

BEHAVE
Internet-Draft
Expires: January 12, 2007

J. Rosenberg
Cisco Systems
C. Huitema
Microsoft
R. Mahy
Plantronics
D. Wing
Cisco Systems
July 11, 2006

Simple Traversal Underneath Network Address Translators (NAT) (STUN)
draft-ietf-behave-rfc3489bis-04

Status of this Memo

By submitting this Internet-Draft, each author represents that any applicable patent or other IPR claims of which he or she is aware have been or will be disclosed, and any of which he or she becomes aware will be disclosed, in accordance with [Section 6 of BCP 79](#).

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF), its areas, and its working groups. Note that other groups may also distribute working documents as Internet-Drafts.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

The list of current Internet-Drafts can be accessed at <http://www.ietf.org/ietf/1id-abstracts.txt>.

The list of Internet-Draft Shadow Directories can be accessed at <http://www.ietf.org/shadow.html>.

This Internet-Draft will expire on January 12, 2007.

Copyright Notice

Copyright (C) The Internet Society (2006).

Abstract

Simple Traversal Underneath NATs (STUN) is a lightweight protocol that serves as a tool for application protocols in dealing with NAT traversal. It allows a client to determine the IP address and port

allocated to them by a NAT and to keep NAT bindings open. It can also serve as a check for connectivity between a client and a server in the presence of NAT, and for the client to detect failure of the server. 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	5
2.	Introduction	5
3.	Terminology	6
4.	Definitions	6
5.	Overview of Operation	7
6.	STUN Message Structure	11
7.	STUN Transactions	14
7.1.	Request/Response Transactions	14
7.2.	Indications	15
8.	Client Behavior	15
8.1.	Discovery	15
8.2.	Obtaining a Shared Secret	16
8.3.	Request/Response Transactions	17
8.3.1.	Formulating the Request Message	17
8.3.2.	Processing Responses	18
8.4.	Indication Transactions	21
9.	Server Behavior	22
9.1.	Request/Response Transactions	22
9.1.1.	Receive Request Message	23
9.1.2.	Constructing the Response	25
9.1.3.	Sending the Response	26
9.2.	Indication Transactions	26
10.	Demultiplexing of STUN and Application Traffic	27
11.	STUN Attributes	28
11.1.	MAPPED-ADDRESS	28
11.2.	USERNAME	29
11.3.	PASSWORD	29
11.4.	MESSAGE-INTEGRITY	30
11.5.	FINGERPRINT	30
11.6.	ERROR-CODE	30
11.7.	REALM	32
11.8.	NONCE	32
11.9.	UNKNOWN-ATTRIBUTES	32
11.10.	XOR-MAPPED-ADDRESS	33
11.11.	SERVER	34
11.12.	ALTERNATE-SERVER	34
11.13.	REFRESH-INTERVAL	34
12.	STUN Usages	34

12.1.	Binding Discovery	35
12.1.1.	Applicability	35
12.1.2.	Client Discovery of Server	36
12.1.3.	Server Determination of Usage	36
12.1.4.	New Requests or Indications	37
12.1.5.	New Attributes	37
12.1.6.	New Error Response Codes	37
12.1.7.	Client Procedures	37
12.1.8.	Server Procedures	37
12.1.9.	Security Considerations for Binding Discovery	37
12.2.	Connectivity Check	37
12.2.1.	Applicability	37
12.2.2.	Client Discovery of Server	38
12.2.3.	Server Determination of Usage	38
12.2.4.	New Requests or Indications	38
12.2.5.	New Attributes	39
12.2.6.	New Error Response Codes	39
12.2.7.	Client Procedures	39
12.2.8.	Server Procedures	39
12.2.9.	Security Considerations for Connectivity Check	39
12.3.	NAT Keepalives	39
12.3.1.	Applicability	39
12.3.2.	Client Discovery of Server	40
12.3.3.	Server Determination of Usage	40
12.3.4.	New Requests or Indications	40
12.3.5.	New Attributes	40
12.3.6.	New Error Response Codes	40
12.3.7.	Client Procedures	41
12.3.8.	Server Procedures	41
12.3.9.	Security Considerations for NAT Keepalives	41
12.4.	Short-Term Password	41
12.4.1.	Applicability	42
12.4.2.	Client Discovery of Server	42
12.4.3.	Server Determination of Usage	42
12.4.4.	New Requests or Indications	42
12.4.5.	New Attributes	43
12.4.6.	New Error Response Codes	43
12.4.7.	Client Procedures	43
12.4.8.	Server Procedures	44
12.4.9.	Security Considerations for Short-Term Password	44
13.	Security Considerations	46
13.1.	Attacks on STUN	46
13.1.1.	Attack I: DDoS Against a Target	46
13.1.2.	Attack II: Silencing a Client	47
13.1.3.	Attack III: Assuming the Identity of a Client	47
13.1.4.	Attack IV: Eavesdropping	47
13.2.	Launching the Attacks	47
13.2.1.	Approach I: Compromise a Legitimate STUN Server	48

13.2.2.	Approach II: DNS Attacks	48
13.2.3.	Approach III: Rogue Router or NAT	48
13.2.4.	Approach IV: Man in the Middle	49
13.2.5.	Approach V: Response Injection Plus DoS	49
13.2.6.	Approach VI: Duplication	50
13.3.	Countermeasures	50
13.4.	Residual Threats	52
14.	IAB Considerations	52
14.1.	Problem Definition	52
14.2.	Exit Strategy	52
14.3.	Brittleness Introduced by STUN	53
14.4.	Requirements for a Long Term Solution	54
14.5.	Issues with Existing NAT Boxes	55
15.	IANA Considerations	56
15.1.	STUN Message Type Registry	56
15.2.	STUN Attribute Registry	56
16.	Changes Since RFC 3489	57
17.	Acknowledgements	58
18.	References	58
18.1.	Normative References	58
18.2.	Informational References	59
	Authors' Addresses	61
	Intellectual Property and Copyright Statements	62

1. Applicability Statement

This protocol is not a cure-all for the problems associated with NAT. It is a tool that is typically used in conjunction with other protocols, such as Interactive Connectivity Establishment (ICE) [[11](#)] for a more complete solution. The binding discovery usage, defined by this specification, can be used by itself with numerous application protocols as a solution for NAT traversal. However, when used in that way, STUN will not work with applications that require incoming TCP connections through NAT. It will allow incoming UDP packets through NAT, but only through a subset of existing NAT types. In particular, the STUN binding usage by itself does not enable incoming UDP packets through NATs whose mapping property is address dependent or address and port dependent [[12](#)]. Furthermore, the binding usage, when used by itself, does not work when a client is communicating with a peer which happens to be behind the same NAT. Nor will it work when the STUN server is not in a common shared address realm.

The STUN relay usage, defined in [[14](#)], allows a client to obtain an IP address and port that actually reside on the STUN server. The STUN relay usage, when used by itself, eliminates all of the limitations of using the binding usage by itself, as described above. However, it requires a server to act as a relay for application traffic, which can be expensive to provide, operate and manage.

For multimedia protocols based on the offer/answer model [[20](#)], including the Session Initiation Protocol (SIP) [[9](#)], Interactive Connectivity Establishment (ICE) uses both the binding usage and relay usage, and furthermore makes use of the connectivity check usage defined here to help decide which of those two mechanisms ought to be used.

Implementors should be aware of the specific deployment scenarios that are of interest, and of the specific protocol (whether its SIP or something else) in order to determine whether STUN is suitable as a tool to facilitate NAT traversal, and which usage should be used.

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 [[18](#)] 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](#) [[19](#)]) 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 Underneath 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 entities to validate connectivity and liveness. 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 and receives STUN responses. Clients can also generate STUN indications.

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

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 transport 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.

Long Term Credential: A username and associated password which represent a shared secret between client and server. Long term credentials are generally granted to the client when a subscriber enrolls in a service and persists until the subscriber leaves the service or explicitly changes the credential.

Long Term Password: The password from a long term credential.

Short Term Credential: A temporary username and associated password which represent a shared secret between client and server. A short term credential has an explicit temporal scope, which may be based on a specific amount of time (such as 5 minutes) or on an event (such as termination of a SIP dialog). The specific scope of a short term credential is defined by the application usage. A short term credential can be obtained from a Shared Secret request, though other mechanisms are possible.

Short Term Password: The password component of a short term credential.

