BEHAVE
Internet-Draft
Expires: August 5, 2006

J. Rosenberg
Cisco Systems
C. Huitema
Microsoft
R. Mahy
Plantronics
D. Wing
Cisco Systems
February 2006

Simple Traversal of UDP Through Network Address Translators (NAT) (STUN)

draft-ietf-behave-rfc3489bis-03

Status of this Memo

Copyright Notice

Abstract

Simple Traversal of UDP Through NATs (STUN) is a lightweight protocol that provides the ability for applications to determine the public IP addresses and ports allocated to them by the NAT and to keep NAT

bindings open. These addresses and ports can be placed into protocol payloads where a client needs to provide a publically routable IP address. STUN works with many existing NATs, and does not require any special behavior from them. As a result, it allows a wide variety of applications to work through existing NAT infrastructure.

Table of Contents

1. Applicability Statement

This protocol is not a cure-all for the problems associated with NAT. It does not enable incoming TCP connections through NAT. It allows incoming UDP packets through NAT, but only through a subset of existing NAT types. In particular, STUN does not enable incoming UDP packets through "symmetric NATs", which is

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

This type of NAT is common in large enterprises. STUN does not work when it is used to obtain an address to communicate with a peer which happens to be behind the same NAT. STUN does not work when the STUN server is not in a common shared address realm.

In order to work with such a NAT, a media relay such as TURN [3] is

required. All other types of NATs work without a media relay.

For a more complete discussion of the limitations of STUN, see Section 14.

2. Introduction

Network Address Translators (NATs), while providing many benefits, also come with many drawbacks. The most troublesome of those drawbacks is the fact that they break many existing IP applications, and make it difficult to deploy new ones. Guidelines have been developed [17] that describe how to build "NAT friendly" protocols, but many protocols simply cannot be constructed according to those guidelines. Examples of such protocols include almost all peer-to-peer protocols, such as multimedia communications, file sharing and games.

To combat this problem, Application Layer Gateways (ALGs) have been embedded in NATs. ALGs perform the application layer functions required for a particular protocol to traverse a NAT. Typically, this involves rewriting application layer messages to contain translated addresses, rather than the ones inserted by the sender of the message. ALGs have serious limitations, including scalability, reliability, and speed of deploying new applications.

Many existing proprietary protocols, such as those for online games (such as the games described in RFC3027 [18]) and Voice over IP, have developed tricks that allow them to operate through NATs without changing those NATs and without relying on ALG behavior in the NATs. This document takes some of those ideas and codifies them into an interoperable protocol that can meet the needs of many applications.

The protocol described here, Simple Traversal of UDP Through NAT (STUN), provides a toolkit of functions. These functions allow entities behind a NAT to learn the address bindings allocated by the NAT, to keep those bindings open, and communicate with other STUN-aware to validate connecivity. STUN requires no changes to NATs, and works with an arbitrary number of NATs in tandem between the application entity and the public Internet.

3. Terminology

In this document, the key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" are to be interpreted as described in BCP 14, RFC 2119 [1] and indicate requirement levels for compliant STUN implementations.

4. Definitions

STUN Client
: A STUN client (also just referred to as a client) is an entity that generates STUN requests.

STUN Server
: A STUN Server (also just referred to as a server) is an entity that receives STUN requests, and sends STUN responses.

Transport Address
: The combination of an IP address and (UDP or TCP) port.

Reflexive Transport Address
: A transport address learned by a client which identifies that client as seen by another host on an IP network, typically a STUN server. When there is an intervening NAT between the client and the other host, the reflexive address represents the binding allocated to the client on the public side of the NAT. Reflexive transport addresses are learned from the mapped address attribute (MAPPED-ADDRESS or XOR-MAPPED-ADDRESS) in STUN responses.

Mapped Address
: The source IP address and port of the STUN Binding Request packet received by the STUN server and inserted into the mapped address attribute (MAPPED-ADDRESS or XOR-MAPPED-ADDRESS) of the Binding Response message.

5. Overview of Operation

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

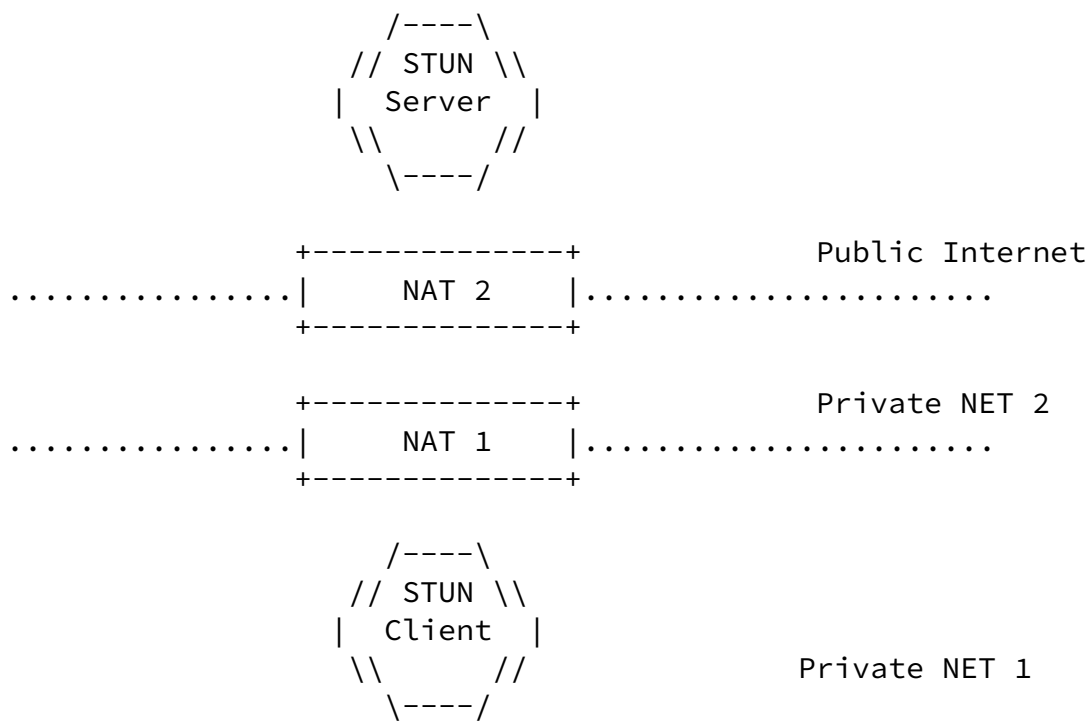


Figure 1: Typical STUN Server Configuration

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

STUN is a simple client-server protocol. Two types of messages are available -- request/response in which client sends a request to a server, and the server returns a response; and indications which can be initiated by the client or by the server and which do not elicit a response. There are two types of requests defined in this specification - Binding Requests, sent over UDP, and Shared Secret Requests, sent over TLS [6] over TCP. Shared Secret Requests ask the server to return a temporary username and password. This username

and password are used in a subsequent Binding Request and Binding

Response, for the purposes of authentication and message integrity.

Binding requests are used to determine the bindings allocated by NATs. The client sends a Binding Request to the server, over UDP. The server examines the source IP address and port of the request, and copies them into a response that is sent back to the client -- this is the 'mapped address'. There are attributes for providing message integrity and authentication.

The STUN client is typically embedded in an application which needs to obtain a public IP address and port that can be used to receive data. For example, it might need to obtain an IP address and port to receive Real Time Transport Protocol (RTP [14]) traffic. When the application starts, the STUN client within the application sends a STUN Shared Secret Request to its server, obtains a username and password, and then sends it a Binding Request. STUN servers can be discovered through DNS SRV records [4], and it is generally assumed that the client is configured with the domain to use to find the STUN server. Generally, this will be the domain of the provider of the service the application is using (such a provider is incented to deploy STUN servers in order to allow its customers to use its application through NAT). Of course, a client can determine the address or domain name of a STUN server through other means. A STUN server can even be embedded within an end system.

The STUN Binding Request is used to discover the public IP address and port mappings generated by the NAT. Binding Requests are sent to the STUN server using UDP. When a Binding Request arrives at the STUN server, it may have passed through one or more NATs between the STUN client and the STUN server. As a result, the source address of the request received by the server will be the mapped address created by the NAT closest to the server. The STUN server copies that source IP address and port into a STUN Binding Response, and sends it back to the source IP address and port of the STUN request. Every type of NAT will route that response so that it arrives at the STUN client.

When the STUN client receives the STUN Binding Response, it compares the IP address and port in the packet with the local IP address and port it bound to when the request was sent. If these do not match, the STUN client knows is behind one or more NATs. If the STUN server is publicly routable the IP address and port in the STUN Binding Response are also publicly routable, and can be used by any host on the public Internet to send packets to the application that sent the STUN request. An application need only listen on the IP address and port from which the STUN request was sent. Packets sent by a host on the public Internet to the public address and port learned by STUN will be received by the application, so long as conditions permit.

The conditions in which these packets will not be received by the client are described in Section 1.

It should be noted that the configuration in Figure 1 is not the only permissible configuration. The STUN server can be located anywhere, including within another client. The only requirement is that the STUN server is reachable by the client, and if the client is trying to obtain a publicly routable address, that the server reside on the public Internet.

