                STUN - Simple Traversal of User Datagram Protocol (UDP)
                      Through Network Address Translators (NATs)

Status of this Memo

Copyright Notice

Abstract

   Simple Traversal of User Datagram Protocol (UDP) Through Network
   Address Translators (NATs) (STUN) is a lightweight protocol that
   allows applications to discover the presence and types of NATs and
   firewalls between them and the public Internet. It also provides the
   ability for applications to determine the public Internet Protocol
   (IP) addresses allocated to them by the NAT. STUN works with many
   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.

Table of Contents

1. Applicability Statement

   This protocol is not a cure-all for the problems associated with NAT.
   It does not enable incoming TCP connections through NAT. It allows
   incoming UDP packets through NAT, but only through a subset of
   existing NAT types. In particular, STUN does not enable incoming UDP
   packets through symmetric NATs (defined below), which are common in
   large enterprises. STUN's discovery procedures are based on
   assumptions on NAT treatment of UDP; such assumptions may prove
   invalid down the road as new NAT devices are deployed. STUN does not
   work when it is used to obtain an address to communicate with a peer
   which happens to be behind the same NAT. STUN does not work when the
   STUN server is not in a common shared address realm. For a more
   complete discussion of the limitations of STUN, see Section 14.

2. 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. Guidelines have been
   developed [8] 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 this 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 application layer messages to contain
   translated addresses, rather than the ones inserted by the sender of
   the message. ALGs have serious limitations, including scalability,
   reliability, and speed of deploying new applications. To resolve
   these problems, the Middlebox Communications (MIDCOM) protocol is
   being developed [9]. MIDCOM allows an application entity, such as an
   end client or network server of some sort (like a Session Initiation
   Protocol (SIP) proxy [10]) 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 NATed 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 [11]) and Voice over IP,
have developed tricks that allow them to operate through NATs without
changing those NATs. This document 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), 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.

3. 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 BCP 14, RFC 2119
[1] and indicate requirement levels for compliant STUN
implementations.

4. 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 user's PC, or can run in a
      network element, such as a conferencing 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.

5. NAT Variations

   It is assumed that the reader is familiar with NATs. It has been
   observed that NAT treatment of UDP varies among 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, it may need to take the particular
behavior into account.

6. Overview of Operation

This section is descriptive only. Normative behavior is described in
Sections 8 and 9.

