Behave J. Rosenberg
Internet-Draft Cisco Systems
Expires: August 31, 2006 R. Mahy
Plantronics
C. Huitema
Microsoft
February 27, 2006

Obtaining Relay Addresses from Simple Traversal of UDP Through NAT (STUN)
draft-ietf-behave-turn-00

Status of this Memo

Copyright Notice

Abstract

This specification defines a usage of the Simple Traversal of UDP Through NAT (STUN) Protocol for asking the STUN server to relay packets towards a client. This usage is useful for elements behind NATs whose mapping behavior is address and port dependent. The

extension purposefully restricts the ways in which the relayed address can be used. In particular, it prevents users from running well general purpose servers from ports obtained from the STUN server.

Table of Contents

1. Introduction

The Simple Traversal of UDP Through NAT (STUN) [1] provides a suite of tools for facilitating the traversal of NAT. Specifically, it defines the Binding Request, which is used by a client to determine its reflexive transport address towards the STUN server. The reflexive transport address can be used by the client for receiving packets from peers, but only when the client is behind "good" NATs. In particular, if a client is behind a NAT whose mapping behavior [15] is address or address and port dependent (sometimes called "bad" NATs), the reflexive transport address will not be usable for corresponding with a peer.

The only way to obtain a transport address that can be used for corresponding with a peer through such a NAT is to make use of a relay. The relay sits on the public side of the NAT, and allocates transport addresses to clients reaching it from behind the private side of the NAT. These allocated addresses are from interfaces on the relay. When the relay receives a packet on one of these allocated addresses, the relay forwards it towards the client.

This specification defines a usage of STUN, called the relay usage, that allows a client to request an address on the STUN server itself, so that the STUN server acts as a relay. To accomplish that, this usage defines two new requests - the Allocate request and the Set Active Destination request. It also defines two indications - Data and Send. The Allocate request is the principal component of this usage, and it is used to provide the client with a transport address that is relayed through the STUN server. A transport address which relays through an intermediary is called a relayed transport address.

Though a relayed address is highly likely to work when corresponding with a peer, it comes at high cost to the provider of the STUN server. As a consequence, relayed transport addresses should only be used as a last resort. Protocols using relayed transport addresses should make use of mechanisms to dynamically determine whether such an address is actually needed. One such mechanism, defined for multimedia session establishment protocols based on the offer/answer protocol [7] is Interactive Connectivity Establishment (ICE) [14].

The mechanism defined here was previously a standalone protocol called Traversal Using Relay NAT (TURN), and is now defined as a usage of STUN.

2. Terminology

In this document, the key words MUST, MUST NOT, REQUIRED, SHALL, SHALL NOT, SHOULD, SHOULD NOT, RECOMMENDED, MAY, and OPTIONAL are to

be interpreted as described in RFC 2119 [2] and indicate requirement levels for compliant TURN implementations.

3. Definitions

Relayed Transport Address: A transport address that terminates on a server, and is forwarded towards the client. The STUN Allocate Request can be used to obtain a relayed transport address, for example.

4. Overview of Operation

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

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


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

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

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

Figure 1

The STUN relay usage defines several new messages that add the ability of the STUN server to act as a relay for packets. The client sends an Allocate request to the server. This request is authenticated by the server. The client can include requests for

specific ports, transport protocols and IP addresses to be allocated by the STUN server. The STUN server honors these if it can, and then generates a response to the Allocate request. This response informs the client of the address and port allocated to it, called the allocated transport address. This address and port resides on the STUN server itself.

The allocation will remain active as long as the client refreshes it with subsequent Allocate requests. A basic negotiation mechanism is defined which allows the client to request a specific lifetime, and for the server to lower it and indicate the actual lifetime.

Once the client has obtained the allocated address from the STUN server, it can use it to receive packets. However, when a packet arrives at the allocated address, the STUN server does not forward the packet. Instead, it will only forward a packet received from some correspondent X if the client had previously sent a packet to X through the relay. In that way, the relay is much like a NAT itself.

To send a packet through the relay towards some correspondent X, the client issues a Send Indication to the STUN server. This indication includes the destination address and port where the packet should be sent to, and the data to send. The relay takes the data, and sends it to X. It also adds a permission towards X, so that X can now send packets to the allocated address, and the STUN server will relay those towards the client. The clients are relayed towards the client by encapsulating them in a Data Indication. This is a STUN Indication which contains the data that was received by the STUN server, along with the identity of the correspondent.

Since the primary usage of the STUN relay usage is in support of multimedia communications, efficiency is a key design goal of this STUN extension. The mechanism described so far will allow a client behind the NAT to communicate with a correspondent. However, all packets sent to and from the client will be encapsulated as STUN Indications; a Send indication for data sent from the client to the STUN server, and a Data indication for packets from the STUN server to the client. This encapsulation adds 44 bytes to each packet. With voice contents typically around 30 bytes (30 milliseconds of G.729), this is a significant amount of overhead.

To optimize it, the relay usage provides a cut-through technique. When the client has decided it would like to optimize the transmission of packets with a particular correspondent, it issues a Set Active Destination request to the server, and provides the IP address and port of the correspondent. After a brief time during which the client and server can determine they are synchronized on the usage of the mechanism, the server enables an optimized path.

Packets received from this correspondent are relayed to the client without encapsulation in a STUN Data indication, and the client can send unencapsulated packets to the server, which will be forwarded towards the correspondent. This mechanism requires the STUN server and client to disambiguate STUN from other packets when received on the same IP address and port. That is provided by the magic cookie field in the STUN message. This cookie reduces the likelihood of a data packet from being confused with a STUN packet to 2.32x10^-10, which is deemed sufficiently unlikely.

To do all of this, the STUN server will maintain a binding between an internal 5-tuple and 1 or more external 5-tuples, as shown in Figure 2. The internal 5-tuple represents the "connection" between the STUN server and the STUN client. It is the actual connection in the case of TCP, and in the case of UDP, it is the combination of the IP address and port from which the STUN client sent its Allocate Request, with the IP address and port to which that Allocate Request was sent. The external local transport address is the IP address and port allocated to the STUN client (the allocated transport address). The external 5-tuple is the combination of the external local transport address and the IP address and port of an external client that the STUN client is communicating with through the STUN server. Initially, there aren't any external 5-tuples, since the STUN client hasn't communicated with any other hosts yet. As packets are received on or sent from the allocated transport address, external 5-tuples are created.

```
                                                               +---------+
                                                               |         |
                                                               | External|
                                                             / | Client  |
                                                           //  |         |
                                                          /    |         |
                                                        //     +---------+
                                                       /
                                                     //
                     +-+                            /
                     | |                           /
                     | |                         //
      +---------+    | |          +---------+   /              +---------+
      |         |    |N|          |         | //               |         |
      | STUN    |    | |          |         |/                 | External|
      | Client  |----|A|----------|   STUN  |------------------| Client  |
      |         |    | |^        ^|  Server |^                ^|         |
      |         |    |T||        ||         ||                ||         |
      +---------+    | ||        |+---------+|                |+---------+
         ^           | ||        |           |                |
         |           | ||        |           |                |
         |           +-+|        |           |                |
         |              |        |           |                |
         |

                    Internal     Internal    External         External
     Client         Remote       Local       Local            Remote
     Performing     Transport    Transport   Transport        Transport
     Allocations    Address      Address     Address          Address

                        |          |            |                |
                        +-----+----+            +--------+-------+
                              |                          |
                              |                          |

                            Internal                External
                            5-Tuple                 5-tuple
```

Figure 2

5. Applicability Statement

STUN requires all usages to define the applicability of the usage [1]. This section contains that information for the relay usage.

The relayed transport address obtained from the Allocate request has specific properties which limit its applicability. The transport address will only be useful for applications that require a client to place a transport address into a protocol message, with the expectation that the client will be able to receive packets from a small number of hosts (typically one), and only after sending packets towards those hosts. Because of this limitation, relayed transport addresses obtained from an Allocate request are only useful when combined with rendezvous protocols of some sort, which allow the client to discover the addresses of the hosts it will be corresponding with. Examples of such protocols include the Session Initiation Protocol (SIP) [6].

This limitation is purposeful. Because a client must send a packet to a peer before it can receive packets from that peer, relayed transport addresses obtained from the Allocate request can not be used to run general purpose servers, such as a web or email server. This means that the relay usage can be safely permitted to pass through NATs and firewalls without fear of compromising the purpose of having them there in the first place. Indeed, a relayed transport address obtained from TURN has many of the properties of a transport address obtained from a NAT whose filtering policies are address dependent, but whose mapping properties are endpoint independent [15], and thus "good" NATs. Indeed, to some degree, the relay turns a bad NAT into a good NAT by, quite ironically, adding another NAT function - the relay itself.

6. Client Discovery of Server

STUN requires all usages to define the mechanism by which a client discovers the server [1]. This section contains that information for the relay usage.

