TRAM
Internet-Draft
Obsoletes: 5389 (if approved)
Intended status: Standards Track
Expires: September 10, 2015

M. Petit-Huguenin
Impedance Mismatch
G. Salgueiro
J. Rosenberg
D. Wing
Cisco
R. Mahy
Plantronics
P. Matthews
Avaya

March 09, 2015

Session Traversal Utilities for NAT (STUN)
draft-ietf-tram-stunbis-02

Abstract

Session Traversal Utilities for NAT (STUN) is a protocol that serves as a tool for other protocols in dealing with Network Address Translator (NAT) traversal. It can be used by an endpoint to determine the IP address and port allocated to it by a NAT. It can also be used to check connectivity between two endpoints, and as a keep-alive protocol to maintain NAT bindings. STUN works with many existing NATs, and does not require any special behavior from them.

STUN is not a NAT traversal solution by itself. Rather, it is a tool to be used in the context of a NAT traversal solution.

This document obsoletes RFC 5389.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at http://datatracker.ietf.org/drafts/current/.

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."

This Internet-Draft will expire on September 10, 2015.

Table of Contents

1. Introduction

The protocol defined in this specification, Session Traversal Utilities for NAT, provides a tool for dealing with NATs. It provides a means for an endpoint to determine the IP address and port allocated by a NAT that corresponds to its private IP address and port. It also provides a way for an endpoint to keep a NAT binding alive. With some extensions, the protocol can be used to do connectivity checks between two endpoints [RFC5245], or to relay packets between two endpoints [RFC5766].

In keeping with its tool nature, this specification defines an extensible packet format, defines operation over several transport protocols, and provides for two forms of authentication.

STUN is intended to be used in context of one or more NAT traversal solutions. These solutions are known as STUN usages. Each usage describes how STUN is utilized to achieve the NAT traversal solution. Typically, a usage indicates when STUN messages get sent, which optional attributes to include, what server is used, and what authentication mechanism is to be used. Interactive Connectivity Establishment (ICE) [RFC5245] is one usage of STUN. SIP Outbound [RFC5626] is another usage of STUN. In some cases, a usage will require extensions to STUN. A STUN extension can be in the form of new methods, attributes, or error response codes. More information on STUN usages can be found in Section 13.

Implementations and deployments of a STUN usage should follow the recommendations in [I-D.ietf-uta-tls-bcp].

2. Overview of Operation

This section is descriptive only.

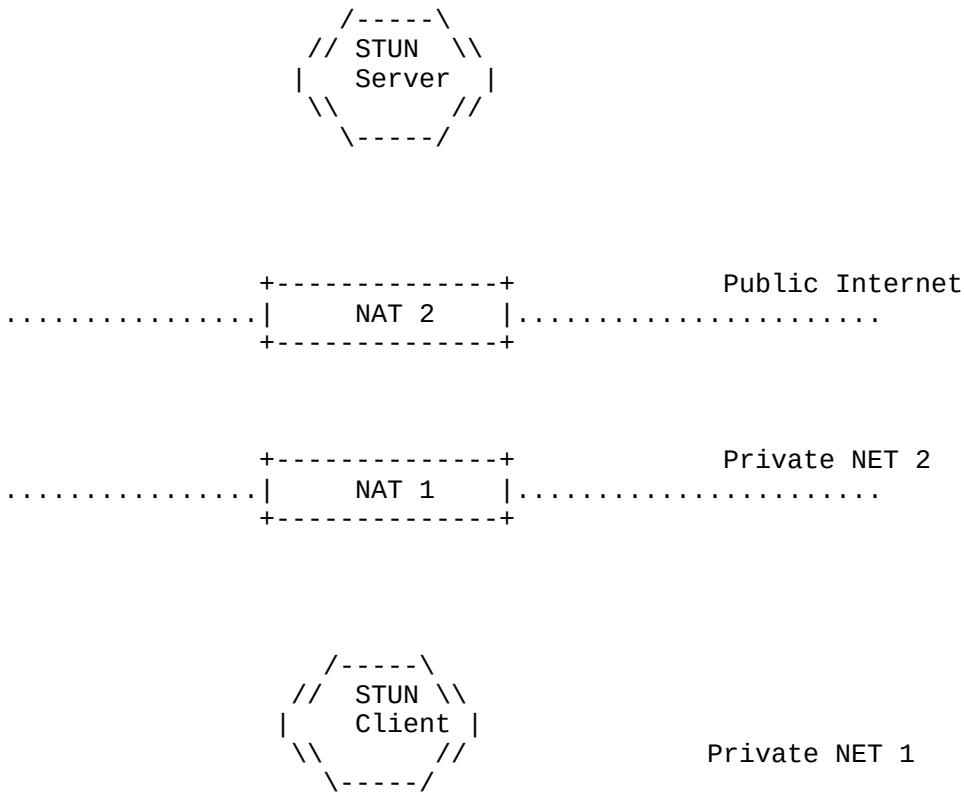


Figure 1: One Possible STUN Configuration

One possible STUN configuration is shown in Figure 1. In this configuration, there are two entities (called STUN agents) that implement the STUN protocol. The lower agent in the figure is the client, and 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 upper agent in the figure is the server, and resides on the public Internet.

STUN is a client-server protocol. It supports two types of transactions. One is a request/response transaction in which a client sends a request to a server, and the server returns a response. The second is an indication transaction in which either agent -- client or server -- sends an indication that generates no response. Both types of transactions include a transaction ID, which

is a randomly selected 96-bit number. For request/response transactions, this transaction ID allows the client to associate the response with the request that generated it; for indications, the transaction ID serves as a debugging aid.

All STUN messages start with a fixed header that includes a method, a class, and the transaction ID. The method indicates which of the various requests or indications this is; this specification defines just one method, Binding, but other methods are expected to be defined in other documents. The class indicates whether this is a request, a success response, an error response, or an indication. Following the fixed header comes zero or more attributes, which are Type-Length-Value extensions that convey additional information for the specific message.

This document defines a single method called Binding. The Binding method can be used either in request/response transactions or in indication transactions. When used in request/response transactions, the Binding method can be used to determine the particular "binding" a NAT has allocated to a STUN client. When used in either request/response or in indication transactions, the Binding method can also be used to keep these "bindings" alive.

In the Binding request/response transaction, a Binding request is sent from a STUN client to a STUN 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 the Binding request message passes through a NAT, the NAT will modify the source transport address (that is, the source IP address and the source port) of the packet. As a result, the source transport address of the request received by the server will be the public IP address and port created by the NAT closest to the server. This is called a reflexive transport address. The STUN server copies that source transport address into an XOR-MAPPED-ADDRESS attribute in the STUN Binding response and sends the Binding response back to the STUN client. As this packet passes back through a NAT, the NAT will modify the destination transport address in the IP header, but the transport address in the XOR-MAPPED-ADDRESS attribute within the body of the STUN response will remain untouched. In this way, the client can learn its reflexive transport address allocated by the outermost NAT with respect to the STUN server.

In some usages, STUN must be multiplexed with other protocols (e.g., [RFC5245], [RFC5626]). In these usages, there must be a way to inspect a packet and determine if it is a STUN packet or not. STUN provides three fields in the STUN header with fixed values that can be used for this purpose. If this is not sufficient, then STUN

packets can also contain a FINGERPRINT value, which can further be used to distinguish the packets.

STUN defines a set of optional procedures that a usage can decide to use, called mechanisms. These mechanisms include DNS discovery, a redirection technique to an alternate server, a fingerprint attribute for demultiplexing, and two authentication and message-integrity exchanges. The authentication mechanisms revolve around the use of a username, password, and message-integrity value. Two authentication mechanisms, the long-term credential mechanism and the short-term credential mechanism, are defined in this specification. Each usage specifies the mechanisms allowed with that usage.

In the long-term credential mechanism, the client and server share a pre-provisioned username and password and perform a digest challenge/response exchange inspired by (but differing in details) to the one defined for HTTP [RFC2617]. In the short-term credential mechanism, the client and the server exchange a username and password through some out-of-band method prior to the STUN exchange. For example, in the ICE usage [RFC5245] the two endpoints use out-of-band signaling to exchange a username and password. These are used to integrity protect and authenticate the request and response. There is no challenge or nonce used.

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 [RFC2119] and indicate requirement levels for compliant STUN implementations.

4. Definitions

STUN Agent: A STUN agent is an entity that implements the STUN protocol. The entity can be either a STUN client or a STUN server.

STUN Client: A STUN client is an entity that sends STUN requests and receives STUN responses. A STUN client can also send indications. In this specification, the terms STUN client and client are synonymous.

STUN Server: A STUN server is an entity that receives STUN requests and sends STUN responses. A STUN server can also send indications. In this specification, the terms STUN server and server are synonymous.

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

Reflexive Transport Address: A transport address learned by a client that 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 mapped address 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: Same meaning as reflexive address. This term is retained only for historic reasons and due to the naming of the MAPPED-ADDRESS and XOR-MAPPED-ADDRESS attributes.

Long-Term Credential: A username and associated password that 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 persist 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 that represent a shared secret between client and server. Short-term credentials are obtained through some kind of protocol mechanism between the client and server, preceding the STUN exchange. 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.

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

STUN Indication: A STUN message that does not receive a response.

Attribute: The STUN term for a Type-Length-Value (TLV) object that can be added to a STUN message. Attributes are divided into two types: comprehension-required and comprehension-optional. STUN agents can safely ignore comprehension-optional attributes they don’t understand, but cannot successfully process a message if it contains comprehension-required attributes that are not understood.

RTO: Retransmission TimeOut, which defines the initial period of time between transmission of a request and the first retransmit of that request.

5. STUN Message Structure

STUN messages are encoded in binary using network-oriented format (most significant byte or octet first, also commonly known as big-endian). The transmission order is described in detail in Appendix B of RFC791 [RFC0791]. Unless otherwise noted, numeric constants are in decimal (base 10).

All STUN messages MUST start with a 20-byte header followed by zero or more Attributes. The STUN header contains a STUN message type, magic cookie, transaction ID, and message length.

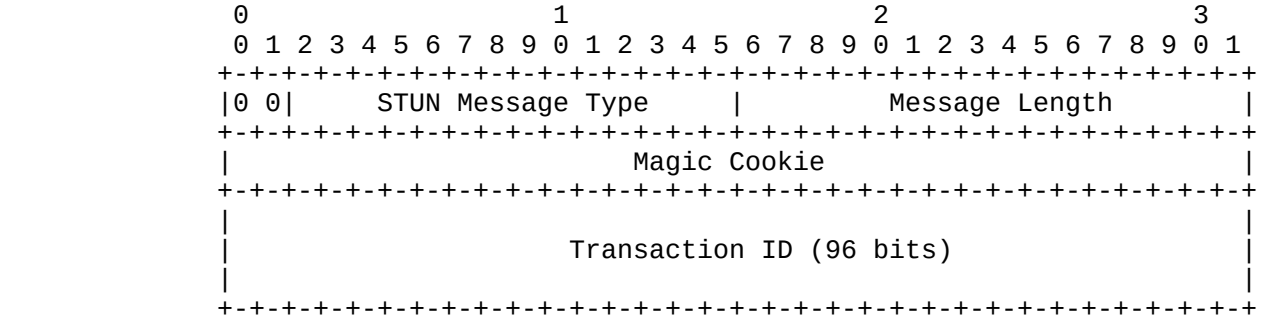


Figure 2: Format of STUN Message Header

The most significant 2 bits of every STUN message MUST be zeroes. This can be used to differentiate STUN packets from other protocols when STUN is multiplexed with other protocols on the same port.

The message type defines the message class (request, success response, failure response, or indication) and the message method (the primary function) of the STUN message. Although there are four message classes, there are only two types of transactions in STUN: request/response transactions (which consist of a request message and a response message) and indication transactions (which consist of a single indication message). Response classes are split into error and success responses to aid in quickly processing the STUN message.

The message type field is decomposed further into the following structure:

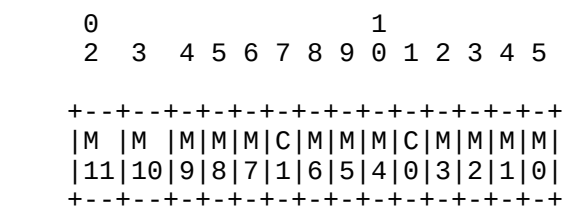


Figure 3: Format of STUN Message Type Field

Here the bits in the message type field are shown as most significant (M11) through least significant (M0). M11 through M0 represent a 12-bit encoding of the method. C1 and C0 represent a 2-bit encoding of the class. A class of 0b00 is a request, a class of 0b01 is an indication, a class of 0b10 is a success response, and a class of 0b11 is an error response. This specification defines a single method, Binding. The method and class are orthogonal, so that for each method, a request, success response, error response, and indication are possible for that method. Extensions defining new methods MUST indicate which classes are permitted for that method.

For example, a Binding request has class=0b00 (request) and method=0b000000000001 (Binding) and is encoded into the first 16 bits as 0x0001. A Binding response has class=0b10 (success response) and method=0b000000000001, and is encoded into the first 16 bits as 0x0101.

> Note: This unfortunate encoding is due to assignment of values in [RFC3489] that did not consider encoding Indications, Success, and Errors using bit fields.

The magic cookie field MUST contain the fixed value 0x2112A442 in network byte order. In RFC 3489 [RFC3489], this field was part of the transaction ID; placing the magic cookie in this location allows a server to detect if the client will understand certain attributes that were added in this revised specification. In addition, it aids in distinguishing STUN packets from packets of other protocols when STUN is multiplexed with those other protocols on the same port.

The transaction ID is a 96-bit identifier, used to uniquely identify STUN transactions. For request/response transactions, the transaction ID is chosen by the STUN client for the request and echoed by the server in the response. For indications, it is chosen by the agent sending the indication. It primarily serves to correlate requests with responses, though it also plays a small role

in helping to prevent certain types of attacks. The server also uses the transaction ID as a key to identify each transaction uniquely across all clients. As such, the transaction ID MUST be uniformly and randomly chosen from the interval 0 .. 2**96-1, and SHOULD be cryptographically random. Resends of the same request reuse the same transaction ID, but the client MUST choose a new transaction ID for new transactions unless the new request is bit-wise identical to the previous request and sent from the same transport address to the same IP address. Success and error responses MUST carry the same transaction ID as their corresponding request. When an agent is acting as a STUN server and STUN client on the same port, the transaction IDs in requests sent by the agent have no relationship to the transaction IDs in requests received by the agent.

The message length MUST contain the size, in bytes, of the message not including the 20-byte STUN header. Since all STUN attributes are padded to a multiple of 4 bytes, the last 2 bits of this field are always zero. This provides another way to distinguish STUN packets from packets of other protocols.

Following the STUN fixed portion of the header are zero or more attributes. Each attribute is TLV (Type-Length-Value) encoded. The details of the encoding, and of the attributes themselves are given in Section 14.

6. Base Protocol Procedures

This section defines the base procedures of the STUN protocol. It describes how messages are formed, how they are sent, and how they are processed when they are received. It also defines the detailed processing of the Binding method. Other sections in this document describe optional procedures that a usage may elect to use in certain situations. Other documents may define other extensions to STUN, by adding new methods, new attributes, or new error response codes.

6.1. Forming a Request or an Indication

When formulating a request or indication message, the agent MUST follow the rules in Section 5 when creating the header. In addition, the message class MUST be either "Request" or "Indication" (as appropriate), and the method must be either Binding or some method defined in another document.

The agent then adds any attributes specified by the method or the usage. For example, some usages may specify that the agent use an authentication method (Section 9) or the FINGERPRINT attribute (Section 7).

If the agent is sending a request, it SHOULD add a SOFTWARE attribute to the request. Agents MAY include a SOFTWARE attribute in indications, depending on the method. Extensions to STUN should discuss whether SOFTWARE is useful in new indications.

For the Binding method with no authentication, no attributes are required unless the usage specifies otherwise.

All STUN messages sent over UDP or DTLS-over-UDP [RFC6347] SHOULD be less than the path MTU, if known.

If the path MTU is unknown for UDP, messages SHOULD be the smaller of 576 bytes and the first-hop MTU for IPv4 [RFC1122] and 1280 bytes for IPv6 [RFC2460]. This value corresponds to the overall size of the IP packet. Consequently, for IPv4, the actual STUN message would need to be less than 548 bytes (576 minus 20-byte IP header, minus 8-byte UDP header, assuming no IP options are used).

If the path MTU is unknown for DTLS-over-UDP, the rules described in the previous paragraph need to be adjusted to take into account the size of the (13-byte) DTLS Record header, the MAC size, and the padding size.

If a STUN client needs to send requests that are larger than the MTU, or if an application envisions that a response would be larger than the MTU, then it MUST use SCTP-over-UDP or SCTP-over-DTLS-over-UDP as a transport, unless the purpose of sending an oversized packet is to probe for MTU characteristics (see [RFC5780]).

Adding an ORIGIN attribute to a request, as specified in [I-D.ietf-tram-stun-origin], may increase the size of a request beyond the MTU such that it consequently triggers the use of SCTP-over-UDP or SCTP-over-DTLS-over-UDP as a transport.

6.2. Sending the Request or Indication

The agent then sends the request or indication. This document specifies how to send STUN messages over UDP, TCP, TLS-over-TCP, DTLS-over-UDP, SCTP-over-UDP, or SCTP-over-DTLS-over-UDP; other transport protocols may be added in the future. The STUN usage must specify which transport protocol is used, and how the agent determines the IP address and port of the recipient. Section 8 describes a DNS-based method of determining the IP address and port of a server that a usage may elect to use. STUN may be used with anycast addresses, but only with UDP and in usages where authentication is not used.

At any time, a client MAY have multiple outstanding STUN requests with the same STUN server (that is, multiple transactions in progress, with different transaction IDs). Absent other limits to the rate of new transactions (such as those specified by ICE for connectivity checks or when STUN is run over TCP), a client SHOULD space new parallel transactions to a server by RTO and SHOULD limit itself to ten outstanding transactions to the same server.

Parallel transactions are defined as transactions that can be sent independently of each other. Serial transactions, on the other hand, are a series of transactions that are each dependent on the completion of the previous transaction (e.g., the second transaction of a long term authentication). In contrast to parallel transactions, a client need not space new serial transactions to a server by RTO.