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


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


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


                            /-----\
                          // STUN  \\
                         |   Client  |
                          \\        //         Private NET 1
                            \-----/
```
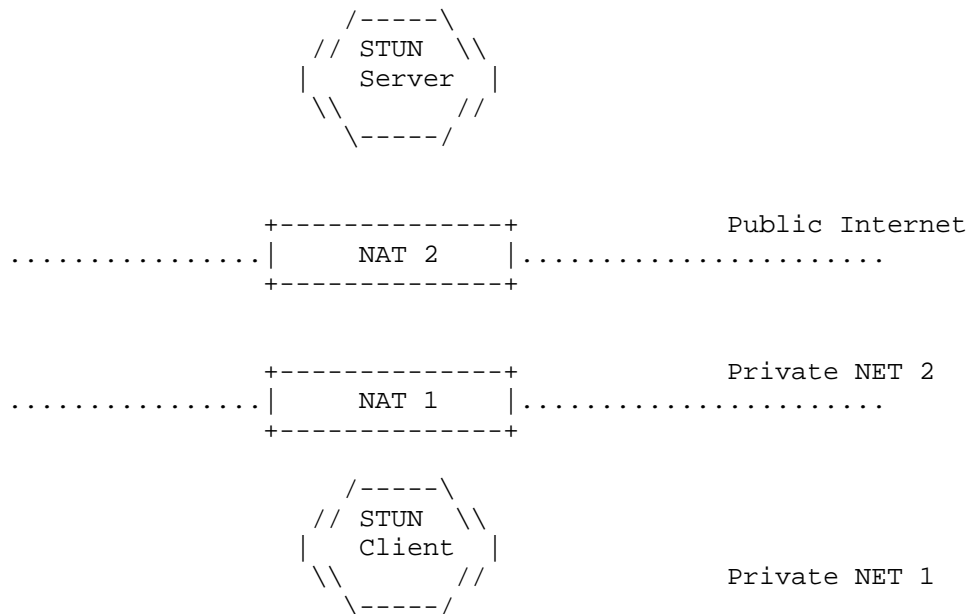
Figure 1: STUN Configuration

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. The STUN server resides on the public
Internet.

STUN is a simple client-server protocol. A client sends a request to
a server, and the server returns a response. There are two types of
requests - Binding Requests, sent over UDP, and Shared Secret
Requests, sent over TLS [2] over TCP. Shared Secret Requests ask the
server to return a temporary username and password. This username
and password are used in a subsequent Binding Request and Binding
Response, for the purposes of authentication and message integrity.

Binding requests are used to determine the bindings allocated by
NATs. The client sends a Binding Request to the server, over UDP.
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 attributes for
providing message integrity and authentication.

The trick is using STUN to discover the presence of NAT, 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 Real Time Transport Protocol (RTP) [12] traffic. When the
application starts, the STUN client within the application sends a
STUN Shared Secret Request to its server, obtains a username and
password, and then sends it a Binding Request. STUN servers can be
discovered through DNS SRV records [3], and it 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). Of course, a client can determine the
address or domain name of a STUN server through other means. A STUN
server can even be embedded within an end system.

The STUN Binding Request is used to discover the presence of a NAT,
and to discover the public IP address and port mappings generated by
the NAT. Binding Requests are sent to the STUN server using UDP.
When a Binding 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
Binding 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 Binding 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. Any packets sent by a host on the public
Internet to the public address and port learned by STUN will be
received by the application.

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 Binding Requests. The exact
procedure is flexible, but would generally work as follows. The
client would send a second STUN Binding Request, this time to a
different IP address, 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 it's behind a full-cone NAT, the client can send a
STUN Binding 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 Binding Request to
IP address/port A/B using a source IP address/port of X/Y, the STUN
server would send the Binding 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 Binding
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 by the client, and if the client is trying
to obtain a publicly routable address, that the server reside on the
public Internet.

7. 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 STUN payload. The payload is a series of STUN
   attributes, the set of which depends on the message type. The STUN
   header contains a STUN message type, transaction ID, and length. The
   message type can be Binding Request, Binding Response, Binding Error
   Response, Shared Secret Request, Shared Secret Response, or Shared
   Secret Error Response. The transaction ID is used to correlate
   requests and responses. The length indicates the total length of the
   STUN payload, not including the header. This allows STUN to run over
   TCP. Shared Secret Requests are always sent over TCP (indeed, using
   TLS over TCP).

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 Binding Response, and it indicates the source IP address and port
the server saw in the Binding Request. There is also a RESPONSE-
ADDRESS attribute, which contains an IP address and port. The
RESPONSE-ADDRESS attribute can be present in the Binding Request, and
indicates where the Binding Response is to be sent. It's optional,
and when not present, the Binding Response is sent to the source IP
address and port of the Binding Request.

The third attribute is the CHANGE-REQUEST attribute, and it contains
two flags to control the IP address and port used to send the
response. These flags are called "change IP" and "change port"
flags. The CHANGE-REQUEST attribute is allowed only in the Binding
Request. The "change IP" and "change port" flags 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
Binding Responses from a different source IP address and port. The
CHANGE-REQUEST attribute is optional in the Binding Request.

The fourth attribute is the CHANGED-ADDRESS attribute. It is present
in Binding 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 Binding Responses. It indicates the source IP address and
port where the response was sent from. It is useful for detecting
twice NAT configurations.

The sixth attribute is the USERNAME attribute. It is present in a
Shared Secret Response, which provides the client with a temporary
username and password (encoded in the PASSWORD attribute). The
USERNAME is also present in Binding Requests, serving as an index to
the shared secret used for the integrity protection of the Binding
Request. The seventh attribute, PASSWORD, is only found in Shared
Secret Response messages. The eight attribute is the MESSAGE-
INTEGRITY attribute, which contains a message integrity check over
the Binding Request or Binding Response.

The ninth attribute is the ERROR-CODE attribute. This is present in
the Binding Error Response and Shared Secret Error Response. It
indicates the error that has occurred. The tenth attribute is the
UNKNOWN-ATTRIBUTES attribute, which is present in either the Binding
Error Response or Shared Secret Error Response. It indicates the
mandatory attributes from the request which were unknown. The
eleventh attribute is the REFLECTED-FROM attribute, which is present
in Binding Responses. It indicates the IP address and port of the

sender of a Binding Request, used for traceability purposes to
prevent certain denial-of-service attacks.

8. Server Behavior

The server behavior depends on whether the request is a Binding
Request or a Shared Secret Request.

8.1 Binding Requests

A STUN server MUST be prepared to receive Binding Requests on four
address/port combinations - (A1, P1), (A2, P1), (A1, P2), and (A2,
P2). (A1, P1) represent the primary address and port, and these are
the ones obtained through the client discovery procedures below.
Typically, P1 will be port 3478, the default STUN port. A2 and P2
are arbitrary. A2 and P2 are advertised by the server through the
CHANGED-ADDRESS attribute, as described below.

It is RECOMMENDED that the server check the Binding Request for a
MESSAGE-INTEGRITY attribute. If not present, and the server requires
integrity checks on the request, it generates a Binding Error
Response with an ERROR-CODE attribute with response code 401. If the
MESSAGE-INTEGRITY attribute was present, the server computes the HMAC
over the request as described in Section 11.2.8. The key to use
depends on the shared secret mechanism. If the STUN Shared Secret
Request was used, the key MUST be the one associated with the
USERNAME attribute present in the request. If the USERNAME attribute
was not present, the server MUST generate a Binding Error Response.
The Binding Error Response MUST include an ERROR-CODE attribute with
response code 432. If the USERNAME is present, but the server
doesn't remember the shared secret for that USERNAME (because it
timed out, for example), the server MUST generate a Binding Error
Response. The Binding Error Response MUST include an ERROR-CODE
attribute with response code 430. If the server does know the shared
secret, but the computed HMAC differs from the one in the request,
the server MUST generate a Binding Error Response with an ERROR-CODE
attribute with response code 431. The Binding Error Response is sent
to the IP address and port the Binding Request came from, and sent
from the IP address and port the Binding Request was sent to.

Assuming the message integrity check passed, processing continues.
The server MUST check for any attributes in the request with values
less than or equal to 0x7fff which it does not understand. If it
encounters any, the server MUST generate a Binding Error Response,
and it MUST include an ERROR-CODE attribute with a 420 response code.

That response MUST contain an UNKNOWN-ATTRIBUTES attribute listing
the attributes with values less than or equal to 0x7fff which were
not understood. The Binding Error Response is sent to the IP address
and port the Binding Request came from, and sent from the IP address
and port the Binding Request was sent to.

Assuming the request was correctly formed, the server MUST generate a
single Binding Response. The Binding Response MUST contain the same
transaction ID contained in the Binding Request. The length in the
message header MUST contain the total length of the message in bytes,
excluding the header. The Binding Response MUST have a message type
of "Binding Response".

The server MUST add a MAPPED-ADDRESS attribute to the Binding
Response. The IP address component of this attribute MUST be set to
the source IP address observed in the Binding Request. The port
component of this attribute MUST be set to the source port observed
in the Binding Request.

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

The source address and port of the Binding Response depend on the
value of the CHANGE-REQUEST attribute and on the address and port the
Binding Request was received on, and are summarized in Table 1.

Let Da represent the destination IP address of the Binding Request
(which will be either A1 or A2), and Dp represent the destination
port of the Binding Request (which will be either P1 or P2). Let Ca
represent the other address, so that if Da is A1, Ca is A2. If Da is
A2, Ca is A1. Similarly, let Cp represent the other port, so that if
Dp is P1, Cp is P2. If Dp is P2, Cp is P1. If the "change port"
flag was set in CHANGE-REQUEST attribute of the Binding Request, and
the "change IP" flag was not set, the source IP address of the
Binding Response MUST be Da and the source port of the Binding
Response MUST be Cp. If the "change IP" flag was set in the Binding
Request, and the "change port" flag was not set, the source IP
address of the Binding Response MUST be Ca and the source port of the
Binding Response MUST be Dp. When both flags are set, the source IP
address of the Binding Response MUST be Ca and the source port of the
Binding Response MUST be Cp. If neither flag is set, or if the
CHANGE-REQUEST attribute is absent entirely, the source IP address of
the Binding Response MUST be Da and the source port of the Binding
Response MUST be Dp.

Flags	Source Address	Source Port	CHANGED-ADDRESS
none	Da	Dp	Ca:Cp
Change IP	Ca	Dp	Ca:Cp
Change port	Da	Cp	Ca:Cp
Change IP and			
Change port	Ca	Cp	Ca:Cp

Table 1: Impact of Flags on Packet Source and CHANGED-ADDRESS

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

The server MUST add a CHANGED-ADDRESS attribute to the Binding
Response. This contains the source IP address and port that would be
used if the client had set the "change IP" and "change port" flags in
the Binding Request. As summarized in Table 1, these are Ca and Cp,
respectively, regardless of the value of the CHANGE-REQUEST flags.

If the Binding Request contained both the USERNAME and MESSAGE-
INTEGRITY attributes, the server MUST add a MESSAGE-INTEGRITY
attribute to the Binding Response. The attribute contains an HMAC
[13] over the response, as described in Section 11.2.8. The key to
use depends on the shared secret mechanism. If the STUN Shared
Secret Request was used, the key MUST be the one associated with the
USERNAME attribute present in the Binding Request.

If the Binding Request contained a RESPONSE-ADDRESS attribute, the
server MUST add a REFLECTED-FROM attribute to the response. If the
Binding Request was authenticated using a username obtained from a
Shared Secret Request, the REFLECTED-FROM attribute MUST contain the
source IP address and port where that Shared Secret Request came
from. If the username present in the request was not allocated using
a Shared Secret Request, the REFLECTED-FROM attribute MUST contain
the source address and port of the entity which obtained the
username, as best can be verified with the mechanism used to allocate
the username. If the username was not present in the request, and
the server was willing to process the request, the REFLECTED-FROM
attribute SHOULD contain the source IP address and port where the
request came from.

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.

8.2 Shared Secret Requests

   Shared Secret Requests are always received on TLS connections. When
   the server receives a request from the client to establish a TLS
   connection, it MUST proceed with TLS, and SHOULD present a site
   certificate. The TLS ciphersuite TLS_RSA_WITH_AES_128_CBC_SHA [4]
   SHOULD be used. Client TLS authentication MUST NOT be done, since
   the server is not allocating any resources to clients, and the
   computational burden can be a source of attacks.

   If the server receives a Shared Secret Request, it MUST verify that
   the request arrived on a TLS connection. If it did not receive the
   request over TLS, it MUST generate a Shared Secret Error Response,
   and it MUST include an ERROR-CODE attribute with a 433 response code.
   The destination for the error response depends on the transport on
   which the request was received. If the Shared Secret Request was
   received over TCP, the Shared Secret Error Response is sent over the
   same connection the request was received on. If the Shared Secret
   Request was receive over UDP, the Shared Secret Error Response is
   sent to the source IP address and port that the request came from.

   The server MUST check for any attributes in the request with values
   less than or equal to 0x7fff which it does not understand. If it
   encounters any, the server MUST generate a Shared Secret Error
   Response, and it MUST include an ERROR-CODE attribute with a 420
   response code. That response MUST contain an UNKNOWN-ATTRIBUTES
   attribute listing the attributes with values less than or equal to
   0x7fff which were not understood. The Shared Secret Error Response
   is sent over the TLS connection.

   All Shared Secret Error Responses MUST contain the same transaction
   ID contained in the Shared Secret Request. The length in the message
   header MUST contain the total length of the message in bytes,
   excluding the header. The Shared Secret Error Response MUST have a
   message type of "Shared Secret Error Response" (0x0112).

   Assuming the request was properly constructed, the server creates a
   Shared Secret Response. The Shared Secret Response MUST contain the
   same transaction ID contained in the Shared Secret Request. The
   length in the message header MUST contain the total length of the
   message in bytes, excluding the header. The Shared Secret Response
   MUST have a message type of "Shared Secret Response". The Shared
