NAT Traversal techniques in VoIP protocols

Master of Science Thesis

submitted for the academic degree of
Diplomingenieur (Dipl.-Ing.)

carried out at the
Research Group for Industrial Software
Institute of Computer Aided Automation
Vienna University of Technology

under the guidance of
Univ.-Prof. DI Dr. Thomas Grechenig

by
Günther Starnberger
Allerheiligengasse 5/2/1
1200 Wien
gst@sysfrog.org

Vienna, October 12, 2007

Contents

Acknowledgements

First of all, I would like to sincerely thank Prof. Thomas Grechenig for the guidance of this diploma thesis and his invaluable counsel. I would also like to thank my contributing advisor Dipl.-Ing. Roland Pezzei for his support and helpful suggestions of organizing this thesis. In addition, I am particularly thankful to Dr. Brigitte Brem, Mag. Martin Marktl and Marion Tischler for formal reviews and thoroughly proofreading the manuscript.

Günther Starnberger
October 2007

Abstract

Network Address Translation (NAT) poses an inherent problem to many peer-to-peer protocols. If one or more of the hosts which want to communicate are located behind a NAT gateway, the hosts outside of this network are not able to address IP packets to the hosts inside the network. Methods which are used to work around this problem are denoted under the umbrella term NAT traversal techniques.

This thesis examines the NAT traversal capabilities of common VoIP protocols. At the beginning, network technologies, which are used by VoIP protocols, are examined. Afterwards, the design of NAT devices is described. The work continues by inspecting VoIP protocols, as well as the NAT traversal techniques used within these protocols.

A comparative survey examines each VoIP protocol for a set of predefined NAT environments. It is shown, that state of the art protocols are not able to provide optimal traversal capabilities for all types of NAT devices. Some protocols are able to establish a connection in all examined cases. However, in some setups, they revert to an inefficient relayed connection, instead of using a direct peer-to-peer connection. This causes a higher latency and additional bandwidth requirements at the location of the VoIP provider.

The work concludes by proposing enhancements to existing NAT traversal techniques, which allow the traversal of NAT gateways in cases where current techniques would fail to establish a direct connection. The first technique uses the NAT-PMP protocol in order to explicitly request a binding on the NAT device, while the second technique tries to predict the port-number, which the NAT device assigns to a particular binding. Both techniques can be implemented as extensions to the Interactive Connectivity Establishment (ICE) protocol.

Zusammenfassung

Network Address Translation (NAT) stellt ein wesentliches Problem für viele Peer-to-Peer Protokolle dar. Rechnern, die sich außerhalb eines NAT-Gateways befinden, ist es nicht möglich, IP Pakete an Rechner innerhalb des NAT-Gateways zu adressieren. Techniken, welche es erlauben, direkte Verbindungen trotz NAT-Gateways zu etablieren, werden als NAT-Traversal-Techniken bezeichnet.

Diese Arbeit untersucht die NAT-Traversal-Fähigkeiten von aktuellen VoIP-Protokollen. Zuerst werden Netzwerktechnologien, die von VoIP Protokollen verwendet werden, erklärt. Danach wird der Aufbau von NAT Gateways beschrieben. Anschließend werden aktuelle VoIP Protokolle, sowie die in diesen Protokollen verwendeten NAT-Traversal-Techniken untersucht.

Die Eigenschaften der Protokolle werden für unterschiedliche Typen von NAT-Gateways verglichen. Es wird gezeigt, dass die evaluierten Protokolle nicht immer eine optimale Traversal-Technik verwenden. Einige Protokolle schaffen es in allen untersuchten Fällen eine Verbindung aufzubauen, allerdings wird in einigen dieser Fälle eine ineffiziente indirekte Verbindung anstatt einer direkten Peer-to-Peer Verbindung verwendet. Dies führt zu einer höheren Latenzzeit bei Gesprächen und zu höherem Datenaufkommen bei dem VoIP-Provider.

Abschließend werden Erweiterungen zu dem aktuellen Stand der Technik vorgestellt, welche das Herstellen von Verbindungen auch dann erlauben, wenn dieses mit den etablierten Verfahren nicht möglich ist. Das erste Verfahren basiert auf dem NAT-PMP Protokoll und erlaubt es, explizit eine Weiterleitung auf einem NAT-Gateway anzufordern. Das zweite Verfahren versucht vorauszusagen, welche Portnummer der NAT-Gateway an ein bestimmte Weiterleitung zuweist. Beide Techniken können als Erweiterungen zu dem Interactive Connectivity Establishment (ICE) Protokoll implementiert werden.

Chapter 1

Introduction

During the past years, the Internet evolved from a dumb network to a semi-intelligent network [1]. One reason for this are Network Address Translation (NAT) devices, which map the private IP space used within a private network to a single IP address which can be used on the public Internet. Their goal is to allow multiple hosts on the private network to access the Internet with a single IP address. Unfortunately, these middleboxes break the end-to-end principle [2] upon which the Internet is based. This results in additional complexity which is introduced into the network [3].

NAT devices use various heuristics in order to assign packets sent to their public address to a host on the internal network [4]. These heuristics have been optimized for traditional client-server protocols, where a client within the network establishes a connection to a server on the public Internet. Unfortunately, peer-to-peer protocols usually require specific workarounds, if one of the peers is located behind a NAT [5] [6]. A reason for this is, that the public IP address of a peer behind a NAT is not distinct anymore. A NAT device discards a packet, if it is not able to assign it to a host on the private network.

One of the areas affected by these problems is Voice-over-IP (VoIP). VoIP protocols are usually built upon the peer-to-peer paradigm. Reasons for this include lower latencies than a server based approach and lower bandwidth requirements at the location of the VoIP provider [7].

In order to be able to establish connections to peers behind a NAT device, VoIP protocols use various NAT traversal techniques. These techniques range from simple relay proxies which are able to relay the data between two peers, to more sophisticated solutions like Simple Traversal

of UDP through NATs (STUN), which tries to identify port numbers on the public interface of the NAT device, which can be used to reach a specific host inside the NAT.

The first VoIP protocols did not implement any traversal techniques. As a result, the establishment of connections between two peers failed, if at least one of them was located behind a NAT device. Newer techniques tried to make educated guesses about the topology of the underlying network and about the NAT device in use. As implementations of NAT devices differ, this guesses are not guaranteed to be correct. In the case of a wrong guess the consequences reach from the usage of an inefficient traversal technique to a failure in the establishment of the connection. New techniques currently in development improve existing NAT traversal protocols by avoiding dependence on assumptions about the underlying network.

The thesis is divided into four main chapters:

Chapter 2 describes the network technologies which are affected by Network Address Translation. It starts by discussing the OSI Reference Model, which describes the different layers of network protocols. Afterwards, the TCP/IP protocol stack is examined together with the network protocols which are part of this stack. The chapter concludes by describing how Network Address Translation devices are implemented.

In Chapter 3 traversal techniques, which can be used to bypass NAT devices, are described. The first section, which covers the UDP Hole Punching technique, describes the fundamental concepts of NAT traversal. The later sections describe particular traversal techniques, which have been designed for the usage in VoIP protocols.

In Chapter 4 three widely used VoIP signaling protocols are examined. These signaling protocols make use of the traversal techniques which have been described in Chapter 3.

Chapter 5 contains a comparative survey of the traversal capabilities of VoIP protocols. The traversal mechanisms for each evaluated VoIP protocol are checked against each evaluated type of NAT device. The chapter concludes by proposing new ideas, which allow to improve upon the current state of the art of NAT traversal technologies.

Chapter 2

Network technologies

This chapter covers the network technologies on which VoIP protocols operate. As the name indicates, VoIP protocols are built upon the Internet Protocol (IP).

VoIP protocols are usually split into two parts:

- A signaling part, which is responsible for tasks like call setup and call shutdown.
- A data part, which is responsible for the transfer of the voice data between caller and callee.

Voice data is usually transferred using the Real-time Transport Protocol (RTP). RTP is based upon the User Datagram Protocol (UDP), which itself is based upon IP. For signaling, either UDP or the Transmission Control Protocol (TCP) is used. Like UDP, TCP is based on IP.

The IP network can be built upon different underlying protocols, for example Ethernet, PPP or ATM. These protocols can affect the VoIP communication, as they influence characteristics like latency and packet loss of the IP network.

This chapter first describes the OSI Reference Model which influenced the design of later protocols like TCP/IP. Afterwards, the protocols within the TCP/IP suite, which affect NAT traversal techniques, are introduced. These are: IP, UDP, UDP-Lite, RTP, TCP. At the end of the chapter, the mode of operation of Network Address Translation is described.

2.1 OSI Reference Model

The OSI Reference Model describes the seven layers of the OSI Network Protocol. The OSI Reference Model and the OSI Network Protocol are part of a networking suite, which was developed at the International Organization for Standardization (ISO). The OSI Reference Model itself is not a network protocol, as it only describes the tasks of the different layers.

While the OSI Network Protocol is not used on the Internet, the OSI Reference Model is still used to describe network protocols. Most of today's protocols follow the TCP/IP model. The separation of the layers in the OSI Network Protocol is stricter than the separation in the TCP/IP model. Also, the TCP/IP model contains less layers than the OSI Network Protocol. Therefore, it is not possible, to map each layer of the OSI Network Protocol to exactly one layer of the TCP/IP model. [8]

The principles that were applied to design the layers of the OSI Reference Model are [9]:

- Layers are created where different abstractions are needed
- Layers perform a well defined function
- The information flow across the interfaces between the layers should be minimized

The tasks of the layers are:

- Application Layer

  This layer is the highest layer of the OSI model. The application layer does not provide services to any other layer. It contains protocols which are commonly needed by users. Examples of protocols which are part of the application layer are: HTTP, SMTP and FTP.

- Presentation Layer

  This layer manages data structures and allows the definition of higher level data structures. It specifies how the data structures are encoded on the wire. The application layer can use these data structures without needing to know how the data is actually encoded.

- Session Layer

  This Layer manages the sessions which are established between different machines. It provides mechanisms for organizing and structuring interactions between communication processes.

- Transport Layer

  The transport layer accepts data and splits it into smaller units. Furthermore, the transport layer ensures that the data arrives correctly at the remote endpoint.

  There are different kinds of abstractions which the transport layer may offer to higher layers. Examples are:

  - Error-free Point-to-Point channel: This channel exhibits the behavior of a FIFO. It ensures that all messages which were sent are received, and that all messages arrive in the correct order. Its behavior is similar to the behavior of TCP.
  - Isolated messages: This channel does not guarantee that messages arrive in the correct order or that messages arrive at all. Its behavior is similar to UDP.

  As the Transport Layer provides these kinds of virtual channels to the upper layers, the upper layers do not need to take care of the issues involved.

- Network Layer

  The Network Layer is responsible for the routing of packets, for intercommunication between networks, for the Quality of Service (QOS) and for congestion issues. The routing path can be determined by using either static or dynamic routing tables.

- Data Link Layer

  The Data Link Layer splits an input stream into multiple frames. It is able to detect and it may be able to correct errors, which have been introduced at the Physical Layer.

  The Data Link Layer can be implemented as either a reliable or an unreliable channel. If it is implemented as a reliable channel, acknowledgment packets are sent in order to verify the correct transmission of a frame.

- Physical Layer

  The task of the Physical Layer is to transfer raw bits over a communication channel. It defines how 0 and 1 are represented physically on the communication channel. If an electrical communication channel is used, this includes the definition of the voltages that are used for the representation of the bits. This layer also deals with timing issues, which define for how long a single bit is sent.

2.2 TCP/IP Protocol Stack

The protocols specified within the TCP/IP stack are used on the public Internet. The design of the stack is based upon the TCP/IP reference model, which is similar to the OSI reference model described in the previous chapter.

In [9], Andrew Tanenbaum describes the four layers which form the TCP/IP reference model: Application layer, Transport layer, Internet layer, Host-to-Network layer. The Presentation and Session layers of the OSI Model are not part of the TCP/IP model. Furthermore, the Data Link and the Physical Layer of the OSI model are represented by the Host-to-Network Layer.

The tasks of the layers in the TCP/IP model are:

- Application Layer: This layer defines higher level protocols which are used by applications. An example is the Hypertext Transfer Protocol (HTTP) protocol which is used for the transfer of web pages or the Simple Mail Transfer Protocol (SMTP) protocols specifies how emails are transferred between two entities.
- Transport Layer: The function of this layer is similar to the function of the OSI Transport Layer. It defines protocols which are used by applications for end-to-end communication. Two of the specified protocols are TCP and UDP.
- Internet Layer: The function of this layer is similar to the function of the OSI Network Layer. Its main tasks are the routing of packets and the prevention of congestion. The Internet Protocol is part of this layer.
- Host-to-Network Layer: This layer describes the protocol used by the host to connect to the network. The specification of the protocol is outside the scope of the TCP/IP protocol stack.

This chapter introduces the different protocols used within the TCP/IP protocol stack. It starts with the Internet Protocol upon which all other protocols inside the stack are based. Afterwards, it discusses the User Datagram Protocol (UDP), UDP-lite and the Real-time Transport Protocol (RTP). It concludes by describing the Transmission Control Protocol (TCP).

2.2.1 Internet Protocol

The Internet Protocol (IP) is the lowest layer of the TCP/IP protocol suite. It depends on a lower layer protocol like Ethernet or FDDI which provides access to the underlying network. It provides services to higher level protocols like TCP and UDP. IP is defined in RFC 791 [10].

IP was initially designed by the US Department of Defense (DoD) for the use on the Advanced Research Projects Agency Network (ARPANET). It is based upon an earlier version of the ARPA Internet Protocol. IP was designed for the use in interconnected packet switched computer communication systems. It allows entities on the network to send and receive packets and to fragment and reassemble a packet if it is too long. Each entity on the network is identified by a 32-bit network address.

Applications do not directly use the Internet Protocol. Instead, they use higher level protocols which are build upon the Internet Protocol. Usually TCP is used if an error free byte stream connection is required, and UDP is used if connections may exhibit packet loss behavior. A full IP stack is commonly referred to as TCP/IP.

Currently[1] there are two versions of IP in use: IP Version 4 (IPv4) and IP Version 6 (IPv6). IPv4 is used on the public Internet. IPv6 is the successor to IPv4. Its main advantage is that it provides a larger address space than IPv4. While there are some experimental and production level networks which use IPv6, it is not widely used on the public Internet. This work will only deal with IPv4. References to IP without any version number always refer to IP Version 4.

An IP packet can be divided into two parts: The header and the payload. The header can be split into two further parts: A first part with a fixed length of 20 bytes and an optional second part with a variable length. All fields are encoded in big-endian order[2].

IP Addresses Hosts on an IP network are identified by unique IP addresses. IPv4 addresses have a size of 32-bit, the size of IPv6 addresses is 128-bit.

An IP address is split into two parts. The first part identifies the address of the network where the host is located. The second part identifies the host within the network.

[1] as of 2007

[2] In a big-endian order the most significant byte value is stored in the memory address with the lowest value.

Historically classes have been used to specify which part of the IP address belongs to the network and which part is used for the host. Three different classes have been defined: class A, class B and class C. In a class A network the first byte addresses the network, and the next 3 bytes the host. This means that a maximum of 16.777.214 addresses are available within a class A network. A class B network assigns 2 bytes to the network and two bytes to the host and a class C network assigns 3 bytes to the network and 1 byte to the host.

Today, classless routing is used. A subnet mask is used to split an IP address into a network address and a host address. A subnet mask has the same size as an IP address. It starts with either zero or more ones, the rest of the mask is filled with zeros. The parts of the IP address where the bits of the subnet mask are one indicate the network, the parts where the bits are zero indicate the host. An alternative notation for subnet masks is the prefix length. The prefix length specifies the number of ones in the subnet mask.

When a host needs to send a packet to another host, it first checks if this host is within the same subnet. If this is the case, the packet is directly sent to this host. Otherwise, the host checks if it has a route in its routing table, which can be used to reach the IP address of the destination. If no route exists the packet is dropped, otherwise it is forwarded to the next host. The next host continues the algorithm, until the packet arrives at the final destination.

There are two different types of IP addresses which can be assigned to hosts:

- Public space addresses

  These addresses are officially assigned to a network by a Regional Internet Registry (RIR). Each address can only be used by one host, so it is possible to uniquely identify a host using a public space address.

- Private space addresses

  Private space IP ranges are reserved for usage within a private network. They may not be used and they are not routed on the public Internet. By using a Network Address Translation (NAT) device, it is possible to access the Internet using a private space address. This device converts the private space address on the private network to a public space address which can be used on the Internet. NAT is described in detail in Section 2.3.

  RFC 1918 [11] lists the IP ranges which are reserved as private space:

  - 10.0.0.0/8
  - 172.16.0.0/12

– 192.168.0.0/16

IP Header Figure 2.1 shows a possible header of an IP datagram. As the length of the variable part may differ, the header of an actual IP packet may look different.

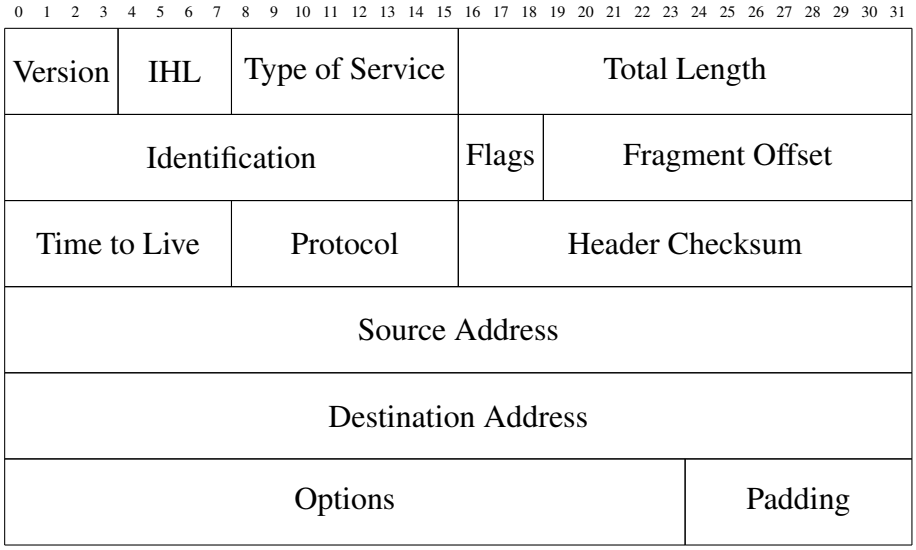

Figure 2.1: Internet Datagram Header

The fields in the packet are:

- Version

  This field contains the version number of the IP protocol used. Possible values are either 4 for IPv4 or 6 for IPv6.

- IHL

  This field contains the length of the IP header measured in 32-bit words.

- Type of Service

  This field is used by the sender to set the Type of Service. Routers on a network can use this information to treat the packets according to their type. Different Types of Service place different demands on the network. While an audio stream requires low delays and causes a medium throughput, for an FTP download delay characteristics do not matter and it requires a high throughput. A router can, but does not need to use the information in this field.

- Total Length

  This field contains the length of the full datagram – including header and data – measured in octets.

- Identification

  If a packet is split into multiple fragments, all the fragments share the same value of this field. The destination of the packet can use this field to reassemble the packets.

- Flags

  There are three different flags which can be set. The first flag is reserved and not used. The second flag `DF` (Don't Fragment) specifies, if the packet may be fragmented by an intermediate router. The third flag `MF` (More Fragments) indicates if this is the last fragment, or if more fragments will follow.

- Fragment Offset

  This field declares at which position in the datagram the fragment starts. The offset value is measured in units of 8 octets.

- Time to Live

  This field specifies the lifetime of a packet. If an entity on the network forwards the packet to another entity, this value is decreased by one. If an entity receives a packet with a Time to Live of zero, it must discard the packet.

- Protocol

  This field specifies the protocol which is used on the next higher layer. Historically the list of protocol numbers has been published as RFCs[3]. Today the protocol numbers are published in an online database which is available at the HTTP URL `http://www.iana.org/`.

  An extract of the protocol numbers specified:

  - 1: Internet Control Message Protocol (ICMP)
  - 6: Transmission Control Protocol (TCP)
  - 17: User Datagram Protocol (UDP)
  - 47: Generic Routing Encapsulation (GRE)
  - 50: Encapsulating Security Payload (ESP)

[3]The first RFC which contained protocol numbers was RFC 790 [12].

  - 89: Open Shortest Path First (OSPF)
  - 136: UDP-Lite

- Header Checksum

  The header checksum allows an entity to detect, if the header of the packet is corrupted. The checksum is calculated by using the 16-bit one's complement of the one's complement sum of all 16-bit words in the header.

  If a field in the header is modified, the checksum must be recalculated.

- Source Address

  The source address of the IP packet.

- Destination Address

  The destination address of the IP packet.

- Options

  The Options field is a variable length field, which may contain optional header information. The following options are possible:

  - Security
  - Loose Source Routing
  - Strict Source Routing
  - Record Route
  - Stream ID
  - Internet Timestamp

- Padding

  If the header does not end on a 32-bit boundary, a padding is inserted between the header and the payload. The padding aligns the beginning of the data to the next 32-bit boundary.

2.2.2 User Datagram Protocol

UDP is another protocol which is part of the TCP/IP stack. Unlike TCP, it is a stateless protocol. UDP allows applications to send and receive datagrams. It does neither guarantee that datagrams arrive in the correct order, nor that they arrive at all. UDP is defined in RFC 768 [13]. If a

datagram is lost, it is not retransmitted. UDP port numbers identify sockets on a host on which packets can be send and received. The width of a port number is 16-bit, there are 65.535 possible port numbers per IP address. Port numbers below 1024 are called well-known port numbers. The list of well-known port numbers is published at `http://www.iana.org/`.

Areas in which UDP is used are client-server Remote Procedure Call (RPC) protocols and real-time multimedia applications. In VoIP applications, retransmission of lost voice packets is often not reasonable, as it may take too long. This is one of the reasons why the voice data in VoIP calls is usually transmitted with a protocol which is based upon UDP.

In Figure 2.2 the structure of an UDP datagram is shown.

- Source Port

  This field contains the number of the source port at the host which sends the packet. This port number can be used by the recipient to reply to packets. Specifying a source port is optional – if no replies are expected, the value can be set to zero.

- Destination Port

  This field contains the number of the destination port at the host addressed in the packet.

- Length

  This field contains the length of the UDP header plus the length of the payload.

- Checksum

  This field contains the 16-bit checksum of the packet. In order to calculate the checksum, it is assumed that a pseudo header is prepended to the actual header. The fields of the pseudo header are taken from the IP packet in which the UDP packet is embedded. The format of the pseudo header is displayed in Figure 2.3. To build the checksum, the 16-bit one's complement of the one's complement sum of all 16-bit words in the pseudo header, the header and the payload is calculated.

2.2.3 UDP-Lite

The UDP-Lite protocol is based on the UDP protocol. While the checksum of an UDP packet covers all of the payload, UDP-Lite allows the checksum to be calculated only for a part of the payload. Such a partial checksum is useful in cases where recipients prefer a corrupt packet to

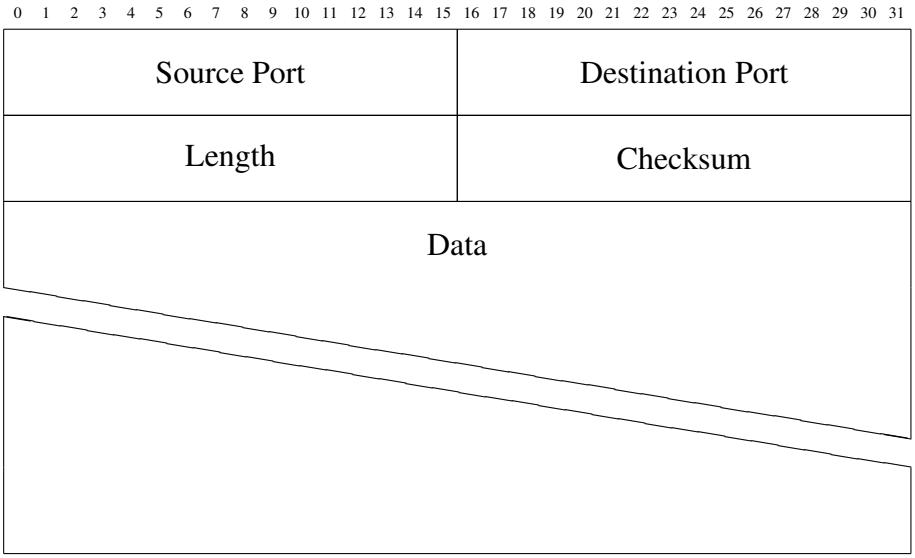

Figure 2.2: UDP datagram header

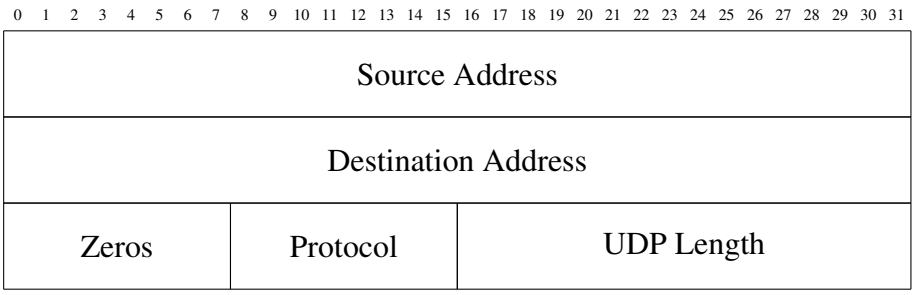

Figure 2.3: UDP datagram pseudo header

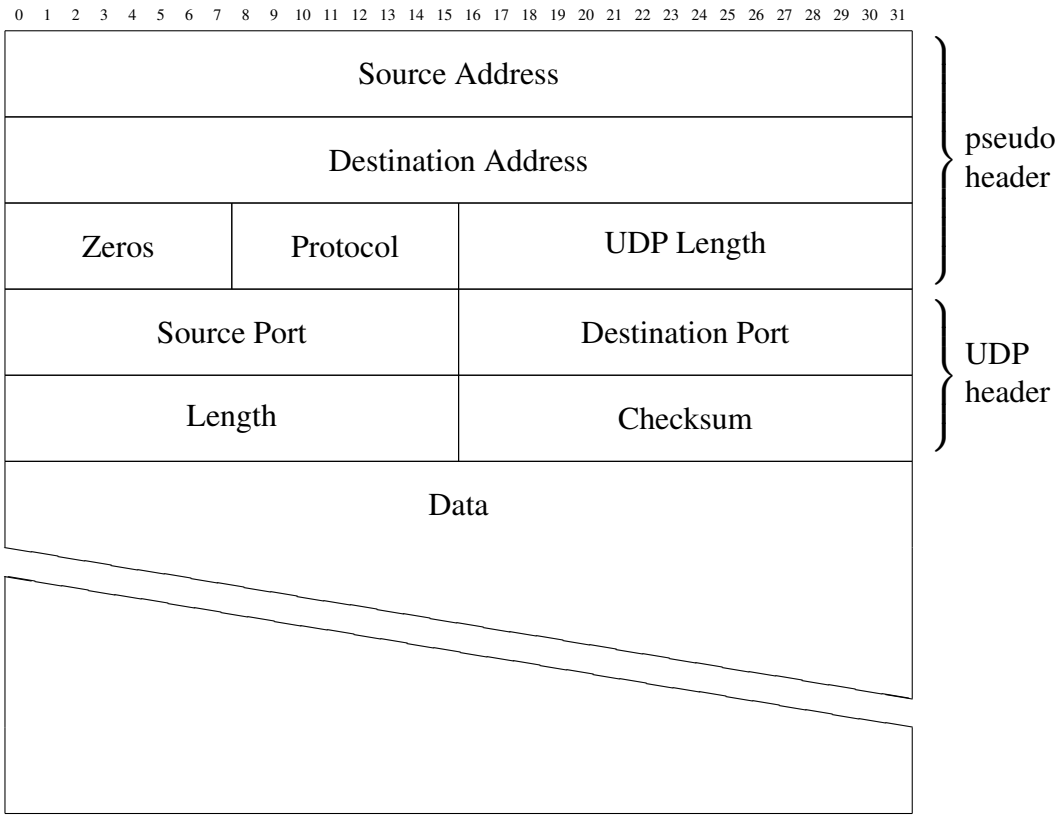

Figure 2.4: UDP datagram including the pseudo header as used for checksum calculation

a dropped packet. An example would be VoIP or audio streaming applications, where the audio codec is able to identify and correct bit errors by itself. UDP-Lite is defined in RFC 3828 [14].

