Internet Engineering Task Force   SIP WG
Internet Draft   J.Rosenberg,J.Weinberger,H.Schulzrinne
draft-rosenberg-sip-entfw-02.txt   dynamicsoft,Columbia U.
July 20, 2001
Expires: February 2002

NAT Friendly SIP

STATUS OF THIS MEMO

Abstract

In this draft, we discuss how SIP can traverse enterprise and residential NATs. This environment is challenging because we assume here that the end user or SIP provider has no control over the NAT, and that the NAT is completely ignorant of SIP. Our approach is to make SIP "NAT friendly", with a few minor, backwards compatible extensions. These extensions allow UDP and TCP-based SIP to traverse NATs. We also handle RTP traversal using a combination of symmetric (aka connection-oriented) RTP and a new NAT detection and binding discovery mechanism. The results of the approach are that direct UDP-based RTP is used whenever provably possible in any given nat configuration. We use a network intermediary - in our case, an off-the-shelf router - to handle the case when both caller and called party are behind symmetric NATs. Our approach for binding discovery is effectively a pre-midcom solution that allows binding allocations by talking to a server behind the nat, rather than talking to the nat

directly.

1 Introduction

The problem of getting applications through NATs has received a lot of attention [1]. Getting SIP through NATs is particularly troublesome. In a previous draft [2] we discussed some of the general issues regarding traversal of firewalls, and discussed some solutions for it. Our solutions were based on having a proxy server control the firewall/NAT with a control protocol of some sort [3]. This protocol can open and close pinholes in the firewall, and/or obtain NAT address bindings to use in rewriting the SDP in a SIP message.

The use of a control protocol in the midcom architecture is ideal for carriers, but it does not work when the SIP service provider is not the same as the ISP and transport provider of the end user. This is frequently the case for users behind enterprise and NATs who are trying to access SIP services outside of their networks. The same happens for residential NATs. These devices are often used by consumers who have cable modem and DSL connections, and wish to connect multiple computers using the single address provided by the cable company or DSL company. [1] often referred to as cable/DSL routers, and are manufactured by companies like Linksys, Netopia, and Netgear.

Ultimately, it is our belief and hope that NATs will disappear with the deployment of IPv6. However, that is not likely to happen for some time.

Given the existence of NATs, one way to handle SIP is to embed a SIP ALG within enterprise NATs. However, this has not happened. The top commercial NAT products continue to be SIP-unaware. Even if SIP ALG support were added tomorrow, there is still a huge installed based of NATs that do not understand SIP. As a result, there is going to be a long period of time during which users will be behind NATs that are ignorant of SIP, probably at least two to three years. The SIP community cannot wait for ubiquituous deployment of SIP aware NATs. Interim solutions are needed NOW to enable SIP services to be delivered to users behind these devices.

In this draft, we propose solutions for getting SIP through enterprise and residential NATs that does not require changes to these devices or to their configurations. NATs are a reality, and SIP

---

[1] The author of this draft is amongst those who have such a residential NAT, and thus feels highly motivated to solve this particular problem

deployment is being hampered by the lack of support for SIP ALGs in these boxes. A solution MUST be found, and we provide one here.

2 Some Philosophy

Our solution centers on the principle that applications, including components within network servers and end systems, need to take an active role in nat traversal.

This is counter to much of the existing work in nat traversal, which focuses on construction of ALGs embedded within NATs to make the existence of nats totally transparent to end systems and application layer network servers. The midcom efforts [3] have taken a step forward by recognizing that applications (either within end systems or network servers) are best suited to take a role in controlling NAT behavior. We believe that this approach needs to be taken one step farther, in that applications, especially those with components in end systems, need to adapt to the existence of non-midcom enabled NATs as well. In fact, we believe that the application of the end-to-end principle in this case argues in favor of our approach.

The end-to-end principle argues that:

> The function in question can completely and correctly be implemented only with the knowledge and help of the application standing at the end points of the communication system. Therefore, providing that questioned function as a feature of the communication system itself is not possible.

It is clear that the end-to-end principle would argue against the existence of NATs in the first place. However, there existence is a matter of reality. In order to properly engineer future protocols and applications, we are forced to take their existence as a given, and then investigate how our network design principles provide guidance on how to deal with them.

So, given that NATs exist, the end-to-end principle would tell us that only the applications can know what the impact of NAT will be on the functioning of the application. Since the end system is the one invoking the application, it is often best suited to determine how to deal with it. The overall system is much simpler and robust when the application in the end systems takes active participation in dealing with NAT.

Another way to view it is from the perspective of application adaptation. It has been a common design principle in real time applications for the end systems to adapt to the network conditions. Networks

might provide best effort, some level of QoS, or be overprovisioned for real time media. Rather than force the network to always deliver a specific level of quality, the applications detect the network conditions, and adapt to whatever they find. The result are robust applications and an overall simpler architecture.

We are arguing that this principle still makes sense when extended to other IP network "characteristics", including the presence of NAT. The existence of NAT, and the type of function it provides, are another axis in the overall space of IP network service. Applications will be the most robust and will perform best when they detect what level of network service (including QoS and NAT) is being provided, and then adapt to it in an optimal fashion. Just as QoS varies, so too do the types of NATs vary. By detecting what type of NAT is present, an end system can figure out how to achieve the best level of service given the existence of that NAT.

This approach means that applications can handle cases where there are ALGs (which still makes sense in many scenarios), application-unaware NATs, or what have you. When NATs disappear entirely, the applications will continue to function, and their performance will improve, in fact.

3 Overview of the Approach

Our approach consists of several pieces that are put together for a complete solution. The first is a set of SIP extensions that allow just SIP (but not neccesarily the sessions it establishes) to traverse NATs. Our extensions are relatively minor, backwards compatible, and allow NAT traversal for UDP and TCP transports. These extensions to SIP are described in Section 4.

Providing traversal for the media streams is more complex. The first step in the process is to allow end systems to detect whether there is a NAT between them and their SIP provider, and furthermore, to detect what type of treatment the NAT affords to UDP. We define a simple protocol which enables that to happen. Once the NAT type is detected, our protocol allows the end system to detect what its public facing address is on the other side of the NAT. We also discuss a router configuration which allows outside entities to send packets to this public address even under the strictest of NAT behaviors (which we call a symmetric NAT). These protocol mechanisms are discussed in Section 5.