   Secret Response MUST contain a USERNAME attribute and a PASSWORD
   attribute. The USERNAME attribute serves as an index to the
   password, which is contained in the PASSWORD attribute. The server
   can use any mechanism it chooses to generate the username. However,
   the username MUST be valid for a period of at least 10 minutes.
   Validity means that the server can compute the password for that

username. There MUST be a single password for each username. In
other words, the server cannot, 10 minutes later, assign a different
password to the same username. The server MUST hand out a different
username for each distinct Shared Secret Request. Distinct, in this
case, implies a different transaction ID. It is RECOMMENDED that the
server explicitly invalidate the username after ten minutes. It MUST
invalidate the username after 30 minutes. The PASSWORD contains the
password bound to that username. The password MUST have at least 128
bits. The likelihood that the server assigns the same password for
two different usernames MUST be vanishingly small, and the passwords
MUST be unguessable. In other words, they MUST be a
cryptographically random function of the username.

These requirements can still be met using a stateless server, by
intelligently computing the USERNAME and PASSWORD. One approach is
to construct the USERNAME as:

    USERNAME = <prefix,rounded-time,clientIP,hmac>

Where prefix is some random text string (different for each shared
secret request), rounded-time is the current time modulo 20 minutes,
clientIP is the source IP address where the Shared Secret Request
came from, and hmac is an HMAC [13] over the prefix, rounded-time,
and client IP, using a server private key.

The password is then computed as:

    password = <hmac(USERNAME,anotherprivatekey)>

With this structure, the username itself, which will be present in
the Binding Request, contains the source IP address where the Shared
Secret Request came from. That allows the server to meet the
requirements specified in Section 8.1 for constructing the
REFLECTED-FROM attribute. The server can verify that the username
was not tampered with, using the hmac present in the username.

The Shared Secret Response is sent over the same TLS connection the
request was received on. The server SHOULD keep the connection open,
and let the client close it.

9. Client Behavior

The behavior of the client is very straightforward. Its task is to
discover the STUN server, obtain a shared secret, formulate the
Binding Request, handle request reliability, and process the Binding
Responses.

9.1 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 using the SRV procedures specified in RFC 2782 [3].

   Specifically, the service name is "stun". The protocol is "udp" for
   sending Binding Requests, or "tcp" for sending Shared Secret
   Requests. 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.

   For STUN requests, failure occurs if there is a transport failure of
   some sort (generally, due to fatal ICMP errors in UDP or connection
   failures in TCP). Failure also occurs if the transaction fails due
   to timeout. This occurs 9.5 seconds after the first request is sent,
   for both Shared Secret Requests and Binding Requests. See Section
   9.3 for details on transaction timeouts for Binding Requests. If a
   failure occurs, the client SHOULD create a new request, which is
   identical to the previous, but has a different transaction ID and
   MESSAGE INTEGRITY attribute (the HMAC will change because the
   transaction ID has changed). That request is sent to the next
   element in the list as specified by RFC 2782.

   The default port for STUN requests is 3478, for both TCP and UDP.
   Administrators SHOULD use this port in their SRV records, but MAY use
   others.

   If no SRV records were found, the client performs an A 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.

9.2 Obtaining a Shared Secret

   As discussed in Section 12, there are several attacks possible on
   STUN systems. Many of these are prevented through integrity of
   requests and responses. To provide that integrity, STUN makes use of
   a shared secret between client and server, used as the keying
   material for an HMAC used in both the Binding Request and Binding
   Response. STUN allows for the shared secret to be obtained in any
   way (for example, Kerberos [14]). However, it MUST have at least 128

bits of randomness. In order to ensure interoperability, this
specification describes a TLS-based mechanism. This mechanism,
described in this section, MUST be implemented by clients and
servers.

First, the client determines the IP address and port that it will
open a TCP connection to. This is done using the discovery
procedures in Section 9.1. The client opens up the connection to
that address and port, and immediately begins TLS negotiation [2].
The client MUST verify the identity of the server. To do that, it
follows the identification procedures defined in Section 3.1 of RFC
2818 [5]. Those procedures assume the client is dereferencing a URI.
For purposes of usage with this specification, the client treats the
domain name or IP address used in Section 9.1 as the host portion of
the URI that has been dereferenced.

Once the connection is opened, the client sends a Shared Secret
request. This request has no attributes, just the header. The
transaction ID in the header MUST meet the requirements outlined for
the transaction ID in a binding request, described in Section 9.3
below. The server generates a response, which can either be a Shared
Secret Response or a Shared Secret Error Response.

If the response was a Shared Secret Error Response, the client checks
the response code in the ERROR-CODE attribute. Interpretation of
those response codes is identical to the processing of Section 9.4
for the Binding Error Response.

If a client receives a Shared Secret Response with an attribute whose
type is greater than 0x7fff, the attribute MUST be ignored. If the
client receives a Shared Secret Response with an attribute whose type
is less than or equal to 0x7fff, the response is ignored.

If the response was a Shared Secret Response, it will contain a short
lived username and password, encoded in the USERNAME and PASSWORD
attributes, respectively.

The client MAY generate multiple Shared Secret Requests on the
connection, and it MAY do so before receiving Shared Secret Responses
to previous Shared Secret Requests. The client SHOULD close the
connection as soon as it has finished obtaining usernames and
passwords.

Section 9.3 describes how these passwords are used to provide
integrity protection over Binding Requests, and Section 8.1 describes
how it is used in Binding Responses.

9.3 Formulating the Binding Request

   A Binding Request formulated by the client follows the syntax rules
   defined in Section 11. Any two requests that are not bit-wise
   identical, and 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 distributed between 0 and 2**128 - 1.
   The large range is needed because the transaction ID serves as a form
   of randomization, helping to prevent replays of previously signed
   responses from the server. The message type of the request MUST be
   "Binding Request".

   The RESPONSE-ADDRESS attribute is optional in the Binding Request.
   It is used if the client wishes the response to be sent to a
   different IP address and port than the one the request was sent from.
   This is useful for determining whether the client is behind a
   firewall, and for applications that have separated control and data
   components. See Section 10.3 for more details. The CHANGE-REQUEST
   attribute is also optional. Whether it is present depends on what
   the application is trying to accomplish. See Section 10 for some
   example uses.

   The client SHOULD add a MESSAGE-INTEGRITY and USERNAME attribute to
   the Binding Request. This MESSAGE-INTEGRITY attribute contains an
   HMAC [13]. The value of the username, and the key to use in the
   MESSAGE-INTEGRITY attribute depend on the shared secret mechanism.
   If the STUN Shared Secret Request was used, the USERNAME must be a
   valid username obtained from a Shared Secret Response within the last
   nine minutes. The shared secret for the HMAC is the value of the
   PASSWORD attribute obtained from the same Shared Secret Response.

   Once formulated, the client sends the Binding 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. Retransmissions
   continue with intervals of 1.6s until a response is received, or a
   total of 9 requests have been sent. If no response is received by 1.6
   seconds after the last request has been sent, the client SHOULD
   consider the transaction to have failed. In other words, requests
   would be sent at times 0ms, 100ms, 300ms, 700ms, 1500ms, 3100ms,
   4700ms, 6300ms, and 7900ms. At 9500ms, the client considers the
   transaction to have failed if no response has been received.

9.4 Processing Binding Responses

   The response can either be a Binding Response or Binding Error
   Response. Binding Error Responses are always received on the source
   address and port the request was sent from. A Binding Response will

be received on the address and port placed in the RESPONSE-ADDRESS
attribute of the request. If none was present, the Binding Responses
will be received on the source address and port the request was sent
from.

If the response is a Binding Error Response, the client checks the
response code from the ERROR-CODE attribute of the response. For a
400 response code, the client SHOULD display the reason phrase to the
user. For a 420 response code, the client SHOULD retry the request,
this time omitting any attributes listed in the UNKNOWN-ATTRIBUTES
attribute of the response. For a 430 response code, the client
SHOULD obtain a new shared secret, and retry the Binding Request with
a new transaction. For 401 and 432 response codes, if the client had
omitted the USERNAME or MESSAGE-INTEGRITY attribute as indicated by
the error, it SHOULD try again with those attributes. For a 431
response code, the client SHOULD alert the user, and MAY try the
request again after obtaining a new username and password. For a 500
response code, the client MAY wait several seconds and then retry the
request. For a 600 response code, the client MUST NOT retry the
request, and SHOULD display the reason phrase to the user. Unknown
attributes between 400 and 499 are treated like a 400, unknown
attributes between 500 and 599 are treated like a 500, and unknown
attributes between 600 and 699 are treated like a 600. Any response
between 100 and 399 MUST result in the cessation of request
retransmissions, but otherwise is discarded.

If a client receives a response with an attribute whose type is
greater than 0x7fff, the attribute MUST be ignored. If the client
receives a response with an attribute whose type is less than or
equal to 0x7fff, request retransmissions MUST cease, but the entire
response is otherwise ignored.

If the response is a Binding Response, the client SHOULD check the
response for a MESSAGE-INTEGRITY attribute. If not present, and the
client placed a MESSAGE-INTEGRITY attribute into the request, it MUST
discard the response. If present, the client computes the HMAC over
the response as described in Section 11.2.8. The key to use depends
on the shared secret mechanism. If the STUN Shared Secret Request
was used, the key MUST be same as used to compute the MESSAGE-
INTEGRITY attribute in the request. If the computed HMAC differs
from the one in the response, the client MUST discard the response,
and SHOULD alert the user about a possible attack. If the computed
HMAC matches the one from the response, processing continues.

Reception of a response (either Binding Error Response or Binding
Response) to a Binding Request will terminate retransmissions of that
request. However, clients MUST continue to listen for responses to a
Binding Request for 10 seconds after the first response. If it

receives any responses in this interval with different message types
(Binding Responses and Binding Error Responses, for example) or
different MAPPED-ADDRESSes, it is an indication of a possible attack.
The client MUST NOT use the MAPPED-ADDRESS from any of the responses
it received (either the first or the additional ones), and SHOULD
alert the user.

Furthermore, if a client receives more than twice as many Binding
Responses as the number of Binding Requests it sent, it MUST NOT use
the MAPPED-ADDRESS from any of those responses, and SHOULD alert the
user about a potential attack.

If the Binding Response is authenticated, and the MAPPED-ADDRESS was
not discarded because of a potential attack, the CLIENT MAY use the
MAPPED-ADDRESS and SOURCE-ADDRESS attributes.

10. Use Cases

The rules of Sections 8 and 9 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.

10.1 Discovery Process

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

o On the open Internet

o Firewall that blocks UDP

o 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 a symmetric UDP Firewall)

o Full-cone NAT

o Symmetric NAT

o Restricted cone or restricted port cone NAT

Which of the six scenarios applies can be determined through the flow
chart described in Figure 2. The chart refers only to the sequence
of Binding Requests; Shared Secret Requests will, of course, be
needed to authenticate each Binding Request used in the sequence.

The flow makes use of three tests. In test I, the client sends a
STUN Binding Request to a server, without any flags set in the
CHANGE-REQUEST attribute, 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. In test II, the client sends a
Binding Request with both the "change IP" and "change port" flags
from the CHANGE-REQUEST attribute set. In test III, the client sends
a Binding 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 and port are 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 and port from the CHANGED-ADDRESS attribute
from the response to test I. If the IP address and port returned in
the MAPPED-ADDRESS attribute are not the same as the ones from the
first test I, the client knows its behind a symmetric NAT. If the
address and port are 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 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.

Typically, a client will re-do this discovery process periodically to
detect changes, or look for inconsistent results. It is important to
note that when the discovery process is redone, it should not

generally be done from the same local address and port used in the
previous discovery process. If the same local address and port are
reused, bindings from the previous test may still be in existence,
and these will invalidate the results of the test. Using a different
local address and port for subsequent tests resolves this problem.
An alternative is to wait sufficiently long to be confident that the
old bindings have expired (half an hour should more than suffice).

10.2 Binding Lifetime Discovery

STUN can also be used to discover the lifetimes of the bindings
created by the NAT. In many cases, the client will need to refresh
the binding, either through a new STUN request, or an application
packet, in order for the application to continue to use the binding.
By discovering the binding lifetime, the client can determine how
frequently it needs to refresh.