6. STUN Message Structure

STUN messages are TLV (type-length-value) encoded using big endian (network ordered) binary. STUN messages are encoded using binary fields. All integer fields are carried in network byte order, that is, most significant byte (octet) first. This byte order is commonly known as big-endian. The transmission order is described in detail in Appendix B of RFC791 [2]. Unless otherwise noted, numeric constants are in decimal (base 10). All STUN messages start with a single STUN header followed by a STUN payload. The payload is a series of STUN attributes, the set of which depends on the message type. The STUN header contains a STUN message type, transaction ID, and length. The length indicates the total length of the STUN payload, not including the 20-byte header.

There are two categories of STUN message types: Requests and Indications.

Upon receiving a STUN request, a STUN server will send a STUN success response or a STUN error response. All STUN success responses MUST have a type whose value is 0x100 higher than their associated request, and all STUN error responses MUST have a type whose value is 0x110 higher than their associated request. Any newly defined STUN message types MUST use message type values 0x100 and 0x110 higher for their success and error responses, respectively. STUN Requests are sent reliably (Section 7.1). The transaction ID is used to correlate requests and responses.

An indication message can be sent from the client to the server, or from the server to the client. Indication messages are not sent reliably do not have an associated success response message type or associated error response message type. Indication messages can be sent by the STUN client to the server, or from the STUN server to the client. The transaction ID is used to distinguish indication messages.

All STUN messages consist of a 20 byte header:

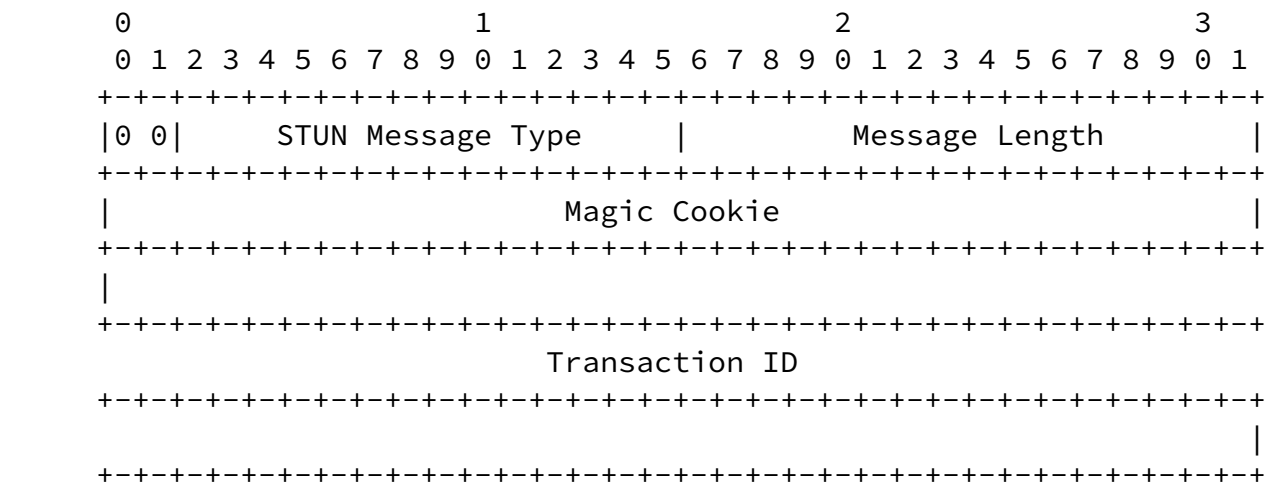


Figure 2: Format of STUN Message Header

The most significant two bits of every STUN message are 0b00. This, combined with the magic cookie, aids in differentiating STUN packets from other protocols when STUN is multiplexed with other protocols on the same port.

The STUN message types Binding Request, Response, and Error Response are defined in Section 8 and Section 9.1. The Shared Secret Request, Response, and Error Response are described in Section 12.5. Their values are enumerated in Section 15.

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

The magic cookie is a fixed value, 0x2112A442. In the previous version of this specification [13] this field was part of the transaction ID. This fixed value affords easy identification of a STUN message when STUN is multiplexed with other protocols on the same port, as is done for example in [12] and [15]. The magic cookie additionally indicates the STUN client is compliant with this specification. The magic cookie is present in all STUN messages -- requests, success responses and error responses.

The transaction ID is a 96 bit identifier. STUN transactions are identified by their unique 96-bit transaction ID. This transaction ID is chosen by the STUN client and MUST be unique for each new STUN transaction by that STUN client. Any two requests that are not bit-wise identical, and not sent to the same server from the same IP address and port, MUST have a different transaction ID. The transaction ID MUST be uniformly and randomly distributed between 0 and 2**96 - 1. The large range is needed because the transaction ID

serves as a form of randomization, helping to prevent replays of previously signed responses from the server.

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

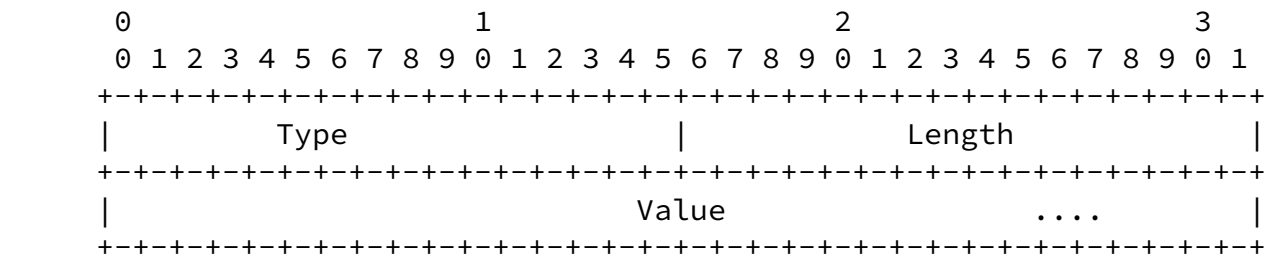


Figure 3: Format of STUN Attributes

The attribute types defined in this specification are in Section 11 .

7. STUN Transactions

STUN clients are allowed to pipeline STUN requests. That is, a STUN client MAY have multiple outstanding STUN requests with different transaction IDs and not wait for completion of a STUN request/ response exchange before sending another STUN request.

7.1. Request Transaction Reliability

When running STUN over UDP it is possible that the STUN request or its response might be dropped by the network. Reliability of STUN request message types is is accomplished through client retransmissions. Clients SHOULD retransmit the request starting with

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

When running STUN over TCP, TCP is responsible for ensuring delivery. The STUN application SHOULD NOT retransmit STUN requests when running over TCP.

For STUN requests, failure occurs if there is a transport failure of some sort (generally, due to fatal ICMP errors in UDP or connection

failures in TCP) or if retransmissions of the same STUN Request doesn't elicit a Response. If a failure occurs and the SRV query indicated other STUN servers are available, the client SHOULD create a new request, which is identical to the previous, but has a different transaction ID and MESSAGE INTEGRITY attribute (the HMAC will change because the transaction ID has changed). That request is sent to the next element in the list as specified by RFC2782.

The Indication message types are not sent reliably.

8. General Client Behavior

There are two classes of client behavior -- one for the request message types and another for the indication message types.

8.1. Request Message Types

This section applies to client behavior for the Request message types -- Binding Request and Shared Secret Request. For Request message types, the client must discover the STUN server's address and port, obtain a shared secret, formulate the Request, transmit the request reliability, process the Binding Response, and use the information in the Response.

8.1.1. Discovery

Unless stated otherwise by a STUN usage, DNS is used to discover the STUN server following these procedures.

The client will be configured with a domain name of the provider of the STUN servers. This domain name is resolved to an IP address and port using the SRV procedures specified in RFC2782 [4]. The mechanism for configuring the STUN client with the domain name to look up is not in scope of this document.

The DNS SRV service name is "stun". The protocol is "udp" for sending Binding Requests, or "tcp" for sending Shared Secret Requests. The procedures of RFC 2782 are followed to determine the server to contact. RFC 2782 spells out the details of how a set of SRV records are sorted and then tried. However, RFC2782 only states that the client should "try to connect to the (protocol, address, service)" without giving any details on what happens in the event of failure; those details for STUN are described in Section 8.1.3.

The default port for STUN requests is 3478, for both TCP and UDP. Administrators SHOULD use this port in their SRV records, but MAY use others. If no SRV records were found, the client performs an A or

AAAA record lookup of the domain name. The result will be a list of IP addresses, each of which can be contacted at the default port.

8.1.2. Obtaining a Shared Secret

As discussed in Section 13, there are several attacks possible on STUN systems. Many of these attacks are prevented through integrity protection of requests and responses. To provide that integrity, STUN makes use of a shared secret between client and server which is used as the keying material for the MESSAGE-INTEGRITY attribute in STUN messages. STUN allows for the shared secret to be obtained in any way (for example Kerberos [16] or ICE [12]). The shared secret MUST have at least 128 bits of randomness.

When a client is needs to send a Request or an Indication, it can do one of three things:

1. send the message without MESSAGE-INTEGRITY, if permitted by the

STUN usage.

