Internet Engineering Task Force   MIDCOM WG
Internet Draft

J. Rosenberg
dynamicsoft
J. Weinberger
dynamicsoft
C. Huitema
Microsoft
R. Mahy
Cisco

draft-rosenberg-midcom-stun-01.txt
March 1, 2002
Expires: September 2002

STUN - Simple Traversal of UDP Through NATs

STATUS OF THIS MEMO

Abstract

Simple Traversal of UDP Through NATs (STUN) is a lightweight protocol that allows applications to discover the presence and types of Network Address Translators (NATs) and firewalls between them and the public Internet. It also provides the ability for applications to determine the public IP addresses allocated to them by the NAT. STUN works with nearly all existing NATs, and does not require any special behavior from them. As a result, it allows a wide variety of applications to work through existing NAT infrastructure. The STUN

protocol is very simple, being almost identical to echo.

Table of Contents

1 Introduction

Network Address Translators (NATs), while providing many benefits, also come with many drawbacks. The most troublesome of those drawbacks is the fact that they break many existing IP applications, and make it difficult to deploy new ones. Guidlines have been developed [4] that describe how to build "NAT friendly" protocols, but many protocols simply cannot be constructed according to those guidelines. Examples of such protocols include almost all peer-to-peer protocols, such as multimedia communications, file sharing and games.

To combat that problem, Application Layer Gateways (ALGs) have been embedded in NATs. ALGs perform the application layer functions required for a particular protocol to traverse a NAT. Typically, this involves rewriting messages to contain translated addresses, rather than the ones inserted by the sender of the protocol message. ALGs have serious limitations, including scalability, reliability, and speed of deploying new applications. To resolve these problems, the Middlebox Communciations (MIDCOM) protocol is being developed [5]. MIDCOM allows an application entity, such as an end client or network server of some sort (like a SIP proxy [6]) to control a NAT (or firewall), in order to obtain NAT bindings and open or close pinholes. In this way, NATs and applications can be separated once more, eliminating the need for embedding ALGs in NATs, and resolving the limitations imposed by current architectures.

Unfortunately, MIDCOM requires upgrades to existing NAT and firewalls, in addition to application components. Complete upgrades of these NAT and firewall products will take a long time, potentially years. This is due, in part, to the fact that the deployers of NAT and firewalls are not the same people who are deploying and using applications. As a result, the incentive to upgrade these devices will be low in many cases. Consider, for example, an airport Internet lounge that provides access with a NAT. A user connecting to the natted network may wish to use a peer-to-peer service, but cannot, because the NAT doesn't support it. Since the administrators of the lounge are not the ones providing the service, they are not motivated to upgrade their NAT equipment to support it, using either an ALG, or MIDCOM.

Another problem is that the MIDCOM protocol requires that the agent controlling the middleboxes know the identity of those middleboxes, and have a relationship with them which permits control. In many configurations, this will not be possible. For example, many cable access providers use NAT in front of their entire access network. This NAT could be in addition to a residential NAT purchased and operated by the end user. The end user will probably not have a

control relationship with the NAT in the cable access network, and may not even know of its existence.

Many existing proprietary protocols, such as those for online games (such as the games described in RFC 3027 [7]) and Voice over IP, have developed tricks that allow them to operate through NATs without changing those NATs. This draft is an attempt to take some of those ideas, and codify them into an interoperable protocol that can meet the needs of many applications.

The protocol described here, Simple Traversal of UDP Through NAT (STUN), provides is an extremely simple protocol that allows entities behind a NAT to first discover the presence of a NAT and the type of NAT, and then to learn the addresses bindings allocated by the NAT. STUN requires no changes to NATs, and works with an arbitrary number of NATs in tandem between the application entity and the public Internet.

2 Terminology

In this document, the key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" are to be interpreted as described in RFC 2119 [1] and indicate requirement levels for compliant STUN implementations.

3 Definitions

STUN Client: A STUN client (also just referred to as a client) is an entity that generates STUN requests. A STUN client can execute on an end system, such as a users PC, or can run in a network element, such as a server.

STUN Server: A STUN Server (also just referred to as a server) is an entity that receives STUN requests, and sends STUN responses. STUN servers are generally attached to the public Internet. STUN servers are stateless.

4 NAT Variations

It is assumed that the reader is familiar with NATs. It has been observed that NAT treatment of UDP is variable amongst implementations. The four treatments observed in implementations are:

Full Cone: A full cone NAT is one where all requests from the same internal IP address and port are mapped to the same external IP address and port. Furthermore, any external host can send a packet to the internal host, by sending a packet to the mapped external address.

Restricted Cone: A restricted cone NAT is one where all requests from the same internal IP address and port are mapped to the same external IP address and port. Unlike a full cone NAT, an external host (with IP address X) can send a packet to the internal host only if the internal host had previously sent a packet to IP address X.

Port Restricted Cone: A port restricted cone NAT is like a restricted cone NAT, but the restriction includes port numbers. Specifically, an external host can send a packet, with source IP address X and source port P, to the internal host only if the internal host had previously sent a packet to IP address X and port P.