```
                            +--------+
                            |  Test  |
                            |   I    |
                            +--------+
                                 |
                                 |
                                 V
                                /\              /\
                             N /  \ Y        /  \ Y             +--------+
              UDP     <-------/Resp\-------->/ IP \------------>|  Test  |
              Blocked        \ ?  /         \Same/             |   II   |
                              \  /           \? /              +--------+
                               \/             \/                    |
                                             | N                    |
                                             |                      V
                                             V                     /\
                                        +--------+  Sym.      N /  \
                                        |  Test  |  UDP    <---/Resp\
                                        |   II   |  Firewall   \ ?  /
                                        +--------+             \  /
                                             |                  \/
                                             V                  |Y
               /\                           /\                  |
  Symmetric  N /  \        +--------+   N  /  \                 V
     NAT    <--- / IP \<-----|  Test  |<--- /Resp\             Open
               \Same/      |   I    |     \ ?  /             Internet
                \? /       +--------+      \  /
                 \/                         \/
                 |                          |Y
                 |                          |
                 |                          V
                 |                        Full
                 |                        Cone
                 V                 /\
            +--------+            /  \ Y
            |  Test  |------>/Resp\---->Restricted
            |  III   |            \ ?  /
            +--------+             \  /
                                   \/
                                   |N
                                   |          Port
                                   +------>Restricted
```
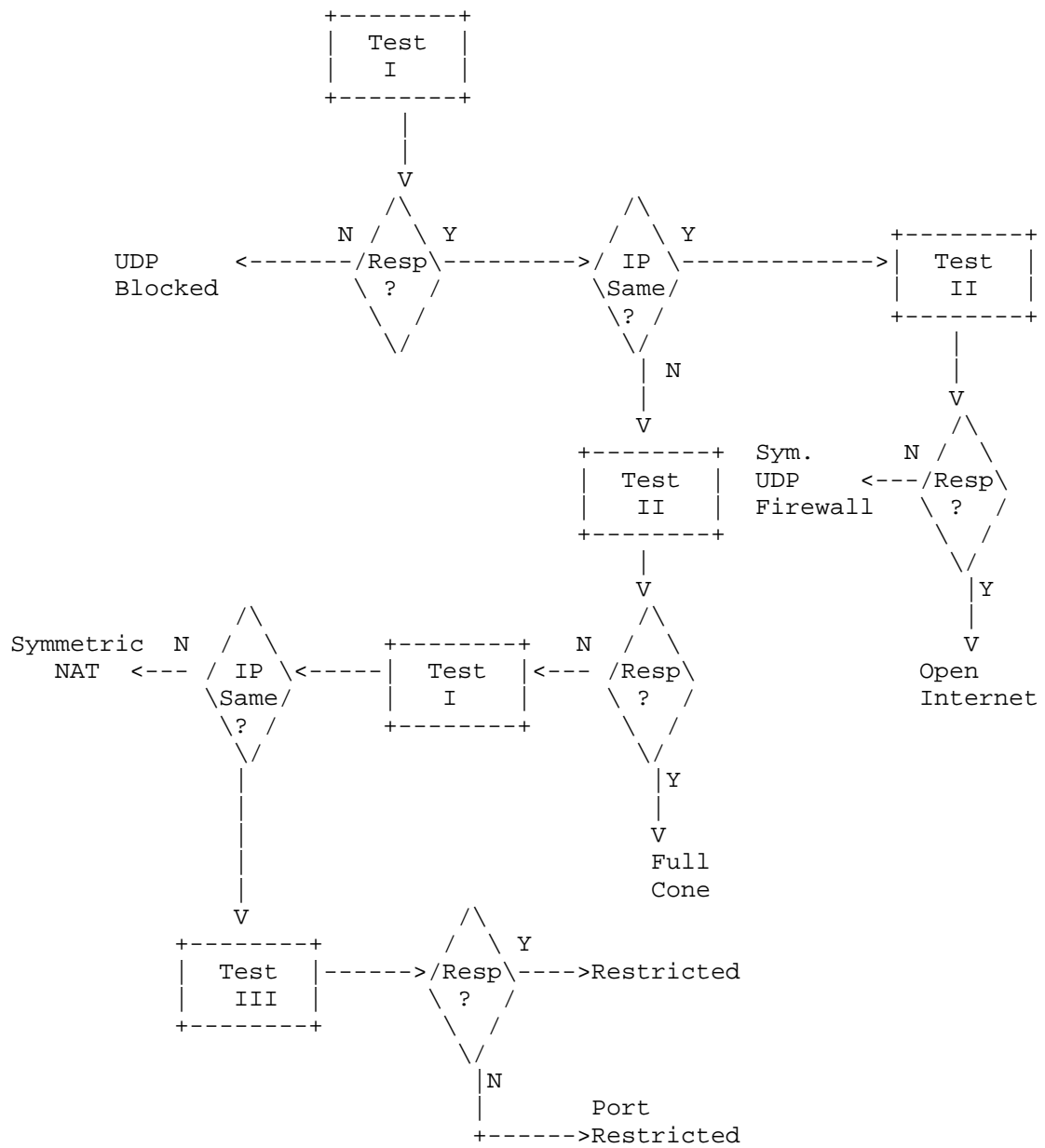
                 Figure 2: Flow for type discovery process

To determine the binding lifetime, the client first sends a Binding
Request to the server from a particular socket, X. This creates a
binding in the NAT. The response from the server contains a MAPPED-
ADDRESS attribute, providing the public address and port on the NAT.
Call this Pa and Pp, respectively. The client then starts a timer
with a value of T seconds. When this timer fires, the client sends
another Binding Request to the server, using the same destination
address and port, but from a different socket, Y. This request
contains a RESPONSE-ADDRESS address attribute, set to (Pa,Pp). This
will create a new binding on the NAT, and cause the STUN server to
send a Binding Response that would match the old binding, if it still
exists. If the client receives the Binding Response on socket X, it
knows that the binding has not expired. If the client receives the
Binding Response on socket Y (which is possible if the old binding
expired, and the NAT allocated the same public address and port to
the new binding), or receives no response at all, it knows that the
binding has expired.

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.

This discovery process takes quite a bit of time, and is something
that will typically be run in the background on a device once it
boots.

It is possible that the client can get inconsistent results each time
this process is run. For example, if the NAT should reboot, or be
reset for some reason, the process may discover a lifetime than is
shorter than the actual one. For this reason, implementations are
encouraged to run the test numerous times, and be prepared to get
inconsistent results.

10.3 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 [12]. 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 sends a Shared Secret
Request to the server, obtains a shared secret, and then sends a
Binding Request to the server. No CHANGE-REQUEST attribute is
present in the Binding Request, and neither is the RESPONSE-ADDRESS
attribute. The Binding Response contains a mapped address. The
control component then formulates a second Binding Request. This
request contains a RESPONSE-ADDRESS, which is set to the mapped
address learned from the previous Binding Response. This Binding
Request is passed to the media component, along with the IP address
and port of the STUN server. The media component sends the Binding
Request. The request goes to the STUN server, which sends the
Binding 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; 30 seconds is
common). 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, and unfortunately, does not work in many
commercial NATs. In such cases, the clients may need to retry using
private addresses.

11. Protocol Details

   This section presents the detailed encoding of a STUN message.

   STUN is a request-response protocol. Clients send a request, and the
   server sends a response. There are two requests, Binding Request,
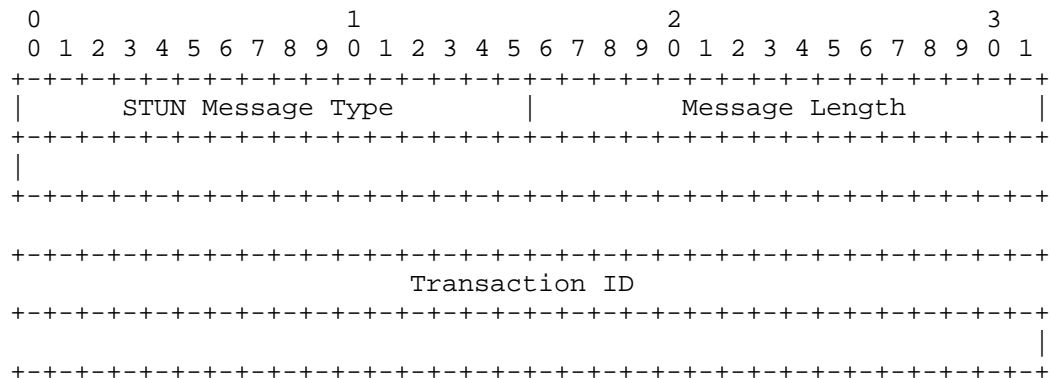   and Shared Secret Request. The response to a Binding Request can

either be the Binding Response or Binding Error Response. The
response to a Shared Secret Request can either be a Shared Secret
Response or a Shared Secret Error Response.

STUN messages are encoded using binary fields. All integer fields
are carried in network byte order, that is, most significant byte
(octet) first. This byte order is commonly known as big-endian. The
transmission order is described in detail in Appendix B of RFC 791
[6]. Unless otherwise noted, numeric constants are in decimal (base
10).

11.1 Message Header

All STUN messages consist of a 20 byte header:

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|      STUN Message Type        |         Message Length        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+

+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
                         Transaction ID
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

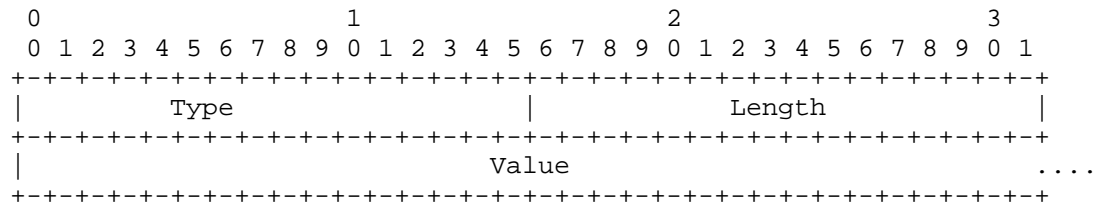The Message Types can take on the following values:

```
0x0001  :  Binding Request
0x0101  :  Binding Response
0x0111  :  Binding Error Response
0x0002  :  Shared Secret Request
0x0102  :  Shared Secret Response
0x0112  :  Shared Secret Error Response
```

The message length is the count, in bytes, of the size of the
message, not including the 20 byte header.

The transaction ID is a 128 bit identifier. It also serves as salt
to randomize the request and the response. All responses carry the
same identifier as the request they correspond to.

11.2 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:

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|         Type                  |            Length             |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                             Value                             ....
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

   The following types are defined:

   0x0001: MAPPED-ADDRESS
   0x0002: RESPONSE-ADDRESS
   0x0003: CHANGE-REQUEST
   0x0004: SOURCE-ADDRESS
   0x0005: CHANGED-ADDRESS
   0x0006: USERNAME
   0x0007: PASSWORD
   0x0008: MESSAGE-INTEGRITY
   0x0009: ERROR-CODE
   0x000a: UNKNOWN-ATTRIBUTES
   0x000b: REFLECTED-FROM

   To allow future revisions of this specification to add new attributes
   if needed, the attribute space is divided into optional and mandatory
   ones. Attributes with values greater than 0x7fff are optional, which
   means that the message can be processed by the client or server even
   though the attribute is not understood. Attributes with values less
   than or equal to 0x7fff are mandatory to understand, which means that
   the client or server cannot process the message unless it understands
   the attribute.

   The MESSAGE-INTEGRITY attribute MUST be the last attribute within a
   message. 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.

   Table 2 indicates which attributes are present in which messages. An
   M indicates that inclusion of the attribute in the message is
   mandatory, O means its optional, C means it's conditional based on
   some other aspect of the message, and N/A means that the attribute is
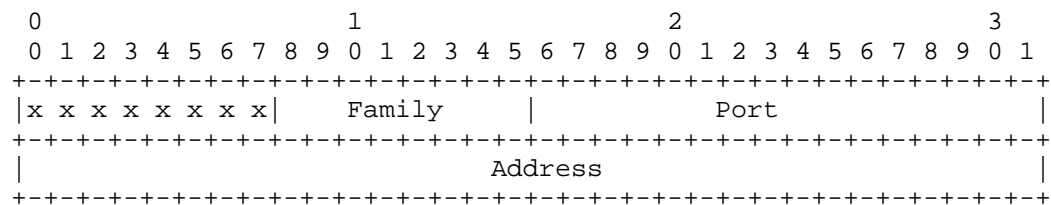   not applicable to that message type.

Att.	Binding Req.	Binding Resp.	Binding Error Resp.	Shared Secret Req.	Shared Secret Resp.	Shared Secret Error Resp.
MAPPED-ADDRESS	N/A	M	N/A	N/A	N/A	N/A
RESPONSE-ADDRESS	O	N/A	N/A	N/A	N/A	N/A
CHANGE-REQUEST	O	N/A	N/A	N/A	N/A	N/A
SOURCE-ADDRESS	N/A	M	N/A	N/A	N/A	N/A
CHANGED-ADDRESS	N/A	M	N/A	N/A	N/A	N/A
USERNAME	O	N/A	N/A	N/A	M	N/A
PASSWORD	N/A	N/A	N/A	N/A	M	N/A
MESSAGE-INTEGRITY	O	O	N/A	N/A	N/A	N/A
ERROR-CODE	N/A	N/A	M	N/A	N/A	M
UNKNOWN-ATTRIBUTES	N/A	N/A	C	N/A	N/A	C
REFLECTED-FROM	N/A	C	N/A	N/A	N/A	N/A

Table 2: Summary of Attributes

The length refers to the length of the value element, expressed as an
unsigned integral number of bytes.

11.2.1 MAPPED-ADDRESS

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

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|x x x x x x x x|    Family     |           Port                |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                             Address                           |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

The port is a network byte ordered representation of the mapped port.
The address family is always 0x01, corresponding to IPv4. The first
8 bits of the MAPPED-ADDRESS are ignored, for the purposes of
aligning parameters on natural boundaries. The IPv4 address is 32
bits.

11.2.2 RESPONSE-ADDRESS

The RESPONSE-ADDRESS attribute indicates where the response to a
Binding Request should be sent. Its syntax is identical to MAPPED-
ADDRESS.

11.2.3 CHANGED-ADDRESS

   The CHANGED-ADDRESS attribute indicates the IP address and port where
   responses would have been sent from if the "change IP" and "change
   port" flags had been set in the CHANGE-REQUEST attribute of the
   Binding Request. The attribute is always present in a Binding
   Response, independent of the value of the flags. Its syntax is
   identical to MAPPED-ADDRESS.

11.2.4 CHANGE-REQUEST

   The CHANGE-REQUEST attribute is used by the client to request that
   the server use a different address and/or port when sending the
   response. The attribute is 32 bits long, although only two bits (A
   and B) are used:

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 A B 0|
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

   The meaning of the flags is:

   A: This is the "change IP" flag. If true, it requests the server
      to send the Binding Response with a different IP address than the
      one the Binding Request was received on.

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

11.2.5 SOURCE-ADDRESS

   The SOURCE-ADDRESS attribute is present in Binding 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.

11.2.6 USERNAME

   The USERNAME attribute is used for message integrity. It serves as a
   means to identify the shared secret used in the message integrity
   check. The USERNAME is always present in a Shared Secret Response,
   along with the PASSWORD. It is optionally present in a Binding
   Request when message integrity is used.

The value of USERNAME is a variable length opaque value. Its length
MUST be a multiple of 4 (measured in bytes) in order to guarantee
alignment of attributes on word boundaries.

11.2.7 PASSWORD

The PASSWORD attribute is used in Shared Secret Responses. It is
always present in a Shared Secret Response, along with the USERNAME.

The value of PASSWORD is a variable length value that is to be used
as a shared secret. Its length MUST be a multiple of 4 (measured in
bytes) in order to guarantee alignment of attributes on word
boundaries.

11.2.8 MESSAGE-INTEGRITY

The MESSAGE-INTEGRITY attribute contains an HMAC-SHA1 [13] of the
STUN message. It can be present in Binding Requests or Binding
Responses. Since it uses the SHA1 hash, the HMAC will be 20 bytes.
The text used as input to HMAC is the STUN message, including the
header, up to and including the attribute preceding the MESSAGE-
INTEGRITY attribute. That text is then padded with zeroes so as to be
a multiple of 64 bytes. As a result, the MESSAGE-INTEGRITY attribute
MUST be the last attribute in any STUN message. The key used as
input to HMAC depends on the context.

11.2.9 ERROR-CODE

The ERROR-CODE attribute is present in the Binding Error Response and
Shared Secret Error Response. It is a numeric value in the range of
100 to 699 plus a textual reason phrase encoded in UTF-8, and is
consistent in its code assignments and semantics with SIP [10] and
HTTP [15]. The reason phrase is meant for user consumption, and can
be anything appropriate for the response code. The lengths of the
reason phrases MUST be a multiple of 4 (measured in bytes). This can
be accomplished by added spaces to the end of the text, if necessary.
Recommended reason phrases for the defined response codes are
presented below.

To facilitate processing, the class of the error code (the hundreds
digit) is encoded separately from the rest of the code.