Unfortunately, the mechanism of Section 5 requires an intermediate RTP relay (which is implemented using another NAT in our proposal) when the user is behind a symmetric NAT. To fix that problem, we define symmetric RTP, which is a new RTP usage scenario. It

effectively provides connection-oriented RTP over UDP. It is completely backwards compatible, and can avoid the need for an intermediary so long as one side in the call is not behind a symmetric NAT. Symmetric RTP, and the SDP extensions required to support it, are described in Section 6.

Finally, in Section 7, we put it all together, and show the various call flows that would exist for a variety of different configurations. The end result of our mechanisms are that end-to-end UDP media transport, directly between the two parties in a call leg, is always provided so long as it is provably possible. Only in the cases where it is provably impossible for direct media connectivity do we use an intermediary in the service provider domain.

The overall architecture we assume for the discussion is shown in Figure 1.

The caller is a UA in enterprise or residence A, and the called party is a UA in enterprise or residence B. The caller uses proxy X as its local outbound proxy, which forwards the call to the proxy of the called party, Y, also outside of the firewall or NAT. The call is then forwarded to the called party within enterprise or residence B.

4 SIP Extensions for NAT Traversal

This section discusses extensions to SIP that allow SIP itself to traverse NATs. There are two primary extensions - via ports and the contact cookie.

4.1 Via Ports

The first problem with SIP traversal through NATs is sending a request from a client behind a NAT to a server on the outside.

SIP specifies that for UDP, the response is sent to the port number in the Via header and the IP address the request came from. However, due to NAT, the port number in the Via header will be wrong. This means that the response will not be sent to the proper location. However, with TCP, responses are sent over the connection the INVITE arrived on. This means that a response sent over the TCP connection will be received properly by a caller behind a NAT. Therefore, one solution for traversal of requests from inside to outside is to use persistent TCP connections. However, many VoIP endpoints do not support TCP, so a UDP based solution is desirable.

Our approach is to define a new Via header parameter, called the response port, encoded as "rport". This parameter is inserted by

```
                       +-------+       +-------+
                       | SIP   |       | SIP   |
                       | Proxy |       | Proxy |
                       |  X    |       |   Y   |
                       |       |       |       |
                       +-------+       +-------+




          +-------+                           +-------+
  ........|FW/NAT |............       ........|FW/NAT |............
  .       |       |           .       .       |       |           .
  .       +-------+           .       .       +-------+           .
  .                           .       .                           .
  .                           .       .                           .
  .                           .       .                           .
  .                           .       .                           .
  .                           .       .                           .
  .                           .       .                           .
  .       +-------+           .       .       +-------+           .
  .       | SIP UA|           .       .       | SIP UA|           .
  .       |  Joe  |           .       .       |  Bob  |           .
  .       +-------+           .       .       +-------+           .
  .............................       .............................

       Enterprise or                       Enterprise or
         Residence A                        Residence B
```

Figure 1: Network Architecture

clients (which can be proxies or UACs) when they wish for the response to be sent to the IP address and port the request was sent from. The parameter is inserted with no value to flag this feature. When received at a server, the server inserts the port the request was received from as the value of this parameter. That port is used to forward the response.

```
response-port = ``rport'' [``='' 1*DIGIT]
```

A client inserting the rport into the Via header MUST wait for responses on the socket the request is sent on, and MUST also list, in the sent-by field, the local port of that socket the request was sent from. The latter is mandatory for backwards compatibility.

Consider an example. A client sends an INVITE which looks like:

```
INVITE sip:user@domain SIP/2.0
Via: SIP/2.0/UDP 10.1.1.1:4540;rport
```

This INVITE is sent with a source port of 4540 and source IP of 10.1.1.1. The request is natted, so that the source IP appears as 68.44.20.1 and the source port as 9988. This is received at a proxy. The proxy forwards the request, but not before appending a value to the rport parameter in the proxied request:

```
INVITE sip:user@domain2 SIP/2.0
Via: SIP/2.0/UDP proxy.domain.com
Via: SIP/2.0/UDP 10.1.1.1:4540;received=68.44.20.1;rport=9988
```

This request generates a response, which arrives at the proxy:

```
SIP/2.0 200 OK
Via: SIP/2.0/UDP proxy.domain.com
Via: SIP/2.0/UDP 10.1.1.1:4540;received=68.44.20.1;rport=9988
```

The proxy strips its top Via, and then examines the next one. It

contains both a received param, and an rport. The result is that the follow response is sent to IP address 68.44.20.1, port 9988:

```
SIP/2.0 200 OK
Via: SIP/2.0/UDP 10.1.1.1:4540;received=68.44.20.1;rport=9988
```

The NAT rewrites the destination address of this packet back to IP 10.1.1.1, port 4540, and is received by the client.

This works fine when the server supports this extension, so long as there are no nats between the client and server. Consider a server that does not understand it. In this case, it will ignore the rport parameter, and send the following response to IP 10.1.1.1, port 4540:

```
SIP/2.0 200 OK
Via: SIP/2.0/UDP 10.1.1.1:4540;rport
```

As specified by SIP, this response is sent to the source IP of the request, and the port in the Via header. Since the client is listening on 4540, the response is received correctly.

In the case where the server does not support the extension, but there is a nat between the client and the server, the response is sent to the source IP and port in the Via, which will be dropped by the nat. This is the same behavior exhibited by SIP today. As a result, our extension is backwards compatible, in the sense that it always works at least as well as baseline SIP. When both sides support it, and there is a nat in the middle, traversal works correctly.

For the response to always be received, the NAT binding must remain in existence for the duration of the transaction. Most UDP NAT bindings appear to have a timeout of one minute. Therefore, non-INVITE transactions will have no problem. For INVITE transactions, the client may need to retransmit its INVITE every 20 seconds or so, even after receiving a provisional response, in order to keep the binding open to receive the final response.

Because of the increased network traffic generated to keep the UDP bindings active, it is RECOMMENDED that TCP be used instead, as it generates much less data.

4.2 Contact Translation

The received port parameter will allow requests initiated from inside the NAT (and their responses), to work. However, getting requests from a proxy outside the NAT, to a host inside, is a different story.