The relay usage differs from the other usages defined in [1] in that it demands substantial resources from the STUN server. In addition, it seems likely that administrators might want to block connections from clients to the STUN server for relaying separated from connections for the purposes of binding discovery. As a consequence, the relay usage is defined to run on a separate port from other usages. The client discovers the address and port of the STUN server for the relay usage using the same DNS procedures defined in [1], but using an SRV service name of "stun-relay" instead of just "stun".

[[TODO: Still need to sort out discovery for TLS vs. non-TLS, usage of NAPTR, and so on.]]

7. Server Determination of Usage

STUN requires all usages to define the mechanism by which the server determines the usage [1]. This section contains that information for the relay usage.

The relay usage is defined by a specific set of requests and indications. As a consequence, the server knows that this usage in being used because those request and indications were used.

8. New Requests and Indications

This usage defines two new requests (along with their success and error responses) and two indications. It also defines processing rules for the STUN server and client on receipt of non-STUN messages. See Section 11 and Section 12

The new messages are:

```
0x0003  :  Allocate Request
0x0103  :  Allocate Response
0x0113  :  Allocate Error Response
0x0004  :  Send Indication
0x0115  :  Data Indication
0x0006  :  Set Active Destination Request
0x0106  :  Set Active Destination Response
0x0116  :  Set Active Destination Error Response
0x0007  :  Connect Request
0x0107  :  Connect Response
0x0117  :  Connect Error Response
```

The server will receive the Allocate Request, Send Indiaction and Set Active Destination Request on the transport address it has advertised in DNS or that has been provided to clients through configuration. However, the server will also receive non-STUN packets, meant for relaying, on this port. STUN packets are disambiguated from data packets through the MAGIC-COOKIE in the STUN header. Similarly, the client will receive Allocate Responses, Allocate Error Responses, Data Indications, Set Active Destination Responses, and Set Active Destination Error Responses on the ephemeral port it uses to connect to the STUN server. It will also receive non-STUN packets, relayed to it by the STUN server, on this port. Like the server, it disambiguates STUN and non-STUN packets through the presence of the magic cookie.

[[OPEN ISSUE: The usage of a magic cookie in the STUN header provides a nice, generic way to disambiguate stun from application packets for

the turn usage, as well as sip-outbound, ice and other applications. But, it introduces a problem as a consequence of this generalization. When TURN is used with ICE, the agents will send p2p stun connectivity checks through the turn relay. These being valid stun packets, will also have the same magic cookie, and be processed by the turn server, rather than the ice agent! The proposed remedy for this is to use the DESTINATION-ADDRESS attribute in Allocate requests, indicating the server to which the request is targeted. If the turn server picks up a packet because of a magic cookie, but the destination-address is not it or not there, it would forward the packet as a regular datagram.]]

8.1 Allocate Request

8.1.1 Server Behavior

The server first processes the request according to the general request processing rules in [1]. This includes performing authentication and checking for mandatory unknown attributes. Due to the fact that the STUN server is allocating resources for processing the request, Allocate requests MUST be authenticated, and furthermore, MUST be authenticated using either a shared secret known between the client and server, or a short term password derived from it.

Note that Allocate requests, like all other STUN requests, can be sent to the STUN server over UDP, TCP, or TCP/TLS.

The behavior of the server when receiving an Allocate Request depends on whether the request is an initial one, or a subsequent one. An initial request is one whose source and destination transport address matches the internal remote and local transport addresses of an existing internal 5-tuple. A subsequent request is one whose source and destination transport address do not match the internal remote and local transport address of an existing internal 5-tuple.

8.1.1.1 Initial Requests

The server attempts to allocate transport addresses. It first looks for the BANDWIDTH attribute for the request. If present, the server determines whether or not it has sufficient capacity to handle a binding that will generate the requested bandwidth.

If it does, the server attempts to allocate a transport address for the client. The Allocate request can contain several additional attributes that allow the client to request specific characteristics of the transport address. First, the server checks for the REQUESTED-TRANSPORT attribute. This indicates the transport protocol

requested by the client. This specification defines values for UDP and TCP. The server MUST allocate a port using the requested transport protocol. If the REQUESTED-TRANSPORT attribute contains a value of the transport protocol unknown to the server, or known to the server but not supported by the server, the server MUST reject the request and include a 442 (Unsupported Transport Protocol) in the response, or else redirect the request. [[OPEN ISSUE: Should we include a list of supported ones? Is this really an issue? If its just ever TCP and UDP its not needed. Can always add it later, as the hooks are here.]]. If the request did not contain a REQUESTED-TRANSPORT attribute, the server MUST use the same transport protocol as the request arrived on.

As a consequence of the REQUESTED-TRANSPORT attribute, it is possible for a client to connect to the server over UDP and request a TCP transport address, and for it to connect to the server over TCP (and TLS, which uses TCP) and request a UDP transport address. In such a case, the server will relay data between them.

Next, the server checks for the REQUESTED-IP attribute. If present, it indicates a specific interface from which the client would like its transport address allocated. If this interface is not a valid one for allocations on the server, the server MUST reject the request and include a 443 (Invalid IP Address) error code in the response, or else redirect the request to a server that is known to support this IP address. If the IP address is one that is valid for allocations (presumably, the server is configured to know the set of IP addresses from which it performs allocations), the server MUST provide an allocation from that IP address. If the attribute was not present, the selection of an IP address is at the discretion of the server.

Finally, the server checks for the REQUESTED-PORT attribute. If present, it indicates a specific port property desired by the client. If the property is for a Specific Port, the server MUST attempt to allocate that specific port for the client. If the port is not available, the server MUST reject the request with a 444 (Invalid Port) response or redirect to an alternate server. If the property is for an even port, the server MUST attempt to allocate an even port for the client. If an even port cannot be obtained, the server MUST reject the request with a 444 (Invalid Port) response or redirect to an alternate server. If the property is for an odd port, the server MUST attempt to allocate an odd port for the client. If an odd port cannot be obtained, the server MUST reject the request with a 444 (Invalid Port) response or redirect to an alternate server. Finally, the Even port with hold of the next higher port is similar to Even port. It is a request for an even port, and MUST be rejected by the server if an even port cannot be provided, or redirected to an alternate server. However, it is also a hint from the client that

the client will request the next higher port with a separate Allocate request. As such, it is a request for the server to allocate an even port whose one higher port is also available, and furthermore, a request for the server to not allocate that one higher port to any other request except for one that asks for that port explicitly. The server can honor this request for adjacency at its discretion. The only constraint is that the allocated port has to be even.

If any of the requested or desired constraints cannot be met, whether it be bandwidth, transport protocol, IP address or port, instead of rejecting the request, the server can alternately redirect the client to a different server that may be able to fulfill the request. This is accomplished using the 300 error response and ALTERNATE-SERVER attribute.

Furthermore, if the clients source port was in the range 1024-65535, it is RECOMMENDED that the server allocate a port in that range. If the clients source port was in the range of 1-1024, port selection is at the discrtion of the administrator. It is RECOMMENDED that a port in the range of 1024-65535 be allocated. This is one of several ways to prohibit relayed transport addresses from being used to attempt to run standard services. These guidelines are meant to be consistent with [15], since the relay is effectively a NAT.

Once the port is allocated, the server associates it with the internal 5-tuple and fills in that 5-tuple. The internal remote transport address of the internal 5-tuple is set to the source transport address of the Allocate Request. The internal local transport address of the internal 5-tuple is set to the destination transport address of the Allocate Request. For TCP, this amounts to associating the TCP connection from the TURN client with the allocated transport address.

If the Allocate request was authenticated using a shared secret between the client and server, this credential MUST be associated with the allocation. If the request was authenticated using a short term password derived from a shared secret, that shared secret MUST be associated with the allocation. This is used in subsequent Allocate requests to ensure that only the same client can refresh or modify the characteristics of the allocation it was given.

The allocation created by the Allocate request is also associated with a transport address, called the active destination. This transport address is used for forwarding data through the TURN server, and is described in more detail later. It is initially set to null when the allocation is created. In addition, the allocation created by the server is associated with a set of permissions. Each permission is a specific IP address identifying an external client.

Initially, this list is null. Send Indications, Connect requests and Set Active Destination requests add values to this list.

If the LIFETIME attribute was present in the request, and the value is larger than the maximum duration the server is willing to use for the lifetime of the allocation, the server MAY lower it to that maximum. However, the server MUST NOT increase the duration requested in the LIFETIME attribute. If there was no LIFETIME attribute, the server may choose a default duration at its discretion. In either case, the resulting duration is added to the current time, and a timer, called the allocation expiration timer, is set to fire at or after that time. Section 12.3 discusses behavior when the timer fires. Note that the LIFETIME attribute in the request can be zero. This typically happens for subsequent Allocations, and provides a mechanism to delete the allocation. It will force the immediate firing of the allocation expiration timer.

