected to be a proposal for a new Internet protocol with larger addresses and we now know that the next generation of IP will be IPv6, see Section 8.2 above.

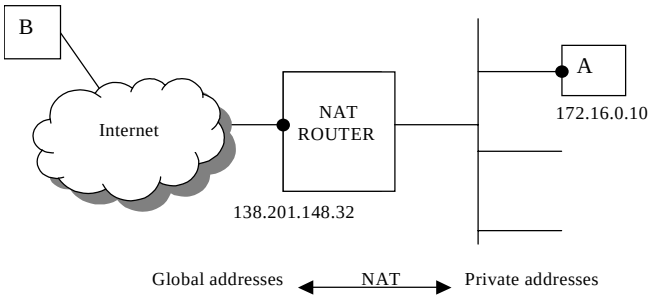

Figure 22. Network Address Translation

Figure 20 shows the idea of address translation. When host A, who is using the private address 172.16.0.10, communicates with host B, who is assigned a global address on by his ISP, the IP addresses in the IP header must be exchanged from 172.16.0.10 to 138.201.148.32 as packets move across the NAT router. After the exchange it will look to host B as if host A really has 138.201.148.32 and it will thus send any replies to that address. Host B, could in fact be behind a NAT. There is no way to tell.

It is worth noting that the NAT router does the translation transparently. This means that the NAT router will swap the IP address in the IP header without any notification to either of the hosts.

For every session between internal and external hosts the NAT router will have to set up an entry in a translation table. This is done to be able to correctly map packets to the right host on the other side, i.e. for the translation to work properly. Such a table could look like the one in Figure 23.

NAT Table	
Internal IP	External IP
172.16.0.10 172.16.0.11	138.201.148.32 138.201.148.151

Figure 23. NAT Table

The Internal IP address is the private IP address of the internal host and the external IP is a public IP address on the NAT router, which is assigned to map the private address. That is, in the outgoing packets the NAT router will exchange the address in the "source" part, 172.16.0.10, of the IP header to the corresponding external IP address, 138.201.148.32. In the same way the "destination" part of incoming messages be exchanged and sent out on the internal interface if it matches any of the external addresses in the table, thus 138.201.148.32 will be exchanged for 172.16.0.10. Figure 24 shows the procedure.

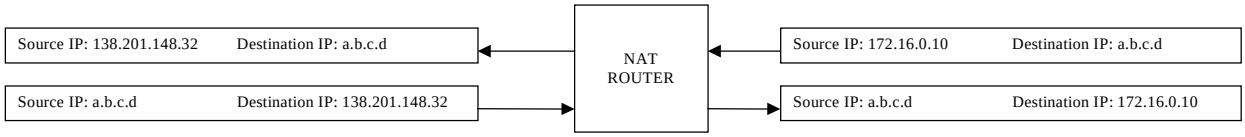

Figure 24. Address translation in practice

This seems to be very simple, and it is, but there are a few problems that can occur. Lets use SIP to show them. The first problem occurs if a session is not initiated from the inside, because then there will be no easy way to set up the translation table. The address to the right internal host must be found somehow. A second problem is that some application protocols like SIP for instance (FTP is an other), put IP addresses and port numbers inside the application data in the IP packets. This information must also be processed somehow. In SIP this information tells the other party of a call where to send the media. To solve these problems, an Application Level Gateway (ALG) is needed to analyze these messages.

I will not try to define general solutions to these problems for all protocols. I will focus only on SIP. In order to solve the first problem the internal clients must be registered at the ALG. This will give the ALG the internal address of the client, whose name was found in the To field. The second problem, which only applies to outbound messages, is also solved by having the ALG analyzing the message and exchange the necessary parts. See Section 10, for a more detailed discussion on this subject.

8.4.1 Flavors of NAT

NAT comes in several flavors. They are called static NAT, dynamic NAT, and NAPT (Network Address and Port Translation)[2]. They will all be discussed further, but the main focus here will be on NAPT. There are actually even more flavors available [RFC2663], but they will not be considered here.

8.4.2 Static NAT

Static NAT requires the same number of globally unique IP addresses as there are hosts, in the private environment, that want to be able to connect to outside networks (all of them most likely). As the name suggests, the mapping between local addresses and global addresses is intended to stay the same for a long period of time.

[2] In Linux this is called masquerading

The mapping between internal IP address and external IP address is set up manually by the network administrator or perhaps this work be automated by using protocols like DHCP [RFC2131]. By using DHCP the host would be assigned an IP addresses by the DHCP server at boot time.

By using Static NAT the problem with sessions initiated from the outside with inbound requests can be avoided. This is because of the one to one mapping in the translation table, i.e. when a new session is started from the outside the NAT router will always know which internal IP address to use.

8.4.3 Dynamic NAT

In dynamic NAT the NAT router collects the external, i.e. global or public, IP addresses into an IP address pool; with this type of NAT it is not necessary to have the same number of external IP addresses as there are hosts wanting to connect to the outside, see Figure 25. In dynamic NAT it is assumed that all hosts do not want to connect to the outside at the same time and thus it is possible to reduce the number of external addresses that are needed.

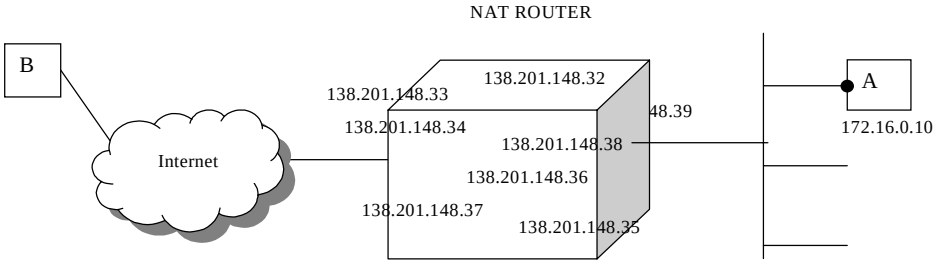

Figure 25. A NAT router with a pool of addresses

When a host wants to connect to the outside an external IP address is allocated from pool of addresses managed by the NAT. For each one allocated it reduces the number available. This can continue until all the IP addresses are allocated. At which point the next host that tries to send outbound messages will not be able to do so until a connection is closed down and the used IP address is returned to the pool of available IP addresses.

For TCP it is possible to tell if the packet closes down the connection by looking in the TCP header, both the FIN bit (orderly release) and RST bit (abortive release) are considered [RFC2663]. UDP creates a problem since it does not have these features. This means that there is no way to look at only the UDP header and tell if a certain datagram will release the connection and thus give the back the address to the address pool.

One way to handle the problem of when NAT can reclaim the address is to use timeouts. When a mapping has not been accessed for a certain period of time the mapping is said to be dead, and the NAT router reclaims the allocated address. This scheme is even used with TCP connections since connections can be cut without any FIN message has been sent. [RFC2663] suggests that a TCP session may be terminated if it has not been used for 24 hours. For non-TCP a timeout of a few minutes is suggested.

8.4.4 NAPT – Network Address and Port Translation

NAPT is a special case of dynamic NAT and this flavor of NAT is the one most commonly used. What makes this solution popular is that many hosts can share one, or perhaps a few, external IP addresses. Instead NAPT uses port numbers as the basics for the address translation. In this scheme the number of connections per IP address is limited by the number of ports available, that is 2^{16} = 65536. This is only a theoretical number since many ports are assigned to specific services, see [RFC1700] for a complete list (not updated since 1994) of assigned ports. It is also assumed that when a port is used for a translation, then both the TCP port and the UDP port are marked as used, for simplicity reasons.

Figure 26 shows how NAPT works. The source IP and source port of host A are exchanged on outgoing packets, compare item 1 and item 2 of the figure. When the user on host B replies to the request, he sends 3. The router will recognize the information contained in the headers, translate the information, see part 4, and send it off to host A.

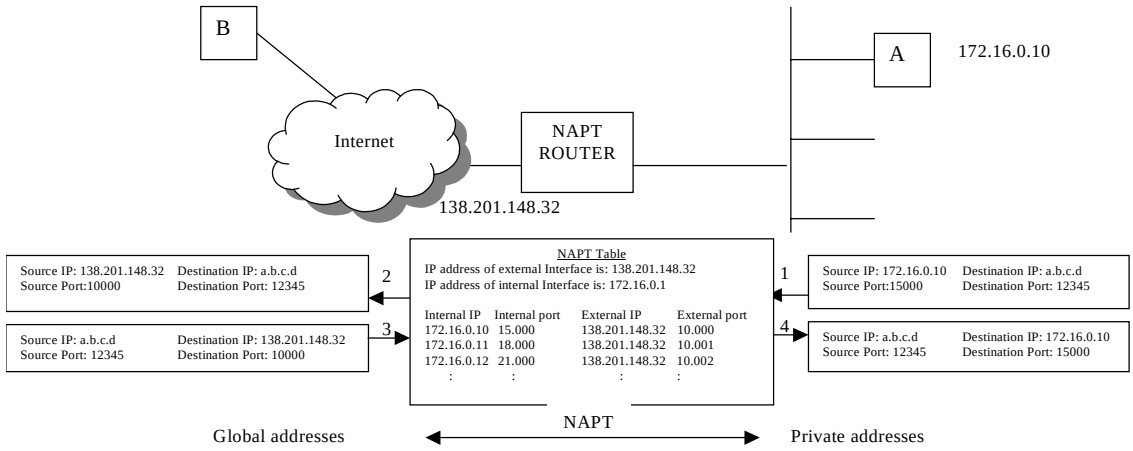

Figure 26. Network Address and Port Translation

9 Introduction to Security in SIP

SIP includes a wide variety of security features, such as encryption[3] and authentication[4]. SIP also has one more feature and that is the possibility to hide the route which a certain message has taken, as this is otherwise recorded at intermediate proxies as they add their address in the Via list. Inserting the appropriate header fields activates the features, see the following subsections for detailed information.

For more detailed information than the one given below, see [RFC2543].

9.1 Encryption

Encryption can take place either end-to-end between user agents, or hop-by-hop between any two SIP entities.

Hop-by-hop encryption encrypts the whole SIP message and is supposed to work on the transport level or the network layer. The algorithm used is not specified, but IPsec [RFC2401] is suggested.

When using end-to-end encryption between user agents some basic rules are necessary to follow. They are:

1. All header fields must not be encrypted since they are needed to be understood by intermediate SIP entities, see Table 4.
2. All header fields that are not encrypted must precede those that are encrypted, see Figure 27.
3. It is not necessary to encrypt any SIP headers, see Figure 28.
4. An encryption header must be inserted to indicate the encryption mechanism
5. The responses to encrypted requests should be encrypted with a key given in the Response-key header field in the request. If none is given then the answer should be sent unencrypted.
6. The headers that were encrypted in the request should also be encrypted in the response.

[3] Encryption enables the possibility to ensure privacy, i.e. no one except the intended receiver will be able to read the message
[4] Authentication is the concept of being sure of whom you are talking to.