The problem has to do with registrations. In Figure 1, the callee, Bob, will receive requests at their UA because they had previously sent a REGISTER request to their registrar, which is co-located with proxy Y. This registration contains a Contact header which lists the address where the incoming requests should be sent to. However, in the case of NAT, this address will be wrong. It will contain a domain name or IP address that is within the private space of enterprise B. Thus, the REGISTER might look like:

```
REGISTER sip:Y.com SIP/2.0
From: sip:bob@Y.com
To: sip:bob@Y.com
Contact: sip:bob@10.0.1.100
```

This address is not reachable by the proxy.

To solve this problem, we need two things. First, we need a persistent "connection" to be established from Bob to Y. Secondly, we need a way for incoming requests destined for B to be routed over this connection.

To address this first problem, clients have to send REGISTER reuests over a TCP or TLS connection, or use UDP along with the response port parameter in the Via header. If TCP is used, this connection is kept open indefinitely. We further recommend that the proxy/registrar hold this connection in a table, where the table is indexed by the remote side of the transport connection. For UDP, the client holds on to the socket, and uses it for REGISTER refreshes and to receive incoming calls. The server also holds on to the "connection". In the case of UDP, that means that server stores the local IP/port that the request was received on, and indexes it by the source IP and port the request was sent from. When the proxy wishes to send a packet to some server at IP address M, port N, transport O, it looks up the tuple (M,N,O) in the table to see if a connection already exists, and then uses it.

The NAT bindings are kept fresh through REGISTER refreshes (see Section 4.2.1).

Now, a connection is available for contacting the user. However, this connection must be associated with sip:bob@Y.com. Unfortunately, it is not. Calls for sip:bob@Y.com are translated to sip:bob@10.0.1.100,

which does not correspond to the remote side connection used to send the register, as seen by the proxy. Thats because of NAT, which will make the remote side appear to be a publically routable address.

To handle this problem, the proxy could, in principal, record the IP address and port from the remote side of the connection used to send a REGISTER. Then, it can create a Contact entry of the form sip:bob@[ip-addr]:[port], where [ip-addr] and [port] are the IP address and port of the remote side of the connection. However, this is assuming that the registration is for the purposes of connecting the address in the To field with the machine the connection is coming from. That may not be the intent of the registration. The registration may be used to set up a call forwarding service, for example.

As a result, it is our proposal that clients be allowed to explicitly ask a proxy to create a Contact entry corresponding to the machine a REGISTER is sent from. To do that, the UA inserts a Translate header into the request. This header contains the URL (which MUST be one of the Contact URLs) that is to be translated, along with a parameter that indicates the type of NAT the client is behind.

```
translate-header = ``Translate'' ``:'' SIP-URL [``;'' ``nat'' ``=''
nat-types]
nat-types = ``sym'' | ``cone''
```

If a server receives a REGISTER request with a translate header, it finds the matching Contact header, and replaces the host value with the source IP address of the REGISTER, and the port value with the source port of the REGISTER. This is the actual Contact stored in the registration database, and returned to the client in the response.

The nat-type parameter is an optional parameter that tells the registrar what type of NAT the client is behind. This information is very helpful for some faul tolerance and scalability scenarios, described below. Section 5 discusses how a client can determine what type of NAT it is behind.

Consider once more the architecture of Figure 1. The callee has an IP address of 10.0.1.100. It sends a REGISTER from port 2234 to port 5060 on the proxy. This connection goes through the NAT, and the source address is rewritten to 77.2.3.88, and the source port to 2937. The registration looks like:

```
REGISTER sip:Y.com SIP/2.0
```

```
From: sip:bob@Y.com
To: sip:bob@Y.com
Via: SIP/2.0/UDP 10.0.1.100;rport
Translate: sip:bob@10.0.1.100:2234
Contact: sip:bob@10.0.1.100:2234
```

The proxy Y then stores the socket the request was received on into a table, indexed by the source port:

```
(77.2.3.88,2397,UDP) -> [reference to UDP socket]
```

It also translates the Contact header to sip:bob@77.2.3.88:2397, and stores that in the registration database. It then responds to the REGISTER:

```
SIP/2.0 200 OK
From: sip:bob@Y.com
To: sip:bob@Y.com
Via: SIP/2.0/UDP 10.0.1.100;rport=2397;received=77.2.3.88
Contact: sip:bob@77.2.3.88:2397
```

This response is sent to 77.2.3.88:2397 because of the rport. The NAT translates this to 10.0.1.00:2234, which is then received by the client.

Now, when an INVITE arrives for sip:b@Y.com, it is looked up in the registration database. The contact is extracted, and the proxy tries to send the request to that address. To do so, it checks its connection table to an open connection to the IP address, port and transport where the request is destined. In this case, such a connection is available, and the request is forwarded over it. Because it is over a connection with an existing NAT binding, it is properly routed through the NAT. The response from the callee is also routed over the same connection.

In order for this connection to be used for re-INVITEs or BYEs, the proxy needs to record route.

4.2.1 Refresh Interval

Since the connection used for the registrations is held persistently in order to receive incoming calls, the NAT binding must be maintained. To avoid timeout, data must traverse the NAT over that connection with some minimum period. When UDP is used, registrations will need to be refreshed at least once every minute. The clients SHOULD include an Expires header or parameter with this value. For TCP, a longer interval can be used. 10 minutes is RECOMMENDED.

To test whether the interval is short enough, proxy servers MAY attempt to send OPTIONS requests to the client shortly before the registration expires. If the OPTIONS requests generates no response at all, the server SHOULD lower the value of the Expires header in the next registration. Servers SHOULD cache and reuse the largest successful refresh interval that they discover for a given Contact value.

4.2.2 Routing to the Ingress Proxy

A complication arises when a domain supports multiple proxy servers. Consider the scenario shown in Figure 2

A user joe in domain.com is behind a NAT. In DNS, domain.com contains an SRV entry that points to three servers, 1.domain.com, 2.domain.com and 3.domain.com. When the user registers, they will resolve domain.com to one of these. Assume its 1.domain.com. As a result of this, the connection state is stored proxy 1.

In the case of TCP, this connection state is important. Unless calls for joe@domain.com arrive to proxy 1, they won't be routable to the UA. In the case of UDP, whether it is important or not depends on the type of NAT the user is behind. One type of NAT, which we call "symmetric", treats UDP much like TCP. When A sends a request from inside to B on the outside, UDP messages back to A must come from B, with a source port equal to the destination port of messages from A to B. In the other case, which we call "cone", which is described in [4], UDP messages back to A can have any source port and IP address.