2. use a short term credential, as determined by the STUN usage. In this case, the STUN Request or STUN Indication would contain the USERNAME and MESSAGE-INTEGRITY attributes. The message would not contain the NONCE attribute. The key for MESSAGE-INTEGRITY is the password.

3. use long term credential, as determined by STUN usage. In this case, the STUN request contains the USERNAME, REALM, and MESSAGE-INTEGRITY attributes. The request does not contain the NONCE attribute. The key for MESSAGE-INTEGRITY is MD5(unq(USERNAME-value) ":" unq(REALM-value) ":" password).

Based on the STUN usage, the server does one of four things:

1. The server processes the request and generates a response. If the request included the MESSAGE-INTEGRITY attribute, the server would also include MESSAGE-INTEGRITY in its response.

2. The server generates an error response indicating that MESSAGE-INTEGRITY with short-term or with long-term credentials are required (error 401). To indicate that short-term credentials are required, the REALM attribute MUST NOT be present in the error response. To indicate short-term credentials are required, the REALM attribute MOST be present in the error response.

3. The server generates an error response indicating that a NONCE attribute is required (error 435) or that the supplied NONCE attribute's value is stale (error 437).

4. The server generates an error response indicating that the short-term credentials are no longer valid (error 430). The client will have to obtain new short-term credentials appropriate to its STUN usage.

In all of the above error responses, the NONCE attribute MAY optionally be included in the error response, in which case the client MUST include that NONCE in the subsequent STUN transaction. The NONCE value is not stored by the STUN client; it is only valid for the subsequent STUN transaction and that transactions retransmissions.

STUN messages generated in order to obtain the shared secret are formulated like other messages by following Section 8.1.3.

8.1.3. Formulating the Request Message

The client follows the syntax rules defined in Section 6 and the transmission rules of Section 7. The message type of the MUST be a request type; "Binding Request" or "Shared Secret Request" are the two defined by this document.

The client creates a STUN message following the STUN message structure described in Section 6. The client SHOULD add a MESSAGE-INTEGRITY and USERNAME attribute to the Request message.

Once formulated, the client sends the Binding Request. Reliability is accomplished through client retransmissions, following the procedure in Section 7.1.

The client MAY send multiple requests on the connection, and it may pipeline requests (that is, it can have multiple requests outstanding at the same time). When using TCP the client SHOULD close the connection as soon as it has received the STUN Response.

8.1.4. Processing Responses

All responses, whether success responses or error responses, MUST first be authenticated by the client. Authentication is performed by first comparing the Transaction ID of the response to an oustanding request. If there is no match, the client MUST discard the response. Then the client SHOULD check the response for a MESSAGE-INTEGRITY attribute. If not present, and the client placed a MESSAGE-INTEGRITY attribute into the associated request, it MUST discard the response. If MESSAGE-INTEGRITY is present, the client computes the HMAC over the response as described in Section 11.8. The key to use depends on the shared secret mechanism. If the STUN Shared Secret Request was used, the key MUST be same as used to compute the MESSAGE-INTEGRITY

attribute in the request.

If the computed HMAC matches the one from the response, processing continues. The response can either be a Binding Response or Binding

Error Response.

If the response is an Error Response, the client checks the response code from the ERROR-CODE attribute of the response. For a 400 response code, the client SHOULD display the reason phrase to the user. For a 420 response code, the client SHOULD retry the request, this time omitting any attributes listed in the UNKNOWN-ATTRIBUTES attribute of the response. For a 430 response code, the client SHOULD obtain a new one-time username and password, and retry the Allocate Request with a new transaction. For 401 and 432 response codes, if the client had omitted the USERNAME or MESSAGE-INTEGRITY attribute as indicated by the error, it SHOULD try again with those attributes. A new one-time username and password is needed in that case. For a 431 response code, the client SHOULD alert the user, and MAY try the request again after obtaining a new username and password. For a 300 response code, the client SHOULD attempt a new transaction to the server indicated in the ALTERNATE-SERVER attribute. For a 500 response code, the client MAY wait several seconds and then retry the request with a new username and password. For a 600 response code, client MUST NOT retry the request and SHOULD display the reason phrase to the user. Unknown response codes between 400 and 499 are treated like a 400, unknown response codes between 500 and 599 are treated like a 500, and unknown response codes between 600 and 699 are treated like a 600. Any response between 100 and 299 MUST result in the cessation of request retransmissions, but otherwise is discarded.

Binding Responses containing unknown optional attributes (greater than 0x7FFF) MUST be ignored by the STUN client. Binding Responses containing unknown mandatory attributions (less than or equal to 0x7FFF) MUST be discarded and considered immediately as a failed transaction.

It is also possible for an IPv4 host to receive a XOR-MAPPED-ADDRESS or MAPPED-ADDRESS containing an IPv6 address, or for an IPv6 host to receive a XOR-MAPPED-ADDRESS or MAPPED-ADDRESS containing an IPv4 address. Clients MUST be prepared for this case.

8.1.5. Using the Mapped Address

This section applies to the Binding Response message type. The Binding Response message type always includes either the MAPPED-ADDRESS attribute or the XOR-MAPPED-ADDRESS attribute, depending on the presence of the magic cookie in the corresponding Binding

Request.

The mapped address present in the binding response can be used by clients to facilitate traversal of NATs for many applications. NAT traversal is problematic for applications which require a client to insert an IP address and port into a message, to which subsequent messages will be delivered by other entities in a network. Normally, the client would insert the IP address and port from a local interface into the message. However, if the client is behind a NAT, this local interface will be a private address. Clients within other address realms will not be able to send messages to that address.

An example of a such an application is SIP, which requires a client to include IP address and port information in several places, including the Session Description Protocol (SDP [19]) body carried by SIP. The IP address and port present in the SDP is used for receipt of media.

To use STUN as a technique for traversal of SIP and other protocols, when the client wishes to send a protocol message, it figures out the places in the protocol data unit where it is supposed to insert its own IP address along with a port. Instead of directly using a port allocated from a local interface, the client allocates a port from the local interface, and from that port, initiates the STUN procedures described above. The mapped address in the Binding Response (XOR-MAPPED-ADDRESS or MAPPED- ADDRESS) provides the client with an alternative IP address and port which it can then include in the protocol payload. This IP address and port may be within a different address family than the local interfaces used by the client. This is not an error condition. In such a case, the client would use the learned IP address and port as if the client was a host with an interface within that address family.

In the case of SIP, to populate the SDP appropriately, a client would generate two STUN Binding Request messages at the time a call is initiated or answered. One is used to obtain the IP address and port for RTP, and the other, for the Real Time Control Protocol (RTCP)[14]. The client might also need to use STUN to obtain IP addresses and ports for usage in other parts of the SIP message. The detailed usage of STUN to facilitate SIP NAT traversal is outside the scope of this specification.

As discussed above, the addresses learned by STUN may not be usable with all entities with whom a client might wish to communicate. The way in which this problem is handled depends on the application protocol. The ideal solution is for a protocol to allow a client to include a multiplicity of addresses and ports in the PDU. One of

those can be the address and port determined from STUN, and the

others can include addresses and ports learned from other techniques. The application protocol would then provide a means for dynamically detecting which one works. An example of such an an approach is Interactive Connectivity Establishment (ICE [12]).

8.2. Indication Message Types

This section applies to client behavior for the Indication message types.

8.2.1. Formulating the Indication Message

The client follows the syntax rules defined in Section 6 and the transmission rules of Section 7. The message type MUST be one of the Indication message types; none are defined by this document.

The client creates a STUN message following the STUN message structure described in Section 6. The client SHOULD add a MESSAGE-INTEGRITY and USERNAME attribute to the Request message.

Once formulated, the client sends the Indication message. Indication message types are not sent reliably, do not elicit a response from the server, and are not retransmitted.

The client MAY send multiple indications on the connection, and it may pipeline indication messages. When using TCP the client SHOULD close the TCP connection as soon as it has transmitted the indication message.

9. General Server Behavior

9.1. Request Message Types

The server behavior for receiving request message types is described in this section.

9.1.1. Receive Request Message

A STUN server MUST be prepared to receive Request and Indication

messages on the IP address and UDP or TCP port that will be discovered by the STUN client when the STUN client follows its discovery procedures described in Section 8.1.1. Depending on the usage, the STUN server will listen for incoming UDP STUN messages, incoming TCP STUN messages, or incoming TLS exchanges followed by TCP STUN messages. The usages describe how the STUN server determines the usage.

The server checks the request for a MESSAGE-INTEGRITY attribute. If not present, the server generates an error response with an ERROR-CODE attribute and a response code of 401. That error response MUST include a NONCE attribute, containing a nonce that the server wishes the client to reflect back in a subsequent request (and therefore include in the message integrity computation). The error response MUST include a REALM attribute, containing a realm from which the username and password are scoped [8].

If the MESSAGE-INTEGRITY attribute was present, the server checks for the existence of the REALM attribute. If the attribute is not present, the server MUST generate an error response. That error response MUST include an ERROR-CODE attribute with response code of 434. That error response MUST also include a NONCE and a REALM attribute.