6.2.1. Sending over UDP or DTLS-over-UDP

When running STUN over UDP or STUN over DTLS-over-UDP [RFC7350], it is possible that the STUN message might be dropped by the network. Reliability of STUN request/response transactions is accomplished through retransmissions of the request message by the client application itself. STUN indications are not retransmitted; thus, indication transactions over UDP or DTLS-over-UDP are not reliable.

A client SHOULD retransmit a STUN request message starting with an interval of RTO ("Retransmission TimeOut"), doubling after each retransmission. The RTO is an estimate of the round-trip time (RTT), and is computed as described in RFC 6298 [RFC6298], with two exceptions. First, the initial value for RTO SHOULD be greater than 500 ms. The exception cases for this "SHOULD" are when other mechanisms are used to derive congestion thresholds (such as the ones defined in ICE for fixed rate streams), or when STUN is used in non-Internet environments with known network capacities. In fixed-line access links, a value of 500 ms is RECOMMENDED. Second, the value of RTO SHOULD NOT be rounded up to the nearest second. Rather, a 1 ms accuracy SHOULD be maintained. As with TCP, the usage of Karn’s algorithm is RECOMMENDED [KARN87]. When applied to STUN, it means that RTT estimates SHOULD NOT be computed from STUN transactions that result in the retransmission of a request.

The value for RTO SHOULD be cached by a client after the completion of the transaction, and used as the starting value for RTO for the next transaction to the same server (based on equality of IP address). The value SHOULD be considered stale and discarded after 10 minutes without any transactions to the same server.

Retransmissions continue until a response is received, or until a total of Rc requests have been sent. Rc SHOULD be configurable and SHOULD have a default of 7. If, after the last request, a duration equal to Rm times the RTO has passed without a response (providing ample time to get a response if only this final request actually succeeds), the client SHOULD consider the transaction to have failed. Rm SHOULD be configurable and SHOULD have a default of 16. A STUN transaction over UDP or DTLS-over-UDP is also considered failed if there has been a hard ICMP error [RFC1122]. For example, assuming an RTO of 500ms, requests would be sent at times 0 ms, 500 ms, 1500 ms, 3500 ms, 7500 ms, 15500 ms, and 31500 ms. If the client has not received a response after 39500 ms, the client will consider the transaction to have timed out.

6.2.2. Sending over TCP or TLS-over-TCP

For TCP and TLS-over-TCP [RFC5246], the client opens a TCP connection to the server.

In some usages of STUN, STUN is sent as the only protocol over the TCP connection. In this case, it can be sent without the aid of any additional framing or demultiplexing. In other usages, or with other extensions, it may be multiplexed with other data over a TCP connection. In that case, STUN MUST be run on top of some kind of framing protocol, specified by the usage or extension, which allows for the agent to extract complete STUN messages and complete application layer messages. The STUN service running on the well-known port or ports discovered through the DNS procedures in Section 8 is for STUN alone, and not for STUN multiplexed with other data. Consequently, no framing protocols are used in connections to those servers. When additional framing is utilized, the usage will specify how the client knows to apply it and what port to connect to. For example, in the case of ICE connectivity checks, this information is learned through out-of-band negotiation between client and server.

Reliability of STUN over TCP and TLS-over-TCP is handled by TCP itself, and there are no retransmissions at the STUN protocol level. However, for a request/response transaction, if the client has not received a response by Ti seconds after it sent the SYN to establish the connection, it considers the transaction to have timed out. Ti SHOULD be configurable and SHOULD have a default of 39.5s. This value has been chosen to equalize the TCP and UDP timeouts for the default initial RTO.

In addition, if the client is unable to establish the TCP connection, or the TCP connection is reset or fails before a response is received, any request/response transaction in progress is considered to have failed.

The client MAY send multiple transactions over a single TCP (or TLS-over-TCP) connection, and it MAY send another request before receiving a response to the previous. The client SHOULD keep the connection open until it:

- has no further STUN requests or indications to send over that connection, and
- has no plans to use any resources (such as a mapped address (MAPPED-ADDRESS or XOR-MAPPED-ADDRESS) or relayed address [RFC5766]) that were learned though STUN requests sent over that connection, and
- if multiplexing other application protocols over that port, has finished using that other application, and
- if using that learned port with a remote peer, has established communications with that remote peer, as is required by some TCP NAT traversal techniques (e.g., [RFC6544]).

At the server end, the server SHOULD keep the connection open, and let the client close it, unless the server has determined that the connection has timed out (for example, due to the client disconnecting from the network). Bindings learned by the client will remain valid in intervening NATs only while the connection remains open. Only the client knows how long it needs the binding. The server SHOULD NOT close a connection if a request was received over that connection for which a response was not sent. A server MUST NOT ever open a connection back towards the client in order to send a response. Servers SHOULD follow best practices regarding connection management in cases of overload.

6.2.3. Sending over SCTP-over-UDP or SCTP-over-DTLS-over-UDP

For SCTP-over-UDP [RFC6951] and SCTP-over-DTLS-over-UDP [I-D.ietf-tsvwg-sctp-dtls-encaps], the client opens a Stream Control Transmission Protocol (SCTP) connection to the server.

For some STUN usages, STUN is sent over SCTP as the only protocol over the UDP association. In this case, it can be sent without the aid of any additional demultiplexing. In other usages, or with other extensions, it may be multiplexed with other data over a UDP association. In that case, the SCTP source and destination ports MUST be chosen so the two most significant bits are 0b11.

Reliability of STUN over SCTP-over-UDP and STUN over SCTP-over-DTLS-over-UDP is handled by SCTP itself and there are no retransmissions at the STUN protocol level. However, for a request/response

transaction, if the client has not received a response by Ti seconds after it sent the INIT to establish the connection, it considers the transaction to have timed out. Ti SHOULD be configurable and SHOULD have a default of 39.5s. This value has been chosen to equalize the SCTP-over-UDP, TCP, and UDP timeouts for the default initial RTO.

In addition, if the client is unable to establish the SCTP connection, or the SCTP connection is reset or fails before a response is received, any request/response transaction in progress is considered to have failed.

The client MAY send multiple transactions over a single SCTP (or SCTP-over-DTLS) connection and it MAY send another request before receiving a response to the previous. Each transaction MUST use a different SCTP stream ID. The client SHOULD keep the connection open until it:

- has no further STUN requests or indications to send over that connection, and
- has no plans to use any resources (such as a mapped address (MAPPED-ADDRESS or XOR-MAPPED-ADDRESS) or relayed address [RFC5766]) that were learned through STUN requests sent over that connection, and
- has finished using all corresponding applications if multiplexing other application protocols over that port

When using SCTP-over-UDP, the SCTP source port and destination port MUST be selected so the two most significant bits are set to "1". This permits multiplexing of STUN-over-UDP, STUN-over-SCTP-over-UDP, DTLS, and RTP/RTCP on the same socket.

STUN indications MAY be sent unreliably by using the SCTP extension in [RFC3758], augmented with the policies of [I-D.ietf-tsvwg-sctp-prpolicies]. Each STUN usage MUST specify the conditions under which STUN indications are sent reliably or not, and MUST specify the policy for allocating an SCTP stream ID. The NAT Discovery usage described in this document does not use STUN indications.

At the server end, the server SHOULD keep the connection open and let the client close it unless the server has determined that the connection has timed out (for example, due to the client disconnecting from the network). Bindings learned by the client will remain valid in intervening NATs only while the connection remains open. Only the client knows how long it needs the binding. The server SHOULD NOT close a connection if a request was received over

that connection for which a response was not sent. A server MUST NOT ever open a connection back towards the client in order to send a response. Servers SHOULD follow best practices regarding connection management in cases of overload.

6.2.4. Sending over TLS-over-TCP or DTLS-over-UDP or SCTP-over-DTLS-over-UDP

When STUN is run by itself over TLS-over-TCP or DTLS-over-UDP or SCTP-over-DTLS-over-UDP, the TLS_DHE_RSA_WITH_AES_128_GCM_SHA256 and TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256 cipher suites MUST be implemented and other cipher suites MAY be implemented. Perfect Forward Secrecy (PFS) cipher suites MUST be preferred over non-PFS cipher suites. Cipher suites with known weaknesses, such as those based on (single) DES and RC4, MUST NOT be used. Implementations MUST disable TLS-level compression.

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 or 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 using the following procedure.

STUN clients that are using the mechanism in Section 8, and that have established that all DNS Resource Records from the Source Domain to the Host Name are secure according to DNSsec [RFC4033] (i.e., that the AD bit is set in all the DNS responses) MUST lookup a TLSA Resource Record [RFC6698] for the protocol, port and Host Name selected. If the TLSA Resource Record is secure then the STUN client MUST use it to validate the certificate presented by the STUN server. If there is no TLSA Resource Record or if the Resource Record is not secure, then the client MUST fallback to the validation process defined in Section 3.1 of RFC 2818 [RFC2818].

Alternatively, a client MAY be configured with a set of domains or IP addresses that are trusted. If a certificate is received that identifies one of those trusted domains or IP addresses, the client considers the identity of the server to be verified.

When STUN is multiplexed with other protocols over a TLS-over-TCP connection or a DTLS-over-UDP or a SCTP-over-DTLS-over-UDP association, the mandatory ciphersuites and TLS handling procedures operate as defined by those protocols.

6.3. Receiving a STUN Message

This section specifies the processing of a STUN message. The processing specified here is for STUN messages as defined in this specification; additional rules for backwards compatibility are defined in Section 11. Those additional procedures are optional, and usages can elect to utilize them. First, a set of processing operations is applied that is independent of the class. This is followed by class-specific processing, described in the subsections that follow.

When a STUN agent receives a STUN message, it first checks that the message obeys the rules of Section 5. It checks that the first two bits are 0, that the magic cookie field has the correct value, that the message length is sensible, and that the method value is a supported method. It checks that the message class is allowed for the particular method. If the message class is "Success Response" or "Error Response", the agent checks that the transaction ID matches a transaction that is still in progress. If the FINGERPRINT extension is being used, the agent checks that the FINGERPRINT attribute is present and contains the correct value. If any errors are detected, the message is silently discarded. In the case when STUN is being multiplexed with another protocol, an error may indicate that this is not really a STUN message; in this case, the agent should try to parse the message as a different protocol.

The STUN agent then does any checks that are required by a authentication mechanism that the usage has specified (see Section 9).

Once the authentication checks are done, the STUN agent checks for unknown attributes and known-but-unexpected attributes in the message. Unknown comprehension-optional attributes MUST be ignored by the agent. Known-but-unexpected attributes SHOULD be ignored by the agent. Unknown comprehension-required attributes cause processing that depends on the message class and is described below.

At this point, further processing depends on the message class of the request.

6.3.1. Processing a Request

If the request contains one or more unknown comprehension-required attributes, the server replies with an error response with an error code of 420 (Unknown Attribute), and includes an UNKNOWN-ATTRIBUTES attribute in the response that lists the unknown comprehension-required attributes.

The server then does any additional checking that the method or the specific usage requires. If all the checks succeed, the server formulates a success response as described below.

When run over UDP or DTLS-over-UDP or SCTP-over-UDP or SCTP-over-DTLS-over-UDP, a request received by the server could be the first request of a transaction, or a retransmission. The server MUST respond to retransmissions such that the following property is preserved: if the client receives the response to the retransmission and not the response that was sent to the original request, the overall state on the client and server is identical to the case where only the response to the original retransmission is received, or where both responses are received (in which case the client will use the first). The easiest way to meet this requirement is for the server to remember all transaction IDs received over UDP or DTLS-over-UDP and their corresponding responses in the last 40 seconds. However, this requires the server to hold state, and will be inappropriate for any requests which are not authenticated. Another way is to reprocess the request and recompute the response. The latter technique MUST only be applied to requests that are idempotent (a request is considered idempotent when the same request can be safely repeated without impacting the overall state of the system) and result in the same success response for the same request. The Binding method is considered to be idempotent. Note that there are certain rare network events that could cause the reflexive transport address value to change, resulting in a different mapped address in different success responses. Extensions to STUN MUST discuss the implications of request retransmissions on servers that do not store transaction state.

6.3.1.1. Forming a Success or Error Response

When forming the response (success or error), the server follows the rules of Section 6. The method of the response is the same as that of the request, and the message class is either "Success Response" or "Error Response".

For an error response, the server MUST add an ERROR-CODE attribute containing the error code specified in the processing above. The reason phrase is not fixed, but SHOULD be something suitable for the error code. For certain errors, additional attributes are added to the message. These attributes are spelled out in the description where the error code is specified. For example, for an error code of 420 (Unknown Attribute), the server MUST include an UNKNOWN-ATTRIBUTES attribute. Certain authentication errors also cause attributes to be added (see Section 9). Extensions may define other errors and/or additional attributes to add in error cases.

If the server authenticated the request using an authentication mechanism, then the server SHOULD add the appropriate authentication attributes to the response (see Section 9).

The server also adds any attributes required by the specific method or usage. In addition, the server SHOULD add a SOFTWARE attribute to the message.

For the Binding method, no additional checking is required unless the usage specifies otherwise. When forming the success response, the server adds a XOR-MAPPED-ADDRESS attribute to the response, where the contents of the attribute are the source transport address of the request message. For UDP and DTLS-over-UDP, this is the source IP address and source UDP port of the request message. For TCP and TLS-over-TCP, this is the source IP address and source TCP port of the TCP connection as seen by the server.

6.3.1.2. Sending the Success or Error Response

The response (success or error) is sent over the same transport as the request was received on. If the request was received over UDP or DTLS-over-UDP, the destination IP address and port of the response are the source IP address and port of the received request message, and the source IP address and port of the response are equal to the destination IP address and port of the received request message. If the request was received over TCP or TLS-over-TCP, the response is sent back on the same TCP connection as the request was received on.

6.3.2. Processing an Indication

If the indication contains unknown comprehension-required attributes, the indication is discarded and processing ceases.

The agent then does any additional checking that the method or the specific usage requires. If all the checks succeed, the agent then processes the indication. No response is generated for an indication.

For the Binding method, no additional checking or processing is required, unless the usage specifies otherwise. The mere receipt of the message by the agent has refreshed the "bindings" in the intervening NATs.

Since indications are not re-transmitted over UDP or DTLS-over-UDP (unlike requests), there is no need to handle re-transmissions of indications at the sending agent.

6.3.3. Processing a Success Response

If the success response contains unknown comprehension-required attributes, the response is discarded and the transaction is considered to have failed.

The client then does any additional checking that the method or the specific usage requires. If all the checks succeed, the client then processes the success response.

For the Binding method, the client checks that the XOR-MAPPED-ADDRESS attribute is present in the response. The client checks the address family specified. If it is an unsupported address family, the attribute SHOULD be ignored. If it is an unexpected but supported address family (for example, the Binding transaction was sent over IPv4, but the address family specified is IPv6), then the client MAY accept and use the value.

6.3.4. Processing an Error Response

If the error response contains unknown comprehension-required attributes, or if the error response does not contain an ERROR-CODE attribute, then the transaction is simply considered to have failed.

The client then does any processing specified by the authentication mechanism (see Section 9). This may result in a new transaction attempt.

The processing at this point depends on the error code, the method, and the usage; the following are the default rules:

- If the error code is 300 through 399, the client SHOULD consider the transaction as failed unless the ALTERNATE-SERVER extension is being used. See Section 10.
- If the error code is 400 through 499, the client declares the transaction failed; in the case of 420 (Unknown Attribute), the response should contain a UNKNOWN-ATTRIBUTES attribute that gives additional information.
- If the error code is 500 through 599, the client MAY resend the request; clients that do so MUST limit the number of times they do this.

Any other error code causes the client to consider the transaction failed.

7. FINGERPRINT Mechanism

This section describes an optional mechanism for STUN that aids in distinguishing STUN messages from packets of other protocols when the two are multiplexed on the same transport address. This mechanism is optional, and a STUN usage must describe if and when it is used. The FINGERPRINT mechanism is not backwards compatible with RFC3489, and cannot be used in environments where such compatibility is required.

In some usages, STUN messages are multiplexed on the same transport address as other protocols, such as the Real Time Transport Protocol (RTP). In order to apply the processing described in Section 6, STUN messages must first be separated from the application packets.

Section 5 describes three fixed fields in the STUN header that can be used for this purpose. However, in some cases, these three fixed fields may not be sufficient.

When the FINGERPRINT extension is used, an agent includes the FINGERPRINT attribute in messages it sends to another agent. Section 14.6 describes the placement and value of this attribute.

When the agent receives what it believes is a STUN message, then, in addition to other basic checks, the agent also checks that the message contains a FINGERPRINT attribute and that the attribute contains the correct value. Section 6.3 describes when in the overall processing of a STUN message the FINGERPRINT check is performed. This additional check helps the agent detect messages of other protocols that might otherwise seem to be STUN messages.

8. DNS Discovery of a Server

This section describes an optional procedure for STUN that allows a client to use DNS to determine the IP address and port of a server. A STUN usage must describe if and when this extension is used. To use this procedure, the client must know a STUN URI [RFC7064]; the usage must also describe how the client obtains this URI. Hard-coding a STUN URI into software is NOT RECOMMENDED in case the domain name is lost or needs to change for legal or other reasons.

When a client wishes to locate a STUN server on the public Internet that accepts Binding request/response transactions, the STUN URI scheme is "stun". When it wishes to locate a STUN server that accepts Binding request/response transactions over a TLS, or DTLS, or SCTP-over-DTLS session, the URI scheme is "stuns".

The syntax of the "stun" and "stuns" URIs are defined in Section 3.1 of [RFC7064]. STUN usages MAY define additional URI schemes.

8.1. STUN URI Scheme Semantics

If the <host> part contains an IP address, then this IP address is used directly to contact the server. A "stuns" URI containing an IP address MUST be rejected, unless the domain name is provided by the same mechanism that provided the STUN URI, and that domain name can be passed to the verification code.

If the URI does not contain an IP address, the domain name contained in the <host> part is resolved to a transport address using the SRV procedures specified in [RFC2782]. The DNS SRV service name is the content of the <host> part. The protocol in the SRV lookup is the transport protocol the client will run STUN over: "udp" for UDP, "tcp" for TCP, and "sctp-udp" for SCTP-over-UDP.

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 is sorted and then tried. However, RFC 2782 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. When following these procedures, if the STUN transaction times out without receipt of a response, the client SHOULD retry the request to the next server in the ordered defined by RFC 2782. Such a retry is only possible for request/response transmissions, since indication transactions generate no response or timeout.