If the user is behind a NAT that operates in cone mode, any of the proxies in the proxy farm will be able to reach the customer through the NAT. All will send requests to the public IP address and port binding created by the NAT, but with different source IP addresses and ports. Since source addressing doesn't matter, things work well. In this case, the proxy need not even store connection state as described in Section 4.

If the user is behind a NAT that operates in symmetric mode, calls to the user must come in through the proxy that the user registered to.

```
                                            --
                                          //  \\
                                          /    \
                                         |  DB  |
                                         |      |
                                          \    /
                                          \\  //
                                            --


            +-----+   +-----+   +-----+
            |     |   |     |   |     |      domain.com
            |Proxy|   |Proxy|   |Proxy|
            |  1  |   |  2  |   |  3  |
            +-----+   +-----+   +-----+



            +-------------------------+
            |         NAT             |
            +-------------------------+



                      +-----+
                      |     |
                      |UA   |
                      |     |
                      +-----+
```

Figure 2: Multiple Proxy Configuration

In order to enable this, we recommend that the location server database store not only the contact, but the proxy that the user connected to. When a call comes in for that user, the proxy receiving the INVITE looks up the user in the database. The database entry indicates the proxy the user is connected to (call this the connected proxy). If the connected proxy is not the proxy which received the INVITE, the proxy that received the INVITE uses a route header to force the call through the connected proxy. In the case where joe registered at proxy1, and the incoming INVITE arrived at proxy 2, the request sent by proxy 2 would look like:

```
INVITE sip:proxy1.domain.com SIP/2.0
Route: sip:joe@22.1.20.3:3038
```

This request will first go to proxy1, and from there, over the existing connection to joe.

The differing proxy behaviors for symmetric and cone NATs explains the presence of the nat-type attribute in the Translate header. Assuming the client can determine which type it is behind (using the mechanisms described below), it can simply inform the proxy, allowing it to take the proper action.

4.2.3 INVITE Usage

The 200 OK response to the REGISTER request contains the SIP URL that the registrar placed into the database. This address has the important property that it is routable to the client from the proxy on the public side of the NAT. As a result, the client needs to place this URL as the Contact header in its INVITE requests and 2xx responses to INVITE, so that it can be reached from the proxy on the outside.

5 RTP/RTCP NAPT Identification and Traversal

In this section, we provide a protocol and basic architecture that allows a client to detect what type of NAT it is behind (cone or symmetric), and obtain the public address for an RTP stream.

The general idea is to make use of reflectors that return back to the client the source IP address and port that a request came from. The general configuration is showin Figure 3. In this figure, the hosts that wish to make or receive a call are behind enterprise or residential NATs. They are making use of a service provider that deploys, along with its proxies, three different reflectors, along with a few off-the-shelf routers configured in a specific fashion to act as a

media intermediary.

Reflector A is responsible for letting the user know whether they are behind a symmetric NAT, and for providing the address of another reflector (type C) which can be used to obtain an address binding on on a network intermediary.

Reflector B is used to let the user know whether they are behind a cone NAT (one which allows packets back to a natted host from any source port and IP, not just the one the outbound packet was sent to). It MUST be on a different IP address and port than reflector A. This is to deal with NATs which may allow packets back to an internal address from the same IP the packet was sent to, but different port. This kind of "partial-cone" NAT would be equivalent to a symmetrical one for the purposes of RTP.

Reflector C is used to allow the user to determine an address binding that is created on a NAT in the service provider domain. This NAT, and the routers around it, are configured so that the user can receive UDP packets through their enterprise NAT, even if its a symmetric NAT.

5.1 At initial power-up of Host A

When a client boots up, it first attempts to determine whether it is behind a NAT, and if so, what type. The following procedure is used:

1. Host A sends initial probe (probe type one) to Reflector A from its RTP and RTCP listener ports. Reflector A is the same IP address as the proxy server configured for this endpoint but an incremented port value (i.e. 5062). Reflector A could be the same physical device as the proxy server or on a seperate host by a static address translation.

2. Reflector A responds to Host A with an initial acknowledgement (probe response type one). This will create a symmetrical NAPT translation if the NAPT was initial a partial cone that migrates to symmetrical based on a response. Host A will re-transmit the probe packet every 50ms (until a timeout period of one minute) or until it receives this acknowledgement. The acknowledgement (probe response type one) will not contain the externally visible IP address of Host A; rather it will identify itself as the initial acknowledgement and contain a transaction timeout value. This value indicates the maximum time that Host A should wait for a message from Reflector B before determining it is behind a symmetrical NAPT. If Host A does not receive a

```
                                    +---------+
                                    |         |
                                    /Reflector|
                                   /|   B     |
                                   /+---------+
                                  /
                                  /
                                 /  +---------+
                                 /  |         |
                                /   /Reflector|
                                /  /|   A     |
                               /  / +---------+
+---------+   +---------+      / /
|         |   |Ent. NAPT|     / /
| Host A  -----Router A \     / /
|         |   |         |\   / /
+---------+   +---------+ \  //   +---------+
                           \//    |Service  |
                            /------Provider |
                            /     |Router A |
                           /      +----|----+
+---------+   +---------+ /            |
|         |   |Ent. NAPT|/             |
| Host B  -----Router B /              |
|         |   |         |              |
+---------+   +---------+              |
                                  +----|----+        +---------+
                                  |Ser. Prov|        |Reflector|
                                  |  NAPT   ----------    C    |
                                  | Router  |        |         |
                                  +---------+        +----|----+
                                                          |
                                                          |
                                                          |
                                                          |
                                                     +----|----+
                                                     |         |
                                                     |Registrar|
                                                     |         |
                                                     +---------+
```

   Figure 3: Configuration for NAPT Identification and Traversal

message from Reflector B within the specified timeframe, Host A will know that it is behind a symmetrical NAPT and send a subsequent message to Reflector A in which it asks for the address of Reflector C. By placing the request for the address of Reflector C after Host A has failed to hear from Reflector B, the provider can utilize deterministic load-balancing mechanisms for its Symmetrical Media Server. For this reason, Reflector A should be transaction state-ful. If a request for the address of Reflector C comes that does not match transaction information (i.e. source IP address) and is outside of the designated transaction timeout value plus one second, then Reflector A should respond with an error (i.e. 481). This will help limit attacks on Reflector A in which the attacker tries to throw off any load balancing mechanisms that the provider might be using when selecting the address for Reflector C to be used in the responding to hosts.