The field Checksum Coverage which is shown in Figure 2.5 defines how the checksum is calculated. If the value is zero or equal to the length of the UDP packet, the checksum is calculated for the whole packet. Otherwise, the value defines the position in the packet until which the checksum is calculated. The UDP header, which is 8 octets long, must always be covered by the checksum, therefore values of the Checksum Coverage field from 1 to 7 are invalid. Packets containing these invalid values must be discarded by the recipient.

The pseudo header used by UDP-Lite is mostly the same as in UDP. The only difference is that the value of the Length field in the pseudo header is calculated by using information from the IP-header. In UDP this value is calculated by using the value of the Length field in the UDP header. In UDP-Lite this field is not available, as it has been replaced by the Checksum Coverage field.

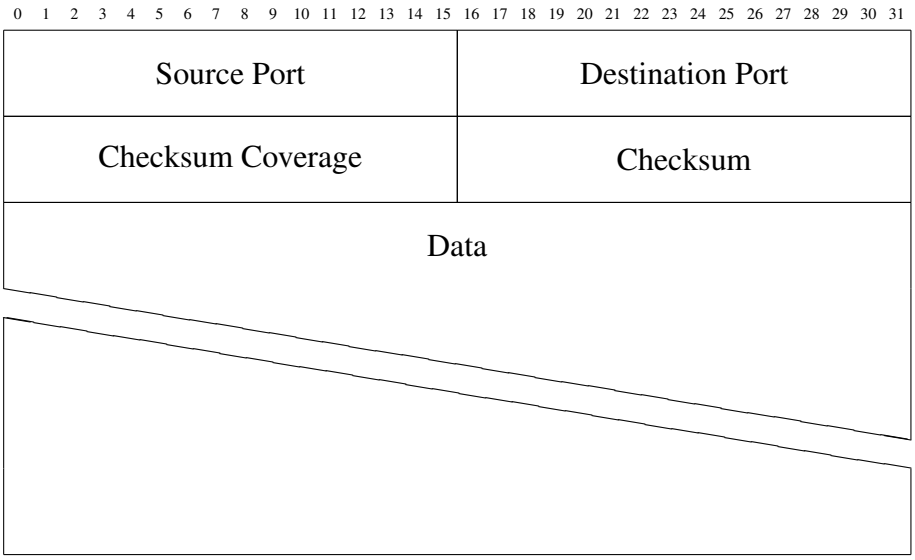

Figure 2.5: UDP-Lite Datagram Header

2.2.4 Transmission Control Protocol

TCP is a protocol located on the same layer as UDP. Unlike UDP, it provides retransmission of lost packets and reordering of out-of-order packets. Furthermore, it provides an end-to-end byte stream over the IP network. Applications do not send single packets, but are able to access the

network over a byte stream interface. They write a byte stream into a TCP socket and the TCP stack takes care of correctly transmitting the data over the underlying packet based IP network.

TCP was first published in RFC 793 [15]. Updates to the specifications are available in RFC 1122 [16]. RFC 1323 [17] introduces extensions which provide better performance on fast networks. RFC 2581 [18] and RFC 3168 [19] deal with congestion control and congestion notification.

A TCP endpoint is also labeled as a socket. A socket is identified by its 32-bit IP address and its 16-bit port-number. To transfer data between two hosts, a connection is established between the socket at the sender and the socket at the destination. Each socket may be used for multiple connections. TCP connections are full-duplex – messages can be sent in both directions. Figure 2.6 shows how a connection is set up. Figure 2.7 shows how the packets which are transmitted for a simple request-response protocol might look.

TCP uses congestion windows to prevent congestion [18]. A congestion occurs, when the sender sends data at a rate faster than the speed which can be processed by the network or by the recipient. The congestion window specifies the maximum amount of data, which may be sent to the network without receiving an acknowledgment. To send more data, the sender needs to wait until previous segments are acknowledged. The size of the congestion window is dynamically adapted to the characteristics of the underlying network.

The TCP protocol is based upon a state machine. A TCP connection is always in one well-defined state. The state `CLOSED` is a fictional state implicating that no connection exists. A real connection can therefore never be in the `CLOSED` state.

The following states are defined in RFC 793:

- `CLOSED`

  There is no connection state because no connection exists.

- `LISTEN`

  The socket is waiting for an incoming connection from any remote host.

- `SYN-SENT`

  The socket has already sent a connection request. It is waiting for a connection request from the remote endpoint.

- `SYN-RECEIVED`

  The socket has received a connection request. It has replied with a connection request and an acknowledgment of the received request. It is waiting for an acknowledgment (`ACK`) to its own request.

- `ESTABLISHED`

  The connection between two sockets is open. This is the normal state when data is transferred.

- `FIN WAIT 1`

  The local application wants to close the TCP connection. A connection termination request has been sent to the remote endpoint. The socket is waiting for either a connection termination request or an acknowledgment from the remote endpoint.

- `FIN WAIT 2`

  The socket is waiting for a connection termination request of the remote end.

- `CLOSE WAIT`

  The other side wants to close the connection. The socket is waiting for a connection termination request from the local user.

- `CLOSING`

  The socket is waiting for a connection termination request acknowledgment from the remote side.

- `LAST ACK`

  A connection termination request has been sent and received. Furthermore, an acknowledgment of the remote connection termination request has been sent. The socket is waiting for the remote acknowledgment.

- `TIMED WAIT`

  The socket waits some time to make sure that the remote host has received the acknowledgment of its connection termination request.

Figure 2.8 shows the structure of a TCP header. The following fields are part of the TCP header:

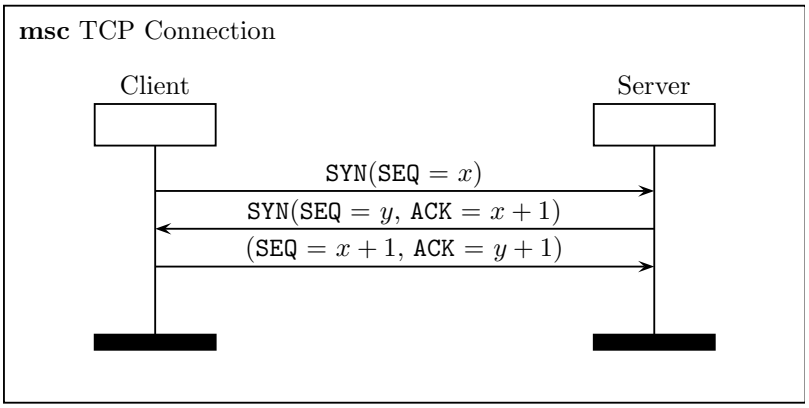

Figure 2.6: TCP connection setup

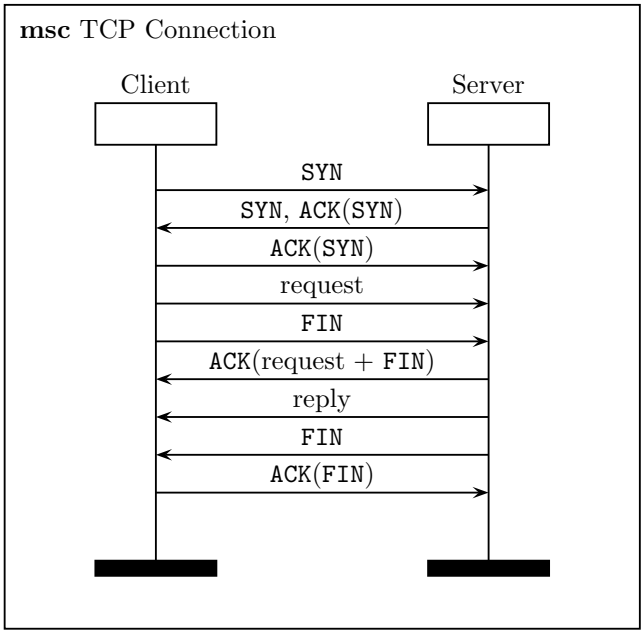

Figure 2.7: TCP connection setup, data transfer and shutdown

- Source Port

  The number of the source port.

- Destination Port

  The number of the destination port. A list of well-known port numbers is published by the Internet Assigned Numbers Authority at `http://www.iana.org/`.

- Sequence Number

  A 32-bit field containing the sequence number of the first byte in a packet. In case of an overflow, the sequence number cycles back to 0.

- Acknowledgment Number

  A 32-bit field which contains the value of the next sequence number which the sender of the packet is expecting to receive.

- Data Offset

  This field contains the length of the TCP header measured in 32-bit words. The value points to the beginning of the payload in the packet.

- Reserved

  This field has been reserved for future use. It must be set to zero.

- `URG` - Urgent Pointer field significant

  If the urgent pointer is used this flag is set to 1. Usually data is buffered at the source before being sent to the recipient. The urgent pointer instructs the TCP stack to stop accumulating data and to immediately sent all the data to the destination. When the urgent data is received at the destination, the receiving application is notified with a signal that urgent data arrived.

- `ACK` - Acknowledgment field significant

  If this flag is 1, it indicates that the packet contains an acknowledgment to a former packet. Otherwise the Acknowledgment Number field of the packet is ignored.

- `PSH` - Push Function

  When this flag is set, the remote host should deliver the data to the receiving application without buffering it.

Usually data is buffered by the TCP stack before delivering it to the application. In realtime applications it might be useful to deliver the data as soon as it arrives. An example is a terminal application which needs to react as soon as a key is pressed.

- `RST` - Reset the connection

  This flag is used to reset a connection when an error was detected.

- `SYN` - Synchronize sequence numbers

  This flag is used to establish connections. It is also set when an acknowledgment to a SYN packet is sent.

- `FIN` - No more data from sender

  An application sets this flag if it has no more data to send. It can still continue to receive data until the remote endpoint also sends a FIN.

- Window

  This field contains the number of octets which the sender is going to accept. They are counted from the byte which is acknowledged in the packet.

  Historically, the maximum window size that could be defined in this 16-bit field was 64 kb. On fast connections this window size is too small, as the TCP stack spends more time waiting for acknowledgments than sending the actual data. To resolve this issue, RFC 1323 [17] defines a window scaling option, which allows the sender to specify a scale factor which is multiplied to the window size.

- Checksum

  This field contains the checksum of the TCP packet. For the calculation of the checksum it is assumed, that the pseudo header shown in Figure 2.9 is prepended to the actual header. Before the checksum is calculated, the value of the checksum field itself is set to 0. The algorithm is the same as in UDP: The checksum is calculated by using the 16-bit one's complement of the one's complement sum of all 16-bit words in the header.

- Urgent Pointer

  If the urgent pointer flag is set, this field contains a byte offset from the current sequence number which contains the beginning of the urgent data.

- Options

  This field allows TCP to utilize options which are not part of the fixed TCP header. An example is the Maximum Segment Size (MSS) option. This allows a host to announce

the maximum segment size, that it is willing to accept, to the remote endpoint. Another example is the window scaling option which is defined in RFC 1323 [17].

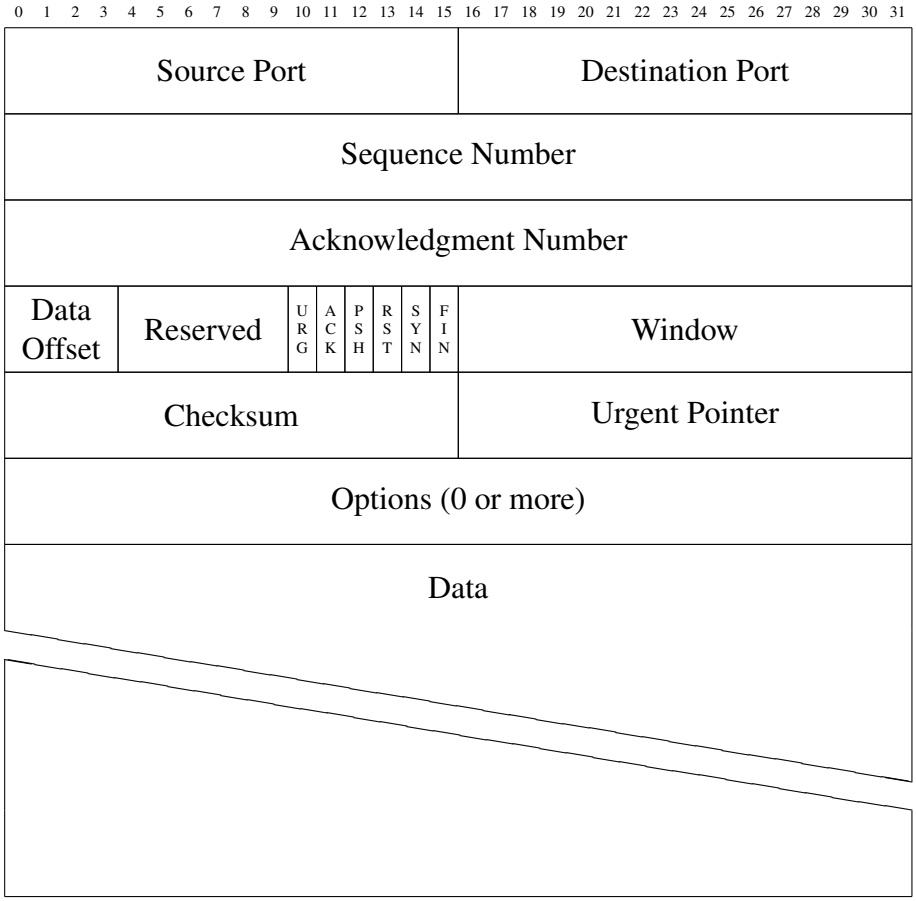

Figure 2.8: TCP Header

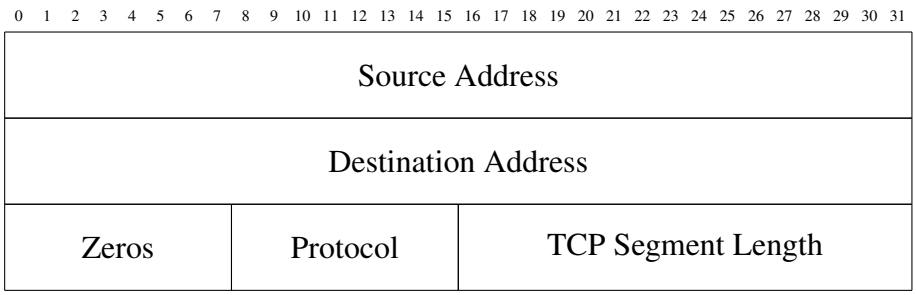

Figure 2.9: TCP Datagram Pseudo-header

2.2.5 Real-Time Transport Protocol

The Real-Time Transport Protocol (RTP) is designed to deliver audio and video data over the Internet. In VoIP applications RTP is usually used for the transfer of the audio data. As RTP packets are typically embedded inside UDP packets, it does not require any special support by the operating system or by NAT gateways. Most RTP libraries are implemented in userspace, RTP packets can be send to either unicast or multicast destinations. RTP was initially specified in RFC 1889 [20]. Later this RFC was obsoleted by RFC 3550 [21].

As RTP is used for realtime transfer of audio or video data, it does not implement flow control, error control or acknowledgment/retransmit capabilities of lost packets.

In order to enable applications to adapt the RTP protocol to their specific requirements, RTP allows to define profiles and encodings. A profile defines aspects which are left unspecified in the RTP protocol definition. Examples of these aspects are the meaning of the `Marker` field or the static default mapping of payload type values to payload formats. One of the available profiles is the RTP Profile for Audio and Video Conferences with Minimal Control which is defined in RFC 3551 [22]. An encoding determines how a particular payload type is stored within a RTP packet. An example of a payload type would be the Speex audio codec. As an encoding may be useful to multiple profiles, it is described independently from a specific profile.

In Figure 2.10 the structure of a RTP datagram is shown.

- Version (`V`)

  The field contains the version number of the RTP protocol used within the packet. According to the current standard, version number 2 must be used.

- Padding (`P`)

  This bit specifies if padding is enabled. When it is set, the last octet of the packet contains the number of padding octets appended to the payload.

- Extension (`X`)

  This bit specifies if a header extension is used. When it is set, the fixed RTP header is followed by exactly one header extension.

- Contributing Source (CSRC) count (`CC`)

  The field contains the length of the CSRC list that follows the fixed RTP header.

- Marker (M)

  The interpretation of the marker bit depends upon the RTP profile which is used. An example usage of the marker bit would be to mark frame boundaries.

- Payload type (PT)

  This field specifies the format which is used for the RTP payload. Examples for payload types are G.729 and GSM [22].

- Sequence Number

  This field contains the sequence number of the packet. For each sent RTP packet the sequence number is incremented by one. It allows the receiver to detect if any packets are missing or if packets have been received out-of-order. Missing packets are not retransmitted by the RTP protocol. If a missing packet is detected, the recipient can use techniques like interpolation to fill the missing gap.

- Timestamp

  The timestamp field contains the value of the sampling instant of the first octet of the payload. The timestamp does necessarily not need to represent the real time, but it must increase monotonically and linearly.

- SSRC

  The Synchronization Source (SSRC) field is used to identify the synchronization source.

- CSRC

  The Contributing Source (CSRC) field lists the sources which contributed to the payload. The width of each source is 32-bit. This field may appear from 0 to 15 times.

Real-time Transport Control Protocol

If a RTP server requires feedback about the transmitted information, the Real-time Transport Control Protocol (RTCP) is used. RTCP sends statistics about a session to all participants in the session. It does not transmit any multimedia data. An example usage is a client, which reports statistics about packet loss back to the server. The server can use this information to make an educated guess about congestion on the network. As a result, the server could decide to avoid further congestion by starting transmit at a lower audio and/or video quality.

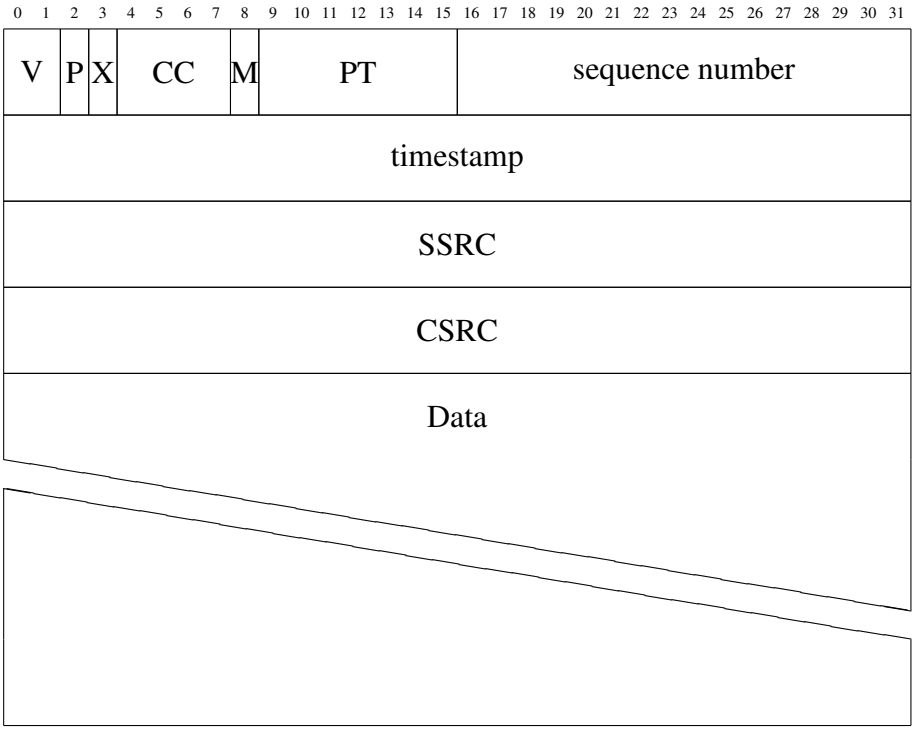

Figure 2.10: RTP Header

Encryption

In VoIP communication participants expect, that their communication is secure. Two different options allow the RTP and RTCP protocols to use encryption. The first one is the Secure Real-time Transport Protocol (SRTP) together with the Secure Real-time Transport Control Protocol (SRTCP). The second option is the ZRTP protocol.

- SRTP/SRTCP

  These protocols provide confidentiality, message authentication and replay protection. The SRTP framework is implemented as a profile which is an extension to the RTP Audio/Video Profile [22]. It is described in RFC 3711 [23]. Only SRTCP message authentication is mandatory, all other features are optional.

- ZRTP

  ZRTP uses opportunistic encryption which does not require the clients to exchange their public keys in advance. If two clients already know about each other, a shared secret is used to prevent man-in-the-middle attacks. If the two clients do not have a shared secret yet,

a Short Authentication String (SAS) is generated on each side. The short authentication string is a cryptographic hash of the two Diffie-Hellman values used for the encryption. The two participants can then compare their short authentication strings. If it matches on both sides, the chance for a man-in-the-middle attacks is unlikely. On subsequent communications the clients can cache the shared secret, so that the participants do not need to compare their short authentication strings again.

The advantage of ZRTP over SRTP is, that neither the participants need to exchange keys in advance, nor a certification authority is needed. In order to use encryption, only ZRTP software support by the clients is required, there is no need for additional infrastructure.

2.3 Network address translation

Network Address Translation (NAT) was initially designed in order to overcome the shortage of available IP addresses [24]. It operates by translating an address space used within an internal network into another address space used on the external public network. If a host on the internal network tries to communicate with a host on the public network, the NAT gateway translates the internal private-space address of this host to an address usable on the public network. On the internal network of a NAT gateway usually private space IP addresses [11] are used.

RFC 2663 [25] lists the following characteristics which are common among most of the current NAT implementations:

- Transparent address assignment

  NAT assigns IP addresses or port numbers temporarily to hosts on the internal network, if they try to communicate with hosts on the external network. Depending on the implementation, this assignment can be either static or dynamic.

- Transparent routing through address translation

  The NAT gateway connects the internal private network to the external public network. Therefore it is responsible for rewriting the IP addresses and port numbers in the IP packets. As modifications of the headers of an IP packet invalidate the old checksum of the packet, the gateway needs to recalculate the checksum.

- ICMP error packet payload translation

  Not only the actual TCP/UDP packets must be rewritten, but also ICMP packets[4] must be inspected. The NAT gateway also needs to rewrite the payload of ICMP packets which contain a related IP packet.

2.3.1 Types of NAT devices

There are two different flavors of NAT implementations. Basic NAT and Network Address Port Translation (NAPT). Basic NAT is able to dynamically assign external IP addresses to hosts on the internal network. However, it still requires one external IP address per active host. NAPT allows hosts on the internal network to share a single external IP address. Both variants are described in RFC 3022 [26]:

Basic NAT

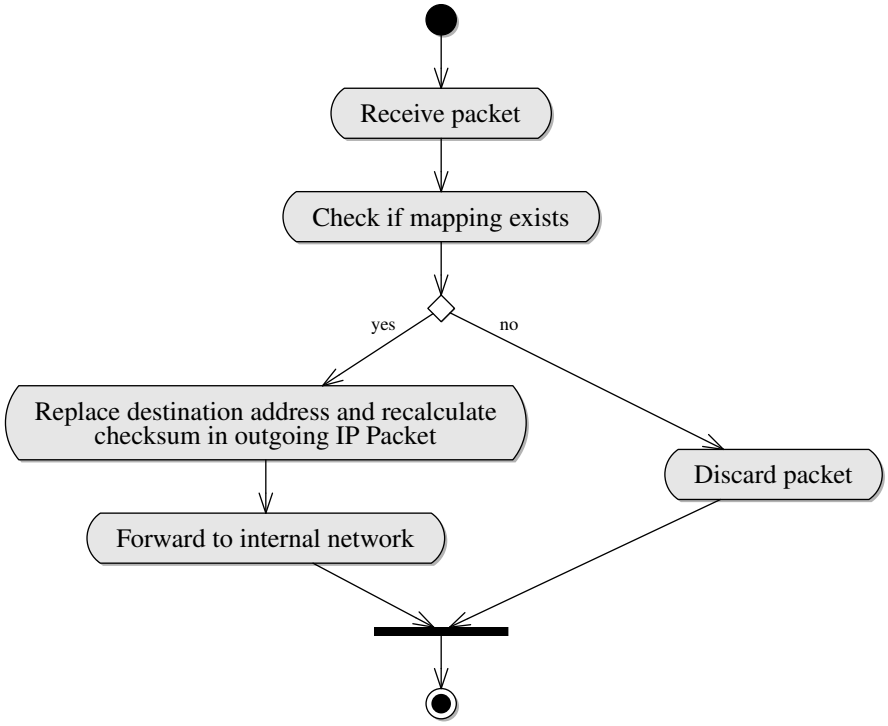

Figure 2.11: Activity diagram of incoming packet processing with Basic NAT

[4] with the exception of redirect messages

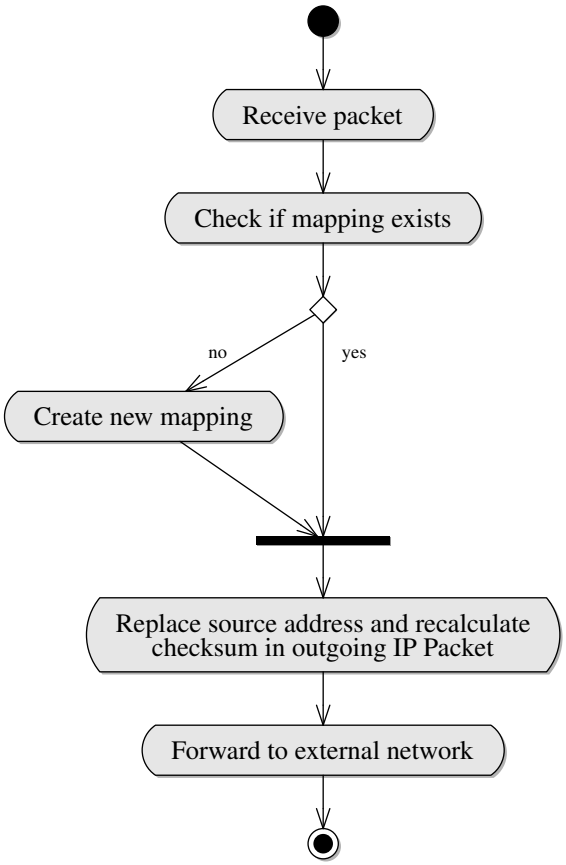

Figure 2.12: Activity diagram of outgoing packet processing with Basic NAT

The first NAT implementations used a method called Basic NAT, where each NAT gateway has a pool of reusable IP addresses. When an internal host sends a packet to the external network, a mapping (X,Y) between the internal address and one of the unused addresses in the pool is created by the NAT router. X represents the IP address of the host on the internal network and Y represents an IP address of the address pool.

When a packet traverses the NAT gateway in the outgoing direction, the gateway checks if a mapping (X,Y) already exists. If no such mapping exists, a new mapping (X,Y) is created. Y gets assigned one of the free addresses in the address pool. After the mapping is created or if the mapping already existed, the NAT gateway replaces the source address X of the packet by Y. Furthermore, the checksum of the IP packet is modified to account for the changed data in the header.

When the NAT gateway receives a packet in the incoming direction, it checks if a mapping (X,Y) exists for the destination address Y of the IP packet. If this is the case, the destination address is rewritten to X, the checksum is recalculated and the packet forwarded to the internal network. Otherwise the packet is dropped.