The default port for STUN requests is 3478, for both TCP and UDP. The default port for STUN over TLS and STUN over DTLS requests is 5349. The default port for STUN over SCTP-over-UDP requests is XXXX. The default port for STUN over SCTP-over-DTLS-over-UDP requests is XXXX. Servers can run STUN over DTLS on the same port as STUN over UDP if the server software supports determining whether the initial message is a DTLS or STUN message. Servers can run STUN over TLS on the same port as STUN over TCP if the server software supports determining whether the initial message is a TLS or STUN message.

Administrators of STUN servers SHOULD use these ports in their SRV records for UDP and TCP. In all cases, the port in DNS MUST reflect the one on which the server is listening.

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, the client connects to one of the IP addresses using the default STUN over TLS port. For usages that require DTLS, the client connects to one of the IP addresses using the default STUN over DTLS port.

9. Authentication and Message-Integrity Mechanisms

This section defines two mechanisms for STUN that a client and server can use to provide authentication and message integrity; these two mechanisms are known as the short-term credential mechanism and the long-term credential mechanism. These two mechanisms are optional, and each usage must specify if and when these mechanisms are used. Consequently, both clients and servers will know which mechanism (if any) to follow based on knowledge of which usage applies. For example, a STUN server on the public Internet supporting ICE would have no authentication, whereas the STUN server functionality in an agent supporting connectivity checks would utilize short-term credentials. An overview of these two mechanisms is given in Section 2.

Each mechanism specifies the additional processing required to use that mechanism, extending the processing specified in Section 6. The additional processing occurs in three different places: when forming a message, when receiving a message immediately after the basic checks have been performed, and when doing the detailed processing of error responses.

9.1. Short-Term Credential Mechanism

The short-term credential mechanism assumes that, prior to the STUN transaction, the client and server have used some other protocol to exchange a credential in the form of a username and password. This credential is time-limited. The time limit is defined by the usage. As an example, in the ICE usage [RFC5245], the two endpoints use out-of-band signaling to agree on a username and password, and this username and password are applicable for the duration of the media session.

This credential is used to form a message-integrity check in each request and in many responses. There is no challenge and response as in the long-term mechanism; consequently, replay is prevented by virtue of the time-limited nature of the credential.

9.1.1. HMAC Key

For short-term credentials the HMAC key is defined as follow:

```
key = SASLprep(password)
```

where SASLprep() is defined in RFC 4013 [RFC4013].

9.1.2. Forming a Request or Indication

For a request or indication message, the agent MUST include the USERNAME, MESSAGE-INTEGRITY2, and MESSAGE-INTEGRITY attributes in the message. The HMAC for the MESSAGE-INTEGRITY attribute is computed as described in Section 14.4 and the HMAC for the MESSAGE-INTEGRITY2 attributes is computed as described in Section 14.5. Note that the password is never included in the request or indication.

9.1.3. Receiving a Request or Indication

After the agent has done the basic processing of a message, the agent performs the checks listed below in order specified:

- If the message does not contain 1) a MESSAGE-INTEGRITY or a MESSAGE-INTEGRITY2 attribute and 2) a USERNAME attribute:
  - If the message is a request, the server MUST reject the request with an error response. This response MUST use an error code of 400 (Bad Request).
  - If the message is an indication, the agent MUST silently discard the indication.
- If the USERNAME does not contain a username value currently valid within the server:
  - If the message is a request, the server MUST reject the request with an error response. This response MUST use an error code of 401 (Unauthorized).
  - If the message is an indication, the agent MUST silently discard the indication.
- If the MESSAGE-INTEGRITY2 attribute is present compute the value for the message integrity as described in Section 14.5, using the password associated with the username. If the MESSAGE-INTEGRITY2 attribute is not present, and using the same password, compute the value for the message integrity as described in Section 14.4. If the resulting value does not match the contents of the MESSAGE-INTEGRITY2 attribute or the MESSAGE-INTEGRITY attribute:
  - If the message is a request, the server MUST reject the request with an error response. This response MUST use an error code of 401 (Unauthorized).
  - If the message is an indication, the agent MUST silently discard the indication.

If these checks pass, the agent continues to process the request or indication. Any response generated by a server to a request that contains a MESSAGE-INTEGRITY2 attribute MUST include the MESSAGE-INTEGRITY2 attribute, computed using the password utilized to authenticate the request. Any response generated by a server to a request that contains only a MESSAGE-INTEGRITY attribute MUST include the MESSAGE-INTEGRITY attribute, computed using the password utilized to authenticate the request. The response MUST NOT contain the USERNAME attribute.

If any of the checks fail, a server MUST NOT include a MESSAGE-INTEGRITY2, MESSAGE-INTEGRITY, or USERNAME attribute in the error response. This is because, in these failure cases, the server cannot determine the shared secret necessary to compute the MESSAGE-INTEGRITY2 or MESSAGE-INTEGRITY attributes.

9.1.4. Receiving a Response

The client looks for the MESSAGE-INTEGRITY2 or the MESSAGE-INTEGRITY attribute in the response. If present, the client computes the message integrity over the response as defined in Section 14.4 or Section 14.5, using the same password it utilized for the request. If the resulting value matches the contents of the MESSAGE-INTEGRITY or MESSAGE-INTEGRITY2 attribute, the response is considered authenticated. If the value does not match, or if both MESSAGE-INTEGRITY and MESSAGE-INTEGRITY2 were absent, the response MUST be discarded, as if it was never received. This means that retransmits, if applicable, will continue.

9.1.5. Sending Subsequent Requests

A client sending subsequent requests to the same server a MAY choose to send only the MESSAGE-INTEGRITY2 or the MESSAGE-INTEGRITY attribute depending upon the attribute that was received in the response to the initial request.

9.2. Long-Term Credential Mechanism

The long-term credential mechanism relies on a long-term credential, in the form of a username and password that are shared between client and server. The credential is considered long-term since it is assumed that it is provisioned for a user, and remains in effect until the user is no longer a subscriber of the system, or is changed. This is basically a traditional "log-in" username and password given to users.

Because these usernames and passwords are expected to be valid for extended periods of time, replay prevention is provided in the form

of a digest challenge. In this mechanism, the client initially sends a request, without offering any credentials or any integrity checks. The server rejects this request, providing the user a realm (used to guide the user or agent in selection of a username and password) and a nonce. The nonce provides the replay protection. It is a cookie, selected by the server, and encoded in such a way as to indicate a duration of validity or client identity from which it is valid. The client retries the request, this time including its username and the realm, and echoing the nonce provided by the server. The client also includes a message-integrity, which provides an HMAC over the entire request, including the nonce. The server validates the nonce and checks the message integrity. If they match, the request is authenticated. If the nonce is no longer valid, it is considered "stale", and the server rejects the request, providing a new nonce.

In subsequent requests to the same server, the client reuses the nonce, username, realm, and password it used previously. In this way, subsequent requests are not rejected until the nonce becomes invalid by the server, in which case the rejection provides a new nonce to the client.

Note that the long-term credential mechanism cannot be used to protect indications, since indications cannot be challenged. Usages utilizing indications must either use a short-term credential or omit authentication and message integrity for them.

Since the long-term credential mechanism is susceptible to offline dictionary attacks, deployments SHOULD utilize passwords that are difficult to guess. In cases where the credentials are not entered by the user, but are rather placed on a client device during device provisioning, the password SHOULD have at least 128 bits of randomness. In cases where the credentials are entered by the user, they should follow best current practices around password structure.

9.2.1. HMAC Key

For long-term credentials that do not use a different algorithm, as specified by the PASSWORD-ALGORITHM attribute, the key is 16 bytes:

```
key = MD5(username ":" realm ":" SASLprep(password))
```

Where MD5 is defined in RFC 1321 [RFC1321] and SASLprep() is defined in RFC 4013 [RFC4013].

The 16-byte key 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, as taken from the USERNAME attribute (in which case SASLprep has already been applied); (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 and after processing using SASLprep. For example, if the username was 'user', the realm was 'realm', and the password was 'pass', then the 16-byte HMAC key would be the result of performing an MD5 hash on the string 'user:realm:pass', the resulting hash being 0x8493fbc53ba582fb4c044c456bdc40eb.

The structure of the key when used with long-term credentials facilitates deployment in systems that also utilize SIP. Typically, SIP systems utilizing SIP's digest authentication mechanism do not actually store the password in the database. Rather, they store a value called H(A1), which is equal to the key defined above.

When a PASSWORD-ALGORITHM is used, the key length and algorithm to use are described in Section 17.4.1.

9.2.2. Forming a Request

There are two cases when forming a request. In the first case, this is the first request from the client to the server (as identified by its IP address and port). In the second case, the client is submitting a subsequent request once a previous request/response transaction has completed successfully. Forming a request as a consequence of a 401 or 438 error response is covered in Section 9.2.4 and is not considered a "subsequent request" and thus does not utilize the rules described in Section 9.2.2.2.

9.2.2.1. First Request

If the client has not completed a successful request/response transaction with the server (as identified by hostname, if the DNS procedures of Section 8 are used, else IP address if not), it SHOULD omit the USERNAME, MESSAGE-INTEGRITY, MESSAGE-INTEGRITY2, REALM, NONCE, PASSWORD-ALGORITHMS, and PASSWORD-ALGORITHM attributes. In other words, the very first request is sent as if there were no authentication or message integrity applied.

9.2.2.2. Subsequent Requests

Once a request/response transaction has completed successfully, the client will have been presented a realm and nonce by the server, and selected a username and password with which it authenticated. The client SHOULD cache the username, password, realm, and nonce for subsequent communications with the server. When the client sends a subsequent request, it SHOULD include the USERNAME, REALM, NONCE, and PASSWORD-ALGORITHM attributes with these cached values. It SHOULD include a MESSAGE-INTEGRITY attribute or a MESSAGE-INTEGRITY2

attribute, computed as described in Section 14.4 and Section 14.5 using the cached password. The choice between the two attributes depends on the attribute received in the response to the first request.

9.2.3. Receiving a Request

After the server has done the basic processing of a request, it performs the checks listed below in the order specified:

- If the message does not contain a MESSAGE-INTEGRITY or MESSAGE-INTEGRITY2 attribute, the server MUST generate an error response with an error code of 401 (Unauthorized). This response MUST include a REALM value. It is RECOMMENDED that the REALM value be the domain name of the provider of the STUN server. The response MUST include a NONCE, selected by the server. The server MUST ensure that the same NONCE cannot be selected for clients that use different IP addresses and/or different ports. The server MAY support alternate password algorithms, in which case it can list them in preferential order in a PASSWORD-ALGORITHMS attribute. If the server adds a PASSWORD-ALGORITHMS attribute it MUST prepend the NONCE attribute value with the character string "obMatJos2". The response SHOULD NOT contain a USERNAME, MESSAGE-INTEGRITY or MESSAGE-INTEGRITY2 attribute.

- If the message contains a MESSAGE-INTEGRITY or a MESSAGE-INTEGRITY2 attribute, but is missing the USERNAME, REALM, or NONCE attribute, the server MUST generate an error response with an error code of 400 (Bad Request). This response SHOULD NOT include a USERNAME, NONCE, REALM, MESSAGE-INTEGRITY or MESSAGE-INTEGRITY2 attribute.

- If the NONCE attribute starts with the value "obMatJos2" but the PASSWORD-ALGORITHMS attribute is not present or is not identical to the PASSWORD-ALGORITHMS attribute sent in the response, the server MUST generate an error response with an error code of 400 (Bad Request). This response SHOULD NOT include a USERNAME, NONCE, REALM, MESSAGE-INTEGRITY, or MESSAGE-INTEGRITY2 attribute.

- If the NONCE is no longer valid, the server MUST generate an error response with an error code of 438 (Stale Nonce). This response MUST include NONCE and REALM attributes and SHOULD NOT include the USERNAME, MESSAGE-INTEGRITY, or MESSAGE-INTEGRITY2 attribute. Servers can invalidate nonces in order to provide additional security. See Section 4.3 of [RFC2617] for guidelines.

- If the username in the USERNAME attribute is not valid, the server MUST generate an error response with an error code of 401

(Unauthorized). This response MUST include a REALM value. It is RECOMMENDED that the REALM value be the domain name of the provider of the STUN server. The response MUST include a NONCE, selected by the server. The response SHOULD NOT contain a USERNAME, MESSAGE-INTEGRITY or MESSAGE-INTEGRITY2 attribute.

- o If the MESSAGE-INTEGRITY2 attribute is present compute the value for the message integrity as described in Section 14.5, using the password associated with the username. Else, using the same password, compute the value for the message integrity as described in Section 14.4. If the resulting value does not match the contents of the MESSAGE-INTEGRITY attribute or the MESSAGE-INTEGRITY2 attribute, the server MUST reject the request with an error response. This response MUST use an error code of 401 (Unauthorized). It MUST include REALM and NONCE attributes and SHOULD NOT include the USERNAME, MESSAGE-INTEGRITY, or MESSAGE-INTEGRITY2 attribute.

If these checks pass, the server continues to process the request. Any response generated by the server (excepting the cases described above) MUST include both the MESSAGE-INTEGRITY and MESSAGE-INTEGRITY2 attributes, computed using the username and password utilized to authenticate the request. The REALM, NONCE, and USERNAME attributes SHOULD NOT be included.

9.2.4. Receiving a Response

If the response is an error response with an error code of 401 (Unauthorized), the client MUST test if the NONCE attribute value starts with the character string "obMatJos2". If the test succeeds and no PASSWORD-ALGORITHMS attribute is present, then the client MUST NOT retry the request with a new transaction.

If the response is an error response with an error code of 401 (Unauthorized), the client SHOULD retry the request with a new transaction. This request MUST contain a USERNAME, determined by the client as the appropriate username for the REALM from the error response. The request MUST contain the REALM, copied from the error response. The request MUST contain the NONCE, copied from the error response. If the response contains a PASSWORD-ALGORITHMS attribute, the request MUST contain the PASSWORD-ALGORITHMS attribute with the same content. If the response contains a PASSWORD-ALGORITHMS attribute, and this attribute contains at least one algorithm that is supported by the client then the request MUST contain a PASSWORD-ALGORITHM attribute with the first algorithm supported on the list. if the response contains a MESSAGE-INTEGRITY2 attribute then the request MUST contain a MESSAGE-INTEGRITY2 attribute, computed using the password associated with the username in the USERNAME attribute.

Else the request MUST contain the MESSAGE-INTEGRITY attribute, computed using the password associated with the username in the USERNAME attribute. The client MUST NOT perform this retry if it is not changing the USERNAME or REALM or its associated password, from the previous attempt.

If the response is an error response with an error code of 438 (Stale Nonce), the client MUST retry the request, using the new NONCE attribute supplied in the 438 (Stale Nonce) response. This retry MUST also include the USERNAME, REALM and either the MESSAGE-INTEGRITY or MESSAGE-INTEGRITY2 attributes.

The client looks for the MESSAGE-INTEGRITY or MESSAGE-INTEGRITY2 attribute in the response (either success or failure). If present, the client computes the message integrity over the response as defined in Section 14.4 or Section 14.5, using the same password it utilized for the request. If the resulting value matches the contents of the MESSAGE-INTEGRITY or MESSAGE-INTEGRITY2 attribute, the response is considered authenticated. If the value does not match, or if both MESSAGE-INTEGRITY and MESSAGE-INTEGRITY2 were absent, the response MUST be discarded, as if it was never received. This means that retransmits, if applicable, will continue.

10. ALTERNATE-SERVER Mechanism

This section describes a mechanism in STUN that allows a server to redirect a client to another server. This extension is optional, and a usage must define if and when this extension is used.

A server using this extension redirects a client to another server by replying to a request message with an error response message with an error code of 300 (Try Alternate). The server MUST include an ALTERNATE-SERVER attribute in the error response. The error response message MAY be authenticated; however, there are uses cases for ALTERNATE-SERVER where authentication of the response is not possible or practical. If the transaction uses TLS or DTLS and if the transaction is authenticated by a MESSAGE-INTEGRITY2 attribute and if the server wants to redirect to a server that uses a different certificate, then it MUST include an ALTERNATE-DOMAIN attribute containing the subjectAltName of that certificate.

A client using this extension handles a 300 (Try Alternate) error code as follows. The client looks for an ALTERNATE-SERVER attribute in the error response. If one is found, then the client considers the current transaction as failed, and reattempts the request with the server specified in the attribute, using the same transport protocol used for the previous request. That request, if authenticated, MUST utilize the same credentials that the client

would have used in the request to the server that performed the redirection. If the transport protocol uses TLS or DTLS, then the client looks for an ALTERNATE-DOMAIN attribute. If the attribute is found, the domain MUST be used to validate the cartificate. If the attribute is not found, the same domain that was used for the original request MUST be used to validate the certificate. If the client has been redirected to a server on which it has already tried this request within the last five minutes, it MUST ignore the redirection and consider the transaction to have failed. This prevents infinite ping-ponging between servers in case of redirection loops.

11. Backwards Compatibility with RFC 5389

In addition to the backward compatibility already described in Section 12 of [RFC5389], DTLS MUST NOT be used with RFC 3489 STUN [RFC3489] (also referred to as "classic STUN"). Any STUN request or indication without the magic cookie (see Section 6 of [RFC5389]) over DTLS MUST always result in an error.

12. Basic Server Behavior

This section defines the behavior of a basic, stand-alone STUN server. A basic STUN server provides clients with server reflexive transport addresses by receiving and replying to STUN Binding requests.

The STUN server MUST support the Binding method. It SHOULD NOT utilize the short-term or long-term credential mechanism. This is because the work involved in authenticating the request is more than the work in simply processing it. It SHOULD NOT utilize the ALTERNATE-SERVER mechanism for the same reason. It MUST support UDP and TCP. It MAY support STUN over TCP/TLS or STUN over UDP/DTLS; however, DTLS and TLS provide minimal security benefits in this basic mode of operation. It MAY utilize the FINGERPRINT mechanism but MUST NOT require it. Since the stand-alone server only runs STUN, FINGERPRINT provides no benefit. Requiring it would break compatibility with RFC 3489, and such compatibility is desirable in a stand-alone server. Stand-alone STUN servers SHOULD support backwards compatibility with [RFC3489] clients, as described in Section 11.

It is RECOMMENDED that administrators of STUN servers provide DNS entries for those servers as described in Section 8.

A basic STUN server is not a solution for NAT traversal by itself. However, it can be utilized as part of a solution through STUN usages. This is discussed further in Section 13.