3. Reflector A instructs Reflector B to send a message (probe response type two) to Host A. This message will contain the externally visible address of Host A and the transaction timeout value that was sent to Host A.

4. Reflector B will send the message (probe response type two) to Host A and inform Reflector A that it has sent the mes-sage to Host A. Reflector A will continue to instruct Reflector B to send the message to Host A every 20ms or until it receives the acknowledgement from Reflector B that the message has been sent.

5. If Host A receives the message (probe response type two) from Reflector B it will know that it is behind a full-cone style NAPT. Host A will send an acknowledgement to Reflec-tor B. Reflector B will continue to retransmit the message to Host A every 50ms for up to the transaction timeout value specified by Reflector A or until it receives an ack-nowledgement from Host A.

6. If Host A does not get a probe response type two within the timeout value specified by Reflector A of sending its ini-tial probe packet, it will assume that it is behind a sym-metrical NAPT. If this occurs, Host A sends a message to Reflector A (Probe Type Three) informing it that it is behind a symmetrical NAPT. Reflector A will respond to this message with an acknowledgement that includes the IP address of Reflector C. Reflector A will retransmit this response every 50ms for up to 30 seconds or until it receives an acknowledgement from Host A.

A call flow for the case where Host A is behind a full-cone NAPT is show in Figure 4, and if Host A is behind a symmetrical NAPT, Figure 5.

```
    Host A            Reflector A       Reflector B
      |                    |                  |
      ---Probe Type One--->|                  |
      |                    |                  |
      |<-Probe Response-----                  |
      |    Type One        |                  |
      |                    -----Instruct----->|
      |                    |                  |
      |                    |<----Acknowledge---
      |                    |                  |
      |                                       |
      |<--------Probe Response Type Two--------
      |                                       |
      ---------------Acknowledge------------->|
      |                                       |
```

Figure 4: Full cone flow

5.2 When forming an Invite or 18n response

At some point later, host A either wishes to make a call, or wishes to answer an incoming call. In either case, if its behind a NAT, it needs to place an address and port in the SDP in the offer or answer which can be used to receive media. The approach that is used depends on what type of NAT the client determined it was behind.

If Host A determined it was behind a full-cone NAPT:

1. Host A sends a pre-Invite probe (probe type two) to Reflector A from its RTP and RTCP listener ports.

2. Reflector A responds to Host A with Host A's externally visible IP addresses. Host A then uses this address and port in the SDP header of the SIP message (note that this requires the SDP to carry RTCP address and port information).

3. If Host A does not receive a response from Reflector A, it will retransmit the pre- Invite probe every 50ms for up to

```
      Host A            Reflector A       Reflector B
        |                    |                  |
        +--Probe Type One--->|                  |
        |                    |                  |
        |<-Probe Response----+                  |
        |    Type One        |                  |
        |                    +----Instruct----->|
        |                    |                  |
        |                    |<----Acknowledge--+
        |                    |                  |
        |                                       |
        |   ....--Probe Response Type Two-------+
        |                    |                  |
        +-Probe Type Three-->|
        |                    |
        |<-Probe Response----+
        |  Type Three (with  |
        |  IP address of     |
        |  Reflector C)      |
        |                    |
        +---Acknowledge----->|
        |                    |
```

Figure 5: Symmetric flow

10 seconds. If Host A does not receive a response from Reflector A, it will inform the user that a network error has occurred an re-run the power-on test detailed above.

The message flow for RTP is as follows:

Step	Device	Addressing
1	RTP listener port on Host A	DA=192.1.3.2:6060, SA=X:Y
2	Enterprise NAPT router A	DA=193.1.3.2:6060, SA=X1:Y1
3	Router (receives packet and passes it to E0)	
4	Service Provider NAPT router, Ethernet port 0	DA=193.1.3.2:6060, SA=X2:Y2
5	Reflector	Response created with SA=X2:Y2
6	Service Provider NAPT router, Ethernet port 0	SA=193.1.3.2:6060, DA=X1:Y1
7	Enterprise NAPT router	SA=193.1.3.2:6060, DA=X:Y

8	RTP listener port on Host A	SA=193.1.3.2:6060, DA=X:Y with payload reflecting external SA of X2:Y2

The SIP endpoint now places X2:Y2 into its SDP header as its RTP and RTCP listener. In the above example, the address is 193.1.1.3 with a randomly selected port. This address and port actual exist on the RTP NAPT router and is addressable via the public internet.

Now Host B receives this information in an Invite or 180 message and sends RTP media to X2:Y2 (e.g. 193.1.1.3:32001). Since this is a public address, the packet is sent as follows:

Step	Device	Addressing
1	RTP sender on Host B	DA=X2:Y2, SA=A:Z
2	Enterprise NAPT router B	DA=X2:Y2, SA=A1:Z1
3	Router (receives packet and passes it to E1)	
4	Service Provider NAPT router, Ethernet port 1	DA=X2:Y2, SA=193.1.3.2:6060
5	Service Provider NAPT router, Ethernet port 0	DA=X1:Y1, SA=193.1.3.2:6060
6	Enterprise NAPT router	DA=X:Y, SA=193.1.3.2:6060
7	RTP listener on Host A (receives packet)	DA=X:Y, SA=193.1.3.2:6060

5.3 During the Call (Full Cone NAPT)

The media path, in this situation, is end-to-end via the enterprise NAPT routers. Media does not traverse the service provider's reflectors or symmetrical media servers. During the life of the call, Host A would need to send a periodic heartbeat (i.e. every 30 seconds) either to the reflector or Host B (the callee's endpoint) from the RTP listener port (RTCP packets should be sent regardless of media). The heartbeat ensures that a media path (i.e. NAPT translations) are not torn down due to prolonged silence

There is no need for endpoints behind Full Cone NAPTs to inform the reflectors about the termination of a call since the media does not affect the consumption of service provider resources.

5.4 If Host A determined that it was behind a symmetrical NAPT:

If the host is behind a symmetric enterprise NAT, things are more complex. With normal RTP, a network intermediary needs to be used. The user receives media packets from this intermediary, and the other party in the call sends packets to the intermediary.