Once the port has been obtained from the operating system and the activity timer started for the port binding, the server generates an Allocate Response using the general procedures defined in [1]. The transport address allocated to the client MUST be included in the RELAY-ADDRESS attribute in the response. In addition, this response MUST contain the MAPPED-ADDRESS attribute. This allows the client to determine its reflexive transport address in addition to a relayed transport address, from the same Allocate request.

The server MUST add a LIFETIME attribute to the Allocate Response. This attribute contains the duration, in seconds, of the allocation expiration timer associated with this allocation.

The server MUST add a BANDWIDTH attribute to the Allocate Response. This MUST be equal to the attribute from the request, if one was present. Otherwise, it indicates a per-binding cap that the server is placing on the bandwidth usage on each binding. Such caps are needed to prevent against denial-of-service attacks (See Section 13.

The server MUST add, as the final attribute of the request, a MESSAGE-INTEGRITY attribute. The key used in the HMAC MUST be the same as that used to validate the request.

If the allocated port was for TCP, the server MUST be prepared to receive a TCP connection request on that port.

8.1.1.2 Subsequent Requests

A subsequent Allocate request is one received whose source and destination IP address and ports match the internal 5-tuple of an existing allocation. The request is processed used the general

server procedures in [1] and is processed identically to Section 8.1.1.1, with a few important exceptions.

First, the request MUST be authenticated using the same shared secret as the one associated with the allocation, or be authenticated using a short term password derived from that shared secret. If the request was authenticated but not with such a matching credential, the server MUST generate an Allocate Error Response with a 441 response code.

Secondly, if the allocated transport address given out previously to the client still matches the constraints in the request (in terms of request ports, IP addresses and transport protocols), the same allocation granted previously MUST be returned. However, if one of the constraints is not met any longer, because the client changed some aspect of the request, the server MUST free the previous allocation and allocate a new request to the client.

Finally, a subsequent Allocate request will set a new allocation expiration timer for the allocation, effectively canceling the previous timer that was running.

8.1.2 Client Behavior

Client behavior for Allocate requests depends on whether the request is an initial one, for the purposes of obtaining a new relayed transport address, or a subsequent one, used for refreshing an existing allocation.

8.1.2.1 Initial Requests

When a client wishes to obtain a transport address, it sends an Allocate Request to the server. This request is constructed and sent using the general procedures defined in [1]. The server will challenge the request for credentials. The client MAY either provide its credentials to the server directly, else obtain a short-term set of credentials using the Shared Secret request, and then use those as the credentials in the Allocate request.

The client SHOULD include a BANDWIDTH attribute, which indicates the maximum bandwidth that will be used with this binding. If the maximum is unknown, the attribute is not included in the request.

The client MAY request a particular lifetime for the allocation by including it in the LIFETIME attribute in the request.

The client MAY include a REQUESTED-PORT, REQUESTED-TRANSPORT, or REQUESTED-IP attribute in the request to obtain specific types of

transport addresses. Whether these are needed depends on the application using the relay usage. As an example, the Real Time Transport Protocol (RTP) [5] requires that RTP and RTCP ports be even and odd respectively, and contiguous. The REQUESTED-PORT attribute allows the client to ask the relay for those properties.

Processing of the response follows the general procedures of [1]. A successful response will include both a RELAY-ADDRESS and MAPPED-ADDRESS attribute, providing both a relayed transport address and a reflexive transport address, respectively, to the client. The server will expire the allocation after LIFETIME seconds have passed if not refreshed by another Allocate request. The server will allow the user to send and receive no more than the amount of data indicated in the BANDWIDTH attribute.

If the response is an error response and contains a 442, 443 or 444 error code, the client knows that its requested properties could not be met. The client MAY retry with different properties, with the same properties (in a hope that something has changed on the server), or give up, depending on the needs of the application. However, if the client retries, it SHOULD wait 500ms, and if the request fails again, wait 1 second, then 2 seconds, and so on, exponentially backing off.

8.1.2.2 Subsequent Requests

Before 3/4 of the lifetime of the allocation has passed (the lifetime of the allocation is conveyed in the LIFETIME attribute of the Allocate Response), the client SHOULD refresh the allocation with another Allocate Request if it wishes to keep the allocation.

To perform a refresh, the client generates an Allocate Request as described in Section 8.1.2.1. If the initial request was authenticated with a shared secret P that the client holds with the server, or using a short term password derived from P through a Shared Secret request, the client MUST use shared secret P, or a short-term password derived from it, in the subsequent request.

In a successful response, the RELAY-ADDRESS contains the same transport address as previously obtained, indicating that the binding has been refreshed. The LIFETIME attribute indicates the amount of additional time the binding will live without being refreshed. Note that an error response do not imply that the binding has been expired, just that the refresh has failed.

If the client wishes to explicitly remove the allocation because it no longer needs it, it generates a subsequent Allocate request, but sets the LIFETIME attribute to zero. This will cause the server to

remove the allocation.

8.2 Connect Request

The Connect Request is used by a client when it has obtained an allocated transport address that is TCP. The Connect request asks the server to open a TCP connection to a specified destination address, included in the request.

8.2.1 Server Behavior

Once the server has identified a request as a Connect request, the server verifies that it has arrived with a source and destination transport address that matches the internal remote and local transport address of an internal 5-tuple associated with an existing allocation. If there is no matching allocation, the server MUST generate a 437 (No Binding) Send Error Response.

The request MUST be authenticated using the same shared secret as the one associated with the allocation, or be authenticated using a short term password derived from that shared secret. If the request was authenticated but not with such a matching credential, the server MUST generate an error response with a 441 response code.

If the allocation is not for TCP, the server MUST reject the request with a 445 (Operation for TCP Only) response.

If the request does not contain a DESTINATION-ADDRESS attribute, the server sends a Connect response, but otherwise does nothing.

If the request contains a DESTINATION-ADDRESS attribute, the IP address contained within it is added to the permissions for this allocation, if it was not already present. This happens regardless of whether the subsequent TCP connection attempt succeeds or not.

The server then checks to see if it has any TCP connections in existence from the allocated transport address to the IP address and port in DESTINATION-ADDRESS. If it does, the server responds to the request with a Connect response, indicating to the client that a connection exists already.

Next, the server attempts to open a TCP connection from the allocated transport address to the IP address and port in the DESTINATION-ADDRESS attribute. If the connection succeeds, the server generates a Connect Response. If the connection attempt fails or times out, the server generates a Connect Error Response and includes an error response of 446 (Connection Failure). If the connection attempt is still pending prior to the the timeout of the STUN transaction, the

server MUST send a 447 (Connection Timeout) error response. However, the server continues to wait for the connection to get set up. If it succeeds, the client holds on to the connection. The client can retry the request at a later time, and if the connection has been succesfully setup, it will result in a Success Response as described above.

8.2.2 Client Behavior

If a client wishes to send data towards a peer on a TCP allocated transport address, the client must first tell the server to open a TCP connection towards the destination. To do that, the client sends a Connect request to the server. The client MUST NOT send this request for non-TCP allocated transport addresses. The request SHOULD contain a DESTINATION-ADDRESS attribute indicating the desired target for the connection attempt.

If the Connect request generates a successful response, it means that a connection was opened, or was already opened, towards DESTINATION-ADDRESS. If it generates a Connect Error response with a response code of 446, it means that the servers attempt at the connection has failed. If it generates a Connect Error response with a response code of 447, it means that the server is still trying to connect, but the attempt could not be completed before the STUN transaction needed to end. Whether the client wishes to retry depends on the application using the request. If the client wishes to determine the disposition of the attempt, it MAY send a Connect request with the same DESTINATION-ADDRESS at a later time.

[[OPEN ISSUE: yes, this is a hack. STUN transactions were designed for immediate responses, and so the handshake is two-way, like SIP non-INVITE. However, I am reluctant to include yet another new transaction to SIP. The alternative to the above design is to have the server send a request to the client when the connection completes.]]

If the Connect request generates a 437, it means that the client's allocation no longer exists, possibly due to server or network failures. The client MAY obtain a new allocation if the application so desires.

8.3 Set Active Destination Request

8.3.1 Server Behavior

The Set Active Destination Request is used by a client to set an external 5-tuple that will be used as the forwarding destination of all data that isn't to be processed by the STUN server itself. In

addition, all data received from that external client will be forwarded to the STUN client without encapsulation in a Data Indication.

Once the server has identified a request as a Set Active Destination request, the server verifies that it has arrived with a source and destination transport address that matches the internal remote and local transport address of an internal 5-tuple associated with an existing allocation. If there is no matching allocation, the server MUST generate a 437 (No Binding) Send Error Response.

The request MUST be authenticated using the same shared secret as the one associated with the allocation, or be authenticated using a short term password derived from that shared secret. If the request was authenticated but not with such a matching credential, the server MUST generate an error response with a 441 response code.