Symmetric: A symmetric NAT is one where all requests from the same internal IP address and port, to a specific destination IP address and port, are mapped to the same external IP address and port. If the same host sends a packet with the same source address and port, but to a different destination, a different mapping is used. Furthermore, only the external host that receives a packet can send a UDP packet back to the internal host.

Determining the type of NAT is important in many cases. Depending on what the application wants to do, the particular behavior may need to be taken into account.

5 Overview of Operation

This section is descriptive only. Normative behavior is described in Sections 7 and 8.

The typical STUN configuration is shown in Figure 1. A STUN client is connected to private network 1. This network connects to private network 2 through NAT 1. Private network 2 connects to the public Internet through NAT 2. On the public Internet is a STUN server.

STUN is a simple client-server protocol. Its operation is trivial. A client sends a request to a server. The server examines the source IP address and port of the request, and copies them into a response that is sent back to the client. There are some parameters in the request that allow the client to ask that the response be sent elsewhere, or that the server send the response from a different address and port. There are also security capabilities that allow the server to sign the response. Thats it.

The trick is using this simple protocol to discover the presence of

```
                           /-----\
                         // STUN  \\
                        |   Server  |
                         \\       //
                           \-----/



                      +--------------+             Public Internet
      ................|     NAT 2    |.......................
                      +--------------+



                      +--------------+             Private NET 2
      ................|     NAT 1    |.......................
                      +--------------+




                          /-----\
                        //  STUN \\
                       |    Client |
                        \\       //               Private NET 1
                          \-----/
```

Figure 1: STUN Configuration

nats, and to learn and use the bindings they allocate.

The STUN client is typically embedded in an application which needs to obtain a public IP address and port that can be used to receive data. For example, it might need to obtain an IP address and port to receive RTP [8] traffic. When the application starts, the STUN client within the application sends a STUN request to its STUN server. STUN servers are discovered through DNS SRV records [2], and is generally assumed that the client is configured with the domain to use to find the STUN server. Generally, this will be the domain of the provider of the service the application is using (such a provider is incented to deploy STUN servers in order to allow its customers to use its

application through NAT).

The STUN request is used to discover the presence of a NAT, and to discover the public IP address and port mappings generated by the NAT. Requests are sent to the STUN server using UDP. When a request arrives at the STUN server, it may have passed through one or more NATs between the STUN client and the STUN server. As a result, the source address of the request received by the server will be the mapped address created by the nat closest to the server. The STUN server copies that source IP address and port into a STUN response, and sends it back to the source IP address and port of the STUN request. For all of the NAT types above, this response will arrive at the STUN client.

When the STUN client receives the STUN response, it compares the IP address and port in the packet with the local IP address and port it bound to when the request was sent. If these do not match, the STUN client is behind one or more NATs. In the case of a full-cone NAT, the IP address and port in the body of the STUN response are public, and can be used by any host on the public Internet to send packets to the application that sent the STUN request. An application need only listen on the IP address and port from which the STUN request was sent, and send the IP address and port learned in the STUN response to hosts that wish to communicate with it.

Of course, the host may not be behind a full-cone NAT. Indeed, it doesn't yet know what type of NAT it is behind. To determine that, the client uses additional STUN requests. The exact procedure is flexible, but would generally work as follows. The client would send a second STUN request, this time to a different STUN server, but from the same source IP address and port. If the IP address and port in the response are different from those in the first response, the client knows it is behind a symmetric NAT. To determine if its behind a full-cone NAT, the client can send a STUN request with flags that tell the STUN server to send a response from a different IP address and port than the request was received on. In other words, if the client sent a request to IP address/port A/B using a source IP address/port of X/Y, the STUN server would send the response to X/Y using source IP address/port C/D. If the client receives this response, it knows it is behind a full cone NAT.

STUN also allows the client to ask the server to send the response from the same IP address the request was received on, but with a different port. This can be used to detect whether the client is behind a port restricted cone nat or just a restricted cone nat.

It should be noted that the configuration in Figure 1 is not the only permissible configuration. The STUN server can be located anywhere,

including within another client. The only requirement is that the STUN server is reachable at a public IP address.

6 Message Overview

STUN messages are TLV (type-length-value) encoded using big endian (network ordered) binary. All STUN messages start with a STUN header, followed by a series of STUN attributes. The STUN header contains a STUN message type, transaction ID, and length. The message type can be request or response. The transaction ID is used to correlate requests and responses. The length indicates the total length of the STUN message. This allows STUN to run over TCP. Usage over TCP is needed in order to fetch certificates from the server.

Several STUN attributes are defined. The first is a MAPPED-ADDRESS attribute, which is an IP address and port. It is always placed in the response, and it indicates the source IP address and port the server saw in the request. There is also a RESPONSE-ADDRESS attribute, which is also an IP address and port. The RESPONSE-ADDRESS attribute can be present in the request, and indicates where the response is to be sent. Its optional, and when not present, the response is sent to the source IP address and port of the request.

The third attribute is the FLAGS attribute, and it contains boolean flags to control behavior. Three flags are defined: "discard", "change IP" and "change port". The FLAG attribute is allowed only in the request. The discard attribute tells the server to not send a reply. The change IP and change port attributes are useful for determining whether the client is behind a restricted cone nat or restricted port cone nat. They instruct the server to send the responses from a different source IP address and port. The FLAGS attribute is optional in the request.