1. Host A sends a pre-Invite probe (probe type two) to Reflector C from its RTP and RTCP listener ports.

2. Reflector C responds to Host A with an authentication challenge (i.e. 401 Not Authorized). It is suggested that digest authentication (rfc2069) be used and that the user information be based on their SIP profiles stored in a registrar. The nonce created by Reflector C could be comprised of an element of time (i.e. UMT), the externally visible IP address and port on which the pre-Invite probe appears to be sourced from when it reaches Reflector C, and a private key configured on Reflector C. Since Host A will have no knowledge of its externally visible address at this point, spoofing/replaying a response to this challenge becomes difficult.

3. Host A responds to the challenge by hashing the SIP userid and password based on the nonce provided by Reflector C.

4. Reflector C digests the results of this challenge and forwards a query of the user's information in the registrar. The connection between Reflector C and the registrar should be over a secure tunnel (i.e. TLS).

5. The registrar will keep track of the number of concurrent connections requested by Host A. This should be on the centralized registrar rather than the reflector in the event that multiple reflectors exist. If the registrar determines that Host A is at its pre-determined maximum number of concurrent sessions, the registrar will fail the query despite credentials matching and return an appropriate error to Reflector C. Reflector C will subsequently reply to Host A with a probe response challenge failure (max sessions).

6. If Host A is within the number of allowed concurrent sessions but does not provide correct credentials, the registrar will fail the query and return the appropriate message to Reflector C. Reflector C will subsequently reply to Host A with a probe response challenge failure (invalid user).

7. If successful, Reflector C returns a probe response type two to Host A which includes the externally visible IP address of Host A and a unique call id. There will be a

separate response for both RTP and RTCP and they will have unique call ids since the Reflector may not be able to match probe requests for RTP and RTCP. This call id is used later when informing the reflector that this call has been torn down.

8. Reflector C will inform the registrar that Host A now has an additional active connection (there will be two per call for each host: one for RTP and another for RTCP). The registrar will send an acknowledgement to Reflector C.

9. Host A sends an acknowledgement to Reflector C. Reflector C will re-transmit the probe response to Host A every 20ms until it receives an acknowledgement for up to 30 seconds. If Host A does not acknowledge the probe response type two, Reflector C will begin an independent call timer that sends a message to the registrar to remove one concurrent call for Host A after a pre-determined amount of time (i.e. 180 seconds). This timer is to ensure that endpoints cannot exploit the service providers NAPT router by intentionally failing to acknowledge the probe response (and therefore creating more concurrent calls than they are allotted) without penalizing the subscriber for a possible network failure.

A call flow for this case is shown in Figure 6.

5.5 During the Call (Symmetrical NAPT)

This section only applies when the endpoint is using the service provider's Symmetrical Media Server.

Host A now proceeds by sending its SIP message with an SDP header that includes the information obtained from the reflector. Note that the SDP must carry RTCP information. During the life of the call, Host A would need to send a periodic heartbeat (i.e. every 30 seconds) to the reflector for both RTP and RTCP. This heartbeat would include the call id. The heartbeat serves two purposes: it ensures that a media path (i.e. NAPT translations) are not torn down due to prolonged silence and that the concurrent session counter is eventually decremented in the event of an endpoint failure. In regards to decrementing the counter, Reflector C will keep a delta timer for each call id based on heartbeat. Should the delta time exceed a pre-configured value that is a multiplier of the heartbeat frequency but greater than the independent session timer (i.e. 210 seconds), Reflector C will believe that the call is no longer active and inform the registrar to decrement the counter. As noted above, optionally

```
        Host A          Reflector C          Registrar
          |                  |                    |
          ---Probe Type Two->|                    |
          |                  |                    |
          |<-Challenge (401)--                    |
          |                  |                    |
          -----Response----->|                    |
          |                  |                    |
          |                  -------Query-------->|
          |                  |                    |
          |                  |<-----Response-------
          |                  |                    |
          |<-Reply (Auth)-----                    |
          |                  |                    |
          |                  -----Inform--------->|
          |                  |                    |
          |                  |<---Acknowledge------
          ------Acknowledge->|                    |
          |                  |
```

Figure 6: Symmetrical NAT call flow

the reflector could instruct the service provider's NAPT router to remove the translation.

5.6 Call Teardown (Symmetrical NAPT)

This section only applies when the endpoint is using the service provider's Symmetrical Media Server.

1. At the end of the call (Bye, Cancel, or in response to a 400/500/600 SIP message), Host A sends a post-call closure messages (probe type four) to Reflector C with a matching call id from the earlier probe type two response from both its RTP and RTCP listener ports.

2. Reflector C responds to Host A with an authentication challenge (same mechanism is used as when setting up the call). This authentication is done in order to protect against service attacks (hackers sends closure messages for other systems).

3. Host A responds to the challenge.

4. Reflector C compares the results of this challenge to the user's information in the registrar.

5. If successful, Reflector C informs the registrar to remove one concurrent session from the counter. Optionally, Reflector C can instruct the service provider's NAPT router to remove the translation for this session. The registrar will acknowledge the decrementing of thecurrent session counter.

6. Reflector C sends an acknowledgement to Host A.

7. If unsuccessful, Reflector C replies to Host A with a probe response challenge failure (invalid user).

```
Host A          Reflector C          Registrar
  |                   |                    |
  +-Probe Type Four->|                    |
  |                   |                    |
  |<-Challenge (401)-+                    |
  |                   |                    |
  +----Response----->|                    |
  |                   |                    |
  |                   +------Query-------->|
  |                   |                    |
  |                   |<-----Response------+
  |                   |                    |
  |                   +----Instruct------->|
  |                   |                    |
  |                   |<---Acknowledge-----+
  |                   |                    |
  |<----Acknowledge--+                    |
  |                   |
```

Figure 7: Call Teardown

6 Symmetric RTP

The approach in section 5 requires the use of an intermediary when either of the parties is behind a symmetric NAT. This can be avoided so long as both of the parties are not behind symmetric NAT. The idea is to use symmetric RTP. Symmetric RTP is a new convention for RTP usage within SIP, and is described below.

The trick to getting RTP through a NAT is to make sure it exhibits two characteristics. First, any users behind a NAT have to send the first packet to establish a NAT binding. Secondly, media sent back to that user must be to the source port where the media came from. In other words, if Joe calls Bob, and only Joe is behind a NAT, Joe must send the first UDP packet to Bob. Let's say Joe sends from IP address and port pair A,B to Bob at public address and port C,D. The NAT will translate port pair A,B to X,Y. Bob receives the media. To talk to Joe, it is essential that Joe send his media with source port C,D to destination port X,Y. This will be received by the NAT, and have the destination translated to A,B, where it is sent to Joe.