```
    0                   1                   2                   3
    0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |                   0                     |Class|     Number    |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |      Reason Phrase (variable)                                ..
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

The class represents the hundreds digit of the response code. The
value MUST be between 1 and 6. The number represents the response
code modulo 100, and its value MUST be between 0 and 99.

The following response codes, along with their recommended reason
phrases (in brackets) are defined at this time:

400 (Bad Request): The request was malformed. The client should not
     retry the request without modification from the previous
     attempt.

401 (Unauthorized): The Binding Request did not contain a MESSAGE-
     INTEGRITY attribute.

420 (Unknown Attribute): The server did not understand a mandatory
     attribute in the request.

430 (Stale Credentials): The Binding Request did contain a MESSAGE-
     INTEGRITY attribute, but it used a shared secret that has
     expired. The client should obtain a new shared secret and try
     again.

431 (Integrity Check Failure): The Binding Request contained a
     MESSAGE-INTEGRITY attribute, but the HMAC failed verification.
     This could be a sign of a potential attack, or client
     implementation error.

432 (Missing Username): The Binding Request contained a MESSAGE-
     INTEGRITY attribute, but not a USERNAME attribute. Both must be
     present for integrity checks.

433 (Use TLS): The Shared Secret request has to be sent over TLS, but
     was not received over TLS.

500 (Server Error): The server has suffered a temporary error. The
     client should try again.

600 (Global Failure:) The server is refusing to fulfill the request.
     The client should not retry.

11.2.10 UNKNOWN-ATTRIBUTES

   The UNKNOWN-ATTRIBUTES attribute is present only in a Binding Error
   Response or Shared Secret Error Response when the response code in
   the ERROR-CODE attribute is 420.

