Table of Contents

Network Address Translation (NAT) is being used by many service providers and private individuals as a way to get around the problem of not having enough IP addresses. An enterprise may have a block of IP addresses assigned to them, but many more computers than the allocated IP addresses. Alternatively, an individual may have a DSL connection with one IP address, but want to have multiple computers hooked up to the Internet. NAT solves this problem by mapping internal addresses to external or public addresses. An internal IP address:port pair is mapped to an external IP:port, and whenever the NAT receives a packet with the external IP:port, it knows how to reroute the packet back to the internal IP address and port. The mapping is valid for some predefined mapping interval after which, in the absence of network traffic between the two communicating parties, this mapping may be expunged. In all cases, we must assume that an application will send and receive packets on the same port.
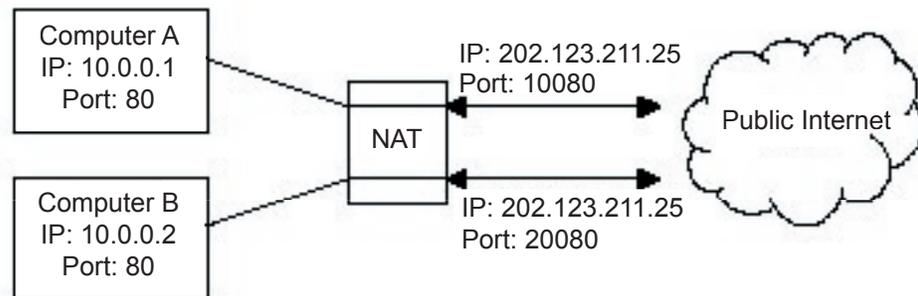

Figure 1: NAT schematic

There are four types of NATs. As defined in [1] they are:

1. Full Cone
2. Restricted Cone
3. Port Restricted Cone
4. Symmetric

For a given internal address, the first three types of NAT maintain a mapping of this internal address that is independent of the destination address being sought. The fourth type of NAT will allocate a new mapping for each independent destination address.

Unless the NAT has a static mapping table, the mapping that opens when the first packet is sent out from a client through the NAT may only be valid for a certain amount of time (typically a few minutes), unless packets continue to be sent and received on that IP:port.

Full Cone

In the case of the full cone, the mapping is well established and anyone from the public Internet that wants to reach a client behind a NAT, needs only to know the mapping scheme in order to send packets to it. For example, a computer behind a NAT with IP 10.0.0.1 sending and receiving on port 8000, is mapped to the external IP:port on the NAT of 202.123.211.25:12345. Anyone on the Internet can send packets to that IP:port and those packets will be passed on to the client machine listening on 10.0.0.1:8000.
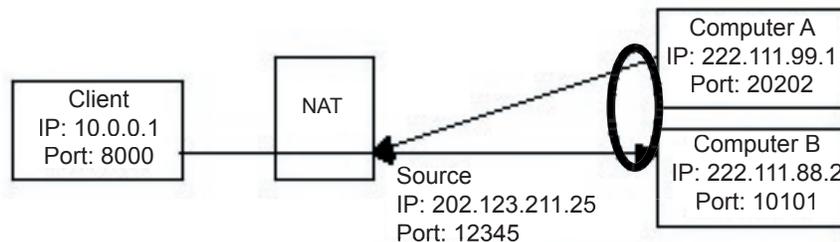

Figure 2: Full Cone NAT

Restricted Cone

In the case of a restricted cone NAT, the external IP:port pair is only opened up once the internal computer sends out data to a specific destination IP. For example, in the case where the client sends out a packet to external computer 1, the NAT maps the client's 10.0.0.1:8000 to 202.123.211.25:12345, and External 1 can send back packets to that destination. However, the NAT will block packets coming from External 2, until the client sends out a packet to External 2's IP address. Once that is done, both External 1 and External 2 can send packets back to the client, and they will both have the same mapping through the NAT.

Port Restricted Cone

A port restricted cone type NAT is almost identical to a restricted cone, but in this case the NAT will block all packets unless the client had previously sent out a packet to the IP AND port that is sending to the NAT. So if the client sends to External 1 to port 10101, the NAT will only allow through packets to the client that come from 222.111.88.2:10101. Again, if the client has sent out packets to multiple IP:port pairs, they can all respond to the client, and all of them will respond to the same mapped IP:port on the NAT.