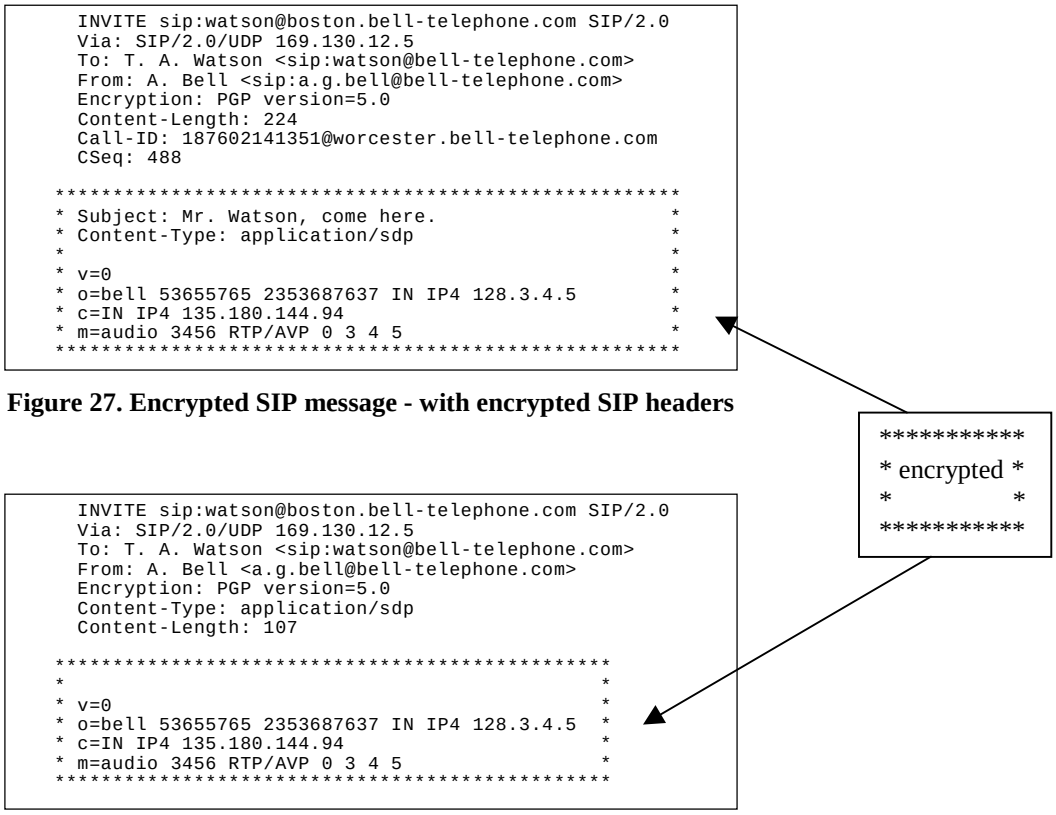

Figure 27. Encrypted SIP message - with encrypted SIP headers

Figure 28. Encrypted SIP message - with no encrypted SIP headers

9.2 Authentication

When using authentication the user digitally signs the message that is about to be sent. The signature extends over the whole SIP message, from the first line, i.e. the request-line or the status-line, to the message body, but not all SIP header fields are included in the signature. Those header fields that are not included are those that changes between hops, such as the Via field for instance. If it were included then the message integrity calculation on the receiving side would produce an error if any intermediate proxies had entered (which they must do!) their address in the Via list. Figure 29 below shows how this works. At the receiving side those headers that are not included in the signature must be removed before the integrity calculation can be performed.

```
INVITE sip:watson@boston.bell-telephone.com SIP/2.0
Via: SIP/2.0/UDP 169.130.12.5
Authorization: PGP version=5.0, signature=...
From: A. Bell <sip:a.g.bell@bell-telephone.com>
To: T. A. Watson <sip:watson@bell-telephone.com>
Call-ID: 187602141351@worcester.bell-telephone.com
Subject: Mr. Watson, come here.
Content-Type: application/sdp
Content-Length: ...

v=0
o=bell 53655765 2353687637 IN IP4 128.3.4.5
c=IN IP4 135.180.144.94
m=audio 3456 RTP/AVP 0 3 4 5
```

Bold characters are unsigned

```
INVITESIP/2.0From: A. Bell <sip:a.g.bell@bell-telephone.com>
To: T. A. Watson <sip:watson@bell-telephone.com>
Call-ID: 187602141351@worcester.bell-telephone.com
Subject: Mr. Watson, come here.
Content-Type: application/sdp
Content-Length: ...

v=0
o=bell 53655765 2353687637 IN IP4 128.3.4.5
c=IN IP4 135.180.144.94
m=audio 3456 RTP/AVP 0 3 4 5
```

Figure 29. Upper - the SIP request that is to be sent. Lower - the signed parts of the message

Messages that are not authenticated may be challenged by any SIP entity downstream[5] from the user. The challenge could either be included in a 401 (Unauthorized) or a 407 (Proxy authorization required) response message, which may only be used by proxies.

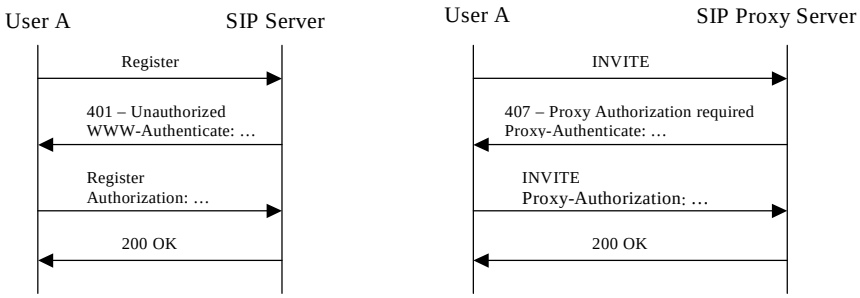

Figure 30. Authentication

9.3 Hide Route

The hide route feature is sometimes called Via field encryption, since that is what it does. When the feature is enabled intermediate proxies encrypts either the entire Via list or they encrypt only the top-most entry in the Via list, depending on the option entered by the client, hide-route or hide-hop.

The reason for having this feature is that the information in the Via list contains information about traffic patterns, which eavesdroppers might find very interesting.

10 Implementation of an Application Level Gateway for SIP

This section will not only give detailed information about SIP, SDP, and NAT and their interaction, but also some basic suggestions on how a SIP ALG can be implemented. Almost everything described here has been implemented as a part of or in connection with this Master's thesis project.

The NAT will give the SIP ALG every IP packet that uses the default SIP port, 5060, in the destination address, no matter if the packet comes from the private network side or the Internet side. With this scheme a SIP client cannot

[5] Downstream is the direction from the client issuing the request to the intended recipient and upstream is the opposite way.

reach another SIP client unless it uses the default SIP port, this eliminates the possibility of running several SIP clients on a number of different ports on a host, but it should not limit the possibility of reaching the intended client entirely.

In the case of incoming SIP INVITE messages from external clients intended for internal clients it is clear that some kind of location service must be available in the SIP ALG in order to direct the message to the correct host on the inside. Implementing a simple SIP registrar server together with the SIP ALG solves this problem. The internal clients would have to register at the registrar server in order for the server to get the clients internal IP address. Unfortunately this scheme has not been tested in practice, because of the lack of time for the implementation Instead, the SIP ALG was coded to always use the same IP address and NAT mapping for the single internal client I had.

For outgoing calls it is not necessary for the internal client to be registered at the server.

10.1 What is the SIP ALG supposed to achieve?

The SIP ALG will have to achieve two main objectives in order to set up an IP Telephony session between clients on the internal network and clients on the Internet. The first is to remove any occurrences of the internal clients private IP address in the outgoing SIP messages and replace it with the IP address that the SIP ALG has been configured to use. This address will be the one used by the external interface in the firewall. This procedure will naturally have to be reversed when SIP messages (requests and responses) for the same call leg are coming in, any occurrences of the firewall's IP address will have to be replaced with the internal client's private IP address again.

The second objective for the SIP ALG is to take care of IP addresses in the SIP message body, i.e. the session description. SIP can include any type of information in the message body, but for the purposes of this thesis it is assumed that only SDP is used for conveying the session description. Anything other than SDP in the SIP body would generate a parse error in the SDP parser and cause the ALG to drop the SIP message. In the SDP the clients insert the IP address that they want to receive the RTP data streams on in at least one field, the connection-field, denoted by a "c". It can also happen that the originator-field, denoted by an "o", contains this address. Additionally, the client inserts the port number(s) that the RTP data stream(s) will be directed to in the media-fields, denoted by an "m". Both the IP address and the port number(s) must be replaced by the ALG in outbound SIP messages. Examples of the fields mentioned can be seen in Figure 31 below.

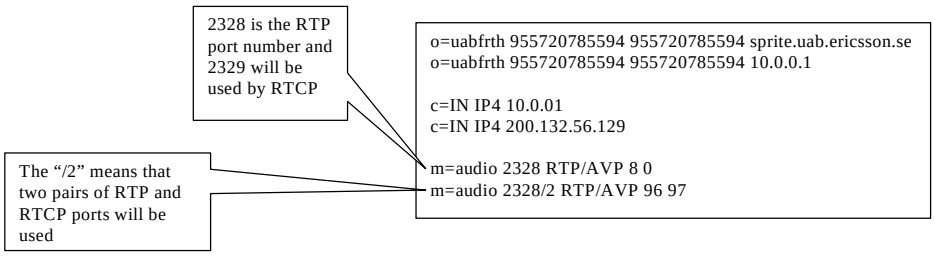

Figure 31. Examples of origin-, connection-, and media-fields

The replacement of the IP address in SDP follows the pattern used for the SIP part of the message. For outbound SIP messages containing a session description the ALG will have to replace the IP address of the originating host with that of the firewall, and reversing the procedure for inbound SIP messages, as mentioned earlier. The SDP of inbound SIP messages will not have to be touched by ALG since the data contained there only applies to the host which sent the message.

The procedure for replacing the port numbers in the media field is more complicated than just replacing them right away. The ALG first has to find out what they will be replaced with. This is done by creating a NAT mapping for the expected incoming data. As the observant reader has noticed, the RTP data is always associated with RTCP at the next higher port number. This means that not only one, but two mappings have to be created. One for incoming RTP data and one for incoming RTCP data. Again, not even this is the whole truth. In the last line of Figure 31 the port number is followed by "/2" and this indicates that the client wishes to use media with two layers, e.g. MPEG-2 [RFC2343]. Each layer uses a pair of RTP and RTCP ports and this example gives a total of four ports. For this

scheme to work these ports must be consecutive. When the call is made to the NAT to create a mapping for this media it would have to do a search to find four consecutive ports and return the first of these ports and insert it into the media field. If more than one media field exists, several of these mappings need to be created.

If NAT is unable to find enough consecutive ports, possibly because of fragmentation of the ports space managed by NAT or maybe because all ports have been allocated already then a SIP message should be generated telling the client that the call is not possible to connect at the moment and to try again later. Two error messages that could be used are 486 – Busy Here and 600 – Busy Everywhere. It does not feel right that the ALG whose purpose is to replace IP addresses and ports inside the message will also take an active part in the SIP signaling by generating SIP messages, but no other way has been found to relay this information back to the host sending the SIP message.

After the ALG has accomplished these two objectives it will also have to manipulate the IP header and UDP header of the IP packet, which contained the SIP message. For outbound messages the address of the SIP ALG should be entered as source address and source port should be set to be the default SIP port, 5060. Before the IP packet can be sent off the new length of the packet needs to be entered and checksums needs to be recalculated. For incoming messages the destination address and port needs to be altered to be the address and port of the internal SIP client. The updates of the length and the check sum fields work in the same way as for the outbound messages.

When the SIP session is terminated, the “holes” in the firewall need to be closed. This is normally done after the ALG intercepts a BYE message from either of the clients. The implementation done in this Master’s Thesis project counts on that BYE messages always being sent upon termination of the session. This is not a realistic situation though. Messages might be lost on the Internet and the SIP clients may be terminated without sending the BYE message. A draft [SessionTimer], recently submitted to the IETF, proposes a solution to solve the problem with lost BYE messages.

10.2 What happens in the ALG?

The work of the ALG is started when the NAT gives the IP packet with the SIP message to the ALG. In order to work properly the ALG also receives information about which network interface the message came from.

To begin with, the SIP ALG parses the SIP part of the message to make sure that no illegitimate packets are let through. If the first part is OK, then the SDP part, if it exists, is parsed for the same reason. If the message does not pass the both of parsers, then the SIP message is silently dropped. One cannot be forgiving when enforcing a security policy!

If this is an INVITE for a new session then a lot of state information is collected from the message. It is the headers that constitute the call leg that are kept for later use. They are needed for finding out if other SIP messages belong to this call leg or not. The To-, From-, and Call-ID field of this first INVITE message must also be compared against other active SIP sessions before it can be concluded that it really is an INVITE for a new session.

During the parsing of the message the fields containing IP addresses were found, if there are any. They are recorded in a linked list, where each item in the list has a pointer to the first character of the IP address that had been found. This ALG implementation works in a very crude way when it comes to the replacement of IP addresses in outbound and inbound messages. What it did for outbound messages was only to find out which items in the list that had the same IP address as the internal host and then replace that address with the address configured for the ALG. If the address had a port associated with it, then the default SIP port, 5060, replaces it. The ALG reversed this replacement as SIP messages came in. This was all very neat and very simple.