If the Set Active Destination request contains a DESTINATION-ADDRESS attribute, the IP address contained within it is added to the permissions for this allocation, if it was not already present.

Unfortunately, there is a race condition associated with the active destination concept. Consider the case where the active destination is set, and the server is relaying packets towards the client. The client knows the IP address and port where the packets came from - the current value of the active destination. The client issues a Set Active Destination Request to change the active destination, and receives a response. A moment later, a data packet is received, not encapsulated in a STUN Data Indication. What is the source if this packet? Is it the active destination that existed prior to the Set Active Destination request, or the one after? If the transport between the client and the STUN server is not reliable, there is no way to know.

To deal with this problem, a small state machine is used to force a "cooldown" period during which the server will not relay packets towards the client without encapsulating them. This cooldown period gives enough time for the client to be certain that any old data packets have left the network. Once the cooldown period ends, the server can begin relaying packets without encapsulation. There is an instance of this state machine for each allocation.

```
                +-----+
                |     | Req Recvd, DA absent
                |     |
                |     |
                |     |
                |     V
             +-----------+
             |           |                    timer fires
             |           |                    -----------
             |  None     |                    active=null
             |   Set     |<--------------------------------+
             |           |                                 |
             |           |                                 |
             +-----------+                                 |
                   |                                       |   Req Recvd
                   |                                       |   ---------
                   | Req Recvd, DA present                 |   439
                   | ----------------------                |  +----+
                   | active = DA                           |  |    |
                   |                                       |  |    |
                   |                                       |  |    |
                   V         Req Recvd,                    |  |    V
             +-----------+   DA!=active,absent           +-----------+
             |           |   -----------------           |           |
             |           |   Set timer                   |           |
             |  Set      |------------------------------>|  Trans-   |
             |           |                               |  itioning |
             |           |<------------------------------|           |
             |           |                 timer fires   |           |
             +-----------+                 -----------   +-----------+
                |     ^                    active=DA
                |     |
                |     |
                |     |
                +-----+
                 Req Recvd, DA=active
```

Figure 4

When the allocation is originally created, the active destination is null, and the server sets the state to "None Set". In this state, the server will relay all received packets in encapsulated form towards the client. If the server receives a Set Active Destination request, but the request contained no DESTINATION-ADDRESS attribute, the state machine stays in the same state. The request is responded to with a Set Active Destination Response. If, however, the Set Active Destination request contained a DESTINATION-ADDRESS, the

server sets the active destination to the transport address from the DESTINATION-ADDRESS attribute, and enters the "Set" state. The request is responded to with a Set Active Destination Response. In this state, the server will relay packets from that transport address towards the client in unencapsulated form.

If the server receives another Set Active Destination request while in this state, and the DESTINATION-ADDRESS is present, but has a value equal to the current active destination, the request causes no change. The request is responded to with a Set Active Destination Response. If, however, the request contained a DESTINATION-ADDRESS which did not match the existing active destination, or omitted the active destination, the server enters the "transitioning" state. The request is responded to with a Set Active Destination Response. In this state, the server will forward all packets to the client in encapsulated form. In addition, when this state is entered, the client sets a timer to fire in Ta seconds. If the connection between the client and server is unreliable, this timer SHOULD be configurable. It is RECOMMENDED that it be set to three seconds. If the connection between the client and server is reliable, the timer SHOULD be set to 0 seconds, causing it to fire immediately. This makes the transitioning state transient for reliable transports. The value of the timer used by the server, regardless of the transport protocol, MUST be included in a TIMER-VAL attribute in the Set Active Destination response.

If, while in the "transitioning" state, the server receives a Set Active Destination Request, it generates a Set Active Destination Error Response that includes a 439 (Transitioning) response code. Once the timer fires, the server transitions to the "Set" state if the Set Active Destination request that caused the server to enter "transitioning" had contained the DESTINATION-ADDRESS. In this case, the active destination is set to this transport address. If the Set Active Destination request had not contained a DESTINATION-ADDRESS attribute, the server enters the "Not Set" state and sets the active destination to null.

8.3.2 Client Behavior

The Set Active Destination address allows the client to create an optimized relay function between it and the server. When the server receives packets from a particular preferred external client, the server will forward those packets towards the client without encapsulating them in a Data Indication. Similarly, the client can send non-STUN packets to the server without encapsulation, and these are forwarded to the external client. Sending and receiving data in unencapsulated form is critical for efficiency purposes. One of the primary use cases for the STUN relay usage is in support of Voice

over IP (VoIP), which uses very small UDP packets to begin with. The extra overhead of an additional layer of encapsulation is considered unacceptable.

The Set Active Destination request is used by the client to provide the identity of this preferred external client. The request also has the side effect of adding a permission for the target of the DESTINATION-ADDRESS.

The Set Active Destination address MAY contain a DESTINATION-ADDRESS attribute. This attribute, when present, provides the address of the preferred external client to the server. When absent, it clears the value of the preferred external client.

In order for the client to know where incoming non-STUN packets were sent from, and to be sure where non-STUN packets sent to the server will go to, it is necessary to coordinate the value of the active destination between the client and the server. As discussed above, there is a race condition involved in this coordination which requires a state machine to execute on both the client and the server.

```
                +-----+
                |     |  OK Recvd, DA absent
                |     |
                |     |
                |     |
                |     V
             +-----------+
    439 Recvd|           |                    timer fires
      +------|           |                    -----------
      |      |  None     |                    active=null
      |      |   Set     |<--------------------------------+
      +----->|           |                                 |
             |           |                                 |
             +-----------+                                 |
                   |                                       |
                   |                                       |
                   |  OK Recvd, DA present                 |
                   | ----------------------                |
                   |  active = DA                          |
                   |                                       |
                   |                                       |
                   V          OK Recvd,                    |
             +-----------+    DA!=active,absent          +-----------+
             |           |    -----------------          |           |
             |           |    Set timer                  |           |
             |  Set      |------------------------------>|  Trans-   |
             |           |                               |  itioning |
             |           |<------------------------------|           |
             |           |                 timer fires   |           |
             +-----------+                 -----------   +-----------+
                |     ^                    active=DA
                |     |
                |     |
                |     |
                +-----+
                 439 Recvd,
                  OK Recvd, DA=active
```

Figure 5

The state machine is shown in Figure 5. The client starts in the "None Set" state. When the client is in either the "None Set" or "Set" state, it can send Set Active Destination requests. The transitions in the state machines are governed by responses to those requests. Only success and 439 responses cause changes in state. A 437 response implies that the allocation has been removed, and thus the state machine destroyed. A client MUST NOT send a new Set Active

Destination request prior to the receipt of a response to the previous. The state machine will further limit the transmission of subsequent Set Active Destination requests.

If, while in the "None Set" state, the client sent a Set Active Destination request without a DESTINATION-ADDRESS, and got a successful response, there is no change in state. If a successful response was received, but there was a DESTINATION-ADDRESS in the request, the state machine transitions to the "Set" state, and the client sets the active destination to the value of the DESTINATION-ADDRESS attribute that was in the request.

If, while in the "Set" state, the client sends a Set Active Destination request and received a 439 response, it means that there was a temporal misalignment in the states between client and server. The client thought that the active destination was updated on the server, but the server was still in its transitioning state. When this error is received, the client remains in the "Set" state. The client SHOULD retry its Set Active Destination request, but no sooner than 500ms after receipt of the 439 response. In addition, if, while in the "Set" state, the client sends a Set Active Destination request whose DESTINATION-ADDRESS attribute equals the current active destination, and that request generates a success response, the client remains in the "Set" state.

However, if, while in the "Set" state, the client sends a Set Active Destination request whose DESTINATION-ADDRESS was either absent or not equal to the current active destination, and receives a success response, the client enters the "Transitioning" state. While in this state, the client MUST NOT send a new Set Active Destination request. The value of the active destination remains unchanged. In addition, the client sets a timer. This timer MUST have a value equal to the value of the TIMER-VAL attribute from the Set Active Destination response. This is necessary for coordinating the state machines between client and server.

Once the timer fires, if the DESTINATION-ADDRESS was not absent from the Set Active Destination request which caused the client to start the timer, the client moves back to the "Set" state, and then updates the value of the active destination to the value of DESTINATION-ADDRESS. If DESTINATION-ADDRESS was absent, the client sets the active destination to null and enders the "None Set" state.

8.4 Send Indication

8.4.1 Server Behavior

A Send Indication is sent by a client after it has completed its

Allocate transaction, in order to create permissions in the server and send data to an external client.

Once the server has identified a message as a Send Indication, the server verifies that it has arrived with a source and destination transport address that matches the internal remote and local transport address of an internal 5-tuple associated with an existing allocation. If there is no matching allocation, the indication is discarded. If there was no DESTINATION-ADDRESS, the indication is discarded. If there was no DATA attribute, the indication is discarded.