The fourth attribute is the CHANGED-ADDRESS attribute. It is present in all responses. It informs the client of the source IP address and port that would be used if the client requested the "change IP" and "change port" behavior.

The fifth attribute is the SOURCE-ADDRESS attribute. It is only present in responses. It indicates the source IP address and port where the response was sent from. It is useful for detecting twice NAT configurations.

The final two attributes provide security features. The SMS-SIGNED-DATA attribute allows the server to provide a signature over its response. This is useful for preventing several address-stealing attacks that would otherwise be possible. See Section 11 for details on the types of attacks possible. The COOKIE attribute enables a

four-way handshake between the client and the server for preventing distributed denial-of-service attacks. When the server receives a request without this cookie, or with an invalid one, it generates a response that contains the cookie attribute. The client then retries the request, including the cookie attribute. If the server receives a request with a valid cookie attribute, it continues to process the request.

7 Server Behavior

If the request contains the flag attribute, and the discard flag is true, the server MUST discard the request.

The server MUST generate a single response when a request is received (assuming the request is not discarded). The response MUST contain the same transaction ID contained in the request. The length in the message header MUST contain the total length of the message in bytes, excluding the header. The response MUST have a message type of "Response".

The server MUST add a MAPPED-ADDRESS attribute to the response. The IP address component of this attribute MUST be set to the source IP address observed in the request. The port component of this attribute MUST be set to the source port observed in the query request.

If the request arrived over TCP, the response MUST be sent on the same connection the request was received on. The server MAY close the connection after sending the response, but SHOULD instead wait for the client to close the connection.

If the request arrived over UDP, the procedures for sending the response are as follows.

If the RESPONSE-ADDRESS attribute was absent from the Query request, the destination address and port of the response MUST be the same as the source address and port of the request. Otherwise, the destination address and port of the response MUST be the value of the IP address and port in the RESPONSE-ADDRESS attribute.

The source address and port of the response are computed as follows. If the "change port" FLAG was set in the request, and the "change IP" flag was not set, the source port of the response MUST NOT be the same as the destination port of the query request, and the source address of the response MUST be the same as the destination address of the request. If the "change IP" FLAG was set in the request, and the "change port" flag was not set, the source IP address of the response MUST NOT be the same as the destination IP address of the query request, and the source port of the response MUST be the same

as the destination port of the request. When both flags are set, both the source port and source address MUST be different. Furthermore, the source port MUST be the same as if the the "change port" flag alone was present in the request, and the source address MUST be the same as if the "change IP" flag alone was present in the request. The result is that if the source port differs from the received port, it is the same independently of whether the source IP address is different from the received IP address. Exactly how this is implemented is a local decision.

The server MUST add a SOURCE-ADDRESS attribute to the response, containing the address and port used to send the response.

The server MUST add a CHANGED-ADDRESS attribute to the response. This contains the source IP address and port that would be used if the client requested the "change IP" and "change port" capabilities of the server. This address MUST be invariant across requests with the same source IP address and port for a duration of 10 minutes. In other words, if the client sends a request from a particular socket, and the response contains a specific CHANGED-ADDRESS, subsequent requests from the same socket should return the same CHANGED-ADDRESS.

One potential way to implement the change-IP feature is for the server to generate its own request, and send it to another server, running on a different host. That request is the same as the request received by the first server, except that a RESPONSE-ADDRESS attribute has been added, containing the source address and port of the original request. If the server receives a request with a RESPONSE-ADDRESS attribute, it must send the response to the address and port in that attribute. The second server will therefore send the response back to the original client. Since the response is sent by a different server, the IP address and port are different. This is shown in Figure 2.

The server can optionally sign the response, in order to provide additional security capabilities. The signature SHOULD NOT be present if the request did not contain a valid COOKIE attribute. Valid, in this case, means that the COOKIE was equal to a value previously handed out by the server to the same client. See subsequent paragraphs for more details. To sign the response, the server adds a CMS-SIGNED-DATA attribute as the last attribute in the response. If the response is sent over UDP (which it will, if the request came over UDP), the SignedData object MUST NOT contain any certificates. This is because the certificates would likely overflow the MTU, causing IP level fragmentation. This will not function in the presence of NAT. If the request arrived over TCP, the response MUST contain certificates. The server SHOULD sign the response using a site certificate whose domain matches the domain of the server, as

listed in DNS. However, an end user certificate MAY be used instead.

If there was no COOKIE in the request, or the COOKIE was invalid, the response MAY contain a COOKIE attribute. This attribute is a 128 bit opaque value, created by the server. It SHOULD be constructed in such a way so that it can be reconstructed by the server based on a subsequent request from the same client within a brief time interval. It is RECOMMENDED that it be computed as a cryptographic hash of the source IP address and port, in addition to a time stamp rounded to the nearest 5 seconds. When the client receives the response, if it wishes to authenticate the information it just received, it will create a new request, identical to the previous, except with a different transaction identifier, and this COOKIE reflected back. The server then recomputes the COOKIE, using the same algorithm it used to construct it previously, and verifies that this is the same value in the request. If the values are the same, the COOKIE in the request is valid. The purpose of this cookie exchange is to prevent distributed denial of service attacks that would force a server to perform expensive public key signatures. Instead, the server only provides the signature in responses that come from valid clients; valid, in this case, means the client meant to send the request, instead of being manipulated into doing so through a dDoS attack.