Unfortunately, RTP does not work this way. When used with SIP, a conversation between Joe and Bob will result in two RTP sessions, one from Joe to the address Bob provided in his SDP, and one from Bob to the address provided by Joe in his SDP. This will not work with symmteric NAT without an intermediary.

6.1 Operation

Our solution is simple: we define symmetric RTP. Symmetric RTP runs over UDP. Like TCP, one side initiates a connection to the other side. As a result, one side is active (initiates the connection), and the other side is passive (waits for the connection). Like TCP, data in the reverse direction is sent to the port where the connection came from. Unlike TCP, a symmetric RTP connection is created when the first packet arrives; there is no explicit handshake or setup. There are no retransmissons or changes to the RTP protocol operation. The only difference is that symmetric RTP involves sending media on the same socket used to receive it.

An example flow using symmetric media is shown in Figure 8. Joe calls Bob. Assume for this flow that Joe is behind a NAT, and Bob is not. For simplicities sake, we don't show proxies, and don't show much of the SIP detail. Joe indicates, in his SDP in the INVITE, that he is capable of symmetric RTP, and wishes to be the active side of the connection (more on this later). Bob receives the INVITE, and responds with a 200 OK. His SDP indicates that he can be the passive side, and he provides the IP address and port to connect to. When Joe receives the 200 OK, an ACK is sent. Then, Joe sends a RTP packet to the IP address and port provided by Bob. The RTP packet passes through the NAT, and has its source address rewritten. When Bob receives this packet, the connection is established. Bob now has the IP address and port to send media back to. This address/port is the one from the source address of the RTP packet Bob just received (which has been natted). Bob sends media to this address. Those pack- ets have their destination address natted, translated back to the

address Joe used to send the first packet.

In traditional unidirectional RTP, Joe would have included an IP address and port in the INVITE, and Bob would have sent media to this address, rather than the one in the RTP packet received from Joe. This does not work through NAT, since this address is wrong, and since no NAT binding has been established. Symmetric RTP does not suffer this problem; note how Joe does not actually need to provide an IP address in the SDP in his INVITE (although must be provided for backwards compatibility).

The call flow when Bob is behind the NAT is very similar, and is shown in Figure 9. Instead of Joe being the active side of the connection, Bob is the active side. It is important to note that the role of active or passive for the RTP connection is not tied to who makes the call.

As a result, when only one the participants is behind a NAT, a direct UDP connection can be used between them. When both are behind NATs, a different solution is needed, and this is discussed below.

6.2 SDP Extensions

SDP extensions are needed to allow the signaling discussed above to take place. Specifically, extensions are needed to indicate that a media stream is symmetric RTP, and to allow each side to indicate that they are active, passive, or can play either role.

As it turns out, this is exactly the kind of signaling provided in the SDP extensions for TCP media [5]. That draft only handles TCP and TLS, but the semantics for TCP are identical to symmetric UDP. Therefore, the transport remains UDP, but the direction attribute and the exchange procedures defined in [5] for TCP works as described for UDP. The fact that the stream is symmetric is signaled by the presence of the active, passive, or both attributes.

Revisiting the flow in Figure 8, the SDP in the INVITE would actually appear as:

```
c=IN IP4 10.0.1.1
m=audio 9 RTP/UDP 0
a=direction:active
```

and in the 200 OK as:

```
     |                |                              |
     |                |                              |
     |---------------------------------------------> |
     |                | INV sip:bob@Y.com            |
     |                | active                       |
     |                |                              |
     |                |                              |
     |                |                              |
     |<--------------------------------------------- |
     |                | 200 OK                       |
     |                | passive                      |
     |                | 4.5.11.3:4444                |
     |                |                              |
     |                |                              |
     |---------------------------------------------> |
     |                | ACK                          |
     |                |                              |
     |                |                              |
     |                | RTP from Joe to Bob          |
     |----------------->--------------------------> |
     |S:10.0.1.1:12   |S:7.1.1.1:227                 |
     |D:4.5.11.3:4444 |D:4.5.11.3:4444               |
     |                |                              |
     |                | RTP from Bob to Joe          |
     |<--------------<-------------------------------|
     |S:4.5.11.3:4444 |             S:4.5.11.3:4444 |
     |D:10.0.1.1:12   |             D:7.1.1.1:227   |
     |                |                              |
     |                |                              |
     |                |                              |
     |                |                              |
     |                |                              |
     |                |                              |
     |                |                              |
     |                |                              |
     |                |                              |
     |                |                              |
     |                |                              |
     |                |                              |
     |                |                              |

    Joe              NAT                            Bob
```

Figure 8: Symmetric RTP Flow

```
     |                          |                    |
     |                          |                    |
     |---------------------------------------------> |
     |                          |  INV sip:bob@Y.com |
     |                          |  either            |
     |                          |  7.1.1.1:88        |
     |                          |                    |
     |                          |                    |
     |<--------------------------------------------- |
     |                          |             200 OK |
     |                          |             active |
     |                          |                    |
     |                          |                    |
     |                          |                    |
     |---------------------------------------------> |
     |                          |               ACK  |
     |                          |                    |
     |                          |                    |
     |                          |RTP from Bob to Joe |
     |<----------------<---------------------------< |
     |S:4.5.11.3:654            |    S:10.0.1.1:44   |
     |D:7.1.1.1:88              |    D:7.1.1.1:88    |
     |                          |                    |
     |                          |RTP from Bob to Joe |
     |>-------------->------------------------------>|
     |                          |                    |
     |S:7.1.1.1:88              |      S:7.1.1.1:88  |
     |D:4.5.11.3:654            |      D:10.0.1.1:44 |
     |                          |                    |
     |                          |                    |
     |                          |                    |
     |                          |                    |
     |                          |                    |
     |                          |                    |
     |                          |                    |
     |                          |                    |
     |                          |                    |
     |                          |                    |
     |                          |                    |
     |                          |                    |

   Joe                         NAT                  Bob
```

Figure 9: Symmetric RTP Flow, NAT role reversed

```
c=IN IP4 4.5.11.3
m=audio 4444 RTP/UDP 0
a=direction:passive
```

For reasons of backwards compatibility, a host that indicates active only in an INVITE must still list an IP address and port in the SDP, and be prepared to receive media on it. When the 200 OK comes, if it contains no direction attribute at all, the client knows that the server did not support this SDP extension. As a result, the server will ignore the direction attribute in the INVITE, and proceed to send media to the IP address and port in the INVITE.