5. Overview of Operation

This section is descriptive only. Normative behavior is described in [Section 8](#) and [Section 9](#)

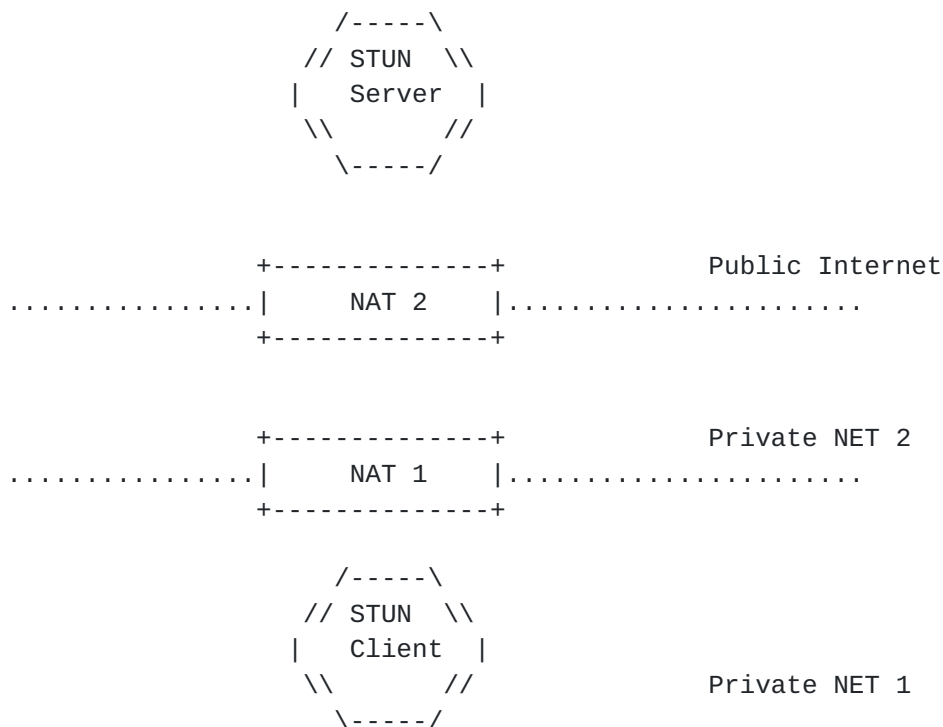


Figure 1: Typical STUN Server Configuration

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. It supports two types of transactions. One is a request/response transaction in which client sends a request to a server, and the server returns a response. The second are indications which are initiated by the server or the client and do not elicit a response. There are two types of requests defined in this specification - Binding Requests and Shared Secret Requests. There are no indications defined by this specification.

Binding Requests are sent from the client towards the server. When the Binding Request arrives at the STUN server, it may have passed through one or more NATs between the STUN client and the STUN server (in Figure 1, there were two such NATs). As a result, the source transport 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. From this response, the client knows its IP address and port allocated by the outermost NAT towards the STUN server.

STUN provides several mechanisms for authentication and message integrity. The client and server can share a pre-provisioned shared secret, which is used for a digest challenge/response authentication operation. This is known as a long-term credential or long-term shared secret.

Alternatively, if the shared secret is obtained by some out-of-bands means and has a lifetime that is temporally scoped, a simple HMAC is provided, without a challenge operation. This is known as a short term credential or short term password. 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.

STUN itself provides a mechanism for obtaining such short term credentials, using the Shared Secret Request. Shared Secret requests are sent over TLS [5] over TCP. Shared Secret Requests ask the server to return a temporary username and password that can be used in subsequent STUN requests.

There are many ways in which these basic mechanisms can be used to accomplish a specific task. As a result, STUN has the notion of a usage. A usage is a specific use case for the STUN protocol. The usage will define what it is that the client does with the mapped address it receives, defines when the client would send Binding requests and why, and would constrain the set of authentication mechanisms or attributes that get used in that usage. STUN usages can also define new attributes and message types, if needed. This specification defines four STUN usages - binding discovery, connectivity check, NAT keepalives, and short-term password.

The binding discovery usage is sometimes referred to as 'classic STUN', since it is the usage originally envisioned in RFC 3489 [13], the predecessor to this specification. The purpose of the binding discovery usage is for the client to obtain an IP address and port at which it is reachable, that it can include in application layer signaling messages, such as the Session Description Protocol (SDP) [17] body of a SIP message, utilized to receive Real Time Transport Protocol (RTP [15]) traffic. In this usage, the STUN server is typically located on the public Internet and run by the service provider offering the application service (such as a SIP provider), though this need not be the case. The client would utilize the STUN request just prior to sending a protocol message (such as a SIP

INVITE request or 200 OK response) which requires the client to embed its IP address in it.

In the connectivity check usage, two hosts on the Internet have used a protocol such as SIP to rendezvous, and have used it to exchange IP addresses and ports at which they might be reachable. However, each host does not know whether it can actually connect to the other host using that IP address and port, and whether that remote host can reach it. To figure this out, each host will send a STUN Binding Request to the other host, and if a reply is received, it knows that the remote host was reachable. Furthermore, the mapped address returned in the response tells the host the address and port at which it can be reached by the remote host. The connectivity check usage is used by ICE [\[11\]](#), for example.

In the binding keepalive usage, a client sends an application protocol message (such as a SIP REGISTER message) to a server. The server notes the source IP address and port of the request, and remembers it. Later on, if it needs to reach the client, it sends a message to that IP address and port. However, this message will only be received by the client if the binding in the NAT is still alive. Since bindings allocated by NAT expire unless refreshed, the client must generate keepalive messages towards the server to refresh the binding. Rather than use expensive application layer messages, a STUN binding request is sent by the client to the server, and is sent to the exact same IP address and port used by the server for the application protocol. In the case of SIP, this would typically mean port 5060 or 5061. This has the effect of keeping the bindings in the NAT alive. The STUN binding responses also inform the client that the server is still responsive, and also inform the client if its IP address and port towards the server have changed, in which case it may need application layer protocol messaging to update its IP address and port as seen by the server. The binding keepalive usage is used by the SIP outbound mechanism, for example [\[16\]](#).

These three usages all utilize the same Binding Request message, and all require the same basic processing on the server. They differ only in where the server is (embedded in a peer host, a standalone server in the network, or embedded in an application layer server), when the Binding Request is used, and what the client does with the mapped address that is returned.

The short-term password usage makes use of the Shared Secret request and response, and allows a client to obtain a temporary set of credentials to authenticate itself with the STUN server. The credentials obtained from this usage can be used in requests for any other usage.

Many of the usages (such as the binding keepalive and connectivity check usages) require STUN messages to be sent on the same IP address and port as some application protocol, such as RTP or SIP. To facilitate the demultiplexing of the two, STUN defines a special field in the message called the magic cookie, which is a fixed 32 bit value that identifies STUN traffic. STUN requests also contain a fingerprint, which is a cryptographic hash of the message, that allow for validation that the message was a STUN request and not a data packet that happened to have the same 32 bit value in the right place in the message.

STUN servers can be discovered through DNS, though this is not necessary in all usages. For those usages where it is needed, STUN makes use of SRV records [3] to facilitate discovery. This discovery allows for different IP addresses and ports to be found for different usages.

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 types of transactions in STUN - request/response transactions, which utilize a request message and a response message, and an indication transaction, which utilizes a single indication message. Furthermore, responses are broken into two types - success responses and error responses. The specific message (for example, that it is a Binding Response or a Shared Secret Request) is encoded into the message type field of the STUN header. For a particular request message, its success response has a type that is always 0x100 higher than its own type, and its error response has a type that is 0x110 higher than its own type. Extensions defining new requests, responses and error responses MUST use message type values 0x100 and 0x110 higher for their success and error responses, respectively.

STUN Requests are sent reliably. STUN can run over UDP or TCP. When run over UDP, STUN requests are retransmitted in order to achieve

reliability. 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 can be sent over TCP or UDP. STUN itself does not provide reliability for these messages, though they will be delivered reliably when sent over TCP. Indication messages do not have an associated success response message type or associated error response message type. Indication messages can be sent from the server to the client or client to server. The transaction ID is used to distinguish indication messages.

All STUN messages consist of a 20 byte header:

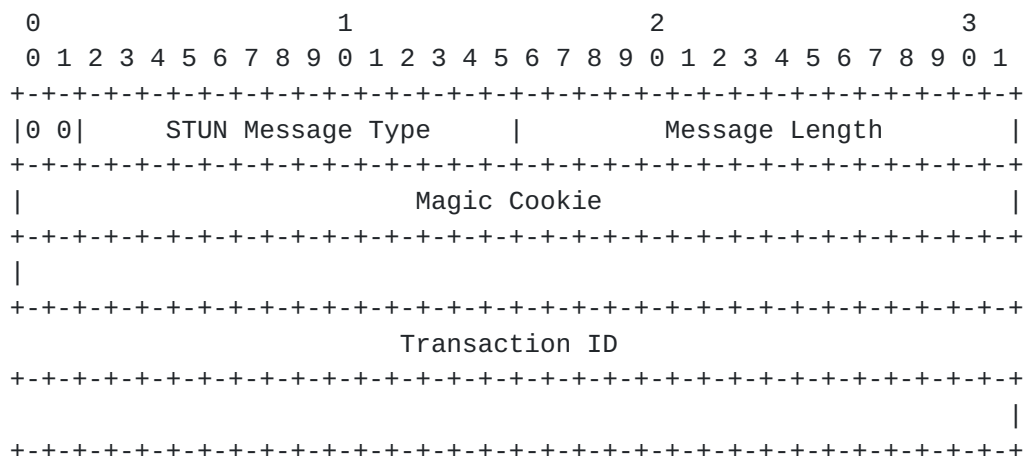


Figure 2: Format of STUN Message Header

The most significant two bits of every STUN message are both zeroes. This, combined with the magic cookie and the fingerprint attribute, aid 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 12.4](#). 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 is used as part of the identification of a STUN message when STUN is multiplexed with other

protocols on the same port, as is done for example in [11] and [16]. 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, error responses and indications.