Symmetric

The last type of NAT – symmetric - is different from the first three in that a specific mapping of internal IP:port to the NAT's public IP:port is dependant on the destination IP address that the packet is sent to. So for example, if the client sends from 10.0.0.1:8000 to Computer B, it may be mapped as 202.123.211.25:12345, whereas if the client sends from the same port (10.0.0.1:8000) to a different IP, it is mapped differently (202.123.211.25:45678).
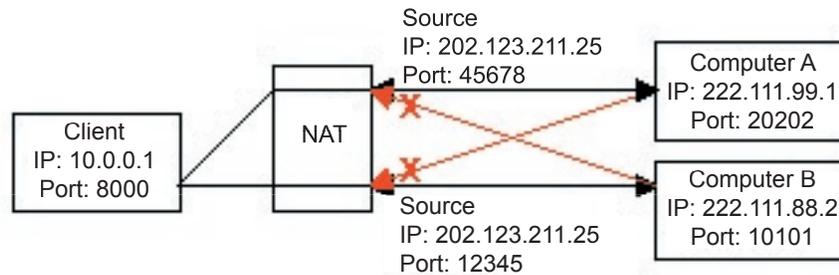

Figure 3: Symmetric NAT

Computer B can only respond to it's mapping and Computer A can only respond to it's mapping. If either one tries to send to the other's mapped IP:port, those packets will be dropped. As in the case of the restricted NAT, the external IP:port pair is only opened up once the internal computer sends out data to a specific destination.

There are two parts to a SIP-based phone call. The first is the signaling – that is the protocol messages that set up the phone call – and the second is the actual media stream, i.e. the RTP packets that travel directly between the end devices (e.g. the client and gateway).

SIP Signaling

The SIP signaling can traverse NATs in a fairly straightforward way, since there is typically one proxy, the first hop away from NAT, that receives SIP messages from the client (through the NAT) and then returns messages to the same place. The proxy needs to return SIP packets on the same port it received them to the IP:port that the packets were sent from (not to any standard SIP port, e.g. 5060). SIP has tags that tell the proxy to do this – the "received" tag tells the proxy to return a packet to a specific IP and the "rport" tag [2] keeps the port to return to. Most proxies have not yet implemented the "rport" tag, and some clients will not parse the SIP messages correctly if these tags are present, but at least in principle the mechanism does exist for NAT traversal. Another, simpler way to traverse NATs is to use TCP for SIP signaling between the client and the proxy. Since the TCP connection is opened through the NAT directly from client to proxy, the SIP signaling will proceed unhindered. Again, many proxies have not yet implemented the TCP option and only work using UDP.

Note that SIP signaling should be able to traverse any of the four types of NATs as long as the proxy returns SIP messages to the NAT from the same source port that it received the initial message on. The initial SIP message, sent to the proxy IP:port, opens up the mapping on the NAT, and the proxy returns packets to the NAT from that same IP:port. This is allowed in any NAT scenario.

Registering a client that is behind a NAT requires either a registrar that can save the IP:port in the registration information based on the port and IP that it sees as the source of the SIP message, or a client that is aware of its external mapped address and port and can insert them into the Contact information as the IP:port to receive SIP messages. Care should be taken to use a registration interval shorter than the keep alive time for the NAT mapping.

RTP – Media Stream

The RTP that must traverse a NAT does not allow for as easy a solution as the SIP signaling. In the case of RTP, the SIP message body contains the information that the endpoints need in order to communicate directly with each other. This information is contained in the SDP message. The end point clients fill in that information according to what they know about themselves. A client sitting behind a NAT knows only its internal IP:port, and that is what it puts in the SDP body of the outgoing SIP message. When the destination endpoint wants to start sending packets to the originating endpoint, it will use the received SDP information containing the internal IP:port of the originating endpoint and the packets never get there. Here is an example of a trace of an INVITE message from a SIP client behind a NAT as received by the gateway. There is a proxy in the middle.

```
001     INVITE sip:12125551212@211.123.66.222 SIP/2.0
002     Via: SIP/2.0/UDP 211.123.66.223:5060;branch=a71b6d57-507c77f2
003     Via: SIP/2.0/UDP 10.0.0.1:5060;received=202.123.211.25;rport=12345
004     From: <sip:2125551000@211.123.66.223>;tag=108bcd14
005     To: sip: 12125551212@211.123.66.222
006     Contact: sip: 2125551000@10.0.0.1
007     Call-ID: 4c88fd1e-62bb-4abf-b620-a75659435b76@10.3.19.6
008     CSeq: 703141 INVITE
009     Content-Length: 138
010     Content-Type: application/sdp
011     User-Agent: HearMe SoftPHONE
012
013     v=0
014     o=deltathree 0 0 IN IP4 10.0.0.1
015     s=deltathree
016     c=IN IP4 10.0.0.1
017      t=0 0
018     m=audio 8000 RTP/AVP 4
019     a=ptime:90
020     a=x-ssrc:00aea3c0
```