[[OPEN ISSUE: should message integrity checks be done for send? THey cannot be challenged!]]

The server takes the contents of the DATA attribute present in the indication. If the allocation was a UDP allocation, the server creates a UDP packet whose payload equals that content. The server sets the source IP address of the packet equal to the allocated transport address. The destination transport address is set to the contents of the DESTINATION-ADDRESS attribute. The server then sends the UDP packet. Note that any retransmissions of this packet which might be needed are not handled by the server. It is the clients responsibility to generate another Send indication if needed. If the TURN client hasn't previously sent to this destination IP address and port, an external 5-tuple is instantiated in the TURN server. Its local and remote transport addresses, respectively, are set to the source and destination transport addresses of the UDP packet.

The server then adds the IP address of the DESTINATION-ADDRESS attribute to the permission list for this allocation.

In the case of a TCP allocation, the server checks if it has an existing TCP connection open from the allocated transport address to the address in the DESTINATION-ADDRESS attribute. If so, the server extracts the content of the DATA attribute and sends it on the matching TCP connection. If the server doesn't have an existing TCP connection to the destination, it discards the data and does nothing. The client must first open a TCP connection with the Connect request before it can send data.

8.4.2 Client Behavior

Before receiving any UDP or TCP data, a client has to send first. Prior to the establishment of an active destination, or while the client is in the transitioning state, transmission of data towards a peer through the relay is done using the Send Indication. Indeed, if the client is in the transitioning state, and it wishes to send data

through the relay, it MUST use a Send indication.

For TCP allocated transport addresses, the client MUST first open a connection towards an external client with a Connect request prior to using the Send request. Data sent with a Send request prior to the opening of a TCP connection is discarded silently by the server.

The Send Indication MUST contain a DESTINATION-ADDRESS attribute, which contains the IP address and port that the data is being sent to. The DATA attribute MAY be present, and contains the data that is to be sent towards DESTINATION-ADDRESS. If absent, the server will send an empty UDP packet in the case of UDP. In the case of TCP, the server will do nothing.

Since Send is an Indication, it generates no response. The client must relay on application layer mechanisms to determine if the data was received by the peer.

8.5 Data Indication

8.5.1 Server Behavior

A server MUST send data packets towards the client using a Data Indication under the conditions described in Section 12.1. Data Indications MUST contain a DATA attribute containing the data to send, and MUST contain a REMOTE-ADDRESS attribute indicating where the data came from.

8.5.2 Client Behavior

Once a client has obtained an allocation and created permissions for a particular external client, the server can begin to relay packets from that external client towards the client. If the external client is not the active destination, this data is relayed towards the client in encapsulated form using the Data Indication.

The Data Indication contains two attributes - DATA and REMOTE-ADDRESS. The REMOTE-ADDRESS attribute indicates the source transport address that the request came from, and it will equal the external remote transport address of the external client. When processing this data, a client MUST treat the data as if it came from this address, rather than the stun server itself. The DATA attribute contains the data from the UDP packet or TCP segment that was received. Note that the TURN server will not retransmit this indication over UDP.

9. New Attributes

The STUN relay usage defines the following new attributes:

```
0x000d: LIFETIME
0x0010: BANDWIDTH
0x0011: DESTINATION-ADDRESS
0x0012: REMOTE-ADDRESS
0x0013: DATA
0x0016: RELAY-ADDRESS
0x0018: REQUESTED-PORT
0x0019: REQUESTED-TRANSPORT
0x0020: REQUESTED-IP
0x0021: TIMER-VAL
```

9.1 LIFETIME

The lifetime attribute represents the duration for which the server will maintain an allocation in the absence of data traffic either from or to the client. It is a 32 bit value representing the number of seconds remaining until expiration.

```
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                         Lifetime                              |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

9.2 BANDWIDTH

The bandwidth attribute represents the peak bandwidth, measured in kbits per second, that the client expects to use on the binding. The value represents the sum in the receive and send directions. [[Editors note: Need to define leaky bucket parameters for this.]]

```
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                         Bandwidth                             |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

9.3 DESTINATION-ADDRESS

The DESTINATION-ADDRESS is present in Send Indications and Set Active Destination Requests. It specifies the address and port where the data is to be sent. It is encoded in the same way as MAPPED-ADDRESS.

[[OPEN ISSUE: Should some of thes be xor-encoded? I don't see a need really...]]

9.4 REMOTE-ADDRESS

The REMOTE-ADDRESS is present in Data Indications. It specifies the address and port from which a packet was received. It is encoded in the same way as MAPPED-ADDRESS.

9.5 DATA

The DATA attribute is present in Send Indications and Data Indications. It contains raw payload data that is to be sent (in the case of a Send Request) or was received (in the case of a Data Indication).

9.6 RELAY-ADDRESS

The RELAY-ADDRESS is present in Allocate responses. It specifies the address and port that the server allocated to the client. It is encoded in the same way as MAPPED-ADDRESS.