The server SHOULD NOT retransmit the response. Reliability is achieved by having the client periodically resend the request, each of which triggers a response from the server.

8 Client Behavior

The behavior of the client is very simple. Its main task is to discover the STUN server, formulate the request, handle request reliability, and authenticate the response.

8.1 Discovery

Generally, the client will be configured with a domain name of the provider of the STUN servers. This domain name is resolved to an IP address and port of using the SRV procedures specified in RFC 2782 [2].

Specifically, the service name is "stun". The protocol is "udp" or "tcp". The procedures of RFC 2782 are followed to determine the server to contact. RFC 2782 spells out the details of how a set of SRV records are sorted and then tried. However, it only states that the client should "try to connect to the (protocol, address, service)" without giving any details on what happens in the event of failure. Those details are described here for STUN.

```
                                           +---------+
                             +-+           |  Query  |
                             | |           |  Server |
                             | |   ------->|    1    |
                             | |---        +---------+
              Query          | |                |
              S:10.0.1.1  ---| | Query          |
                        --   | | S:14.1.2.2     |    Query
                     --- (1) |N|                |    RESPONSE-ADDRESS=
                  ---        | |                |(2)   14.1.2.2
       +------+ --           |A|                |
       |      |              | |                |
       |Client|              |T|                |
       |      |<---          | |                |
       +------+    ------    | |                \/
                         ----| |   (3)     +---------+
               Query         | |-----      |  Query  |
               Response      | |     ------|  Server |
               D: 10.0.1.1   | | Query     |    2    |
                             +-+ Response  +---------+
                                 D:14.1.2.2
```

Figure 2: Sending a response from a different address/port

For STUN requests, failure occurs if there is a transport failure of some sort (generally, due to fatal ICMP errors in UDP use or connection failures in TCP). Failure also occurs if the the request does not solicit a response after 30 seconds. If a failure occurs, the client SHOULD create a new request, which is identical to the previous, but has a different transaction ID. That request is sent to the next element in the list as specified by RFC 2782.

The default port for STUN requests is [to be assigned by IANA]. Administrators SHOULD use this port in their SRV records, but MAY use others.

If no SRV records were found, the client performs an A or AAAA record lookup of the domain name. The result will be a list of IP addresses, each of which can be contacted at the default port.

This would allow a firewall admin to open the STUN port, so hosts within the enterprise could access new applications. Whether they will or won't do this is a good question.

8.2 Formulating the Request

A request formulated by the client follows the syntax rules defined in Section 10. Any two requests that are not bit-wise identical, or not sent to the same server from the same IP address and port, MUST carry different transaction IDs. The transaction ID MUST be uniformly and randomly chosen between 0 and 2^^32 - 1. The message type of the request MUST be "Request".

The RESPONSE-ADDRESS attribute is optional in the request. It is used if the client wishes the response to be sent to a different IP address and port. This is useful for determining whether the client is behind a firewall, and for applications that have separated control and data components. See Section 9.3 for more details. The FLAGS attribute is also optional. Whether it is present depends on what the application is trying to accomplish. See Section 9 for some example uses.

Once formulated, the client sends the request. Reliability is accomplished through client retransmissions. Clients SHOULD retransmit the request starting with an interval of 100ms, doubling every retransmit until the interval reaches 1.6s. Retranmissions continue with intervals of 1.6s until a total of 9 requests have been sent, at which time the client SHOULD give up.

The response will contain the MAPPED-ADDRESS and SOURCE-ADDRESS attributes.

8.3 Authenticating the Response

As discussed in Section 11, there are serious security vulnerabilities introduced if the STUN response is not authenticated and integrity protected. To combat that problem, STUN provides for server signatures using CMS. The procedure for obtaining the signature and certificates to validate it are as follows.

The initial STUN reply will arrive with a COOKIE attribute. The client can choose to use the information in the STUN reply, validating it in parallel, or can choose to authenticate the information before using it. To authenticate the information, the client formulates a new STUN request, identical to the initial one except for two changes. First, is the usage of a new transaction ID. Second, is the addition of the COOKIE attribute, which is copied from the previous STUN response into the new request. The response to this

second STUN request will contain a CMS-SIGNED-DATA attribute. This attribute is equal to the SignedData object defined in RFC 2630 [3]. If the other attributes in the response match the first response, and the signature is valid, the client can trust that the information has not been tampered with, and is authentic. If the certificate used for the signature is a site certificate, the client SHOULD validate that the domain it used to perform the STUN query is a sub-domain of the domain in the site certificate. In other words, if the client queries stun.example.com, the client SHOULD validate that the signature was certified as coming from example.com.

However, the CMS-SIGNED-DATA attribute in the second response will not contain any certificates. The client may have cached certificates from a previous exchange with this server, in which case those certificates can be used. If the client does not have a certificate chain for the server, the client creates a third STUN request. This one is identical to the second, except it contains a new transaction ID, and is sent over TCP. The response will contain a CMS-SIGNED-DATA attribute, this time, with certificates.