If the REALM attribute was present, the server checks for the existence of the NONCE attribute. If the NONCE attribute is not present, the server MUST generate an error response. That error response MUST include an ERROR-CODE attribute with a response code of 435. That error response MUST include a NONCE attribute and a REALM attribute.

If the NONCE attribute was present, the server checks for the existence of the USERNAME attribute. If it was not present, the server MUST generate an error response. The error response MUST include an ERROR-CODE attribute with a response code of 432. It MUST include a NONCE attribute and a REALM attribute.

If the USERNAME attribute was present, the server computes the HMAC over the request as described in Section 11.8. The key is computed as MD5(unq(USERNAME-value) ":" unq(REALM-value) ":" password), where the password is the password associated with the username and realm

provided in the request. If the server does not have a record for that username within that realm, the server generates an error response. That error response MUST include an ERROR-CODE attribute with a response code of 436. That error response MUST include a NONCE attribute and a REALM attribute.

> This format for the key was chosen so as to enable a common authentication database for SIP and STUN, as it is expected that credentials are usually stored in their hashed forms.

If the computed HMAC differs from the one from the MESSAGE-INTEGRITY attribute in the request, the server MUST generate an error response with an ERROR-CODE attribute with a response code of 431. This response MUST include a NONCE attribute and a REALM attribute.

If the computed HMAC doesn't differ from the one in the request, but the nonce is stale, the server MUST generate an error response. That error response MUST include an ERROR-CODE attribute with response code 430. That error response MUST include a NONCE attribute and a REALM attribute.

The server MUST check for any mantadory attributes in the request (values less than or equal to 0x7fff) which it does not understand. If it encounters any, the server MUST generate a Binding Error Response, and it MUST include an ERROR-CODE attribute with a 420 response code. Any attributes that are known, but are not supposed to be present in a message (MAPPED-ADDRESS in a request, for example) MUST be ignored.

9.1.2. Constructing the Response

To construct the STUN Response the STUN server follows the message structure described in Section 6. The server then copies the Transaction ID from the Request to the Response. If the STUN response is a success response, the STUN server adds 0x100 to the Message Type; if a failure response the STUN server adds 0x110 to the Message Type.

Depending in the Request message type and the message attributes of the request, the response is constructed; see Figure 4.

9.1.3. Sending the Response

All Response messages are sent to the IP address and port the associated Binding Request came from, and sent from the IP address and port the Binding Request was sent to.

9.2. Indication Message Types

Indication messages cause the server to change its state. Indication message types to not cause the server to send a response message.

Indication message types are defined in other documents, for example in [3].

10. Short-Term Passwords

Short-term passwords are useful to provide authentication and integrity protection to STUN Request and STUN Response messages. Short-term passwords are useful when there is no long-term relationship with a STUN server and thus no long-term password is shared between the STUN client and STUN server. Even if there is a

long-term password, the issuance of a short-term password is useful to prevent dictionary attacks.

Short-term passwords can be used multiple times for as long as a usage allows the same short-term password to be used. The duration of validity is determined by usage.

11. STUN Attributes

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

In order to align attributes on word boundaries, the length of the all message attributes values MUST be 0 or a multiple of 4 bytes. Extensions to this specification MUST also follow this requirement.

The values of the message attributes are enumerated in Section 15.

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

Attribute	Request	Response	Error Response
MAPPED-ADDRESS	-	C	-
USERNAME	O	-	-
PASSWORD	-	-	-
MESSAGE-INTEGRITY	O	O	O
ERROR-CODE	-	-	M
ALTERNATE-SERVER	-	-	C
REALM	C	C	C
NONCE	C	-	C
UNKNOWN-ATTRIBUTES	-	-	C
XOR-MAPPED-ADDRESS	-	M	-
XOR-ONLY	O	-	-
SERVER	-	O	O
BINDING-LIFETIME	-	O	-

Figure 4: Mandatory Attributes and Message Types

11.1. MAPPED-ADDRESS

The MAPPED-ADDRESS attribute indicates the mapped IP address and port. It consists of an eight bit address family, and a sixteen bit port, followed by a fixed length value representing the IP address. If the address family is IPv4, the address is 32 bits. If the address family is IPv6, the address is 128 bits.

For backwards compatibility with RFC3489-compliant STUN clients, if the magic cookie was not present in the associated Binding Request,

this attribute MUST be present in the associated response.

Discussion: Some NATs rewrite the 32-bit binary payloads containing the NAT's public IP address, such as STUN's MAPPED-ADDRESS attribute. Such behavior interferes with the operation of STUN and also causes failure of STUN's message integrity checking.

Presence of the magic cookie in the STUN Request indicates the client is compatible with this specification and is capable of processing XOR-MAPPED-ADDRESS.

The format of the MAPPED-ADDRESS attribute is:

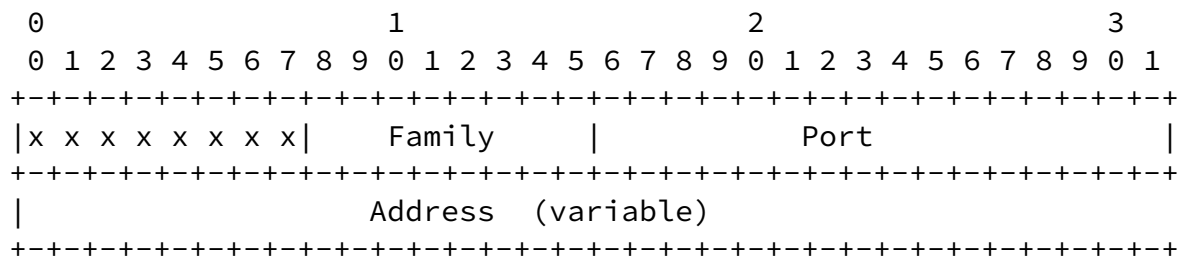


Figure 5: Format of MAPPED-ADDRESS attribute

The address family can take on the following values:
0x01: IPv4
0x02: IPv6

The port is a network byte ordered representation of the port the Binding Request arrived from.

The first 8 bits of the MAPPED-ADDRESS are ignored for the purposes of aligning parameters on natural boundaries.

11.2. RESPONSE-ADDRESS

The RESPONSE-ADDRESS attribute indicates where the response to a

Binding Request should be sent. Its syntax is identical to MAPPED-ADDRESS.

This attribute is not used by any STUN usages defined in this document except for backwards compatibility with RFC3489 clients when

using the Binding Discovery usage (Section 12.2). Section 12.2.8 describes when this attribute must be included in a binding response.

Issue: should this attribute be made specific to Binding Discovery or moved to another document entirely.

11.3. CHANGED-ADDRESS

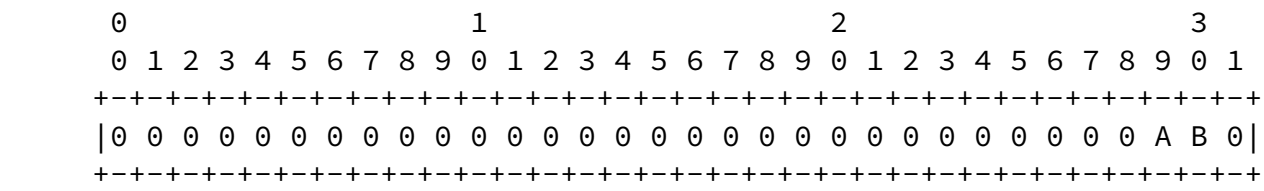
The CHANGED-ADDRESS attribute indicates the IP address and port where responses would have been sent from if the "change IP" and "change port" flags had been set in the CHANGE-REQUEST attribute of the Binding Request. Its syntax is identical to MAPPED-ADDRESS.

This attribute is not used by any STUN usages defined in this document except for backwards compatibility with RFC3489 clients when using the Binding Discovery usage (Section 12.2). Section 12.2.8 describes when this attribute must be included in a binding response.

Issue: should this attribute be made specific to Binding Discovery or moved to another document entirely.

11.4. CHANGE-REQUEST

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



The meaning of the flags are:

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

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

This attribute is not used by any STUN usages defined in this document.

Issue: should this attribute be made specific to Binding Discovery or moved to another document entirely.

11.5. SOURCE-ADDRESS

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

This attribute is not used by any STUN usages defined in this document except for backwards compatibility with RFC3489 clients when using the Binding Discovery usage (Section 12.2). Section 12.2.8 describes when this attribute must be included in a binding response.

Issue: should this attribute be made specific to Binding Discovery or moved to another document entirely.

11.6. USERNAME

The USERNAME attribute is used for message integrity. It identifies the shared secret used in the message integrity check. The USERNAME is always present in a Shared Secret Response, along with the PASSWORD. When message integrity is used with Binding Request messages, the USERNAME attribute MUST be included.

The value of USERNAME is a variable length opaque value.

11.7. PASSWORD

If the message type is Shared Secret Response it MUST include the PASSWORD attribute.

The value of PASSWORD is a variable length opaque value. The password returned in the Shared Secret Response is used as the HMAC in the MESSAGE-INTEGRITY attribute of a subsequent STUN transaction.