Network Address Port Translation (NAPT)

NAPT works similar to Basic NAT, but additionally to the IP addresses, also port numbers are used in the mappings. A tuple $((X,x),(Y,y))$ describes a mapping between an internal IP address X and a port number x to an external IP address Y and an external port number y.

The advantage of this method is that one single external IP address can be used by multiple internal hosts at the same time, as long as the external port numbers y_n differ from each other.

Today, most NAT devices usually implement NAPT, while Basic NAT implementations are rarely used.

2.3.2 NAT Classification

NAT devices can be divided into different classes. While all NAT devices solve the same problem, there are subtle differences in the heuristic, which decides if a packet should be forwarded to the internal network.

RFC 3489 [4] describes the following flavors which have been identified among UDP NAT implementations:

- Full Cone

  Once a binding $((X,x),(Y,y))$ has been created for an internal host (X,x), the same external address (Y,y) is also used if packets are sent to a different location than the one for which the binding was originally created.

  All external hosts can send a packet to (Y,y) which then gets forwarded to (X,x) – even if (X,x) did not send any packet to the particular host first.

- Restricted Cone

  Basically, this is the same as Full Cone, but with an additional restriction: If an external host (Z,z) wants to send a packet to the (Y,y) mapping, the internal host (X,x) must first have sent a packet to the IP address Z[5].

- Port Restricted Cone

  Together with the restriction introduced in the Restricted Cone variant, the packet, which the external host requires from the internal host prior to transmission must also match the port number z. This means, that (X,x) needs to send a packet to (Z,z) before (Z,z) is able to use the (Y,y) binding to contact (X,x).

- Symmetric

  A Symmetric NAT uses a binding (Y_n,y_n) for each packet which is sent from a source (X,x) to a destination (Z_n,z_n). This means, that for each new external $(IP, port)$ pair a new mapping is created.

  Unlike the other classes, symmetric NATs usually pose a problem to NAT traversal techniques. If two hosts want to communicate with each other, but each one is behind a symmetric NAT, they have no possibility to find out the values used in the (Y_n,y_n) tuple [27]. Therefore, they cannot directly communicate with each other, but need to use a third party relay.

Usage of this classification in the STUN protocol showed that the classification is too coarse. NAT implementations may exhibit a mixture between this characteristics or they may change their behavior depending on several factors.

[5] The packet does not need to be addressed to the port number z.

RFC 4787 – which was published by the Behavior Engineering for Hindrance Avoidance Working Group at the IETF – tries to overcome this problem by specifying a more detailed set of characteristics together with instructions describing which characteristic must be used in implementations.

Some of this characteristics are similar to the four variations described in RFC 3489. But while in RFC 3489 the conditions for creating new mappings and for forwarding incoming packets are combined, RFC 4787 describes mapping and filtering details independent of each other:

- Address and Port Mapping
  - Endpoint-Independent Mapping

    An endpoint-independent mapping reuses an existing mapping $((X,x),(Y,y))$ for each destination of (Z,z). This means that (Y_1,y_1) is equal to (Y_2,y_2) if (X_1,x_1) is equal to (X_2,x_2).
  - Address-Dependent Mapping

    An address-dependent mapping only reuses an existing mapping if the destination IP address is the same. (Y_1,y_1) is equal to (Y_2,y_2) if (X_1,x_1) is equal to (X_2,x_2) and Z_1 is equal to Z_2.
  - Address and Port-Dependent Mapping

    An address and port-dependent mapping adds the restriction, that not only the IP address but also the port number of the destination needs to be the same. (Y_1,y_1) is equal to (Y_2,y_2) if (X_1,x_1) is equal to (X_2,x_2) and (Z_1,z_1) is equal to (Z_2,z_2).
- Port Assignment
  - Port Assignment Behavior

    This specifies how the external port y of the NAT mapping is chosen. The two possibilities are:
    - Random Assignment
      In this case the port number y is randomly assigned from the set of unused port numbers.
    - Port Preservation
      In this case the NAT gateway tries to assign y to the same port number as x. In the case of collisions the gateway may either revert to Random Assignment or use Port Overloading which uses the same port number for two or more mappings.

Port Overloading is only possible if the destination tuples (Z,z) for a specific port differ from each other. If this is not the case, the NAT gateway would not be able to correctly assign incoming packets to a mapping.

  - Port Parity

    This specifies if an even port number is always mapped to an even port number and an odd port number is always mapped to an odd port number.

    The reason behind this characteristic is that some protocols require even or odd port numbers. As an example RTP must be used on an even port number and RTCP must be used on an odd port number.

  - Port Contiguity

    Port Contiguity assumes that applications which use RTP and RTCP will first use the RTP port and afterwards the RCTP port. Therefore, Port Contiguity will assign an even port number y_1 to the first mapping and an odd port number y_2 to the second mapping.

    The assumptions behind this characteristic are not valid in the general case.

- Mapping Refresh

  Specifies a timeout value after which an open mapping will be closed. The measurement of the idle time may use different values: Either the time of the last incoming packet, the time of the last outgoing packet or the time of the last packet in either direction.

- Conflicting Internal and External IP Address Spaces

  Describes if a NAT implementation is able to function correctly when the network address space on the internal interface and the network address space on the external interface are either equal or overlapping.

- Filtering Behavior

  - Endpoint Independent Filtering

    If a NAT gateway has an active mapping $((X,x),(Y,y))$ each host on the external network is able to reach (X,x) by using the (Y,y) binding.

  - Address Dependent Filtering

    If a NAT gateway has an active mapping $((X,x),(Y,y))$ a host (Z,z) on the external network is only able to reach (X,x) by using the (Y,y) binding if the mapping $((X,x),(Y,y))$ was used to send a packet to Z first.

  - Address and Port Dependent Filtering

    If a NAT gateway has an active mapping $((X,x),(Y,y))$ a host (Z,z) on the external network is only able to reach (X,x) by using the (Y,y) binding if the mapping $((X,x),(Y,y))$ was used to send a packet to (Z,z) first.

- Hairpinning Behavior

  Specifies if an internal host is able to reach another internal host via an active NAT mapping. E.g. if there is a mapping $((X_1,x_1),(Y_1,y_1))$ and (X_2,x_2) can reach (X_1,x_1) by sending a packet to (Y_1,y_1), the NAT supports Hairpinning.

- Application Level Gateways

  Some NAT implementations provide Application Level Gateways for some protocols. If a packet using these protocols is detected, the NAT gateway will not only modify packets at the IP layer, but also at the protocol layer.

  This allows the usage of protocols which would normally not operate through NATs.

- Deterministic Properties

  Specifies if the behavior of a NAT is deterministic, e.g. if the observed behavior of an implementation may change.

  An example would be an implementation which uses an Endpoint-Independent Mapping and Port Overloading as long as the destination endpoints differ in IP address or port number, but switches to Address and Port Dependent Mapping and Port Preservation as soon as a conflict is detected.

- ICMP Destination Unreachable Behavior

  Specifies if a NAT is able to correctly rewrite ICMP packets. This includes not only the rewriting of the IP headers of the packet itself, but also the rewriting of the headers of the packet, which is contained as payload in the ICMP packet.

- Fragmentation of Outgoing Packets

  Specifies if a NAT is able to send ICMP "Fragmentation needed and DF set" ICMP messages to an internal host if the packet size used by the internal host is too large. This could be the case, if the NAT gateway uses PPPoE[6] to connect to the external network.

- Receiving Fragmented Packets

[6] Point-to-Point Protocol over Ethernet

Specifies if a NAT is able to handle fragmented IP packets. This is not trivial, as the packet which contains the IP header may not be the first packet. Furthermore, it is also possible that individual fragments arrive out of order.

2.3.3 Drawbacks of Network Address Translation

While in an ideal world NATs would be completely transparent to an application, in the real world there are complications with several protocols, when they are used across NAT gateways. Client-server protocols – where the server is located outside the NAT and the client inside the NAT – cope pretty well with NATs. But peer-to-peer style applications, which need to open connections to a host inside a NAT, usually have problems [5].

The main reason for these problems is, that NATs only rewrite IP addresses used on IP level. If the protocol itself uses IP addresses within the protocol, the NAT gateway does not detect this. Therefore, connections to such IP addresses might fail.

Chapter 3

Traversal Techniques

Due to the complications described in Chapter 2.3 that are caused by NAT, several techniques are required in order to establish connections through one or more NAT devices. As it was shown that NAT implementations differ from each other, the optimal traversal strategy depends upon the implementation of the NAT device as well as the network topology.

There are different approaches to NAT traversal. Some techniques exploit the properties of NAT, in order to establish a direct connection across NAT devices. Other techniques assume that a reliable direct communication across NATs is not possible and therefore work around this problem by relaying the data over central servers. More sophisticated techniques use a combination of both methods: If it is not possible to establish a direct peer-to-peer connection between two peers, they fall back to a central server mechanism.

3.1 UDP Hole Punching

UDP hole punching [5] is a technique, which tries to exploit the behavior of NATs in order to dynamically create ports on the public NAT interface, which forward packets to an internal host. If a host on the internal network sends a packet to a host on the external network, a binding on the NAT interface is created. Depending on the type of NAT[1], only the external host which originally received the packet, or any other host may use this binding in order to contact the host on the internal network. Examples of protocols which employ UDP Hole Punching techniques are STUN [4] and ICE [28].

[1] the different classifications are described in section 2.3.2

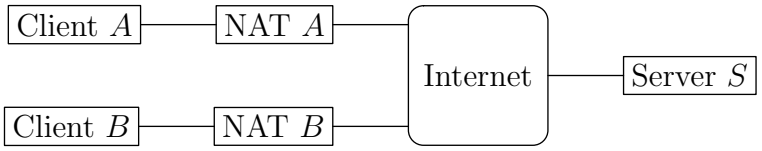

Figure 3.1: UDP Hole Punching

Figure 3.1 illustrates how UDP hole punching works for two clients A and B which want to open a direct UDP connection between each other:

- Both clients – A and B send a UDP packet to a server S on the public Internet.
- S waits until it received an UDP packet from both A and B.
- S replies to A with a UDP packet containing B's public address and port number and it replies to B with a UDP packet containing A's public address and port number.
- A and B try to establish a connection to each other with the contact information provided to them by S.

The result of this technique is dependent on the type of NAT devices in use. If both of the NAT devices exhibit the behavior of a Full Cone NAT, there are no problems with the establishment of the connection. In the case of a Restricted Cone or Port Restricted Cone NAT device, the additional restriction described in Section 2.3.2 – that the host within the NAT must first have sent a packet to the host outside the NAT – applies. Thus it is possible, that some of the first incoming packets are lost, if they are received by the NAT device before it has received an outgoing packet from the internal host. When symmetric NATs are used, this technique is not possible. The reason is, that for each destination (Z_n, z_n) a new binding (Y_n, y_n) is created. A technique which tries to guess the port number of the binding is described later in this chapter in Subsection 3.1.1.

When UDP hole punching is used there are several difficulties which need to be taken care of:

- Timeout of NAT Bindings

  As UDP is a stateless protocol [13], it is not possible for a layer 4 device to detect the end of a session. NAT devices use timeout values between two UDP packets as a heuristic to detect the end of a session. In order to guarantee that a NAT binding is kept open between two hosts, these hosts need to regularly exchange keep-alive packets in both directions within a time interval which is shorter than the timeout value of the NAT device [29].

- Hairpinning - Both of the hosts are located behind the same NAT

  If two hosts are located behind a common NAT gateway, an optimization to the hole punching algorithm is possible: Instead of establishing the connection across the NAT gateway, it is possible, that these two devices communicate directly to each other on the internal network. In cases where the NAT gateway does not support hairpinning, this technique is required in order to establish a connection between the two devices.

  In a possible implementation of this technique the hosts on the internal network include their internal IP address and port number in the packets which they send to the rendezvous server. If the rendezvous server detects, that the public IP addresses of both hosts are equal, it does notify the two hosts of their internal addresses instead of their public addresses. However, a problem with this technique occurs when multiple layers of NAT devices are used. In this case it is possible, that both of the hosts share the same public IP address although they are located behind different NAT gateways.

- Rewriting of IP addresses

  If workarounds for hairpinning are used, where the hosts within the internal network include their private space IP addresses in the packets, care must be taken to avoid that the NAT device modifies this addresses. Some NAT devices try to implement a generic way of rewriting packets, by detecting possible IP address and port number combinations within packets and rewriting them to the address and port number combination of the NAT binding, when forwarding the packet to the Internet. A simple solution for this problem is to use an obfuscation mechanism – like exclusive or'ing addresses in packets with another field – in order to avoid that the NAT device rewrites them [30].

3.1.1 Symmetric NATs

As symmetric NATs create a new binding for each new outgoing IP address and port combination, the UDP hole punching algorithm described in the previous chapter does not work with them. [5] describes a variation to this algorithm which does also work with symmetric NATs.

The trick is, that many symmetric NAT implementations assign the port numbers of the bindings in a predictable fashion. E.g. by increasing the port number of each new binding by one. By opening several outgoing connections to one or more rendezvous servers and by receiving replies by them about the used port numbers, the hosts inside the NAT can try to guess which port

number will be used if they open a new mapping. However, other hosts within the same network might interfere with this detection[2] or a port number might already be used[3]. Therefore, this method cannot guarantee the successful establishment of a connection. It is possible to improve this method by scanning a port range instead of a single port number.

The disadvantage of this method is, that the successful establishment of the connection depends on the exact implementation of the NAT device and on the allocation strategy for the bindings. Therefore, this algorithm is not guaranteed to work in every situation.

3.2 NAT Port Mapping Protocol (NAT-PMP)

The traditional ways to reach clients within a NAT are to either try to dynamically create a binding on the device by sending specially crafted outgoing packets or to manually configure a port forwarding on the NAT device. Both methods have drawbacks: Dynamically assigned bindings are often unreliable and manual configuration of the bindings requires additional effort from the user.

NAT-PMP is a UDP-based protocol currently in draft state at the IETF, which makes it possible for applications to automatically create port forwardings on a NAT device in a standardized way [31]. NAT-PMP was designed for small networks which are connected by a single uplink and where the NAT device is also the default gateway.

Figure 3.2 shows the format of one possible NAT-PMP query. The version field indicates the version of the used protocol. Currently only version 0 is specified. The opcodes from 0 to 127 indicate client requests. Opcodes from 128 to 255 are server responses. Requests by the client are sent to port number 5351 of the gateway device.

NAT-PMP allows clients to request the following functions from a NAT router:

- Request the public IP address of the NAT device

  Clients within the NAT are able to request the public IP address of the NAT device. To do this they send a packet with the format specified in Figure 3.2 to the gateway device. The gateway responds with a packet as described in Figure 3.3.

[2] because they open NAT bindings by themselves
[3] and therefore be skipped

- Announce changes of public IP address to clients

  If the address of the public interface of the gateway changes, it sends an announcement to the multicast address 224.0.0.1 and port number 5351. The format of the announcement is the same as the one shown in Figure 3.3.

- Create a port mapping on the NAT device

  To create a port mapping on the NAT device, the host inside the NAT sends a packet as described in Figure 3.4 to the NAT gateway. Requested Public Port is the port on the public interface of the NAT device and Private Port is the destination port number for the port forwarding. The opcode indicates if TCP or UDP packets should be forwarded. If it is 1 UDP packets are forwarded, if it is 2 TCP packets are forwarded.

  If the mapping was created successfully, the NAT gateway replies with a packet as shown in Figure 3.5.

- Destroy a port mapping on the NAT device

  In order to delete an existing port mapping, a host must create a new mapping with a lifetime of 0 seconds.

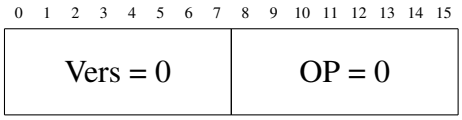

Figure 3.2: Packet sent by client to query the public IP address of NAT device

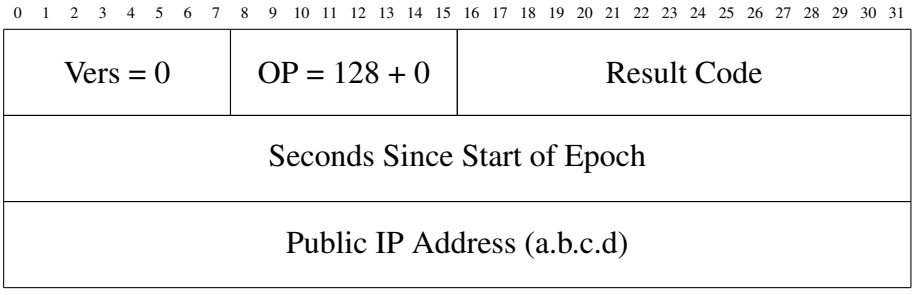

Figure 3.3: Response by the NAT device which contains the public IP address

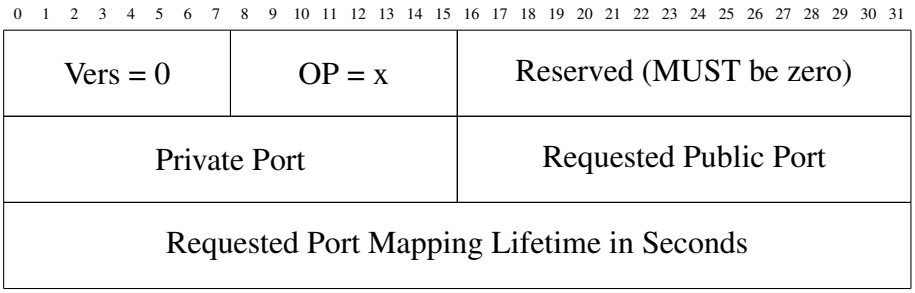

Figure 3.4: Packet sent by the client to request a new mapping on the NAT device

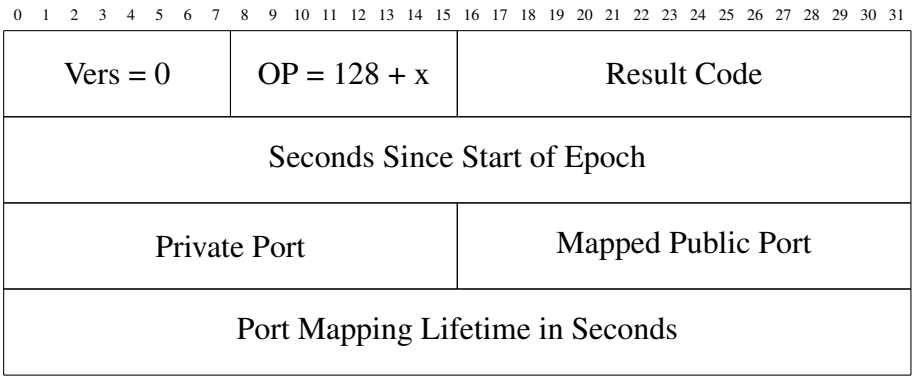

Figure 3.5: Response by the NAT device after the client requested a new mapping

3.3 Symmetric RTP

In the original design of the SIP protocol each client opened a port on the local interface as RTP port and announced it to its peer. After both of the clients received the port number of their peer, they started to send their RTP data to this port number.

The problem with this strategy is, that while it may be possible to initiate a direct RTP connection between these two clients[4], in some cases only one direction of the RTP stream will succeed and audio data will only arrive at one of the peers.

Symmetric RTP connections are RTP connections where the same source port is used for sending and receiving RTP packets. The advantage of symmetric RTP is, that the outgoing RTP connection opens a binding on the NAT router which can be used for incoming connections. If ports for sending and receiving would differ, the incoming RTP direction would not be able to use the binding of the outgoing RTP direction. Therefore, it would not be possible to send incoming packets to the client.

For NAT implementations which do not rewrite source port numbers, symmetric RTP may be sufficient as NAT traversal solution. For NAT implementations where this is not the case, two different mechanisms are used together with Symmetric RTP:

- TCP-Based Media Transport in the Session Description Protocol (SDP) [32]

  This protocol is an extension to the SDP protocol and allows to specify the direction of a TCP session. It does this by specifying a new attribute `setup`. The syntax for this attribute is `a=setup:<role>` where `<role>` is one of the following:

  - `active`

    The endpoint is going to establish an outgoing connection.
  - `passive`

    The endpoint is waiting for an incoming connection.
  - `actpass`

    The endpoint is able to establish an outgoing connection, but it is also willing to accept an incoming connection.

[4] E.g. when only one of the clients is located behind a NAT.

  - `holdconn`

    The endpoint does not want any connection to be established at all.

  While the protocol only defines the direction for TCP sessions, it is also used in the context of symmetric RTP [28]. An example is the VoIP client SJphone[5], which uses the `a=setup` field for UDP connections.

- Reflecting incoming RTP streams

  Another method, which is currently used by VoIP applications like Asterisk, YATE and OpenSER/RTPProxy, is to first send the RTP packets to the destination specified by the protocol. As soon as packets from the peer arrive, the source address of the peer is chosen as new destination address.

  There are two variations of these mechanism:

  - Hold back the own RTP packets until a packet from the peer arrives. Use the address of the peer as destination address for further packets.
  - First send the packets to the SDP destination. As soon as a packet has been received by the peer, change the destination address to the address of the peer.

  This technique is especially suitable when it is known, that one of the two endpoints is not located behind a NAT gateway. An example is a VoIP server which is located outside a NAT and which terminates the signaling as well as the media channel. As long as the NAT gateway is able to support UDP client-server connections, this technique should work.

  A drawback is, that this method does not support early-media[6] as the client needs to send RTP data to the server before the server is able to send data back to the client.

3.4 Media Relays

Media relays are a reliable and simple traversal technique which relays voice data across a server. The server is located on the public Internet – usually at the location of the operator of the VoIP service. If two clients want to communicate with each other, both of them open an outgoing

[5] SJphone version 1.61.321a includes the `a=setup:active` SDP field in `INVITE` messages

[6] E.g. a RTP stream with a ringing tone which is sent from the server to the client before the connection is established

connection to the relay server. Once both of the connections are established, the relay server starts to proxy the data sent between both connections [27].

One advantage of this method is, that it works with most NAT implementations, as it converts the peer-to-peer setup of the VoIP protocol to a traditional client-server setup.

A disadvantage is, that relay proxies introduce additional latency into the audio stream. Additionally, the bandwidth requirements can be considerable – especially when newer wideband audio or video codecs are used. Furthermore, media relays depend upon symmetric RTP connections in order to be able to traverse a NAT.

Currently there are two approaches which are used for Media Relays:

- Transparent Media Relay

  A transparent media relay is usually used to relay data over a media relay even though this is not intended by the respective protocol. Addresses of the remote peers are rewritten by a server in the signaling path, so that the client contacts the media proxy instead of the remote peer.

- Traversal Using Relay NAT (TURN)

  TURN is an Internet Draft by the IETF, which is expected to become a new standard for media relays. Instead of transparently rewriting IP addresses, clients themselves know about the TURN server and rewrite the addresses in the protocol accordingly.

  One advantage of TURN is, that it does not need to break the end-to-end principle by transparently modifying the data sent by the clients.

3.4.1 Media Relays at local SIP proxy/server

This describes a technique, where an intermediate node for the signaling – in the case of SIP this would be a SIP proxy – transparently modifies the address information in the signaling protocol, so that the RTP stream is not sent directly between two clients, but relayed through one intermediate node instead. This intermediate node is located on the public Internet.

Both clients send their RTP stream to the RTP proxy. By doing so, they open an UDP binding on the NAT router. The RTP proxy waits until it receives packets from both of the clients.

Afterwards, the RTP server can use the reflection technique explained in section 3.3 to relay the stream between the two clients.

The disadvantage of this method is, that an attacker may be able to hijack the connection by sending a RTP packet to the server before the legitimate client is able to do so. Precautions like dropping the connection if UDP packets are received from more sources than expected are able to solve this problem. However, this would make it trivial for an attacker to drop calls of other users.

3.4.2 Traversal Using Relay NAT

Figure 3.6 shows a typical network setup of a client using TURN. The TURN client is located behind a NAT and opens an outgoing TCP connection to the TURN server. External clients can connect to the TURN server in order to communicate with the TURN client. As soon as both peers – the TURN client and an external client – are connected to the TURN server, the TURN server starts to relay the data between them.

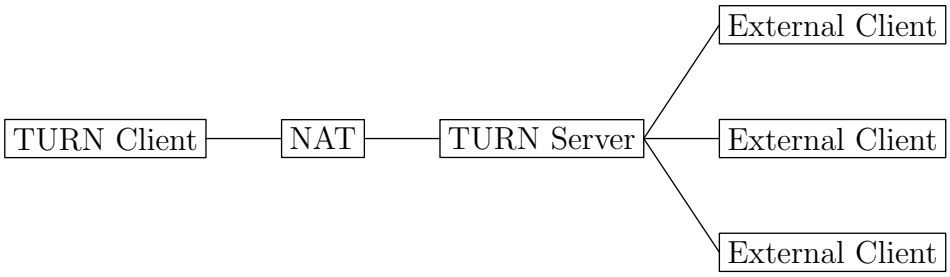

Figure 3.6: TURN network setup

3.5 Simple Traversal of User Datagram Protocol

Simple Traversal of User Datagram Protocol (STUN) allows a host on the internal network to detect the type of NAT device used on the network and to obtain a binding on the NAT device. The current version STUN is defined in RFC 3489 [4] and only deals with UDP traversal. However, there are proposals to extend STUN to TCP networking [33].

There are two types of requests:

- Binding Requests, sent over UDP

  Binding Requests are sent by the STUN client to the STUN server using UDP. The server replies to the Binding Request and includes the public port number and IP address of the client in its response packet. The client may use several parameters in its request in order to instruct the server to send the reply from different IP addresses and/or port numbers.

  After sending a number of specially crafted packets, the client can determine its public IP address, port number and the type of NAT it is located behind. In the case of a permissive NAT, this address information can be used by the client in order to be contacted by other clients.

- Shared Secret Requests, sent over TLS over TCP

  Shared Secret Requests are always sent over TLS. A client sends a Shared Secret Request to a STUN server and the STUN server replies with an username and a password. This information is used in the binding requests for authentication and message integrity. Shared Secret Requests must be sent over TLS – otherwise the server must return an error.

3.5.1 Message Encoding

STUN messages are TLV (Type-Length-Value) encoded. They start with a STUN header followed by the STUN payload, which contains a series of STUN attributes. A STUN attribute first contains a 16-bit type value followed by the a 16-bit length value. The value of an attribute can have a variable length. The format of a STUN attribute is shown in figure 3.7.

The list of valid STUN attributes is:

- 0x0001: `MAPPED-ADDRESS`

  This attribute is contained within the binding response and contains the source address and port number of the binding request packet.

- 0x0002: `RESPONSE-ADDRESS`

  This is an optional attribute of the binding request which indicates where the server should send the binding response. If the `RESPONSE-ADDRESS` is not given, the response is sent to the `MAPPED-ADDRESS`.

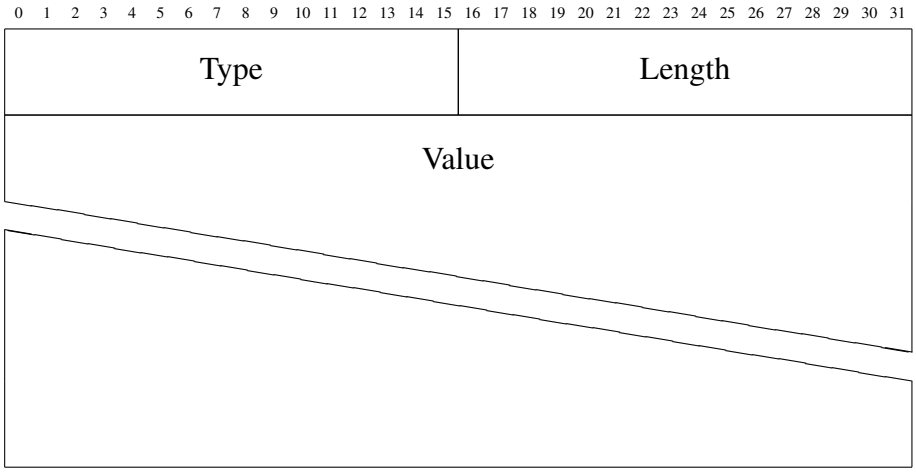

Figure 3.7: Format of STUN attribute

- 0x0003: `CHANGE-REQUEST`

  This is an optional attribute which allows the binding request to instruct the server to use a different source IP address and/or port number for the binding response. This allows the client to make assumptions about the type of NAT behind which it is located.