9 Use Cases

The rules of Sections 7 and 8 describe exactly how a client and server interact to send requests and get responses. However, they do not dictate how the STUN protocol is used to accomplish useful tasks. That is at the discretion of the client. Here, we provide some useful scenarios for applying STUN.

9.1 Discovery Process

In this scenario, a user is running a multimedia application which needs to determine which of the following scenarios applies to it:

- On the open Internet

- Firewall that blocks UDP

- Firewall that allows UDP out, and responses have to come back to the source of the request (like a symmetric NAT, but no translation. We call this symmetric UDP Firewall)

- Full-cone NAT

- Symmetric NAT

- Restricted cone or restricted port cone NAT

Which of the six scenarios applies can be determined through the flow

chart described in Figure 3.

The flow makes use of three tests. In test I, the client sends a STUN request to a server, without any flags set, and without the RESPONSE-ADDRESS attribute. This causes the server to send the response back to the address and port that the request came from. This response provides the IP address and port for the third party address that would be used if the source IP and/or port were changed. In test II, the client sends a request with both the "change IP" and "change port" flags set. In test III, the client sends a request with only the "change port" flag set.

The client begins by initiating test I. If this test yields no response, the client knows right away that it is not capable of UDP connectivity. If the test produces a response, the client examines the MAPPED-ADDRESS attribute. If this address is the same as the local IP address and port of the socket used to send the request, the client knows that it is not natted. It executes test II. If a response is received, the client knows that it has open access to the Internet (or, at least, its behind a firewall that behaves like a full-cone NAT, but without the translation). If no response is received, the client knows its behind a symmetric UDP firewall.

In the event that the IP address and port of the socket did not match the MAPPED-ADDRESS attribute in the response to test I, the client knows that it is behind a NAT. It performs test II. If a response is received, the client knows that it is behind a full-cone NAT. If no response is received, it performs test I again, but this time, does so to the address from the CHANGED-ADDRESS attribute. If the IP address returned in the MAPPED-ADDRESS attribute is not the same as the one from the first test I, the client knows its behind a symmetric NAT. If the address is the same, the client is either behind a restricted or port restricted NAT. To make a determination about which one it is behind, the client initiates test III. If a response is received, its behind a restricted NAT, and if no response is received, its behind a port restricted NAT.

This simple procedure yields substantial information about the operating condition of the client application. In the event of multiple NATs between the client and the Internet, the type that is discovered will be the type of the most restrictive NAT between the client and the Internet. The types of NAT, in order of restrictiveness, from most to least, are symmetric, port restricted cone, restricted cone, and full cone.

9.2 Binding Lifetime Discovery

STUN can also be used to discover the lifetimes of the bindings

created by the NAT. To do that, the client first sends a simple request (no attributes) to server A. The response from A will contain the CHANGED-ADDRESS attribute. The client sends another simple request to that address (server B). It then starts a timer with a value of T seconds. When this timer fires, the client sends a request to server A, with the "change IP" and "change port" flags set. If the binding is still active, this response should be received through all nat types. The client can find the value of the binding lifetime by doing a binary search through T, arriving eventually at the value where the response is not received for any timer greater than T, but is received for any timer less than T.

9.3 Binding Acquisition

Consider once more the case of a VoIP phone. It used the discovery process above when it started up, to discover its environment. Now, it wants to make a call. As part of the discovery process, it determined that it was behind a full-cone NAT.

Consider further that this phone consists of two logically separated components - a control component that handles signaling, and a media component that handles the audio, video, and RTP [8]. Both are behind the same NAT. Because of this separation of control and media, we wish to minimize the communication required between them. In fact, they may not even run on the same host.

In order to make a voice call, the phone needs to obtain an IP address and port that it can place in the call setup message as the destination for receiving audio.

To obtain an address, the control component first sends a STUN request to a server. No flags are present, and neither is the RESPONSE-ADDRESS field. The response contains a mapped address. The control component then formulates a second request. This request contains a RESPONSE-ADDRESS, which is set to that mapped address. This request is passed to the media component, along with the IP address and port of the STUN server. The media component sends the request. The request goes to the STUN server, which sends the response back to the control component. The control component receives this, and now has learned an IP address and port that will be routed back to the media component that sent the request.

The client will be able to receive media from anywhere on this mapped address.

In the case of silence suppression, there may be periods where the client receives no media. In this case, the UDP bindings could timeout (UDP bindings in nats are typically short). To deal with

this, the application can periodically retransmit the query in order to keep the binding fresh.

It is possible that both participants in the multimedia session are behind the same NAT. In that case, both will repeat this procedure above, and both will obtain public address bindings. When one sends media to the other, the media is routed to the nat, and then turns right back around to come back into the enterprise, where it is translated to the private address of the recipient. This is not particularly efficient, but it does work.

10 Protocol Details

This section presents the detailed encoding of a STUN message.