Before any enthusiastic reader starts implementing his/her own SIP ALG it must be said that the above solution does not always work. The reason for this is that the ALG is not context aware when replacing the addresses. It is important for the ALG to know not only from which side (external/internal) that first initiated the call, but also if the SIP is a response or a request. An example will show why it is important for the ALG to be context aware.

Example 1. Context awareness

The example given here is an extension to the example given in [SIP, Section 6.40]. What has been added is the port number in the received parameter. This port number must be there if the NAT really is a NAPT, otherwise it will not be able to match the incoming IP packet to any mapping. In the figure below only the header fields relevant to this example have been outlined.

The scenario here is that Bob wants to reach Adam at his home. On their separate networks they happen to use the same private addresses (10.0.0.1) for their respective host machine.

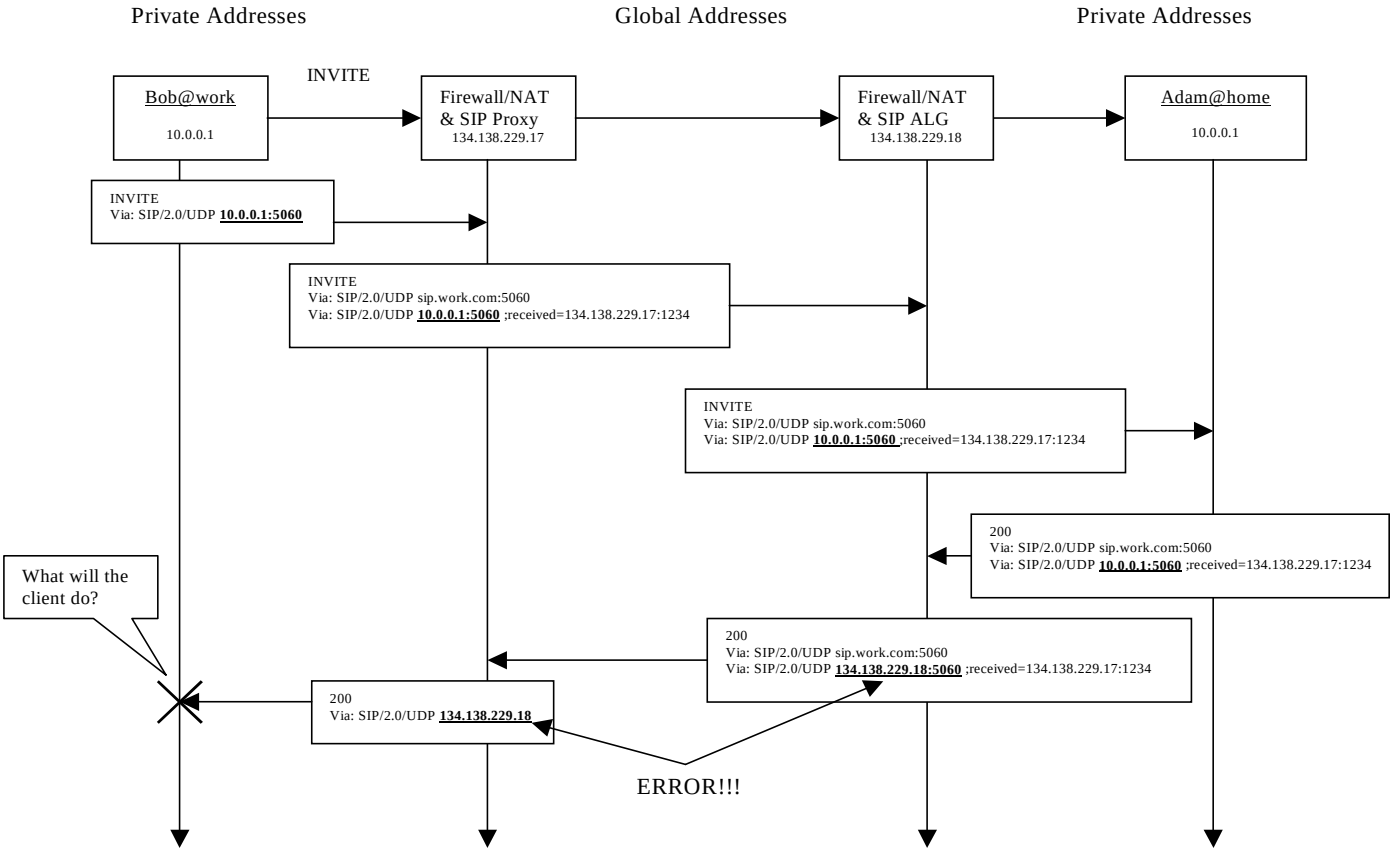

Figure 32. Receiver tagged Via field

When the INVITE from Bob reaches the SIP proxy, the IP packet containing the SIP message has already traversed the NAT, which has changed the source address of the IP packet to be the globally routable address of the firewall, 134.138.229.17. The proxy adds it own address in the top-most Via field, as it should. The proxy has the responsibility to make sure that the address given in the Via header matches the source address in the IP header, because it needs to be able to route the responses in the correct way. The proxy therefore adds a received parameter to the Via field received from Bob.

The ALG validates an incoming INVITE, sees that it is an INVITE for a new session, sets up the necessary holes in the firewall and passes it on to the internal host. In the response generated by the internal user the ALG looks for the host's IP address within the SIP message. It will now find the Via field with the address it is supposed to look for and change this to the address configured to the ALG, in this case 134.138.229.18. At the proxy the received parameter and the top-most Via field are removed and then the message will be sent to the destination address that was given in the received parameter. The NAT will get a hold get a hold of the message and forward it to the origination client, Bob. Bob will now receive a SIP message with a Via field that does not contain his own address. What happens now is up to Bob's software. Since the software knows it works as a client and not as a server that forewords SIP messages it might chose to ignore the Via field and settle for the call leg parameters, since the To-, From-, and the Call-ID header field match and thus accept the response given.

The solution to this problem with IP address being exchanged when they really should not is to add more context awareness in the SIP ALG. For the example above it would only have required a rule saying: **"if this is an outbound SIP message and it is a response then IP addresses in the Via fields should not be altered"**.

The table below shows how the ALG should behave in different situations. Notice that for several of the header fields it is clear that the ALG must know more information than just if the message came from the outside or the inside. It must also know which client who initiated the call and if the message is a request or a response. Notice also the Via field have the rule discussed above.

	Fields	Initiated from inside	Initiated from outside
Inbound request	**To**	don't touch	don't touch
	From	replace the ALG address with the local address	don't touch
	Call-ID	replace the ALG address with the local address	don't touch
	Via	don't touch	don't touch
	Request-URI	don't touch	don't touch
	Contact	don't touch	don't touch
	Record-route	don't touch	don't touch
	Route (I)	(V)	(V)
Inbound response	**To**	don't touch	don't touch
	From	replace the ALG address with the local address	don't touch
	Call-ID	replace the ALG address with the local address	don't touch
	Via	replace the ALG address with the local address	replace the ALG address with the local address
	Request-URI	(II)	(II)
	Contact	replace the ALG address with the local address	replace the ALG address with the local address
	Record-route	don't touch (VII)	don't touch (VII)
	Route (I)	(II)	(II)
Outbound request	**To**	don't touch	don't touch
	From	replace the local address with the ALG address	don't touch
	Call-ID	replace the local address with the ALG address	don't touch
	Via	replace the local address with the ALG address	replace the local address with the ALG address
	Request-URI	replace the local address with the ALG address	replace the local address with the ALG address
	Contact	replace the local address with the ALG address	replace the local address with the ALG address
	Record-route	(III),(IV)	(III),(IV)
	Route (I)	don't touch (VII)	don't touch (VII)
Outbound response	**To**	don't touch	don't touch
	From	replace the local address with the ALG address	don't touch
	Call-ID	replace the local address with the ALG address	don't touch
	Via	don't touch	don't touch
	Request-URI	(II)	(II)
	Contact (V)	replace the local address with the ALG address	replace the local address with the ALG address
	Record-route	don't touch (VI)	don't touch (VI)
	Route (I)	(II)	(II)

Table 5. Summary of ALG actions

(I) The route-header contains the addresses of the SIP equipment that a request will need to pass through. It will not contain the address of the client sending the request.
(II) Responses do not have this field
(III) Record-route cannot be used in a request where a route header is.
(IV) The internal clients will not add a record-route header to any requests. Only proxies can insert this field. There will be none on the inside of the ALG
(V) A SIP client will never see a route field. If there was one earlier it is removed by earlier proxies.
(VI) The internal client will not add anything to the record route header, only reverse the order of the entries.
(VII) The route field will not contain any local addresses. The internal client will take the entries of the previous record-route header, reverse the order, and last it enters the contact-field value, i.e. the address of the external client[6].

[6] It could include a route field if there is a hierarchy (at least one) of SIP proxies on the private network. This thesis does not consider this topology.

This does not solve all the problems that can occur, unfortunately some still remain. For instance, what should the SIP ALG do with a SIP URL looking like: ”sip:uabfrth@sprite.uab.ericsson.se:5060”? Here we have a combination of a hostname and a port number. Since this implementation of a SIP ALG does not have the capabilities to resolve the hostname to an IP address (this feature would of course a must in a commercial version. One cannot have a policy of not reveling the private addresses if one goes ahead and gives the hostnames away. This would be a strange inconsistency if it was done like this!) it will not be able to see if the hostname and the IP address really point to the same logical entity or not. The only possible thing to do in this case is to not do any changes at all, even if the port number is the right one. Some context awareness can of course result in that the SIP URL was in a field that did not need to be processed.

The SDP part of the SIP messages raises a few issues concerning the SIP protocol. SIP says that it is OK not to have a session description in the INVITE and instead include it in the ACK (if one wants to receive anything at all). SIP also states that if a session description was included in the INVITE it is OK to change it completely by the time the ACK is sent. It is also allowed to leave the message body empty in the ACK, indicating that the session description session sent in the INVITE was accepted.

This many degrees of freedom make it hard for the ALG to open the right holes in the firewall and to decide at what time these holes should be opened to the passing media streams. Given the circumstances is seems like the solution to the latter is to wait with opening the holes until the ACK has gone through. The solution to the first part is to introduce the concept of reserved ports. This means that the ALG can allocate ports for the media, see Section 10.1, but leave them in an inactive state (drop everything that is received on them). These ports are now said to be reserved by the SIP ALG. The reservation is done if the INVITE contained a session description. When the ACK is received in the ALG and the media description either matches the one in the INVITE or if the message body is empty, then it is OK to change the state of the reserved ports to active. If new session description is entered in the ACK then the reserved ports from the INVITE must be dropped and new ones need to be reserved instead. The newly reserved ports need to be put in active state immediately. If the ACK instead contained a changed session description compared to the one in the INVITE then those reserved ports that are not being used need to be scratched and new ports need to reserved and opened for the new media.

So far we have only looked at sessions with just one client at each end, which has been sort of straight forward to handle in the SIP ALG. Now we will have to face the fact that one single INVITE can result in at lot of responses, each with its own media description that must be processed in the ALG. It is the concept of “forking proxies”, mentioned in Section 6.3.6, which we will have to deal with. When an INVITE is received at a forking proxy then if the callee is registered on several hosts it will forward the INVITE to all the places the recipient is registered on.

Example 2. Forking proxy

Lets say that Bob is registered at three locations at this time, his home, his work and at his psychologists office (assuming that during his session at the schrink he is allowed to receive calls!), at this particular time. Lets also say that right now he is under hypnosis at the schrink when Bob’s friend Adam calls to find out how he’s doing. All three phones will start ringing, thanks to the forking proxy, at approximately the same time. At home Bob’s wife answers, at work Bob’s boss happen to pass Adam’s desk wondering where Bob is and being a nice guy he answers the phone for him, and finally at the psychologist Bob’s schrink answers the phone even though it says Bob in the display since Bob is currently under hypnosis. As these nice people pick up the phone a tagged 200 OK message will be sent back to the caller, each containing a session description. What is now going to happen at the SIP ALG? The easy way out would have been to only accept the first response and throw away the rest. This is however not the path chosen here. Responses are allowed during a period of 32 seconds [RFC2543 – Section 10.1.2] and they have to be taken care of. As the responses pour in to the SIP ALG, each of them will have the To field tagged with an identifier in order to separate them into distinct call legs. According to this definition we now have three call legs where media can flow. The following three figures show how the SIP signaling works in this case. Due to the human interaction it is not certain that the messages occur in exactly this order.