9.7 REQUESTED-PORT

This attribute allows the client to request certain properties for the port that is allocated by the server. The attribute can be used with any transport protocol that has the notion of a 16 bit port space (including TCP and UDP). The attribute is 32 bits long. Its format is:

```
                                x
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|        Property               |         Port Filter           |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

The property is an unsigned integer from 0 to 65535 which identifies the specific property that is desired. The meaning of the port filter depends on the port property, and is not used for certain port properties.

This specification defines the following port properties:

0x0000: Even Port
0x0001: Odd Port
0x0002: Even Port, hold next higher port
0x0003: Specific Port

Even Port is a request to the server to allocate a port with even parity. The port filter is not used with this property. Odd Port is a request to the server to allocate a port with odd parity. The port filter is not used with this property. Even port, with a hold on the next higher port, is a request to the server to allocate an even port. Furthermore, the client indicates that it will want the next higher port as well. As such, the client requests that the server, if it can, not allocate the next higher port to anyone unless that port is explicitly requested, which the client will itself do. The port filter is not used with this property. Finally, the Specific Port property is a request for a specific port. The port that is requested is contained in the Port filter.

Extensions to the relay usage can define additional port properties. [[TODO: Add IANA registry]]

9.8 REQUESTED-TRANSPORT

This attribute is used by the client to request a specific transport protocol for the allocated transport address. It is a 32 bit unsigned integer. Its values are:

0x0000 0000: UDP
0x0000 0001: TCP

If an Allocate request is sent over TCP and requests a UDP allocation, or an Allocate request is sent over UDP and requests a TCP allocation, the server will relay data between the two transports.

Extensions to the relay usage can define additional transport protocols. [[TODO: Add IANA registry]]

9.9 REQUESTED-IP

The REQUESTED-IP attribute is used by the client to request that a specific IP address be allocated to it. This attribute is needed since it is anticipated that STUN relays will be multi-homed so as to be able to allocate more than 64k transport addresses. As a consequence, a client needing a second transport address on the same interface as a previous one can make that request.

The format of this attribute is identical to MAPPED-ADDRESS. However, the port component of the attribute is ignored by the server. If a client wishes to request a specific IP address and port, it uses both the REQUESTED-IP and REQUESTED-PORT attributes.

9.10 TIMER-VAL

The TIMER-VAL attribute is used only in conjunction with the Set Active Destination response. It conveys from the server, to the client, the value of the timer used in the server state machine. Coordinated values are needed for proper operation of the mechanism.

The attribute is a 32 bit unsigned integer representing the number if milliseconds used by the server for its timer.

10. New Error Response Codes

The STUN relay usage defines the following new Error response codes:

437 (No Binding): A request was received by the server that requires an allocation to be in place. However, there is none yet in place.

439 (Transitioning): A Set Active Destination request was received by the server. However, a previous request was sent within the last few seconds, and the server is still transitioning to that active destination. Please repeat the request later.

441 (Wrong Username): A TURN request was received for an allocated binding, but it did not use the same username and password that were used in the allocation. The client must supply the proper credentials, and if it cannot, it should teardown its binding, allocate a new one time password, and try again.

442 (Unsupported Transport Protocol): The Allocate request asked for a transport protocol to be allocated that is not supported by the server.

443 (Invalid IP Address): The Allocate request asked for a transport address to be allocated from a specific IP address that is not valid on the server.

444 (Invalid Port): The Allocate request asked for a port to be allocated that is not available on the server.

445 (Operation for TCP Only): The client tried to send a request to perform a TCP-only operation on an allocation, and allocation is UDP.

446 (Connection Failure): The attempt by the server to open the connection failed.

447 (Connection Timeout): The attempt by the server to open the connection could not be completed, and is still in progress.

11. Client Procedures

If a client no longer needs a binding, it SHOULD tear it down. For TCP, this is done by closing the connection. For UDP, this is done by performing a refresh, as described in Section 8.1.2.2, but with a LIFETIME attribute indicating a time of 0.

11.1 Receiving and Sending Unencapsulated Data

Once the active destination has been set, a client will receive both STUN and non-STUN data on the socket on which the Allocate Request was sent. If the client receives non-STUN data (disambiguated through the magic cookie), it MUST be processed as if it had a source IP address and port equal to the value of the active destination.

In addition, once the active destination has been set, if the client is in the "Set" state, it MAY send data to the active destination by sending data on that same socket. Unencapsulated data MUST NOT be sent while in the "Not Set" or "Transitioning" states. However, it is RECOMMENDED that the client not send unencapsulated data for approximately 500 milliseconds after the client enters the "Set" state. This eliminates any synchronization problems resulting from network delays. Of course, even if the active destination is set, the client can send data to that destination at any time by using the Send Indication.

12. Server Procedures

Besides the processing of the request and indications described above, this specification defines rules for processing of data packets received by the STUN server. There are two cases - receipt of any packets on an allocated address, and receipt of non-STUN data on its internal local transport address.

12.1 Receiving Data on Allocated Transport Addresses

12.1.1 TCP Processing

If a server receives a TCP connection request on an allocated TCP transport address, it checks the permissions associated with that allocation. If the source IP address of the TCP SYN packet match one

of the permissions, the TCP connection is accepted. Otherwise, it is rejected. No information is passed to the client about the acceptance of the connection; rather, data passed to the client with a source transport address it has not seen before serves this purpose.

If a server receives data on a TCP connection that terminates on the allocated TCP transport address, the server checks the value of the active destination. If it equals the source IP address and port of the data packet (in other words, if the active destination identifies the other side of the TCP connection), the server checks the state machine of the allocation. If the state is "Set", the data is taken from the TCP connection and sent towards the client in unencapsulated form. Otherwise, the data is sent towards the client in a Data Indication, also known as encapsulated form. In this form, the server MUST add a REMOTE-ADDRESS which corresponds to the external remote transport address of the TCP connection, and MUST add a DATA attribute containing the data received on the TCP connection.

Sending of the data towards the client, whether in encapsulated or unencapsulated form, depends on the linkage with the client. If the linkage with the client is over UDP, the data is placed in a UDP datagram and sent over the linkage. Note that the server will not retransmit this data to ensure reliability. If the linkage with the client is over TCP, the data is placed into the TCP connection corresponding to the linkage. If the TCP connection generates an error (because, for example, the incoming TCP packet rate exceeds the throughput of the TCP connection to the client), the data is discarded silently by the server.

Note that, because data is forwarded blindly across TCP bindings, TLS will successfully operate over a TURN allocated TCP port if the linkage to the client is also TCP.

12.1.2 UDP Processing

If a server receives a UDP packet on an allocated UDP transport address, it checks the permissions associated with that allocation. If the source IP address of the UDP packet matches one of the permissions, the UDP packet is accepted. Otherwise, it is discarded.

Assuming the packet is accepted, it must be forwarded to the client. It will be forwarded in either encapsulated or unencapsulated form. To determine which, the server checks the value of the active destination. If it equals the source IP address and port of the UDP packet, the server checks the state machine of the allocation. If the state is "Set", the data is taken from the UDP payload and sent towards the client in unencapsulated form. Otherwise, the data is

sent towards the client in a Data Indication, also known as encapsulated form. In this form, the server MUST add a REMOTE-ADDRESS which corresponds to the external remote transport address of the UDP packet, and MUST add a DATA attribute containing the data payload of the UDP packet.

Sending of the data towards the client, whether in encapsulated or unencapsulated form, depends on the linkage with the client. If the linkage with the client is over UDP, the data is placed in a UDP datagram and sent over the linkage. Note that the server will not retransmit this data to ensure reliability. If the linkage with the client is over TCP, the data is placed into the TCP connection corresponding to the linkage. If the TCP connection generates an error (because, for example, the incoming UDP packet rate exceeds the throughput of the TCP connection), the data is discarded silently by the server.

12.2 Receiving Data on Internal Local Transport Addresses

If a server receives a UDP packet from the client on its internal local transport address, and it is coming from an internal remote transport address associated with an existing allocation, it represents UDP data that the client wishes to forward. If the active destination is not set, the server MUST discard the packet. If the active destination is set, and the allocated transport protocol is TCP, the server selects the TCP connection from the allocated transport address to the active destination. The data is then sent over that connection. If the transmission fails due to a TCP error, the data is discarded silently by the server. If the active destination is set, and the allocated transport protocol is UDP, the server places the data from the client in a UDP payload, and sets the destination address and port to the active destination. The UDP packet is then sent with a source IP address and port equal to the allocated transport address. Note that the server will not retransmit the UDP datagram.

If a server receives data on a TCP connection to a client, the server retrieves the allocation bound to that connection. If the active destination for the allocation is not set, the server MUST discard the data. If the active destination is set, and the allocated transport protocol is TCP, the server selects the TCP connection from the allocated transport address to the active destination. The data is then sent over that connection. If the transmission fails due to a TCP error, the data is discarded silently by the server. If the active destination is set, and the allocated transport protocol is UDP, the server places the data from the client in a UDP payload, and sets the destination address and port to the active destination. The UDP packet is then sent with a source IP address and port equal to

the allocated transport address. Note that the server will not retransmit the UDP datagram.

If a TCP connection from a client is closed, the associated allocation is destroyed. This involves terminating any TCP connections from the allocated transport address to external clients (applicable only when the allocated transport address was TCP), and then freeing the the allocated transport address (and all associated state maintained by the server) for use by other clients.

Note that the state of the allocation, whether it is "Set", "Not Set", or "Transitioning", has no bearing on the rules for forwarding of packets received from clients. Only the value of the active destination is relevant.

12.3 Lifetime Expiration

When the allocation expiration timer for a binding fires, the server MUST destroy the allocation. This involves terminating any TCP connections from the allocated transport address to external clients (applicable only when the allocated transport address was TCP), and then freeing the the allocated transport address (and all associated state maintained by the server) for use by other clients.

[[OPEN ISSUE: This is a change from the previous version, which allowed data traffic to keep allocations alive. This change was made based on implementation considerations, as it allows an easier separation of packet processing and signaling. Is this OK?]]

13. Security Considerations

TODO: Need to spend more time on this.

STUN servers implementing this relay usage allocate bandwidth and port resources to clients, in constrast to the usages defined in [1]. Therefore, a STUN server providing the relay usage requires authentication and authorization of STUN requests. This authentication is provided by mechanisms defined in the STUN specification itself. In particular, digest authentication and the usage of short-term passwords, obtained through a digest exchange over TLS, are available. The usage of short-tem passwords ensures that the Allocate Requests, which often do not run over TLS, are not susceptible to offline dictionary attacks that can be used to guess the long lived shared secret between the client and the server.

Because STUN servers implementing the relay usage allocate resources, they can be susceptible to denial-of-service attacks. All Allocate Requests are authenticated, so that an unknown attacker cannot launch

an attack. An authenticated attacker can generate multiple Allocate Requests, however. To prevent a single malicious user from allocating all of the resources on the server, it is RECOMMENDED that a server implement a modest per user cap on the amount of bandwidth that can be allocated. Such a mechanism does not prevent a large number of malicious users from each requesting a small number of allocations. Attacks as these are possible using botnets, and are difficult to detect and prevent. Implementors of the STUN relay usage should keep up with best practices around detection of anomalous botnet attacks.

A client will use the transport address learned from the RELAY-ADDRESS attribute of the Allocate Response to tell other users how to reach them. Therefore, a client needs to be certain that this address is valid, and will actually route to them. Such validation occurs through the message integrity checks provided in the Allocate response. They can guarantee the authenticity and integrity of the allocated addresss. Note that the STUN relay usage is not susceptible to the attacks described in Section 12.2.3, 12.2.4, 12.2.5 or 12.2.6 of RFC 3489 [[TODO: Update references once 3489bis is more stable]]. These attacks are based on the fact that a STUN server mirrors the source IP address, which cannot be authenticated. STUN does not use the source address of the Allocate Request in providing the RELAY-ADDRESS, and therefore, those attacks do not apply.

The relay usage cannot be used by clients for subverting firewall policies. The relay usage has fairly limited applicability, requiring a user to send a packet to a peer before being able to receive a packet from that peer. This applies to both TCP and UDP. Thus, it does not provide a general technique for externalizing TCP and UDP sockets. Rather, it has similar security properties to the placement of an address-restricted NAT in the network, allowing messaging in from a peer only if the internal client has sent a packet out towards the IP address of that peer. This limitation means that the relay usage cannot be used to run web servers, email servers, SIP servers, or other network servers that service a large number of clients. Rather, it facilitates rendezvous of NATted clients that use some other protocol, such as SIP, to communicate IP addresses and ports for communications.

Confidentiality of the transport addresses learned through Allocate requests does not appear to be that important, and therefore, this capability is not provided.

Relay servers are useful even for users not behind a NAT. They can provide a way for truly anonymous communications. A user can cause a call to have its media routed through a STUN server, so that the

user's IP addresses are never revealed.

TCP transport addresses allocated by Allocate requests will properly work with TLS and SSL. However, any relay addresses learned through an Allcoate will not operate properly with IPSec Authentication Header (AH) [11] in transport mode. IPSec ESP [12] and any tunnel-mode ESP or AH should still operate.

14. IANA Considerations

TODO.

15. 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 [13]. TURN 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.

15.1 Problem Definition

From RFC 3424 [13], 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 problem being solved by TURN is for a client, which may be located behind a NAT of any type, to obtain an IP address and port on the public Internet, useful for applications that require a client to place a transport address into a protocol message, with the expectation that the client will be able to receive packets from a single host that will send to this address. Both UDP and TCP are addressed. It is also possible to send packets so that the recipient sees a source address equal to the allocated address. TURN, by design, does not allow a client to run a server (such as a web or SMTP server) using a TURN address. TURN is useful even when NAT is not present, to provide anonymity services.

15.2 Exit Strategy

From [13], 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.

It is expected that TURN will be useful indefinitely, to provide anonymity services. When used to facilitate NAT traversal, TURN does not iself provide an exit strategy. That is provided by the Interactive Connectivity Establishment (ICE) [14] mechanism. ICE allows two cooperating clients to interactively determine the best addresses to use when communicating. ICE uses TURN-allocated addresses as a last resort, only when no other means of connectivity exists. As a result, as NATs phase out, and as IPv6 is deployed, ICE will increasingly use other addresses (host local addresses). Therefore, clients will allocate TURN addresses, but not use them, and therefore, de-allocate them. Servers will see a decrease in usage. Once a provider sees that its TURN servers are not being used at all (that is, no media flows through them), they can simply remove them. ICE will operate without TURN-allocated addresses.

15.3 Brittleness Introduced by TURN

From [13], 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.

TURN introduces brittleness in a few ways. First, it adds another server element to any system, which adds another point of failure. TURN requires clients to demultiplex TURN packets and data based on hunting for a MAGIC-COOKIE in the TURN messages. It is possible (with extremely small probabilities) that this cookie could appear within a data stream, resulting in mis-classification. That might introduce errors into the data stream (they would appear as lost packets), and also result in loss of a binding. TURN relies on any NAT bindings existing for the duration of the bindings held by the TURN server. Neither the client nor the TURN server have a way of reliably determining this lifetime (STUN can provide a means, but it is heuristic in nature and not reliable). Therefore, if there is no activity on an address learned from TURN for some period, the address might become useless spontaneously.

TURN will result in potentially significant increases in packet latencies, and also increases in packet loss probabilities. That is because it introduces an intermediary on the path of a packet from point A to B, whose location is determined by application-layer processing, not underlying routing topologies. Therefore, a packet

sent from one user on a LAN to another on the same LAN may do a trip around the world before arriving. When combined with ICE, some of the most problematic cases are avoided (such as this example) by avoiding the usage of TURN addresses. However, when used, this problem will exist.

Note that TURN does not suffer from many of the points of brittleness introduced by STUN. TURN will work with all existing NAT types known at the time of writing, and for the forseeable future. TURN does not introduce any topological constraints. TURN does not rely on any heuristics for NAT type classification.

15.4 Requirements for a Long Term Solution

From [13]}, 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 TURN continues to validate our belief in the requirements outlined in Section 14.4 of STUN.

15.5 Issues with Existing NAPT Boxes

From [13], any UNSAF proposal must provide:

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

A number of NAT boxes are now being deployed into the market which try and provide "generic" ALG functionality. These generic ALGs hunt for IP addresses, either in text or binary form within a packet, and rewrite them if they match a binding. This will interfere with proper operation of any UNSAF mechanism, including TURN. However, if a NAT tries to modify a MAPPED-ADDRESS in a TURN Allocate Response, this will be detected by the client as an attack.

16. Example

In this example, a client is behind a NAT. The client has a private address of 10.0.1.1. The STUN server is on the public side of the NAT, and is listening for STUN relay requests on 192.0.2.3:8776. The public side of the NAT has an IP address of 192.0.2.1. The client is attempting to send a SIP INVITE to a peer, and wishes to allocate an IP address and port for inclusion in the SDP of the INVITE. Normally, TURN would be used in conjunction with ICE when applied to SIP. For simplicities sake, TURN is showed without ICE.

The client communicates with a SIP user agent on the public network. This user agent uses a 192.0.2.17:12734 for receipt of its RTP packets.