- 0x0004: `SOURCE-ADDRESS`

  This is part of the binding response and contains the source IP address and port number of the binding response. This allows the client to detect if the binding response packet was rewritten by a NAT.

- 0x0005: `CHANGED-ADDRESS`

  This is part of the binding response and contains the IP address and port numbers that would be used if a `CHANGE-REQUEST` attribute would be sent by the client.

- 0x0006: `USERNAME`

  The `USERNAME` is part of the binding request and shared secret response and provides a temporary username to the client.

- 0x0007: `PASSWORD`

  The `PASSWORD` is part of the shared secret response.

- 0x0008: `MESSAGE-INTEGRITY`

  This is part of both binding request and binding response and contains a message integrity check.

- 0x0009: `ERROR-CODE`

  This is part of the response messages by the STUN server and included if an error occurred. It contains the error code.

- 0x000a: `UNKNOWN-ATTRIBUTES`

  If mandatory attributes are missing in a request, the server answers with an `UNKNOWN-ATTRIBUTES` message which contains the list of this missing attributes.

- 0x000b: `REFLECTED-FROM`

  This is part of the binding response. It contains the IP address and the port number of the binding request.

3.5.2 STUN Tests

STUN works by conducting a series of connectivity test between the STUN client and the STUN server. The exact type of tests which are conducted depends upon the result of former tests. The activity diagram for the tests is shown in Figure 3.8.

There are three different types of tests which are conducted:

- `Test I`

  This test requests a response which is sent from the same address and port number where the request was sent to.

- `Test II`

  This test requests a response which is sent from a different address and port number than that where the request was sent to.

- `Test III`

  This test requests a response which is sent from the same address, but different port number than that, where the request was sent to.

These tests are conducted on two different STUN servers: `Server I` and `Server II`. It is possible to use two different STUN servers, or to use one STUN server which binds to two interfaces with different IP addresses.

As illustrated in Figure 3.8, the following tests are conducted by the STUN client:

- The first test requests an echo without any CHANGE-REQUEST. If this test does not succeed this means that UDP is blocked. Therefore, no UDP communication between the client and a host outside the NAT is possible.
- If the first test succeeded, it is checked whether the MAPPED-ADDRESS is the same as the source address. If this is the case, no NAT is in operation, but it may be possible that a firewall is located between the client and the server. If this is not the case, a NAT is detected.
- In the next step a test with a set CHANGE-REQUEST flag for the IP address and port number is conducted.

  If a firewall was detected in the former test:

  - A succeeding test indicates that no firewall is in use. Therefore communication is possible.
  - A failed test indicates that a firewall is active. Communication is not possible.

  If no firewall was detected in the former test:

  - If the test succeeds, this indicates that a Full Cone NAT was detected. Communication is possible.
  - If the test fails, further tests are conducted in order to detect the exact type of the NAT.
- In this test a response is requested from the second STUN server without any MAPPED-ADDRESS flag set. It is checked, whether the MAPPED-ADDRESS differs from the result of the first test.
  - If the test succeeds: Further tests are conducted to determinate the type of NAT.
  - If the test fails: A Symmetric NAT is detected. Communication is not possible.
- The last test uses MAPPED-ADDRESS to request a response from the same IP address but from a different port number.
  - If the test succeeds: A Restricted Cone NAT is detected. Communication is possible.
  - If the test fails: A Restricted Port NAT is detected. Communication is possible.

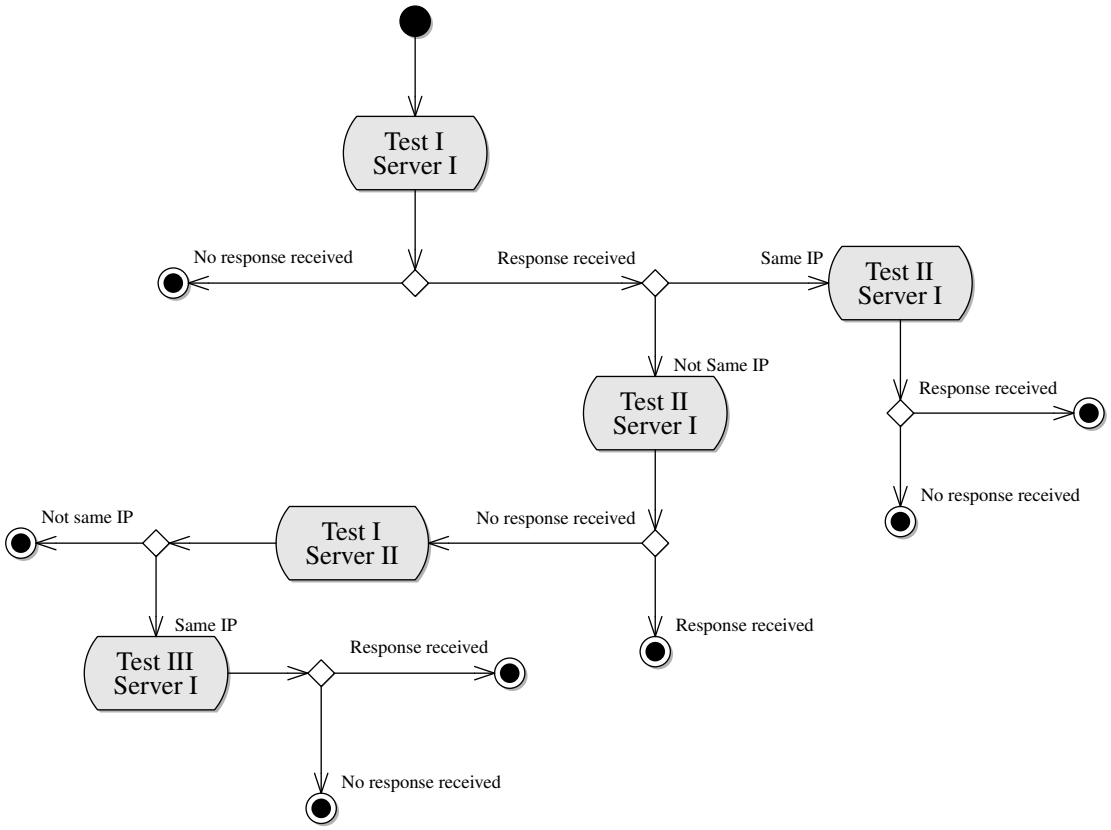

Figure 3.8: Activity Diagram of STUN algorithm

3.6 Interactive Connectivity Establishment (ICE)

Interactive Connectivity Establishment (ICE) is an extension to the SIP protocol which was introduced in 2003. It is currently in draft state [28], but already implemented in some SIP stacks. Notable ICE implementations as of 2007 are some SNOM phones [34] and the Eyeball AnyFirewall Engine [35].

In 2006, ICE was also adopted for usage in the XMPP Jingle protocol. The Jingle ICE extension is expected to be declared as Standards Track XMPP extension. As of today, the draft is still in experimental state.

While there are already several NAT traversal techniques used together with SIP, the best NAT traversal technique depends upon the network topologies of both the caller and the callee. Relaying of the RTP stream works in most cases, but has rather high bandwidth requirements. Using STUN in order to establish a direct connection between both peers would avoid these bandwidth requirements, but does not work in all situations.

ICE tries to identify one or more paths which can be used by the endpoints to communicate with each other. To each of these paths a specific priority is assigned. Usually, direct connections are preferred, while relayed connections are only used if there is no other alternative.

The advantage of ICE, when compared to older NAT traversal techniques used in SIP is, that ICE will always find the best path between two endpoints, if such a path exists. While in STUN a common problem is, that in some cases only oneway audio connections are possible, ICE should always be able to establish bidirectional connections.

3.6.1 ICE candidates

ICE tries to find the best path between two endpoints. In order to do this, both endpoints are working together to try out the available alternatives. In ICE terminology an IP address and port number combination under which a peer may be reachable is termed a candidate. A pair of two candidates – which indicates a possible path between the two endpoints – is termed a candidate pair.

There are several types of candidates:

- Host candidates: These candidates are obtained directly from a local interface, by binding to a port on this interface. If there is more than one interface on the host, candidates are obtained on each of these interfaces. These candidates are further divided into candidates which are obtained on physical interfaces (e.g. Ethernet cards) and candidates which are obtained on logical interfaces (e.g. VPN connections).
- STUN and TURN candidates

  There are two different types of candidates which are obtained by the usage of STUN and TURN:

  - Server reflexive candidates

    This candidate type represents an IP/port combination on the public site of a NAT.
  - Relayed candidates

    This candidate type represents an IP/port combination on a relaying server.

None of the endpoints knows in advance which of the candidates might be usable. This depends on the network setup and the network location of both of the endpoints. Two endpoints which are behind the same NAT might use an internal address to contact each other. A host which is outside this NAT cannot use such an internal address to contact a host inside the NAT.

To each of the candidates a specific priority is assigned. The priority will later be used to decide which candidate should be used if multiple options are possible. Candidates on local interfaces usually get a high priority, while relayed candidates only get a low priority.

After the caller has generated its list of candidates, it orders them from high to low priority and sends them to the callee. Afterwards, the callee generates its own list of candidates and sends them to the caller. Both endpoints pair up their own candidates with the remote candidates and generate a list of candidate pairs. Afterwards, they send a STUN request using these candidate pairs to check if connectivity is possible. While these check messages are sent, new candidates may be detected – e.g. if a host receives a packet from the remote endpoint which has not been a candidate yet. A new candidate is treated like a normal candidate. It is announced to the remote host and a connectivity check is done.

After this algorithm has finished, the ICE protocol returns the succeeding candidate information with the highest priority to the underlying signaling protocol. The signaling protocol then uses this information for the transmission of the media data.

Chapter 4

Voice over IP protocols

This chapter covers a selection of commonly used VoIP protocols. It begins with the Session Initiation Protocol (SIP) which is one of the most widely used VoIP protocols as of today. It continues with the Inter-Asterisk eXchange (IAX2) protocol, which was initially created for the usage in the Asterisk telephony platform[1]. Due to its operation across a wide range of NAT devices, it is now also used by other VoIP implementations. The chapter concludes with an introduction to the Jingle extension of the Extensible Messaging and Presence Protocol (XMPP). The latter is primarily used for instant messaging applications, however the Jingle extension adds VoIP capabilities to XMPP.

VoIP protocols are split into two parts: A signaling part, responsible for the setup and management of the voice call. This includes tasks like: initialization of a voice call, call transfer to another party and authentication to a VoIP server. And a media part which deals with the transmission of the voice data between two or more peers.

Both parts of the protocol are affected by NAT issues:

- The signaling part can either be implemented in a peer-to-peer style setup between to clients, or the data can be relayed across a central server. While peer-to-peer and UDP-based connections may be affected by NAT issues, newer protocols like SIP and XMPP allow to use client-server connections which are established using TCP.
- The media part can also be implemented in either peer-to-peer or client-server style. However, there are many benefits if the media part uses peer-to-peer style direct connections

[1] `http://www.asterisk.org/`

between the clients. One benefit is the lower round-trip delay of the media data. Furthermore, there is no need to reserve bandwidth at a central location which relays the media stream.

The protocols introduced in this chapter are signaling protocols. An example of a media protocol would be the Real-time Transport Protocol (RTP). It is described in Section 2.2.5. Except IAX2, all of the signaling protocols described in this chapter use RTP as media protocol.

4.1 Session Initiation Protocol

The Session Initiation Protocol allows to create and manage sessions between one or more participants. It can be used on top of several transport protocols including UDP and TCP. Proxy servers are used to route calls and to authenticate and authorize users. SIP was initially specified in 1999 in RFC 2543 [36]. The current version is described in RFC 3261 [37].

This section first describes the structure of the SIP protocol, followed by a discussion of SIP proxies and Back-to-Back User Agents (B2BUA).

4.1.1 Acronyms

- UAC - User-Agent Client

  This is a logical entity representing the part of an user agent which is acting as a caller.

- UAS - User-Agent Server

  This is a logical entity representing the part of an user agent which is acting as a callee.

- UA - User-Agent

  The UA represents the device at the end users location. The logical entities UAC and UAS are part of the UA.

- Proxy Server

  The primary task of a proxy servers is to forward messages between one or more clients. Apart from `ACK` and `CANCEL` requests, proxy servers cannot originate requests. Furthermore, proxy servers can rewrite specific parts of a request and decide which requests should be forwarded and which should be dropped.

Software implementations of SIP proxies usually include Registrar Service, Location Service and Redirect Service functionalities.

Proxy servers are subdivided into two different types:

- Stateless Proxies

  A Stateless Proxy does not keep information about messages. The routing decision is only based upon the content of each single message.

- Stateful Proxies

  A Stateful Proxy keeps track about the received and sent messages. One application of a Stateful Proxy is the retransmission of lost messages. The Stateful Proxy can use a timer to detect if it has received a reply to a message in time. If this is not the case, the proxy is able to retransmit the message.

  A special case of a Stateful Proxy is a Transaction Stateful Proxy. Unlike a Stateful Proxy a Transaction Stateful Proxy keeps the state only for a single transaction.

- Registrar Service

  The Registrar Service processes `REGISTER` requests. It extracts the location information contained within the `REGISTER` requests and stores them in the Location Service.

- Location Service

  The Location Service stores the set of possible locations of a client.

- Redirect Service

  The Redirect Service uses `3xx` responses to instruct the client to contact an alternate set of URIs.

- B2BUA - Back-to-Back User-Agent

  A B2BUA is defined as a combination of an UAC and an UAS entity which are connected back-to-back. When it receives a request it acts as an UAS. When the B2BUA answers a request it acts as UAC. It differs from a proxy by keeping the dialog state. Furthermore, it can originate other requests apart from `ACK` and `CANCEL`.

4.1.2 Protocol Design

The design of the SIP protocol is similar to both the Hypertext Transfer Protocol (HTTP) and the Simple Mail Transfer Protocol (SMTP). The request/response model, syntax and semantics of

the header fields and authentication mechanisms were taken from the HTTP protocol. Security mechanisms were taken from both HTTP and SMTP. However, not all parts of these protocols have been used exactly as in the original RFCs. For example, the chunked transfer encoding of HTTP/1.1 is not allowed in SIP. Furthermore, the SIP RFC does not extend these two protocols. While many parts of the HTTP and SIP specifications are similar, these two specifications are independent of each other. [37]

In Figure 4.3 the messages sent during a SIP call setup are shown. While it represents valid call setup, it does not include additional entities like SIP proxies, which are likely to be used in a real-world call setup. Client A starts the call by sending an `INVITE` message to Client B. Figure 4.2 shows the content of this `INVITE` message[2] in detail. Client B notifies Client A that it plays a ringtone by responding with a `180 Ringing` message. As soon as the callee answers the phone a `200 OK` message is sent from Client B to Client A. After Client A acknowledges this message with an `ACK` response, the media session is established between both clients. If one of the clients wants to close the session, it sends a `BYE` message to the other client. This message is acknowledged by an `200 OK` response.

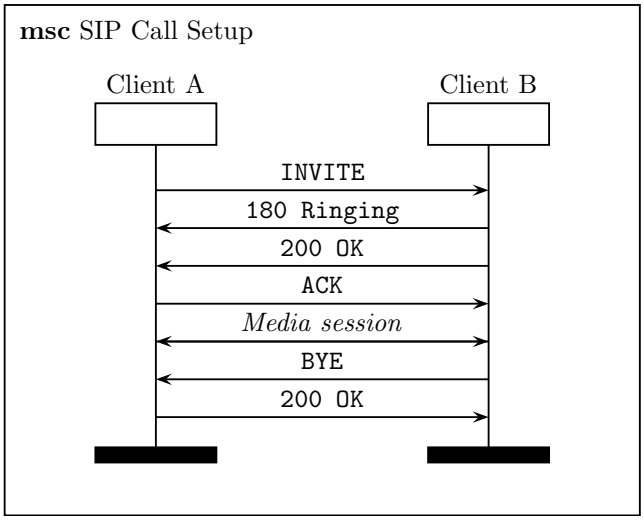

Figure 4.1: SIP call setup

The full list of methods and headers specified for SIP is available at `http://www.iana.org/assignments/sip-parameters`. As of September 2007, this list includes 95 headers an 14 methods, some of them specified as extensions to RFC 3261. As a full discussion of

[2]This message was sent by a SNOM 190 device, however for the sake of example, the `Max-Forwards` header field has been added manually.

```
INVITE sip:client-b@gst.priv.at SIP/2.0
Via: SIP/2.0/UDP 192.168.0.2:5060;branch=z9hG4bK17aee7a1;rport=5060
Max-Forwards: 20
From: <sip:client-a@gst.priv.at;user=phone>;tag=as4c7f201b
To: "Client B" <sip:client-b@gst.priv.at>;tag=tdog0uloyt
Call-ID: 3c2766e76419-rcwjn4pdyopg@snom190
CSeq: 102 INVITE
Contact: <sip:client-b@192.168.0.1:1088;line=c371zoiv>
Session-Expires: 3600
User-Agent: snom190/3.60k
Allow: INVITE, ACK, CANCEL, BYE, REFER, OPTIONS, NOTIFY, SUBSCRIBE,
PRACK, MESSAGE, INFO
Allow-Events: talk, hold, refer
Supported: timer, 100rel, replaces
Content-Type: application/sdp
Content-Length: 219

v=0
o=root 2049132700 2049132701 IN IP4 192.168.0.3
s=call
c=IN IP4 192.168.0.3
t=0 0
m=audio 4146 RTP/AVP 0 101
a=rtpmap:0 pcmu/8000
a=rtpmap:101 telephone-event/8000
a=fmtp:101 0-15
a=ptime:20
a=sendrecv
```

Figure 4.2: SIP `INVITE` message

the SIP protocol is outside the scope of this chapter, only fundamental parts of the protocol and parts which are relevant to Network Address Translation are covered.

SIP Methods

Each request message in the SIP protocol has a specific type, also termed as method. RFC 3261 [37] defines six methods: `INVITE`, `REGISTER`, `BYE`, `ACK`, `CANCEL`, `OPTIONS`. Further methods[3]are defined as extensions to the SIP protocol. [38]

The first line of each message contains the method name. It is followed by a list of header fields. The rest of the message may contain an optional body.

Each request contains a request URI, which identifies the destination of the request. The request URI does not necessarily need to be identical to the URI contained within the `To` header field. Unlike the `To` header field which remains the same for each session and must not be changed by proxies, the request URI may be modified by proxy servers. [39] An example is a proxy server which forks an incoming message to several destinations.

- `INVITE`

  An `INVITE` request is used by a client to initiate a media session. If the request arrives at a proxy it will be forwarded to the recipient. If it arrives at an UAS, there are three possibilities. The UAS can either:

  - Send a provisional response to the UAC (e.g. `180 Ringing`). A final response (accept or reject) will follow later.
  - Accept the request with a `2xx` response.
  - Reject the request with a `3xx`, `4xx`, `5xx` or `6xx` response.

  `INVITE` messages need to be acknowledged with an `ACK` message by the receiving party.

  The optional SDP body within an `INVITE` describes the session properties. It is possible to change them by sending a second `INVITE` message with the same `Call-ID`, `From` and `To` tags as the initial `INVITE`. Such a repeated invite is also termed as re-`INVITE`.

[3]including `REFER`, `SUBSCRIBE`, `NOTIFY`, `MESSAGE`, `UPDATE`, `INFO`, `PRACK`

- REGISTER

  The REGISTER method is used to register a client with a registrar service. The registrar service stores this information at the location service. A SIP proxy can use the information at the location service to route an incoming call to the client. Sending a REGISTER message is not required in order to place outgoing calls.

- BYE

  The BYE method is used to cancel an established session which has been acknowledged by either an 2xx or an ACK response. The BYE method should not be used for pending INVITEs.

- ACK

  The ACK method is used to acknowledge final responses[4] to the INVITE method. If the initial INVITE did not contain an application/sdp type body, this information may be included in the ACK message. If the INVITE request already included such a body, the ACK message is not allowed to contain a body with a different content.

- CANCEL

  The CANCEL method can be used to cancel a previously sent request. A CANCEL request is not guaranteed to succeed. If the recipient has already processed the initial request, it can ignore the CANCEL request. If the recipient is able to cancel the request[5], it responds with an 487 Request Terminated message.

- OPTIONS

  The OPTIONS method allows an UA to query another client about the options which it implements. The responding UA may set the Allow, Accept, Accept-Encoding, Accept-Language and Supported header fields, which give information about its capabilities.

SIP Headers

SIP messages must include the following header fields inside each request: Max-Forwards, To, From, Call-ID, CSeq. Furthermore, an INVITE message must also include the Contact header field [38].

[4] 2xx, 3xx, 4xx, 5xx, 6xx, but not 1xx

[5] e.g. if the initial request was an INVITE and the recipient is in ringing state

- `Via`

  If a SIP device sends or forwards a message, it includes its own address in the `Via` header field. The first part of the header field contains the version number of the SIP protocol together with the version number of the transport protocol which is used. The example in Figure 4.2 uses UDP as transport protocol, other possible options would be: TCP, TLS and SCTP. Separated by a space follows the address of the host which was inserting the `Via` header field. The `branch` parameter identifies the transaction. The `rport` parameter is defined in RFC 3581 [40] and allows a client to request that the replies to its messages should be sent back to the source address of the messages.

- `Max-Forwards`

  This field specifies the number of times a message may be forwarded. It is decremented by one each time a message is forwarded. If the value is zero, the message is dropped. One application for this header field is the prevention of loops.

- `From`, `To`

  These fields contain the originator and the destination of a SIP request. A `tag` parameter which contains random values may be attached to these fields for identification purposes.

  The `To` header field is intended for the remote UA. It does not necessarily need to be identical to the request URI. [39]

- `Call-ID`

  This field uniquely identifies a SIP session. It is set by the originator of the request. Like the message ID of an email message it contains a random part which is separated by an `@`-sign from an identifier for the device.

- `CSeq`

  This field contains a sequence number followed by the method name. The sequence number is incremented for each new request.

- `Contact`

  The `Contact` field contains a direct route to the originator of the request. It is usually composed of a username plus either a hostname or an IP address. This field tells other entities where future requests should be sent to.

- `User-Agent`

  This field contains information about the user agent which sent the message.

- `Allow`

  This field includes the list of methods which are supported by the sending device.

- `Supported`

  This field includes the list of extensions which are supported by the device which sent the message.

- `Content-Type`

  This field contains the message type of the body, if present. E.g. if the body contains information which was encoded in the Session Description Protocol, the corresponding message type is `application/sdp`.

- `Content-Length`

  This field contains the length of the body. If no body is present, the value of this field is set to zero.

- `Record-Route` and `Route`

  In SIP the `Contact` header field is used by clients in order to establish a direct signaling connection. However, there are cases where a SIP proxy might want to stay in the signaling path. One example would be a proxy which collects accounting information. Another example is a proxy which wants to modify or filter certain types of messages.

  The `Record-Route` and `Route` header fields allow a proxy to stay in the signaling path. If the proxy receives a request, it adds a `Record-Route` field with its address to the message. If the message is passed to a subsequent proxy, this proxy may append its address to the `Record-Route` field too.

  When the message arrives at the callee, the callee includes the `Record-Route` field in the response message. Furthermore, the callee also adds its own `Contact` field to the response.

  After the caller receives the response, it builds a `Route` header field consisting of the information within the `Record-Route` and `Contact` fields. The caller includes this header field in all subsequent requests. The `Route` field instructs proxies about the next destination to which they should forward the request.

SIP Response Codes

Response messages to SIP requests are distinguished from each other by a three digit status code. The first digit specifies the class of the status code. The remaining two digits provide a more exact identification.

The SIP protocol defines six different classes for status codes:

- `1xx`: Provisional

  These are informational responses, which inform the recipient that a message is currently processed. As soon as the processing is finished a final response needs to be sent. An example is the `180 Ringing` message which notifies the caller, that the phone at the callee is ringing.

- `2xx`: Success

  These status codes are used if an action has been successfully completed.

- `3xx`: Redirection
- `4xx`: Client Error
- `5xx`: Server Error
- `6xx`: Global Failure

4.1.3 Session Description Protocol (SDP)

The Session Description Protocol (SDP) is a format which allows to describe media sessions. While it was initially created to describe multimedia conference sessions on the MBone, its flexibility also allows the usage for different purposes. SDP was originally defined in RFC 2327 [41], the current version is defined in RFC 4566 [42].

SDP is not a transport protocol. The SDP RFC does not specify how SDP information should be distributed. Instead, SDP information is usually embedded within other transport protocols like SIP. It would also be possible to exchange SDP information by HTTP or email.

SDP is a text-based protocol. RFC 2327 recommends the use of the ISO 10646 character set in combination with UTF-8 as encoding, but it also allows other character sets such as ISO 8859-1.

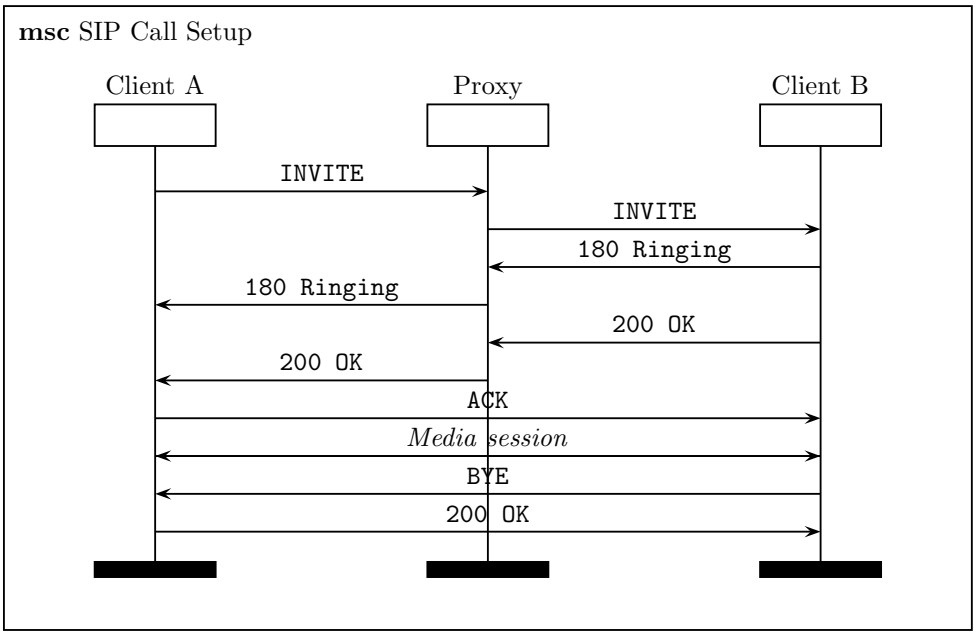

Figure 4.3: SIP proxy call setup

Field and attribute names however must be encoded using the US-ASCII subset of the UTF-8 character set. Field values may use the full ISO 10646 character set.

Figure 4.4 shows an example of a SDP message. The format of a SDP message is:

```
x=parameter1 parameter2 ...  parameterN
```

There must not be any spaces before or after the field name and field values are separated by exactly one space. Lines are terminated with CLFR. [38]

RFC 4566 defines the fields shown in Figure 4.5. Optional items are marked with an $\star$ character.

According to [38] the SDP protocol contains the following information about a media session:

- IP address

  This can either be a: IPv4 address, IPv6 address or a hostname.