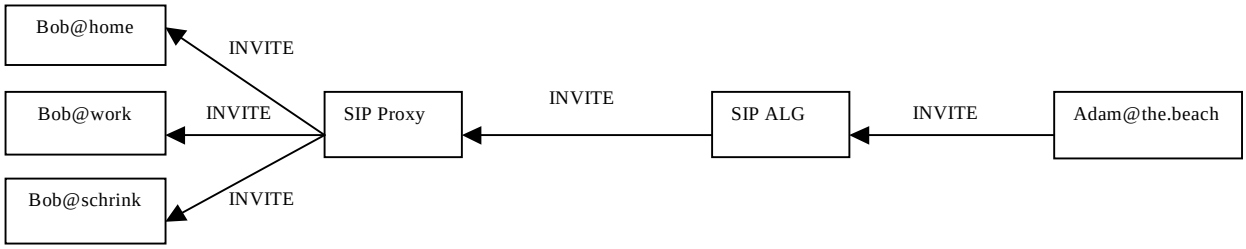

Figure 33. Forking Proxy - INVITE message

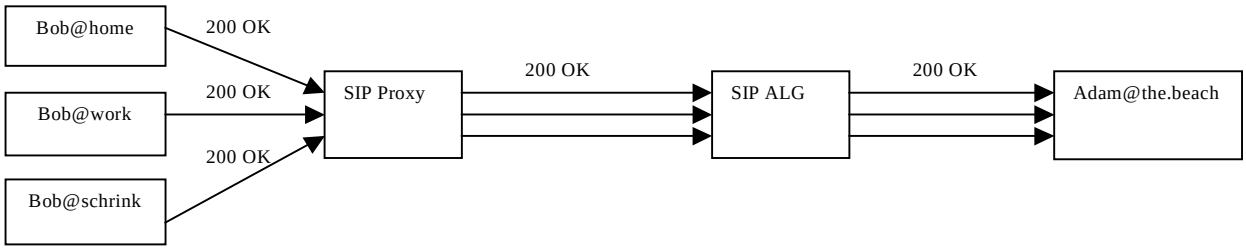

Figure 34. Forking Proxy - 200 OK message

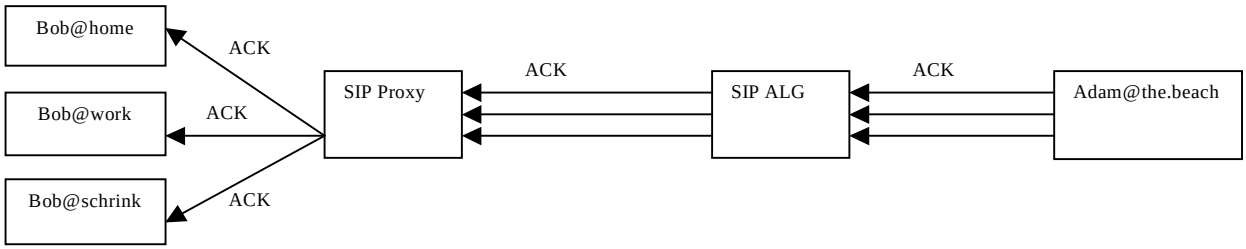

Figure 35. Forking Proxy - ACK message

The SIP ALG must in order to let the media flow from the caller to the callees create NAT mappings for all three responses. The audio sessions set up will work like this: all three callees will hear what Adam says and Adam will hear will hear what the callees say, but the callees will not be able hear or talk to each other. I think all participants in this session will be very confused. It must be noted that this is likely scenario since people want to be reached wherever they are.

This forking proxy scenario can easily become even more complex since SIP allows for changing the session description in the ACK. Adam might decide to use a different codec for each of the answering SIP clients. I don't know how on earth this scenario would be done in practice, but since SIP allows it, it better be supported. The amount of state information that has to be kept in the SIP ALG will of course increase with the number of participants and the number of different media they are using.

10.3 What level of security will this design give?

What the ALG can do is filter out "bad" sip messages and messages which do not have any recipients on the internal LAN, it keeps the LAN clean. "Bad" messages could for instance be messages not written according to the SIP version the ALG supports or it could be that the caller/callee is issuing retransmissions much more frequently than specified in the RFC. It could also be that that the caller/callee (or anyone on the Internet/Intranet actually!) is trying to replay old messages for some reason. One thing is sure, the more the ALG has to remember in order to do this filtering of bad messages the more resources it needs and as we all know resources are usually scarce. This is true even for firewalls.

In some ways the ALG actually limits the security. The ALG cannot handle encrypted SIP messages and therefore encryption cannot be allowed. Another thing that the ALG does not support is the concept of signed (authenticated) messages. The ALG will always wreck the signature by messing with IP addresses and port numbers within the message.

11 Other solutions

A protocol called Real Specific IP (RSIP), which is now being developed by the NAT working group of the IETF. RSIP is proposing a solution to the problem with NAT and application protocols that transmits IP addresses and ports in their messages without the need for an ALG for every protocol.

11.1 An introduction to RSIP

The idea of RSIP is to have the internal clients ask an RSIP server, running on a boarder gateway between the private and the public environment, for the specific public resource required by the applications, see Figure 36. The required public resource could for instance be an IP address or an IP address and a certain number of ports. Via RSIP the clients of one addressing realm (the private) with all possible ports are given resources from another addressing realm (the public), hence the name of the protocol. The internal clients are in this way given a public presence during the time they are leasing the resources from the RSIP server. The allocated resource can then be used as desired in the outbound messages that are going to be sent.

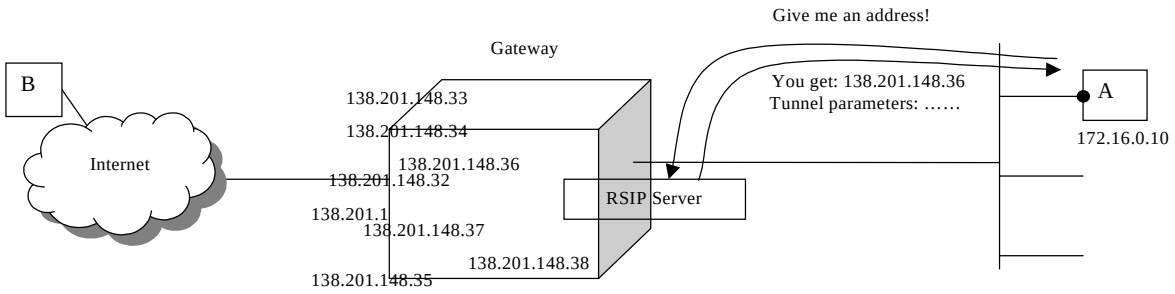

Figure 36. Request to an RSIP server for a public address

Figure 37 below, shows that the datagrams that the private host wants to send out to the public realm are encapsulated into another IP datagram and tunneled across the private address space to the RSIP server, which then removes the outer IP header and sends out the original datagram put together by the private host. Messages from the public realm directed to the address allocated by the internal host are treated similarly to the outbound datagrams. The only difference is that now the RSIP server does the encapsulation and the private host decapsulates it. The tunnel used may be encrypted, but it does not have to be.

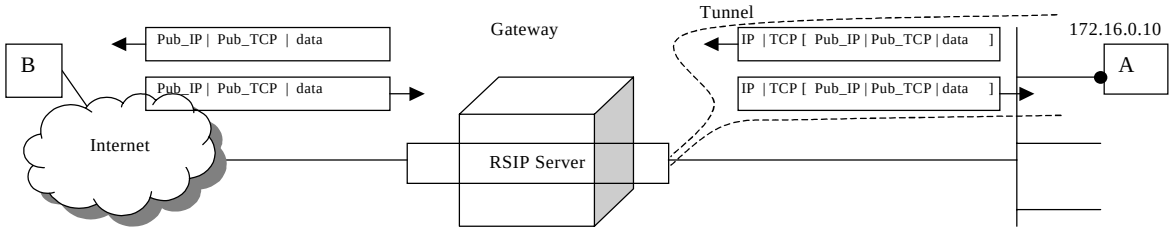

Figure 37. Tunneling with RSIP

The scenario above shows that the need for an application level gateway vanishes when RSIP is used. Since the need for an ALG that has to be able to read the data part of the datagrams has vanished it is now also possible to use end-to-end encryption between hosts.

It is clear that RSIP proposes an attractive generic solution for the problems with NAT and the need ALGs, but from what I have been able to see in [rsip-frame] and [rsip-prot] there has not yet been a solution proposed of how sessions initiated from the public realm to internal hosts will be supported. This problem has yet to be solved. This fact does not affect the possibility to receive inbound SIP calls, as is shown below.

11.2 How to use SIP with RSIP instead of NAPT

To get anything to work the SIP clients on the private network must be RSIP enabled. This is the main step necessary once the RSIP server is installed in the gateway. The client needs one port to run the application on, i.e. to listen for SIP messages on. This port needs to be requested from the RSIP server. Once this is done the client has gained global presence, thus it will be possible to reach it with SIP requests from anywhere on the Internet.

When the client wishes to start a SIP session there will be a need for more public ports to be able to receive inbound RTP streams from the other client. These will also have to be requested from the RSIP server. When the responding client on the Internet accepts the session and sends a 200 OK, he will say in his media description which addresses and ports he wants to receive RTP data streams on. The number of ports that have to be used for sending him media may be greater than, equal to or less than the number of media in the INVITE. Since the ports requested in the INVITE are opened bi-directionally, the internal client could reuse them when he wants to send to the external client. But what would happen if the number of media streams is higher in the outbound direction? Then the internal client would have to request additional ports from the RSIP server in order to be able to send the last few media streams. Instead one can make the design a little bit easier and decide that the internal client will asks the RSIP server for the amount of ports specified in the 200 OK response. These last ports are of course intended to be used when sending to the Internet host.

Sessions started from hosts on the Internet will work just as well as for sessions started from the inside, as will be shown below. It will be problem to reach the internal client from a host on the Internet as already mentioned in Section 11.1. So, when the INVITE arrives at the internal SIP client, the client will have to analyze the SDP and contact the RSIP server to open ports for the outbound RTP streams. However, this request for public ports may be delayed (and extended by the ports used for inbound RTP data streams) until the call is accepted and the 200 OK is created and sent. This saves one request to the RSIP server.

11.3 Evaluation – The choice of using an ALG or RSIP

In my opinion the RSIP solution is very attractive to use with SIP instead of NAPT with ALGs. On the positive side for RSIP is the possibility to use encryption and authentication. It will not have to be updated as SIP evolves and it will not have to support several releases of SIP. On the negative side for RSIP is that we lose the possibility to ensure that only correctly formed SIP messages are allowed into the internal client. If the client software has any security holes it might be easier to exploit them with the RSIP solution.

Both solutions have good and bad parts and it is impossible to say if one is better than the other. It might even be possible to combine the both solutions. If a combination is implemented the ALG can then check the content of the SIP message and make sure that it is OK, but without having to alter anything. Again, this will not work with encrypted SIP messages.

12 Conclusion and discussion

The goals of this Master's thesis, outlined in Section 4.1, have been achieved. The first one was to study what was required by the firewall in order to pass SIP signaling in and out of a private network that uses NAT. This subject has been discussed in great detail throughout the entire paper. The second goal was to implement an application layer gateway for SIP. How this was done is described in Section 10.

During the work with this thesis a draft for a new exiting protocol, RSIP, has been proposed to the IETF. This protocol has attributes, which enables end-to-end encryption and authentication within SIP sessions. How RSIP and SIP could work together was shown in Section 11.

This report has not in any way studied or proposed any solutions as to how the media streams in a SIP session can have ensured privacy and authentication. This is an issue that remains to be solved.