   The attribute contains a list of 16 bit values, each of which
   represents an attribute type that was not understood by the server.
   If the number of unknown attributes is an odd number, one of the
   attributes MUST be repeated in the list, so that the total length of
   the list is a multiple of 4 bytes.

```
0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|      Attribute 1 Type           |     Attribute 2 Type        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|      Attribute 3 Type           |     Attribute 4 Type    ...
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

11.2.11 REFLECTED-FROM

   The REFLECTED-FROM attribute is present only in Binding Responses,
   when the Binding Request contained a RESPONSE-ADDRESS attribute. The
   attribute contains the identity (in terms of IP address) of the
   source where the request came from. Its purpose is to provide
   traceability, so that a STUN server cannot be used as a reflector for
   denial-of-service attacks.

   Its syntax is identical to the MAPPED-ADDRESS attribute.

12. Security Considerations

12.1 Attacks on STUN

   Generally speaking, attacks on STUN can be classified into denial of
   service attacks and eavesdropping attacks. Denial of service attacks
   can be launched against a STUN server itself, or against other
   elements using the STUN protocol.

   STUN servers create state through the Shared Secret Request
   mechanism. To prevent being swamped with traffic, a STUN server
   SHOULD limit the number of simultaneous TLS connections it will hold
   open by dropping an existing connection when a new connection request
   arrives (based on an Least Recently Used (LRU) policy, for example).
   Similarly, it SHOULD limit the number of shared secrets it will
   store, in the event that the server is storing the shared secrets.

The attacks of greater interest are those in which the STUN server
and client are used to launch DOS attacks against other entities,
including the client itself.

Many of the attacks require the attacker to generate a response to a
legitimate STUN request, in order to provide the client with a faked
MAPPED-ADDRESS. The attacks that can be launched using such a
technique include:

12.1.1 Attack I: DDOS Against a Target

In this case, the attacker provides a large number of clients with
the same faked MAPPED-ADDRESS that points to the intended target.
This will trick all the STUN clients into thinking that their
addresses are equal to that of the target. The clients then hand out
that address in order to receive traffic on it (for example, in SIP
or H.323 messages). However, all of that traffic becomes focused at
the intended target. The attack can provide substantial
amplification, especially when used with clients that are using STUN
to enable multimedia applications.

12.1.2 Attack II: Silencing a Client

In this attack, the attacker seeks to deny a client access to
services enabled by STUN (for example, a client using STUN to enable
SIP-based multimedia traffic). To do that, the attacker provides
that client with a faked MAPPED-ADDRESS. The MAPPED-ADDRESS it
provides is an IP address that routes to nowhere. As a result, the
client won't receive any of the packets it expects to receive when it
hands out the MAPPED-ADDRESS.

This exploitation is not very interesting for the attacker. It
impacts a single client, which is frequently not the desired target.
Moreover, any attacker that can mount the attack could also deny
service to the client by other means, such as preventing the client
from receiving any response from the STUN server, or even a DHCP
server.

12.1.3 Attack III: Assuming the Identity of a Client

This attack is similar to attack II. However, the faked MAPPED-
ADDRESS points to the attacker themself. This allows the attacker to
receive traffic which was destined for the client.

12.1.4 Attack IV: Eavesdropping

   In this attack, the attacker forces the client to use a MAPPED-
   ADDRESS that routes to itself. It then forwards any packets it
   receives to the client. This attack would allow the attacker to
   observe all packets sent to the client. However, in order to launch
   the attack, the attacker must have already been able to observe
   packets from the client to the STUN server. In most cases (such as
   when the attack is launched from an access network), this means that
   the attacker could already observe packets sent to the client. This
   attack is, as a result, only useful for observing traffic by
   attackers on the path from the client to the STUN server, but not
   generally on the path of packets being routed towards the client.

12.2 Launching the Attacks

   It is important to note that attacks of this nature (injecting
   responses with fake MAPPED-ADDRESSes) require that the attacker be
   capable of eavesdropping requests sent from the client to the server
   (or to act as a MITM for such attacks). This is because STUN
   requests contain a transaction identifier, selected by the client,
   which is random with 128 bits of entropy. The server echoes this
   value in the response, and the client ignores any responses that
   don't have a matching transaction ID. Therefore, in order for an
   attacker to provide a faked response that is accepted by the client,
   the attacker needs to know what the transaction ID in the request
   was. The large amount of randomness, combined with the need to know
   when the client sends a request, precludes attacks that involve
   guessing the transaction ID.

   Since all of the above attacks rely on this one primitive - injecting
   a response with a faked MAPPED-ADDRESS - preventing the attacks is
   accomplished by preventing this one operation. To prevent it, we
   need to consider the various ways in which it can be accomplished.
   There are several:

12.2.1 Approach I: Compromise a Legitimate STUN Server

   In this attack, the attacker compromises a legitimate STUN server
   through a virus or Trojan horse. Presumably, this would allow the
   attacker to take over the STUN server, and control the types of
   responses it generates.

   Compromise of a STUN server can also 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.

12.2.2 Approach II: DNS Attacks

   STUN servers are discovered using DNS SRV records. If an attacker
   can compromise the DNS, it can inject fake records which map a domain
   name to the IP address of a STUN server run by the attacker. This
   will allow it to inject fake responses to launch any of the attacks
   above.

12.2.3 Approach III: Rogue Router or NAT

   Rather than compromise the STUN server, an attacker can cause a STUN
   server to generate responses with the wrong MAPPED-ADDRESS by
   compromising a router or NAT on the path from the client to the STUN
   server. When the STUN request passes through the rogue router or
   NAT, it rewrites the source address of the packet to be that of the
   desired MAPPED-ADDRESS. This address cannot be arbitrary. If the
   attacker is on the public Internet (that is, there are no NATs
   between it and the STUN server), and the attacker doesn't modify the
   STUN request, the address has to have the property that packets sent
   from the STUN server to that address would route through the
   compromised router. This is because the STUN server will send the
   responses back to the source address of the request. With a modified
   source address, the only way they can reach the client is if the
   compromised router directs them there. If the attacker is on the
   public Internet, but they can modify the STUN request, they can
   insert a RESPONSE-ADDRESS attribute into the request, containing the
   actual source address of the STUN request. This will cause the
   server to send the response to the client, independent of the source
   address the STUN server sees. This gives the attacker the ability to
   forge an arbitrary source address when it forwards the STUN request.

   If the attacker is on a private network (that is, there are NATs
   between it and the STUN server), the attacker will not be able to
   force the server to generate arbitrary MAPPED-ADRESSes in responses.
   They will only be able force the STUN server to generate MAPPED-
   ADDRESSes which route to the private network. This is because the
   NAT between the attacker and the STUN server will rewrite the source
   address of the STUN request, mapping it to a public address that
   routes to the private network. Because of this, the attacker can
   only force the server to generate faked mapped addresses that route
   to the private network. Unfortunately, it is possible that a low
   quality NAT would be willing to map an allocated public address to
   another public address (as opposed to an internal private address),
   in which case the attacker could forge the source address in a STUN
   request to be an arbitrary public address. This kind of behavior
   from NATs does appear to be rare.

12.2.4 Approach IV: MITM

   As an alternative to approach III, if the attacker can place an
   element on the path from the client to the server, the element can
   act as a man-in-the-middle. In that case, it can intercept a STUN
   request, and generate a STUN response directly with any desired value
   of the MAPPED-ADDRESS field. Alternatively, it can forward the STUN
   request to the server (after potential modification), receive the
   response, and forward it to the client. When forwarding the request
   and response, this attack is subject to the same limitations on the
   MAPPED-ADDRESS described in Section 12.2.3.

12.2.5 Approach V: Response Injection Plus DoS

   In this approach, the attacker does not need to be a MITM (as in
   approaches III and IV). Rather, it only needs to be able to
   eavesdrop onto a network segment that carries STUN requests. This is
   easily done in multiple access networks such as ethernet or
   unprotected 802.11. To inject the fake response, the attacker
   listens on the network for a STUN request. When it sees one, it
   simultaneously launches a DoS attack on the STUN server, and
   generates its own STUN response with the desired MAPPED-ADDRESS
   value. The STUN response generated by the attacker will reach the
   client, and the DoS attack against the server is aimed at preventing
   the legitimate response from the server from reaching the client.
   Arguably, the attacker can do without the DoS attack on the server,
   so long as the faked response beats the real response back to the
   client, and the client uses the first response, and ignores the
   second (even though it's different).

12.2.6 Approach VI: Duplication

   This approach is similar to approach V. The attacker listens on the
   network for a STUN request. When it sees it, it generates its own
   STUN request towards the server. This STUN request is identical to
   the one it saw, but with a spoofed source IP address. The spoofed
   address is equal to the one that the attacker desires to have placed
   in the MAPPED-ADDRESS of the STUN response. In fact, the attacker
   generates a flood of such packets. The STUN server will receive the
   one original request, plus a flood of duplicate fake ones. It
   generates responses to all of them. If the flood is sufficiently
   large for the responses to congest routers or some other equipment,
   there is a reasonable probability that the one real response is lost
   (along with many of the faked ones), but the net result is that only
   the faked responses are received by the STUN client. These responses
   are all identical and all contain the MAPPED-ADDRESS that the
   attacker wanted the client to use.

   The flood of duplicate packets is not needed (that is, only one faked
   request is sent), so long as the faked response beats the real
   response back to the client, and the client uses the first response,
   and ignores the second (even though it's different).

   Note that, in this approach, launching a DoS attack against the STUN
   server or the IP network, to prevent the valid response from being
   sent or received, is problematic. The attacker needs the STUN server
   to be available to handle its own request. Due to the periodic
   retransmissions of the request from the client, this leaves a very
   tiny window of opportunity. The attacker must start the DoS attack
   immediately after the actual request from the client, causing the