- Port number

  The port number is used for the TCP and UDP transports.

```
v=0
o=root 2049132700 2049132701 IN IP4 192.168.0.3
s=call
c=IN IP4 192.168.0.3
t=0 0
m=audio 4146 RTP/AVP 0 101
a=rtpmap:0 pcmu/8000
a=rtpmap:101 telephone-event/8000
a=fmtp:101 0-15
a=ptime:20
a=sendrecv
```

Figure 4.4: SDP message

- Media type

  The type of the media data. E.g. audio or video.

- Media encoding scheme

  Specifies which encoding format is used for the media data. Examples are Speex or G.729.

The meaning of the fields in the SDP message shown in Figure 4.4 is [42]:

- `v=0`

  The `v` field contains the version number of the used SDP protocol version. Currently the only defined version is 0, so this field must be set to 0.

- `o=root 2049132700 2049132701 IN IP4 192.168.0.3`

  The `o` field (Origin) contains information about the originator of the session. It can be used to uniquely identify a session.

  The names and the meaning of the items in the `o` field are:

  `o=<username> <sess-id> <sess-version> <nettype> <addrtype> <unicast-address>`

  - `username`: Username on the originating host (may be set to "−" if the host does not use usernames).
  - `sess-id`: An ID for the session. The host which inserts the ID may decide how it is created. The SDP RFC recommends to use a Network Time Protocol (NTP) timestamp as session ID.

```
Session description
         v=  (protocol version)
         o=  (originator 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=* (zero or more bandwidth information lines)
         One or more time descriptions ("t=" and "r=" lines; see below)
         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, if present
         m=  (media name and transport address)
         i=* (media title)
         c=*
         (connection information -- optional if included at
          session level)
         b=* (zero or more bandwidth information lines)
         k=* (encryption key)
         a=* (zero or more media attribute lines)
```

Figure 4.5: SDP fields as described in RFC 4566 [42]

  - `sess-version`: A version number for the session. It should be increased when modifications to session data are made. It is recommended to use a NTP timestamp.
  - `nettype`: Currently only the `IN` nettype, which means Internet, is defined.
  - `addrtype`: Currently `IP4` for IPv4 and `IP6` for IPv6 are defined.
  - `unicast-address`: The address of the machine that created the session. Either an IPv4 address, an IPv6 address or a hostname.

- `s=call`

  The name of the session. If the session does not have a name, a single space should be used as value.

- `c=IN IP4 192.168.0.3`

  The `c` field contains information about the connection. The format is:

  `c=<nettype> <addrtype> <connection-address>`

  The format of `nettype` and `addrtype` is the same as in the `o` field. The format of `connection-address` is the same as the format of `unicast address` in the `o` field.

- `t=0 0`

  This field contains the NTP values of the start time and the stop time of a session.

  The format is: `t=<start-time> <stop-time>`

  A value of 0 for either `start-time` or `stop-time` means, that the value is not bounded. If both `start-time` and `stop-time` are set to 0, a permanent session is indicated.

- `m=audio 4146 RTP/AVP 0 101`

  The `m` field contains the media description.

  The format is: `m=<media> <port> <proto> <fmt> ...`

  - `media`: Specifies the media type in use. Possible values are: `audio`, `video`, `text`, `application` and `message`.
  - `port`: Specifies the port number which is used by the transport protocol.
  - `proto`: Specifies the transport protocol. Valid values are `udp` for any protocol which is based on UDP, `RTP/AVP` which denotes the Real-time Transport Protocol (RTP) under the RTP Profile for Audio and Video Conferences with Minimal Control [22] and `SRTP/AVP` which denotes the Secure Real-time Transport Protocol.

  - fmt: The fmt list contains additional information about the used media codecs. Usually the media payload types, which are defined for RTP, are used.

- a=rtpmap:0 pcmu/8000
- a=rtpmap:101 telephone-event/8000
- a=fmtp:101 0-15
- a=ptime:20
- a=sendrecv

  The attribute fields may contain additional information about the media session. E.g. the rtpmap fields map a RTP payload type to an encoding name. The sendrecv value indicates that media should be sent and received (other options would be recvonly and sendonly.

SIP and SDP combination

The SDP protocol was initially designed to carry information about multicast sessions. As a consequence, it misses some of the features required for unicast sessions. RFC 3264 [43] (An Offer/Answer Model with the Session Description Protocol (SDP)) is an extension to SDP which adapts it to unicast sessions, like media calls established with the SIP protocol [38].

In the offer/answer model described in RFC 3264, one party sends a SDP message which includes the media streams, codecs, IP addresses and port numbers which it wants to use. The answer by the other party includes a SDP message with the same list of media streams. For each media stream it indicates, if it accepts or if it rejects the offer.

Like in the RFC 2327 and RFC 4566 specifications, RFC 3264 still requires SDP information to be transported by a higher level protocol. In SIP, the SDP information is embedded within the SIP messages. The content type of SDP data is application/sdp. Not all of the mandatory fields in SDP have a meaning when used together with the SIP protocol. For compatibility reasons these fields are still required.

The SIP messages which may carry SDP information are: INVITE, PRACK, UPDATE plus the reliable types of 18x and 200 responses (e.g. ACK). Usually the caller includes its capabilities in either the INVITE or the ACK messages, while the callee includes its capabilities in the 200 OK response.

If the caller includes the request in the initial `INVITE` message, the response is sent by the callee. The response is then included by the callee in the `200 OK` response. A caller can allow a callee to send the SDP request by not including a SDP request in the initial `INVITE` message. If this is the case, the callee includes the SDP offer in the `200 OK` response to the `INVITE` request and the caller replies with the SDP request in the corresponding `ACK` [44].

The SDP message shown in Figure 4.4 was created for a SIP unicast session. The `o` (origin), `s` (subject) and `t` (time) fields are included for compatibility reasons. The origin is set according to RFC 4566, the subject can be set to a random string with a minimum length of one character (some implementations use the string `call`). The start time as well as the end time are set to zero in the response.

The SDP fields which are used inside SIP are `v` (version), `c` (connection) as well as the `a` and `m` fields carrying media attributes.

Either caller or callee may put the other side on hold by sending a SIP re-`INVITE` with new SDP information. An `a=sendonly` attribute puts the call on hold. To activate the call again a new SDP message with an `a=sendrecv` attribute is sent.

SDP and NAT Traversal

While the base SIP RFC does not mention any NAT traversal techniques, there are several extensions to SIP which add NAT traversal capabilities. These extensions do not exclude each other. Each of these extensions has been written to solve one single issue. One extension which is specified in [32] is described in Section 3.3. It allows the usage of symmetric RTP streams.

Another extension is the `rport` extension, which is specified in RFC 3581 [40]. The `rport` extension allows a client to request, that the response to SIP signaling requests is sent to the source IP address and port number of the respective UDP packet instead of the address contained within the `Via` header. In order to use `rport`, a client must add a parameter called `rport` to the `Via` header.

If the SIP server does not support the `rport` extension, it ignores the `rport` parameter. If the SIP server supports the extension and it receives a `rport` parameter with no value, it must set the value of the parameter to the source port number of the UDP packet containing the request.

Furthermore, the server must also add a `received` parameter to the `Via` header which contains the value of the source IP address of the initial UDP packet.

If a server sends a response, it checks if both a `rport` parameter and a `received` parameter are present. If this is the case, it sends the reply to the IP address and port number specified within these parameters. When forwarding the response, the server must use the IP address and port number on which it received the initial request as source address. This is a requirement for the traversal of symmetric NATs.

4.1.4 ISP Setup

The SIP protocol does not require any central servers. SIP clients which implement UAC logic as well as UAS logic are able to directly communicate with each other. In an actual setup however, often one or more servers are located between two SIP clients. One of the reasons for this is naming. Two clients which want to communicate with each other need to locate each other. Without a central server a client would need to know the network location of the other client. On today's Internet, where many users get assigned dynamic IP addresses and where users utilize wireless networks to connect from different locations, calling a contact by his network address quickly becomes impractical. Another reason are additional services. If a client is not reachable, a server may connect the caller to a mailbox where he can leave a message. Without a server such a setup would not be possible.

This work will describe two typical setups which are used at providers of SIP services. The first approach uses a proxy server. The second approach uses a PBX which internally connects two or more calls.

Proxy setup

In this setup the client registers at a SIP proxy. This SIP proxy implements a registrar service and a location service. By registering at the proxy server, the user is reachable via a SIP URI like `username@proxyname`. If SIP messages for the user arrive, they are forwarded by the proxy server to the client of the user. If the user places a call, he may either use the proxy as outgoing proxy, which means that all SIP messages are sent through this proxy, or he may contact a remote

SIP URI directly. A proxy may also fork a call, which means that a single `INVITE` is forwarded to multiple destinations.

Figure 4.6 shows a setup where two clients connect to different proxy servers, Figure 4.7 shows a variation where these two clients connect to the same proxy server.

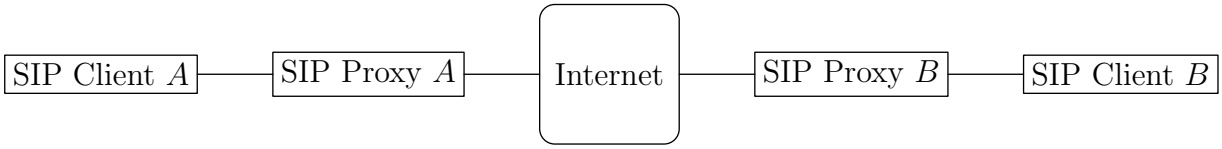

Figure 4.6: Two SIP clients connected to different SIP proxies

Figure 4.7: Two SIP clients connected to the same SIP proxy

Besides routing tasks, a proxy server is allowed to make modifications to the messages in transit. Furthermore, it may also translate between different transport protocols. E.g. it may accept SIP messages on a TCP socket and forward them to an UDP socket.

Usually only signaling data is transferred via SIP proxies. However it is possible for a proxy to modify the SDP body in order to allow two clients which are located behind a NAT to open connections to each other. This works by modifying the SDP body in a way that causes both clients to contact a media-relay server on the public Internet, instead of each other. This technique is described in detail in section 3.4.1.

PBX / switch setup

In this setup a PBX is used instead of proxy servers. A PBX is similar to a B2BUA by including an UAC as well as an UAS part. Unlike a proxy, a PBX does not route SIP messages between two clients. A PBX acts as UA and terminates a SIP call. A client typically uses the PBX as outgoing proxy and forwards all packets to the PBX.

SIP Calls which arrive at the PBX are terminated at the PBX. Depending upon the destination of the call the PBX decides how to continue. One possibility would be to open a call to another destination based on the initial `To` destination of the call. If this second call is successful, the PBX can internally switch these two calls together, so that they are connected to each other. This

usually means, that the PBX forwards the audio data between the two endpoints. The second call does not necessarily need to be established using the SIP protocol. If different voice codecs are used for the two calls the PBX is able to translate between them.

The advantage of a PBX over a proxy is that it has more control over a call. While a proxy can only route SIP messages, a PBX can e.g. also present an Interactive Voice Response (IVR) menu to the user. Furthermore, a PBX may also translate between different VoIP protocols or calls to the Public Switched Telephone Network (PSTN).

However, a disadvantage of PBXs compared to proxies is, that they violate the end-to-end principle explained in [2]. A proxy just forwards SIP messages between two endpoints, allowing the two endpoints to use extensions to the SIP protocol which are not known to the proxy server. A PBX terminates each of the calls. Unsupported headers inserted in SIP messages are usually not included in the outgoing calls.

One example, where the violation of the end-to-end principle leads to an actual problem is the ICE protocol, which would allow the endpoints to find the best way to traverse a NAT. While ICE does work well with SIP proxies, ICE NAT traversal between two clients is not possible with PBXs which do not support the ICE protocol. Examples are the commonly used PBXs Asterisk and YATE.

4.2 Inter-Asterisk eXchange

The Inter-Asterisk eXchange (IAX2) protocol was first created for the usage in the Asterisk PBX. Unlike other signaling protocols, IAX2 uses a single UDP port[6] for both signaling and media data. As media data is also transferred by the IAX2 protocol, the RTP protocol is not used [45].

The use of a single NAT port has the advantage that NAT problems are less common. If signaling is possible between two endpoints, media transfer will usually succeed as well, because exactly the same UDP port numbers – and therefore the same NAT mappings – are used.

[6] 4569 is the IANA assigned port number for IAX2

4.2.1 Protocol Specification

IAX2 is a binary protocol which is specified in [45]. IAX2 messages are termed as frames. There are two different types of frames: full frames and mini frames. The `F` bit – which is the first bit of a message – is used to differentiate between full frames and mini frames. If the `F` bit is set to 1, this indicates that the frame is a full frame. Otherwise the frame is a mini frame. Usually full frames are used for control messages and mini frames are used for media data. For synchronization purposes, full frames are sometimes used for media data instead of mini frames. Both frame types are sent in a single UDP stream using the same port numbers.

The advantage of full frames is, that they provide retransmission capabilities in case of lost packets. Furthermore, they can also contain control messages. Mini frames are kept simple in order to save bandwidth. Because mini frames do not use sequence numbers, retransmission of lost frames is not possible. As mini frames are only used for media transmission, this is acceptable[7].

4.2.2 Full Frame

Figure 4.8 shows the binary structure of a full frame. The fields are described in figure 4.9.

Full frames are used for the reliable transmission of information. They include sequence numbers. Non acknowledged frames are resent. An acknowledgment of a frame can either be implicit – an example would be an `ACCEPT` message which is sent in response to a `NEW` message. Another option are explicit acknowledgments which are send as `ACK` messages – they are used if no other response packet is sent by the client in return.

Full frames are split into frame types which are identified by the frame type field in the full frame. The list of available subclasses is shown in figure 4.10.

Control frames

Control frames which are identified by a subclass value of $0x04$ in the `C` field of a full frame, are used for session control. This includes tasks like: call setup, call teardown and transfers. The list of possible control frame values is shown in figure 4.11.

[7] Retransmission of media data is not reasonable, as it is likely to take too long.

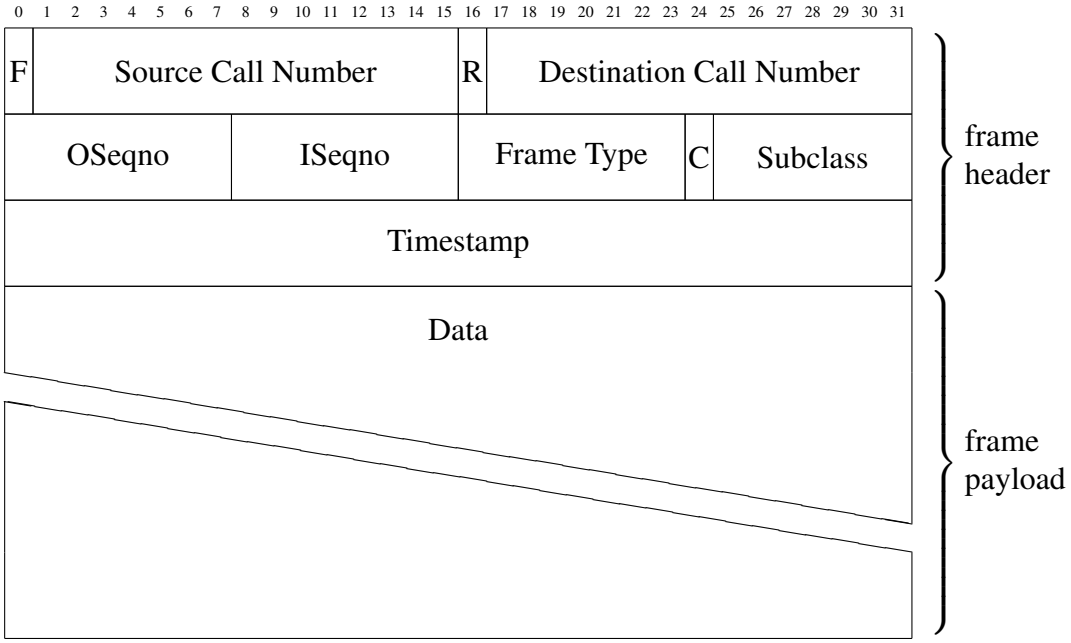

Figure 4.8: IAX2 full frame binary format as described in [45]

field	description
F	1 for full frame, 0 for mini frame
Source Call Number	number of the caller
R	1 if the frame is retransmitted, otherwise 0
Destination Call Number	number of the callee
Timestamp	32-bit timestamp
OSeqno	Outbound stream sequence number. This number always starts at 0 and increases by 1 for each packet.
ISeqno	Inbound stream sequence number
Frame Type	frame type
C	Subclass value format

Figure 4.9: IAX2 full frame field description as described in [45]

Frame type	Description
0x01	DTMF
0x02	Voice Data
0x03	Video
0x04	Control
0x05	Null
0x06	IAX2 Control
0x07	Text
0x08	Image
0x09	HTML
0x0a	Comfort Noise

Figure 4.10: IAX2 control frame subclass values as described in [45]

IAX2 frames

IAX2 frames which are identified by a subclass value of $0x06$ in the C field of a full frame, are used for IAX2 signaling. Unlike control frames, they do not handle session specific control. The list of possible IAX2 frame values is shown in figure 4.12.

Subclass	Description
0x01	Hangup
0x02	Ring
0x03	Ringing (ringback)
0x04	Answer
0x05	Busy Condition
0x08	Congestion Condition
0x09	Flash Hook
0x0a	Wink
0x0b	Option
0x0c	Key Radio
0x0d	Unkey Radio
0x0e	Call Progress

Figure 4.11: IAX2 control frame subclass values as described in [45]

Subclass	Mnemonic	Description
0x01	NEW	Initiate a new call
0x02	PING	Ping request
0x03	Reserved	
0x04	ACK	Acknowledgement
0x05	HANGUP	Initiate call teardown
0x06	REJECT	Reject
0x07	ACCEPT	Accepted
0x08	AUTHREQ	Authentication request
0x09	AUTHREP	Authentication reply
0x0a	INVAL	Invalid call
0x0b	LAGRQ	Lag request
0x0c	LAGRP	Lag reply
0x0d	REGREQ	Registration request
0x0e	REGAUTH	Registration authenticate
0x0f	REGACK	Registration acknowledgement
0x10	REGREJ	Registration reject
0x11	REGREL	Registration release
0x12	VNAK	Video/Voice retransmit request
0x13	DPREQ	Dialplan request
0x14	DPREP	Dialplan response
0x15	DIAL	Dial
0x16	TXREQ	Transfer request
0x17	TXCNT	Transfer connect
0x18	TXACC	Transfer accept
0x19	TXREADY	Transfer ready
0x1a	TXREL	Transfer release
0x1b	TXREJ	Transfer reject
0x1c	QUELCH	Halt audio/video transmission
0x1d	UNQUELCH	Resume audio/video transmission
0x1e	POKE	Poke request
0x1f	PAGE	Paging call description
0x20	MWI	Message waiting indication
0x21	UNSUPPORT	Unsupported message
0x22	TRANSFER	Remote transfer request

Figure 4.12: IAX2 frame subclass values as described in [45]

4.2.3 Mini Frame

For media data usually mini frames are used. They are simpler and therefore require less bandwidth. Figure 4.13 and 4.14 show the bit structure of a mini frame. A notable difference to a full frame is, that the timestamp does only have a width of 16-bits. These are the lower 16-bits of a timestamp – the higher 16-bits are assumed to be equivalent to the ones sent in the last full frame. Therefore, a change in one of the higher 16-bits of a timestamp requires a new full frame to be sent.

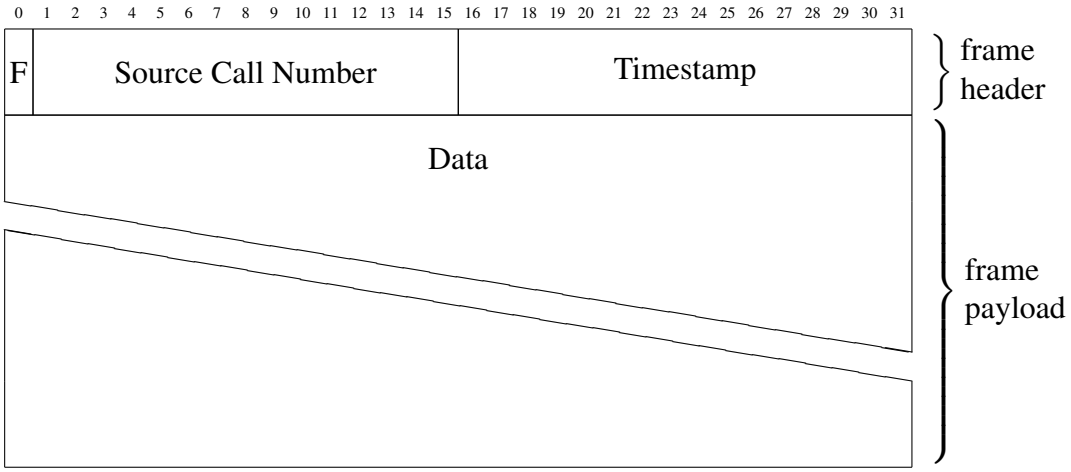

Figure 4.13: IAX2 mini frame binary format as described in [45]

field	description
F	1 for full frame, 0 for mini frame
Source Call Number	number of the caller
Timestamp	16-bit timestamp

Figure 4.14: IAX2 mini frame field description as described in [45]

4.2.4 Sequence diagrams

Figure 4.15 and 4.16 show message sequence diagrams for call setup and call teardown.

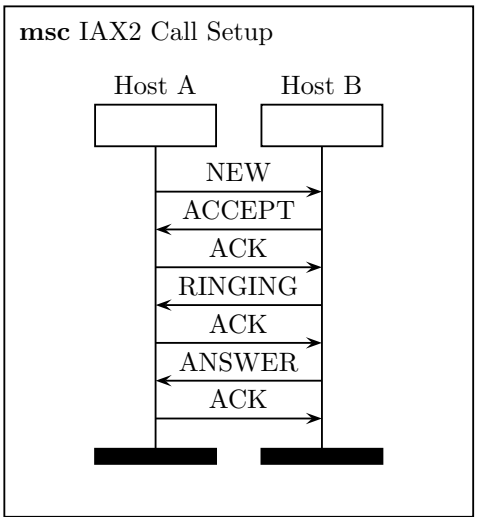

Figure 4.15: Message sequence chart of IAX2 call setup

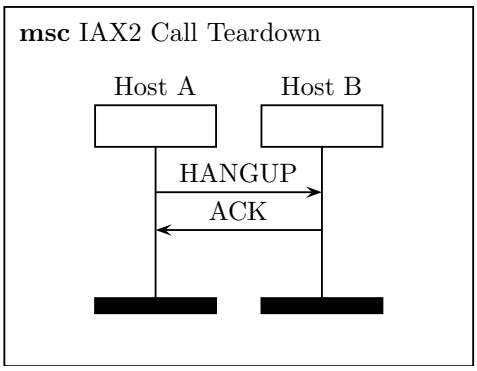

Figure 4.16: Message sequence chart of IAX2 call teardown

4.2.5 NAT compatibility

IAX2 uses the single UDP port number 4569 for communication with the remote endpoint. A client typically registers with its local IAX2 server and uses keep-alive packets to keep the NAT binding open.

If a client calls another client, it contacts the IAX2 server of the callee. All IAX2 communication between caller and callee is relayed through the IAX2 server of the callee.

Hence the only requirement to the network in respect to NAT is, that the IAX2 port of the callees IAX2 server must be reachable from the public Internet. Caller and callee only require a NAT router which is capable of routing UDP traffic. There are no special requirements on the type of NAT implementation.

4.2.6 Call path optimization

In a normal IAX2 setup the IAX2 server of the callee relays all IAX2 packets between caller and callee. This may lead to high bandwidth requirements on the server and to high latencies between the clients.

In order to resolve this issue the IAX2 server is able to remove itself from the call path once the connection has been established. This is done by initiating a supervised transfer. The supervised transfer is started by sending a `TXREQ` message to both peers in a call. This message contains the public IP address and public port number information of the peers as known by the server.

Once the endpoints receive this information, they try to establish a direct connection between each other. One of the clients needs to have an IP address and port number which is reachable by the other client. Once a client receives an UDP packet by the other client, it can use the open NAT binding used by this packet to send back its replies.

If the endpoints are able to establish a direct connection between each other, they send a `TXREADY` message to the server. After the server received `TXREADY` messages from both clients, it sends a `TXREL` message to the clients. This instructs the clients to use the new established direct connection instead of relaying the data over the server. After sending the `TXREL` message, the server removes itself from the call path.

If the transfer is not successful, one (or both) of the clients send a `TXREJ` message to the server. This notifies the server that a transfer is not possible and that the call will continue to be routed over the server.

The advantage of the call transfer mechanism in IAX2 is that it is very robust. As long as an UDP connection is possible to the IAX2 server, calls between two endpoints are possible. If the call path optimization fails, this does not result in a loss of connectivity – the call just continues as before. Furthermore, the mechanism is transparent to the user. A user does not notice if the call is routed over an intermediate IAX2 server or over a direct connection.

The disadvantage of this method is, that it only uses the IP addresses and port numbers which are known to the IAX2 server, instead of all possible addresses available at the client. This might pose a problem, if both, caller and callee, are behind the same NAT gateway and if this gateway does not support pinholing. If one of the endpoints tries to reach the other endpoint on the public IP address and port number combination as known by the IAX2 server, this connection will fail. Therefore, IAX2 sometimes continues to relay calls over a central server, although a direct connection would theoretically be possible.

Another disadvantage comes from the fact, that on a successful call path optimization the server removes itself from both, the signaling and the media stream. As a result, it is not possible to collect detailed CDR information on the server.

4.3 Extensible Messaging and Presence Protocol / Jingle

The Jabber protocol was invented in 1999 by Jeremie Miller as an alternative to the proprietary IM solutions at this time [46]. It is a client-server protocol which is similar in its architecture to SMTP. Jabber is build as a streaming XML protocol. In 2004 the IETF published the XMPP specification, which is based upon the Jabber protocol.

In 2005 an extension to XMPP called Jingle was published. Jingle allows two XMPP entities to directly transfer binary data between each other. An application of Jingle is VoIP [47].

XMPP users are identified by an XMPP ID which is also termed as JID[8]. Like an email address, a JID consists of a local part and a domain part, which are separated by an `@`-sign. An example

[8] Jabber ID

of a JID is: `user@domain.example/resource_identifier`.

The following elements are part of a JID:

- Domain Identifier

  This is the domain name which is used to identify the XMPP server.

- Node Identifier

  This is the local name of the user on a specific XMPP server. The username is distinct on the server, but may occur multiple times on the Internet.

- Resource Identifier

  The resource identifier assigns a unique identifier to each connection of a given user ID. This allows several connections from the same user ID to a single XMPP server, which are distinguished by the resource identifier.

Only the domain identifier is required for a JID to be valid, the node identifier and the resource identifier are optional.

4.3.1 XMPP

Architecture

XMPP is based upon a client-server architecture which is similar to the architecture used in SMTP. There are two different types of entities:

- XMPP Clients

  XMPP clients are used to connect to XMPP servers. If a client wants to communicate with another client, it sends its stanzas to the XMPP server.

- XMPP Servers

  XMPP servers are used by the clients to communicate with each other. If two clients on different servers try to communicate, the servers start a connection with each other and relay the data.

  In order to prevent spoofing of source addresses, XMPP supports a mechanism called dial-back, which is shown in Figure 4.17. When the originating server establishes a connection

to the receiving server, it also sends a dialback key to the receiving server. Afterwards, the receiving server establishes a new connection to the authoritative server – which is the server listed in the DNS entry for the domain – to validate if the dialback key is correct. As an alternative to the dialback mechanism it is also possible to use SASL in order to check the identity of a given server.