10.1 Message Header

All STUN messages consist of a 64 bit header:

```
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|      STUN Message Type        |         Message Length        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                      Transaction ID                           |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

The Message Types can take on the following values:

```
0x0001  :  Request
0x0101  :  Response
```

The message length is the count, in bytes, of the size of the message, not including the 64 bit header.

The transaction ID is a 32 bit identifier. All responses carry the same identifier as the request they correspond to.

10.2 Message Attributes

After the header are 0 or more attributes. Each attribute is TLV encoded, with a 16 bit type, 16 bit length, and variable value:

```
    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
    |         Type                  |            Length             |
    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
    |                             Value                             ....
    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

The following types are defined:

```
0x0001: MAPPED-ADDRESS
0x0002: RESPONSE-ADDRESS
0x0003: FLAGS
0x0004: SOURCE-ADDRESS
0x0005: CHANGED-ADDRESS
0x0006: CMS-SIGNED-DATA
0x0007: COOKIE
```

Future extensions MAY define new attributes. If a stun client or server receives a message with an unknown attribute with a type lower than or equal to 0x7fff, the message MUST be discarded. If the type is greater than 0x7fff, the attribute MUST be ignored. The ordering of attributes within a message is not important, and a client or server MUST be prepared to receive them in any order. Any attributes that are known, but are not supposed to be present in a message (MAPPED-ADDRESS in a request, for example) MUST be ignored.

The length refers to the length of the value element.

10.2.1 MAPPED-ADDRESS

The MAPPED-ADDRESS attribute indicates the mapped IP address and port. It consists of a sixteen bit port, eight bit address family, followed by a fixed length value representing the IP address.

```
    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
    |           Port                |    Family     |x x x x x x x x|
    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
    |                             Address..
    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

The port is a network byte ordered representation of the mapped port.

The following families are defined:

```
0x01:   IPv4
0x02:   IPv6
```

The family is followed by 8 bits which are ignored, for the purposes of aligning the address on 32 bit boundaries.

For IPv4 addresses, the address is 32 bits. For IPV6, it is 128 bits.

New address families MAY be defined by extensions. A message with an unknown address family is discarded.

10.2.2 RESPONSE-ADDRESS

The RESPONSE-ADDRESS attribute indicates where the response to a request is sent. Its syntax is identical to MAPPED-ADDRESS.

10.2.3 CHANGED-ADDRESS

The CHANGED-ADDRESS attribute indicates the IP address and port of a STUN server where responses will be sent from if the "change IP" and/or "change port" flags are set. Its syntax is identical to MAPPED-ADDRESS.

10.2.4 FLAGS

The FLAGS attribute is a series of boolean flags. It is 32 bits long:

```
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|A|B|C|                                                         |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Only three flags, A,B,C, are currently defined. The other bits MAY be used by extensions to define additional flags. Unknown flags are ignored.

Each flag is a binary one if true, zero otherwise.

The meaning of the flags is:

A: This is the "change IP" flag. If true, it requests the server

to send the response with a different IP address than the one the request was received on.

B: This is the "change port" flag. If true, it requests the server to send the response with a different port than the one the request was received on.

C: This is the discard flag. If true, the message is discarded.

10.2.5 SOURCE-ADDRESS

The SOURCE-ADDRESS attribute is present in responses. It indicates the source IP address and port that the server is sending the response from. Its syntax is identical to that of MAPPED-ADDRESS.

10.2.6 CMS-SIGNED-DATA

STUN responses can be signed. The signatures are conveyed using the Cryptographic Message Syntax (CMS), RFC 2630 [3]. Specifically. the CMS-SIGNED-DATA is exactly equal to the CMS SignedData object, using detached signatures. When present in a response, the CMS-SIGNED-DATA attribute MUST be the last attribute in the response. The contentData covered by the signature includes all of the bytes from the start of the STUN message, up to, but not including, the CMS-SIGNED-DATA attribute.

When the response is sent over UDP, the response won't contain certificates (see Section 7. Responses over TCP will contain certificates.