   correct response to be discarded, and then cease the DoS attack in
   order to send its own request, all before the next retransmission
   from the client. Due to the close spacing of the retransmits (100ms
   to a few seconds), this is very difficult to do.

   Besides DoS attacks, there may be other ways to prevent the actual
   request from the client from reaching the server. Layer 2
   manipulations, for example, might be able to accomplish it.

   Fortunately, Approach IV is subject to the same limitations
   documented in Section 12.2.3, which limit the range of MAPPED-
   ADDRESSes the attacker can cause the STUN server to generate.

12.3 Countermeasures

   STUN provides mechanisms to counter the approaches described above,
   and additional, non-STUN techniques can be used as well.

   First off, it is RECOMMENDED that networks with STUN clients
   implement ingress source filtering (RFC 2827 [7]). This is
   particularly important for the NATs themselves. As Section 12.2.3
   explains, NATs which do not perform this check can be used as
   "reflectors" in DDoS attacks. Most NATs do perform this check as a
   default mode of operation. We strongly advise people that purchase
   NATs to ensure that this capability is present and enabled.

   Secondly, it is RECOMMENDED that STUN servers be run on hosts
   dedicated to STUN, with all UDP and TCP ports disabled except for the
   STUN ports. This is to prevent viruses and Trojan horses from
   infecting STUN servers, in order to prevent their compromise. This
   helps mitigate Approach I (Section 12.2.1).

   Thirdly, to prevent the DNS attack of Section 12.2.2, Section 9.2
   recommends that the client verify the credentials provided by the
   server with the name used in the DNS lookup.

Finally, all of the attacks above rely on the client taking the
mapped address it learned from STUN, and using it in application
layer protocols. If encryption and message integrity are provided
within those protocols, the eavesdropping and identity assumption
attacks can be prevented. As such, applications that make use of
STUN addresses in application protocols SHOULD use integrity and
encryption, even if a SHOULD level strength is not specified for that
protocol. For example, multimedia applications using STUN addresses
to receive RTP traffic would use secure RTP [16].

The above three techniques are non-STUN mechanisms. STUN itself
provides several countermeasures.

Approaches IV (Section 12.2.4), when generating the response locally,
and V (Section 12.2.5) require an attacker to generate a faked
response. This attack is prevented using the message integrity
mechanism provided in STUN, described in Section 8.1.

Approaches III (Section 12.2.3) IV (Section 12.2.4), when using the
relaying technique, and VI (12.2.6), however, are not preventable
through server signatures. Both approaches are most potent when the
attacker can modify the request, inserting a RESPONSE-ADDRESS that
routes to the client. Fortunately, such modifications are
preventable using the message integrity techniques described in
Section 9.3. However, these three approaches are still functional
when the attacker modifies nothing but the source address of the STUN
request. Sadly, this is the one thing that cannot be protected
through cryptographic means, as this is the change that STUN itself
is seeking to detect and report. It is therefore an inherent
weakness in NAT, and not fixable in STUN. To help mitigate these
attacks, Section 9.4 provides several heuristics for the client to
follow. The client looks for inconsistent or extra responses, both
of which are signs of the attacks described above. However, these
heuristics are just that - heuristics, and cannot be guaranteed to
prevent attacks. The heuristics appear to prevent the attacks as we
know how to launch them today. Implementors should stay posted for
information on new heuristics that might be required in the future.
Such information will be distributed on the IETF MIDCOM mailing list,
midcom@ietf.org.

12.4 Residual Threats

None of the countermeasures listed above can prevent the attacks
described in Section 12.2.3 if the attacker is in the appropriate
network paths. Specifically, consider the case in which the attacker
wishes to convince client C that it has address V. The attacker
needs to have a network element on the path between A and the server
(in order to modify the request) and on the path between the server

and V so that it can forward the response to C. Furthermore, if
there is a NAT between the attacker and the server, V must also be
behind the same NAT. In such a situation, the attacker can either
gain access to all the application-layer traffic or mount the DDOS
attack described in Section 12.1.1. Note that any host which exists
in the correct topological relationship can be DDOSed. It need not
be using STUN.

13. IANA Considerations

   STUN cannot be extended. Changes to the protocol are made through a
   standards track revision of this specification. As a result, no IANA
   registries are needed. Any future extensions will establish any
   needed registries.

14. 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 (RFC 3424 [17]). 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.

14.1 Problem Definition

   From RFC 3424 [17], 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 detect the presence of one or more
      NATs between it and another client, where the second client is
      reachable from the first, but it is not known whether the second
      client resides on the public Internet.

   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.

14.2 Exit Strategy

   From [17], 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 10.1, 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 middleboxes. Thus, STUN provides a way to help transition to
   full midcom-aware networks.

14.3 Brittleness Introduced by STUN

   From [17], 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. There 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 accommodate
      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 depending 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 to wait for an incoming message, 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 the STUN server as an additional network element
   introduces another point of failure. If the client cannot locate
   a STUN server, or if the server should be unavailable due to
   failure, the application cannot function.

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.

o The discovery of binding lifetimes is prone to error. It assumes
   that the same lifetime will exist for all bindings. This may not
   be true if the NAT uses dynamic binding lifetimes to handle
   overload, or if the NAT itself reboots during the discovery
   process.

o STUN imposes some restrictions on the network topologies for
   proper operation. If client A obtains an address from STUN server
   X, and sends it to client B, B may not be able to send to A using
   that IP address. The address will not work if any of the
   following is true:

   - The STUN server is not in an address realm that is a common
      ancestor (topologically) of both clients A and B. For example,
      consider client A and B, both of which have residential NAT
      devices. Both devices connect them to their cable operators,
      but both clients have different providers. Each provider has a
      NAT in front of their entire network, connecting it to the
      public Internet. If the STUN server used by A is in A's cable
      operator's network, an address obtained by it will not be
      usable by B. The STUN server must be in the network which is a
      common ancestor to both - in this case, the public Internet.

   - The STUN server is in an address realm that is a common
      ancestor to both clients, but both clients are behind the same
      NAT connecting to that address realm. For example, if the two
      clients in the previous example had the same cable operator,
      that cable operator had a single NAT connecting their network
      to the public Internet, and the STUN server was on the public
      Internet, the address obtained by A would not be usable by B.
      That is because some NATs will not accept an internal packet
      sent to a public IP address which is mapped back to an internal
      address. To deal with this, additional protocol mechanisms or
      configuration parameters need to be introduced which detect
      this case.

   o Most significantly, STUN introduces potential security threats
      which cannot be eliminated. This specification describes
      heuristics that can be used to mitigate the problem, but it is
      provably unsolvable given what STUN is trying to accomplish.
      These security problems are described fully in Section 12.

14.4 Requirements for a Long Term Solution

   From [17], 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.

14.5 Issues with Existing NAPT Boxes

   From [17], 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 [12] 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 [18]. 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 occasional silence packets to keep
   the binding alive. However, the result is additional brittleness;
   proper operation depends on 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.

14.6 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 unpredictable,
   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.

15. Acknowledgments

   The authors would like to thank Cedric Aoun, Pete Cordell, Cullen
   Jennings, Bob Penfield and Chris Sullivan for their comments, and
   Baruch Sterman and Alan Hawrylyshen for initial implementations.
   Thanks for Leslie Daigle, Allison Mankin, Eric Rescorla, and Henning
   Schulzrinne for IESG and IAB input on this work.

16. Normative References

   [1] Bradner, S., "Key words for use in RFCs to indicate requirement
        levels", BCP 14, RFC 2119, March 1997.

   [2] Dierks, T. and C. Allen, "The TLS protocol Version 1.0", RFC
        2246, January 1999.

   [3] Gulbrandsen, A., Vixie, P. and L. Esibov, "A DNS RR for
        specifying the location of services (DNS SRV)", RFC 2782,
        February 2000.

   [4] Chown, P., "Advanced Encryption Standard (AES) Ciphersuites for
        Transport Layer Security (TLS)", RFC 3268, June 2002.

   [5] Rescorla, E., "HTTP over TLS", RFC 2818, May 2000.

   [6] Postel, J., "Internet Protocol", STD 5, RFC 791, September 1981.

   [7] Ferguson, P. and D. Senie, "Network Ingress Filtering: Defeating
        Denial of Service Attacks which employ IP Source Address
        Spoofing", BCP 38, RFC 2827, May 2000.

17. Informative References

   [8] Senie, D., "Network Address Translator (NAT)-Friendly
        Application Design Guidelines", RFC 3235, January 2002.

   [9] Srisuresh, P., Kuthan, J., Rosenberg, J., Molitor, A. and A.
        Rayhan, "Middlebox Communication Architecture and Framework",
        RFC 3303, August 2002.

   [10] Rosenberg, J., Schulzrinne, H., Camarillo, G., Johnston, A.,
        Peterson, J., Sparks, R., Handley, M. and E. Schooler, "SIP:
        Session Initiation Protocol", RFC 3261, June 2002.

   [11] Holdrege, M. and P. Srisuresh, "Protocol Complications with the
        IP Network Address Translator", RFC 3027, January 2001.

[12] Schulzrinne, H., Casner, S., Frederick, R. and V. Jacobson,
     "RTP: A Transport Protocol for Real-Time Applications", RFC
     1889, January 1996.

[13] Krawczyk, H., Bellare, M. and R. Canetti, "HMAC: Keyed-Hashing
     for Message Authentication", RFC 2104, February 1997.

[14] Kohl, J. and C. Neuman, "The kerberos Network Authentication
     Service (V5)", RFC 1510, September 1993.

[15] Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L.,
     Leach, P. and T. Berners-Lee, "Hypertext Transfer Protocol --
     HTTP/1.1", RFC 2616, June 1999.

[16] Baugher M., et al., "The secure real-time transport protocol",
     Work in Progress.

[17] Daigle, L., Editor, "IAB Considerations for UNilateral Self-
     Address Fixing (UNSAF) Across Network Address Translation", RFC
     3424, November 2002.

[18] Huitema, C., "RTCP attribute in SDP", Work in Progress.

18. 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
101 Cooper St
Santa Cruz, CA 95060

EMail: rohan@cisco.com

19. Full Copyright Statement

Acknowledgement