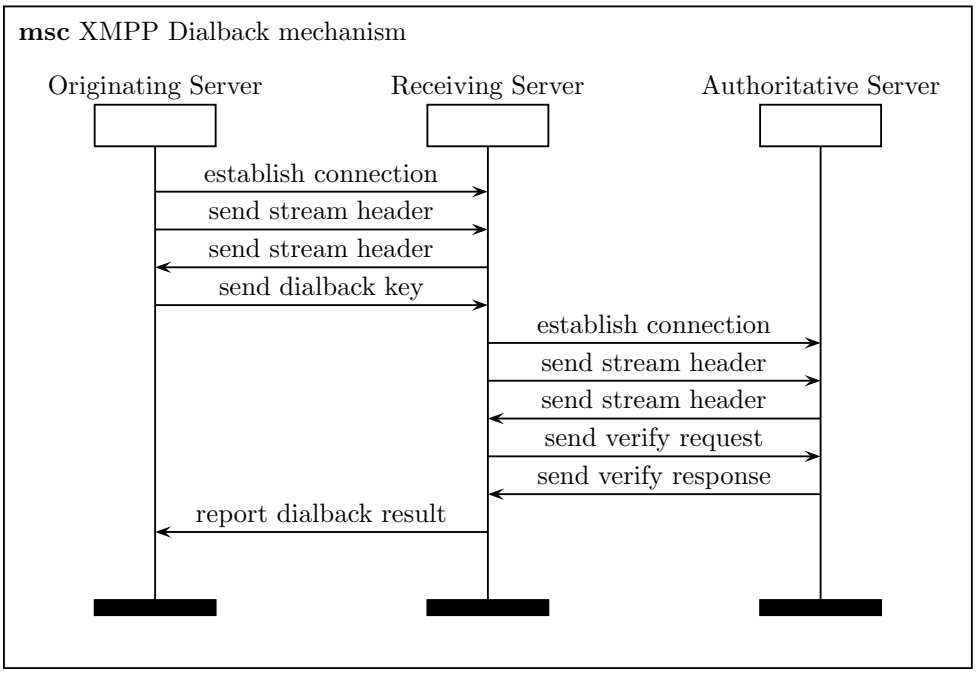

Figure 4.17: XMPP Dialback mechanism as described in RFC 3920

Figure 4.18 shows an example of a possible XMPP network setup. Client 1 and client 2 are connected to server 1, client 3 and client 4 are connected to server 2. If client 1 and client 2 communicate with each other, message are routed along server 1. If a client connected to server 1 wants to communicate with a client connected to server 2, each of them stays connected only to its own servers, but the servers exchange messages with each other.

Protocol

XMPP is defined in the IETF RFCs 3920–3923:

- RFC3920 [48] defines the core of the XMPP protocol.

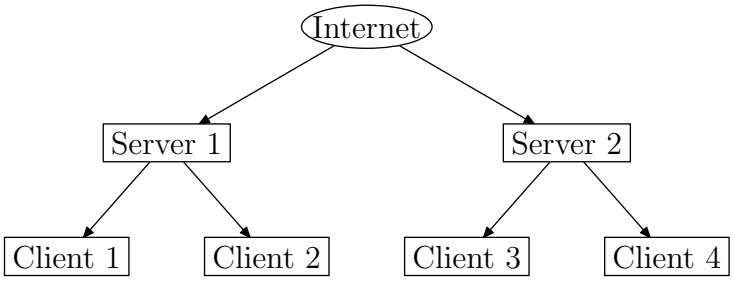

Figure 4.18: Network setup of XMPP

This RFC defines a streaming XML protocol, which is able to exchange messages. The content of these message is not defined.

- RFC3921 [49] deals with the extensions for instant messaging and presence.

  This RFC extends RFC3920 for the usage as Instant Messaging protocol. It defines the message types used for instant messaging. Furthermore, it defines how presence information is propagated.

- RFC3922 [50] defines a mapping between the Common Profile for Instant Messaging (CPIM) model [51] and XMPP.

- RFC3923 [52] specifies how end-to-end signing and encryption are used in the XMPP protocol.

Extensions to XMPP are called XMPP Extension Protocols (XEPs) and are published by the XMPP Standards Foundation (XSF). The published XEPs are available online at `http://www.xmpp.org/extensions/`.

XMPP is based on XML and uses XML streams. An XML stream is an open-ended XML document which is sent over a TCP connection. The root element of this XML stream is called `<stream/>`. The first level elements within the `<stream/>` element are called stanzas.

There are three different stanza core types:

- `<message/>`

  These stanzas are used in order to send messages between two XMPP entities. An example is shown in Figure 4.19.

- `<presence/>`

These stanzas are used to broadcast information to multiple users. In IM applications the `<presence/>` stanza can be used to notify the contacts of a user about the status[9] of the user. An example is shown in Figure 4.20.

- `<iq/>`

  These stanzas provide a request-response mechanism. An entity can query another entity via an `<iq/>` stanza. The other entity then sends back a result. An example is shown in Figure 4.21.

```
<message from='sender@domain.example'
         to='recipient@domain.example'>
    <body>Hello - this is a message!</body>
</message>
```

Figure 4.19: XMPP `<message/>` stanza example

```
<presence from='sender@domain.example/test'
          to='recipient@domain.example'/>
```

Figure 4.20: XMPP `<presence/>` stanza example

```
<iq type='error' id='some-id'>
  <error type='modify'>
    <bad-request xmlns='urn:ietf:params:xml:ns:xmpp-stanzas'/>
    <too-many-parameters xmlns='application-ns'/>
  </error>
</iq>
```

Figure 4.21: XMPP `<iq/>` stanza example

Basically when an XMPP client opens a connection to an XMPP server, it first opens a TCP connection. Afterwards, both client and server send a `<stream>` element in order to start the XMPP connection.

Subsequently, these two entities continuously parse incoming stanzas and send stanzas by themselves. At the end of the XMPP session (e.g. if the clients logs out) an `</stream>` element is sent in order to close the XML stream.

XMPP servers also use XML streams to communicate with each other.

[9] e.g. available or idle

Client Server Connections

As XMPP is based upon a client-server paradigm, clients usually connect to a single server. Connections from clients to servers are termed as C2S. The name of the server can either be hardcoded in the client or autodetected by resolving the `_xmpp-client._tcp SRV` record for the domain part of the JID.

Different clients may use the same JID to connect to the same server as long as the resource identifier of the XMPP address is distinct.

The IANA assigned port number for C2S connections is 5222.

Server Server Connections

If two clients on different servers want to communicate with each other, the respective servers connect to each other and start to relay the data. The server for a specific domain is detected by resolving the `_xmpp-server._tcp SRV` record for the domain part of the JID.

The IANA assigned port number for S2S connections is 5269.

Service Discovery

Service Discovery was published as XEP-0030. Service Discovery allows one XMPP entity to find out information about another XMPP entity. An example would be an XMPP client, which asks another client if and which type of VoIP connection it supports.

There are three different types of information which can be discovered by the usage of Service Discovery:

- The Basic Identity

  This describes the type and the category of the entity.

- Features and Protocols

  This describes the features implemented by the entity and the protocols which it supports.

- Additional items associated with the entity

  These items (child elements) may or may not be addressable with a JID. If they are addressable, they can also be queried using Service Discovery.

NAT and firewall compatibility

XMPP uses a traditional client-server architecture. Clients need to be able to open an outgoing TCP connection to a specified port at the server, while servers need to be able to receive incoming TCP connections. If they want to communicate with other servers, they also need to be able to establish outgoing TCP connections.

One of the use cases for NAT routers are outgoing client-server connections. As this is exactly what is used by C2S connections, they work well across NAT routers.

XMPP servers which are located behind a NAT need a manually configured port forwarding to their XMPP port.

Many firewalls do not allow outbound connections to untrusted ports, therefore client-server XMPP connections may sometimes be blocked. While some firewalls allow to use port 80 as workaround, there are firewalls which force the usage of the HTTP protocol on this port [53].

To work around this problem, XMPP defines XEP-0124 Bidirectional-streams Over Synchronous HTTP (BOSH) [54][10] which allows XMPP streams to operate over HTTP. BOSH allows to tunnel the XML stream used in the XMPP connection over a HTTP connection. Figure 4.22 shows how BOSH operates. Instead of connecting directly to the XMPP Server, the XMPP client connects to the BOSH connection manager. The connection between the XMPP client and the connection manager is based upon the HTTP protocol. The connection manager converts the BOSH XML stream to an XMPP XML stream which it sends to the XMPP server.

In order to achieve high bandwidth efficiency and low latency, BOSH employees a technique called Comet, which uses multiple synchronous HTTP request/response pairs. After the client initially opened a HTTP connection to the connection manager, the connection manager does not close the connection, but leaves it open until it wants to reply with an XMPP stanza. As soon as the connection manager receives an XMPP stanza which should be sent to the client, it sends the stanza and closes the connection afterwards.

[10] this supersedes the former XEP-0025: Jabber HTTP Polling [53]

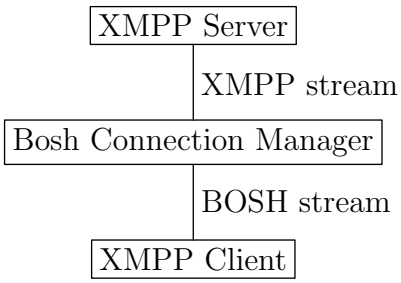

Figure 4.22: Bidirectional-streams Over Synchronous HTTP (BOSH)

If the client wants to send a stanza to the server while the HTTP connection is already open, it opens another HTTP connection to send the stanza. Only one HTTP connection from each client is kept in the `open` state by the server. Therefore, a maximum of two concurrent HTTP connections are needed.

BOSH is compatible to HTTP proxies. This means that the HTTP connection between the XMPP client and the BOSH connection manager may be routed across an HTTP proxy. BOSH is also compatible to the HTTP restrictions placed on JavaScript. It is possible to implement an XMPP client written in JavaScript which, accesses the XMPP server by the usage of the BOSH protocol.

4.3.2 Jingle

XMPP communication takes place over an XML messaging channel which is routed across XMPP servers. While this works for types of communication which do not have real-time requirements and which only have low bandwidth requirements, there are several types of communication where this method is not suitable. An example are VoIP sessions and file transfers between two XMPP entities.

To overcome this problem, the XMPP Standards Foundation published XEP-0166: Jingle. Jingle is a framework which allows two XMPP entities to establish a direct peer-to-peer session between each other. The Jingle XEP itself only defines the session management. The content description, which describes the content of the session and the transport method, which describes how the two entities communicate, are specified in their own respective XEPs.

Jingle Session Setup

To setup a Jingle session, an XMPP entity (the sender) sends a request using an `<iq/>` stanza to another XMPP entity (the receiver). If this second entity accepts the connection, it sends back an acknowledgment. Afterwards, both peers exchange transport candidates and try to establish connections using these candidates. As soon as a connection is established, the receiver acknowledges this by sending a `session-accept` message to the sender. After this message is received, both parties can start to transfer data over the newly created Jingle session.

Jingle Transports

While the Jingle XEP describes a framework for negotiating and managing out-of-band data sessions over XMPP, it does not define transport methods or content formats. These parts are therefore defined in their own respective XEPs.

Currently two different transport methods have been published as XEPs:

- Raw UDP
- Jingle ICE

Raw UDP The raw UDP transport tries to identify an IP address and port number that is most likely to succeed when a connection is established. In order to do this, it needs to classify the permissiveness of the NAT router or the firewall, and then assign the IP address and port number of the transport according to the rules in Figure 4.23. Figure 4.24 shows how a host announces a transport candidate to another host. Figure 4.25 shows how the remote host acknowledges the acceptance of the transport.

After the recipient receives the transport candidate, he must immediately check if it is working. To check the transport candidate, the recipient finds a candidate pair on its own site, which he sends to the initial sender. He continues to check connectivity from its own local transport to the remote transport pair. This is done by using the echo protocol described in RFC 862 [55]. At the same time he announces its own local candidate to the sender as shown in Figure 4.26. The sender proceeds to acknowledge this candidate as shown in Figure 4.27.

If the transport check succeeded, the party informs its peer by sending an Information Payload Element in an IQ set. The content of the Information Payload Element is selected according to the rules shown in Figure 4.28.

NAT Type	Recommended Raw UDP Candidate Type
None Symmetric (not permissive) Permissive	Host candidate Relay candidate Server reflexive or peer reflexive candidate discovered via RFC 3489 [4]

Figure 4.23: Raw UDP Candidate Assignment

```
<iq from='caller@example/client' to='callee@example/client'
    id='jingle1' type='set'>
 <jingle xmlns='http://www.xmpp.org/extensions/xep-0166.html#ns'
         action='session-initiate'
         initiator='caller@domain.example/client'
         sid='cb592c0854588e38'>
  <content creator='initiator' name='audio-content'>
   <description ... />
   <transport xmlns='http://www.xmpp.org/extensions/xep-0177.html#ns'>
     <candidate ip='192.168.0.130' port='32768' generation='0'/>
   </transport>
  </content>
 </jingle>
</iq>
```

Figure 4.24: Jingle UDP transport transmission

```
<iq from='caller@example/client' to='callee@example/client'
    type='result' id='accept1'/>
```

Figure 4.25: Jingle UDP transport transmission accept

Jingle ICE Jingle ICE is a Jingle transport method based on the Interactive Connectivity Establishment (ICE) [28] specification published by the IETF. It specifies two transport methods which can be used to transfer out-of-band data between two XMPP entities:

- `ice-udp`

  This specifies a lossy transport method which is suitable for media applications. E.g. VoIP

```
<iq from='callee@example/client' to='caller@example/client'
    id='jingle2' type='set'>
 <jingle xmlns='http://www.xmpp.org/extensions/xep-0166.html#ns'
         action='transport-info'
         initiator='caller@domain.example/client'
         sid='cb592c0854588e38'>
  <content creator='initiator' name='audio-content'>
   <transport xmlns='http://www.xmpp.org/extensions/xep-0177.html#ns'>
    <candidate ip='64.79.194.88' port='5454' generation='0'/>
   </transport>
  </content>
 </jingle>
</iq>
```

Figure 4.26: Jingle UDP transport transmission

```
<iq from='caller@example/client' to='callee@example/client'
    type='result' id='jingle2'/>
```

Figure 4.27: Jingle UDP transport transmission accept

Element	Meaning
<failed/>	Connectivity checks failed.
<succeeded/>	Connectivity checks succeeded.
<trying/>	Connectivity checks are underway.

Figure 4.28: Information Payload Elements

- `ice-tcp`

  This specification defines a transport based on TCP, which can be used for applications like file sharing where packet loss is not acceptable.

Unlike the Jingle UDP transport method which only sends the IP address and port number of the transport that is most likely to succeed, the ICE transport method identifies a set of possible IP address and port number combinations and checks which of them does succeed. The details of the ICE algorithm are discussed in Chapter 3.6.

Unlike the ICE draft which is specified by the IETF, the Jingle ICE transport mechanism maps the SDP attributes of ICE to attributes of the candidate XML element.

Chapter 5

Evaluation of Traversal Techniques in VoIP Protocols

In this chapter the NAT traversal capabilities of the VoIP protocols introduced in Chapter 4 are evaluated. Signaling and media protocols are evaluated independently from each other. First, different criteria upon which NAT traversal techniques can be evaluated are introduced. Afterwards, NAT traversal capabilities of signaling and media protocols are evaluated for different network environments. This chapter concludes with suggestions for new techniques, which can be used to improve the NAT traversal capabilities of VoIP protocols.

5.1 Assessment Criterions

NAT traversal techniques can be evaluated upon different assessment criterions. One essential criterion is the fact, that the establishment of a connection is possible for a given network setup. If the establishment is not possible, it does not make sense to examine any further criterions. If the establishment is possible, other criterions can be used to differentiate between the capabilities of the protocols.

One example where additional criterions can help to find the protocol which is better suited for a given setup, are these two NAT traversal protocols: The first protocol is able to establish a direct point-to-point connection using UDP, but only under the precondition that the NAT device exhibits Full Cone behavior. The second protocol tunnels data over a central server using HTTP. While it is able to traverse any type of NAT, it introduces a high latency into the media stream.

In the case of a Full Cone NAT, both protocols are able to successfully establish a connection. In order to differentiate between them, an additional criterion, which represents the latency of the media stream, could be used. By applying this criterion one could show, that in the case of a Full Cone NAT the first protocol provides better traversal capabilities, while in all other cases the second protocol has a better performance.

This chapter introduces several criterions which can be used to assess different NAT traversal techniques. However, no weights are given for the individual criterions. The weighting of a criterion depends upon the network setup in which the traversal technique is employed. As an example, consider a protocol which relays data over a central server. If this server is located on the public Internet, this protocol might be a suboptimal choice, as it introduces additional bandwidth requirements and latency when compared to a direct point-to-point connection. However, in a network setup where the relay proxy is located on the same local area network as the respective clients, bandwidth requirements and latency may be negligible.

The criterions defined are:

- Reliability and Stability

  This criterion identifies the reliability and stability of a given NAT traversal technique. Some techniques work only for a particular type of NAT devices, while other techniques work for almost all combinations of NAT devices and network setups. In a given environment, a technique provides a higher reliability than another technique, if the failure rate of the connection establishment is smaller.

  There are several approaches which can be used to provide a high reliability: Some traversal techniques use methods which do not require UDP hole punching, one example of these methods are relay servers. However, relay servers have the disadvantage of introducing additional latency into the media stream. Other more advanced techniques are able to use multiple methods. They try to establish a direct peer-to-peer connection, although this may not work in all cases. If the establishment of this connection fails, they fall back to another more reliable technique.

- Time required to establish connection

  For some traversal techniques the establishment of a connection may take a considerable amount of time.

  Some of these delays are caused by the fact that not all NAT devices send out Internet Control Message Protocol (ICMP) notifications if packets are addressed to an invalid binding

on the NAT device. Suppose a traversal protocol, which first tries to establish a direct connection, but falls back to a relayed connection if the establishment of the direct connection fails. As long as it does not receive any replies to its packets, it cannot assert if packets are just delayed or if the establishment of the connection failed.

One way to decide whether the establishment of the connection has failed is the usage of a timeout value. If the traversal technique cannot detect a successful connection within a time period equal to the timeout value, it assumes that the establishment has failed. However, in the case of a low timeout value, connections where the setup takes longer than expected are wrongfully discarded. In the case of a high timeout value, it takes a long time until the traversal technique can switch to a fallback method.

There are several optimizations, which can be applied in order to keep the required setup time small. A VoIP protocol might start the traversal as soon as the caller places the call, instead of waiting for the callee to pick up the phone. It is also possible to cache the traversal results of former calls, to avoid the rechecking of all available traversal techniques on each new call.

- Latency of the media stream

  The latency of the media stream is defined by the time it takes for a packet which leaves the host of the caller to reach the host of the callee. Direct peer-to-peer connections usually exhibit a smaller delay than connections which are routed over a central media relay.

  But also in the case of direct connections different traversal techniques may exhibit different latencies. Suppose two hosts are located on the same LAN. The LAN is connected to a WAN uplink, which itself is connected to a NAT device. If a traversal technique does not notice that both hosts are located on the same LAN, it might try to establish a connection using bindings on the NAT device. However, if the clients use such a connection, the media data needs to traverse the WAN link twice: First the data is sent from the caller to the NAT device and afterwards the NAT device sends the data back to the callee. If the traversal technique would be able to detect, that both hosts are located on the same network, it could just use their internal addresses as destination for the packets.

- Bandwidth requirements at VoIP provider

  Some NAT traversal methods might require additional bandwidth at the location of the VoIP provider. E.g. if the VoIP provider operates relay servers or a STUN service.

  Bandwidth required for STUN is negligible, as only few packets at the beginning of each call are exchanged. However, bandwidth requirements for relay servers can be exten-

sive. Consider a relay server which relays 100 concurrent calls which are encoded using a 64 kbit/s audio codec. The media stream is bidirectional as both sides of a call concurrently send data to each other. Furthermore, each packet which is sent to the relay server needs to be counted twice: First it is sent to the media server by one of the participants in a call, afterwards the media server needs to forward the packet to the other participant in the call. Therefore, the bandwidth requirements of these 100 concurrent calls are: 100 calls times 64 kbit/s times two[1] times two[2]: 25 Mbit/s.

5.2 Traversal Capabilities in VoIP Signaling

This chapter examines the NAT traversal capabilities of VoIP signaling protocols. A signaling protocol is responsible for the setup and the management of the voice call. It does not transport any media data, this is the task of a media protocol like RTP.

In contrast to media protocols, relatively simple NAT traversal techniques are sufficient for signaling protocols. Signaling protocols do neither have high bandwidth nor low latency requirements. This makes relaying of signaling data a viable alternative. Furthermore, signaling information does not need to be exchanged using the UDP protocol. Some signaling protocols even allow to tunnel signaling data over the HTTP protocol.

5.2.1 Session Initiation Protocol (SIP)

The SIP protocol does not specify which underlying transport protocol must be used to transport SIP messages. While the specification mentions several possible transport protocols, SIP works independently of any particular transport protocol. According to RFC 3261 the support for UDP and TCP transport protocols is mandatory for SIP implementations. However, SIP implementations may also support other additional transport protocols like the Stream Control Transmission Protocol (SCTP).

SIP clients do not necessarily need to use the same transport protocol in order to communicate with each other. SIP proxies are able to translate SIP messages between different transport

[1] because of the bidirectional calls
[2] because the packet is sent and received

protocols. E.g. one of the clients may talk to the SIP proxy using UDP, while the other client is connected using TCP. Of course, messages need to traverse the SIP proxy in order to allow the proxy to translate between transport protocols.

In the following section, the NAT traversal capabilities of the SIP signaling protocol are evaluated for the UDP and TCP transport protocols:

- User Datagram Protocol (UDP)

  When examining the NAT traversal capabilities of the UDP transport protocol, there are two different options for the behavior of the SIP protocol: Either the `Record-Route` header which is described in Chapter 4.1.2 may be used or it may not be used.

  If the `Record-Route` header field is not used, only the first SIP message and the response to it are routed through the SIP proxy. All subsequent messages are directly sent to the remote client, which is the destination listed in the `Contact` header field. The problem is, that the information in the `Contact` header field may contain private-space or public-space IP addresses of NAT bindings, which are not reachable by the remote client. Therefore, messages may not arrive correctly at the destination.

  If the `Record-Route` header field is set by a proxy server, this allows the proxy server to stay in the signaling path. All subsequent messages are also routed via the proxy server.

  The callee, which initially sent the SIP `INVITE` message to the proxy server, does not need to take any special measures in order for a successful NAT traversal. By sending an outgoing UDP packet to the SIP server, it opened a binding on its local NAT device on which it can receive replies sent by the SIP proxy. It does not matter which type of NAT device is used, as the UDP replies are expected from the same IP address and port number which was also used as destination for the initial packets. All types of NAT devices work equally well. The only precondition is, that the NAT device must support routing of the UDP protocol. If the `Contact` header field contains a private-space IP address, the SIP proxy must use the source IP address and port number of the initial packet which it received by the client as the destination for all further packets. This is required, as the private-space IP address is not directly reachable from the public Internet. It ensures, that the SIP proxy correctly replies to the binding on the NAT device.

  If a proxy wants to reach the client which is the recipient of an `INVITE` message, the steps required for NAT traversal are slightly different: A SIP message is sent to the proxy

server which is responsible for the given domain. In order to receive SIP messages intended for a user at the domain, a SIP client issues a `REGISTER` request to the SIP server. The `REGISTER` request instructs the server to forward incoming SIP messages which are addressed to the particular username to the client. A `REGISTER` request also contains an IP address and a port number, under which the client is reachable. However, in the case of private space IP addresses a SIP proxy cannot use them to send SIP messages to the client. In order to reach the client, the SIP proxy needs to send the messages to the source IP address and port number of the initial `REGISTER` message.

The above steps should be sufficient for NAT traversal when the SIP protocol is used together with UDP as transport layer protocol. However, if the time period between any two SIP messages is too large, it may be possible that a timeout occurs at the NAT binding. To prevent such a case, the SIP proxy can send keep-alive packets in regular intervals. The SIP RFC does not specify how keep-alive packets should be constructed. SIP implementations usually use methods which do not alter the state of the remote endpoint inside keep-alive packets. Examples of such methods are the `OPTIONS` and the `NOTIFY` method.

- Transmission Control Protocol (TCP)

  TCP is another transport layer for the SIP signaling protocol. As in the case of UDP, the `Record-Route` header field needs to be used in order to facilitate a successful NAT traversal. If the `Record-Route` header field is not used, the SIP proxy will try to establish a TCP connection with the address information contained within the `Contact` header field. Most likely this will not succeed if the client is located behind a NAT device.

  If TCP is used together with the `Record-Route` header field, the TCP connection which has been established from the client to the proxy is also used by the proxy to send replies back to the client. The caller has initially established this TCP connection by transmitting the `INVITE` message to the proxy. At the side of the callee, the connection which has initially been created for the `REGISTER` message is used. Both clients only need to establish outgoing TCP connections to the proxy server. Therefore, this works for all types of NAT implementations.

5.2.2 Extensible Messaging and Presence Protocol (XMPP)

Jingle is an extension to the XMPP protocol, which allows the establishment of VoIP sessions. XMPP uses TCP as underlying transport protocol. All XMPP traffic must be routed over an

XMPP server. This behavior is similar to the SIP protocol with an enabled `Record-Route` option and an enabled outgoing proxy: Each XMPP client connects to its local XMPP server. If an XMPP client intends to send a message to another client, it forwards the message to its local XMPP server. The local XMPP server transmits the message to the remote XMPP server, which forwards the message to the remote client.

As in the case of the SIP protocol used together with the TCP transport layer protocol, XMPP clients only need to have the possibility to open an outgoing TCP connection to the XMPP server. As no hole punching is required, the establishment of a connection is possible with all types of NAT devices.

If a firewall filters traffic destined for specific port numbers, usage of the XMPP protocol might still be possible. While the IANA reserves a well-known port for the XMPP protocol, the operator of an XMPP server may use DNS `SRV` records in order to notify a client about the port numbers which can be used to contact the XMPP server. One option to avoid firewalls is the usage of the port number, which is reserved for SSL encrypted HTTP connections.

XMPP also provides the option of using HTTP instead of TCP in order to connect to the XMPP server. This may be useful, if an firewall blocks all types of traffic except HTTP. XMPP data is tunneled within HTTP data using the BOSH mechanism which is described in the XEP-0124 XMPP extension[3]. If the BOSH extension is used, establishing XMPP connections is possible as soon as outgoing HTTP connections can be established.

5.2.3 Inter-Asterisk eXchange (IAX2)

The IAX2 protocol uses UDP as underlying transport protocol. The usage of other transport protocols is not possible. A characteristic of the IAX2 protocol is, that it only uses one single protocol for the signaling and for the media path. Both parts of the protocol are transferred using the same UDP connection, it is not possible for the media part of the protocol to use another route than the signaling part.

IAX2 connections are established by opening an outgoing UDP connection to an IAX2 server. Clients which want to be reachable under a particular username at a server, can register at that server using a username and a password. The UDP binding which is established from the client

[3]XEP-0124: Bidirectional-streams Over Synchronous HTTP (BOSH)

to the server is also used by the server to send replies back to the client. All IAX2 packets are routed through the IAX2 server. Unlike other signaling protocols, IAX2 does not require NAT devices to support UDP hole punching.

However, IAX2 also allows to establish a direct peer-to-peer connection, if the NAT device supports UDP hole punching. If the `transfer` option is enabled at an IAX2 server, the call setup first starts equal to a normal IAX2 call setup. As soon as the call is established, the IAX2 server tries to induce a direct connection by notifying both clients of their respective public IP addresses and port numbers as seen by the IAX2 server. The IAX2 clients then try to establish a direct peer-to-peer connection using the information provided by the IAX2 server. If they are able to do this, the direct connection is used instead of the relayed connection. Otherwise, the clients continue to use the relayed connection. A detailed discussion about the NAT traversal capabilities of the IAX2 protocol for different types of NAT devices is given in Section 5.3.

The drawback of establishing a direct peer-to-peer connection between the two clients is, that the signaling part of the protocol is coupled to the media part. Both parts must take the same route. If a direct peer-to-peer connection is established, certain features which require the signaling data to traverse a central server are not possible any more. Examples of these features are the central collection of Call Details Records (CDR) and Music On Hold (MOH) with an audio file which is stored on a central server.