In the above trace, the IP address in line 003 of the SIP header is the IP address that the client thinks it is – i.e. the internal IP address (10.0.0.1). But the proxy knows from which IP address it actually received the packet, so it adds the "received" and "rport" tags with the IP address and port after the NAT mapping. These tags allow the proxy to forward SIP messages back to the client via the NAT.

But the information that is used in order to pass voice data through – the RTP connection – is held lower down in the message, in lines 014 and 016. The client expects to receive on port 8000 (m=) at IP 10.0.0.1 (c=), which is the IP:port that it sees itself as, and that is where the second endpoint will return packets to. The result will be that the call is set up (since the SIP goes through) but no voice is received.

If the client is behind one of the first three NAT types, then the solution for NAT traversal is fairly simple. The client must find out how its internal IP:port looks to the world (i.e. the NAT mapping) and then it must put that information into the SDP message instead of the information reflecting its internal IP:port. There are two methods for a client to determine the NAT mapped public IP:port. The first is to ask the NAT, the second is to ask someone outside the NAT on the public Internet.

UPnP

A client can ask the NAT how it would map a particular IP:port through a protocol called Universal Plug and Play (UPnP). This is a solution that is being pushed by Microsoft (among others). The client queries the NAT via UPnP asking what mapping it should use if it wants to receive on port x. The NAT responds with the IP:port pair that someone on the public Internet should use to reach the client on that port. Many NAT device manufacturers have already included UPnP in their products.

One problem with UPnP is that it will not work in the case of cascading NATs. For example, say an ISP owns a block of IP addresses, but not enough to service its user base. The ISP would use a NAT to provide IP addresses to its customers. One of those customers may require many IP addresses (for example, an internet café) so it would set up its own NAT to share its one address between many computers. If a client running on one of the local computers were to use UPnP to determine its public IP:port, then it would only get back the innermost mapping (that of the internet café's NAT) but would still have a one way voice problem. That is because the public Internet would still not recognize the IP:port that the client was giving, since a second translation occurs between the internet café's NAT and the public Internet via the ISP's NAT. There are also security issues that have not yet been addressed with UPnP.

Furthermore, there is a huge installed base of existing NATs that do not support UPnP. It is not realistic to retrofit this installed base in the near future.

External Query

In the absence of a method of communicating with the NAT device, the next best way for a client to determine its external IP:port is to ask a server sitting outside the NAT on the public Internet how it sees the source of a packet coming from this client. In this scenario, a server sits listening for packets (call this a NAT probe). When it receives a packet, it returns a message from the same port to the source of the received packet containing the IP:port that it sees as the source of that packet. In every case (all 4 NAT cases), the client will receive the return packet. The client can then determine

1. If it is behind a NAT (if the IP:port contained within the return packet is different than the IP:port that it thinks it is).

2. Which public IP:port it should use in the SDP message in order for the endpoint to reach it.

For example, if the client wants to be reached on 10.0.0.1:8000, it will first send out a query to the NAT probe from port 8000. The NAT probe will actually receive the query packet from 202.123.211.25:12345 and so it will respond to that IP:port with a packet containing 202.123.211.25:12345. The client then puts into its SDP "m= AUDIO 12345" and "c=202.123.211.25" while the client itself listens on 10.0.0.1:8000.
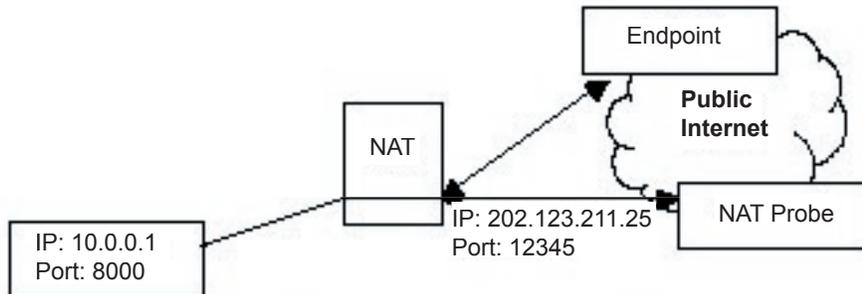

Figure 4: Discovery of public IP:port

This will work with the following stipulations:

1. The client must send and receive RTP on the same port.

2. The client must send out the SIP message shortly after sending out a query to the NAT probe. If there is a long delay, the mapping may change.