10.2.7 COOKIE

```
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                   Cookie word 1                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                   Cookie word 2                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                   Cookie word 3                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                   Cookie word 4                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

The COOKIE attribute is used to prevent distributed denial of service attacks on the STUN server. It is a 128 bit value, generated by the server, and echoed by the client back to the server.

11 Security Considerations

There are several potential threats in a STUN system worthy of consideration.

STUN can potentially introduce attacks which result in the theft of addresses. When a client sends a request, an attacker can guess the value of the mapped address used by the nat, and quickly generate its own faked response, sending it to that address. This response would contain a faked MAPPED-ADDRESS which actually routes to a different host. This could enable DoS attacks, by using a victim's address, or theft attacks, by using the address of the host run by the attacker. This threat is potentially very serious. To combat it, STUN supports public key authentication of responses. This allows a client to verify that the response was indeed generated by the server to which the request was sent.

Interestingly, there is little need for strong authentication of requests. STUN servers are stateless. Their processing is not user specific. The server is hardly more than an echo server. As a result, client authentication provides no value. However, a request from a client can result in expensive public key operations at the server, for the purposes of signing the request. This makes a STUN server potentially subject to denial-of-service attacks. To prevent against such threats, STUN provides a simple cookie mechanism. The server will not sign any responses until it has successfully passed a cookie to the client and received it back in a subsequent request. This cookie mechanism is similar to the techniques used to prevent the TCP SYN attack.

Compromise of a STUN server can lead to discovery of open ports. Knowledge of an open port creates an opportunity for DoS attacks on those ports (or DDoS attacks if the traversed NAT is a full cone NAT). Discovering open ports is already fairly trivial using port probing, so this does not represent a major threat.

STUN servers constitute a reflector type of server, and can therefore be used as launching grounds for distributed DoS attacks [9]. The problem is amplified by the existence of the RESPONSE-ADDRESS attribute, which can render ingress filtering useless in prevention of attacks. Interestingly, the MAPPED-ADDRESS in the response provides a form of traceback in order to counter such attacks. An attacker would need to spoof their source address in order to avoid the traceback mechanism.

STUN has the important property that compromise of the STUN servers cannot cause security breaches of a firewall when the client is within an enterprise. The only thing that a compromised server can do

is return false addresses, resulting in the inability of the client to receive any data at all. However, the attacker cannot send packets to arbitrary servers within the enterprise, if the firewall prohibits such communication.

12 IANA Considerations

There are no IANA considerations associated with this specification.

13 IAB Considerations

The IAB has studied the problem of "Unilateral Self Address Fixing", which is the general process by which a client attempts to determine its address in another realm on the other side of a NAT through a collaborative protocol reflection mechanism [10]. STUN is an example of a protocol that performs this type of function. The IAB has mandated that any protocols developed for this purpose document a specific set of considerations. This section meets those requirements.

13.1 Problem Definition

From [10], any UNSAF proposal must provide:

> Precise definition of a specific, limited-scope problem that is to be solved with the UNSAF proposal. A short term fix should not be generalized to solve other problems; this is why "short term fixes usually aren't".

The specific problems being solved by STUN are:

- o Provide a means for a client to detect the presence of one or more NATs between it and a server run by a service provider on the public Internet. The purpose of such detection is to determine additional steps that might be necessary in order to receive service from that particular provider.

- o Provide a means for a client to obtain an address on the public Internet from a non-symmetric NAT, for the express purpose of receiving incoming UDP traffic from another host targeted to that address.

STUN does not address TCP, either incoming or outgoing, and does not address outgoing UDP communications.

13.2 Exit Strategy

From [10], any UNSAF proposal must provide:

> Description of an exit strategy/transition plan. The better short term fixes are the ones that will naturally see less and less use as the appropriate technology is deployed.

STUN comes with its own built in exit strategy. This strategy is the detection operation that is performed as a precursor to the actual UNSAF address-fixing operation. This discovery operation, documented in Section 9.1, attempts to discover the existence of, and type of, any NATS between the client and the service provider network. Whilst the detection of the specific type of NAT may be brittle, the discovery of the existence of NAT is itself quite robust. As NATs are phased out through the deployment of IPv6, the discovery operation will return immediately with the result that there is no NAT, and no further operations are required. Indeed, the discovery operation itself can be used to help motivate deployment of IPv6; if a user detects a NAT between themselves and the public Internet, they can call up their access provider and complain about it.

STUN can also help facilitate the introduction of midcom. As midcom-capable NATs are deployed, applications will, instead of using STUN (which also resides at the application layer), first allocate an address binding using midcom. However, it is a well-known limitation of midcom that it only works when the agent knows the middleboxes through which its traffic will flow. Once bindings have been allocated from those middleboxes, a STUN detection procedure can validate that there are no additional middleboxes on the path from the public Internet to the client. If this is the case, the application can continue operation using the address bindings allocated from midcom. If it is not the case, STUN provides a mechanism for self-address fixing through the remaining midcom-unaware middlboxes. Thus, STUN provides a way to help transition to full midcom-aware networks.

13.3 Brittleness Introduced by STUN

From [10], any UNSAF proposal must provide:

> Discussion of specific issues that may render systems more "brittle". For example, approaches that involve using data at multiple network layers create more dependencies, increase debugging challenges, and make it harder to transition.

STUN introduces brittleness into the system in several ways:

- o The discovery process assumes a certain classification of devices based on their treatment of UDP. Their could be other types of NATs that are deployed that would not fit into one of these molds. Therefore, future NATs may not be properly detected by STUN. STUN clients (but not servers) would need to change to accomodate that.