11.8. MESSAGE-INTEGRITY

The MESSAGE-INTEGRITY attribute contains an HMAC-SHA1 [9] of the STUN message. The MESSAGE-INTEGRITY attribute can be present in any STUN

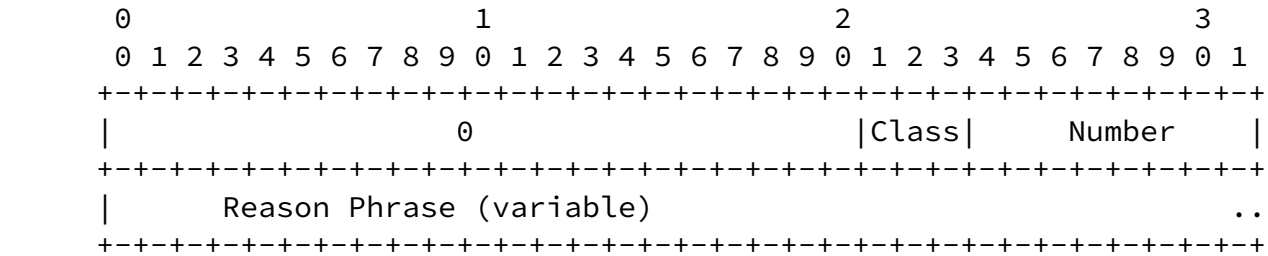
message type. Since it uses the SHA1 hash, the HMAC will be 20 bytes. The text used as input to HMAC is the STUN message, including the header, up to and including the attribute preceding the MESSAGE-INTEGRITY attribute. That text is then padded with zeroes so as to be a multiple of 64 bytes. As a result, the MESSAGE-INTEGRITY attribute is the last attribute in any STUN message. However, STUN clients MUST be able to successfully parse and process STUN messages which have additional attributes after the MESSAGE-INTEGRITY attribute. STUN clients that are compliant with this specification SHOULD ignore attributes that are after the MESSAGE-INTEGRITY attribute.

The key used as input to HMAC depends on the STUN usage and the shared secret mechanism.

11.9. ERROR-CODE

The ERROR-CODE attribute is present in the Binding Error Response and Shared Secret Error Response. It is a numeric value in the range of 100 to 699 plus a textual reason phrase encoded in UTF-8, and is consistent in its code assignments and semantics with SIP [10] and HTTP [11]. The reason phrase is meant for user consumption, and can be anything appropriate for the response code. The length of the reason phrase MUST be a multiple of 4 (measured in bytes), accomplished by added spaces to the end of the text, if necessary. Recommended reason phrases for the defined response codes are presented below.

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



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

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

300 (Try Alternate): The client should contact an alternate server for this request.

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

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

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

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

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

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

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

434 (Missing Realm): The REALM attribute was not present in the request.

435 (Missing Nonce): The NONCE attribute was not present in the

request.

436 (Unknown Username): The USERNAME supplied in the Request is not known or is not known in the given REALM.

437 (Stale Nonce): The NONCE attribute was present in the request but wasn't valid.

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

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

Issue: Do 300/500/600 mean that other STUN servers returned in the same SRV lookup should be retried / not retried? With same SRV Priority?

11.10. REFLECTED-FROM

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

This attribute is not used by any STUN usages defined in this document.

Issue: should this attribute be made specific to Binding Discovery or moved to another document entirely.

11.11. ALTERNATE-SERVER

The alternate server represents an alternate IP address and port for a different TURN server to try. It is encoded in the same way as MAPPED-ADDRESS.

11.12. REALM

The REALM attribute is present in Requests and Responses. It contains text which meets the grammar for "realm" as described in RFC3261 [10], and will thus contain a quoted string (including the quotes).

Presence of the REALM attribute indicates that long-term credentials are used for the values of the USERNAME, PASSWORD, and MESSAGE-INTEGRITY attributes.

11.13. NONCE

The NONCE attribute is present in Requests and in Error responses. It contains a sequence of qdtext or quoted-pair, which are defined in RFC3261 [10].

11.14. UNKNOWN-ATTRIBUTES

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

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

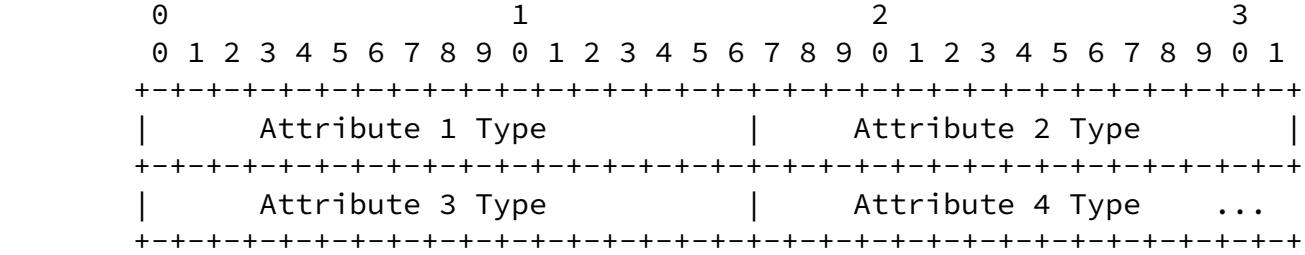


Figure 9: Format of UNKNOWN-ATTRIBUTES attribute

11.15. XOR-MAPPED-ADDRESS

The XOR-MAPPED-ADDRESS attribute is only present in Binding Responses. It provides the same information that would present in the MAPPED-ADDRESS attribute but because the NAT's public IP address is obfuscated through the XOR function, STUN messages are able to pass through NATs which would otherwise interfere with STUN. See the discussion in Section 11.1.

This attribute MUST always be present in a Binding Response.

> Note: Version -02 of this Internet Draft used 0x8020 for this attribute, which was in the Optional range of attributes. This attribute has been moved back to 0x0020 as a Mandatory attribute. [This paragraph should be removed prior to publication as an RFC.]

The format of the XOR-MAPPED-ADDRESS is:

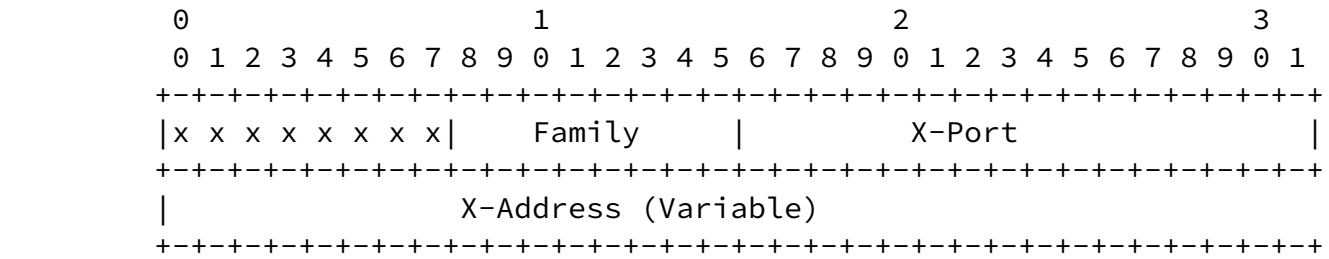


Figure 10: Format of XOR-MAPPED-ADDRESS Attribute

The Family represents the IP address family, and is encoded identically to the Family in MAPPED-ADDRESS.

X-Port is the mapped port, exclusive or'd with most significant 16 bits of the magic cookie. If the IP address family is IPv4, X-Address is mapped IP address exclusive or'd with the magic cookie. If the IP address family is IPv6, the X-Address is the mapped IP address exclusively or'ed with the magic cookie and the 96-bit transaction ID.

> Issue: The motivation for XORing the IP address is clear. Is there a motivation for XORing the port?

For example, using the "^" character to indicate exclusive or, if the IP address is 192.168.1.1 (0xc0a80101) and the port is 5555 (0x15B3), the X-Port would be 0x15B3 ^ 0x2112 = 0x34A1, and the X-Address would be 0xc0a80101 ^ 0x2112A442 = 0xe1baa543.

11.16. SERVER

The server attribute contains a textual description of the software being used by the server, including manufacturer and version number. The attribute has no impact on operation of the protocol, and serves only as a tool for diagnostic and debugging purposes. The length of the server attribute MUST be a multiple of 4 (measured in bytes), accomplished by added spaces to the end of the text, if necessary. The value of SERVER is variable length.

11.17. ALTERNATE-SERVER

The alternate server represents an alternate IP address and port for a different STUN server to try. It is encoded in the same way as MAPPED-ADDRESS.

This attribute is MUST only appear in an Error Response. This attribute MUST only appear when using the TURN usage.

11.18. BINDING-LIFETIME

The binding lifetime indicates the number of seconds the NAT binding will be valid. This attribute MUST only be present in Response messages. This attribute MUST NOT be present unless the STUN server is aware of the minimum binding lifetime of all NATs on the path between the STUN client and the STUN server.

12. STUN Usages

STUN is a simple request/response protocol that provides a useful capability in several situations. In this section, different usages of STUN are described. Each usages may differ in how STUN servers are discovered, the message types, and the message attributes that are supported.

This specification defines the STUN usages for binding discovery (Section 12.2), connectivity check (Section 12.3), NAT keepalives (Section 12.4) and short-term password (Section 12.5).