3. In the case of Restricted Cone or Port Restricted Cone NATs, the client must send out a packet to the endpoint before the NAT will allow packets from the endpoint through to the client.

This will not work in the case of symmetric NATs, since the IP address of the NAT probe is different than that of the endpoint, and therefore the mapping the NAT probe sees is different than the mapping that the endpoint would use to send packets through to the client on that IP:port.

STUN

Simple Traversal of UDP Through NATs (STUN) [4] is a protocol for setting up the kind of NAT Probe that was just described. It actually does a bit more than just return the public IP:port – it can also help determine which kind of NAT you are behind. Clients are already being developed that are STUN aware and can set their SDP messages accordingly.

STUN requests specify the following parameters:

■ RESPONSE-ADDRESS

- Change IP

- Change Port

The STUN server will send its response to the IP:port specified in the RESPONSE-ADDRESS attribute. If that field is not present, then the server sends its response to the IP:port that it received the request from.

If both the Change IP and Change Port flags are not set, the STUN server responds from the IP: port that the initial packet was sent to. If the change IP flag is set, the server replies from a different IP, and if the Change Port flag is set, the server replies from a different port.

The STUN response contains the following information:

- MAPPED-ADDRESS – the IP:port of the client as seen by the first STUN server outside the NAT to receive the STUN request.

- CHANGED-ADDRESS - the IP address that would be the source of the returned response if the request had the change IP flag set.

- SOURCE-ADDRESS – the IP:port where the STUN response was sent from.

Using a combination of different requests to a STUN server, a client can determine:

- If it is on the open Internet

- If it is behind a firewall that blocks UDP

- If it is behind a NAT, and what type of NAT it is behind

Automatic Detection of NAT Environment

Four tests are required in order for the client to determine the environment that it is situated in. The following table shows the parameters set and responses that are expected from each of these tests. Assume that there are two STUN servers available, IP1 and IP2, and they can return responses either from port 1 or port 2. Sending a request to IP1 without the Change IP or Change Port flags set will cause the STUN server to respond from IP1, port 1. Setting the Change IP flag will elicit a response from IP2:1, etc.

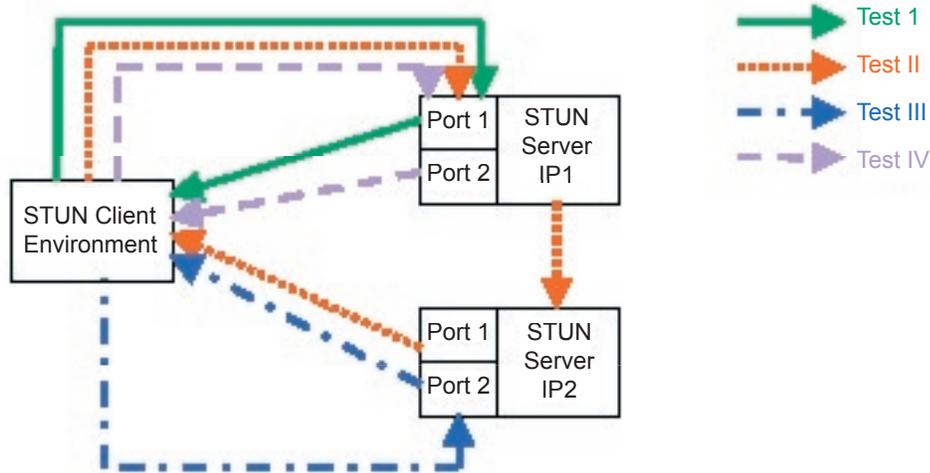| Test	Destination	Change IP	Change Port	Return IP:port
Test I	*IP1:1*	*N*	*N*	*IP1:1*
Test II	*IP1:1*	*Y*	*Y*	*IP2:2*
Test III	*IP2:1*	*N*	*N*	*IP2:1*
Test IV	*IP1:1*	*N*	*Y*	*IP1:2*

Figure 5: Test required for NAT environment discovery

In order for the client to discover its NAT environment, these four tests are run according to the following flow:

1. Test I is performed.

   ✔ If no response is received, then the client knows it is behind a firewall that blocks UDP.