13. STUN Usages

STUN by itself is not a solution to the NAT traversal problem. Rather, STUN defines a tool that can be used inside a larger solution. The term "STUN usage" is used for any solution that uses STUN as a component.

At the time of writing, three STUN usages are defined: Interactive Connectivity Establishment (ICE) [RFC5245], Client-initiated connections for SIP [RFC5626], and NAT Behavior Discovery [RFC5780]. Other STUN usages may be defined in the future.

A STUN usage defines how STUN is actually utilized -- when to send requests, what to do with the responses, and which optional procedures defined here (or in an extension to STUN) are to be used. A usage would also define:

- Which STUN methods are used.
- What transports are used. If DTLS-over-UDP is used then implementing the denial-of-service countermeasure described in Section 4.2.1 of [RFC6347] is mandatory.
- What authentication and message-integrity mechanisms are used.
- The considerations around manual vs. automatic key derivation for the integrity mechanism, as discussed in [RFC4107].
- What mechanisms are used to distinguish STUN messages from other messages. When STUN is run over TCP, a framing mechanism may be required.
- How a STUN client determines the IP address and port of the STUN server.
- Whether backwards compatibility to RFC 3489 is required.
- What optional attributes defined here (such as FINGERPRINT and ALTERNATE-SERVER) or in other extensions are required.

In addition, any STUN usage must consider the security implications of using STUN in that usage. A number of attacks against STUN are known (see the Security Considerations section in this document), and any usage must consider how these attacks can be thwarted or mitigated.

Finally, a usage must consider whether its usage of STUN is an example of the Unilateral Self-Address Fixing approach to NAT

traversal, and if so, address the questions raised in RFC 3424 [RFC3424].

14. STUN Attributes

After the STUN header are zero or more attributes. Each attribute MUST be TLV encoded, with a 16-bit type, 16-bit length, and value. Each STUN attribute MUST end on a 32-bit boundary. As mentioned above, all fields in an attribute are transmitted most significant bit first.

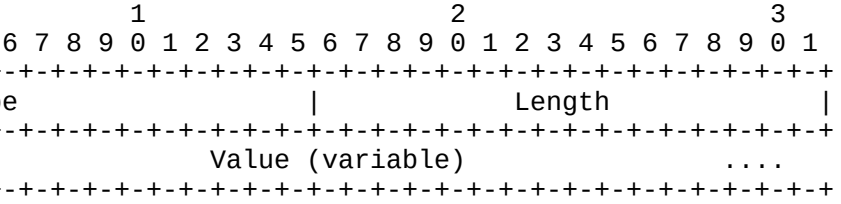


Figure 4: Format of STUN Attributes

The value in the length field MUST contain the length of the Value part of the attribute, prior to padding, 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 its value contains a multiple of 4 bytes. The padding bits are ignored, and may be any value.

Any attribute type MAY appear more than once in a STUN message. Unless specified otherwise, the order of appearance is significant: only the first occurrence needs to be processed by a receiver, and any duplicates MAY be ignored by a receiver.

To allow future revisions of this specification to add new attributes if needed, the attribute space is divided into two ranges. Attributes with type values between 0x0000 and 0x7FFF are comprehension-required attributes, which means that the STUN agent cannot successfully process the message unless it understands the attribute. Attributes with type values between 0x8000 and 0xFFFF are comprehension-optional attributes, which means that those attributes can be ignored by the STUN agent if it does not understand them.

The set of STUN attribute types is maintained by IANA. The initial set defined by this specification is found in Section 17.2.

The rest of this section describes the format of the various attributes defined in this specification.

14.1. MAPPED-ADDRESS

The MAPPED-ADDRESS attribute indicates a reflexive transport address of the client. It consists of an 8-bit address family and a 16-bit port, followed by a fixed-length value representing the IP address. If the address family is IPv4, the address MUST be 32 bits. If the address family is IPv6, the address MUST be 128 bits. All fields must be in network byte order.

The format of the MAPPED-ADDRESS attribute is:

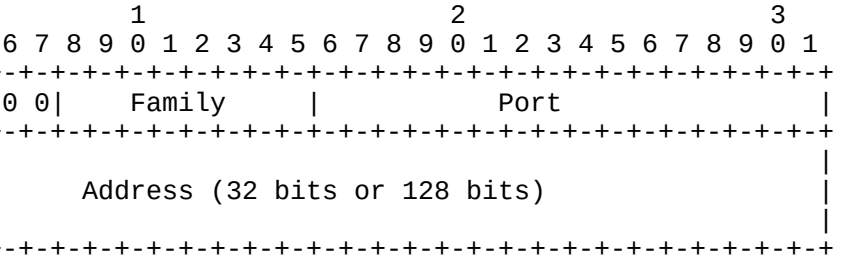


Figure 5: Format of MAPPED-ADDRESS Attribute

The address family can take on the following values:

0x01:IPv4
0x02:IPv6

The first 8 bits of the MAPPED-ADDRESS MUST be set to 0 and MUST be ignored by receivers. These bits are present for aligning parameters on natural 32-bit boundaries.

This attribute is used only by servers for achieving backwards compatibility with RFC 3489 [RFC3489] clients.

14.2. XOR-MAPPED-ADDRESS

The XOR-MAPPED-ADDRESS attribute is identical to the MAPPED-ADDRESS attribute, except that the reflexive transport address is obfuscated through the XOR function.

The format of the XOR-MAPPED-ADDRESS is:

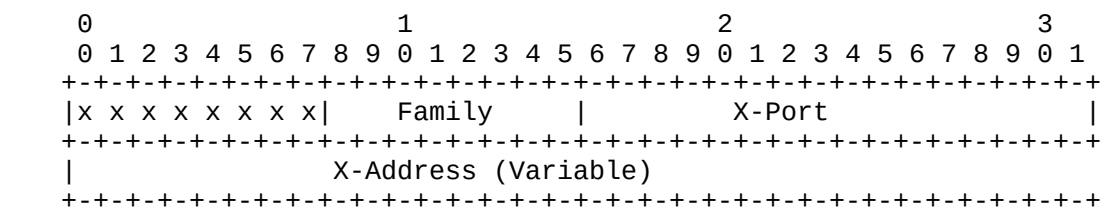


Figure 6: Format of XOR-MAPPED-ADDRESS Attribute

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

X-Port is computed by taking the mapped port in host byte order, XOR'ing it with the most significant 16 bits of the magic cookie, and then the converting the result to network byte order. If the IP address family is IPv4, X-Address is computed by taking the mapped IP address in host byte order, XOR'ing it with the magic cookie, and converting the result to network byte order. If the IP address family is IPv6, X-Address is computed by taking the mapped IP address in host byte order, XOR'ing it with the concatenation of the magic cookie and the 96-bit transaction ID, and converting the result to network byte order.

The rules for encoding and processing the first 8 bits of the attribute's value, the rules for handling multiple occurrences of the attribute, and the rules for processing address families are the same as for MAPPED-ADDRESS.

Note: XOR-MAPPED-ADDRESS and MAPPED-ADDRESS differ only in their encoding of the transport address. The former encodes the transport address by exclusive-or'ing it with the magic cookie. The latter encodes it 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.

14.3. USERNAME

The USERNAME attribute is used for message integrity. It identifies the username and password combination used in the message-integrity check.

The value of USERNAME is a variable-length value. It MUST contain a UTF-8 [RFC3629] encoded sequence of less than 513 bytes, and MUST have been processed using SASLprep [RFC4013].

14.4. MESSAGE-INTEGRITY

The MESSAGE-INTEGRITY attribute contains an HMAC-SHA1 [RFC2104] 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. With the exception of the MESSAGE-INTEGRITY2 and FINGERPRINT attributes, which appear after MESSAGE-INTEGRITY, agents MUST ignore all other attributes that follow MESSAGE-INTEGRITY.

The key for the HMAC depends on which credential mechanism is in use. Section 9.1.1 defines the key for the short-term credential mechanism and Section 9.2.1 defines the key for the long-term credential mechanism. Other credential mechanisms MUST define the key that is used for the HMAC.

Based on the rules above, the hash used to construct MESSAGE-INTEGRITY includes the length field from the STUN message header. Prior to performing the hash, the MESSAGE-INTEGRITY attribute MUST be inserted into the message (with dummy content). The length MUST then be set to point to the length of the message up to, and including, the MESSAGE-INTEGRITY attribute itself, but excluding any attributes after it. Once the computation is performed, the value of the MESSAGE-INTEGRITY attribute can be filled in, and the value of the length in the STUN header can be set to its correct value -- the length of the entire message. Similarly, when validating the MESSAGE-INTEGRITY, the length field should be adjusted to point to the end of the MESSAGE-INTEGRITY attribute prior to calculating the HMAC. Such adjustment is necessary when attributes, such as FINGERPRINT, appear after MESSAGE-INTEGRITY.

14.5. MESSAGE-INTEGRITY2

The MESSAGE-INTEGRITY2 attribute contains an HMAC-SHA-256 [RFC2104] of the STUN message. The MESSAGE-INTEGRITY2 attribute can be present in any STUN message type. Since it uses the SHA-256 hash, the HMAC will be 32 bytes. The text used as input to HMAC is the STUN message, including the header, up to and including the attribute preceding the MESSAGE-INTEGRITY2 attribute. With the exception of the FINGERPRINT attribute, which appears after MESSAGE-INTEGRITY2, agents MUST ignore all other attributes that follow MESSAGE-INTEGRITY2.

The key for the HMAC depends on which credential mechanism is in use. Section 9.1.1 defines the key for the short-term credential mechanism and Section 9.2.1 defines the key for the long-term credential mechanism. Other credential mechanism MUST define the key that is used for the HMAC.

Based on the rules above, the hash used to construct MESSAGE-INTEGRITY2 includes the length field from the STUN message header. Prior to performing the hash, the MESSAGE-INTEGRITY2 attribute MUST be inserted into the message (with dummy content). The length MUST then be set to point to the length of the message up to, and including, the MESSAGE-INTEGRITY2 attribute itself, but excluding any attributes after it. Once the computation is performed, the value of the MESSAGE-INTEGRITY2 attribute can be filled in, and the value of the length in the STUN header can be set to its correct value -- the length of the entire message. Similarly, when validating the MESSAGE-INTEGRITY2, the length field should be adjusted to point to the end of the MESSAGE-INTEGRITY2 attribute prior to calculating the HMAC. Such adjustment is necessary when attributes, such as FINGERPRINT, appear after MESSAGE-INTEGRITY2.

14.6. FINGERPRINT

The FINGERPRINT attribute MAY be present in all STUN messages. The value of the attribute is computed as the CRC-32 of the STUN message up to (but excluding) the FINGERPRINT attribute itself, XOR'ed with the 32-bit value 0x5354554e (the XOR helps in cases where an application packet is also using CRC-32 in it). The 32-bit CRC is the one defined in ITU V.42 [ITU.V42.2002], which has a generator polynomial of x32+x26+x23+x22+x16+x12+x11+x10+x8+x7+x5+x4+x2+x+1. When present, the FINGERPRINT attribute MUST be the last attribute in the message, and thus will appear after MESSAGE-INTEGRITY.

The FINGERPRINT attribute can aid in distinguishing STUN packets from packets of other protocols. See Section 7.

As with MESSAGE-INTEGRITY, the CRC used in the FINGERPRINT attribute covers the length field from the STUN message header. Therefore, this value must be correct and include the CRC attribute as part of the message length, prior to computation of the CRC. When using the FINGERPRINT attribute in a message, the attribute is first placed into the message with a dummy value, then the CRC is computed, and then the value of the attribute is updated. If the MESSAGE-INTEGRITY attribute is also present, then it must be present with the correct message-integrity value before the CRC is computed, since the CRC is done over the value of the MESSAGE-INTEGRITY attribute as well.

14.7. ERROR-CODE

The ERROR-CODE attribute is used in error response messages. It contains a numeric error code value in the range of 300 to 699 plus a textual reason phrase encoded in UTF-8 [RFC3629], and is consistent in its code assignments and semantics with SIP [RFC3261] and HTTP [RFC2616]. The reason phrase is meant for user consumption, and can be anything appropriate for the error code. Recommended reason phrases for the defined error codes are included in the IANA registry for error codes. The reason phrase MUST be a UTF-8 [RFC3629] encoded sequence of less than 128 characters (which can be as long as 763 bytes).

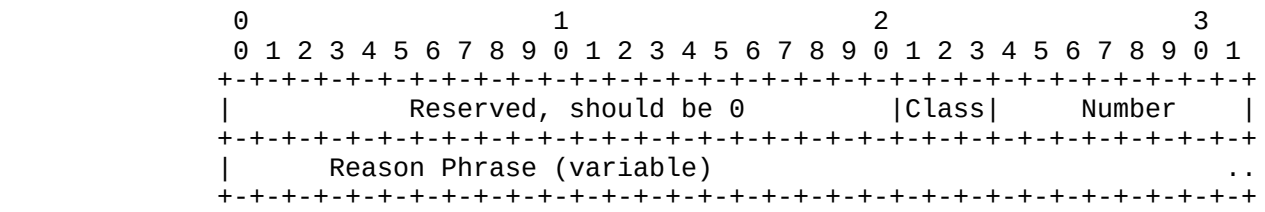


Figure 7: ERROR-CODE Attribute

To facilitate processing, the class of the error code (the hundreds digit) is encoded separately from the rest of the code, as shown in Figure 7.

The Reserved bits SHOULD be 0, and are for alignment on 32-bit boundaries. Receivers MUST ignore these bits. The Class represents the hundreds digit of the error code. The value MUST be between 3 and 6. The Number represents the error code modulo 100, and its value MUST be between 0 and 99.

The following error codes, along with their recommended reason phrases, are defined:

300 Try Alternate: The client should contact an alternate server for this request. This error response MUST only be sent if the request included a USERNAME attribute and a valid MESSAGE-INTEGRITY attribute; otherwise, it MUST NOT be sent and error code 400 (Bad Request) is suggested. This error response MUST be protected with the MESSAGE-INTEGRITY attribute, and receivers MUST validate the MESSAGE-INTEGRITY of this response before redirecting themselves to an alternate server.

Note: Failure to generate and validate message integrity for a 300 response allows an on-path attacker to falsify a 300

response thus causing subsequent STUN messages to be sent to a victim.

400 Bad Request: The request was malformed. The client SHOULD NOT retry the request without modification from the previous attempt. The server may not be able to generate a valid MESSAGE-INTEGRITY for this error, so the client MUST NOT expect a valid MESSAGE-INTEGRITY attribute on this response.

401 Unauthorized: The request did not contain the correct credentials to proceed. The client should retry the request with proper credentials.

420 Unknown Attribute: The server received a STUN packet containing a comprehension-required attribute that it did not understand. The server MUST put this unknown attribute in the UNKNOWN-ATTRIBUTE attribute of its error response.

438 Stale Nonce: The NONCE used by the client was no longer valid. The client should retry, using the NONCE provided in the response.

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

14.8. REALM

The REALM attribute may be present in requests and responses. It contains text that meets the grammar for "realm-value" as described in RFC 3261 [RFC3261] but without the double quotes and their surrounding whitespace. That is, it is an unquoted realm-value (and is therefore a sequence of qdtext or quoted-pair). It MUST be a UTF-8 [RFC3629] encoded sequence of less than 128 characters (which can be as long as 763 bytes), and MUST have been processed using SASLprep [RFC4013].

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.

14.9. NONCE

The NONCE attribute may be present in requests and responses. It contains a sequence of qdtext or quoted-pair, which are defined in RFC 3261 [RFC3261]. Note that this means that the NONCE attribute will not contain actual quote characters. See RFC 2617 [RFC2617], Section 4.3, for guidance on selection of nonce values in a server.

It MUST be less than 128 characters (which can be as long as 763 bytes).

14.10. PASSWORD-ALGORITHMS

The PASSWORD-ALGORITHMS attribute is present only in responses. It contains the list of algorithms that the server can use to derive the long-term password.

The set of known algorithms is maintained by IANA. The initial set defined by this specification is found in Section 17.4.

The attribute contains a list of algorithm numbers and variable length parameters. The algorithm number is a 16-bit value as defined in Section 17.4. The parameters start with the actual length of the parameters as a 16-bit value, followed by the parameters that are specific to each algorithm. The parameters are padded to a 32-bit boundary, in the same manner as an attribute.

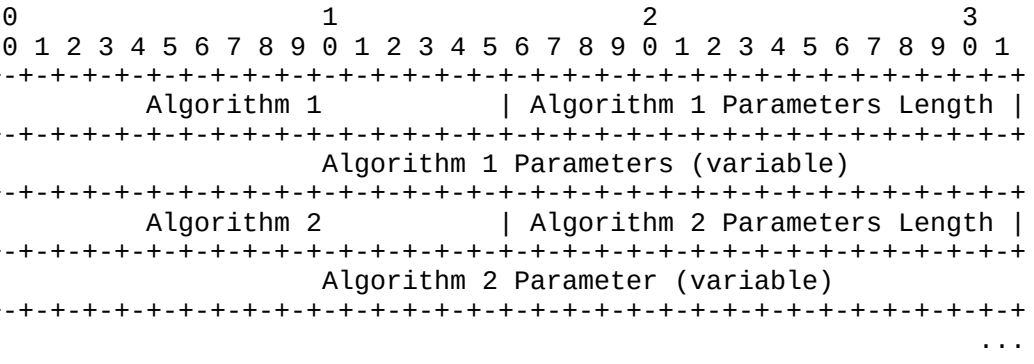


Figure 8: Format of PASSWORD-ALGORITHMS Attribute

14.11. PASSWORD-ALGORITHM

The PASSWORD-ALGORITHM attribute is present only in requests. It contains the algorithms that the server must use to derive the long-term password.

The set of known algorithms is maintained by IANA. The initial set defined by this specification is found in Section 17.4.

The attribute contains an algorithm number and variable length parameters. The algorithm number is a 16-bit value as defined in Section 17.4. The parameters starts with the actual length of the parameters as a 16-bit value, followed by the parameters that are specific to the algorithm. The parameters are padded to a 32-bit boundary, in the same manner as an attribute.