```
          Client                NAT             STUN Srvr        Called Pary
             |                   |                   |                   |
             |(1) Allocate       |                   |                   |
             |S=10.0.1.1:4334    |                   |                   |
             |D=192.0.2.3:8776   |                   |                   |
             |------------------>|                   |                   |
             |                   |                   |                   |
             |                   |                   |                   |
             |                   |                   |                   |
             |                   |(2) Allocate       |                   |
             |                   |S=192.0.2.1:63346  |                   |
             |                   |D=192.0.2.3:8776   |                   |
             |                   |------------------>|                   |
             |                   |                   |                   |
             |                   |                   |                   |
             |                   |                   |                   |
             |                   |(3) Error          |                   |
             |                   |S=192.0.2.3:8776   |                   |
             |                   |D=192.0.2.1:63346  |                   |
             |                   |<------------------|                   |
             |                   |                   |                   |
             |                   |                   |                   |
             |                   |                   |                   |
             |(4) Error          |                   |                   |
             |S=192.0.2.3:8776   |                   |                   |
             |D=10.0.1.1:4334    |                   |                   |
             |<------------------|                   |                   |
             |                   |                   |                   |
             |                   |                   |                   |
             |                   |                   |                   |
             |(5) Allocate       |                   |                   |
             |S=10.0.1.1:4334    |                   |                   |
             |D=192.0.2.3:8776   |                   |                   |
             |------------------>|                   |                   |
             |                   |                   |                   |
             |                   |                   |                   |
             |                   |                   |                   |
             |                   |(6) Allocate       |                   |
             |                   |S=192.0.2.1:63346  |                   |
             |                   |D=192.0.2.3:8776   |                   |
             |                   |------------------>|                   |
             |                   |                   |                   |
             |                   |(7) Response       |                   |
```

```
             |                   |RA=192.0.2.3:32766 |                   |
             |                   |MA=192.0.2.1:63346 |                   |
             |                   |S=192.0.2.3:8776   |                   |
             |                   |D=192.0.2.1:63346  |                   |
             |                   |<------------------|                   |
             |                   |                   |                   |
             |(8) Response       |                   |                   |
             |RA=192.0.2.3:32766 |                   |                   |
             |MA=192.0.2.1:63346 |                   |                   |
             |S=192.0.2.3:8776   |                   |                   |
             |D=10.0.1.1:4334    |                   |                   |
             |<------------------|                   |                   |
             |                   |                   |                   |
             |                   |                   |                   |
             |                   |                   |                   |
             |                   |                   |                   |
             |(9) INVITE         |                   |                   |
             |SDP=192.0.2.3:32766|                   |                   |
             |---------------------------------------------------------->|
             |                   |                   |                   |
             |                   |                   |                   |
             |                   |                   |                   |
             |                   |                   |                   |
             |(10) 200 OK        |                   |                   |
             |SDP=192.0.2.17:12734                   |                   |
             |<----------------------------------------------------------|
             |                   |                   |                   |
             |                   |                   |                   |
             |                   |                   |                   |
             |                   |                   |                   |
             |                   |                   |                   |
             |(11) ACK           |                   |                   |
             |---------------------------------------------------------->|
             |                   |                   |                   |
             |(12) Send          |                   |                   |
             |DATA=RTP           |                   |                   |
             |DA=192.0.2.17:12734|                   |                   |
             |S=10.0.1.1:4334    |                   |                   |
             |D=192.0.2.3:8776   |                   |                   |
             |------------------>|                   |                   |
             |                   |                   |                   |
             |                   |(13) Send          |                   |
             |                   |DATA=RTP           |                   |
             |                   |DA=192.0.2.17:12734|                   |
             |                   |S=192.0.2.1:63346  |                   |
             |                   |D=192.0.2.3:8776   |                   |
             |                   |------------------>|                   |
             |                   |                   |                   |
```