In summary, operation of the IAX2 is possible as long as the NAT device allows simple traversal of UDP packets. IAX2 does not require the NAT device to support UDP hole punching. However, if UDP hole punching is possible, the IAX2 protocol may be able to establish a direct peer-to-peer connection between both participants in a call.

5.2.4 Results

All evaluated signaling protocols are able to traverse NATs, as long as the NAT device supports either the UDP protocol (for SIP and IAX2) or the TCP protocol (for XMPP). UDP Hole Punching is not required, therefore these protocols should work with all types of NAT devices.

Sometimes NATs also include firewall functionality and only allow a particular kind of traffic. If UDP connections are blocked, operation of the IAX2 protocol is not possible. XMPP and SIP also work if UDP is not supported. XMPP uses TCP by default and SIP can use TCP as alternative transport level protocol. If the NAT device includes a firewall which only allows

outgoing HTTP connections, neither IAX2 nor SIP will work. However, the XMPP protocol allows clients to tunnel XMPP data over HTTP in order to connect to an XMPP server.

XMPP is the signaling protocol which works under the most possible NAT and firewall setups. SIP is ranked next, however for NAT traversal the `Record-Route` header field should be used for SIP messages. The IAX2 protocol provides the worst NAT traversal capabilities of the three signaling protocols which have been evaluated. If UDP support is not available, operation of the IAX2 protocol is not possible. However, from a practical point of view, the difference between the NAT traversal capabilities of these three protocols is negligible. Almost all NAT devices support the routing of TCP as well as the routing of UDP. XMPP is able to establish a signaling connection if neither UDP nor TCP are supported. However, the benefit of this signaling connection will be minimal, because the Jingle media part of the XMPP protocol still requires UDP support in order to transfer media data.

5.3 Traversal Capabilities in VoIP Media Transports

In this section the NAT traversal capabilities of the media protocols will be examined. Media protocols are responsible for the transmission of the actual audio data between two clients. Unlike signaling data, audio data requires a low latency and a higher bandwidth. While a relayed connection may be acceptable in some situations, a direct peer-to-peer connection allows for a lower latency between both endpoints and lower bandwidth requirements at the VoIP provider.

Some signaling protocols employ standardized media protocols, while other signaling protocols use their own media protocols. The examination in this chapter is structured according to the set of media protocols used by the signaling protocols which have been studied in the former chapter.

The following NAT traversal mechanisms are shared by the SIP and the Jingle signaling protocols:

- Simple Traversal of User Datagram Protocol (STUN)
- Relay Servers
- Interactive Connectivity Establishment (ICE)

The implementation of these media protocols in SIP and Jingle is quite similar. However there are some minor differences. For example the SIP protocol only allows clients to either use manually configured STUN servers or to obtain the names of the STUN servers by using DNS SRV queries [4]. The Jingle protocol also allows clients to use XMPP inband queries in order to obtain the names of the available STUN servers. This capability is specified in XEP-0215 (STUN Server Discovery for Jingle) [56].

In addition to these three media protocols, the IAX2 protocol uses its own proprietary media protocol which is coupled to the IAX2 signaling protocol.

The evaluation assumes, that SIP and Jingle make use of symmetric RTP connections as described in Section 3.3. A symmetric RTP connection receives RTP data on the same port on which it sends RTP data. For some types of NAT devices symmetric RTP simplifies NAT traversal. Suppose two clients are behind their respective NAT devices and both of the NAT devices exhibit Port Restricted Cone behavior as described in Section 2.3.2. In order to receive an incoming packet on a binding of such a type of NAT device, an outgoing packet must have been sent to the remote IP address and port number previously. If the VoIP client would not implement symmetric RTP, it would send its RTP packet to a binding which discards them, as the NAT device did not see any outgoing packets to the remote client. In the case of symmetric RTP this problem does not occur, as the same binding is used for incoming and outgoing packets.

The NAT traversal capabilities of media protocols will be evaluated for the following types of NAT devices:

- Full Cone

  A full-cone NAT does not restrict the source address or port number of incoming packets which want to use a binding on the NAT device. Thus a host on the internal network, which announces the IP address and port number of a binding which it obtained to another host, is able to receive incoming UDP packets which are sent from this host to the binding on the NAT device.

- Restricted Cone

  The behavior of a Restricted Cone NAT is similar to a Full Cone NAT. However, in order for an external host to be able to use a binding on the NAT device, its IP address must first have received an outgoing UDP packet via this binding. If a host on the internal network sends packets to two external IP addresses using the same source IP address and source port number, only one binding is created.

- Port Restricted Cone

  A Port Restricted Cone tightens the restrictions introduced by the Restricted Cone. In order to receive packets by a host on the external network, a packet from the internal network needs to be send to the IP address and port number of the host on the external network. The port number needs to be the same number which is used as the source port number in packets which are sent by the host outside the NAT. As in the case of a Port Restricted Cone, bindings are reused for all packets sent by the same source IP address and port number on the internal network.

- Symmetric

  Unlike the former three NAT variants which reuse bindings for UDP packets which are sent by the same source address and port number, symmetric NATs create a new binding for each new destination of an UDP packet. A destination is the combination of an IP address and a port number.

  This effectively makes it impossible for a host to discover the address of a binding, on which it can receive packets from the external network. While the host can use a method like STUN in order to discover the external IP address and port number of a binding, this binding will only be valid for return packets which are sent by the STUN server. If another host wants to send packets to the host inside the NAT, it cannot use the IP address and port number obtained via STUN.

  Because of this properties, the traversal of symmetric NATs is harder than traversal of the former variants. Unfortunately, there are still reasons why symmetric NATs are in use. One of them is, that the number of possible bindings on the NAT device is effectively only limited by the amount of RAM inside the NAT device: On a cone-type NAT, a binding is reused by all packets sent by the same IP address and port number, regardless of the destination. Each binding requires one UDP port on the external interface of the NAT device. This effectively limits the total number of bindings[4] to 65.536[5].

  Symmetric NATs however do not reuse bindings. Therefore, a binding is not defined by its external IP address on the NAT device, but by the combination of IP address on the NAT device, remote IP address and remote port number. Unlike a cone-type NAT, a symmetric NAT therefore allows up to 65.536 bindings per remote IP address. The total number of

[4] Whereas the number of bindings is equal to the number of IP addresses and port number combinations which are used to communicate with the external network.

[5] Under the assumption that exactly one IP address is assigned to the external interface of the NAT device.

bindings on the NAT device is not limited.

- NATs without support for Hairpinning

  NAT hairpinning support is required, if two hosts on the internal network want to communicate with each other by using a binding on the NAT device. In this case, the NAT device receives a packet from the internal network and forwards it back to the internal network. If hairpinning is not supported, forwarding the packet will fail and it will be dropped. According to [5] only about 24% of the UDP NAT implementations support UDP hairpinning.

  If hairpinning is not supported and two hosts on the same internal network want to communicate with each other, they need to communicate using their internal network addresses instead of sending the packets to the binding at the NAT device.

  However, there are also cases, where establishing a direct connection is not possible if hairpinning is not supported. Suppose, that an Internet Service Provider deploys one shared NAT device for all his customers. All outgoing connections to the public Internet are routed via this NAT device. Two customers *A* and *B* want to communicate with each other. However, each of these customers also operates his own NAT device on his own private network. The NAT device deployed by the Internet Service Provider does not support hairpinning. The NAT devices by the customers may or may not support hairpinning. Figure 5.1 shows the diagram for this network setup.

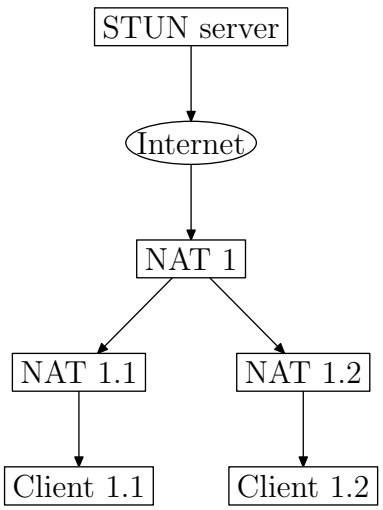

Figure 5.1: Double NAT

Now suppose, that Client 1.1 in the diagram wants to establish a connection to Client 1.2.

Both clients are located on different subnets, therefore they need to traverse at least NAT 1.1 and NAT 1.2 in order to establish a connection.

In the usual case, Client 1.2 would obtain a binding on NAT 1.2 and Client 1.1 would send its packet to the binding at NAT 1.2. However, if no STUN server is located on the internal network of NAT 1, Client 1.2 will only be able to obtain a binding on NAT 1. If NAT 1 does not support hairpinning, the clients are not able able to utilize the binding available on NAT 1. Instead they have to use an indirect relayed connection in order to communicate with each other.

To keep the number of evaluated setups maintainable, the evaluation assumes that both clients are located behind the same type of NAT. In general, cone-type NATs only require one of the NAT devices to be traversed in order to establish a connection. The reason is, that because of the usage of symmetric RTP, a successfully established oneway connection can also be used to send the package in the opposite direction. However, this is not true if one of the clients is located behind a symmetric NAT.

For the evaluation the NAT classification according to RFC 3489 is used. While RFC 4787 provides a more detailed classification, the higher number of features results in a larger number of possible feature combinations. Furthermore, published specifications for STUN are based upon RFC 3489. STUN specifications which are based upon RFC 4787 are still in draft state at the time of writing. However, single features described in RFC 4787 will be mentioned, if they allow to make better predictions about the NAT traversal capabilities of a protocol in specific cases.

5.3.1 Simple Traversal of User Datagram Protocol (STUN)

The STUN protocol tries to obtain a binding on the NAT device. Furthermore, it is able to make an educated guess about the type of NAT device in use. Whether the binding which is obtained by STUN can be used by an external peer, depends upon the type of NAT device.

- Full Cone NAT

  When STUN obtains a binding on a Full Cone NAT, any external client may use this binding in order to send UDP packets to the client which created the binding. This type of NAT does not require the usage of symmetric RTP.

- Restricted Cone NAT

  As a Restricted Cone NAT restricts the IP address of packets which want to use a binding, it might seem that STUN is not able to traverse a Restricted Cone NAT. However, an actual example shows that establishing a connection could still be possible.

  Suppose, that there are two clients: *A* and *B*. *A* sends a packet to a binding at *B*'s NAT device. As *B*'s NAT device is a restricted cone NAT, it drops the packet. Now *B* sends a packet to a binding at *A*'s NAT device. As *A* has already sent an outgoing packet to *B*, the NAT device of *A* allows packets which are sent from *B* to utilize the binding. The packet will therefore be received at *A*. As *B* has sent an outgoing packet to *A*, its own NAT device now allows incoming packets which are sent from *A* to traverse the NAT. Therefore, all subsequent packets sent from either *A* or *B* will arrive at the destination.

  Summarized, a restricted cone NAT may lead to lost packets, if incoming packets are received at a NAT device before outgoing packets have been sent. However, as soon as at least one outgoing packet sent to the remote client has been received by the NAT device, all incoming packets should be able to traverse the NAT device without any issues.

  As in the case of a Full Cone NAT, this type of NAT does not require the usage of symmetric RTP.

- Port Restricted Cone NAT

  For this type of NAT the STUN protocol behaves similar as in the previous case.

  However, unlike in the previous case with a Port Restricted Cone NAT, also the destination port number of outgoing packets must match the source port number of incoming packets. If symmetric RTP is used, this is the case: The source port number on the internal network, which is used to create a binding, is also used as source port number for outgoing packets sent to the remote client. As soon as at least one outgoing packet has been sent to the remote client, incoming packets may be received at the binding.

  If Symmetric RTP is not used, traversal of Port Restricted Cone NATs is not possible. Outgoing packets are sent using a different source port number than the source port number used to create the initial binding. Therefore, it is not possible to receive incoming packets at the binding.

- Symmetric NAT

  [28] shows that STUN is not able to establish a connection if both clients are located behind Symmetric NATs. Suppose there are two clients *A* and *B*. Each of these clients

obtains a binding by the usage of the STUN protocol. However, the binding obtained by STUN cannot be used by the clients as it is only valid for the STUN server which has been used to create the binding. If one of the clients sends a packet to the other client, a new binding is created. As neither one of the clients knows the port number of the mapping at the other client, the clients are not able to establish a direct connection between each other.

Establishing a connection between a client on the public network and another client behind a Symmetric NAT is possible, if Symmetric RTP is used together with one of the extension mechanisms defined in Section 3.3. These features must be supported by both clients. As soon as the client behind the NAT sends a packet to the host on the public network, it creates a binding on which it is able to receive replies by the remote host.

Establishing a connection between a Symmetric NAT and either a Full Cone NAT or a Restricted Cone NAT works fine too, if one of the UDP reflection techniques described above is used: Suppose client *A* is located behind the cone-type NAT and client *B* is located behind a Symmetric NAT. In the case of a Full Cone NAT, client *B* can send its packet directly to the binding obtained by client *A*. As soon as *A* receives the first packet, it can use the source IP address and port number of the packet in order to send replies back to *B*.

If *A* is located behind a Restricted Cone NAT instead of a Full Cone NAT, *B* is initially not able to send any packets to *A*'s binding. *A* needs to send a packet to *B*'s public IP address before *B* is able to send any packets to *A*. While *A* does not know the correct port number of *B*'s binding, this does not pose a problem. It can send its packet to any port number, e.g. the number which is specified by the signaling protocol. However, as soon as it receives any packet from *B*, it must change the destination port number of its outgoing packets to the source IP address of the received incoming packet.

It is not possible to establish a connection between a Port Restricted Cone NAT and a Symmetric NAT. As shown for the previous cases, the client behind the cone-type NAT only knows the public IP address of the client behind the Symmetric NAT, but not the port number. If we assume that *A* in the previous example is located behind a Port Restricted Cone NAT, it has no possibility to allow *B* to use its binding. *A* would have to send an outgoing packet to the IP address and port number from which it expects the incoming packets. As *B* is located behind a Symmetric NAT, *A* is not able to obtain this information, without having received an incoming packet sent by *B* first.

- Hairpinning

  The STUN protocol is able to detect the type of NAT device and the address information

of a binding on the external interface of the NAT device. However, STUN does not detect if the NAT device supports hairpinning. Therefore, even if the STUN protocol indicates, that the client is located behind a type of NAT where NAT traversal should work, it is not guaranteed that STUN is able to establish a connection.

The STUN protocol does work well with NAT devices which exhibit a Full Cone behavior. On other types of NAT devices, it may or may not be possible that the establishment of a connection is successful, depending on the features[6] which are used by both of the clients. However, the exact behavior also depends upon implementation details of the NAT devices which need to be traversed.

5.3.2 Relays Servers

Relay servers receive incoming packets from both of the VoIP clients, which want to establish a connection with each other. After both clients have sent packets to the relay server, the relay server starts to relay data between the two clients. Instead of using the address information transmitted in the signaling protocol, relay servers send the relayed data to the source IP address and port number of the received UDP packets.

In order to be able to receive replies at the source IP address and port number, both clients need to use symmetric RTP. Otherwise, the source port number will not be equal to the port number at which the clients expect the packets.

As relay servers just reply to incoming packets, they are compatible to all four types of NAT devices. As no direct connection between clients is created, the NAT device does not need to support hairpinning. Even if both clients are located on the same subnet, the media data is still sent to and received by the relay server.

A disadvantage of relay servers is, that they are often not the most efficient way to establish a connection. Compared to a direct connection, relay servers introduce a higher latency and higher bandwidth requirements at the location of the VoIP provider.

[6] Symmetric RTP and reflection of incoming media streams.

5.3.3 Interactive Connectivity Establishment (ICE)

The ICE protocol uses a set of different NAT traversal techniques in order to find the best path for a network connection between two clients. Both of the clients need to support the ICE protocol.

First, ICE obtains a set of candidates which may be used to contact the client. A candidate is a combination of an IP address and a port number. The set of obtained candidates includes the addresses of sockets on local network interfaces, the addresses of external bindings on NAT devices discovered by the usage of STUN, as well as the addresses of Relay Servers.

If there are several possible techniques which can be used to establish a connection, ICE uses the technique with the highest priority to establish the media path.

- Full Cone NAT

  In the case of a Full Cone NAT, ICE is able to establish a connection by using the STUN protocol.

- Restricted Cone NAT and Port Restricted Cone NAT

  In the case of a Restricted Cone NAT, ICE is able to establish a connection by using the STUN protocol.

  The evaluation of the STUN protocol showed, that the first packets of the media data may be dropped, if an incoming packet is received at the NAT device before an outgoing packet has been sent. If the ICE protocol is used, this is not the case. As ICE performs connectivity checks before assuming that a specific type of connection is possible, the packets will be lost during the operation of the ICE protocol. As a result, the required bindings are already established when the media data is transferred. Therefore no packets belonging to the media data will be dropped.

- Symmetric NAT

  It is not possible to use STUN to establish a connection between two clients, when each of the clients is connected behind a Symmetric NAT. The STUN connection method of ICE will fail in this case. Therefore, ICE will use a relayed connection in order to still allow the two clients to exchange media data with each other.

  If only one of the clients is located behind a Symmetric NAT, it depends upon the type of NAT if a direct connection is possible. Generally, ICE is able to establish a direct

connection, in the cases where the STUN protocol would be capable of establishing a direction connection too.

- Hairpinning

  Among the candidates which are checked for connectivity by ICE, are sockets on local interfaces. These local candidates have a higher priority than candidates obtained by STUN or candidates on relay servers.

  If both clients are located behind the same NAT device, ICE will establish a direct connection between candidates on local interfaces. As packets will not be relayed through the NAT device, it does not matter if hairpinning is supported.

  In the case of a double NAT as shown in Figure 5.1, a direct connection between the two clients is not possible. Connectivity via STUN is checked by ICE, but will not be possible if the NAT device does not support hairpinning. In this case ICE will use a relay server to establish the connection. Section 5.3.2 has shown, that relay servers are able to establish a connection between two clients, if hairpinning is not supported at the NAT device.

The ICE protocol is able to establish connections in all of the evaluated cases. If direct connections between two clients are possible, they are used. Otherwise the ICE protocol reverts to a relayed connection. Thus ICE combines the efficiency of the STUN protocol with the reliability of relay servers.

But also in cases where STUN is able to establish a connection, ICE may be able to provide a better performance. If two clients behind a common NAT which supports hairpinning want to communicate with each other, STUN would relay the data through the NAT device. ICE, however, is able to detect that both hosts are located on the same network. Therefore it establishes a direct connection between the hosts.

5.3.4 Inter-Asterisk eXchange (IAX2)

The IAX2 protocol is a signaling protocol as well as a media protocol. Initially all established connections are relayed over an IAX2 server. After the connection is established, the server can instruct the clients to try to establish a direct connection between each other. In order to do this, the server notifies the clients of the IP address and port number of their respective peer. Each of the clients tries to contact its peer at this address information. If one of the clients is able to

establish a direct connection with the other client, this connection is used instead of the relayed connection. If establishing a direct connection is not possible, the clients will continue to use the relayed connection.

- Full Cone NAT

  In the case of a Full Cone NAT the address information as seen by the server is also valid for the other client. Therefore the clients are able to establish a direct connection to each other.

- Restricted Cone NAT and Port Restricted Cone NAT

  IAX2 is able to establish a direct connection for this type of NAT, however the same issues as for STUN and ICE apply: It may be possible that incoming packets are dropped by a NAT device, if the NAT device did not see any outgoing packets to the respective IP address and port number yet. As the IAX2 protocol relays the media data until a direct connection is fully established, no packets of the media data will be lost.

  IAX2 always uses a symmetrical type of connection. The source port number is the same port number as the port number where the IAX2 protocol expects incoming packets. Therefore, traversal of Port Restricted Cone NATs is possible.

- Symmetric NAT

  IAX2 is not able to establish a direct connection, if both of the clients are located behind a Symmetric NAT. In this case an indirect relayed connection will be used instead.

  If only one of the clients is located behind a Symmetric NAT, it depends upon the type of NAT if a direct connection is possible. In the general case, IAX2 is able to establish a direct connection, if the STUN protocol would be able to establish a direction connection too.

- Hairpinning

  If both of the clients are located behind the same NAT device which does not support hairpinning, the initial relayed IAX2 connection will work fine. After the call is established, IAX2 tries to initiate a direct connection by notifying the clients of the external IP address and port number of their peer. As the NAT device does not support hairpinning, establishing a direct connection will fail. IAX2 will therefore use an indirect relayed connection.

  Unlike ICE, IAX2 clients do only try to establish a direct connection to the addresses seen at the IAX2 server. Therefore the clients are not able to detect that they are located on the same subnet.

5.3.5 Results

While the comparison of the signaling protocols showed very similar results, there are larger varieties in the results of the comparison of the protocols used for the traversal of the media data.

In SIP and Jingle the STUN, Relay Servers and ICE traversal methods are used. As all of these methods are based upon UDP, UDP communication needs to be possible. However, not all of these protocols require support for UDP hole punching.

The STUN protocol is based upon UDP hole punching techniques and allows to establish a direct connection between two endpoints. Connections which are established using the STUN protocol are characterized by a low latency between the endpoints. Furthermore, the STUN server does only help to establish the connection, the actual data is directly transferred between the two clients. Compared to Relay Servers, the bandwidth requirements of a STUN server are negligible. However, it was shown that STUN is less reliable than Relay Servers, as it does not work with all types of NAT implementations.

Relay Servers are able to provide a high reliability as they convert the peer-to-peer setup of a VoIP protocol to a traditional client-server setup. NATs are designed for this kind of setup [26], therefore Relay Servers are able to traverse all types of NATs. However, Relay Servers need to relay all the transferred media data. This causes a higher latency of the packets and higher bandwidth requirements than in the case of a direct connection between two endpoints. Relay servers are an acceptable fallback method, if no other type of connection is possible. Using Relay Servers as the only traversal technique may not be reasonable for a typical VoIP installation.

ICE does not try to make any assumptions about the network setup. Instead, it checks which types of connections actually work. Unlike the previous two techniques, ICE is able to find paths with a high reliability and a low latency. ICE depends upon underlying transport protocols like STUN and relay servers.

The traversal capabilities of the IAX2 protocol are roughly comparable to ICE. However, ICE has the advantage, that the signaling connection does not need to be coupled to the media data. Furthermore, ICE is able to detect if two hosts are located behind a common NAT device on the same network. In this case ICE allows the two hosts to communicate directly with each other, without relaying the packets through the NAT device. If IAX2 is used, the packets are either routed through the NAT device, if hairpinning is supported, or they are routed over an IAX2

server, if IAX2 is not able to establish a direct connection because of hairpinning issues. This allows ICE to provide better NAT traversal capabilities than the IAX2 protocol.

5.4 Suggestion for Improvement of Traversal Capabilities

In this section techniques, which are able to improve upon existing NAT traversal capabilities of VoIP protocols, are introduced. Evaluation of the NAT traversal techniques has shown, that ICE and IAX2 are able to traverse most types of NAT devices. However, in some cases the establishment of a direct connection between two clients is not possible. In this cases a relayed connection, which leads to high latency and high bandwidth requirements, is used.

In some of the cases, where ICE and IAX2 decide to use a relayed connection, other traversal techniques may be able to establish a direct connection between the two clients. One example is be the establishment of connections between two NAT devices which exhibit a symmetric behavior. STUN is not able to obtain the port number of a binding, which can be used to contact the client, as a symmetric NAT creates a new binding for each new IP address and port number. However, for some symmetric NATs the assignment of port numbers to new bindings is predictable. If the port number for each new binding is incremented by one, a VoIP client may be able to draw conclusions about the port number of a newly created binding.

The following section describes how the ICE protocol may be extended, in order to allow the implementation of additional NAT traversal techniques within the ICE framework. Afterwards, two different traversal techniques, which have not yet been implemented in ICE, are introduced.

5.4.1 Allowed ICE candidates

The ICE specification allows clients to use different methods in order to obtain candidates which could be used for establishing a communication path. Host candidates are sockets on a local interface on which communication may be possible. Examples of local interfaces are Ethernet interfaces or VPN connections.

The ICE specification defines a host candidates as:

> Host Candidate: A candidate obtained by binding to a specific port from an interface on the host. This includes both physical interfaces and logical ones, such as ones obtained through Virtual Private Networks (VPNs) and Realm Specific IP (RSIP) [RFC3102] (which lives at the operating system level).

Unfortunately the ICE specification is not as generic for the other types of candidates. While host candidates can be created on any type of local interface, the other types of candidates are restricted to bindings obtained by either the STUN or the TURN protocol:

> Server Reflexive Candidate: A candidate whose IP address and port are a binding allocated by a NAT for an agent when it sent a packet through the NAT to a server. Server reflexive candidates can be learned by STUN servers using the Binding Request, or TURN servers, which provides both a Relayed and server reflexive candidate.
>
> Peer Reflexive Candidate: A candidate whose IP address and port are a binding allocated by a NAT for an agent when it sent a STUN Binding Request through the NAT to its peer.
>
> Relayed Candidate: A candidate obtained by sending a TURN Allocate request from a host candidate to a TURN server. The relayed candidate is resident on the TURN server, and the TURN server relays packets back towards the agent.

The restriction to STUN and TURN places unnecessary constraints on the types of allowed candidates. It is not possible to use traversal techniques, which do not use either STUN or TURN, in order to obtain a remote binding.

The first proposed extension uses NAT-PMP to obtain a remote binding on a NAT device. The second extension uses port number prediction, in order to make an educated guess about the port number assigned to a binding on the NAT interface. Because this techniques are neither based upon the STUN, nor upon the TURN protocol, they do, strictly speaking, violate the ICE specification. However, to a remote client the new candidate is not distinguishable from a server reflexive candidate. Therefore, individual ICE implementations can use this mechanism to provide better NAT traversal capabilities, while still being compatible to other ICE implementations.

5.4.2 NAT-PMP

NAT-PMP is a technique which allows a client to explicitly request a binding on a NAT device. It is explained in Chapter 3.2. Unlike methods like STUN, where the client sends specially crafted packets and assumes that an appropriate mapping is created on the NAT device, NAT-PMP allows the client to receive direct feedback by the NAT device. This feedback notifies the client if the creation of the binding has been successful and under which public IP address and port number the binding can be reached. Furthermore, an NAT-PMP enabled device also announces chances of its external IP address to the clients on the internal network. Such changes could occur, if a dynamic IP address is assigned to the NAT device.

An extension of an ICE implementation to the NAT-PMP protocol would:

1. Ask the NAT-PMP device to create a new mapping.
2. Verify that the mapping was successfully created.
3. In the ICE data which is sent to the remote peer, the IP address and port number of the NAT-PMP mapping is included as server reflexive candidate.

If the mapping was created successfully, packets by a remote peer are able to traverse the NAT device on which the NAT-PMP mapping was created. When the remote peer sends a packet to the NAT-PMP binding, it implicitly creates a binding on its own NAT device. The client behind the NAT-PMP device only needs to reply to the source IP address and source port number of the packet, in order to reply to the client which sent the packet.

This mechanism is already provided by the ICE framework: If ICE detects a valid path for UDP packets in one direction, it automatically checks if sending packets in the opposite direction is also possible.

NAT-PMP support would provide a reliable technique for NAT traversal. It would be especially valuable in cases where the STUN protocol is not able to establish a direct connection between two clients. As the NAT-PMP RFC is still in draft state, it is not widely implemented yet. However, the same procedure could also be used for other traversal methods, which are similar to NAT-PMP. An example is the proprietary UPnP protocol, which is implemented on a larger number of NAT devices than the NAT-PMP protocol.

5.4.3 Port number prediction

According to the classification given in RFC 3489, it is not possible to obtain the port number of a binding on a Symmetric NAT, without first receiving a packet which has been sent through this binding. The reason is, that a symmetric NAT creates a new binding for each new destination of an IP packet.