2. If a response is received, the IP address in the MAPPED-ADDRESS field of the STUN response is tested against what the client thinks its IP address is.

3. If the IP addresses match, Test II (Change IP and Port) is run.

   ✔ If there is no response, then the client is behind a symmetric UDP firewall – that is its IP address is on the open Internet, but its firewall will only allow UDP in from a given destination once the client has sent a packet out to that destination.

   ✔ If the client receives a response, then the client knows that it is on the open Internet unblocked.

4. If the IP addresses in step 2 are not the same, Test II is run.

✔ If the client receives a response, then it is behind a Full Cone NAT.

5. If no response is received, the client runs Test III and tests the IP address that is returned in the STUN response's MAPPED-ADDRESS field (coming from IP 2) against the MAPPED-ADDRESS that was returned in Test I (from IP 1).

✔ If the two IP addresses are not the same, then the client is behind a Symmetric NAT.

6. If the two IP addresses are the same, the client runs Test IV (Change Port).

✔ If a response is received, then the client is behind a Restricted NAT.

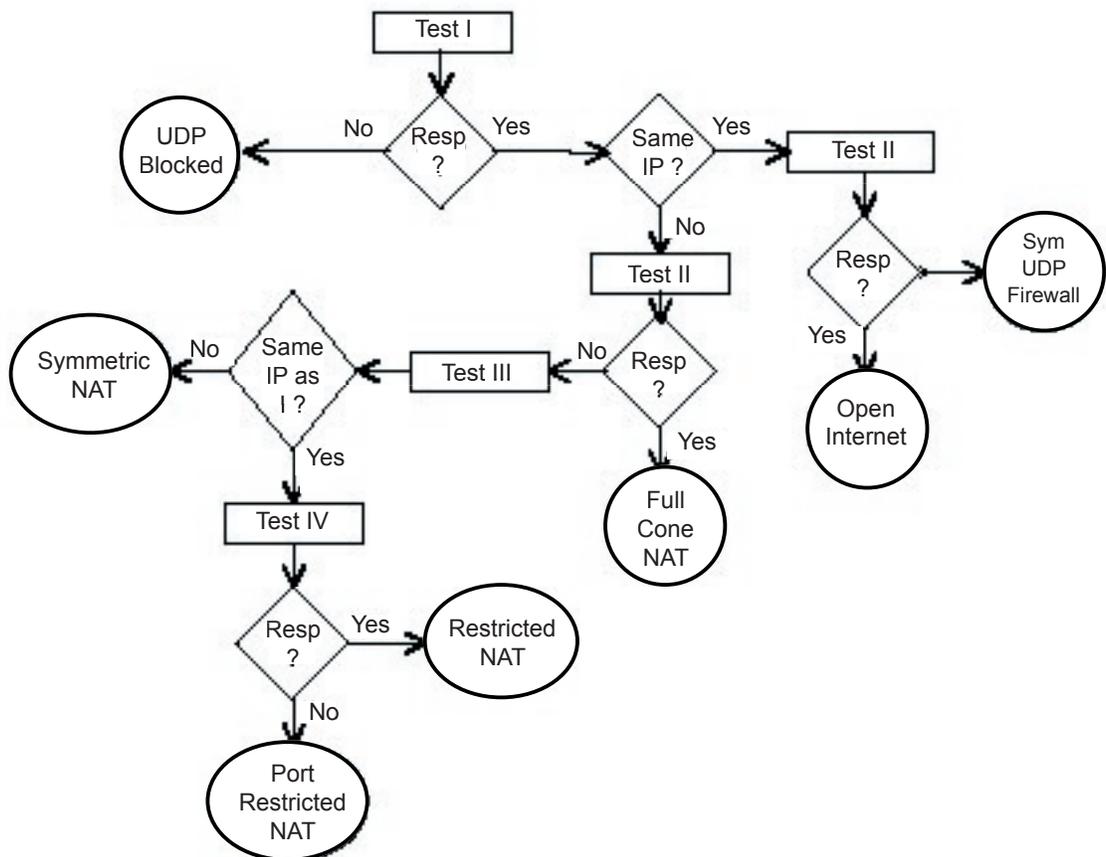✔ If no response is received, then the client is behind a Port Restricted NAT.