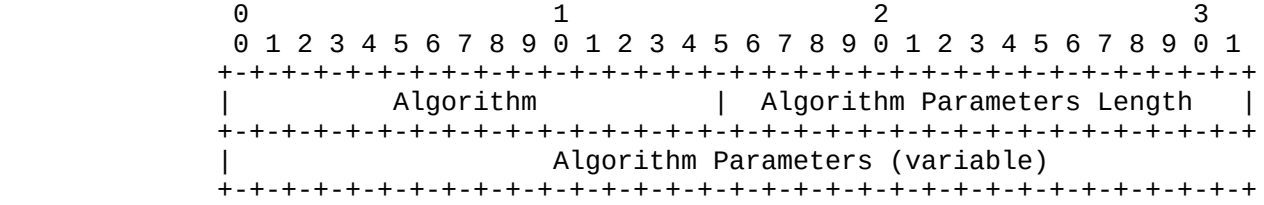


Figure 9: Format of PASSWORD-ALGORITHM Attribute

14.12. 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.

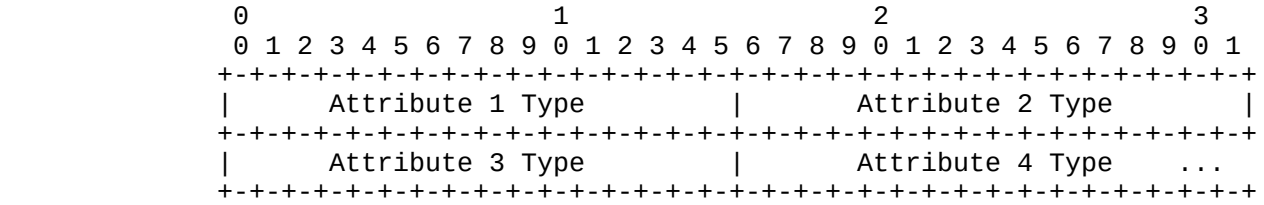


Figure 10: Format of UNKNOWN-ATTRIBUTES Attribute

Note: In [RFC3489], this field was padded to 32 by duplicating the last attribute. In this version of the specification, the normal padding rules for attributes are used instead.

14.13. SOFTWARE

The SOFTWARE attribute contains a textual description of the software being used by the agent sending the message. It is used by clients and servers. Its value SHOULD include 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 SOFTWARE is variable length. It MUST be a UTF-8 [RFC3629]

encoded sequence of less than 128 characters (which can be as long as 763 bytes).

14.14. ALTERNATE-SERVER

The alternate server represents an alternate transport address identifying a different STUN server that the STUN client should try.

It is encoded in the same way as MAPPED-ADDRESS, and thus refers to a single server by IP address. The IP address family MUST be identical to that of the source IP address of the request.

14.15. ALTERNATE-DOMAIN

The alternate domain represents the domain name that is used to verify the IP address in the ALTERNATE-SERVER attribute when the transport protocol uses TLS or DTLS.

The value of ALTERNATE-DOMAIN is variable length. It MUST be a UTF-8 [RFC3629] encoded sequence of less than 128 characters (which can be as long as 763 bytes).

15. Security Considerations

15.1. Attacks against the Protocol

15.1.1. Outside Attacks

An attacker can try to modify STUN messages in transit, in order to cause a failure in STUN operation. These attacks are detected for both requests and responses through the message-integrity mechanism, using either a short-term or long-term credential. Of course, once detected, the manipulated packets will be dropped, causing the STUN transaction to effectively fail. This attack is possible only by an on-path attacker.

An attacker that can observe, but not modify, STUN messages in-transit (for example, an attacker present on a shared access medium, such as Wi-Fi), can see a STUN request, and then immediately send a STUN response, typically an error response, in order to disrupt STUN processing. This attack is also prevented for messages that utilize MESSAGE-INTEGRITY. However, some error responses, those related to authentication in particular, cannot be protected by MESSAGE-INTEGRITY. When STUN itself is run over a secure transport protocol (e.g., TLS), these attacks are completely mitigated.

Depending on the STUN usage, these attacks may be of minimal consequence and thus do not require message integrity to mitigate.

For example, when STUN is used to a basic STUN server to discover a server reflexive candidate for usage with ICE, authentication and message integrity are not required since these attacks are detected during the connectivity check phase. The connectivity checks themselves, however, require protection for proper operation of ICE overall. As described in Section 13, STUN usages describe when authentication and message integrity are needed.

Since STUN uses the HMAC of a shared secret for authentication and integrity protection, it is subject to offline dictionary attacks. When authentication is utilized, it SHOULD be with a strong password that is not readily subject to offline dictionary attacks. Protection of the channel itself, using TLS, mitigates these attacks. However, STUN is most often run over UDP and in those cases, strong passwords are the only way to protect against these attacks.

15.1.2. Inside Attacks

A rogue client may try to launch a DoS attack against a server by sending it a large number of STUN requests. Fortunately, STUN requests can be processed statelessly by a server, making such attacks hard to launch.

A rogue client may use a STUN server as a reflector, sending it requests with a falsified source IP address and port. In such a case, the response would be delivered to that source IP and port. There is no amplification of the number of packets with this attack (the STUN server sends one packet for each packet sent by the client), though there is a small increase in the amount of data, since STUN responses are typically larger than requests. This attack is mitigated by ingress source address filtering.

Revealing the specific software version of the agent through the SOFTWARE attribute might allow them to become more vulnerable to attacks against software that is known to contain security holes. Implementers SHOULD make usage of the SOFTWARE attribute a configurable option.

15.2. Attacks Affecting the Usage

This section lists attacks that might be launched against a usage of STUN. Each STUN usage must consider whether these attacks are applicable to it, and if so, discuss counter-measures.

Most of the attacks in this section revolve around an attacker modifying the reflexive address learned by a STUN client through a Binding request/response transaction. Since the usage of the reflexive address is a function of the usage, the applicability and

remediation of these attacks are usage-specific. In common situations, modification of the reflexive address by an on-path attacker is easy to do. Consider, for example, the common situation where STUN is run directly over UDP. In this case, an on-path attacker can modify the source IP address of the Binding request before it arrives at the STUN server. The STUN server will then return this IP address in the XOR-MAPPED-ADDRESS attribute to the client, and send the response back to that (falsified) IP address and port. If the attacker can also intercept this response, it can direct it back towards the client. Protecting against this attack by using a message-integrity check is impossible, since a message-integrity value cannot cover the source IP address, since the intervening NAT must be able to modify this value. Instead, one solution to preventing the attacks listed below is for the client to verify the reflexive address learned, as is done in ICE [RFC5245]. Other usages may use other means to prevent these attacks.

15.2.1. Attack I: Distributed DoS (DDoS) against a Target

In this attack, the attacker provides one or more clients with the same faked reflexive address that points to the intended target. This will trick the STUN clients into thinking that their reflexive addresses are equal to that of the target. If the clients hand out that reflexive address in order to receive traffic on it (for example, in SIP messages), the traffic will instead be sent to the target. This attack can provide substantial amplification, especially when used with clients that are using STUN to enable multimedia applications. However, it can only be launched against targets for which packets from the STUN server to the target pass through the attacker, limiting the cases in which it is possible.

15.2.2. Attack II: Silencing a Client

In this attack, the attacker provides a STUN client with a faked reflexive address. The reflexive address it provides is a transport 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 reflexive 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. As with the attack in Section 15.2.1, this attack is only possible when the attacker is on path for packets sent from the STUN server towards this unused IP address.

15.2.3. Attack III: Assuming the Identity of a Client

This attack is similar to attack II. However, the faked reflexive address points to the attacker itself. This allows the attacker to receive traffic that was destined for the client.

15.2.4. Attack IV: Eavesdropping

In this attack, the attacker forces the client to use a reflexive 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.

15.3. Hash Agility Plan

This specification uses HMAC-SHA-1 for computation of the message integrity. If, at a later time, HMAC-SHA-1 is found to be compromised, the following is the remedy that will be applied.

We will define a STUN extension that introduces a new message-integrity attribute, computed using a new hash. Clients would be required to include both the new and old message-integrity attributes in their requests or indications. A new server will utilize the new message-integrity attribute, and an old one, the old. After a transition period where mixed implementations are in deployment, the old message-integrity attribute will be deprecated by another specification, and clients will cease including it in requests.

It is also important to note that the HMAC is done using a key that is itself computed using an MD5 of the user’s password. The choice of the MD5 hash was made because of the existence of legacy databases that store passwords in that form. If future work finds that an HMAC of an MD5 input is not secure, and a different hash is needed, it can also be changed using this plan. However, this would require administrators to repopulate their databases.

16. 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 [RFC3424]). STUN can be used to perform this function using a Binding request/response transaction if one agent is behind a NAT and the other is on the public side of the NAT.

The IAB has suggested that protocols developed for this purpose document a specific set of considerations. Because some STUN usages provide UNSAF functions (such as ICE [RFC5245]), and others do not (such as SIP Outbound [RFC5626]), answers to these considerations need to be addressed by the usages themselves.

17. IANA Considerations

IANA has created three new registries: a "STUN Methods Registry", a "STUN Attributes Registry", and a "STUN Error Codes Registry". IANA has also changed the name of the assigned IANA port for STUN from "nat-stun-port" to "stun".

17.1. STUN Methods Registry

A STUN method is a hex number in the range 0x000 - 0xFFF. The encoding of STUN method into a STUN message is described in Section 5.

The initial STUN methods are:

0x000: (Reserved)
0x001: Binding
0x002: (Reserved; was SharedSecret)

STUN methods in the range 0x000 - 0x7FF are assigned by IETF Review [RFC5226]. STUN methods in the range 0x800 - 0xFFF are assigned by Designated Expert [RFC5226]. The responsibility of the expert is to verify that the selected codepoint(s) are not in use and that the request is not for an abnormally large number of codepoints. Technical review of the extension itself is outside the scope of the designated expert responsibility.

17.2. STUN Attribute Registry

A STUN Attribute type is a hex number in the range 0x0000 - 0xFFFF. STUN attribute types in the range 0x0000 - 0x7FFF are considered comprehension-required; STUN attribute types in the range 0x8000 - 0xFFFF are considered comprehension-optional. A STUN agent handles unknown comprehension-required and comprehension-optional attributes differently.

The initial STUN Attributes types are:

Comprehension-required range (0x0000-0x7FFF):
0x0000: (Reserved)
0x0001: MAPPED-ADDRESS
0x0002: (Reserved; was RESPONSE-ADDRESS)
0x0003: (Reserved; was CHANGE-ADDRESS)
0x0004: (Reserved; was SOURCE-ADDRESS)
0x0005: (Reserved; was CHANGED-ADDRESS)
0x0006: USERNAME
0x0007: (Reserved; was PASSWORD)
0x0008: MESSAGE-INTEGRITY
0x0009: ERROR-CODE
0x000A: UNKNOWN-ATTRIBUTES
0x000B: (Reserved; was REFLECTED-FROM)
0x0014: REALM
0x0015: NONCE
0x0020: XOR-MAPPED-ADDRESS
0xXXXX: PASSWORD-ALGORITHM

Comprehension-optional range (0x8000-0xFFFF)
0x8022: SOFTWARE
0x8023: ALTERNATE-SERVER
0x8028: FINGERPRINT
0xXXXX: PASSSORD-ALGORITHMS
0xXXXX: ALTERNATE-DOMAIN

STUN Attribute types in the first half of the comprehension-required range (0x0000 - 0x3FFF) and in the first half of the comprehension-optional range (0x8000 - 0xBFFF) are assigned by IETF Review [RFC5226]. STUN Attribute types in the second half of the comprehension-required range (0x4000 - 0x7FFF) and in the second half of the comprehension-optional range (0xC000 - 0xFFFF) are assigned by Designated Expert [RFC5226]. The responsibility of the expert is to verify that the selected codepoint(s) are not in use, and that the request is not for an abnormally large number of codepoints. Technical review of the extension itself is outside the scope of the designated expert responsibility.

17.3. STUN Error Code Registry

A STUN error code is a number in the range 0 - 699. STUN error codes are accompanied by a textual reason phrase in UTF-8 [RFC3629] that is intended only for human consumption and can be anything appropriate; this document proposes only suggested values.

STUN error codes are consistent in codepoint assignments and semantics with SIP [RFC3261] and HTTP [RFC2616].

The initial values in this registry are given in Section 14.7.

New STUN error codes are assigned based on IETF Review [RFC5226]. The specification must carefully consider how clients that do not understand this error code will process it before granting the request. See the rules in Section 6.3.4.

17.4. Password Algorithm Registry

A Password Algorithm is a hex number in the range 0x0000 - 0xFFFF.

The initial Password Algorithms are:

0x0001: Salted SHA256

Password Algorithms in the first half of the range (0x0000 - 0x7FFF) are assigned by IETF Review [RFC5226]. Password Algorithms in the second half of the range (0x8000 - 0xFFFF) are assigned by Designated Expert [RFC5226].

17.4.1. Password Algorithms

The initial list of password algorithms is taken from [I-D.veltri-sip-alt-auth].

17.4.1.1. Salted SHA256

The key length is 32 bytes and the parameters contains the salt.

```
key = SHA256(username ":" realm ":" SASLprep(password) ":" salt)
```

17.5. STUN UDP and TCP Port Numbers

IANA has previously assigned port 3478 for STUN. This port appears in the IANA registry under the moniker "nat-stun-port". In order to align the DNS SRV procedures with the registered protocol service, IANA is requested to change the name of protocol assigned to port 3478 from "nat-stun-port" to "stun", and the textual name from "Simple Traversal of UDP Through NAT (STUN)" to "Session Traversal Utilities for NAT", so that the IANA port registry would read:

```
stun   3478/tcp   Session Traversal Utilities for NAT (STUN) port
stun   3478/udp   Session Traversal Utilities for NAT (STUN) port
```

In addition, IANA has assigned port number 5349 for the "stuns" service, defined over TCP and UDP. The UDP port is not currently defined; however, it is reserved for future use.

18. Changes since RFC 5389

This specification obsoletes RFC 5389 [RFC5389]. This specification differs from RFC 5389 in the following ways:

o

19. References

19.1. Normative References

[I-D.ietf-tsvwg-sctp-dtls-encaps]
Jesup, R., Loreto, S., Stewart, R., and M. Tuexen, "DTLS Encapsulation of SCTP Packets for RTCWEB", draft-ietf-tsvwg-sctp-dtls-encaps-00 (work in progress), February 2013.

[I-D.ietf-tsvwg-sctp-prpolicies]
Tuexen, M., Seggelmann, R., Stewart, R., and S. Loreto, "Additional Policies for the Partial Reliability Extension of the Stream Control Transmission Protocol", draft-ietf-tsvwg-sctp-prpolicies-05 (work in progress), November 2014.

[ITU.V42.2002]
International Telecommunications Union, "Error-correcting Procedures for DCEs Using Asynchronous-to-Synchronous Conversion", ITU-T Recommendation V.42, 2002.

[RFC0791] Postel, J., "Internet Protocol", STD 5, RFC 791, September 1981.

[RFC1122] Braden, R., "Requirements for Internet Hosts - Communication Layers", STD 3, RFC 1122, October 1989.

[RFC1321] Rivest, R., "The MD5 Message-Digest Algorithm", RFC 1321, April 1992.

[RFC2104] Krawczyk, H., Bellare, M., and R. Canetti, "HMAC: Keyed-Hashing for Message Authentication", RFC 2104, February 1997.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[RFC2460] Deering, S. and R. Hinden, "Internet Protocol, Version 6 (IPv6) Specification", RFC 2460, December 1998.

[RFC2617] 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.

[RFC2782] Gulbrandsen, A., Vixie, P., and L. Esibov, "A DNS RR for specifying the location of services (DNS SRV)", RFC 2782, February 2000.

[RFC2818] Rescorla, E., "HTTP Over TLS", RFC 2818, May 2000.

[RFC3629] Yergeau, F., "UTF-8, a transformation format of ISO 10646", STD 63, RFC 3629, November 2003.

[RFC3758] Stewart, R., Ramalho, M., Xie, Q., Tuexen, M., and P. Conrad, "Stream Control Transmission Protocol (SCTP) Partial Reliability Extension", RFC 3758, May 2004.

[RFC4013] Zeilenga, K., "SASLprep: Stringprep Profile for User Names and Passwords", RFC 4013, February 2005.

[RFC4033] Arends, R., Austein, R., Larson, M., Massey, D., and S. Rose, "DNS Security Introduction and Requirements", RFC 4033, March 2005.

[RFC5246] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", RFC 5246, August 2008.

[RFC6298] Paxson, V., Allman, M., Chu, J., and M. Sargent, "Computing TCP's Retransmission Timer", RFC 6298, June 2011.

[RFC6347] Rescorla, E. and N. Modadugu, "Datagram Transport Layer Security Version 1.2", RFC 6347, January 2012.

[RFC6698] Hoffman, P. and J. Schlyter, "The DNS-Based Authentication of Named Entities (DANE) Transport Layer Security (TLS) Protocol: TLSA", RFC 6698, August 2012.

[RFC6951] Tuexen, M. and R. Stewart, "UDP Encapsulation of Stream Control Transmission Protocol (SCTP) Packets for End-Host to End-Host Communication", RFC 6951, May 2013.

[RFC7064] Nandakumar, S., Salgueiro, G., Jones, P., and M. Petit-Huguenin, "URI Scheme for the Session Traversal Utilities for NAT (STUN) Protocol", RFC 7064, November 2013.

[RFC7350] Petit-Huguenin, M. and G. Salgueiro, "Datagram Transport Layer Security (DTLS) as Transport for Session Traversal Utilities for NAT (STUN)", RFC 7350, August 2014.

19.2. Informational References

[I-D.ietf-tram-stun-origin]
Johnston, A., Uberti, J., Yoakum, J., and K. Singh, "An Origin Attribute for the STUN Protocol", draft-ietf-tram-stun-origin-05 (work in progress), February 2015.

[I-D.ietf-uta-tls-bcp]
Sheffer, Y., Holz, R., and P. Saint-Andre, "Recommendations for Secure Use of TLS and DTLS", draft-ietf-uta-tls-bcp-11 (work in progress), February 2015.

[I-D.veltri-sip-alt-auth]
Veltri, L., Salsano, S., and A. Polidoro, "HTTP digest authentication using alternate password storage schemes", draft-veltri-sip-alt-auth-00 (work in progress), April 2008.

[KARN87] Karn, P. and C. Partridge, "Improving Round-Trip Time Estimates in Reliable Transport Protocols", SIGCOMM 1987, August 1987.

[RFC2616] 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.

[RFC3261] 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.

[RFC3424] Daigle, L. and IAB, "IAB Considerations for UNilateral Self-Address Fixing (UNSAF) Across Network Address Translation", RFC 3424, November 2002.

[RFC3489] 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.