```
             |                   |                   |                   |
             |                   |                   |                   |
             |                   |                   |(14) RTP           |
             |                   |                   |S=192.0.2.3:32766  |
             |                   |                   |D=192.0.2.17:12734 |
             |                   |                   |------------------>|
             |                   |                   |                   |
             |                   |                   |                   |
             |                   |                   |                   |
             |                   |                   |Permission         |
             |                   |                   |Created            |
             |                   |                   |192.0.2.17         |
             |                   |                   |                   |
             |                   |                   |                   |
             |                   |                   |                   |
             |                   |                   |                   |
             |                   |                   |(15) RTP           |
             |                   |                   |S=192.0.2.17:12734 |
             |                   |                   |D=192.0.2.3:32766  |
             |                   |                   |<------------------|
             |                   |                   |                   |
             |                   |(16) DataInd       |                   |
             |                   |DATA=RTP           |                   |
             |                   |RA=192.0.2.17:12734|                   |
             |                   |S=192.0.2.3:8776   |                   |
             |                   |D=192.0.2.1:63346  |                   |
             |                   |<------------------|                   |
             |                   |                   |                   |
             |(17) DataInd       |                   |                   |
             |DATA=RTP           |                   |                   |
             |RA=192.0.2.17:12734|                   |                   |
             |S=192.0.2.3:8776   |                   |                   |
             |D=10.0.1.1:4334    |                   |                   |
             |<------------------|                   |                   |
             |                   |                   |                   |
             |                   |                   |                   |
             |(18) SetAct        |                   |                   |
             |DA=192.0.2.17:12734|                   |                   |
             |S=10.0.1.1:4334    |                   |                   |
             |D=192.0.2.3:8776   |                   |                   |
             |------------------>|                   |                   |
             |                   |                   |                   |
             |                   |                   |                   |
             |                   |(19) SetAct        |                   |
             |                   |DA=192.0.2.17:12734|                   |
             |                   |S=192.0.2.1:63346  |                   |
             |                   |D=192.0.2.3:8776   |                   |
             |                   |------------------>|                   |
```

```
             |                   |                   |                   |
             |                   |                   |                   |
             |                   |                   |                   |
             |                   |(20) Response      |                   |
             |                   |S=192.0.2.3:8776   |                   |
             |                   |D=192.0.2.1:63346  |                   |
             |                   |<------------------|                   |
             |                   |                   |                   |
             |                   |                   |                   |
             |                   |                   |                   |
             |(21) Response      |                   |                   |
             |S=192.0.2.3:8776   |                   |                   |
             |D=10.0.1.1:4334    |                   |                   |
             |<------------------|                   |                   |
             |                   |                   |                   |
             |                   |                   |                   |
             |                   |                   |                   |
             |                   |                   |                   |
             |                   |                   |                   |
             |                   |                   |                   |after
3s
             |                   |                   |                   |
             |                   |                   |                   |
             |                   |                   |                   |
             |                   |                   |                   |
             |                   |                   |(22) RTP           |
             |                   |                   |S=192.0.2.17:12734 |
             |                   |                   |D=192.0.2.3:32766  |
             |                   |                   |<------------------|
             |                   |                   |                   |
             |                   |                   |                   |
             |                   |                   |                   |
             |                   |(23) RTP           |                   |
             |                   |S=192.0.2.3:8776   |                   |
             |                   |D=192.0.2.1:63346  |                   |
             |                   |<------------------|                   |
             |                   |                   |                   |
             |                   |                   |                   |
             |                   |                   |                   |
             |(24) RTP           |                   |                   |
             |S=192.0.2.3:8776   |                   |                   |
             |D=10.0.1.1:4334    |                   |                   |
             |<------------------|                   |                   |
             |                   |                   |                   |
             |                   |                   |                   |
             |                   |                   |                   |
             |                   |                   |                   |
             |                   |                   |                   |
```

| | | |

```
         |                   |                    |                   |
```

Figure 12

The call flow is shown in Figure 12. The client allocates a port from the local operating system on its private interface, obtaining 4334. It then attempts to secure a port for RTP traffic. RTCP processing is not shown. The client sends an Allocate request (1) with a source address (denoted by S) of 10.0.1.1:4334 and a destination (denoted by D) of 192.0.2.3:8776. This passes through the NAT (2), which creates a mapping from the 192.0.2.1:63346 to the source IP address and port of the request, 10.0.1.1:4334. This request is received at the STUN server, which challenges it (3), requesting credentials. This response is passed to the client (4). The client retries the request, this time with credentials (5). This arrives at the server (6). The request is now authenticated. The server provides a UDP allocation, 192.0.2.3:32766, and places it into the RELAY-ADDRESS (denoted by RA) in the response (7). It also reflects the source IP address and port of the request into the MAPPED-ADDRESS (denoted by MA) in the response. This passes through the NAT to the client (8). The client now proceeds to perform a basic SIP call setup. In message 9, it includes the relay address into the SDP of its INVITE. The called party responds with a 200 OK, and includes its IP address - 192.0.2.17:12734. The exchange completes with an ACK (11).

Next, user A sends an RTP packet. Since the active destination has not been set, the client decides to use the Send indication. It does so, including the RTP packet as the contents of the DATA attribute. The DESTINATION-ADDRESS attribute (denoted by DA) is set to 192.0.2.17:12734, learned from the 200 OK. This is sent through the NAT (message 12) and arrives at the STUN server (message 13). The server extracts the data contents, and sends the packet towards DESTINATION-ADDRESS (message 14). Note how the source address and port in this packet is 192.0.2.3:32766, the allocated transport address given to the client. The act of sending the packet with Send causes the STUN server to install a permission for 192.0.2.17.

Indeed, the called party now sends an RTP packet toward the client (message 15). This arrives at the STUN server. Since a permission has been set for the IP address in the source of this packet, it is accepted. As no active destination is set, the STUN server encapsulates the contents of the packet in a Data Indication (message 16), and sends it towards the client. The REMOTE-ADDRESS attribute (denoted by RA) indicates the source of the packet - 192.0.2.17: 12734. This is forwarded through the NAT to the client (message 17).

The client decides to optimize the path for packets to and from 192.0.2.17:12734. So, it issues a Set Active Destination request (message 18) with a DESTINATION-ADDRESS of 192.0.2.17:12734. This passes through the NAT and arrives at the STUN server (message 19). This generates a successful response (message 20) which is passed to the client (message 21). At this point, the server and client are in the transitioning state. A little over 3 seconds later (by default), the state machines transition back to "Set". Until this point, packets from the called party would have been relayed back to the client in Data Indications. Now, the next RTP packet shows up at the STUN server (message 22). Since the source IP address and port match the active destination, the RTP packet is relayed towards the client without encapsulation (message 23 and 24).

17. Acknowledgements

The authors would like to thank Marc Petit-Huguenin for his comments and suggestions.

18. References

18.1 Normative References

[1] Rosenberg, J., "Simple Traversal of UDP Through Network Address Translators (NAT) (STUN)", draft-ietf-behave-rfc3489bis-02 (work in progress), July 2005.

[2] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

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

[4] Franks, J., Hallam-Baker, P., Hostetler, J., Lawrence, S., Leach, P., Luotonen, A., and L. Stewart, "HTTP Authentication: Basic and Digest Access Authentication", RFC 2617, June 1999.

18.2 Informative References

[5] Schulzrinne, H., Casner, S., Frederick, R., and V. Jacobson, "RTP: A Transport Protocol for Real-Time Applications", RFC 3550, July 2003.

[6] 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.

[7] Rosenberg, J. and H. Schulzrinne, "An Offer/Answer Model with Session Description Protocol (SDP)", RFC 3264, June 2002.

[8] Handley, M. and V. Jacobson, "SDP: Session Description Protocol", RFC 2327, April 1998.

[9] Schulzrinne, H., Rao, A., and R. Lanphier, "Real Time Streaming Protocol (RTSP)", RFC 2326, April 1998.

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

[11] Kent, S. and R. Atkinson, "IP Authentication Header", RFC 2402, November 1998.

[12] Kent, S. and R. Atkinson, "IP Encapsulating Security Payload (ESP)", RFC 2406, November 1998.

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

[14] Rosenberg, J., "Interactive Connectivity Establishment (ICE): A Methodology for Network Address Translator (NAT) Traversal for Offer/Answer Protocols", draft-ietf-mmusic-ice-06 (work in progress), October 2005.

[15] Audet, F. and C. Jennings, "NAT Behavioral Requirements for Unicast UDP", draft-ietf-behave-nat-udp-04 (work in progress), September 2005.

Authors' Addresses

Jonathan Rosenberg
Cisco Systems
600 Lanidex Plaza
Parsippany, NJ 07054
US

Phone: +1 973 952-5000
Email: jdrosen@cisco.com
URI: http://www.jdrosen.net

Rohan Mahy
Plantronics

Email: rohan@ekabal.com

Christian Huitema
Microsoft
One Microsoft Way
Redmond, WA 98052-6399
US

Email: huitema@microsoft.com

Intellectual Property Statement

Disclaimer of Validity

Copyright Statement

Acknowledgment

Funding for the RFC Editor function is currently provided by the Internet Society.