New STUN usages may be defined by other standards-track documents. New STUN usages MUST describe their applicability, client discovery of the STUN server, how the server determines the usage, new message types (requests or indications), new message attributes, new error response codes, and new client and server procedures.

12.1. Defined STUN Usages

12.2. Binding Discovery

The previous version of this specification, RFC3489 [13], described only this binding discovery usage.

12.2.1. Applicability

Binding discovery is useful to learn reflexive addresses from servers on the network. That is, it is used to determine your dynamically-bound 'public' IP address and UDP port that is assigned by a NAT between a STUN client and a STUN server. This usage is used with ICE [12].

When short-term passwords are used with binding discovery, the username and password are valid for subsequent transactions for nine (9) minutes.

12.2.2. Client Discovery of Server

The general client discovery of server behavior is sufficient for this usage.

12.2.3. Server Determination of Usage

The general binding server behavior is sufficient for this usage.

12.2.4. New Requests or Indications

This usage does not define any new message types.

12.2.5. New Attributes

This usage does not define any new message attributes.

12.2.6. New Error Response Codes

This usage does not define any new error response codes.

12.2.7. Client Procedures

This usage does not define any new client procedures.

12.2.8. Server Procedures

In this usage, the short-term password is valid for 30 seconds after its initial assignment.

For backwards compatibility with RFC3489-compliant STUN servers, if the STUN server receives a Binding request without the magic cookie, the STUN server MUST include the following attributes in the Binding response; otherwise these attribute MUST NOT be included:

MAPPED-ADDRESS
SOURCE-ADDRESS

Likewise if the STUN server receives a Binding Request containing the CHANGE-REQUEST attribute without the magic cookie, the STUN server MUST include the CHANGED-ADDRESS attribute in its Binding Response.

12.2.9. Security Considerations for Binding Discovery

> Issue: Currently, the security considerations applies to all the various usages. Split it up to talk about each one? Create subsections talking about each usage?

12.3. Connectivity Check

12.3.1. Applicability

This STUN usage primarily provides a connectivity check to a peer discovered through rendezvous protocols and additionally allows learning reflexive address discovery to the peer.

The username and password exchanged in the rendezvous protocol is valid for the duration of the connection being checked.

12.3.2. Client Discovery of Server

The client does not follow the general procedure in Section 8.1.1. Instead, the client discovers the STUN server's IP address and port through a rendezvous protocol such as Session Description Protocol (SDP [19]). An example of such a discovery technique is ICE [12].

12.3.3. Server Determination of Usage

The server is aware of this usage because it signalsed this port through the rendezvous protocol.

When operating in this usage, the STUN server is listening on an ephemeral port rather than the IANA-assigned STUN port. The server is typically multiplexing two protocols on this port, one protocol is STUN and the other protocol is the peer-to-peer protocol using that same port. When used with ICE, the two protocols multiplexed on the same port are STUN and RTP [14].

12.3.4. New Requests or Indications

This usage does not define any new message types.

12.3.5. New Attributes

This usage does not define any new message attributes.

12.3.6. New Error Response Codes

This usage does not define any new error response codes.

12.3.7. Client Procedures

This usage does not define any new client procedures.

12.3.8. Server Procedures

In this usage, the short-term password is valid as long as the UDP port is listening for STUN packets. For example when used with ICE, the short-term password would be valid as long as the RTP session (which multiplexes STUN and RTP) is active.

12.3.9. Security Considerations for Connectivity Check

The username and password, which are used for STUN's message integrity, are exchanged in the rendezvous protocol. Failure to encrypt and integrity protect the rendezvous protocol is equivalent in risk to using STUN without message integrity.

12.4. NAT Keepalives

12.4.1. Applicability

This usage is useful in two cases: keeping a NAT binding open in a client connection to a server and detecting server failure and NAT reboots.

The username and password used for STUN integrity can be used for 24 hours.

Issue: do we need message integrity for keepalives when doing STUN and SIP on the same port? Do we need message integrity for keepalives when doing STUN and RTP on the same port (recvonly, inactive)

If yes, do we continue using same STUN username/password forever (days?)

12.4.2. Client Discovery of Server

In this usage, the STUN server and the application protocol are using

the same fixed port. While the multiplexing of two applications on the same port is similar to the connectivity check (Section 12.3) usage, this usage is differs as the server's port is fixed and the server's port isn't communicated using a rendezvous protocol.

12.4.3. Server Determination of Usage

The server multiplexes both STUN and its application protocol on the same port. The server knows it is has this usage because the URI that gets resolved to this port indicates the server supports this multiplexing.

12.4.4. New Requests or Indications

This usage does not define any new message types.

12.4.5. New Attributes

This usage does not define any new message attributes.

12.4.6. New Error Response Codes

This usage does not define any new error response codes.

12.4.7. Client Procedures

If the STUN Response indicates the client's mapped address has changed from the client's expected mapped address, the client SHOULD inform other applications of its new mapped address. For example, a SIP client should send a new registration message indicating the new mapped address.

12.4.8. Server Procedures

In this usage no authentication is used so there is no duration of the short-term password.

12.4.9. Security Considerations for NAT Keepalives

Issue: Currently, the security considerations applies to all the various usages. Split it up to talk about each one? Create

subsections talking about each usage?

12.5. Short-Term Password

In order to ensure interoperability, this usage describes a TLS-based mechanism to obtain a short-term username and short-term password.

12.5.1. Applicability

To thwart some on-path attacks described in Section 13, it is necessary for the STUN client and STUN server to integrity protect the information they exchange over UDP. In the absence of a long-term secret (password) that is shared between them, a short-term password can be obtained using the usage described in this section.

The username and password returned in the STUN Shared Secret Response are valid for use in subsequent STUN transactions for nine (9) minutes with any hosts that have the same SRV Priority value as discovered via Section 12.5.2. The username and password obtained

with this usage are used as the USERNAME and as the HMAC for the MESSAGE-ID in a subsequent STUN message, respectively.

The duration of validity of the username and password obtained via this usage depends on the usage of the subsequent STUN messages that are protected with that username and password.

12.5.2. Client Discovery of Server

The client follows the procedures in Section 8.1.1, except the SRV protocol is TCP rather than UDP and the service name "stun-tls".

For example a client would look up "_stun-tls._tcp.example.com" in DNS.

12.5.3. Server Determination of Usage

The server advertises this port in the DNS as capable of receiving TLS-protected STUN messages for this usage. The server MAY also advertise this same port in DNS for other TCP usages if the server is capable of multiplexing those different usages. For example, the server could advertise

12.5.4. New Requests or Indications

The message type Shared Secret Request and its associated Shared Secret Response and Shared Secret Error Response are defined in this section. Their values are enumerated in Section 15.

The following figure indicates which attributes are present in the Shared Secret Request, Response, and Error Response. An M indicates that inclusion of the attribute in the message is mandatory, O means its optional, C means it's conditional based on some other aspect of the message, and N/A means that the attribute is not applicable to that message type. Attributes not listed are not applicable to Shared Secret Request, Response, or Error Response.

Attribute	Shared Secret Request	Shared Secret Response	Shared Secret Error Response
USERNAME	-	M	-
PASSWORD	-	M	-
ERROR-CODE	-	-	M
UNKNOWN-ATTRIBUTES	-	-	C
SERVER	-	O	O
REALM	C	-	C

Note: As this usage requires running over TLS, MESSAGE-INTEGRITY isn't necessary.

12.5.5. New Attributes

No new attributes are defined by this usage.

12.5.6. New Error Response Codes

This usage does not define any new error response codes.

12.5.7. Client Procedures

The client opens up the connection to that address and port, and immediately begins TLS negotiation[6]. The client MUST verify the

identity of the server. To do that, it follows the identification procedures defined in Section 3.1 of RFC2818 [5]. Those procedures assume the client is dereferencing a URI. For purposes of usage with this specification, the client treats the domain name or IP address used in Section 9.1 as the host portion of the URI that has been dereferenced. Once the connection is opened, the client sends a Shared Secret request. This request has no attributes, just the header. The transaction ID in the header MUST meet the requirements outlined for the transaction ID in a binding request, described in Section 9.3 below.

If the response was a Shared Secret Error Response, the client checks the response code in the ERROR-CODE attribute. If the response was a Shared Secret Response, it will contain a short lived username and password, encoded in the USERNAME and PASSWORD attributes, respectively.

12.5.8. Server Procedures

After a client has established a TLS session, the server should expect a STUN message containing a Shared Secret Request. The server will generates a response, which can either be a Shared Secret Response or a Shared Secret Error Response.

12.5.9. Security Considerations for Short-Term Password

> Issue: Currently, the security considerations applies to all the various usages. Split it up to talk about each one? Create subsections talking about each usage?

13. Security Considerations

Issue: This section has not been revised to properly consider the attacks on each of STUN's different usages. This needs to be done.