The transaction ID is a 96 bit identifier. STUN transactions are identified by their unique 96-bit transaction ID. For request/response transactions, the transaction ID is chosen by the STUN client and MUST be unique for each new STUN transaction generated 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. A response to the STUN request, whether it be a success or error response, carries the same transaction ID as the request. Indications are also identified by their transaction ID. The value is chosen by the server and MUST be unique for each unique indication generated by the server. 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 $2^{96} - 1$.

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. Each STUN attribute ends on a 32 bit boundary:

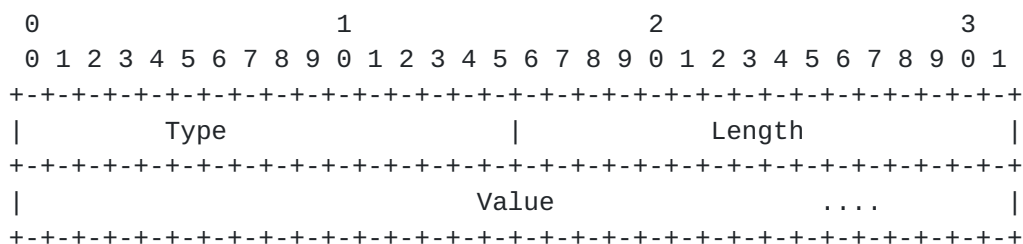


Figure 3: Format of STUN Attributes

The Length refers to the length of the actual useful content of the Value portion of the attribute, measured in bytes. Since STUN aligns attributes on 32 bit boundaries, attributes whose content is not a multiple of 4 bytes are padded with 1, 2 or 3 bytes of padding so that they are a multiple of 4 bytes. Such padding is only needed with attributes that take freeform strings, such as USERNAME and PASSWORD. For attributes that contain more structured data, the attributes are constructed to align on 32 bit boundaries. The value

in the Length field refers to the length of the Value part of the attribute prior to padding - i.e., the useful content. Consequently, when parsing messages, implementations will need to round up the Length field to the nearest multiple of four in order to find the start of the next attribute.

The attribute types defined in this specification are in [Section 11](#).

7. STUN Transactions

STUN defines two types of transactions - request/response transactions and indication transactions.

7.1. Request/Response 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.

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 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. A STUN transaction over UDP is also considered failed if there has been a transport failure of some sort, such as a fatal ICMP error.

When running STUN over TCP, TCP is responsible for ensuring delivery. The STUN application SHOULD NOT retransmit STUN requests when running over TCP. If the client has not received a response after 9500ms, it considers the transaction to have failed.

Regardless of whether TCP or UDP was used for the transaction, if a failure occurs and the client has other servers it can reach (as a consequence of an SRV query which provides a multiplicity of STUN servers [Section 8.1](#), for example), the client SHOULD create a new request, which is identical to the previous, but has a different transaction ID (and consequently a different MESSAGE INTEGRITY and

FINGERPRINT attribute).

7.2. Indications

Indications are sent from the client to the server, or from the server to the client. Though no indications are used by this specification, they are used by the STUN relay usage [\[14\]](#). When sent over UDP, there are no retransmissions, and reliability is not provided. When sent over TCP, reliability is provided by TCP.

If a data indication solicits a fatal ICMP error, or causes a TCP error, the transaction is considered to have failed. In such a case, the client SHOULD create a new transaction, which is identical to the previous, but has a different transaction ID (and consequently a different MESSAGE INTEGRITY and FINGERPRINT attribute). That request is sent to the next element in the list as specified by [RFC2782](#).

8. Client Behavior

Client behavior can be broken down into several steps. The first is discovery of the STUN server. The next is obtaining a shared secret. For request/response transactions, the next steps are formulating the request and processing the response. For indication transactions, the next step is formulating the indication.

8.1. 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](#) [\[3\]](#). 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 depends on the application usage. For the binding usage, the service name is "stun". The protocol can be "udp" for UDP, "tcp" for TCP and "tls" for TLS over TCP. For the short term password application usage, the service name is "stun-pass". The protocol is always "tls" for TLS over TCP. The binding keepalive and connectivity check usages always start with an IP address, so no DNS SRV service names are defined for them. New STUN usages MAY define additional DNS SRV service names. These SHOULD start with "stun".

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 8.3.1](#).

A STUN server supporting multiple usages (such as the short term password and binding discovery usage) MAY use the same ports for different usages, as long as ports are not needed to differentiate the usages. Different ports are not needed to differentiate the usages defined in this specification. Different ports SHOULD be used for TCP and TCP/TLS, so that the server can determine whether the first message it will receive after the TCP connection is set up is a STUN message or a TLS message.

The default port for STUN requests is 3478, for both TCP and UDP. There is no default port for STUN over TLS. Administrators SHOULD use this port in their SRV records for UDP and TCP, 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 using UDP or TCP, independent of the STUN usage. For usages that require TLS, such as the short term password usage, lack of SRV records is equivalent to a failure of the transaction, since the request or indication MUST NOT be sent unless SRV records provided an address and port specifically for TLS.

[8.2](#). 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. The application usage defines the mechanism and required implementation strength for shared secrets.

Some usages, such as the connectivity check usage, assume that out of band protocols, such as ICE, are used to obtain the necessary credentials. Other usages, such as binding keepalives, don't use authentication, as it is not required. Others, such as the binding discovery, allows for authentication using either a long term shared secret or a short term shared secret. The latter can be obtained by using the short term password usage to obtain a short term shared secret.

Consequently, the STUN usages define rules for obtaining shared

secrets prior to sending a request.

8.3. Request/Response Transactions

8.3.1. 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 if the usage employs authentication. The specific credentials to use are described by the STUN usage, which can specify no credentials, a short term credential, or a long term credential. The procedures for each are:

1. If the STUN usage specifies that no credentials are used, the message is sent without MESSAGE-INTEGRITY
2. If a short term credential is to be used, 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. If a long term credential is to be used, the STUN request contains the USERNAME, REALM, and MESSAGE-INTEGRITY attributes. The 16-byte key for MESSAGE-INTEGRITY HMAC is formed by taking the MD5 hash of the result of concatenating the following five fields: (1) The username, with any quotes and trailing nulls removed, (2) A single colon, (3) The realm, with any quotes and trailing nulls removed, (4) A single colon, and (5) The password, with any trailing nulls removed. For example, if the USERNAME field were 'user', the REALM field were '"realm"', and the PASSWORD field were 'pass', then the 16-byte HMAC key would be the result of performing an MD5 hash on the string 'user:realm:pass', or 0x8493fbc53ba582fb4c044c456bdc40eb.

This format for the key was chosen so as to enable a common authentication database for SIP, which uses digest authentication as defined in [RFC 2617](#) [7] and STUN, as it is expected that credentials are usually stored in their hashed forms.

The NONCE is included in the request only if a short or long term credential is being used, and only if the STUN request is a retry as a consequence of a previous error response which provided the client

with a NONCE.

For TCP and TLS-over-TCP, the client opens a TCP connection to the server. The TLS_RSA_WITH_AES_128_CBC_SHA ciphersuite MUST be supported at a minimum by implementers when TLS is used with STUN. Implementers MAY also support any other ciphersuite. When it receives the TLS Certificate message, the client SHOULD verify the certificate and inspect the site identified by the certificate. If the certificate is invalid, revoked, or if it does not identify the appropriate party, the client MUST NOT send the STUN message or otherwise proceed with the STUN transaction. The client MUST verify the identity of the server. To do that, it follows the identification procedures defined in [Section 3.1 of RFC 2818](#) [4]. 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 8.1](#) as the host portion of the URI that has been dereferenced. If DNS was not used, the client MUST be configured with a set of authorized domains whose certificates will be accepted.

The client MUST include a FINGERPRINT attribute as the last attribute.