[RFC4107] Bellovin, S. and R. Housley, "Guidelines for Cryptographic Key Management", BCP 107, RFC 4107, June 2005.

[RFC5226] Narten, T. and H. Alvestrand, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 5226, May 2008.

[RFC5245] Rosenberg, J., "Interactive Connectivity Establishment (ICE): A Protocol for Network Address Translator (NAT) Traversal for Offer/Answer Protocols", RFC 5245, April 2010.

[RFC5389] Rosenberg, J., Mahy, R., Matthews, P., and D. Wing, "Session Traversal Utilities for NAT (STUN)", RFC 5389, October 2008.

[RFC5626] Jennings, C., Mahy, R., and F. Audet, "Managing Client-Initiated Connections in the Session Initiation Protocol (SIP)", RFC 5626, October 2009.

[RFC5766] Mahy, R., Matthews, P., and J. Rosenberg, "Traversal Using Relays around NAT (TURN): Relay Extensions to Session Traversal Utilities for NAT (STUN)", RFC 5766, April 2010.

[RFC5780] MacDonald, D. and B. Lowekamp, "NAT Behavior Discovery Using Session Traversal Utilities for NAT (STUN)", RFC 5780, May 2010.

[RFC6544] Rosenberg, J., Keranen, A., Lowekamp, B., and A. Roach, "TCP Candidates with Interactive Connectivity Establishment (ICE)", RFC 6544, March 2012.

Appendix A. C Snippet to Determine STUN Message Types

Given a 16-bit STUN message type value in host byte order in msg_type parameter, below are C macros to determine the STUN message types:

```
#define IS_REQUEST(msg_type)       (((msg_type) & 0x0110) == 0x0000)
#define IS_INDICATION(msg_type)    (((msg_type) & 0x0110) == 0x0010)
#define IS_SUCCESS_RESP(msg_type)  (((msg_type) & 0x0110) == 0x0100)
#define IS_ERR_RESP(msg_type)      (((msg_type) & 0x0110) == 0x0110)
```

A function to convert method and class into a message type:

```
int type(int method, int cls) {
  return (method & 0x0F80) << 9 | (method & 0x0070) << 5
       | (method & 0x000F) | (cls & 0x0002) << 8
       | (cls & 0x0001) << 4;
  }
```

A function to extract the method from the message type:

```
int method(int type) {
  return (type & 0x3E00) >> 2 | (type & 0x00E0) >> 1
       | (type & 0x000F);
  }
```

A function to extract the class from the message type:

```
int cls(int type) {
  return (type & 0x0100) >> 7 | (type & 0x0010) >> 4;
  }
```

Appendix B. Release notes

This section must be removed before publication as an RFC.

B.1. Open Issues

1. Integrate RFC 5769 (stun vectors) as examples

B.2. Modifications between draft-ietf-tram-stunbis-02 and draft-ietf-tram-stunbis-01

- Prevent the server to allocate the same NONCE to clients with different IP address and/or different port. This prevent sharing the nonce between TURN allocations in TURN.
- Add reference to draft-ietf-uta-tls-bcp
- Add a new attribute ALTERNATE-DOMAIN to verify the certificate of the ALTERNATE-SERVER after a 300 over (D)TLS.
- The RTP delay between transactions applies only to parallel transactions, not to serial transactions. That prevents a 3RTT delay between the first transaction and the second transaction with long term authentication.
- Add text saying ORIGIN can increase a request size beyond the MTU and so require an SCTPoUDP transport.
- Add a new attribute ALTERNATE-DOMAIN to verify the certificate of the ALTERNATE-SERVER after a 300 over (D)TLS.
- The RTP delay between transactions applies only to parallel transactions, not to serial transactions. That prevents a 3RTT delay between the first transaction and the second transaction with long term authentication.

- Add text saying ORIGIN can increase a request size beyond the MTU and so require an SCTPoUDP transport.
- Move the Acknowledgments and Contributor sections to the end of the document, in accordance with RFC 7322 section 4.

B.3. Modifications between draft-ietf-tram-stunbis-01 and draft-ietf-tram-stunbis-00

- Add negotiation mechanism for new password algorithms.
- Describe the MESSAGE-INTEGRITY/MESSAGE-INTEGRITY2 protocol.
- Add support for SCTP to solve the fragmentation problem.
- Merge RFC 7350:
  - Split the "Sending over..." sections in 3.
  - Add DTLS-over-UDP as transport.
  - Update the cipher suites and cipher/compression restrictions.
  - A stuns uri with an IP address is rejected.
  - Replace most of the RFC 3489 compatibility by a reference to the section in RFC 5389.
  - Update the STUN Usages list with transport applicability.
- Merge RFC 7064:
  - DNS discovery is done from the URI.
  - Reorganized the text about default ports.
- Add more C snippets.
- Make clear that the cached RTO is discarded only if there is no new transations for 10 minutes.

B.4. Modifications between draft-salgueiro-tram-stunbis-02 and draft-ietf-tram-stunbis-00

- Draft adopted as WG item.

B.5. Modifications between draft-salgueiro-tram-stunbis-02 and draft-salgueiro-tram-stunbis-01

- Add definition of MESSAGE-INTEGRITY2.
- Update text and reference from RFC 2988 to RFC 6298.
- s/The IAB has mandated/The IAB has suggested/ (Errata #3737).
- Fix the figure for the UNKNOWN-ATTRIBUTES (Errata #2972).
- Fix section number and make clear that the original domain name is used for the server certificate verification. This is consistent with what RFC 5922 (section 4) is doing. (Errata #2010)
- Remove text transitioning from RFC 3489.
- Add definition of MESSAGE-INTEGRITY2.
- Update text and reference from RFC 2988 to RFC 6298.
- s/The IAB has mandated/The IAB has suggested/ (Errata #3737).
- Fix the figure for the UNKNOWN-ATTRIBUTES (Errata #2972).
- Fix section number and make clear that the original domain name is used for the server certificate verification. This is consistent with what RFC 5922 (section 4) is doing. (Errata #2010)

B.6. Modifications between draft-salgueiro-tram-stunbis-01 and draft-salgueiro-tram-stunbis-00

- Restore the RFC 5389 text.
- Add list of open issues.

Acknowledgements

Thanks to Michael Tuexen, Tirumaleswar Reddy, Oleg Moskalenko, Simon Perreault, and Benjamin Schwartz for the comments, suggestions, and questions that helped improve this document.

The authors of RFC 5389 would like to thank Cedric Aoun, Pete Cordell, Cullen Jennings, Bob Penfield, Xavier Marjou, Magnus Westerlund, Miguel Garcia, Bruce Lowekamp, 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.

Contributors

Christian Huitema and Joel Weinberger were original co-authors of RFC 3489.

Authors' Addresses

Marc Petit-Huguenin
Impedance Mismatch

Email: marc@petit-huguenin.org

Gonzalo Salgueiro
Cisco
7200-12 Kit Creek Road
Research Triangle Park, NC 27709
US

Email: gsalguei@cisco.com

Jonathan Rosenberg
Cisco
Edison, NJ
US

Email: jdrosen@cisco.com
URI: http://www.jdrosen.net

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

Email: dwing@cisco.com

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

Email: rohan@ekabal.com

Philip Matthews
Avaya
1135 Innovation Drive
Ottawa, Ontario K2K 3G7
Canada

Phone: +1 613 592 4343 x224
Email: philip_matthews@magma.ca

TRAM
Internet-Draft
Intended status: Standards Track
Expires: July 26, 2015

P. Patil
T. Reddy
D. Wing
Cisco

January 22, 2015

TURN Server Auto Discovery
draft-ietf-tram-turn-server-discovery-01

Abstract

Current Traversal Using Relays around NAT (TURN) server discovery mechanisms are relatively static and limited to explicit configuration. These are usually under the administrative control of the application or TURN service provider, and not the enterprise or the ISP, the network in which the client is located. Enterprises and ISPs wishing to provide their own TURN servers need auto discovery mechanisms that a TURN client could use with no or minimal configuration. This document describes two such mechanisms for TURN server discovery.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at http://datatracker.ietf.org/drafts/current/.

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."

This Internet-Draft will expire on July 26, 2015.

Copyright Notice

Table of Contents

1. Introduction

TURN [RFC5766] is a protocol that is often used to improve the connectivity of Peer-to-Peer (P2P) applications (as defined in section 2.7 of [RFC5128]). TURN allows a connection to be established when one or both sides are incapable of a direct P2P connection. It is an important building block for interactive, real-time communication using audio, video, collaboration etc. While TURN services are extensively used today, the means to auto discover TURN servers do not exist. TURN clients are usually explicitly configured with a well known TURN server. To allow TURN applications operate seamlessly across different types of networks and encourage the use of TURN without the need for manual configuration, it is important that there exists an auto discovery mechanism for TURN services. Web

Real-Time Communication (WebRTC) [I-D.ietf-rtcweb-overview] usages and related extensions, which are mostly based on web applications, need this immediately.

This document describes two discovery mechanisms. The reason for providing two mechanisms is to maximize the opportunity for discovery, based on the network in the which the TURN client sees itself.

- o A resolution mechanism based on straightforward Naming Authority Pointer (S-NAPTR) resource records in the Domain Name System (DNS). [RFC5928] describes details on retrieving a list of server transport addresses from DNS that can be used to create a TURN allocation.

- o A mechanism based on anycast address for TURN.

In general, if a client wishes to communicate using one of its interfaces using a specific IP address family, it SHOULD query the TURN server(s) that has been discovered for that specific interface and address family. How to select an interface and IP address family, is out of the scope of this document.

2. Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119].

3. Discovery Procedure

A TURN client that implements the auto discovery algorithm MUST proceed with discovery in the following order:

1. Local Configuration : Local or manual configuration should be tried first, as it may be an explicit preferred choice of a user. An implementation MAY give the user an opportunity (e.g., by means of configuration file options or menu items) to specify a TURN server for every address family.

2. Service Resolution : The TURN client attempts to perform TURN service resolution using the DNS domain name that the host belongs to OR the hosts' global IP address. The TURN client will attempt to do this for each combination of interface and address family. The retrieved DNS domain names OR IP addresses are then used for NAPTR lookups.

3. Anycast : Send TURN allocate request to the assigned TURN anycast request for each combination of interface and address family.

While it is expected that Step-3 be performed if Step-2 fails, an implementation may choose to perform steps 2 and 3 in parallel.

4. Discovery using Service Resolution

This mechanism is performed in two steps:

1. A DNS domain name is retrieved for each combination of interface and address family.

2. Retrieved DNS domain names are then used for S-NAPTR lookups as per [RFC5928]. Further DNS lookups may be necessary to determine TURN server IP address(es).

On hosts with more than one interface or address family (IPv4/v6), the TURN server discovery procedure has to be run for each combination of interface and address family.

4.1. Retrieving Domain Name

The domain, in which the client is located, can be determined using one of the techniques provided below. An implementation can choose to use any or all techniques.

Implementations may allow the user to specify a default name that is used if no specific name has been configured. Other means of retrieving domain names may be used, which are outside the scope of this document e.g. local configuration.

4.1.1. DHCP

DHCP can be used to determine the domain name related to an interface’s point of network attachment. Network operators may provide the domain name to be used for service discovery within an access network using DHCP. [RFC5986] defines DHCP IPv4 and IPv6 access network domain name options to identify a domain name that is suitable for service discovery within the access network. [RFC2132] defines the DHCP IPv4 domain name option. While this option is less suitable, it still may be useful if the option defined in [RFC5986] is not available.

For IPv6, the TURN server discovery procedure MUST try to retrieve DHCP option 57 (OPTION_V6_ACCESS_DOMAIN). If no such option can be retrieved, the procedure fails for this interface. For IPv4, the TURN server discovery procedure MUST try to retrieve DHCP option 213

(OPTION_V4_ACCESS_DOMAIN). If no such option can be retrieved, the procedure SHOULD try to retrieve option 15 (Domain Name). If neither option can be retrieved the procedure fails for this interface. If a result can be retrieved it will be used as an input for S-NAPTR resolution.

4.1.2. IP Address

Typically, but not necessarily, the DNS domain name is the domain name in which the client is located, i.e., a PTR lookup on the client's IP address (according to [RFC1035], Section 3.5 for IPv4 or [RFC3596], Section 2.5 for IPv6) would yield a similar name. However, due to the widespread use of Network Address Translation (NAT), the client MAY need to determine its public IP address using mechanisms described in [RFC7216].

4.1.3. From own Identity

A TURN client could also wish to extract the domain name from its own identity i.e canonical identifier used to reach the user.

Example

```
SIP   : 'sip:alice@example.com'
JID   : 'alice@example.com'
email : 'alice@example.com'
```

'example.com' is retrieved from the above examples.

The means to extract the domain name may be different based on the type of identifier and is outside the scope of this document.

4.2. Resolution

Once the TURN discovery procedure has retrieved domain names, the resolution mechanism described in [RFC5928] is followed. An S-NAPTR lookup with 'RELAY' application service and the desired protocol tag is made to obtain information necessary to connect to the authoritative TURN server within the given domain.

In the example below, for domain 'example.net', the resolution algorithm will result in IP address, port, and protocol tuples as follows:

```
example.net.
   IN NAPTR 100 10 "" RELAY:turn.udp "" example.net.

   example.net.
   IN NAPTR 100 10 S RELAY:turn.udp "" _turn._udp.example.net.

   _turn._udp.example.net.
   IN SRV   0 0 3478 a.example.net.

   a.example.net.
   IN A     192.0.2.1
```

Order	Protocol	IP address	Port
1	UDP	192.0.2.1	3478

If no TURN-specific S-NAPTR records can be retrieved, the discovery procedure fails for this domain name (and the corresponding interface and IP protocol version). If more domain names are known, the discovery procedure may perform the corresponding S-NAPTR lookups immediately. However, before retrying a lookup that has failed, a client MUST wait a time period that is appropriate for the encountered error (NXDOMAIN, timeout, etc.).

4.2.1. SOA

If no TURN-specific S-NAPTR records can be retrieved using the previous step, additional steps described in this section have to be followed. First, an SOA record for the "reverse zone" i.e., the zone in the in-addr.arpa. or ip6.arpa. domain that contains the IP address(s) in question, has to be retrieved. IP addresses can be determined, if not done already, as described in Section 4.1.2.

A sample SOA record could be:

```
100.51.198.in-addr.arpa
IN  SOA dns1.isp.example.net.   hostmaster.isp.example.net. (
                              1         ; Serial
                         604800         ; Refresh
                          86400         ; Retry
                        2419200         ; Expire
                         604800 )       ; Negative Cache TTL
```

If this lookup fails, the discovery procedure is aborted without a result.

Once the SOA record is available, the discovery procedure extracts the MNAME field, i.e., the responsible master name server from the SOA record. An example MNAME could be: dns1.isp.example.net. Then, an S-NAPTR lookup as specified in the previous step Section 4.2 is performed on this MNAME to discover the TURN service. If no TURN-specific S-NAPTR records can be retrieved, the discovery procedure fails for this domain name (and the corresponding interface and IP protocol version).

5. Discovery using Anycast

IP anycast is an elegant solution for TURN service discovery. A packet sent to an anycast address is delivered to the "topologically nearest" network interface with the anycast address. Using the TURN anycast address, the only two things that need to be deployed in the network are the two things that actually use TURN.

When a client requires TURN services, it sends a TURN allocate request to the assigned anycast address. The TURN anycast server responds with a 300 (Try Alternate) error as described in [RFC5766]; The response contains the TURN unicast address in the ALTERNATE-SERVER attribute. For subsequent communication with the TURN server, the client uses the responding server’s unicast address. This has to be done because two packets addressed to an anycast address may reach two different anycast servers. The client, thus, also needs to ensure that the initial request fits in a single packet. An implementation may choose to send out every new request to the anycast address to learn the closest TURN server each time.

6. Deployment Considerations

6.1. Mobility and Changing IP addresses

A change of IP address on an interface may invalidate the result of the TURN server discovery procedure. For instance, if the IP address assigned to a mobile host changes due to host mobility, it may be required to re-run the TURN server discovery procedure without relying on earlier gained information. New requests should be made to the newly learned TURN servers learned after TURN discovery re-run. However, if an earlier learned TURN server is still accessible using the new IP address, procedures described for mobility using TURN defined in [I-D.wing-mmusic-ice-mobility] can be used for ongoing streams.

7. IANA Considerations

7.1. Anycast

IANA should allocate an IPv4 and an IPv6 well-known TURN anycast address. 192.0.0.0/24 and 2001:0000::/48 are reserved for IETF Protocol Assignments, as listed at

<http://www.iana.org/assignments/iana-ipv4-special-registry/> and

<http://www.iana.org/assignments/iana-ipv6-special-registry/>

8. Security Considerations

In general, it is recommended that a TURN client authenticate with the TURN server to identify a rouge server. [RFC7350] can be potentially used by a client to validate a previously unknown server.

8.1. Service Resolution

The primary attack against the methods described in this document is one that would lead to impersonation of a TURN server. An attacker could attempt to compromise the S-NAPTR resolution. Security considerations described in [RFC5928] are applicable here as well.

In addition to considerations related to S-NAPTR, it is important to recognize that the output of this is entirely dependent on its input. An attacker who can control the domain name can also control the final result. Because more than one method can be used to determine the domain name, a host implementation needs to consider attacks against each of the methods that are used.

If DHCP is used, the integrity of DHCP options is limited by the security of the channel over which they are provided. Physical security and separation of DHCP messages from other packets are commonplace methods that can reduce the possibility of attack within an access network; alternatively, DHCP authentication [RFC3188] can provide a degree of protection against modification. When using DHCP discovery, clients are encouraged to use unicast DHCP INFORM queries instead of broadcast queries which are more easily spoofed in insecure networks.

8.2. Anycast

In a network without any TURN server that is aware of the TURN anycast address, outgoing TURN requests could leak out onto the external Internet, possibly revealing information.

Using an IANA-assigned well-known TURN anycast address enables border gateways to block such outgoing packets. In the default-free zone, routers should be configured to drop such packets. Such configuration can occur naturally via BGP messages advertising that no route exists to said address.

Sensitive clients that do not wish to leak information about their presence can set an IP TTL on their TURN requests that limits how far they can travel into the public Internet.

9. Acknowledgements

Discovery using Service Resolution described in Section 4 of this document was derived from similar techniques described in ALTO Server Discovery [RFC7286] and [I-D.kist-alto-3pdisc].