However, RFC 4787 [57] shows, that some implementations of symmetric NATs may assign port numbers in a predictable fashion. [33] describes two possible ways how port numbers are typically assigned to a mapping. The first one is called port preservation. It tries to use the source port number of the outgoing UDP packet for the binding. Another strategy is to assign monotonically strictly increasing port numbers to the binding. Each time when a new binding is created, a port number which is increased by one or two is assigned. [33] also mentions NAT devices which assign random port numbers to their bindings, however they are not widely used.

The assignment strategy used by the NAT device can be used by a client in order to predict the port number which is used for a newly created binding. A client can use multiple STUN queries, in order to detect how port numbers are assigned by the NAT device. If the client detects, that port numbers are increased by one for each new binding, it can make an educated guess about the assignment of subsequent port numbers. Correctly guessing the port number of a binding on a symmetric NAT implementation would allow a client, located behind this NAT device, to establish a direct connection with another client, located behind another NAT device.

However, there are several complications when this NAT traversal technique is used:

- Multiple hosts or applications may open outgoing connections at the same time. On NAT implementations which assign increasing port numbers, bindings which are created by different applications may interference with each other. If an application creates two bindings one after another, the difference of the port numbers in these two bindings may be greater than one, if another application created a binding in between. This makes it difficult to predict an exact port number.
- A NAT device cannot implement either a pure port preservation strategy, nor a pure assignment of increasing port numbers. The reason is, that some of the port numbers may already be in use before the binding is created. As an example, suppose that two packets with the same source port number, but different source IP addresses, are sent to the same

destination. A symmetric NAT would create two different bindings, but as these two bindings must not use the same UDP port number, port preservation does only work for one of the bindings.

- If a NAT device assigns incrementing port numbers to bindings, it needs to reset the value, if it has reached the maximum value which can be assigned to a port number. However, bindings which have been created in a former iteration and which are still active, still allocate a port number. Therefore, the NAT device might need to skip this port numbers when assigning port numbers to new bindings.

To work around this problem port prediction algorithms do not try to predict a single port number, but a port range which is likely to contain the correct port number of the binding. Checking multiple port numbers gives the client a higher chance of guessing the correct port number.

Care must be taken regarding the size of the port range: If the range is too small, the chance that the right port number is located within the range is low. If the range is too large, the remote peer needs to check a large amount of port numbers for connectivity, a behavior which places unnecessary load on the network.

Chapter 6

Conclusion

In this section a recapitulation of the goals and results of the thesis is given. It includes the comparison of different NAT traversal techniques, as well as the proposal of new traversal techniques, which allow to improve upon the current state of VoIP protocols.

6.1 Summary

The goal of this thesis was to compare existing VoIP protocols in respect to their NAT traversal capabilities and to find possible weaknesses in current implementations.

Chapter 2 described the network protocols, on which VoIP operates. Chapter 3 introduced traversal techniques, which are able to bypass NAT devices. VoIP signaling protocols have been introduced in Chapter 4. Chapter 5 contains the comparative survey of the VoIP protocols as well as proposals for new traversal techniques.

In Section 5.2 the following VoIP signaling protocols have been evaluated:

- Session Initiation Protocol (SIP)
- Inter-Asterisk eXchange (IAX2)
- Extensible Messaging and Presence Protocol (XMPP) / Jingle

The comparison has shown, that all signaling protocols are capable of successfully traversing any kind of NAT device. IAX2 and XMPP relay the signaling data over a central server by default. This client-server setup allows them to avoid the problems typically caused by NAT devices. SIP provides the option of either using a direct or a relayed connection for signaling data. In the case of a direct connection, the traversal capabilities of SIP are comparable to IAX2 and XMPP.

The media traversal protocols which have been evaluated in Section 5.3 are:

- Simple Traversal of User Datagram Protocol (STUN)
- Relay Servers
- Interactive Connectivity Establishment (ICE)
- Inter-Asterisk eXchange (IAX2)

The IAX2 protocol is part of both evaluations, because it is used for signaling as well as media-data. For both types of data different characteristics are relevant, therefore the two evaluations yield different results.

A summarization of the results is shown in Figure 6.1. An “X” indicates that the protocol is able to traverse the NAT by using a direct connection. An “O” indicates, that the protocol is able to traverse the NAT device by using a relayed connection. A “-” denotes, that the particular protocol is not able to traverse the given NAT device.

	STUN	Relay Servers	ICE	IAX2
Full Cone NAT	X	O	X	X
Restricted Cone NAT	X	O	X	X
Port Restricted Cone NAT	X/-	O	X	O
Symmetric NAT	-	O	O	O
Hairpinning	-	O	X	O

Figure 6.1: Summarization of NAT Traversal Capabilities

When relay servers are used, the VoIP clients sent the data over a central server, instead of establishing a peer-to-peer connection. Therefore, each client only needs to be able to establish an outgoing connection to the media-server. This type of setup works across all types of NAT devices, but comes at the cost of additional latency and bandwidth requirements.

STUN always tries to establish a direct peer-to-peer connection. However, this does not succeed for all types of NAT devices. The protocol is able to successfully establish a connection when a

Full Cone NAT or a Restricted Cone NAT is used. With a Port Restricted Cone NAT, STUN may be operational, but it is not guaranteed for all cases. For other types of NAT devices, STUN will fail to establish a connection.

ICE and IAX2 exhibit a similar behavior. Both try to establish a direct peer-to-peer connection. If this fails, they fall back to an indirect relayed connection. This allows them to combine the benefits of STUN with the benefits of Relay Servers. A difference between these two protocols is, that ICE is able to establish a direct connection if hairpinning is not supported by the NAT device. IAX2 uses a relayed connection in this case.

Unlike the evaluation of the signaling protocols, the evaluation of the media-data showed essential differences between the results of different NAT traversal protocols for different types of NAT devices. The main reason is, that it is harder to establish a direct peer-to-peer connection than a relayed connection. While it is acceptable to use a relayed connection for the signaling data, it is advisable to use a direct connection for media-data, if possible.

6.2 Future prospects

The evaluation of NAT traversal techniques has shown, that the ICE protocol is able to establish connections in all of the examined cases. However, there are some cases, where ICE uses an inefficient traversal technique based upon relay servers, although the establishment of a direct connection would be possible.

The two main ideas for improvements of the ICE protocol are the support of the NAT-PMP protocol and support for the prediction of port numbers. Both techniques have been described in Section 5.4. While these features are not included in the ICE specification, clients can still implement them, without being incompatible to other clients.

If port number prediction is used, a carefully chosen size for the range of the checked port numbers needs to be selected. A number which is too low is less likely to identify the correct port, while a number which is too high places unnecessary load on the network.

While possible improvements for the ICE protocol have been identified, the ICE protocol as-is already provides very good NAT traversal capabilities. The only drawback is, that the specification artificially requires some types of candidates to be obtained either by the STUN or by the

TURN protocol. A removal of this rule would allow clients to support NAT traversal techniques not intended by the ICE protocol, without violating the specification.

The evaluation of the protocols has shown, that NAT traversal highly depends upon the types of NAT devices involved. However, at the same time a NAT traversal algorithm cannot safely predict, if the establishment of a particular connection will be possible. ICE shows an exemplary way how to deal with this issue. Instead of trying to make assumptions about the underlying network, it tries to establish a connection by using different traversal strategies. This allows ICE to utilize the best traversal technique for a given network setup.

Glossary

ASCII	American Standard Code for Information Interchange
ATM	Asynchronous Transfer Mode
B2BUA	Back-to-Back-User-Agent
HTTP	Hypertext Transfer Protocol
IANA	Internet Assigned Numbers Authority
IAX2	Inter-Asterisk EXchange
ICMP	Internet Control Message Protocol
JID	Jabber ID
LAN	Local area network
NAT	Network address translation
OSI	Open Systems Interconnection
PBX	Private branch exchange
PPP	Point-to-Point Protocol
PSTN	Public Switched Telephone Network
RFC	Request for Comments
RTP	Real-time Transport Protocol
SCTP	Stream Control Transmission Protocol

SDP	Session Description Protocol
SIP	Session Initiation Protocol
SMTP	Simple Mail Transfer Protocol
TCP	Transmission Control Protocol
UA	User-Agent
UAC	User-Agent Client
UAS	User-Agent Server
UDP	User Datagram Protocol
UTF-8	8-bit UCS/Unicode Transformation Format
VoIP	Voice over IP
XML	Extensible Markup Language
XMPP	Extensible Messaging and Presence Protocol

List of Figures

Chapter 7

Bibliography

[1] M. S. Blumenthal and D. D. Clark, "Rethinking the design of the Internet: the end-to-end arguments vs. the brave new world," *ACM Trans. Inter. Tech.*, vol. 1, no. 1, pp. 70–109, 2001.

[2] J. H. Saltzer, D. P. Reed, and D. D. Clark, "End-to-end arguments in system design," *ACM Transactions on Computer Systems*, vol. 2, no. 4, pp. 277–288, Nov. 1984. [Online]. Available: http://citeseer.ist.psu.edu/saltzer84endtoend.html

[3] P. Francis and R. Gummadi, "IPNL: A NAT-extended internet architecture." in *SIGCOMM*, 2001, pp. 69–80.

[4] J. Rosenberg, J. Weinberger, C. Huitema, and R. Mahy, "STUN - Simple Traversal of User Datagram Protocol (UDP) Through Network Address Translators (NATs)," RFC 3489 (Proposed Standard), Mar. 2003. [Online]. Available: http://www.ietf.org/rfc/rfc3489.txt

[5] B. Ford, D. Kegel, and P. Srisuresh, "Peer-to-Peer Communication Across Network Address Translators," in *Proceedings of the 2005 USENIX Technical Conference*, 2005. [Online]. Available: http://citeseer.ist.psu.edu/731089.html

[6] M. Holdrege and P. Srisuresh, "Protocol Complications with the IP Network Address Translator," RFC 3027 (Informational), Jan. 2001. [Online]. Available: http://www.ietf.org/rfc/rfc3027.txt

[7] H. Sinnreich, "The challenge of P2P Internet communications to network based services," *e & i Elektrotechnik und Informationstechnik*, vol. 123, no. 7-8, pp. 277–282, August 2006. [Online]. Available: http://www.springerlink.com/content/36ux4594gh60h07p/

[8] J. D. Day and H. Zimmermann, "The OSI reference model," *Proceedings of the IEEE*, vol. 71, no. 12, pp. 1334–1340, Dec. 1983.

[9] A. Tanenbaum, *Computer Networks*. Prentice Hall Professional Technical Reference, 2002.

[10] J. Postel, "Internet Protocol," RFC 791 (Standard), Sept. 1981, updated by RFC 1349. [Online]. Available: http://www.ietf.org/rfc/rfc791.txt

[11] Y. Rekhter, B. Moskowitz, D. Karrenberg, G. J. de Groot, and E. Lear, "Address Allocation for Private Internets," RFC 1918 (Best Current Practice), Feb. 1996. [Online]. Available: http://www.ietf.org/rfc/rfc1918.txt

[12] J. Postel, "Assigned numbers," RFC 790 (Historic), Sept. 1981, obsoleted by RFC 820. [Online]. Available: http://www.ietf.org/rfc/rfc790.txt

[13] ——, "User Datagram Protocol," RFC 768 (Standard), Aug. 1980. [Online]. Available: http://www.ietf.org/rfc/rfc768.txt

[14] L.-A. Larzon, M. Degermark, S. Pink, L.-E. Jonsson, and G. Fairhurst, "The Lightweight User Datagram Protocol (UDP-Lite)," RFC 3828 (Proposed Standard), July 2004. [Online]. Available: http://www.ietf.org/rfc/rfc3828.txt

[15] J. Postel, "Transmission Control Protocol," RFC 793 (Standard), Sept. 1981, updated by RFC 3168. [Online]. Available: http://www.ietf.org/rfc/rfc793.txt

[16] R. Braden, "Requirements for Internet Hosts - Communication Layers," RFC 1122 (Standard), Oct. 1989, updated by RFCs 1349, 4379. [Online]. Available: http://www.ietf.org/rfc/rfc1122.txt

[17] V. Jacobson, R. Braden, and D. Borman, "TCP Extensions for High Performance," RFC 1323 (Proposed Standard), May 1992. [Online]. Available: http://www.ietf.org/rfc/rfc1323.txt

[18] M. Allman, V. Paxson, and W. Stevens, "TCP Congestion Control," RFC 2581 (Proposed Standard), Apr. 1999, updated by RFC 3390. [Online]. Available: http://www.ietf.org/rfc/rfc2581.txt

[19] K. Ramakrishnan, S. Floyd, and D. Black, "The Addition of Explicit Congestion Notification (ECN) to IP," RFC 3168 (Proposed Standard), Sept. 2001. [Online]. Available: http://www.ietf.org/rfc/rfc3168.txt

[20] A.-V. T. W. Group, H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson, "RTP: A Transport Protocol for Real-Time Applications," RFC 1889 (Proposed Standard), Jan. 1996, obsoleted by RFC 3550. [Online]. Available: http://www.ietf.org/rfc/rfc1889.txt

[21] H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson, "RTP: A Transport Protocol for Real-Time Applications," RFC 3550 (Standard), July 2003. [Online]. Available: http://www.ietf.org/rfc/rfc3550.txt

[22] H. Schulzrinne and S. Casner, "RTP Profile for Audio and Video Conferences with Minimal Control," RFC 3551 (Standard), July 2003. [Online]. Available: http://www.ietf.org/rfc/rfc3551.txt

[23] M. Baugher, D. McGrew, M. Naslund, E. Carrara, and K. Norrman, "The Secure Real-time Transport Protocol (SRTP)," RFC 3711 (Proposed Standard), Mar. 2004. [Online]. Available: http://www.ietf.org/rfc/rfc3711.txt

[24] P. F. Tsuchiya and T. Eng, "Extending the IP internet through address reuse," *SIGCOMM Comput. Commun. Rev.*, vol. 23, no. 1, pp. 16–33, 1993.

[25] P. Srisuresh and M. Holdrege, "IP Network Address Translator (NAT) Terminology and Considerations," RFC 2663 (Informational), Aug. 1999. [Online]. Available: http://www.ietf.org/rfc/rfc2663.txt

[26] P. Srisuresh and K. Egevang, "Traditional IP Network Address Translator (Traditional NAT)," RFC 3022 (Informational), Jan. 2001. [Online]. Available: http://www.ietf.org/rfc/rfc3022.txt

[27] P. Koski, J. Ylinen, and P. Loula, "The SIP-Based System Used in Connection with a Firewall," in *Telecommunications, 2006. AICT-ICIW '06. International Conference on Internet and Web Applications and Services/Advanced International Conference on*, Feb. 2006, pp. 203–203.

[28] J. Rosenberg, "Interactive Connectivity Establishment (ICE): A Methodology for Network: Address Translator (NAT) Traversal for Offer/Answer Protocols," draft-ietf-mmusic-ice, Jan. 2007. [Online]. Available: http://tools.ietf.org/html/draft-ietf-mmusic-ice

[29] A. Kara, "Private-to-private communications over the internet," *Computer*, vol. 37, no. 5, pp. 53–59, May 2004.

[30] J. Rosenberg, "Session Traversal Utilities for (NAT) (STUN)," RFC 3489bis (work in progress), Mar. 2007. [Online]. Available: http://tools.ietf.org/html/draft-ietf-behave-rfc3489bis/

[31] S. Cheshire and M. Krochmal, "NAT Port Mapping Protocol (NAT-PMP)," draft-cheshire-nat-pmp, Sept. 2006. [Online]. Available: http://tools.ietf.org/id/draft-cheshire-nat-pmp

[32] G. Camarillo, "TCP-Based Media Transport in the Session Description Protocol (SDP)," draft-ietf-mmusic-sdp-comedia, Nov. 2004. [Online]. Available: http://tools.ietf.org/html/draft-ietf-mmusic-sdp-comedia

[33] S. Guha, Y. Takeda, and P. Francis, "NUTSS: a SIP-based approach to UDP and TCP network connectivity," in *FDNA '04: Proceedings of the ACM SIGCOMM workshop on Future directions in network architecture*. New York, NY, USA: ACM Press, 2004, pp. 43–48.

[34] Snom Technology AG, "snom 360," Apr. 2007. [Online]. Available: http://www.snom.com/en/snom360_voip_phone0.html

[35] Eyeball Networks Inc., "Eyeball AnyFirewall^{TM} Engine," Apr. 2007. [Online]. Available: http://www.eyeball.com/products/any_fw_engine.html

[36] M. Handley, H. Schulzrinne, E. Schooler, and J. Rosenberg, "SIP: Session Initiation Protocol," RFC 2543 (Proposed Standard), Mar. 1999, obsoleted by RFCs 3261, 3262, 3263, 3264, 3265. [Online]. Available: http://www.ietf.org/rfc/rfc2543.txt

[37] J. Rosenberg, H. Schulzrinne, G. Camarillo, A. Johnston, J. Peterson, R. Sparks, M. Handley, and E. Schooler, "SIP: Session Initiation Protocol," RFC 3261 (Proposed Standard), June 2002, updated by RFCs 3265, 3853, 4320. [Online]. Available: http://www.ietf.org/rfc/rfc3261.txt

[38] A. Johnston, *SIP: Understanding the Session Initiation Protocol, Second Edition*. Norwood, MA, USA: Artech House, Inc., 2003.

[39] G. Camarillo, *SIP Demystified*. McGraw-Hill Professional, 2001, foreword By-Jonathan Rosenberg.

[40] J. Rosenberg and H. Schulzrinne, "An Extension to the Session Initiation Protocol (SIP) for Symmetric Response Routing," RFC 3581 (Proposed Standard), Aug. 2003. [Online]. Available: http://www.ietf.org/rfc/rfc3581.txt

[41] M. Handley and V. Jacobson, "SDP: Session Description Protocol," RFC 2327 (Proposed Standard), Apr. 1998, obsoleted by RFC 4566, updated by RFC 3266. [Online]. Available: http://www.ietf.org/rfc/rfc2327.txt

[42] M. Handley, V. Jacobson, and C. Perkins, "SDP: Session Description Protocol," RFC 4566 (Proposed Standard), July 2006. [Online]. Available: http://www.ietf.org/rfc/rfc4566.txt

[43] J. Rosenberg and H. Schulzrinne, "An Offer/Answer Model with Session Description Protocol (SDP)," RFC 3264 (Proposed Standard), June 2002. [Online]. Available: http://www.ietf.org/rfc/rfc3264.txt

[44] H. Sinnreich and A. B. Johnston, *Internet communications using SIP: delivering VoIP and multimedia services with Session Initiation Protocol*. New York, NY, USA: John Wiley & Sons, Inc., 2001.

[45] M. Spencer, "IAX2: Inter-Asterisk eXchange Version 2," draft-mspencer-iax2-01, Oct. 2006. [Online]. Available: http://tools.ietf.org/id/draft-guy-iax-02.txt

[46] P. Saint-Andre, "Streaming XML with Jabber/XMPP," *IEEE Internet Computing*, vol. 9, no. 5, pp. 82–89, Sept./Oct. 2005.

[47] ——, "Jingle: Jabber Does Multimedia," *IEEE Multimedia*, vol. 14, no. 1, pp. 90–94, Jan./Mar. 2007.

[48] ——, "Extensible Messaging and Presence Protocol (XMPP): Core," RFC 3920 (Proposed Standard), Oct. 2004. [Online]. Available: http://www.ietf.org/rfc/rfc3920.txt

[49] ——, "Extensible Messaging and Presence Protocol (XMPP): Instant Messaging and Presence," RFC 3921 (Proposed Standard), Oct. 2004. [Online]. Available: http://www.ietf.org/rfc/rfc3921.txt

[50] ——, "Mapping the Extensible Messaging and Presence Protocol (XMPP) to Common Presence and Instant Messaging (CPIM)," RFC 3922 (Proposed Standard), Oct. 2004. [Online]. Available: http://www.ietf.org/rfc/rfc3922.txt

[51] J. Peterson, "Common Profile for Instant Messaging (CPIM)," RFC 3860 (Proposed Standard), Aug. 2004. [Online]. Available: http://www.ietf.org/rfc/rfc3860.txt

[52] P. Saint-Andre, "End-to-End Signing and Object Encryption for the Extensible Messaging and Presence Protocol (XMPP)," RFC 3923 (Proposed Standard), Oct. 2004. [Online]. Available: http://www.ietf.org/rfc/rfc3923.txt

[53] J. Hildebrand, C. Kaes, and D. Waite, "Jabber HTTP Polling," XEP-0025 (Deprecated), July 2006. [Online]. Available: http://www.xmpp.org/extensions/xep-0025.html

[54] I. Paterson, D. Smith, and P. Saint-Andre, "Bidirectional-streams Over Synchronous HTTP (BOSH)," XEP-0124 (Standards Track), July 2006. [Online]. Available: http://www.xmpp.org/extensions/xep-0124.html

[55] J. Postel, "Echo Protocol," RFC 862 (Standard), May 1983. [Online]. Available: http://www.ietf.org/rfc/rfc862.txt

[56] P. Saint-Andre and S. Egan, "STUN Server Discovery for Jingle," XEP-0215 (Standards Track), May 2007. [Online]. Available: http://www.xmpp.org/extensions/xep-0215.html

[57] F. Audet and C. Jennings, "Network Address Translation (NAT) Behavioral Requirements for Unicast UDP," RFC 4787 (Best Current Practice), Jan. 2007. [Online]. Available: http://www.ietf.org/rfc/rfc4787.txt

[58] A. Takahashi, H. Yoshino, and N. Kitawaki, "Perceptual QoS assessment technologies for VoIP," *IEEE Communications Magazine*, vol. 42, no. 7, pp. 28–34, July 2004.

[59] A. Lakaniemi, J. Rosti, and V. I. Raisanen, "Subjective VoIP speech quality evaluation based on networkmeasurements," in *Communications, 2001. ICC 2001. IEEE International Conference on*, vol. 3, Helsinki, Finland, 2001, pp. 748–752.

[60] S. L. Garfinkel, "Skype Security Overview - VoIP and Skype Security," Jan. 2005. [Online]. Available: http://www.simson.net/ref/2005/OSI_Skype6.pdf

[61] L. Roychoudhuri, E. Al-Shaer, H. Hamed, and G. B. Brewster, "On studying the impact of the Internet delays on audio transmission," in *IP Operations and Management, 2002 IEEE Workshop on*, 2002, pp. 208–213.

[62] B. Duysburgh, S. Vanhastel, B. De Vreese, C. Petrisor, and P. Demeester, "On the influence of best-effort network conditions on the perceivedspeech quality of VoIP connections," in *Computer Communications and Networks, 2001. Proceedings. Tenth International Conference on*, Scottsdale, AZ, USA, 2001, pp. 334–339.

[63] D. Cohen, "Specifications for the Network Voice Protocol (NVP)," RFC 741, Nov. 1977. [Online]. Available: http://www.ietf.org/rfc/rfc741.txt

[64] B. Goode, "Voice over Internet protocol (VoIP)," *Proceedings of the IEEE*, vol. 90, no. 9, pp. 1495–1517, Sept. 2002.

[65] D. Bergström, "An analysis of Skype VoIP application for use in a corporate environment." Oct. 2004. [Online]. Available: http://www.geocities.com/bergstromdennis/Skype_Analysis_1_3.pdf

[66] M. Borella, J. Lo, D. Grabelsky, and G. Montenegro, "Realm Specific IP: Framework," RFC 3102 (Experimental), Oct. 2001. [Online]. Available: http://www.ietf.org/rfc/rfc3102.txt

[67] K. Egevang and P. Francis, "The IP Network Address Translator (NAT)," RFC 1631 (Informational), May 1994, obsoleted by RFC 3022. [Online]. Available: http://www.ietf.org/rfc/rfc1631.txt

[68] V. Cerf and R. Kahn, "A Protocol for Packet Network Intercommunication," *Communications, IEEE Transactions on [legacy, pre - 1988]*, vol. 22, no. 5, pp. 637–648, May 1974.

[69] M. Walfish, J. Stribling, M. Krohn, H. Balakrishnan, R. Morris, and S. Shenker, "Middleboxes No Longer Considered Harmful," in *6th Usenix OSDI*, San Francisco, CA, December 2004.

[70] R. Zheng, "netfilter mailinglist posting: iptables and SIP," Dec. 2004. [Online]. Available: http://lists.netfilter.org/pipermail/netfilter/2004-December/057518.html

[71] A. Georgescu, "Best practices for SIP NAT traversal," July 2005. [Online]. Available: http://mediaproxy.ag-projects.com/NATtraversal-BestPractices.pdf

[72] D. D. Clark, "The design philosophy of the DARPA Internet Protocols," *SIGCOMM Comput. Commun. Rev.*, vol. 25, no. 1, pp. 102–111, 1995.

[73] T. Sung, "TCP/IPX Connection Mib Specification," RFC 1792 (Experimental), Apr. 1995. [Online]. Available: http://www.ietf.org/rfc/rfc1792.txt

[74] T. Hoeher, S. Tomic, and R. Menedetter, "SIP collides with IPv6," *icns*, vol. 0, p. 10, 2006.

[75] D. Wing and J. Rosenberg, "Controlling NAT Bindings using STUN," Apr. 2007. [Online]. Available: http://tools.ietf.org/pdf/draft-wing-behave-nat-control-stun-usage-00.pdf

[76] J.-M. Valin, "Speex: A Free Codec For Free Speech," Dec. 2005. [Online]. Available: http://people.xiph.org/~jm/papers/speex_lca2006.pdf

[77] Y. Amir, C. Danilov, S. Goose, D. Hedqvist, and A. Terzis, "1-800-OVERLAYS: Using Overlay Networks to Improve VoIP Quality." [Online]. Available: http://citeseer.ist.psu.edu/amir04overlays.html

[78] B. A. Miller, T. Nixon, C. Tai, and M. D. Wood, "Home networking with Universal Plug and Play," vol. 39, no. 12, pp. 104–109, Dec. 2001.

[79] C. Bettstetter and C. Renner, "A Comparison of Service Discovery Protocols and Implementation of the Service Location Protocol," in *Proceedings EUNICE 2000, Sixth EUNICE Open European Summer School: Innovative Internet Applications*, Twente, Netherlands, 2000.

[80] P. Iyer and U. Warrier, "Internet Gateway Device (IGD) Standardized Device Control Protocol V 1.0," Nov. 2001. [Online]. Available: http://upnp.org/resources/standards.asp

[81] Y. Takeda, "Symmetric NAT Traversal using STUN," draft-takeda-symmetric-nat-

traversal-00, June 2003. [Online]. Available: http://www.cs.cornell.edu/projects/stunt/draft-takeda-symmetric-nat-traversal-00.txt

[82] A. Ganjam and H. Zhang, "Connectivity restrictions in overlay multicast," in *NOSSDAV '04: Proceedings of the 14th international workshop on Network and operating systems support for digital audio and video*. New York, NY, USA: ACM Press, 2004, pp. 54–59.

[83] S. Walberg, "How to configure SIP and NAT," *Linux J.*, vol. 2007, no. 155, p. 3, 2007.

[84] E. Kohler, R. Morris, and M. Poletto, "Modular components for network address translation," in *Open Architectures and Network Programming Proceedings, 2002 IEEE*, 2002, pp. 39–50.

[85] H. Khlifi, J. C. Gregoire, and J. Phillips, "VoIP and NAT/firewalls: issues, traversal techniques, and a real-world solution," *IEEE Communications Magazine*, vol. 44, no. 7, pp. 93–99, July 2006.

[86] S. Guha and P. Francis, "Characterization and measurement of TCP traversal through NATs and firewalls," 2005. [Online]. Available: http://citeseer.ist.psu.edu/guha05characterization.html

[87] R. Stewart, Q. Xie, K. Morneault, C. Sharp, H. Schwarzbauer, T. Taylor, I. Rytina, M. Kalla, L. Zhang, and V. Paxson, "Stream Control Transmission Protocol," RFC 2960 (Proposed Standard), Oct. 2000, updated by RFC 3309. [Online]. Available: http://www.ietf.org/rfc/rfc2960.txt