Next, the client sends the request. For UDP-based requests, reliability is accomplished through client retransmissions, following the procedure in [Section 7.1](#). For TCP (including TLS over TCP), there are no retransmissions.

For TCP and TLS over TCP, 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 or TLS over TCP, the client SHOULD close the connection as soon as it has received the STUN Response, if it has no plans to send further requests.

[8.3.2](#). 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 outstanding 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 11.4](#). The key that is used 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.

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.

If the client receives a 401 response and had not included a MESSAGE-INTEGRITY attribute in the request, it is an indication from the server that credentials are required. If the REALM attribute was present in the response, it is a signal to the client to use a long term shared secret and retry the request. The client SHOULD retry the request, using the username and password associated with the REALM. If the server had provided a nonce in the 401 response, the client SHOULD include the same nonce in the retry. If the REALM attribute was absent in the response, it is a signal to the client to use a short term shared secret and retry the request. If the client doesn't have a short term shared secret, it SHOULD use the Shared Secret request to obtain one, and then retry the request with the username and password obtained as a result. If the 401 response had contained a NONCE attribute, that same nonce is included in the retry.

If the client receives a 401 response but had included a MESSAGE-INTEGRITY attribute in the request, there has been an unrecoverable error. This shouldn't ever happen, but if it does, the client SHOULD NOT retry the request.

If the client receives a 432 response, and the client had omitted the USERNAME from the request but included a MESSAGE-INTEGRITY, the client SHOULD retry the request and include both MESSAGE-INTEGRITY and USERNAME. If the 432 response had contained a NONCE attribute, that same nonce is included in the retry. If the client receives a 432 but had included both MESSAGE-INTEGRITY and USERNAME in the request, there has been an unrecoverable error. This shouldn't ever happen, but if it does, the client SHOULD NOT retry the request.

If the client receives a 435 response, but had included a NONCE in the request, an unrecoverable error has occurred and the client SHOULD NOT retry. However, if it had omitted the NONCE in the request and received a 435, or it had included the NONCE but received a 438, it is a request from the server to retry using the same credential, but with a different nonce. The client SHOULD retry the request, using the nonce provided in the NONCE attribute of the 435 or 438 response.

If the client receives a 430 response, it means that the client used a short term credential which has expired. If the client had submitted the request using a short term credential obtained from a Shared Secret request, the client SHOULD generate a new Shared Secret request to obtain a new short term credential, and then retry the request with that credential. Note that the Shared Secret request may or may not go to the same server which generated the 430 response; the server that receives the Shared Secret request is determined by the DNS procedures defined above. If a 430 response was received and the client had used a short term credential provided through some other means, the client SHOULD obtain a new short term credential using that mechanism. If the client had not used a short term credential in the request, the 430 error is unrecoverable and the request SHOULD NOT be retried. If the 430 response had contained a NONCE attribute, that same nonce is included in the retry.

For a 431 response code, the client SHOULD alert the user, and if a short term credential obtained from a Shared Secret request had been used previously, the client MAY try the request again after obtaining a new short term username and password. If the 431 response had contained a NONCE attribute, that same nonce is included in the retry.

If the client receives a 433 response, and the request was a Shared Secret request which was not sent over TLS, the client SHOULD retry the request, and MUST send it using TLS. If this response is received to any other request except for a Shared Secret request, or if the client had sent the Shared Secret request over TLS, it is an unrecoverable error and the client SHOULD NOT retry. As with other error responses, the 433 can contain a NONCE, and if present, that nonce is used in the request retry.

If the client receives a 434 response, and had omitted the REALM in the request, but had included MESSAGE-INTEGRITY, it is an indication that, though a short-term credential was used for the request, the server desires the client to use a long term credential. The client SHOULD retry the request using the username and password associated with the REALM. As with other error responses, the 434 can contain a NONCE, and if present, that nonce is used in the request retry. If the 434 was received but the request had contained a REALM, and the REALM in the response differs from the REALM in the request, the client SHOULD retry using the username and password associated with the REALM in the response. If the REALMS were equal, this is an unrecoverable error and the client SHOULD NOT retry.

If the client receives a 436 response, it means that the username it provided in the request is unknown. For usages where the username was collected from the user, the client SHOULD alert the user. The

client SHOULD NOT retry with the same username.

For error responses which can contain a NONCE, the client includes the NONCE in a subsequent retry as discussed above. Furthermore, the client SHOULD cache the nonce, and continue using it in subsequent requests sent to the same server, identified by IP address and port.

For a 300 response code, the client SHOULD attempt a new transaction to the server indicated in the ALTERNATE-SERVER attribute. This is useful for load balancing requests across a STUN server cluster, when those requests require some amount of resources to process. Though this specification allows the 300 response to be applied to Binding Requests, it is generally not useful to do so, since the work of redirecting a Binding Request is equal to, if not more than, the work of just processing the Binding Request. Consequently, the 300 response code is targeted for other usages of STUN, such as the relay usage [14].

For a 500 response code, the client MAY wait several seconds and then retry the request on the same server. Or, if the server was learned through DNS SRV records, the client MAY try the request on a different server. The same username and password MAY be used. For a 600 response code, client MUST NOT retry the request on this server, or if the server was learned through DNS, any other server found through the DNS resolution procedures. 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.

Responses containing unknown optional attributes (greater than 0x7FFF) MUST be ignored by the STUN client. 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.

The processing of other attributes in the response, such as the mapped address (present in the XOR-MAPPED-ADDRESS attribute or MAPPED-ADDRESS attribute) depends on the STUN usage.

8.4. Indication Transactions

This section applies to client and server behavior for sending an

Indication message.

The client or server 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.

Indication messages cannot be challenged or rejected. Consequently, they cannot be authenticated using long term credentials. If a STUN usage specifies that authentication is needed for an indication message, it can only be done using a short term credential. In that case, the client or server MUST add a MESSAGE-INTEGRITY and USERNAME attribute to the Request message. The key for MESSAGE-INTEGRITY is the password.

The client or server MUST include a FINGERPRINT attribute as the last attribute of any Indication message.

Typically, indication messages are sent to the same IP address and port, or over the same TCP connections as a previous request message. However, a usage can specify that indication messages are sent based on a DNS query, in which case the discovery procedures in [Section 8.1](#) are followed, along with the TCP/TLS connection establishment procedures defined in [Section 8.3.1](#).

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

For TCP and TLS over TCP, the client or server MAY send multiple indications on the connection. By definition, since indications do not generate a response, they can be pipelined on the connection. For clients, the connection is closed once it determines it has no further messages to send to the server. Servers do not normally close TCP connections.

9. Server Behavior

As with clients, server behavior depends on whether it is a request/response transaction or indication.

9.1. Request/Response Transactions

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

9.1.1.1. Receive Request Message

A STUN server MUST be prepared to receive request 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](#). 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.

When a STUN request is received, the server determines the usage. The usages describe how the STUN server makes this determination.

Based on the usage, the server determines whether the request requires any authentication and message integrity checks. It can require none, short-term credential based authentication, or long-term credential authentication.

If authentication is required, the server checks for the presence of the MESSAGE-INTEGRITY attribute. If not present, the server generates an error response with an ERROR-CODE attribute and a response code of 401. If the server wishes the client to use a short term credential, the REALM is omitted from the response. If the server wishes the client to use a long term credential, the REALM is included in the response containing a realm from which the username and password are scoped [7]. When the REALM is present in the response, the server MUST include a NONCE attribute in the response. The nonce includes a random value that the server wishes the client to reflect back in a subsequent request (and therefore include in the message integrity computation). When the REALM is absent in the response, the server MAY include a NONCE in the response if it wishes to use nonces along with short-term shared secrets. However, there is little reason to do so, since the short-term password is, by definition, short-term, and thus additional temporal scoping through the nonce is not needed.

If the MESSAGE-INTEGRITY 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. If the server is using a long term credential for authentication, the response MUST include a REALM and a NONCE. If the server is using a short-term credential, it MUST NOT include a REALM in the response and MAY include a NONCE.

If the server is using long term credentials for authentication, and the request contained the MESSAGE-INTEGRITY and USERNAME attributes, 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 and the server is using a long term credential for authentication, 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 was absent and the server is authenticating with short term credentials, the server MAY generate an error response with an ERROR-CODE attribute with a response code of 435. This response MUST include a NONCE. If the NONCE was present in the request, but the server has determined it is stale, the server MUST generate an error response with an ERROR-CODE attribute with a response code of 438.

Next, if the server is authenticating the request, it checks for the presence of the USERNAME attribute. If absent, the server generates an error response with an ERROR-CODE attribute with a response code of 432. If the server is authenticating using long term credentials, it MUST include a REALM and NONCE in the response. If the server is authenticating with short term credentials, it MUST NOT include a REALM and MAY include a NONCE.