12.1 SIP development

SIP is continuously being updated, bugs are found and fixed and drafts for new methods are posted (nine up to now). An update of SIP was posted with the name "RFC2543bis" [SIPBIS] and recently Jonathan Rosenberg and the SIP security task force proposed that the SIP security model should be reevaluated. They proposed to keep most of the authentication model (which is based on HTTP/1.1) and that the model for encryption should be reconstructed almost completely. Information on where SIP is being discussed can be found at [SIPWhere].

The speed of the development shows that it might be too early to implement a commercial release of a SIP ALG that supports anything other than basic SIP signaling.

13 Acronyms

AAL5	ATM Adaptation Layer 5
ASN.1	Abstract Syntax Notation 1
ALG	Application Level Gateway
ATM	Asynchronous Transfer Mode
Codec	Coder decoder
IP	Internet Protocol
IPX	Internet Packet Exchange
ISO	International Organization for Standardization
ISP	Internet Service Provider
DHCP	Dynamic Host Configuration Protocol
HTTP	Hypertext Transfer Protocol
LAN	Local Area Network
LDAP	Lightweight Directory Access Protocol
NAT	Network Address Translation
NAPT	Network Address and Port Translation
PSTN	Public Switched Telephony Network
QoS	Quality of Service
RFC	Request For Comments
RTP	Real-Time Protocol
RTCP	Real-Time Control Protocol
RSIP	Realm-Specific IP addressing
SDP	Session Description Protocol
SIP	Session Initiation Protocol
RTO	Retransmission Time Out
RTT	Round Trip Time
TCP	Transport Control Protocol
UA	User Agent
UAC	User Agent Client
UAS	User Agent Server
UDP	User Datagram Protocol
URL	Uniform Resource Locator
UTF	Unicode Transfer Format

14 Table of Figures

15 Table of Tables

References

[Chapman95] Building Internet Firewalls. D. Brent Chapman, Elizabeth D. Zwicky. November 1995.

[Cheswick95] Firewalls and Internet Security – Repelling the Wily Hacker. William R. Cheswick, Steven M. Bellowin. 5th printing, April 1995.

[Cisco] http://www.cisco.com/

[Course488] Learning Tree course 488 – Deploying Internet and Intranet Firewalls. 1999

[DataBeam] A primer on the H.323 Series Standards. ≤1998 http://www.databeam.com/standards/index.html

[Goncalves00] Firewalls, a Complete Guide. Marcus Goncalves. 2000.

[Hasenstein97] Diplomarbeit, IP Network Address Translation. Michael Hasenstein, 1997. http://www.suse.de/~mha/linux-ip-nat/diplom/nat.html

[Intel] The Problems and Pitfalls of Getting H.323 Safely through Firewalls. 1997 http://support.intel.com/support/videophone/trial21/h323_wpr.htm

[ipchains] Linux IPCHAINS-HOWTO. Paul Russell, ipchains@rustcorp.com v1.0.7, 12 March 1999. http://www.linux.org/help/ldp/howto/IPCHAINS-HOWTO.html

[Stevens97] TCP/IP Illustrated, volume 1 – The protocols. W. Richard Stevens. 10th printing July 1997.

[RFC768] User Datagram Protocol. J. Postel. Aug-28-1980.

[RFC793] Transmission Control Protocol. J. Postel. Sep-01-1981.

[RFC822] Standard for the format of ARPA Internet text messages. D.Crocker. Aug-13-1982.

[RFC1483] Multiprotocol Encapsulation over ATM Adaptation Layer 5. Juha Heinanen. Telecom Finland. July 1993.

[RFC1631] The IP Network Address Translator (NAT). K. Egevang, P. Francis. May 1994.

[RFC1700] Assigned Numbers. J. Reynolds, J. Postel. October 1994.

[RFC1777] Lightweight Directory Access Protocol. W. Yeong, T. Howes, S., Kille. March 1995.

[RFC1889] RTP: A Transport Protocol for Real-Time Applications. Audio-Video Transport Working Group, H. Schulzrinne, S. Casner, R. Frederick, V. Jacobson. January 1996

[RFC1890] RTP Profile for Audio and Video Conferences with Minimal Control. Audio-Video Transport Working Group, H. Schulzrinne. January 1996.

[RFC1918] Address Allocation for Private Internets. Y. Rekhter, B. Moskowitz, D. Karrenberg, G. J. de Groot & E. Lear. February 1996.

[RFC1928] SOCKS Protocol Version 5. M. Leech, M. Ganis, Y. Lee, R. Kuris, D. Koblas & L. Jones. April 1996.

[RFC2044] UTF-8, a transformation format of Unicode and ISO 10646. F. Yergeau. October 1996.

[RFC2068] Hypertext Transfer Protocol -- HTTP/1.1. R. Fielding, J. Gettys, J. Mogul, H. Frystyk, T. Berners Lee. January 1997.

[RFC2131] Dynamic Host Configuration Protocol. R. Droms. March 1997.

[RFC2234] Augmented BNF for Syntax Specifications: ABNF. D. Crocker, Ed., P. Overell. November 1997

[RFC2279] UTF-8, a transformation format of ISO 10646. F. Yergeau. January 1998.

[RFC2327] SDP: Session Description Protocol. M. Handley, V. Jacobson. April 1998.

[RFC2343] RTP Payload Format for Bundled MPEG. M. Civanlar, G. Cash, B.Haskell. May 1998.

[RFC2373] IP Version 6 Addressing Architecture. R. Hinden, S. Deering. July 1998.

[RFC2401] Security Architecture for the Internet Protocol. S. Kent, R. Atkinson. November 1998.

[RFC2543] SIP: Session Initiation Protocol. M. Handley, H. Schulzrinne, E. Schooler, J. Rosenberg. March 1999.

[RFC2616] Hypertext Transfer Protocol -- HTTP/1.1. R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, T. Berners-Lee. June 1999.

[RFC2663] IP Network Address Translator (NAT) Terminology and Considerations. P. Srisuresh, M. Holdrege. August 1999

[rsip-frame] draft-ietf-nat-rsip-framework-04. M. Borella, D. Grabelsky 3Com Corp. J. Lo NEC USA. G. Montenegro Sun Microsystems. March 2000.

[rsip-prot] draft-ietf-nat-rsip-protocol-06. M. Borella, D. Grabelsky 3Com Corp. J. Lo, K. Tuniguchi NEC USA. March 2000.

[SessionTimer] draft-ietf-sip-session-timer-01. S.Donovan,J.Rosenberg MCI Worldcom,dynamicsoft SIP WG. March 9, 2000.

[SIPALG] Getting SIP through Firewalls and NATs. J. Rosenberg, D. Drew, H. Schulzrinne. February 2000.

[SIPBIS] draft-ietf-sip-rfc2543bis-00. Internet Engineering Task Force MMUSIC WG Handley/Schulzrinne/Schooler/Rosenberg ACIRI/Columbia U./Caltech/dynamicsoft April 2000.

[SIPNAT] A SIP Application Layer Gateway for Network Address Translation. B. Biggs February 2000.

[Grammar] http://www.cs.columbia.edu/~hgs/sip/SIPgrammar.html

[SIPSite] http://www.cs.columbia.edu/~hgs/sip/

[SIPWhere] http://www.cs.columbia.edu/~hgs/sip/where.html

Appendix

A. Complete Call setup of a two party call using SIP

This section can also be found in [RFC2543-Section 16]. The example does not show the SDP payloads for any message but the first to save space.

For two-party Internet phone calls, the response must contain a description of where to send the data. In the example below, Bell calls Watson. Bell indicates that he can receive RTP audio codings 0 (PCMU), 3 (GSM), 4 (G.723) and 5 (DVI4).

```
C->S: INVITE sip:watson@boston.bell-tel.com SIP/2.0
      Via: SIP/2.0/UDP kton.bell-tel.com
      From: A. Bell <sip:a.g.bell@bell-tel.com>
      To: T. Watson <sip:watson@bell-tel.com>
      Call-ID: 3298420296@kton.bell-tel.com
      CSeq: 1 INVITE
      Subject: Mr. Watson, come here.
      Content-Type: application/sdp
      Content-Length: ...

      v=0
      o=bell 53655765 2353687637 IN IP4 128.3.4.5
      s=Mr. Watson, come here.
      c=IN IP4 kton.bell-tel.com
      m=audio 3456 RTP/AVP 0 3 4 5

S->C: SIP/2.0 100 Trying
      Via: SIP/2.0/UDP kton.bell-tel.com
      From: A. Bell <sip:a.g.bell@bell-tel.com>
      To: T. Watson <sip:watson@bell-tel.com> ;tag=37462311
      Call-ID: 3298420296@kton.bell-tel.com
      CSeq: 1 INVITE
      Content-Length: 0

S->C: SIP/2.0 180 Ringing
      Via: SIP/2.0/UDP kton.bell-tel.com
      From: A. Bell <sip:a.g.bell@bell-tel.com>
      To: T. Watson <sip:watson@bell-tel.com> ;tag=37462311
      Call-ID: 3298420296@kton.bell-tel.com
      CSeq: 1 INVITE
      Content-Length: 0

S->C: SIP/2.0 182 Queued, 2 callers ahead
      Via: SIP/2.0/UDP kton.bell-tel.com
      From: A. Bell <sip:a.g.bell@bell-tel.com>
      To: T. Watson <sip:watson@bell-tel.com> ;tag=37462311
      Call-ID: 3298420296@kton.bell-tel.com
      CSeq: 1 INVITE
      Content-Length: 0

S->C: SIP/2.0 182 Queued, 1 caller ahead
      Via: SIP/2.0/UDP kton.bell-tel.com
      From: A. Bell <sip:a.g.bell@bell-tel.com>
```

```
      To: T. Watson <sip:watson@bell-tel.com> ;tag=37462311
      Call-ID: 3298420296@kton.bell-tel.com
      CSeq: 1 INVITE
      Content-Length: 0

S->C: SIP/2.0 200 OK
      Via: SIP/2.0/UDP kton.bell-tel.com
      From: A. Bell <sip:a.g.bell@bell-tel.com>
      To: <sip:watson@bell-tel.com> ;tag=37462311
      Call-ID: 3298420296@kton.bell-tel.com
      CSeq: 1 INVITE
      Contact: sip:watson@boston.bell-tel.com
      Content-Type: application/sdp
      Content-Length: ...

      v=0
      o=watson 4858949 4858949 IN IP4 192.1.2.3
      s=I'm on my way
      c=IN IP4 boston.bell-tel.com
      m=audio 5004 RTP/AVP 0 3
```

The example illustrates the use of informational status responses. Here, the reception of the call is confirmed immediately (100), then, possibly after some database mapping delay, the call rings (180) and is then queued, with periodic status updates.

Watson can only receive PCMU and GSM. Note that Watson's list of codecs may or may not be a subset of the one offered by Bell, as each party indicates the data types it is willing to receive. Watson will send audio data to port 3456 at c.bell-tel.com, Bell will send to port 5004 at boston.bell-tel.com.

By default, the media session is one RTP session. Watson will receive RTCP packets on port 5005, while Bell will receive them on port 3457.

Since the two sides have agreed on the set of media, Bell confirms the call without enclosing another session description:

```
C->S: ACK sip:watson@boston.bell-tel.com SIP/2.0
      Via: SIP/2.0/UDP kton.bell-tel.com
      From: A. Bell <sip:a.g.bell@bell-tel.com>
      To: T. Watson <sip:watson@bell-tel.com> ;tag=37462311
      Call-ID: 3298420296@kton.bell-tel.com
      CSeq: 1 ACK
```

B. SIP/SDP Message Grammar

Compiled by Arjun Roychowdhury and Henning Schulzrinne.

Based on RFC 2543,RFC 2327, RFC 2616, RFC 1123, RFC 2234, and RFC 2396, Sip Notes and Clarifications.