- o The binding acquisition usage of STUN does not work for all NAT types. It will work for any application for full cone NATs only. For restricted cone and port restricted cone NAT, it will work for some applications dependening on the application. Application specific processing will generally be needed. For symmetric NATs, the binding acquisition will not yield a usable address. The tight dependency on the specific type of NAT makes the protocol brittle.

- o STUN assumes that the server exists on the public Internet. If the server is located in another private address realm, the user may or may not be able to use its discovered address to communicate with other users. There is no way to detect such a condition.

- o The bindings allocated from the NAT need to be continuously refreshed. Since the timeouts for these bindings is very implementation specific, the refresh interval cannot easily be determined. When the binding is not being actively used to receive traffic, but rather just wait for it, the binding refresh will needlessly consume network bandwidth.

- o The use of the STUN server as an additional network element introduces another point of potential security attack. These attacks are largely prevented by the security measures provided by STUN, but not entirely.

- o The use of STUN to discover address bindings will result in an increase in latency for applications. For example, a Voice over IP application will see an increase of call setup delays equal to at least one RTT to the stun server.

13.4 Requirements for a Long Term Solution

From [10], any UNSAF proposal must provide:

Identify requirements for longer term, sound technical solutions -- contribute to the process of finding the right longer term solution.

Our experience with STUN has led to the following requirements for a long term solution to the NAT problem:

Requests for bindings and control of other resources in a NAT need to be explicit. Much of the brittleness in STUN derives from its guessing at the parameters of the NAT, rather than telling the NAT what parameters to use.

Control needs to be "in-band". There are far too many scenarios in which the client will not know about the location of middleboxes ahead of time. Instead, control of such boxes needs to occur in band, traveling along the same path as the data will itself travel. This guarantees that the right set of middleboxes are controlled. This is only true for first-party controls; third-party controls are best handled using the midcom framework.

Control needs to be limited. Users will need to communicate through NATs which are outside of their administrative control. In order for providers to be willing to deploy NATs which can be controlled by users in different domains, the scope of such controls needs to be extremely limited - typically, allocating a binding to reach the address where the control packets are coming from.

Simplicity is Paramount. The control protocol will need to be implement in very simple clients. The servers will need to support extremely high loads. The protocol will need to be extremely robust, being the precursor to a host of application protocols. As such, simplicity is key.

13.5 Issues with Existing NAPT Boxes

From [10], any UNSAF proposal must provide:

Discussion of the impact of the noted practical issues with existing, deployed NA[P]Ts and experience reports.

Several of the practical issues with STUN involve future proofing - breaking the protocol when new NAT types get deployed. Fortunately, this is not an issue at the current time, since most of the deployed NATs are of the types assumed by STUN. The primary usage STUN has found is in the area of VoIP, to facilitate allocation of addresses for receiving RTP [8] traffic. In that application, the periodic keepalives are provided by the RTP traffic itself. However, several practical problems arise for RTP. First, RTP assumes that RTCP traffic is on a port one higher than the RTP traffic. This pairing

property cannot be guaranteed through NATs that are not directly controllable. As a result, RTCP traffic may not be properly received. Protocol extensions to SDP have been proposed which mitigate this by allowing the client to signal a different port for RTCP [11]. However, there will be interoperability problems for some time.

For VoIP, silence suppression can cause a gap in the transmission of RTP packets. This could result in the loss of a binding in the middle of a call, if that silence period exceeds the binding timeout. This can be mitigated by sending occassional silence packets to keep the binding alive. However, the result is additional brittleness; proper operation depends on the the silence suppression algorithm in use, the usage of a comfort noise codec, the duration of the silence period, and the binding lifetime in the NAT.

13.6 In Closing

The problems with STUN are not design flaws in STUN. The problems in STUN have to do with the lack of standardized behaviors and controls in NATs. The result of this lack of standardization has been a proliferation of devices whose behavior is highly predictable, extremely variable, and uncontrollable. STUN does the best it can in such a hostile environment. Ultimately, the solution is to make the environment less hostile, and to introduce controls and standardized behaviors into NAT. However, until such time as that happens, STUN provides a good short term solution given the terrible conditions under which it is forced to operate.

14 Acknowledgements

The authors would like to thank Cullen Jennings for his comments, and Baruch Sterman and Alan Hawrylyshen for initial implementations.

15 Authors 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

Christian Huitema
Microsoft Corporation
One Microsoft Way
Redmond, WA 98052-6399
email: huitema@microsoft.com

Rohan Mahy
Cisco Systems
170 West Tasman Dr, MS: SJC-21/3
Phone: +1 408 526 8570
Email: rohan@cisco.com

16 Normative References

[1] S. Bradner, "Key words for use in RFCs to indicate requirement levels," Request for Comments 2119, Internet Engineering Task Force, Mar. 1997.

[2] A. Gulbrandsen, P. Vixie, and L. Esibov, "A DNS RR for specifying the location of services (DNS SRV)," Request for Comments 2782, Internet Engineering Task Force, Feb. 2000.

[3] R. Housley, "Cryptographic message syntax," Request for Comments 2630, Internet Engineering Task Force, June 1999.

17 Informative References

[4] D. Senie, "Network address translator (nat)-friendly application design guidelines," Request for Comments 3235, Internet Engineering Task Force, Jan. 2002.

[5] P. Srisuresh, J. Kuthan, J. Rosenberg, A. Molitor, and A. Rayhan, "Middlebox communication architecture and framework," Internet Draft, Internet Engineering Task Force, Nov. 2001. Work in progress.

[6] M. Handley, H. Schulzrinne, E. Schooler, and J. Rosenberg, "SIP: session initiation protocol," Request for Comments 2543, Internet Engineering Task Force, Mar. 1999.

[7] M. Holdrege and P. Srisuresh, "Protocol complications with the IP network address translator," Request for Comments 3027, Internet Engineering Task Force, Jan. 2001.

[8] H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson, "RTP: a transport protocol for real-time applications," Request for Comments 1889, Internet Engineering Task Force, Jan. 1996.

[9] V. Paxson, "An analysis of using reflectors for distributed denial of service attacks," ACM Computer Communication Review , Vol. 31, July 2001.

[10] L. Daigle, "IAB considerations for UNilateral self-address fixing (UNSAF)," Internet Draft, Internet Engineering Task Force, Feb. 2002. Work in progress.

[11] C. Huitema, "RTCP attribute in SDP," Internet Draft, Internet Engineering Task Force, Feb. 2002. Work in progress.

Full Copyright Statement