If the server is authenticating the request with a short term credential, it checks the value of the USERNAME field. If the USERNAME was previously valid but has expired, the server generates an error response with an ERROR-CODE attribute with a response code of 430. This error response MAY include a NONCE. If the server is authenticating with either short or long term credentials, it determines whether the USERNAME contains a known entity, and in the case of a long-term credential, known within the realm of the REALM attribute of the request. If the USERNAME is unknown, the server generates an error response with an ERROR-CODE attribute with a response code of 436. For authentication using long-term credentials, that error response MUST include a NONCE attribute and a REALM attribute. For authentication using short-term credentials, it MAY include a NONCE but MUST NOT include a REALM.

At this point, if the server is doing authentication, the request contains everything needed for that purpose. The server computes the HMAC over the request as described in [Section 11.4](#). The key depends on the credential. For short-term credentials, it equals the password associated with the username. For long term credentials, it is computed as described in [Section 8.3.1](#).

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. If long term credentials are being used for authentication, this response MUST include a NONCE attribute and a REALM attribute. If short term credentials are being used, it MAY include a NONCE and MUST NOT include a REALM.

At this point, the request has been authentication checked and integrity verified.

Next, the server MUST check for any mandatory attributes in the request (values less than or equal to 0x7fff) which it does not understand. If it encounters any, the server MUST generate an 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 of the request. If the STUN response is a success response, the STUN server adds 0x110 to the Message Type of the request.

The attributes that get added to the response depend on the type of response. See Figure 4 for a summary.

If the response is a type which can carry either MAPPED-ADDRESS or XOR-MAPPED-ADDRESS (the Binding Response as defined in this specification meets that criteria), the server examines the magic cookie in the STUN header. If it has the value 0x2112A442, it indicates that the client supports this version of the specification. The server MUST insert a XOR-MAPPED-ADDRESS into the response, carrying the exclusive-or of the source IP address and port from the request. If the magic cookie did not have this value, it indicates that the client supports the previous version of this specification. The server MUST insert a MAPPED-ADDRESS attribute into the response, carrying the source IP address and port from the request. Insertion of either XOR-MAPPED-ADDRESS or MAPPED-ADDRESS happens regardless of the transport protocol used for the request.

XOR-MAPPED-ADDRESS and MAPPED-ADDRESS differ only in their encoding of the IP address and port. The former, as implied by the name,

encodes the IP address and port by exclusive-or'ing them with the magic cookie. The latter encodes them directly in binary. [RFC 3489](#) originally specified only MAPPED-ADDRESS. However, deployment experience found that some NATs rewrite the 32-bit binary payloads containing the NAT's public IP address, such as STUN's MAPPED-ADDRESS attribute, in the well-meaning but misguided attempt at providing a generic ALG function. Such behavior interferes with the operation of STUN and also causes failure of STUN's message integrity checking.

The server SHOULD include a SERVER attribute in all responses, indicating the identity of the server generating the response. This is useful for diagnostic purposes.

The server MUST include a FINGERPRINT attribute as the last attribute of any Indication message.

In cases where the server cannot handle the request, due to exhaustion of resources, the server MAY generate a 300 response with an ALTERNATE-SERVER attribute. This attribute identifies an alternate server which can service the requests. It is not expected that 300 responses or this attribute would be used by the requests defined in this specification.

[9.1.3.](#) Sending the Response

All UDP 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. All TCP or TLS over TCP responses messages are sent on the TCP connections that the request arrived on.

[9.2.](#) Indication Transactions

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

A STUN server MUST be prepared to receive 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](#). 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.

When a STUN indication is received, the server determines the usage. The usages describe how the STUN server makes this determination.

Based on the usage, the server determines whether the indication requires any authentication and message integrity checks. It can

require none or short-term credential based authentication. If short-term credentials are utilized, the server follows the same procedures as defined in [Section 9.1.1](#), but if those procedures require transmission of an error response, the server MUST instead silently discard the indication.

Once authenticated (if authentication was in use), the processing of the indication message depends on the message type. This specification doesn't define any indication messages.

10. Demultiplexing of STUN and Application Traffic

In both the connectivity check and binding refresh usages, STUN traffic is multiplexed on the same IP address and port as application traffic, such as RTP. In order to apply the processing described in this specification, STUN messages must first be separated from the application packets. This disambiguation is done identically for requests and responses of all types.

First, all STUN messages start with two bits equal to zero. If STUN is being multiplexed with application traffic where it is known that the topmost two bits are never zeroes, the presence of these two zeroes signals STUN traffic.

If this mechanism doesn't suffice, the magic cookie can be used. All STUN messages have the value 0x2112A442 as the second 32 bit word. If the application traffic can not have this value as the second 32 bit word, then any packets with this value are STUN packets. Even if the application packet can have this value (for example, in cases where the application packets contain random binary data), there is only a one in 2^{32} chance that an application packet will have a value of 0x2112A442 in its second 32 bit word. If this probability is deemed sufficiently small for the application at hand (for example, it is considered adequate for Voice over IP applications), then any packet with this value in its second 32 bit word is processed as a STUN packet.

However, a mis-classification of 1 in 2^{32} may still be too high for some usages of STUN. Consequently, all STUN messages contain a FINGERPRINT attribute. This is a cryptographic hash over the message, covering everything prior to the attribute. This attribute is different from MESSAGE-INTEGRITY. The latter uses a keyed HMAC, and thus requires a shared secret. FINGERPRINT does not use a password, and can be computed just by examining the STUN message. Thus, if a packet appears to be a STUN message because it has a value of 0x2112A442 in its second 32 bit word, a client or server then assumes the message is a STUN message, and computes the value for the

fingerprint. It then looks for the FINGERPRINT attribute in the message, and if the value equals the computed value, the message is considered to be a STUN message. If not, it is considered to be an application packet

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.

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	Indication
MAPPED-ADDRESS	-	C	-	-
USERNAME	O	-	-	O
PASSWORD	-	C	-	-
MESSAGE-INTEGRITY	O	O	O	O
ERROR-CODE	-	-	M	-
ALTERNATE-SERVER	-	-	C	-
REALM	C	-	C	-
NONCE	C	-	C	-
UNKNOWN-ATTRIBUTES	-	-	C	-
XOR-MAPPED-ADDRESS	-	C	-	-
SERVER	-	O	O	O
REFRESH-INTERVAL	-	O	-	-
FINGERPRINT	M	M	M	M

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.

The format of the MAPPED-ADDRESS attribute is:

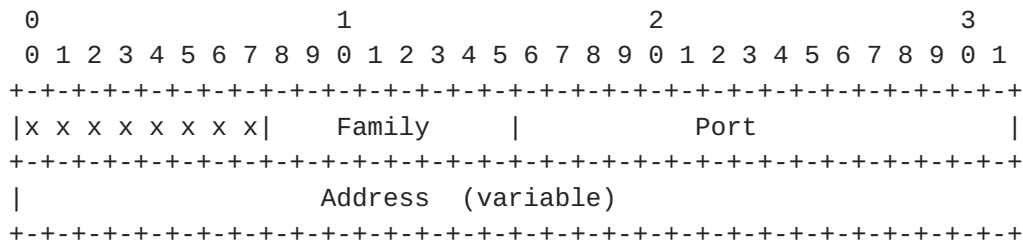


Figure 5: Format of MAPPED-ADDRESS attribute

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 request arrived from.

The first 8 bits of the MAPPED-ADDRESS are ignored for the purposes of aligning parameters on natural 32 bit boundaries.

11.2. USERNAME

The USERNAME attribute is used for message integrity. It identifies the shared secret used in the message integrity check. Consequently, the USERNAME MUST be included in any request that contains the MESSAGE-INTEGRITY attribute.

The USERNAME is also always present in a Shared Secret Response, along with the PASSWORD, which informs a client of a short term password.

The value of USERNAME is a variable length opaque value. Note that, as described above, if the USERNAME is not a multiple of four bytes it is padded for encoding into the STUN message, in which case the attribute length represents the length of the USERNAME prior to padding.

11.3. 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. Note that, as described above, if the USERNAME is not a multiple of four bytes it is padded for encoding into the STUN message, in which case the attribute length represents the length of the USERNAME prior to padding.

11.4. MESSAGE-INTEGRITY

The MESSAGE-INTEGRITY attribute contains an HMAC-SHA1 [8] 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 generally the next to last attribute in any STUN message. With the exception of the FINGERPRINT attribute, which appears after MESSAGE-INTEGRITY, elements MUST ignore all other attributes which follow MESSAGE-INTEGRITY.

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

11.5. FINGERPRINT

The FINGERPRINT attribute is present in all STUN messages. It is computed as a SHA1 hash of the STUN message up to (but excluding) the FINGERPRINT attribute itself. The value of the attribute is the actual binary output of the SHA-1 function. The FINGERPRINT attribute MUST be the last attribute in the message.