The result is a very nice, smooth backwards compatibility from symmetric to traditional RTP usage.

6.2.1 RTCP Address and Port

Unfortunately, the NAT may not allocate consecutive port bindings to the RTP and RTCP packets. THis means that a client will need to signal in the SDP the IP address and port for both RTP and RTCP, separately. An approach for doing this is documented by Huitema

7 Using Symmetric RTP and NAT ID together

In this section, we show how a host would make use of both symmetric RTP and the NAT ID and binding protocol. There are many cases to consider. The caller and callee can either be behind a symmetric NAT, cone NAT, or no NAT. The caller and callee can either support or not support the symmetric RTP extension. The caller or callee can either support or not support the NAT ID proposal. While this may seem like a large number of cases (144 of them), the actual behavior at a host to handle all the cases is quite simple.

Why would a host ever support symmetric RTP, but not NAT ID? This is in cases where the host is some kind of service provider media-enabled device, such as a gateway or conferencing server. These networks are ideally deployed without NAT at all, or with a midcom-based firewall solution. As a result, NAT-ID is not needed, since the host knows it has a public address. Symmetric RTP is still helpful, to allow optimized access to the service from hosts behind a NAT. In considering the cases, though, this case is identical to the one where the host does support NAT ID, since NAT ID will always indicate that the host has a public address. The behavior of the host during

call setup is therefore identical to the case where NAT-ID wasn't there. This case aside, symmetric RTP does require the use of NAT ID to detect whether the host is behind a NAT or not.

We start with the caller. If the caller is an existing client that is unaware of symmetric RTP or the NAT ID protocol, it sends a regular INVITE. Of course, this will only work if the caller is not behind a NAT. If the caller supports NAT ID, it can detect if its behind a NAT. If so, before a call, it determines a public address using the NAT ID protocol, and uses this in the SDP. If it also supports symmetric RTP, and is behind a symmetric NAT, it indicates a direction of active for its media streams. If its behind a cone NAT, it indicates that it supports both active and passive.

It then sends the INVITE. It arrives at the called party. If the called party supports symmetric RTP, it checks whether the caller supported it (known based on the presence of the direction attribute in the SDP). If the caller supported it, and the called party is not behind a NAT, they insert their public address into the SDP in the response, and offer to be the passive side. Otherwise, if the called party is behind a NAT, they obtain an address using the NAT ID protocol, and insert that into the SDP in the response. The called party indicates passive if the caller indicated active, or they indicate active otherwise.

If the called party doesn't support symmetric RTP, it allocates an address binding (if it supports the NAT ID protocol), and places that in the SDP in the response. Since symmetric RTP is not supported, no direction attributes are indicated in the response. If the called party is ignorant of NAT ID, it simply places whatever it thinks is its address in the response.

The result of this fairly simple processing is that media flows directly whenever at all possible, using symmetric RTP whenever possible. Only in the most extreme case, where both caller and callee are behind symmetric NATs, does the service provider NAT get used. We also get smooth backwards compatibility, so that calls work as best they can if one side is ignorant of these extensions.

8 Security Considerations

The allocation of addresses on the service provider NAT consumes resources. Therefore, requests for those resources need to be authenticated, and coupled with the application layer service provided by the provider. This is why we specify the use of SIP authentication mechanisms for the reflector protocol.

Sample Router Configurations

The following are sample configuration files that can be used on a Cisco router in order to provide the NAT functions needed in Figure 3.

Service Provider Router A sample configuration:

```
        int s0
        ip address 63.1.1.1 255.255.255.252

        int e0
        ip address 193.1.2.2 255.255.255.0

        int e1
        ip address 193.1.1.2 255.255.255.0

        ip route 193.1.2.0 25.255.255.0  e0
        ip route 193.1.1.0 255.255.255.0 e1
        ip route 193.1.3.2 255.255.255.255 e0
        ip route 0.0.0.0 0.0.0.0 s0
```

Service Provider NAPT router sample configuration:

```
        int e0
        ip nat inside
        ip address 193.1.2.1 255.255.255.0

        int e1
        ip nat outside
        ip address 193.1.1.1 255.255.255.0

        int e3
        ip address 193.1.3.1 255.255.255.0

        ip nat pool rtp 193.1.1.3 193.1.1.3 prefix 24
        ip nat inside source list 9 pool rtp overload
        ip nat outside source static udp list 9 193.1.3.2 6060
        access-list 9 permit any any

        ip route 0.0.0.0 0.0.0.0 e0
```

A Author's Addresses

Jonathan Rosenberg
dynamicsoft
72 Eagle Rock Avenue

First Floor
East Hanover, NJ 07936
email: jdrosen@dynamicsoft.com

Joel Weinberger
dynamicsoft
72 Eagle Rock Avenue
First Floor
East Hanover, NJ 07936
email: jweinberger@dynamicsoft.com

Henning Schulzrinne
Columbia University
M/S 0401
1214 Amsterdam Ave.
New York, NY 10027-7003
email: schulzrinne@cs.columbia.edu

B Bibliography

[1] M. Holdrege and P. Srisuresh, "Protocol complications with the IP network address translator (NAT)," Internet Draft, Internet Engineering Task Force, Oct. 2000. Work in progress.

[2] J. Rosenberg, D. Drew, and H. Schulzrinne, "Getting SIP through firewalls and NATs," Internet Draft, Internet Engineering Task Force, Feb. 2000. Work in progress.

[3] P. Srisuresh, J. Kuthan, and J. Rosenberg, "Middlebox communication architecture and framework," Internet Draft, Internet Engineering Task Force, Feb. 2001. Work in progress.

[4] C. Huitema, "Short term NAT requirements for UDP based peer-to-peer applications," Internet Draft, Internet Engineering Task Force, Feb. 2001. Work in progress.

[5] D. Yon, "TCP-Based media transport in SDP," Internet Draft, Internet Engineering Task Force, Nov. 2000. Work in progress.