10. References

10.1. Normative References

[RFC1035] Mockapetris, P., "Domain names - implementation and specification", STD 13, RFC 1035, November 1987.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[RFC2132] Alexander, S. and R. Droms, "DHCP Options and BOOTP Vendor Extensions", RFC 2132, March 1997.

[RFC3596] Thomson, S., Huitema, C., Ksinant, V., and M. Souissi, "DNS Extensions to Support IP Version 6", RFC 3596, October 2003.

[RFC5766] Mahy, R., Matthews, P., and J. Rosenberg, "Traversal Using Relays around NAT (TURN): Relay Extensions to Session Traversal Utilities for NAT (STUN)", RFC 5766, April 2010.

[RFC5928] Petit-Huguenin, M., "Traversal Using Relays around NAT (TURN) Resolution Mechanism", RFC 5928, August 2010.

[RFC5986] Thomson, M. and J. Winterbottom, "Discovering the Local Location Information Server (LIS)", RFC 5986, September 2010.

[RFC7216] Thomson, M. and R. Bellis, "Location Information Server (LIS) Discovery Using IP Addresses and Reverse DNS", RFC 7216, April 2014.

[RFC7350] Petit-Huguenin, M. and G. Salgueiro, "Datagram Transport Layer Security (DTLS) as Transport for Session Traversal Utilities for NAT (STUN)", RFC 7350, August 2014.

10.2. Informative References

[I-D.ietf-rtcweb-overview]
Alvestrand, H., "Overview: Real Time Protocols for Browser-based Applications", draft-ietf-rtcweb-overview-13 (work in progress), November 2014.

[I-D.kist-alto-3pdisc]
Kiesel, S., Krause, K., and M. Stiemerling, "Third-Party ALTO Server Discovery (3pdisc)", draft-kist-alto-3pdisc-05 (work in progress), January 2014.

[I-D.wing-mmusic-ice-mobility]
Wing, D., Reddy, T., Patil, P., and P. Martinsen, "Mobility with ICE (MICE)", draft-wing-mmusic-ice-mobility-07 (work in progress), June 2014.

[RFC3188] Hakala, J., "Using National Bibliography Numbers as Uniform Resource Names", RFC 3188, October 2001.

[RFC5128] Srisuresh, P., Ford, B., and D. Kegel, "State of Peer-to-Peer (P2P) Communication across Network Address Translators (NATs)", RFC 5128, March 2008.

[RFC7286] Kiesel, S., Stiemerling, M., Schwan, N., Scharf, M., and H. Song, "Application-Layer Traffic Optimization (ALTO) Server Discovery", RFC 7286, November 2014.

Appendix A. Change History

[Note to RFC Editor: Please remove this section prior to publication.]

A.1. Change from draft-patil-tram-serv-disc-00 to -01

- Added IP address (Section 4.1.2) and Own identity (4.1.3) as new means to obtain domain names
- New Section 4.2.1 SOA (inspired by draft-kist-alto-3pdisc)
- 300 (Try Alternate) response for Anycast

Authors' Addresses

Prashanth Patil
Cisco Systems, Inc.
Bangalore
India

Email: praspati@cisco.com

Tirumaleswar Reddy
Cisco Systems, Inc.
Cessna Business Park, Varthur Hobli
Sarjapur Marathalli Outer Ring Road
Bangalore, Karnataka 560103
India

Email: tireddy@cisco.com

Dan Wing
Cisco Systems, Inc.
170 West Tasman Drive
San Jose, California 95134
USA

Email: dwing@cisco.com

TRAM WG
Internet-Draft
Intended status: Standards Track
Expires: August 7, 2015

T. Reddy, Ed.
Cisco Systems, Inc.
A. Johnston, Ed.
Avaya
P. Matthews
Alcatel-Lucent
J. Rosenberg
jdrosen.net
February 3, 2015

Traversal Using Relays around NAT (TURN): Relay Extensions to Session Traversal Utilities for NAT (STUN)
draft-ietf-tram-turnbis-02

Abstract

If a host is located behind a NAT, then in certain situations it can be impossible for that host to communicate directly with other hosts (peers). In these situations, it is necessary for the host to use the services of an intermediate node that acts as a communication relay. This specification defines a protocol, called TURN (Traversal Using Relays around NAT), that allows the host to control the operation of the relay and to exchange packets with its peers using the relay. TURN differs from some other relay control protocols in that it allows a client to communicate with multiple peers using a single relay address.

The TURN protocol was designed to be used as part of the ICE (Interactive Connectivity Establishment) approach to NAT traversal, though it also can be used without ICE.

Status of This Memo

Table of Contents

1. Introduction

A host behind a NAT may wish to exchange packets with other hosts, some of which may also be behind NATs. To do this, the hosts involved can use "hole punching" techniques (see [RFC5128]) in an attempt discover a direct communication path; that is, a communication path that goes from one host to another through intervening NATs and routers, but does not traverse any relays.

As described in [RFC5128] and [RFC4787], hole punching techniques will fail if both hosts are behind NATs that are not well behaved. For example, if both hosts are behind NATs that have a mapping behavior of "address-dependent mapping" or "address- and port-dependent mapping", then hole punching techniques generally fail.

When a direct communication path cannot be found, it is necessary to use the services of an intermediate host that acts as a relay for the packets. This relay typically sits in the public Internet and relays packets between two hosts that both sit behind NATs.

This specification defines a protocol, called TURN, that allows a host behind a NAT (called the TURN client) to request that another host (called the TURN server) act as a relay. The client can arrange for the server to relay packets to and from certain other hosts (called peers) and can control aspects of how the relaying is done. The client does this by obtaining an IP address and port on the server, called the relayed transport address. When a peer sends a packet to the relayed transport address, the server relays the packet to the client. When the client sends a data packet to the server, the server relays it to the appropriate peer using the relayed transport address as the source.

A client using TURN must have some way to communicate the relayed transport address to its peers, and to learn each peer's IP address and port (more precisely, each peer's server-reflexive transport address, see Section 2). How this is done is out of the scope of the TURN protocol. One way this might be done is for the client and

peers to exchange email messages. Another way is for the client and its peers to use a special-purpose "introduction" or "rendezvous" protocol (see [RFC5128] for more details).

If TURN is used with ICE [RFC5245], then the relayed transport address and the IP addresses and ports of the peers are included in the ICE candidate information that the rendezvous protocol must carry. For example, if TURN and ICE are used as part of a multimedia solution using SIP [RFC3261], then SIP serves the role of the rendezvous protocol, carrying the ICE candidate information inside the body of SIP messages. If TURN and ICE are used with some other rendezvous protocol, then [I-D.rosenberg-mmusic-ice-nonsip] provides guidance on the services the rendezvous protocol must perform.

Though the use of a TURN server to enable communication between two hosts behind NATs is very likely to work, it comes at a high cost to the provider of the TURN server, since the server typically needs a high-bandwidth connection to the Internet . As a consequence, it is best to use a TURN server only when a direct communication path cannot be found. When the client and a peer use ICE to determine the communication path, ICE will use hole punching techniques to search for a direct path first and only use a TURN server when a direct path cannot be found.

TURN was originally invented to support multimedia sessions signaled using SIP. Since SIP supports forking, TURN supports multiple peers per relayed transport address; a feature not supported by other approaches (e.g., SOCKS [RFC1928]). However, care has been taken to make sure that TURN is suitable for other types of applications.

TURN was designed as one piece in the larger ICE approach to NAT traversal. Implementors of TURN are urged to investigate ICE and seriously consider using it for their application. However, it is possible to use TURN without ICE.

TURN is an extension to the STUN (Session Traversal Utilities for NAT) protocol [RFC5389]. Most, though not all, TURN messages are STUN-formatted messages. A reader of this document should be familiar with STUN.

2. Overview of Operation

This section gives an overview of the operation of TURN. It is non-normative.

In a typical configuration, a TURN client is connected to a private network [RFC1918] and through one or more NATs to the public Internet. On the public Internet is a TURN server. Elsewhere in the

Internet are one or more peers with which the TURN client wishes to communicate. These peers may or may not be behind one or more NATs. The client uses the server as a relay to send packets to these peers and to receive packets from these peers.

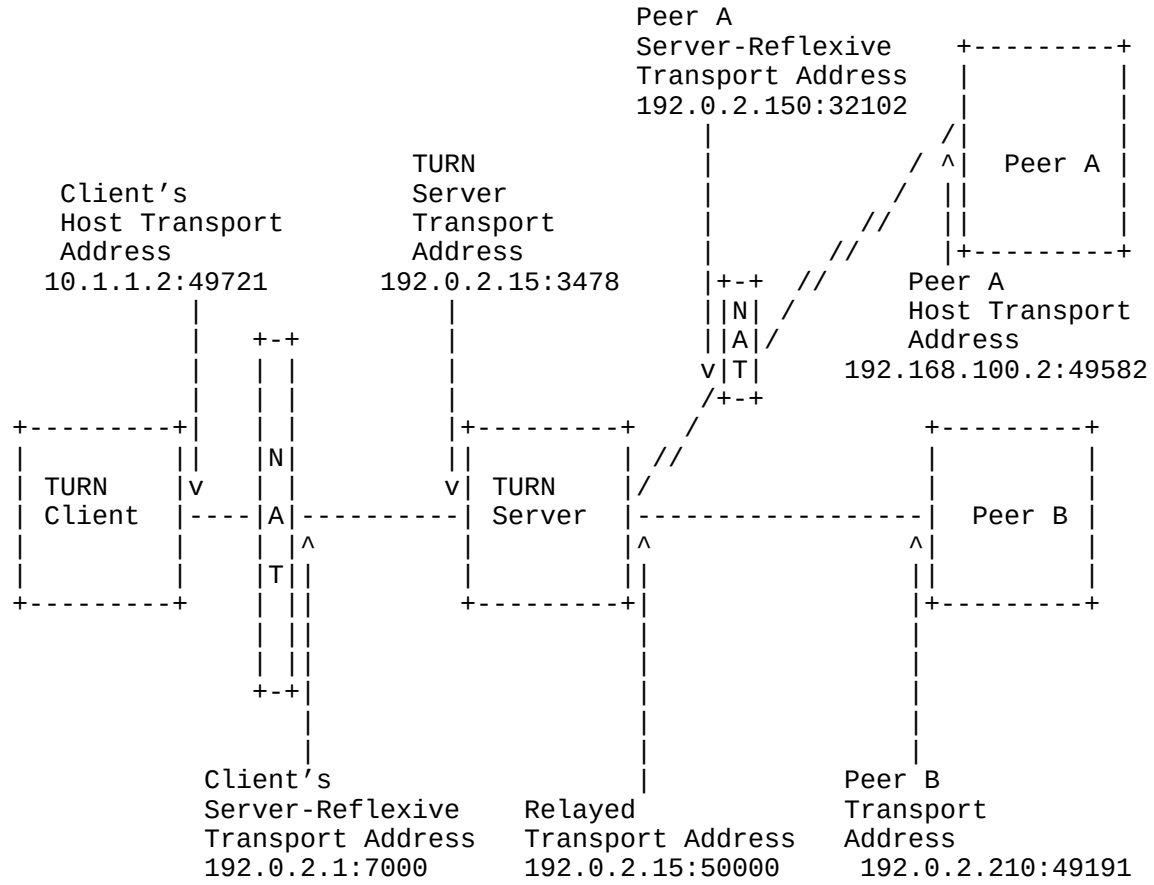


Figure 1

Figure 1 shows a typical deployment. In this figure, the TURN client and the TURN server are separated by a NAT, with the client on the private side and the server on the public side of the NAT. This NAT is assumed to be a "bad" NAT; for example, it might have a mapping property of "address-and-port-dependent mapping" (see [RFC4787]).

The client talks to the server from a (IP address, port) combination called the client's HOST TRANSPORT ADDRESS. (The combination of an IP address and port is called a TRANSPORT ADDRESS.)

The client sends TURN messages from its host transport address to a transport address on the TURN server that is known as the TURN SERVER TRANSPORT ADDRESS. The client learns the TURN server transport address through some unspecified means (e.g., configuration), and this address is typically used by many clients simultaneously.

Since the client is behind a NAT, the server sees packets from the client as coming from a transport address on the NAT itself. This address is known as the client's SERVER-REFLEXIVE transport address; packets sent by the server to the client's server-reflexive transport address will be forwarded by the NAT to the client's host transport address.

The client uses TURN commands to create and manipulate an ALLOCATION on the server. An allocation is a data structure on the server. This data structure contains, amongst other things, the RELAYED TRANSPORT ADDRESS for the allocation. The relayed transport address is the transport address on the server that peers can use to have the server relay data to the client. An allocation is uniquely identified by its relayed transport address.

Once an allocation is created, the client can send application data to the server along with an indication of to which peer the data is to be sent, and the server will relay this data to the appropriate peer. The client sends the application data to the server inside a TURN message; at the server, the data is extracted from the TURN message and sent to the peer in a UDP datagram. In the reverse direction, a peer can send application data in a UDP datagram to the relayed transport address for the allocation; the server will then encapsulate this data inside a TURN message and send it to the client along with an indication of which peer sent the data. Since the TURN message always contains an indication of which peer the client is communicating with, the client can use a single allocation to communicate with multiple peers.

When the peer is behind a NAT, then the client must identify the peer using its server-reflexive transport address rather than its host transport address. For example, to send application data to Peer A in the example above, the client must specify 192.0.2.150:32102 (Peer A's server-reflexive transport address) rather than 192.168.100.2:49582 (Peer A's host transport address).

Each allocation on the server belongs to a single client and has exactly one relayed transport address that is used only by that allocation. Thus, when a packet arrives at a relayed transport address on the server, the server knows for which client the data is intended.

The client may have multiple allocations on a server at the same time.

2.1. Transports

TURN, as defined in this specification, always uses UDP between the server and the peer. However, this specification allows the use of any one of UDP, TCP, Transport Layer Security (TLS) over TCP or Datagram Transport Layer Security (DTLS) over UDP to carry the TURN messages between the client and the server.

TURN client to TURN server	TURN server to peer
UDP	UDP
TCP	UDP
TLS-over-TCP	UDP
DTLS-over-UDP	UDP

If TCP or TLS-over-TCP is used between the client and the server, then the server will convert between these transports and UDP transport when relaying data to/from the peer.

Since this version of TURN only supports UDP between the server and the peer, it is expected that most clients will prefer to use UDP between the client and the server as well. That being the case, some readers may wonder: Why also support TCP and TLS-over-TCP?

TURN supports TCP transport between the client and the server because some firewalls are configured to block UDP entirely. These firewalls block UDP but not TCP, in part because TCP has properties that make the intention of the nodes being protected by the firewall more obvious to the firewall. For example, TCP has a three-way handshake that makes in clearer that the protected node really wishes to have that particular connection established, while for UDP the best the firewall can do is guess which flows are desired by using filtering rules. Also, TCP has explicit connection teardown; while for UDP, the firewall has to use timers to guess when the flow is finished.

TURN supports TLS-over-TCP transport and DTLS-over-UDP transport between the client and the server because (D)TLS provides additional security properties not provided by TURN's default digest authentication; properties that some clients may wish to take advantage of. In particular, (D)TLS provides a way for the client to ascertain that it is talking to the correct server, and provides for confidentiality of TURN control messages. TURN does not require (D)TLS because the overhead of using (D)TLS is higher than that of

digest authentication; for example, using (D)TLS likely means that most application data will be doubly encrypted (once by (D)TLS and once to ensure it is still encrypted in the UDP datagram).

There is an extension to TURN for TCP transport between the server and the peers [RFC6062]. For this reason, allocations that use UDP between the server and the peers are known as UDP allocations, while allocations that use TCP between the server and the peers are known as TCP allocations. This specification describes only UDP allocations.

In some applications for TURN, the client may send and receive packets other than TURN packets on the host transport address it uses to communicate with the server. This can happen, for example, when using TURN with ICE. In these cases, the client can distinguish TURN packets from other packets by examining the source address of the arriving packet: those arriving from the TURN server will be TURN packets.

2.2. Allocations

To create an allocation on the server, the client uses an Allocate transaction. The client sends an Allocate request to the server, and the server replies with an Allocate success response containing the allocated relayed transport address. The client can include attributes in the Allocate request that describe the type of allocation it desires (e.g., the lifetime of the allocation). Since relaying data has security implications, the server requires that the client authenticate itself, typically using STUN's long-term credential mechanism, to show that it is authorized to use the server.

Once a relayed transport address is allocated, a client must keep the allocation alive. To do this, the client periodically sends a Refresh request to the server. TURN deliberately uses a different method (Refresh rather than Allocate) for refreshes to ensure that the client is informed if the allocation vanishes for some reason.

The frequency of the Refresh transaction is determined by the lifetime of the allocation. The default lifetime of an allocation is 10 minutes -- this value was chosen to be long enough so that refreshing is not typically a burden on the client, while expiring allocations where the client has unexpectedly quit in a timely manner. However, the client can request a longer lifetime in the Allocate request and may modify its request in a Refresh request, and the server always indicates the actual lifetime in the response. The client must issue a new Refresh transaction within "lifetime" seconds of the previous Allocate or Refresh transaction. Once a client no

longer wishes to use an allocation, it should delete the allocation using a Refresh request with a requested lifetime of 0.

Both the server and client keep track of a value known as the 5-TUPLE. At the client, the 5-tuple consists of the client's host transport address, the server transport address, and the transport protocol used by the client to communicate with the server. At the server, the 5-tuple value is the same except that the client's host transport address is replaced by the client's server-reflexive address, since that is the client's address as seen by the server.

Both the client and the server remember the 5-tuple used in the Allocate request. Subsequent messages between the client and the server use the same 5-tuple. In this way, the client and server know which allocation is being referred to. If the client wishes to allocate a second relayed transport address, it must create a second allocation using a different 5-tuple (e.g., by using a different client host address or port).

> NOTE: While the terminology used in this document refers to 5-tuples, the TURN server can store whatever identifier it likes that yields identical results. Specifically, an implementation may use a file-descriptor in place of a 5-tuple to represent a TCP connection.