11.6. 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 [9] and HTTP [10]. The reason phrase is meant for user consumption, and can be anything appropriate for the response code. Recommended reason phrases for the defined response codes are presented below.

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

```

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

```

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

If the reason phrase has a length that is not a multiple of four bytes, it is padded for encoding into the STUN message, in which case the attribute length represents the length of the entire ERROR-CODE attribute (including the reason phrase) prior to padding.

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 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 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 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 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 to the server.
- 438 (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.

11.7. REALM

The REALM attribute is present in requests and responses. It contains text which meets the grammar for "realm" as described in [RFC 3261](#) [9], and will thus contain a quoted string (including the quotes).

Presence of the REALM attribute in a request indicates that long-term credentials are being used for authentication. Presence in certain error responses indicates that the server wishes the client to use a long-term credential for authentication.

11.8. 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 [RFC 3261](#) [9]. See [RFC 2617](#) [7] for guidance on selection of nonce values in a server.

11.9. UNKNOWN-ATTRIBUTES

The UNKNOWN-ATTRIBUTES attribute is present only in an 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.

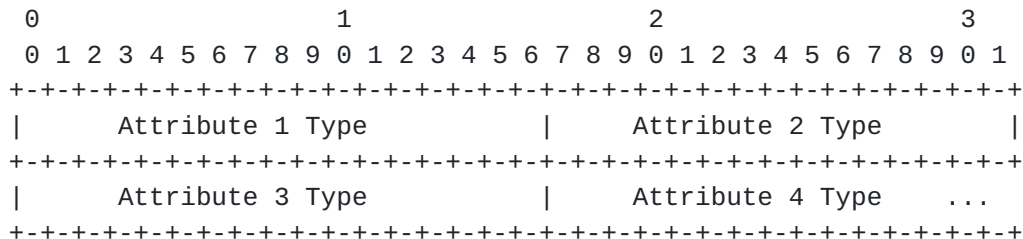


Figure 8: Format of UNKNOWN-ATTRIBUTES attribute

[11.10.](#) XOR-MAPPED-ADDRESS

The XOR-MAPPED-ADDRESS attribute is present in 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.

This attribute **MUST** always be present in a Binding Response and may be used in other responses as well. Usages defining new requests and responses should specify if XOR-MAPPED-ADDRESS is applicable to their responses.

The format of the XOR-MAPPED-ADDRESS is:

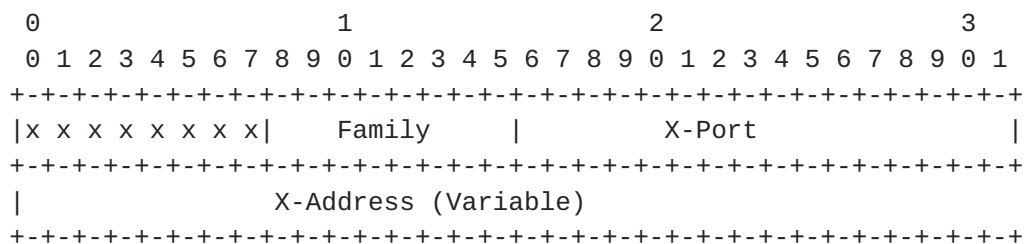


Figure 9: Format of XOR-MAPPED-ADDRESS Attribute

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 the 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.

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 $0x15B3 \wedge 0x2112 = 0x34A1$, and the X-Address would be $0xc0a80101 \wedge 0x2112A442 = 0xe1baa543$.

11.11. 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 value of SERVER is variable length. If the value of SERVER is not a multiple of four bytes, it is padded for encoding into the STUN message, in which case the attribute length represents the length of the USERNAME prior to padding.

11.12. 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.

11.13. REFRESH-INTERVAL

The REFRESH-INTERVAL indicates the number of milliseconds that the server suggests the client should use between refreshes of the NAT bindings between the client and server. Even though the server may not know the binding lifetimes in intervening NATs, this attribute serves as a useful configuration mechanism for suggesting a value for use by the client. Furthermore, when the NAT Keepalive usage is being used, the server may become overloaded with Binding Requests that are being used for keepalives. The REFRESH-INTERVAL provides a mechanism for the server to gradually reduce the load on itself by pushing back on the client.

REFRESH-INTERVAL is specified as an unsigned 32 bit integer, and represents an interval measured in milliseconds. It can be present in Binding Responses.

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 usage may differ in how STUN servers are discovered, when the STUN requests are sent, what message types are used, what message attributes are used, and how authentication is performed.

This specification defines the STUN usages for binding discovery ([Section 12.1](#)), connectivity check ([Section 12.2](#)), NAT keepalives ([Section 12.3](#)) and short-term password ([Section 12.4](#)).

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](#). Binding Discovery

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

[12.1.1](#). Applicability

Binding discovery is used to learn reflexive addresses from servers on the network, generally the public Internet. That is, it is used by a client to determine its dynamically-bound 'public' IP address and UDP port that is assigned by a NAT between a STUN client and a STUN server. This address and port will be present in the mapped address of the STUN Binding Response.

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 [[17](#)]) 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, generates a STUN Binding Request. 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)[[15](#)]. 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 approach is Interactive Connectivity Establishment (ICE [[11](#)]).

[12.1.2.](#) Client Discovery of Server

Clients SHOULD be configured with a domain name for a STUN server to use. In cases where the client has no explicit configuration mechanism for STUN, but knows the domain of its service provider, the client SHOULD use that domain (in the case of SIP, this would be the domain from their Address-of-Record). The discovery mechanisms defined in [Section 8.1](#) are then applied to that domain name.

[12.1.3.](#) Server Determination of Usage

It is anticipated that servers would advertise a specific port in the DNS for the Binding Discovery usage. Thus, when a request arrives at that particular port, the server knows that the binding usage is in

use. This fact is only needed for purposes of determining the authentication and message integrity mechanism to apply.

12.1.4. New Requests or Indications

This usage does not define any new message types.

12.1.5. New Attributes

This usage does not define any new message attributes.

12.1.6. New Error Response Codes

This usage does not define any new error response codes.

12.1.7. Client Procedures

The binding discovery is utilized by a client just prior to generating an application PDU that requires the client to include its IP address and port. The client MAY first obtain a short term credential using the short term password STUN usage. The credential that is obtained is then using in Binding Request messages. A Binding Request message is generated for each distinct IP address and port that the client requires to formulate the application PDU.

12.1.8. Server Procedures

It is RECOMMENDED that servers utilize short term credentials, obtained by the client from a Shared Secret request, for authentication and message integrity. Consequently, if a Binding Request is generated without a short term credential, the server SHOULD challenge for one.

12.1.9. Security Considerations for Binding Discovery

There are no security considerations for this usage beyond those described in [Section 13](#).

12.2. Connectivity Check

12.2.1. Applicability

This STUN usage primarily provides a connectivity check to a peer discovered through rendezvous protocols. The usage presumes that some other mechanism, such as ICE [[11](#)] has been used allow two peer agents to exchange IP addresses and ports. The agents would like to initiate direct communications with each other, using those IP addresses and ports. However, it is not known which IP addresses and

ports will actually work for direct exchange of communications, due to NAT, firewall and other network connectivity issues. Consequently, each agent uses a STUN Binding Request from each of their transport addresses to each of the transport addresses of their peers. This check serves numerous purposes. Firstly, if a response is received to a Binding Request, an agent knows that that particular 5-tuple (the transport address that the Binding Request was sent to, along with the one it was sent from) are viable for direct communications. Secondly, the mapped address from the Binding Response tells the agent its reflexive address towards its peer, which may be another candidate for receipt of communications. Finally, the connectivity checks can keep NAT bindings in intervening NATs active.

It is fundamental to this STUN usage that the addresses and ports used for direct communications are the same ones used for the Binding Requests and responses. Consequently, it will be necessary to demultiplex STUN traffic from whatever the application data traffic is. This demultiplexing is done using the magic cookie along with the FINGERPRINT attribute.

Because the connectivity check usage is always used in conjunction with some kind of rendezvous protocol, it is assumed that the rendezvous protocol provides the exchange of a short term credential. This credential is then used for authentication and message integrity of the STUN Binding Requests and Responses.

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

12.2.2. Client Discovery of Server

The client does not follow the general procedure in [Section 8.1](#). Instead, the client discovers the STUN server's IP address and port through a rendezvous protocol. Note that the STUN server is a logical entity in this usage, and it will be running on the exact same IP address and port as is used for actual communications.