13.1. Attacks on STUN

Generally speaking, attacks on STUN can be classified into denial of

service attacks and eavesdropping attacks. Denial of service attacks can be launched against a STUN server itself, or against other elements using the STUN protocol. STUN servers create state through the Shared Secret Request mechanism. To prevent being swamped with traffic, a STUN server SHOULD limit the number of simultaneous TLS connections it will hold open by dropping an existing connection when a new connection request arrives (based on an Least Recently Used (LRU) policy, for example). Similarly, if the server is storing short-term passwords it SHOULD limit the number of shared secrets it will store. The attacks of greater interest are those in which the STUN server and client are used to launch denial of service (DoS) attacks against other entities, including the client itself. Many of the attacks require the attacker to generate a response to a legitimate STUN request, in order to provide the client with a faked XOR-MAPPED-ADDRESS or MAPPED-ADDRESS. In the sections below, we refer to either the XOR-MAPPED-ADDRESS or MAPPED-ADDRESS as just the mapped address (note the lower case). The attacks that can be launched using such a technique include:

13.1.1. Attack I: DDoS Against a Target

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

13.1.2. Attack II: Silencing a Client

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

interesting for the attacker. It impacts a single client, which is

frequently not the desired target. Moreover, any attacker that can mount the attack could also deny service to the client by other means, such as preventing the client from receiving any response from the STUN server, or even a DHCP server.

13.1.3. Attack III: Assuming the Identity of a Client

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

13.1.4. Attack IV: Eavesdropping

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

13.2. Launching the Attacks

It is important to note that attacks of this nature (injecting responses with fake mapped addresses) require that the attacker be capable of eavesdropping requests sent from the client to the server (or to act as a man in the middle for such attacks). This is because STUN requests contain a transaction identifier, selected by the client, which is random with 96 bits of entropy. The server echoes this value in the response, and the client ignores any responses that don't have a matching transaction ID. Therefore, in order for an attacker to provide a faked response that is accepted by the client, the attacker needs to know the transaction ID of the request. The large amount of randomness, combined with the need to know when the client sends a request and the IP address and UDP ports used for that request, precludes attacks that involve guessing the transaction ID.

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

13.2.1. Approach I: Compromise a Legitimate STUN Server

In this attack, the attacker compromises a legitimate STUN server through a virus or Trojan horse. Presumably, this would allow the attacker to take over the STUN server, and control the types of responses it generates. Compromise of a STUN server can also lead to discovery of open ports. Knowledge of an open port creates an opportunity for DoS attacks on those ports (or DDoS attacks if the traversed NAT is a full cone NAT). Discovering open ports is already fairly trivial using port probing, so this does not represent a major threat.

13.2.2. Approach II: DNS Attacks

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

13.2.3. Approach III: Rogue Router or NAT

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

> Todo: RESPONSE-ADDRESS has been removed from this version of the specification. Reword or remove above paragraph accordingly.

If the attacker is on a private network (that is, there are NATs between it and the STUN server), the attacker will not be able to

force the server to generate arbitrary mapped addresses in responses. They will only be able force the STUN server to generate mapped addresses which route to the private network. This is because the NAT between the attacker and the STUN server will rewrite the source address of the STUN request, mapping it to a public address that routes to the private network. Because of this, the attacker can only force the server to generate faked mapped addresses that route to the private network. Unfortunately, it is possible that a low quality NAT would be willing to map an allocated public address to another public address (as opposed to an internal private address), in which case the attacker could forge the source address in a STUN request to be an arbitrary public address. This kind of behavior from NATs does appear to be rare.

13.2.4. Approach IV: Man in the Middle

As an alternative to approach III (Section 13.2.3), if the attacker can place an element on the path from the client to the server, the element can act as a man-in-the-middle. In that case, it can intercept a STUN request, and generate a STUN response directly with any desired value of the mapped address field. Alternatively, it can forward the STUN request to the server (after potential modification), receive the response, and forward it to the client. When forwarding the request and response, this attack is subject to the same limitations on the mapped address described in Approach III (Section 13.2.3).

13.2.5. Approach V: Response Injection Plus DoS

In this approach, the attacker does not need to be a MitM (as in approaches III and IV). Rather, it only needs to be able to eavesdrop onto a network segment that carries STUN requests. This is easily done in multiple access networks such as ethernet or unprotected 802.11. To inject the fake response, the attacker listens on the network for a STUN request. When it sees one, it simultaneously launches a DoS attack on the STUN server, and generates its own STUN response with the desired mapped address value. The STUN response generated by the attacker will reach the client, and the DoS attack against the server is aimed at preventing

the legitimate response from the server from reaching the client. Arguably, the attacker can do without the DoS attack on the server, so long as the faked response beats the real response back to the client, and the client uses the first response, and ignores the second (even though it's different).

13.2.6. Approach VI: Duplication

This approach is similar to approach V (Section 13.2.5). The

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

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

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

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

Fortunately, this approach is subject to the same limitations documented in Approach III (Section 13.2.3), which limit the range of mapped addresses the attacker can cause the STUN server to generate.

13.3. Countermeasures

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

First off, it is RECOMMENDED that networks with STUN clients implement ingress source filtering [7]. This is particularly important for the NATs themselves. As Section 13.2.3 explains, NATs

which do not perform this check can be used as "reflectors" in DDoS attacks. Most NATs do perform this check as a default mode of operation. We strongly advise people that purchase NATs to ensure that this capability is present and enabled.

Secondly, it is RECOMMENDED that STUN servers be run on hosts dedicated to STUN, with all UDP and TCP ports disabled except for the STUN ports. This is to prevent viruses and Trojan horses from infecting STUN servers, in order to prevent their compromise. This helps mitigate Approach I (Section 13.2.1).

Thirdly, to prevent the DNS attack of Section 13.2.2, Section 8.1.2 recommends that the client verify the credentials provided by the server with the name used in the DNS lookup.

Finally, all of the attacks above rely on the client taking the mapped address it learned from STUN, and using it in application layer protocols. If encryption and message integrity are provided within those protocols, the eavesdropping and identity assumption attacks can be prevented. As such, applications that make use of STUN addresses in application protocols SHOULD use integrity and encryption, even if a SHOULD level strength is not specified for that protocol. For example, multimedia applications using STUN addresses to receive RTP traffic would use secure RTP [20].

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

Approaches IV (Section 13.2.4), when generating the response locally, and V (Section 13.2.5) require an attacker to generate a faked response. A faked response must match the 96-bit transaction ID of the request. The attack further prevented by using the message integrity mechanism provided in STUN, described in Section 11.8.

Approaches III (Section 13.2.3), IV (Section 13.2.4), when using the relaying technique, and VI (Section 13.2.6), however, are not preventable through server signatures. All three approaches are most potent when the attacker can modify the request, inserting a RESPONSE-ADDRESS that routes to the client. Fortunately, such modifications are preventable using the message integrity techniques described in Section 11.8. However, these three approaches are still functional when the attacker modifies nothing but the source address of the STUN request. Sadly, this is the one thing that cannot be protected through cryptographic means, as this is the change that STUN itself is seeking to detect and report. It is therefore an inherent weakness in NAT, and not fixable in STUN.

13.4. Residual Threats

None of the countermeasures listed above can prevent the attacks described in Section 13.2.3 if the attacker is in the appropriate network paths. Specifically, consider the case in which the attacker wishes to convince client C that it has address V. The attacker needs to have a network element on the path between A and the server (in order to modify the request) and on the path between the server and V so that it can forward the response to C. Furthermore, if there is a NAT between the attacker and the server, V must also be behind the same NAT. In such a situation, the attacker can either gain access to all the application-layer traffic or mount the DDOS attack described in Section 13.1.1. Note that any host which exists in the correct topological relationship can be DDOSed. It need not be using STUN.

14. IAB Considerations

Todo: The diagnostic usages have been removed from this document, which reduces the brittleness of STUN. This section should be updated accordingly.

The IAB has studied the problem of "Unilateral Self Address Fixing" (UNSAF), 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 (RFC3424 [21]). STUN is an example of a protocol that performs this type of function. The IAB has mandated that any protocols developed for this purpose document a specific set of considerations. This section meets those requirements.

14.1. Problem Definition

From RFC3424 [21], 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 STUN is to provide a means for a client to obtain an address on the public Internet from a non-symmetric NAT, for the express purpose of receiving incoming UDP traffic from another host, targeted to that address. STUN does not address traversal of NATs using TCP, either incoming or outgoing, and

does not address outgoing UDP communications.

14.2. Exit Strategy

From RFC3424 [21], any UNSAF proposal must provide:

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

STUN by itself does not provide an exit strategy. This is provided

by techniques, such as Interactive Connectivity Establishment (ICE [12]), which allow a client to determine whether addresses learned from STUN are needed, or whether other addresses, such as the one on the local interface, will work when communicating with another host. With such a detection technique, as a client finds that the addresses provided by STUN are never used, STUN queries can cease to be made, thus allowing them to phase out.

STUN can also help facilitate the introduction of other NAT traversal techniques such as MIDCOM [22]. As midcom-capable NATs are deployed, applications will, instead of using STUN (which also resides at the application layer), first allocate an address binding using midcom. However, it is a well-known limitation of MIDCOM that it only works when the agent knows the middleboxes through which its traffic will flow. Once bindings have been allocated from those middleboxes, a STUN detection procedure can validate that there are no additional middleboxes on the path from the public Internet to the client. If this is the case, the application can continue operation using the address bindings allocated from MIDCOM. If it is not the case, STUN provides a mechanism for self-address fixing through the remaining MIDCOM-unaware middleboxes. Thus, STUN provides a way to help transition to full MIDCOM-aware networks.