SIP-message	=	Request | Response	
Request	=	Request-Line *(general-header | request-header | entity-header) double-return [message-body]	; Note - each header ends with a return and the end of headers is a double return. For line folding, the folded header must begin with 1*(SP | HT)
Message-body	=	// Note: This is typically SDP	
Request-Line	=	Method SP Request-URI SP SIP-Version return	
SIP-URL	=	"sip:" [userinfo "@"] hostport url-parameters [headers]	
Userinfo	=	user [":" password]	
User	=	*(unreserved | escaped | "&" | "=" | "+" | "$" | ",")	
Password	=	*(unreserved | escaped | "&" | "=" | "+" | "$" | ",")	
Hostport	=	host [":" port]	
Host	=	hostname | IPv4address	
Hostname	=	*(domainlabel ".") toplabel ["."]	
Domainlabel	=	alphanum | alphanum *(alphanum | "-") alphanum	
Toplabel	=	alpha | alpha *(alphanum | "-") alphanum	
IPv4address	=	1*digit "." 1*digit "." 1*digit "." 1*digit	
Port	=	* digit	; empty port field is allowed
url-parameters	=	*(";" url-parameter)	
url-parameter	=	transport-param | user-param | method-param| ttl-param | maddr-param | other-param	
transport-param	=	"transport=" ("udp" | "tcp")	
ttl-param	=	"ttl=" ttl	
Ttl	=	1*3 DIGIT	; 0 to 255
maddr-param	=	"maddr=" host	
user-param	=	"user=" ("phone" | "ip")	
method-param	=	"method=" Method	
tag-param	=	"tag=" UUID	
UUID	=	1*(hex | "-")	
other-param	=	(token | (token "=" (token | quoted-string)))	
Headers	=	"?"header *("&" header)	
Header	=	hname "=" hvalue	
Hname	=	1*uric	
Hvalue	=	*uric	
Uric	=	reserved | unreserved | escaped	
Reserved	=	";" | "/" | "?" | ":" | "@" | "&" | "=" | "+" | "$" | ","	
Digits	=	1*DIGIT	
telephone-subscriber	=	global-phone-number | local-phone-number	

global-phone-number = "+" 1*phonedigit [isdn-subaddress] [post-dial]

local-phone-number = 1*(phonedigit | dtmf-digit | pause-character) [isdn-subaddress] [post-dial]

isdn-subaddress = ";isub=" 1*phonedigit

post-dial = ";postd=" 1*(phonedigit | dtmf-digit | pause-character)

Phonedigit = DIGIT | visual-separator

visual-separator = "-" | "."

pause-character = one-second-pause | wait-for-dial-tone

one-second-pause = "p"

wait-for-dial-tone = "w"

dtmf-digit = "*" | "#" | "A" | "B" | "C" | "D"

general-header =
Accept
| Accept-Encoding
| Accept-Language
| Call-ID
| Contact
| CSeq
| Date
| Encryption
| Expires
| From
| Record-Route
| Timestamp
| To
| Via

entity-header =
Content-Encoding
| Content-Length
| Content-Type

request-header =
Authorization
| Contact
| Hide
| Max-Forwards
| Organization
| Priority
| Proxy-Authorization
| Proxy-Require
| Route
| Require
| Response-Key
| Subject
| User-Agent
|WWW-Authenticate

response-header	=	Allow | Authorization | Proxy-Authenticate | Retry-After | Server | Unsupported | Warning | WWW-Authenticate	
Method	=	"INVITE" | "ACK" | "OPTIONS" | "BYE" | "CANCEL" | "REGISTER"	
Accept	=	"Accept" ":" #(media-range [accept-params])	
media-range	=	("*/*" | (type "/" "*") | (type "/" subtype)) *(";" parameter)	
Parameter	=	attribute "=" value	
Attribute	=	token	
Value	=	token | quoted-string	
accept-params	=	";" "q" "=" qvalue *(accept-extension)	
accept-extension	=	";" token ["=" (token | quoted-string)]	
Accept-Encoding	=	"Accept-Encoding" ":" 1#(content-coding)	
Codings	=	(content-coding | "*")	
content-coding	=	token	
Accept-Language	=	"Accept-Language" ":" 1#(language-range [";" "q" "=" qvalue])	
language-range	=	((1*8ALPHA *("-" 1*8ALPHA)) | "*")	
Proxy-Authorization	=	"Proxy-Authorization" ":" credentials	
Proxy-Authenticate	=	"Proxy-Authenticate" ":" 1#challenge	
User-Agent	=	"User-Agent" ":" 1*(product | comment)	
Server	=	"Server" ":" 1*(product | comment)	
Response	=	Status-Line *(general-header | response-header | entity-header) double-return [message-body]	; Note - each header ends with a return and the end of headers is a double return. For line folding, the folded header must begin with 1*(SP | HT)
Status-Line	=	SIP-version SP Status-Code SP Reason-Phrase	
SIP-version	=	"SIP/2.0"	
Status-Code	=	Informational | Success | Redirection | Client-Error | Server-Error | Global-Failure | extension-code	
extension-code	=	3 DIGIT	
Reason-Phrase	=	*<TEXT-UTF8, excluding CR, LF >	
Informational	=	"100"	Trying
	|	"180"	; Ringing
	|	"181"	; Call Is Being Forwarded
	|	"182"	; Queued

Success	=	"200"	; OK
Redirection	=	"300"	; Multiple Choices
	|	"301"	; Moved Permanently
	|	"302"	; Moved Temporarily
	|	"303"	; See Other
	|	"305"	; Use Proxy
	|	"380"	; Alternative Service
Client-Error	=	"400"	; Bad Request
	|	"401"	; Unauthorized
	|	"402"	; Payment Required
	|	"403"	; Forbidden
	|	"404"	; Not Found
	|	"405"	; Method Not Allowed
	|	"406"	; Not Acceptable
	|	"407"	; Proxy Authentication Required
	|	"408"	; Request Timeout
	|	"409"	; Conflict
	|	"410"	; Gone
	|	"411"	; Length Required
	|	"413"	; Request Entity Too Large
	|	"414"	; Request-URI Too Large
	|	"415"	; Unsupported Media Type
	|	"420"	; Bad Extension
	|	"480"	; Temporarily not available
	|	"481"	; Call Leg/Transaction Does Not Exist
	|	"482"	; Loop Detected
	|	"483"	; Too Many Hops
	|	"484"	; Address Incomplete
	|	"485"	; Ambiguous
	|	"486"	; Busy Here
Server-Error	=	"500"	; Internal Server Error
	|	"501"	; Not Implemented
	|	"502"	; Bad Gateway
	|	"503"	; Service Unavailable
	|	"504"	; Gateway Time-out
	|	"505"	; SIP Version not supported
Global-Failure	=	"600"	; Busy Everywhere
	|	"603"	; Decline
	|	"604"	; Does not exist anywhere
	|	"606"	; Not Acceptable
message-header	=	field-name ":" [field-value] CRLF	
field-name	=	token	
field-value	=	*(field-content | LWS)	

field-content	=	<the OCTETs making up the field-value and consisting of either *TEXT-UTF8 or combinations of token, separators, and quoted-string >
Allow	=	"Allow" ":" 1#Method
Call-ID	=	("Call-ID" | "i") ":" token
local-id	=	1*uric
Contact	=	("Contact" | "m") ":" ("*" | (1# ((name-addr | addr-spec) [*(";" contact-params)] [comment])))
name-addr	=	[display-name] "<" addr-spec ">"
addr-spec	=	SIP-URL | URI
display-name	=	*token | quoted-string
contact-params	=	"q" "=" qvalue| ("action" "=" "proxy" | "redirect") |("expires" "=" delta-seconds | <"> SIP-date <">) | extension-attribute
extension-attribute	=	extension-name ["=" extension-value]
Content-Encoding	=	("Content-Encoding" | "e") ":" 1#content-coding
Content-Length	=	("Content-Length" | "l") ":" 1*DIGIT
Content-Type	=	("Content-Type" | "c") ":"media-type
media-type	=	type "/"subtype *(";" parameter)
type	=	token
subtype	=	token
CSeq	=	"CSeq" ":" 1*DIGIT Method
Date	=	"Date:" SIP-date
SIP-date	=	rfc1123-date
Encryption	=	"Encryption" ":"encryption-scheme 1*SP #encryption-params
encryption-scheme	=	token
encryption-params	=	token "=" (token | quoted-string)
Expires	=	"Expires" ":" (SIP-date | delta-seconds)
From	=	("From" | "f") ":" (name-addr | addr-spec) *(";" addr-params) *(";" extension-params)
addr-params	=	tag-param
tag-param	=	"tag="UUID
Hide	=	"Hide" ":" ("route" | "hop")
Max-Forwards	=	"Max-Forwards" ":" 1*DIGIT
Organization	=	"Organization" ":" *TEXT-UTF8
Priority	=	"Priority" ":"priority-value
priority-value	=	"emergency" | "urgent" | "normal" | "non-urgent"
Record-Route	=	"Record-Route" ":" 1# name-addr
Require	=	"Require" ":" 1#option-tag
Proxy-Require	=	"Proxy-Require" ":" 1#option-tag
option-tag	=	token
Response-Key	=	"Response-Key" ":" key-scheme 1*SP #key-param
key-scheme	=	token
key-param	=	token "=" (token | quoted-string)

Retry-After	=	"Retry-After" ":"(SIP-date | delta-seconds) [comment] [";" "duration" "=" delta-seconds]	
Route	=	"Route" ":" 1#name-addr	
Subject	=	("Subject" | "s") ":" *TEXT-UTF8	
Timestamp	=	"Timestamp" ":" *(DIGIT) ["." *(DIGIT)] [delay]	
delay	=	*(DIGIT) ["." *(DIGIT)]	
To	=	("To" | "t") ":" (name-addr | addr-spec) *(";" addr-params) *(";" extension-params)	// Extension params are allowed in From to and Via
Unsupported	=	"Unsupported" ":" 1#option-tag	
Via	=	("Via" | "v") ":" 1#(sent-protocol sent-by *(";" via-params) [comment]) *(";" extension-params)	// Extension params are allowed in From to and Via
via-params	=	via-hidden | via-ttl | via-maddr| via-received | via-branch	
via-hidden	=	"hidden"	
via-ttl	=	"ttl" "=" ttl	
via-maddr	=	"maddr" "=" maddr	// Note: maddr not defined but it should be host
via-received	=	"received" "="host	
via-branch	=	"branch" "=" token	
sent-protocol	=	protocol-name "/" protocol-version "/" transport	
protocol-name	=	"SIP" | token	
protocol-version	=	token	
transport	=	"UDP" | "TCP" | token	
sent-by	=	(host [":" port]) | (concealed-host)	
concealed-host	=	token	
Warning	=	"Warning" ":" 1#warning-value	
warning-value	=	warn-code SP warn-agent SP warn-text	
warn-code	=	3 DIGIT	
warn-agent	=	(host [":" port]) | pseudonym	; the name or pseudonym of the server adding the Warning header, for use in debugging
pseudonym	=	token	
warn-text	=	quoted-string	
WWW-Authenticate	=	"WWW-Authenticate" ":" 1#challenge	
auth-scheme	=	token	
challenge	=	auth-scheme 1*SP realm *("," auth-param)	
realm	=	"realm" "=" realm-value	
auth-param	=	token "=" string	
credentials	=	basic-credentials | (auth-scheme #(auth-params))	
basic-credentials	=	"Basic" SP basic-cookie	
basic-cookie	=	<encrypted string>	
pgp-challenge	=	* (pgp-params)	
pgp-params	=	realm | pgp-version | pgp-algorithm | nonce	
realm	=	"realm" "="realm-value	
realm-value	=	quoted-string	
pgp-version	=	"version" "=" <"> digit *("." digit) *letter <">	