12.2.3. Server Determination of Usage

The server is aware of this usage because it signalled this port through the rendezvous protocol. Any STUN packets received on this port will be for the connectivity check 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

A client will generate a connectivity check based on the procedures defined in the rendezvous protocol that uses this usage. Generally this will be done after an exchange of IP addresses and ports has occurred between the two clients, at which point connectivity checks will begin.

Clients MUST include the FINGERPRINT attribute, to aid in disambiguation of STUN and application traffic. Clients MUST use a short term credential obtained through the rendezvous protocol. The specific structure and format of the username and password are defined by the rendezvous protocol. Receipt of any non-recoverable STUN error is an indication that there is no connectivity to that IP address and port.

12.2.8. 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.

12.2.9. 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.

12.3. NAT Keepalives

12.3.1. Applicability

In this STUN usage, a client is connected to a server for a particular application protocol (for example, a SIP proxy server). The connection is long-lived, allowing for asynchronous messaging from the server to the client. The client is connected to the server either using TCP, in which case there is a long-lived TCP connection from the client to the server, or using UDP, in which case the server

stores the source IP address and port of a message from a client (such as SIP REGISTER), and sends messages to the client using that IP address and port.

Since the connection between the client and server is very-long lived, the bindings established by that connection need to be maintained in any intervening NATs. Rather than implement expensive application-layer keepalives, the keepalives can be accomplished using STUN Binding Requests. The client will periodically sending a Binding Request to the server, using the same IP address and ports used for the application protocol. These Binding Requests are demultiplexed at the server using the magic cookie and possibly FINGERPRINT. The response from the server informs the client that the server is still alive. The STUN message also keeps the binding active in intervening NATs. The client can also examine the mapped address in the Binding Response. If it has changed, the client can re-initiate application layer procedures to inform the server of its new IP address and port.

12.3.2. 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 12.2](#)) usage, this usage is differs as the server's port is fixed and the server's port isn't communicated using a rendezvous protocol.

12.3.3. 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.

12.3.4. 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

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.

The client SHOULD NOT include a MESSAGE-INTEGRITY attribute, since authentication is not used with this STUN usage.

12.3.8. Server Procedures

The server MUST NOT authenticate the client or look for a MESSAGE-INTEGRITY attribute. Authentication is not used with this STUN usage.

12.3.9. Security Considerations for NAT Keepalives

This STUN usage does not employ message integrity or authentication of any sort. This is because the client never actually uses the mapped address from the STUN response. It merely treats a change in that address as a hint that the client should re-apply application layer procedures for connection establishment and registration.

An attacker could attempt to inject faked responses, or modify responses in transit. Such an attack would require the attacker to be on-path in order to determine the transaction ID. In the worse case, the attack would cause the client to see a change in IP address or port, and then perform an application layer re-registration. Such a re-registration would not use the IP address and port obtained from the Binding Response. Thus, the worst that the attacker can do is cause the client to re-register every half minute or so, when it otherwise wouldn't need to. Given the difficulty in launching this attack (it requires the attacker to be on-path and to disrupt the actual response from the server) compared to the benefit, there is little motivation for authentication or integrity mechanisms.

12.4. Short-Term Password

In order to ensure interoperability, this usage describes a TLS-based mechanism to obtain a short-term credential. The usage makes use of the Shared Secret Request and Response messages. It is defined as a separate usage in order to allow it to run on a separate port, and to allow it to be more easily separated from the different STUN usages, only some of which require this mechanism.

12.4.1. 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 12.4.2](#). The username and password obtained with this usage are used as the USERNAME and in the HMAC for the MESSAGE-INTEGRITY in a subsequent STUN message, respectively.

12.4.2. Client Discovery of Server

The client follows the procedures in [Section 8.1](#). The SRV protocol is "tls" and the service name "stun-pass".

For example a client would look up "_stun-pass._tls.example.com" in DNS.

12.4.3. Server Determination of Usage

The server advertises this port in the DNS as capable of receiving TLS over TCP connections, along with the Shared Secret messages that run over it. The server MAY also advertise this same port in DNS for other TLS over TCP usages if the server is capable of multiplexing those different usages. For example, the server could advertise the short-term password and binding discovery usages on the same TLS/TCP port.

12.4.4. 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 - 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	O	M	-
PASSWORD	-	M	-
MESSAGE-INTEGRITY	O	O	O
ERROR-CODE	-	-	M
ALTERNATE-SERVER	-	-	C
UNKNOWN-ATTRIBUTES	-	-	C
SERVER	-	O	O
REALM	C	-	C
NONCE	C	-	C

The Shared Secret requests, like other STUN requests, can be authenticated. However, since its purpose is to obtain a short-term credential, the Shared Secret request itself cannot be authenticated with a short-term credential. However, it can be authenticated with a long-term credential.

[12.4.5.](#) New Attributes

No new attributes are defined by this usage.

[12.4.6.](#) New Error Response Codes

This usage defines the 433 error response. Only the MESSAGE-INTEGRITY, ERROR-CODE and SERVER attributes are applicable to this response.

[12.4.7.](#) Client Procedures

Shared Secret requests are formed like other STUN requests, with the following additions. Clients **MUST NOT** use a short-term credential with a Shared Secret request. They **SHOULD** send the request with no credentials (omitting MESSAGE-INTEGRITY and USERNAME).

Processing of the Shared Secret response follows that of any other STUN response. Note that clients **MUST** be prepared to be challenged

for a long-term credential.

If the response was a Shared Secret Response, the it will contain a short lived username and password, encoded in the USERNAME and PASSWORD attributes, respectively. A client SHOULD use these credentials whenever short term credentials are needed for any server discovered using the same domain name as was used to discover the one which returned those credentials. For example, if a client used a domain name of example.com, it would have looked up _stun-pass._tls.example.com in DNS, found a server, and sent a Shared Secret request that provided a credential to the client. The client would use this credential with a server discovered by looking up _stun._udp.example.com in the DNS.

If the response was a Shared Secret Error Response, and ERROR-CODE attribute was present with a response code of 433, and the client had not sent the request over TLS, the client SHOULD establish a TLS connection to the server and retry the request over that connection. If the client had used TLS, this error response is unrecoverable and the client SHOULD NOT retry.

12.4.8. Server Procedures

The procedures for general processing of STUN requests apply to Shared Secret requests. Servers MAY challenge the client for a long-term credential if one was not provided in a request. However, they MUST NOT challenge the request for a short-term credential.

If the Shared Secret Request did not arrive over a TLS connection, the server MUST generate a Shared Secret Error response with an ERROR-CODE attribute that has a response code of 433.

If the request is valid and authenticated (assuming the server is performing authentication), the server MUST create a short term credential for the user. This credential consists of a username and password. The credentials MUST be valid for a duration of at least nine minutes, and SHOULD NOT be valid for a duration of longer than thirty minutes. The username MUST be distinct, with extremely high probabilities, from all usernames that have been handed out across all servers that are returned from DNS SRV queries for the same domain name. Extremely high probability means that the likelihood of collision SHOULD be better than 1 in 2^{64} . The password for each username MUST be cryptographically random with at least 128 bits of entropy.

12.4.9. Security Considerations for Short-Term Password

The security considerations in [Section 13](#) do not apply to the Shared

Secret request and response, since these messages do not make use of mapped addresses, which is the primary source of security consideration discussed there. Rather, shared secret requests are used to obtain short term credentials that are used in the authentication of other messages.

Because the Shared Secret response itself carries a credential, in the form of a username and password, it must be sent encrypted. For this reason, STUN servers **MUST** reject any Shared Secret request that has not arrived over a TLS connection.

Malicious clients could generate a multiplicity of Shared Secret requests, each of which causes the server to allocate shared secrets, each of which might consume memory and processing resources. If shared secret requests are not being authenticated, this leads to a possible denial-of-service attack. Indeed, even if the requestor is authenticated, attacks are still possible.

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, servers **SHOULD** allocate only a small number of shared secrets to a host with a particular source IP address and port. Requests from the same IP address and port which exceed this limit **SHOULD** be rejected with a 600 response. Servers **SHOULD** also limit the total number of shared secrets they will provide at a time across all clients, based on the number of users and expected loads during normal peak usage. If a Shared Secret request arrives and the server has exceeded its limit, it **SHOULD** reject the request with a 500 response.

Furthermore, for servers which are not authenticating shared secret requests, it is **RECOMMENDED** that short-term credentials be constructed in a way such that they do not require memory or disk to store.

This can be done by intelligently computing the username and password. One approach is to construct the USERNAME as:

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

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

The password is then computed as:

```
password = <hmac(USERNAME,anotherprivatekey)>
```

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

13. Security Considerations

Attacks on STUN systems vary depending on the usage. The short term password usage is quite different from the other usages defined here, and its security considerations are unique to it and discussed as part of the usage definition. However, all of the other usages are very similar, and share a similar set of security considerations as a consequence of their usage of the mapped address from STUN Binding Responses. Consequently, these security considerations apply to usage of the mapped address.