Figure 6: Flow of algorithm for NAT discovery

Connection Oriented Media

The above solution (NAT probe or STUN server) will only work for the first 3 types of NAT. The 4th case – symmetric NATs – will not allow this scheme since they have different mappings depending on the target IP address. So the mapping that the NAT assigns between the client and the NAT probe is different than that assigned between the client and the gateway.

In the case of a symmetric NAT, the client must send out RTP to, and receive RTP back from the same IP address. Any RTP connection between an endpoint outside a NAT and one inside a NAT must be established point-to-point, and so (even if a SIP connection has already been established) the endpoint outside the NAT must wait until it receives a packet from the client before it can know where to reply. This is known as Connection Oriented Media.

If an endpoint is meant to speak both to clients that are behind NATs and clients on the open Internet, then it must know when it can trust the SDP message that it receives in the SIP message, and when it needs to wait until it receives a packet directly from the client before it opens a channel back to the source IP:port of that packet.

One proposal [5] for informing the endpoint to wait for the incoming packet is to add a line to the SDP message (coming from the client behind the NAT):

a=direction:active

When the endpoint reads this line, it understands that the initiating client will "actively" set up the IP:port to which the endpoint should return RTP, and that the IP:port found in the SDP message should be ignored.

SIP clients do not currently support the 'a=' tag described here. Until they do, there will have to be some kind of 'translator' inserted into the SIP flow that can key off some other cue in order to determine that the client is behind a Symmetric NAT. Once it makes that determination, the 'translator' will insert the a=direction:active line into the SDP of the SIP message. We suggest that setting the IP address in the 'c=' line of the SDP to 0.0.0.0 is a suitable way of cueing the 'translator' and instructing it to insert the appropriate tag into the message.

RTP-Relay

If an endpoint supports Connection Oriented Media, then the problem of symmetric NAT traversal is solved. Two scenarios are still problematic:

1. If the endpoint does not support the a=direction:active tag.

2. If both endpoints are behind Symmetric NATs

In either of these cases, one solution is to have an RTP Relay in the middle of the RTP flow between endpoints. The RTP Relay acts as the second endpoint to each of the actual endpoints that are attempting to communicate with each other. Typically, there would be a server in the

middle of the SIP flow (herein called a NAT Proxy) that would manipulate the SDP in such a way as to instruct the endpoints to send RTP to the relay instead of directly to each other. The Relay would set up its own internal mapping of a session, noting the source IP:port of each endpoint sending it RTP packets. It then uses that mapping to forward the RTP from endpoint to endpoint.

The following is a typical call flow that might be instantiated between a User Agent behind a symmetric NAT and a voice gateway on the open Internet:
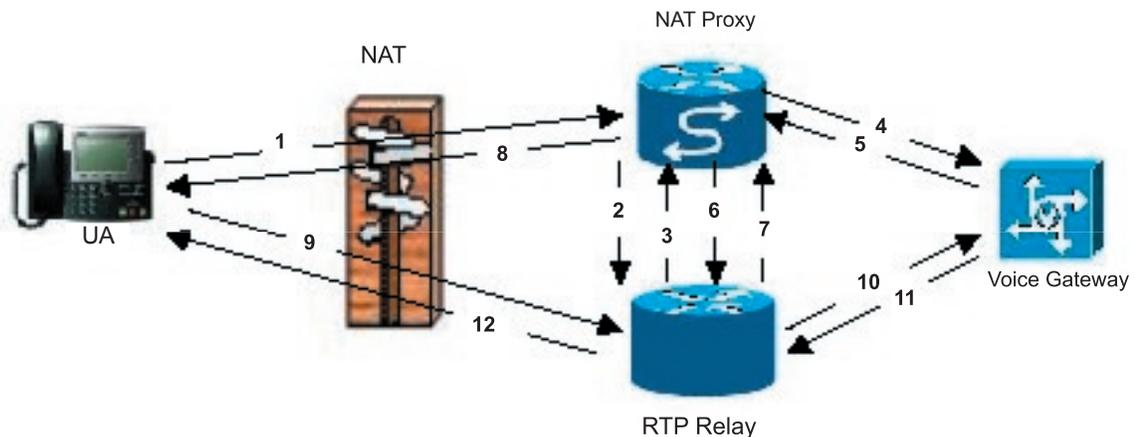
Call Flow

1. UA sends an INVITE to the NAT Proxy through the NAT

2. The NAT Proxy contacts the RTP Relay and requests it to set up a session.

3. The RTP Relay assigns an available pair of ports to this Call. It responds to the NAT Proxy with downstream available port in RTP Relay. The NAT Proxy uses this to modify the SDP information of the received INVITE request.