pgp-algorithm	=	"algorithm" "=" ("md5" | "sha1" |token)	
nonce	=	"nonce" "=" nonce-value	
nonce-value	=	quoted-string	
Authorization	=	"Authorization" ":" credentials	
pgp-response	=	realm | pgp-version | pgp-signature| signed-by | nonce	
pgp-signature	=	"signature" "="quoted-string	
signed-by	=	"signed-by" "=" <"> URI <">	
pgp-eparams	=	1# (pgp-version | pgp-encoding| pgp-key)	
pgp-encoding	=	"encoding" "=" "ascii" | token	
pgp-key	=	"key" "=" quoted-string	
OCTET	=	%x00-ff	; any 8-bit sequence of data
CHAR	=	%x00-7f	; any US-ASCII character (octets 0 - 127)
upalpha	=	"A" | "B" | "C" | "D" | "E" | "F" | "G" | "H" | "I" | "J" | "K" | "L" | "M" | "N" | "O" | "P" | "Q" | "R" | "S" | "T" | "U" | "V" | "W" | "X" | "Y" | "Z"	
lowalpha	=	"a" | "b" | "c" | "d" | "e" | "f" | "g" | "h" | "i" | "j" | "k" | "l" | "m" | "n" | "o" | "p" | "q" | "r" | "s" | "t" | "u" | "v" | "w" | "x" | "y" | "z"	
alpha	=	lowalpha | upalpha	
digit	=	"0" | "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9"	
alphanum	=	alpha | digit	
CTL	=	%x00-1f | 0x7f	; any US-ASCII control character (octets 0 -- 31) and DEL (127)
CR	=	%d13	; US-ASCII CR, carriage return character
LF	=	%d10	; US-ASCII LF, line feed character
SP	=	%d32	; US-ASCII SP, space character
HT	=	%d09	; US-ASCII HT, horizontal tab character
return	=	CR | LF | CRLF	; typically the end of a line
unreserved	=	alphanum | mark	
double-return	=	(CR CR) | (LF LF) | (CR LF CR LF)	
mark	=	"-" | "_" | "." | "!" | "~" | "*" | " ' " | "(" | ")"	
escaped	=	"%"hex hex	
LWS	=	[CRLF] 1*(SP | HT)	; linear whitespace
TEXT-UTF8	=	<any UTF-8 character encoding, except CTLs, but including LWS	
hex	=	"A" | "B" | "C" | "D" | "E" | "F" | "a" | "b" | "c" | "d" | "e" | "f" | digit	
token	=	1* (alphanum | "-" | "." | "$" | "%" | "*" | "_" | "+" | "~" | "`" | "'")	
separators	=	"(" | ")" | "<" | ">" | "@" | "," | ";" | ":" | "\" | "<" | "/" | "[" | "]" | "?" | "=" | "{" | "}" | SP | HT	
comment	=	"(" *(ctext| quoted-pair | comment) ")"	
ctext	=	<anyTEXT-UTF8 excluding "(" and ")" >	
quoted-string	=	(<"> *(qdtext | quoted-pair) <">)	
qdtext	=	<anyTEXT-UTF8 except <"> >	
quoted-pair	=	"\"CHAR	
delta-seconds	=	1*DIGIT	// This is not spec. in 2543 and no reference is given (its in H.3.3.2)

qvalue	=	("0" ["." 0*3 DIGIT]) | ("1" ["." 0*3("0")])	
URI-reference	=	([absoluteURI | relativeURI] ["#" fragment]) | (sip-url)	// According to 2453:4.3 - ReqURI is SIPUri (without certain tokens) or GeneralURI
absoluteURI	=	scheme":" (hier_part | opaque_part)	
relativeURI	=	(net_path | abs_path | rel_path) ["?" query]	
hier_part	=	(net_path | abs_path) ["?" query]	
opaque_part	=	uric_no_slash *uric	
uric_no_slash	=	unreserved | escaped | ";" | "?" | ":" | "@" | "&" | "=" | "+" | "$" | ","	
net_path	=	"//"authority [abs_path]	
abs_path	=	"/"path_segments	
rel_path	=	rel_segment [abs_path]	
rel_segment	=	1*(unreserved | escaped | ";" | "@" | "&" | "=" | "+" | "$" | ",")	
scheme	=	alpha *(alpha | digit | "+" | "-" | ".")	
authority	=	server_H | reg_name	
reg_name	=	1*(unreserved | escaped | "$" | "," | ";" | ":" | "@" | "&" | "=" | "+")	
server_H	=	[[userinfo_H "@"] hostport]	
userinfo_H	=	*(unreserved | escaped | ";" | ":" | "&" | "=" | "+" | "$" | ",")	
path	=	[abs_path | opaque_part]	
path_segments	=	segment *("/" segment)	
segment	=	*pchar *(";" param)	
param	=	*pchar	
pchar	=	unreserved | escaped | ":" | "@" | "&" | "=" | "+" | "$" | ","	
query	=	*uric	
fragment	=	*uric	
product	=	token ["/" product-version]	
product-version	=	token	
rfc1123-date	=	wkday "," SP date1SP time SP "GMT"	
date1	=	2DIGIT SP month SP 4DIGIT	; day month year (e.g., 02 Jun 1982)
time	=	2 DIGIT ":" 2 DIGIT ":" 2DIGIT	; 00:00:00 - 23:59:59
wkday	=	"Mon" | "Tue" | "Wed"| "Thu" | "Fri" | "Sat" | "Sun"	
month	=	"Jan" | "Feb" | "Mar" | "Apr" | "May" | "Jun" | "Jul" | "Aug" | "Sep" | "Oct" | "Nov" | "Dec"	

announcement	=	proto-version origin-field session-name-field information-field uri-field email-fields phone-fields connection-field bandwidth-fields time-fields key-field attribute-fields media-descriptions	
proto-version	=	"v=" 1*DIGIT return	;this memo describes version 0
origin-field	=	"o=" username space sess-id space sess-version space nettype space addrtype space addr return	
session-name-field	=	"s=" text return	
information-field	=	["i=" text return]	
uri-field	=	["u=" uri return]	
email-fields	=	*("e=" email-address return)	
phone-fields	=	*("p=" phone-number return)	
connection-field	=	["c=" nettype space addrtype space connection-address return]	;a connection field must be present ;in every media description or at the ;session-level
bandwidth-fields	=	*("b=" bwtype ":" bandwidth return)	
time-fields	=	1*("t=" start-time space stop-time *(return repeat-fields) return) [zone-adjustments return]	
repeat-fields	=	"r=" repeat-interval space typed-time 1*(space typed-time)	
zone-adjustments	=	time space ["-"] typed-time *(space time space ["-"] typed-time)	
key-field	=	["k=" key-type return]	
key-type	=	"prompt" | "clear:" key-data | "base64:" key-data |"uri:" uri	

key-data	=	email-safe | "~" | "	
attribute-fields	=	*("a=" attribute return)	
media-field	=	"m=" media space port ["/" integer] space proto 1*(space fmt) return	
media-descriptions	=	*(media-field information-field *(connection-field) bandwidth-fields key-field attribute-fields)	
media	=	1*(alpha-numeric)	;typically "audio", "video", "application" ;or "data"
fmt	=	1*(alpha-numeric)	;typically an RTP payload type for audio ;and video media
proto	=	1*(alpha-numeric)	;typically "RTP/AVP" or "udp" for IP4
port	=	1*(DIGIT)	;should in the range "1024" to "65535" inclusive ;for UDP based media
attribute	=	(att-field ":" att-value) | att-field	
att-field	=	1*(alpha-numeric)	
att-value	=	byte-string	
sess-id	=	1*(DIGIT)	;should be unique for this originating username/host
sess-version	=	1*(DIGIT)	;0 is a new session
connection-address	=	multicast-address | addr	
multicast-address	=	3*(decimal-uchar ".") decimal-uchar "/" ttl ["/" integer]	;multicast addresses may be in the range ;224.0.0.0 to 239.255.255.255
start-time	=	time | "0"	
stop-time	=	time | "0"	
time	=	POS-DIGIT 9*(DIGIT)	;sufficient for 2 more centuries
repeat-interval	=	typed-time	

typed-time	=	1*(DIGIT) [fixed-len-time-unit]	
fixed-len-time-unit	=	"d" | "h" | "m" | "s"	
bwtype	=	1*(alpha-numeric)	
bandwidth	=	1*(DIGIT)	
username	=	safe	;pretty wide definition, but doesn't include space
email-address	=	email | email "(" email-safe ")" | email-safe "<" email ">"	
email	=	; defined in RFC822	
phone-number	=	phone | phone "(" email-safe ")" | email-safe "<" phone ">"	
phone	=	"+" POS-DIGIT 1*(space | "-" | DIGIT)	;there must be a space or hyphen between the ;international code and the rest of the number.
nettype	=	"IN"	;list to be extended
addrtype	=	"IP4" | "IP6"	;list to be extended
addr	=	FQDN | unicast-address	
FQDN	=	4*(alpha-numeric|"-"|".")	;fully qualified domain name as specified in RFC1035
unicast-address	=	IP4-address | IP6-address	
IP4-address	=	b1 "." decimal-uchar "." decimal-uchar "." b4	
b1	=	decimal-uchar	;less than "224"; not "0" or "127"
b4	=	decimal-uchar	;not "0"
IP6-address	=		;to be defined
text	=	byte-string	;default is to interpret this as I0-10646 UTF8 ;ISO 8859-1 requires a "a=charset:ISO-8859-1" ;session-level attribute to be used
byte-string	=	1*(0x01..0x09|0x0b|0x0c|0x0e..0xff)	;any byte except NUL, CR or LF
decimal-uchar	=	DIGIT | POS-DIGIT DIGIT | ("1" 2*(DIGIT)) | ("2" ("0"|"1"|"2"|"3"|"4") DIGIT) | ("2" "5" ("0"|"1"|"2"|"3"|"4"|"5"))	

Integer = POS-DIGIT *(DIGIT)

POS-DIGIT = "1"|"2"|"3"|"4"|"5"|"6"|"7"|"8"|"9"

Email-safe = safe | space | tab

Safe = alpha-numeric | "'" | "`" | "-" | "." | "/" | ":" | "?" | "'" | "#" | "$" | "&" | "*" | ";" | "=" | "@" | "[" | "]" | "^" | "_" | "`" | "{" | "|" | "}" | "+" | "~" | "

Space = SP

C. ABNF

Network Working Group
Request for Comments: 2234
Category: Standards Track

D. Crocker, Ed.
Internet Mail Consortium
P. Overell
Demon Internet Ltd.
November 1997

Augmented BNF for Syntax Specifications: ABNF

Status of this Memo

Copyright Notice

TABLE OF CONTENTS

1. INTRODUCTION

Internet technical specifications often need to define a format syntax and are free to employ whatever notation their authors deem useful. Over the years, a modified version of Backus-Naur Form (BNF), called Augmented BNF (ABNF), has been popular among many Internet specifications. It balances compactness and simplicity, with reasonable representational power. In the early days of the Arpanet, each specification contained its own definition of ABNF. This included the email specifications, RFC733 and then RFC822 which have come to be the common citations for defining ABNF. The current document separates out that definition, to permit selective reference. Predictably, it also provides some modifications and enhancements.

The differences between standard BNF and ABNF involve naming rules, repetition, alternatives, order-independence, and value ranges. Appendix A (Core) supplies rule definitions and encoding for a core lexical analyzer of the type common to several Internet specifications. It is provided as a convenience and is otherwise separate from the meta language defined in the body of this document, and separate from its formal status.

2. RULE DEFINITION

2.1 Rule Naming

The name of a rule is simply the name itself; that is, a sequence of characters, beginning with an alphabetic character, and followed by a combination of alphabetics, digits and hyphens (dashes).

NOTE: Rule names are case-insensitive

The names <rulename>, <Rulename>, <RULENAME> and <rUlENamE> all refer to the same rule.

Unlike original BNF, angle brackets ("<", ">") are not required. However, angle brackets may be used around a rule name whenever their presence will facilitate discerning the use of a rule name. This is typically restricted to rule name references in free-form prose, or to distinguish partial rules that combine into a string not separated by white space, such as shown in the discussion about repetition, below.

2.2 Rule Form

A rule is defined by the following sequence:

```
name =  elements crlf
```

where <name> is the name of the rule, <elements> is one or more rule names or terminal specifications and <crlf> is the end-of- line indicator, carriage return followed by line feed. The equal sign separates the name from the definition of the rule. The elements form a sequence of one or more rule names and/or value definitions, combined according to the various operators, defined in this document, such as alternative and repetition.

For visual ease, rule definitions are left aligned. When a rule requires multiple lines, the continuation lines are indented. The left alignment and indentation are relative to the first lines of the ABNF rules and need not match the left margin of the document.