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. 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 mapped address. 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 themselves. 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. Clearly, this attack is only applicable for usages which discover servers through DNS.

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 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 [6]. 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, for usages where the STUN server is not co-located with some kind of application (such as the binding discovery usage), 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 8.2](#) 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 [21].

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 11.4](#).

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. These three approaches are 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

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](#) [22]). STUN is an example of a protocol that performs this type of function for the binding discovery and connectivity check usages. The IAB has mandated that any protocols developed for this purpose document a specific set of considerations. This section meets those requirements for those two usages.

14.1. Problem Definition

From [RFC3424](#) [22], 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 a mapped address which can be used for the receipt of incoming application packets.

14.2. Exit Strategy

From [RFC3424](#) [22], 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 [[11](#)]), 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.

14.3. Brittleness Introduced by STUN

From [RFC3424](#) [[22](#)], any UNSAF proposal must provide:

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

STUN introduces brittleness into the system in several ways:

- o Transport addresses discovered by STUN in the Binding Discovery usage will only be useful for receiving packets from a peer if the NAT does not have address or address and port dependent mapping properties. When this usage is used in isolation, this makes STUN brittle, since its effectiveness depends on the type of NAT. This brittleness is eliminated when the Binding Discovery usage is used in concert with mechanisms which can verify the transport address and use others if it doesn't work. ICE is an example of such a mechanism.
- o Transport addresses discovered by STUN in the Binding Discovery usage will only be useful for receive packets from a peer if the STUN server subtends the address realm of the peer. 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. When this usage is used in isolation, this makes STUN brittle, since its effectiveness depends on the topological placement of the STUN server. This brittleness is eliminated when the Binding Discovery usage is used in concert with mechanisms which can verify the transport address and use others if it doesn't work. ICE is an example of such a

mechanism.

- o The bindings allocated from the NAT need to be continuously refreshed. Since the timeouts for these bindings is very implementation specific, the refresh interval cannot easily be determined. When the binding is not being actively used to receive traffic, but to wait for an incoming message, the binding refresh will needlessly consume network bandwidth.
- o The use of the STUN server in the Binding Discovery usage as an additional network element introduces another point of potential security attack. These attacks are largely prevented by the security measures provided by STUN, but not entirely.
- o The use of the STUN server as an additional network element introduces another point of failure. If the client cannot locate a STUN server, or if the server should be unavailable due to failure, the application cannot function.
- o The use of STUN to discover address bindings may result in an increase in latency for applications.
- o Transport addresses discovered by STUN in the Binding Discovery usage will only be useful for receive packets from a peer behind the same NAT if the STUN server supports hairpinning [12]. When this usage is used in isolation, this makes STUN brittle, since its effectiveness depends on the topological placement of the STUN server. This brittleness is eliminated when the Binding Discovery usage is used in concert with mechanisms which can verify the transport address and use others if it doesn't work. ICE is an example of such a mechanism.
- o Most significantly, STUN introduces potential security threats which cannot be eliminated through cryptographic means. These security problems are described fully in [Section 13](#).

[14.4](#). Requirements for a Long Term Solution

From [RFC3424](#) [22], 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:

- o 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, or knowing what parameters the NAT will use.
- o 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.
- o 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](#) [22], any UNSAF proposal must provide:

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

Originally, [RFC 3489](#) was developed as a standalone solution for NAT traversal for several types of applications, including VoIP. However, practical experience found that the limitations of its usage in isolation made it impractical as a complete solution. There were too many NATs which didn't support hairpinning or which had address and port dependent mapping properties.

Consequently, STUN was revised to produce this specification, which turns STUN into a tool that is used as part of a broader solution. For multimedia communications protocols, this broader solution is ICE. ICE uses the binding discovery and connectivity check usages together. When done this way, ICE eliminates almost all of the brittleness and issues found with [RFC 3489](#) alone.

15. IANA Considerations

IANA is hereby requested to create two new registries STUN Message Types and STUN Attributes. IANA must assign the following values to both registries 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 [23].

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

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
0x0006: USERNAME
0x0007: PASSWORD
0x0008: MESSAGE-INTEGRITY
0x0009: ERROR-CODE
0x000A: UNKNOWN-ATTRIBUTES
0x0014: REALM
0x0015: NONCE
0x0020: XOR-MAPPED-ADDRESS
0x0023: FINGERPRINT
0x8022: SERVER
0x8023: ALTERNATE-SERVER
0x8024: REFRESH-INTERVAL

16. Changes Since [RFC 3489](#)

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

- o 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.
- o Removed the STUN example that centered around the separation of the control and media planes. Instead, provided more information on using STUN with protocols.
- o 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.
- o 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.
- o Explicitly point out that the most significant two bits of STUN are 0b00, allowing easy differentiation with RTP packets when used with ICE.

- o Added support for IPv6. Made it clear that an IPv4 client could get a v6 mapped address, and vice-a-versa.
- o Added long-term credential-based authentication.
- o Added the SERVER, REALM, NONCE, and ALTERNATE-SERVER attributes.
- o Removed recommendation to continue listening for STUN Responses for 10 seconds in an attempt to recognize an attack.
- o Introduced the concept of STUN usages and defined four usages - Binding Discovery, Connectivity Check, NAT Keepalive, and Short term password.

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] Gulbrandsen, A., Vixie, P., and L. Esibov, "A DNS RR for specifying the location of services (DNS SRV)", [RFC 2782](#), February 2000.
- [4] Rescorla, E., "HTTP Over TLS", [RFC 2818](#), May 2000.
- [5] Dierks, T. and C. Allen, "The TLS Protocol Version 1.0", [RFC 2246](#), January 1999.
- [6] 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.
- [7] 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

- [8] Krawczyk, H., Bellare, M., and R. Canetti, "HMAC: Keyed-Hashing for Message Authentication", [RFC 2104](#), February 1997.
- [9] 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.
- [10] 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.
- [11] Rosenberg, J., "Interactive Connectivity Establishment (ICE): A Methodology for Network Address Translator (NAT) Traversal for Offer/Answer Protocols", [draft-ietf-mmusic-ice-08](#) (work in progress), March 2006.
- [12] Audet, F. and C. Jennings, "NAT Behavioral Requirements for Unicast UDP", [draft-ietf-behave-nat-udp-07](#) (work in progress), June 2006.
- [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] Rosenberg, J., "Obtaining Relay Addresses from Simple Traversal of UDP Through NAT (STUN)", [draft-ietf-behave-turn-00](#) (work in progress), March 2006.
- [15] Schulzrinne, H., Casner, S., Frederick, R., and V. Jacobson, "RTP: A Transport Protocol for Real-Time Applications", [RFC 3550](#), July 2003.
- [16] Jennings, C. and R. Mahy, "Managing Client Initiated Connections in the Session Initiation Protocol (SIP)", [draft-ietf-sip-outbound-03](#) (work in progress), March 2006.
- [17] Handley, M., "SDP: Session Description Protocol", [draft-ietf-mmusic-sdp-new-26](#) (work in progress), January 2006.
- [18] Senie, D., "Network Address Translator (NAT)-Friendly Application Design Guidelines", [RFC 3235](#), January 2002.
- [19] Holdrege, M. and P. Srisuresh, "Protocol Complications with the IP Network Address Translator", [RFC 3027](#), January 2001.

- [20] Rosenberg, J. and H. Schulzrinne, "An Offer/Answer Model with Session Description Protocol (SDP)", [RFC 3264](#), June 2002.
- [21] Baugher, M., McGrew, D., Naslund, M., Carrara, E., and K. Norrman, "The Secure Real-time Transport Protocol (SRTP)", [RFC 3711](#), March 2004.
- [22] Daigle, L. and IAB, "IAB Considerations for UNilateral Self-Address Fixing (UNSAF) Across Network Address Translation", [RFC 3424](#), November 2002.
- [23] Narten, T. and H. Alvestrand, "Guidelines for Writing an IANA Considerations Section in RFCs", [BCP 26](#), [RFC 2434](#), October 1998.

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

This document and the information contained herein are provided on an "AS IS" basis and THE CONTRIBUTOR, THE ORGANIZATION HE/SHE REPRESENTS OR IS SPONSORED BY (IF ANY), THE INTERNET SOCIETY AND THE INTERNET ENGINEERING TASK FORCE DISCLAIM ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO ANY WARRANTY THAT THE USE OF THE INFORMATION HEREIN WILL NOT INFRINGE ANY RIGHTS OR ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

Copyright Statement

Copyright (C) The Internet Society (2006). This document is subject to the rights, licenses and restrictions contained in [BCP 78](#), and except as set forth therein, the authors retain all their rights.

Acknowledgment

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