4. The NAT Proxy forwards the SIP INVITE request with modified SDP (reflecting the RTP Relay's IP:port) on to the Voice Gateway.

5. The Gateway replies (in the 200 OK) with its own SDP information including the port to receive RTP packets.

6. The NAT Proxy contacts the RTP Relay to supply the IP:port of the gateway (if the gateway was also behind a symmetric NAT, then the NAT Proxy would instruct the Relay to wait for packets from the Gateway before setting the IP:port to forward RTP on to the Gateway).

7. The Relay responds to the NAT Proxy with the upstream available RTP Port.

8. The NAT Proxy forwards the response upstream back to the UA after modifying the response SDP with the IP:port of the RTP Relay.

9. UA begins sending RTP to the IP:port it received in the 200 OK – to the RTP Relay.

10. RTP Relay notes the IP:port that it received the packet from (for the first packet), and passes on the packet to the IP:port of the gateway.

11. RTP packets proceed from the gateway to the RTP Relay.

12. The RTP Relay forwards those packets to the client (according to IP:port that it saved when it received the first RTP packet from the client).

When BYE is received by the NAT Proxy, it forwards this information over to the RTP Relay which tears down the session.

The following considerations should be noted:

1. The client will always need to send and receive RTP on the same port.

2. This solution will work for all types of NATs, but because of the delay associated with the RTP Relay (which may be substantial, especially if the RTP Relay is not close to at least one of the endpoints), it should probably not be used unless a Symmetric NAT is involved. In other NAT scenarios, modification of the SDP will be sufficient.

3. The client will not hear any voice until the first packet is sent to the RTP Relay. That could cause problems when receiving a 183 message as part of the call setup, since the gateway at that point opens a one-way media stream andpasses back network announcements over that stream. If the client has not yet sent its first RTP packet, the RTP relay does not yet know its public IP:port address.

4. This is just one way of implementing an RTP Relay. There are other possibilities, including schemes that do not insert themselves into the SIP flow.

Everything that has been discussed up until now - in the case of outbound calls – can be applied to inbound calling as well. Register messages will have to be implemented as per the discussion in section 3.1.. Once the issue of the SIP NAT traversal has been solved, the same issues discussed above can be implemented.

1. SDP Manipulation: If the client behind a NAT receives an INVITE, it will go out to a STUN server to find the appropriate IP:port mapping and insert that into the SDP message in the 200 OK that it returns.

2. Connection Oriented Media: The client will return the 'a=direction:active' line in the SDP of the 200 OK. In the case where this is not implemented in the client, it can use the 'c=0.0.0.0' cue in the 200 OK for a translator to pick up and insert the 'a=' line.

3. The RTP Relay and NAT Proxy will function accordingly, manipulating the SDP of the 200 OK instead of or in addition to the INVITE.

[1] "Short Term NAT Requirements for UDP Based Peer-to-Peer Applications", IETF Draft, C. Hitema, Feb. 2001. Work in progress

[2] "NAT Friendly SIP", IETF Draft, draft-rosenberg-sip-entfw-02, J. Rosenberg, J. Weinberger, H. Schulzrinne, July 20, 2001.

[3] "NAT and Firewall Scenarios and Solutions for SIP", IETF Draft,
 draft-rosenberg-sipping-nat-scenarios-00, Rosenberg, Mahy, Sen, November 14, 2001.

[4] "STUN – Simple Traversal of UDP Through NATs", IETF Draft, draft-rosenberg-midcom-stun-00, Rosenberg, Weinberger, Huitema, Mahy, October 1, 2001.

[5] "Connection-Oriented Media Transport in SDP", IETF Draft, draft-ietf-mmusic-sdp-comedia-01, D. Yon, October, 2001.

Figure 4: Discovery of public IP:port

Figure 5: Tests required for NAT environment discovery

Figure 6: Flow of algorithm for NAT discovery

Figure 7: Call flow with RTP Relay