2.3 Terminal Values

Rules resolve into a string of terminal values, sometimes called characters. In ABNF a character is merely a non-negative integer. In certain contexts a specific mapping (encoding) of values into a character set (such as ASCII) will be specified.

Terminals are specified by one or more numeric characters with the base interpretation of those characters indicated explicitly. The following bases are currently defined:

```
b           =  binary

d           =  decimal

x           =  hexadecimal
```

Hence:

```
     CR          =  %d13

     CR          =  %x0D
```

respectively specify the decimal and hexadecimal representation of [US-ASCII] for carriage return.

A concatenated string of such values is specified compactly, using a period (".") to indicate separation of characters within that value. Hence:

```
     CRLF        =  %d13.10
```

ABNF permits specifying literal text string directly, enclosed in quotation-marks. Hence:

```
     command     =  "command string"
```

Literal text strings are interpreted as a concatenated set of printable characters.

```
     NOTE:     ABNF strings are case-insensitive and
               the character set for these strings is us-ascii.
```

Hence:

```
     rulename = "abc"
```

and:

```
     rulename = "aBc"
```

will match "abc", "Abc", "aBc", "abC", "ABc", "aBC", "AbC" and "ABC".

```
              To specify a rule which IS case SENSITIVE,
                 specify the characters individually.
```

For example:

```
     rulename    =  %d97 %d98 %d99
```

or

```
     rulename    =  %d97.98.99
```

will match only the string which comprises only lowercased characters, abc.

2.4 External Encodings

External representations of terminal value characters will vary according to constraints in the storage or transmission environment. Hence, the same ABNF-based grammar may have multiple external encodings, such as one for a 7-bit US-ASCII environment, another for a binary octet environment and still a different one when 16-bit Unicode is used. Encoding details are beyond the scope of ABNF, although Appendix A (Core) provides definitions for a 7-bit US-ASCII environment as has been common to much of the Internet.

By separating external encoding from the syntax, it is intended that alternate encoding environments can be used for the same syntax.

3. OPERATORS

3.1 Concatenation Rule1 Rule2

A rule can define a simple, ordered string of values -- i.e., a concatenation of contiguous characters -- by listing a sequence of rule names. For example:

```
foo         =  %x61           ; a

bar         =  %x62           ; b

mumble      =  foo bar foo
```

So that the rule <mumble> matches the lowercase string "aba".

LINEAR WHITE SPACE: Concatenation is at the core of the ABNF parsing model. A string of contiguous characters (values) is parsed according to the rules defined in ABNF. For Internet specifications, there is some history of permitting linear white space (space and horizontal tab) to be freelyPand implicitlyPinterspersed around major constructs, such as delimiting special characters or atomic strings.

NOTE: This specification for ABNF does not provide for implicit specification of linear white space.

Any grammar which wishes to permit linear white space around delimiters or string segments must specify it explicitly. It is often useful to provide for such white space in "core" rules that are

then used variously among higher-level rules. The "core" rules might be formed into a lexical analyzer or simply be part of the main ruleset.

3.2 Alternatives Rule1 / Rule2

Elements separated by forward slash ("/") are alternatives. Therefore,

```
foo / bar
```

will accept <foo> or <bar>.

NOTE: A quoted string containing alphabetic characters is special form for specifying alternative characters and is interpreted as a non-terminal representing the set of combinatorial strings with the contained characters, in the specified order but with any mixture of upper and lower case..

3.3 Incremental Alternatives Rule1 =/ Rule2

It is sometimes convenient to specify a list of alternatives in fragments. That is, an initial rule may match one or more alternatives, with later rule definitions adding to the set of alternatives. This is particularly useful for otherwise- independent specifications which derive from the same parent rule set, such as often occurs with parameter lists. ABNF permits this incremental definition through the construct:

```
oldrule     =/ additional-alternatives
```

So that the rule set

```
ruleset     =  alt1 / alt2

ruleset     =/ alt3

ruleset     =/ alt4 / alt5
```

is the same as specifying

```
ruleset     =  alt1 / alt2 / alt3 / alt4 / alt5
```

3.4 Value Range Alternatives %c##-##

A range of alternative numeric values can be specified compactly, using dash ("-") to indicate the range of alternative values. Hence:

```
DIGIT       =  %x30-39
```

is equivalent to:

```
DIGIT       =  "0" / "1" / "2" / "3" / "4" / "5" / "6" /

                     "7" / "8" / "9"
```

Concatenated numeric values and numeric value ranges can not be specified in the same string. A numeric value may use the dotted notation for concatenation or it may use the dash notation to specify one value range. Hence, to specify one printable character, between end of line sequences, the specification could be:

```
char-line = %x0D.0A %x20-7E %x0D.0A
```

3.5 Sequence Group (Rule1 Rule2)

Elements enclosed in parentheses are treated as a single element, whose contents are STRICTLY ORDERED. Thus,

```
elem (foo / bar) blat
```

which matches (elem foo blat) or (elem bar blat).

```
elem foo / bar blat
```

matches (elem foo) or (bar blat).

NOTE: It is strongly advised to use grouping notation, rather than to rely on proper reading of "bare" alternations, when alternatives consist of multiple rule names or literals.

Hence it is recommended that instead of the above form, the form:

```
(elem foo) / (bar blat)
```

be used. It will avoid misinterpretation by casual readers.

The sequence group notation is also used within free text to set off an element sequence from the prose.

3.6 Variable Repetition *Rule

The operator "*" preceding an element indicates repetition. The full form is:

```
<a>*<b>element
```

where <a> and <b> are optional decimal values, indicating at least <a> and at most <b> occurrences of element.

Default values are 0 and infinity so that *<element> allows any number, including zero; 1*<element> requires at least one; 3*3<element> allows exactly 3 and 1*2<element> allows one or two.

3.7 Specific Repetition nRule

A rule of the form:

```
<n>element
```

is equivalent to

```
<n>*<n>element
```

That is, exactly <N> occurrences of <element>. Thus 2DIGIT is a 2-digit number, and 3ALPHA is a string of three alphabetic characters.

3.8 Optional Sequence [RULE]

Square brackets enclose an optional element sequence:

```
[foo bar]
```

is equivalent to

```
*1(foo bar).
```

3.9 ; Comment

A semi-colon starts a comment that continues to the end of line. This is a simple way of including useful notes in parallel with the specifications.

3.10 Operator Precedence

The various mechanisms described above have the following precedence, from highest (binding tightest) at the top, to lowest and loosest at the bottom:

Strings, Names formation
Comment
Value range
Repetition
Grouping, Optional
Concatenation
Alternative

Use of the alternative operator, freely mixed with concatenations can be confusing.

Again, it is recommended that the grouping operator be used to make explicit concatenation groups.

4. ABNF DEFINITION OF ABNF

This syntax uses the rules provided in Appendix A (Core).

```
rulelist       =  1*( rule / (*c-wsp c-nl) )

rule           =  rulename defined-as elements c-nl
                       ; continues if next line starts
                       ;  with white space

rulename       =  ALPHA *(ALPHA / DIGIT / "-")

defined-as     =  *c-wsp ("=" / "=/") *c-wsp
                       ; basic rules definition and
                       ;  incremental alternatives

elements       =  alternation *c-wsp

c-wsp          =  WSP / (c-nl WSP)

c-nl           =  comment / CRLF
                       ; comment or newline

comment        =  ";" *(WSP / VCHAR) CRLF

alternation    =  concatenation
                  *(*c-wsp "/" *c-wsp concatenation)
```

```
concatenation  =  repetition *(1*c-wsp repetition)

repetition     =  [repeat] element

repeat         =  1*DIGIT / (*DIGIT "*" *DIGIT)

element        =  rulename / group / option /
                  char-val / num-val / prose-val

group          =  "(" *c-wsp alternation *c-wsp ")"

option         =  "[" *c-wsp alternation *c-wsp "]"

char-val       =  DQUOTE *(%x20-21 / %x23-7E) DQUOTE
                       ; quoted string of SP and VCHAR
                          without DQUOTE

num-val        =  "%" (bin-val / dec-val / hex-val)

bin-val        =  "b" 1*BIT
                  [ 1*("." 1*BIT) / ("-" 1*BIT) ]
                       ; series of concatenated bit values
                       ; or single ONEOF range

dec-val        =  "d" 1*DIGIT
                  [ 1*("." 1*DIGIT) / ("-" 1*DIGIT) ]

hex-val        =  "x" 1*HEXDIG
                  [ 1*("." 1*HEXDIG) / ("-" 1*HEXDIG) ]

prose-val      =  "<" *(%x20-3D / %x3F-7E) ">"
                       ; bracketed string of SP and VCHAR
                          without angles
                       ; prose description, to be used as
                          last resort
```

5. SECURITY CONSIDERATIONS

Security is truly believed to be irrelevant to this document.

6. APPENDIX A - CORE

This Appendix is provided as a convenient core for specific grammars. The definitions may be used as a core set of rules.

6.1 Core Rules

Certain basic rules are in uppercase, such as SP, HTAB, CRLF, DIGIT, ALPHA, etc.

```
     ALPHA          =  %x41-5A / %x61-7A   ; A-Z / a-z

     BIT            =  "0" / "1"

     CHAR           =  %x01-7F
                           ; any 7-bit US-ASCII character,
                              excluding NUL

     CR             =  %x0D
                           ; carriage return

     CRLF           =  CR LF
                           ; Internet standard newline

     CTL            =  %x00-1F / %x7F
                           ; controls

     DIGIT          =  %x30-39
                           ; 0-9

     DQUOTE         =  %x22
                           ; " (Double Quote)

     HEXDIG         =  DIGIT / "A" / "B" / "C" / "D" / "E" / "F"

     HTAB           =  %x09
                           ; horizontal tab

     LF             =  %x0A
                           ; linefeed

     LWSP           =  *(WSP / CRLF WSP)
                           ; linear white space (past newline)

     OCTET          =  %x00-FF
                           ; 8 bits of data

     SP             =  %x20
```

```
                              ; space

        VCHAR          =  %x21-7E
                              ; visible (printing) characters

        WSP            =  SP / HTAB
                              ; white space
```

6.2 Common Encoding

Externally, data are represented as "network virtual ASCII", namely 7-bit US-ASCII in an 8-bit field, with the high (8th) bit set to zero. A string of values is in "network byte order" with the higher-valued bytes represented on the left-hand side and being sent over the network first.

7. ACKNOWLEDGMENTS

The syntax for ABNF was originally specified in RFC 733. Ken L. Harrenstien, of SRI International, was responsible for re-coding the BNF into an augmented BNF that makes the representation smaller and easier to understand.

This recent project began as a simple effort to cull out the portion of RFC 822 which has been repeatedly cited by non-email specification writers, namely the description of augmented BNF. Rather than simply and blindly converting the existing text into a separate document, the working group chose to give careful consideration to the deficiencies, as well as benefits, of the existing specification and related specifications available over the last 15 years and therefore to pursue enhancement. This turned the project into something rather more ambitious than first intended. Interestingly the result is not massively different from that original, although decisions such as removing the list notation came as a surprise.

The current round of specification was part of the DRUMS working group, with significant contributions from Jerome Abela , Harald Alvestrand, Robert Elz, Roger Fajman, Aviva Garrett, Tom Harsch, Dan Kohn, Bill McQuillan, Keith Moore, Chris Newman , Pete Resnick and Henning Schulzrinne.

8. REFERENCES

[US-ASCII] Coded Character Set--7-Bit American Standard Code for Information Interchange, ANSI X3.4-1986.

[RFC733] Crocker, D., Vittal, J., Pogran, K., and D. Henderson, "Standard for the Format of ARPA Network Text Message," RFC 733, November 1977.

[RFC822] Crocker, D., "Standard for the Format of ARPA Internet Text Messages", STD 11, RFC 822, August 1982.

9. CONTACT

David H. Crocker

Internet Mail Consortium
675 Spruce Dr.
Sunnyvale, CA 94086 USA

Phone: +1 408 246 8253
Fax: +1 408 249 6205
EMail: dcrocker@imc.org

Paul Overell

Demon Internet Ltd
Dorking Business Park
Dorking
Surrey, RH4 1HN
UK

paulo@turnpike.com

10. Full Copyright Statement