14.3. Brittleness Introduced by STUN

From RFC3424 [21], any UNSAF proposal must provide:

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

STUN introduces brittleness into the system in several ways:

- The binding acquisition usage is dependant on NAT's behavior when forwarding UDP packets from arbitrary hosts on the public side of the NAT. Application specific processing will generally be

> needed. For symmetric NATs, the binding acquisition will not yield a usable address. The tight dependency on the specific type of NAT makes the protocol brittle.

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

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

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

- The use of the STUN server as an additional network element introduces another point of failure. If the client cannot locate a STUN server, or if the server should be unavailable due to failure, the application cannot function.

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

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

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

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

- Most significantly, STUN introduces potential security threats which cannot be eliminated. This specification describes heuristics that can be used to mitigate the problem, but it is provably unsolvable given what STUN is trying to accomplish. These security problems are described fully in Section 13.

14.4. Requirements for a Long Term Solution

From RFC3424 [21], any UNSAF proposal must provide:

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

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

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

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

- Control needs to be limited. Users will need to communicate through NATs which are outside of their administrative control. In order for providers to be willing to deploy NATs which can be controlled by users in different domains, the scope of such

controls needs to be extremely limited - typically, allocating a

binding to reach the address where the control packets are coming from.

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

14.5. Issues with Existing NAPT Boxes

From RFC3424 [21], any UNSAF proposal must provide:

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

Several of the practical issues with STUN involve future proofing - breaking the protocol when new NAT types get deployed. Fortunately, this is not an issue at the current time, since most of the deployed NATs are of the types assumed by STUN. The primary usage STUN has found is in the area of VoIP, to facilitate allocation of addresses for receiving RTP [14] traffic. In that application, the periodic keepalives are usually (but not always) provided by the RTP traffic itself. However, several practical problems arise for RTP. First, in the absence of [23], RTP assumes that RTCP traffic is on a port one higher than the RTP traffic. This pairing property cannot be guaranteed through NATs that are not directly controllable. As a result, RTCP traffic may not be properly received. [23] mitigates this by allowing the client to signal a different port for RTCP but there will be interoperability problems for some time.

For VoIP, silence suppression can cause a gap in the transmission of RTP packets. If that silence period exceeds the NAT binding timeout, this could result in the loss of a NAT binding in the middle of a call. This can be mitigated by sending occasional packets to keep the binding alive. However, the result is additional brittleness.

14.6. In Closing

The problems with STUN are not design flaws in STUN. The problems in

STUN have to do with the lack of standardized behaviors and controls in NATs. The result of this lack of standardization has been a proliferation of devices whose behavior is highly unpredictable, extremely variable, and uncontrollable. STUN does the best it can in such a hostile environment. Ultimately, the solution is to make the environment less hostile, and to introduce controls and standardized behaviors into NAT. However, until such time as that happens, STUN provides a good short term solution given the terrible conditions

under which it is forced to operate.

15. IANA Considerations

IANA is hereby requsted to create two new registries STUN Message Types and STUN Attributes. IANA must assign the following values to both registeries before publication of this document as an RFC. New values for both STUN Message Type and STUN Attributes are assigned through the IETF consensus process via RFCs approved by the IESG.

15.1. STUN Message Type Registry

For STUN Message Types that are request message types, they MUST be registered including associated Response message types and Error Response message types, and those responses must have values that are 0x100 and 0x110 higher than their respective Request values.

For STUN Message Types that are Indication message types, no associated restriction applies. As the message type field is only 14 bits the range of valid values is 0x001 through 0x3FFF.

The initial STUN Message Types are:

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

15.2. STUN Attribute Registry

STUN attributes values above 0x7FFF are considered optional attributes; attributes equal to 0x7FFF or below are considered mandatory attributes. The STUN client and STUN server process optional and mandatory attributes differently. IANA should assign values based on the RFC consensus process.

The initial STUN Attributes are:

0x0001: MAPPED-ADDRESS
0x0002: RESPONSE-ADDRESS
0x0003: CHANGE-REQUEST
0x0004: SOURCE-ADDRESS
0X0005: CHANGED-ADDRESS
0x0006: USERNAME
0x0007: PASSWORD
0x0008: MESSAGE-INTEGRITY
0x0009: ERROR-CODE
0x000A: UNKNOWN-ATTRIBUTES
0x000B: REFLECTED-FROM
0x000E: ALTERNATE-SERVER
0x0014: REALM
0x0015: NONCE
0x0020: XOR-MAPPED-ADDRESS
0x8022: SERVER
0x8023: ALTERNATE-SERVER
0x8024: BINDING-LIFETIME

16. Changes Since RFC 3489

This specification updates RFC3489 [13]. This specification differs from RFC3489 in the following ways:

- Removed the usage of STUN for NAT type detection and binding lifetime discovery. These techniques have proven overly brittle due to wider variations in the types of NAT devices than described in this document. Removed the RESPONSE-ADDRESS, CHANGED-ADDRESS, CHANGE-REQUEST, SOURCE-ADDRESS, and REFLECTED-FROM attributes.

- Removed the STUN example that centered around the separation of the control and media planes. Instead, provided more information on using STUN with protocols.

- Added a fixed 32-bit magic cookie and reduced length of transaction ID by 32 bits. The magic cookie begins at the same offset as the original transaction ID.

- Added the XOR-MAPPED-ADDRESS attribute, which is included in Binding Responses if the magic cookie is present in the request. Otherwise the RFC3489 behavior is retained (that is, Binding Response includes MAPPED-ADDRESS). See discussion in XOR-MAPPED-ADDRESS regarding this change.

- Explicitly point out that the most significant two bits of STUN are 0b00, allowing easy differentiation with RTP packets when used with ICE.

- Added support for IPv6. Made it clear that an IPv4 client could get a v6 mapped address, and vice-a-versa.

- Added the SERVER, REALM, NONCE, and ALTERNATE-SERVER attributes.

- Removed recommendation to continue listening for STUN Responses for 10 seconds in an attempt to recognize an attack.

17. Acknowledgements

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

Schulzrinne for IESG and IAB input on this work.

18. References

18.1. Normative References

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

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

[3] Rosenberg, J., "Traversal Using Relay NAT (TURN)", draft-rosenberg-midcom-turn-08 (work in progress), September 2005.

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

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

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

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

[8] 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. Informational References

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

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

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

[12] 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.

[13] Rosenberg, J., Weinberger, J., Huitema, C., and R. Mahy, "STUN - Simple Traversal of User Datagram Protocol (UDP) Through Network Address Translators (NATs)", RFC 3489, March 2003.

[14] Schulzrinne, H., Casner, S., Frederick, R., and V. Jacobson, "RTP: A Transport Protocol for Real-Time Applications", STD 64, RFC 3550, July 2003.

[15] Jennings, C. and R. Mahy, "Managing Client Initiated Connections in the Session Initiation Protocol (SIP)", draft-ietf-sip-outbound-01 (work in progress), October 2005.

[16] Kohl, J. and B. Neuman, "The Kerberos Network Authentication Service (V5)", RFC 1510, September 1993.

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

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

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

[20] Baugher, M., McGrew, D., Naslund, M., Carrara, E., and K. Norrman, "The Secure Real-time Transport Protocol (SRTP)",

RFC 3711, March 2004.

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

[22] Srisuresh, P., Kuthan, J., Rosenberg, J., Molitor, A., and A. Rayhan, "Middlebox communication architecture and framework", RFC 3303, August 2002.

[23] Huitema, C., "Real Time Control Protocol (RTCP) attribute in Session Description Protocol (SDP)", RFC 3605, October 2003.

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

Christian Huitema
Microsoft
One Microsoft Way
Redmond, WA 98052
US

Email: huitema@microsoft.com

Rohan Mahy
Plantronics
345 Encinal Street
Santa Cruz, CA 95060
US

Email: rohan@ekabal.com

Dan Wing
Cisco Systems
771 Alder Drive
San Jose, CA 95035
US

Email: dwing@cisco.com

Intellectual Property Statement

The IETF takes no position regarding the validity or scope of any Intellectual Property Rights or other rights that might be claimed to pertain to the implementation or use of the technology described in this document or the extent to which any license under such rights might or might not be available; nor does it represent that it has made any independent effort to identify any such rights. Information on the procedures with respect to rights in RFC documents can be found in BCP 78 and BCP 79.

Copies of IPR disclosures made to the IETF Secretariat and any assurances of licenses to be made available, or the result of an attempt made to obtain a general license or permission for the use of such proprietary rights by implementers or users of this specification can be obtained from the IETF on-line IPR repository at http://www.ietf.org/ipr.

The IETF invites any interested party to bring to its attention any copyrights, patents or patent applications, or other proprietary rights that may cover technology that may be required to implement this standard. Please address the information to the IETF at ietf-ipr@ietf.org.

Disclaimer of Validity

Copyright Statement

Acknowledgment

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