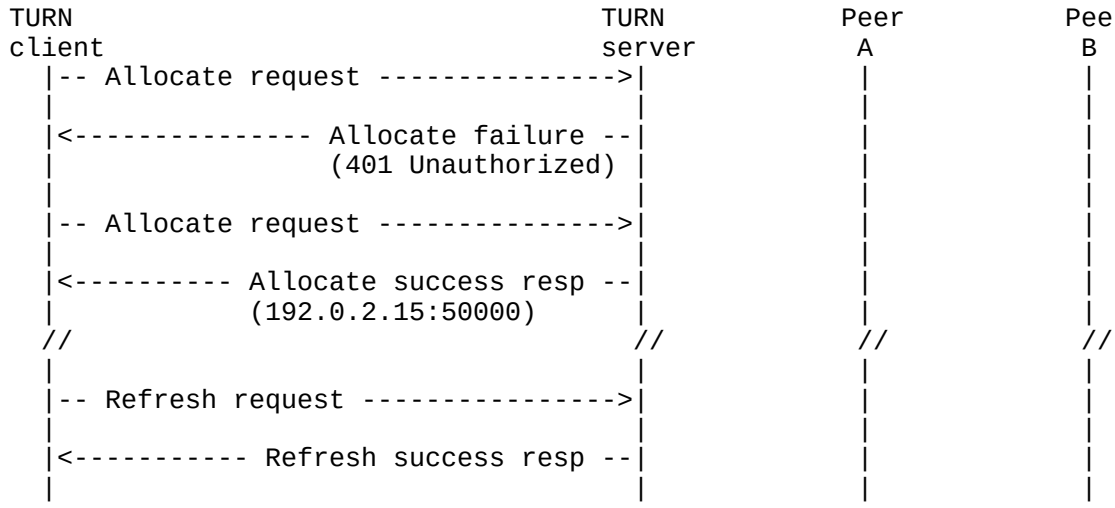


Figure 2

In Figure 2, the client sends an Allocate request to the server without credentials. Since the server requires that all requests be authenticated using STUN's long-term credential mechanism, the server

rejects the request with a 401 (Unauthorized) error code. The client then tries again, this time including credentials (not shown). This time, the server accepts the Allocate request and returns an Allocate success response containing (amongst other things) the relayed transport address assigned to the allocation. Sometime later, the client decides to refresh the allocation and thus sends a Refresh request to the server. The refresh is accepted and the server replies with a Refresh success response.

2.3. Permissions

To ease concerns amongst enterprise IT administrators that TURN could be used to bypass corporate firewall security, TURN includes the notion of permissions. TURN permissions mimic the address-restricted filtering mechanism of NATs that comply with [RFC4787].

An allocation can have zero or more permissions. Each permission consists of an IP address and a lifetime. When the server receives a UDP datagram on the allocation’s relayed transport address, it first checks the list of permissions. If the source IP address of the datagram matches a permission, the application data is relayed to the client, otherwise the UDP datagram is silently discarded.

A permission expires after 5 minutes if it is not refreshed, and there is no way to explicitly delete a permission. This behavior was selected to match the behavior of a NAT that complies with [RFC4787].

The client can install or refresh a permission using either a CreatePermission request or a ChannelBind request. Using the CreatePermission request, multiple permissions can be installed or refreshed with a single request -- this is important for applications that use ICE. For security reasons, permissions can only be installed or refreshed by transactions that can be authenticated; thus, Send indications and ChannelData messages (which are used to send data to peers) do not install or refresh any permissions.

Note that permissions are within the context of an allocation, so adding or expiring a permission in one allocation does not affect other allocations.

2.4. Send Mechanism

There are two mechanisms for the client and peers to exchange application data using the TURN server. The first mechanism uses the Send and Data methods, the second way uses channels. Common to both ways is the ability of the client to communicate with multiple peers using a single allocated relayed transport address; thus, both ways include a means for the client to indicate to the server which peer

should receive the data, and for the server to indicate to the client which peer sent the data.

The Send mechanism uses Send and Data indications. Send indications are used to send application data from the client to the server, while Data indications are used to send application data from the server to the client.

When using the Send mechanism, the client sends a Send indication to the TURN server containing (a) an XOR-PEER-ADDRESS attribute specifying the (server-reflexive) transport address of the peer and (b) a DATA attribute holding the application data. When the TURN server receives the Send indication, it extracts the application data from the DATA attribute and sends it in a UDP datagram to the peer, using the allocated relay address as the source address. Note that there is no need to specify the relayed transport address, since it is implied by the 5-tuple used for the Send indication.

In the reverse direction, UDP datagrams arriving at the relayed transport address on the TURN server are converted into Data indications and sent to the client, with the server-reflexive transport address of the peer included in an XOR-PEER-ADDRESS attribute and the data itself in a DATA attribute. Since the relayed transport address uniquely identified the allocation, the server knows which client should receive the data.

Send and Data indications cannot be authenticated, since the long-term credential mechanism of STUN does not support authenticating indications. This is not as big an issue as it might first appear, since the client-to-server leg is only half of the total path to the peer. Applications that want proper security should encrypt the data sent between the client and a peer.

Because Send indications are not authenticated, it is possible for an attacker to send bogus Send indications to the server, which will then relay these to a peer. To partly mitigate this attack, TURN requires that the client install a permission towards a peer before sending data to it using a Send indication.

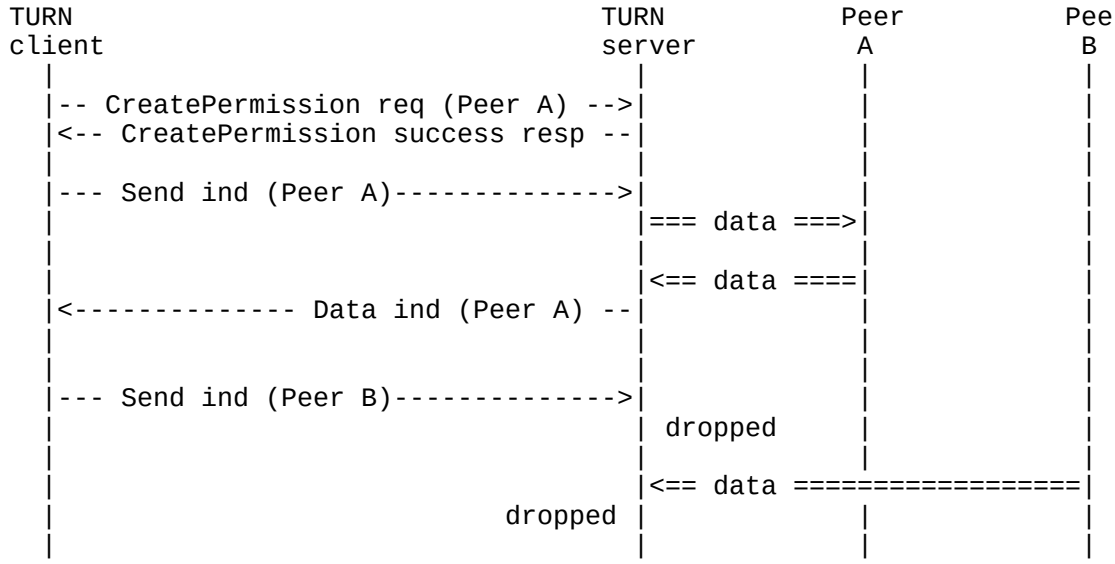


Figure 3

In Figure 3, the client has already created an allocation and now wishes to send data to its peers. The client first creates a permission by sending the server a CreatePermission request specifying Peer A's (server-reflexive) IP address in the XOR-PEER-ADDRESS attribute; if this was not done, the server would not relay data between the client and the server. The client then sends data to Peer A using a Send indication; at the server, the application data is extracted and forwarded in a UDP datagram to Peer A, using the relayed transport address as the source transport address. When a UDP datagram from Peer A is received at the relayed transport address, the contents are placed into a Data indication and forwarded to the client. Later, the client attempts to exchange data with Peer B; however, no permission has been installed for Peer B, so the Send indication from the client and the UDP datagram from the peer are both dropped by the server.

2.5. Channels

For some applications (e.g., Voice over IP), the 36 bytes of overhead that a Send indication or Data indication adds to the application data can substantially increase the bandwidth required between the client and the server. To remedy this, TURN offers a second way for the client and server to associate data with a specific peer.

This second way uses an alternate packet format known as the ChannelData message. The ChannelData message does not use the STUN

header used by other TURN messages, but instead has a 4-byte header that includes a number known as a channel number. Each channel number in use is bound to a specific peer and thus serves as a shorthand for the peer's host transport address.

To bind a channel to a peer, the client sends a ChannelBind request to the server, and includes an unbound channel number and the transport address of the peer. Once the channel is bound, the client can use a ChannelData message to send the server data destined for the peer. Similarly, the server can relay data from that peer towards the client using a ChannelData message.

Channel bindings last for 10 minutes unless refreshed -- this lifetime was chosen to be longer than the permission lifetime. Channel bindings are refreshed by sending another ChannelBind request rebinding the channel to the peer. Like permissions (but unlike allocations), there is no way to explicitly delete a channel binding; the client must simply wait for it to time out.

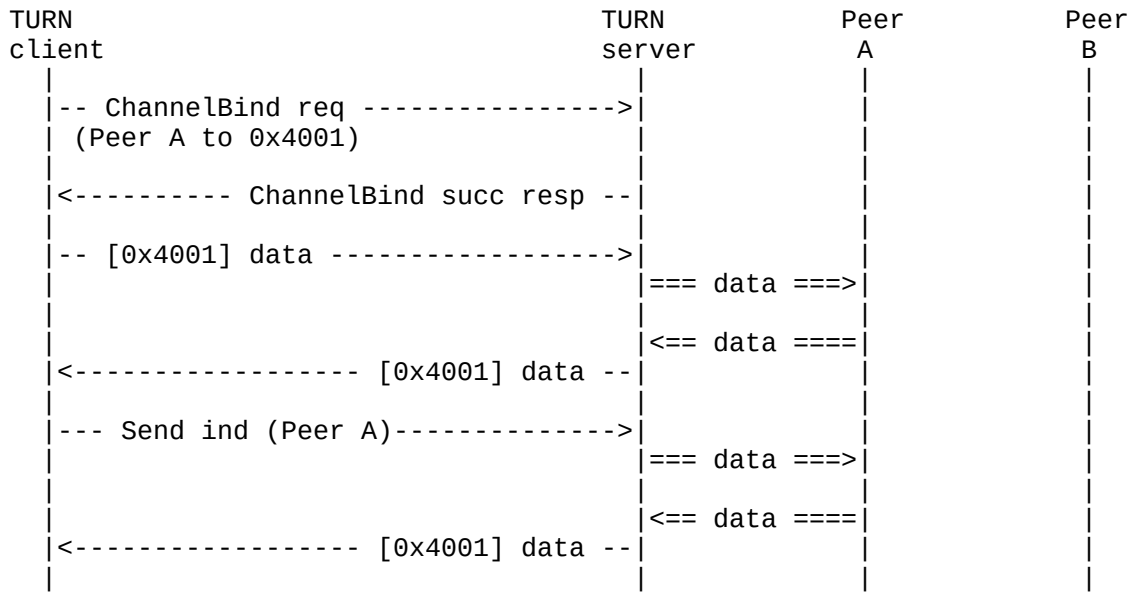


Figure 4

Figure 4 shows the channel mechanism in use. The client has already created an allocation and now wishes to bind a channel to Peer A. To do this, the client sends a ChannelBind request to the server, specifying the transport address of Peer A and a channel number (0x4001). After that, the client can send application data encapsulated inside ChannelData messages to Peer A: this is shown as

"[0x4001] data" where 0x4001 is the channel number. When the ChannelData message arrives at the server, the server transfers the data to a UDP datagram and sends it to Peer A (which is the peer bound to channel number 0x4001).

In the reverse direction, when Peer A sends a UDP datagram to the relayed transport address, this UDP datagram arrives at the server on the relayed transport address assigned to the allocation. Since the UDP datagram was received from Peer A, which has a channel number assigned to it, the server encapsulates the data into a ChannelData message when sending the data to the client.

Once a channel has been bound, the client is free to intermix ChannelData messages and Send indications. In the figure, the client later decides to use a Send indication rather than a ChannelData message to send additional data to Peer A. The client might decide to do this, for example, so it can use the DONT-FRAGMENT attribute (see the next section). However, once a channel is bound, the server will always use a ChannelData message, as shown in the call flow.

Note that ChannelData messages can only be used for peers to which the client has bound a channel. In the example above, Peer A has been bound to a channel, but Peer B has not, so application data to and from Peer B would use the Send mechanism.

2.6. Unprivileged TURN Servers

This version of TURN is designed so that the server can be implemented as an application that runs in user space under commonly available operating systems without requiring special privileges. This design decision was made to make it easy to deploy a TURN server: for example, to allow a TURN server to be integrated into a peer-to-peer application so that one peer can offer NAT traversal services to another peer.

This design decision has the following implications for data relayed by a TURN server:

- The value of the Diffserv field may not be preserved across the server;
- The Time to Live (TTL) field may be reset, rather than decremented, across the server;
- The Explicit Congestion Notification (ECN) field may be reset by the server;
- ICMP messages are not relayed by the server;

- o There is no end-to-end fragmentation, since the packet is re-assembled at the server.

Future work may specify alternate TURN semantics that address these limitations.

2.7. Avoiding IP Fragmentation

For reasons described in [Frag-Harmful], applications, especially those sending large volumes of data, should try hard to avoid having their packets fragmented. Applications using TCP can more or less ignore this issue because fragmentation avoidance is now a standard part of TCP, but applications using UDP (and thus any application using this version of TURN) must handle fragmentation avoidance themselves.

The application running on the client and the peer can take one of two approaches to avoid IP fragmentation.

The first approach is to avoid sending large amounts of application data in the TURN messages/UDP datagrams exchanged between the client and the peer. This is the approach taken by most VoIP (Voice-over-IP) applications. In this approach, the application exploits the fact that the IP specification [RFC0791] specifies that IP packets up to 576 bytes should never need to be fragmented.

The exact amount of application data that can be included while avoiding fragmentation depends on the details of the TURN session between the client and the server: whether UDP, TCP, or (D)TLS transport is used, whether ChannelData messages or Send/Data indications are used, and whether any additional attributes (such as the DONT-FRAGMENT attribute) are included. Another factor, which is hard to determine, is whether the MTU is reduced somewhere along the path for other reasons, such as the use of IP-in-IP tunneling.

As a guideline, sending a maximum of 500 bytes of application data in a single TURN message (by the client on the client-to-server leg) or a UDP datagram (by the peer on the peer-to-server leg) will generally avoid IP fragmentation. To further reduce the chance of fragmentation, it is recommended that the client use ChannelData messages when transferring significant volumes of data, since the overhead of the ChannelData message is less than Send and Data indications.

The second approach the client and peer can take to avoid fragmentation is to use a path MTU discovery algorithm to determine the maximum amount of application data that can be sent without fragmentation.

Unfortunately, because servers implementing this version of TURN do not relay ICMP messages, the classic path MTU discovery algorithm defined in [RFC1191] is not able to discover the MTU of the transmission path between the client and the peer. (Even if they did relay ICMP messages, the algorithm would not always work since ICMP messages are often filtered out by combined NAT/firewall devices).

So the client and server need to use a path MTU discovery algorithm that does not require ICMP messages. The Packetized Path MTU Discovery algorithm defined in [RFC4821] is one such algorithm.

The details of how to use the algorithm of [RFC4821] with TURN are still under investigation. However, as a step towards this goal, this version of TURN supports a DONT-FRAGMENT attribute. When the client includes this attribute in a Send indication, this tells the server to set the DF bit in the resulting UDP datagram that it sends to the peer. Since some servers may be unable to set the DF bit, the client should also include this attribute in the Allocate request -- any server that does not support the DONT-FRAGMENT attribute will indicate this by rejecting the Allocate request.

2.8. RTP Support

One of the envisioned uses of TURN is as a relay for clients and peers wishing to exchange real-time data (e.g., voice or video) using RTP. To facilitate the use of TURN for this purpose, TURN includes some special support for older versions of RTP.

Old versions of RTP [RFC3550] required that the RTP stream be on an even port number and the associated RTP Control Protocol (RTCP) stream, if present, be on the next highest port. To allow clients to work with peers that still require this, TURN allows the client to request that the server allocate a relayed transport address with an even port number, and to optionally request the server reserve the next-highest port number for a subsequent allocation.

2.9. Discovery of Servers

Methods of TURN server discovery, including using anycast, are described in [I-D.ietf-tram-turn-server-discovery].

3. Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [RFC2119].

Readers are expected to be familiar with [RFC5389] and the terms defined there.

The following terms are used in this document:

TURN: The protocol spoken between a TURN client and a TURN server. It is an extension to the STUN protocol [RFC5389]. The protocol allows a client to allocate and use a relayed transport address.

TURN client: A STUN client that implements this specification.

TURN server: A STUN server that implements this specification. It relays data between a TURN client and its peer(s).

Peer: A host with which the TURN client wishes to communicate. The TURN server relays traffic between the TURN client and its peer(s). The peer does not interact with the TURN server using the protocol defined in this document; rather, the peer receives data sent by the TURN server and the peer sends data towards the TURN server.

Transport Address: The combination of an IP address and a port.

Host Transport Address: A transport address on a client or a peer.

Server-Reflexive Transport Address: A transport address on the "public side" of a NAT. This address is allocated by the NAT to correspond to a specific host transport address.

Relayed Transport Address: A transport address on the TURN server that is used for relaying packets between the client and a peer. A peer sends to this address on the TURN server, and the packet is then relayed to the client.

TURN Server Transport Address: A transport address on the TURN server that is used for sending TURN messages to the server. This is the transport address that the client uses to communicate with the server.

Peer Transport Address: The transport address of the peer as seen by the server. When the peer is behind a NAT, this is the peer’s server-reflexive transport address.

Allocation: The relayed transport address granted to a client through an Allocate request, along with related state, such as permissions and expiration timers.

5-tuple: The combination (client IP address and port, server IP address and port, and transport protocol (currently one of UDP, TCP, or (D)TLS)) used to communicate between the client and the server. The 5-tuple uniquely identifies this communication stream. The 5-tuple also uniquely identifies the Allocation on the server.

Channel: A channel number and associated peer transport address. Once a channel number is bound to a peer's transport address, the client and server can use the more bandwidth-efficient ChannelData message to exchange data.

Permission: The IP address and transport protocol (but not the port) of a peer that is permitted to send traffic to the TURN server and have that traffic relayed to the TURN client. The TURN server will only forward traffic to its client from peers that match an existing permission.

Realm: A string used to describe the server or a context within the server. The realm tells the client which username and password combination to use to authenticate requests.

Nonce: A string chosen at random by the server and included in the message-digest. To prevent reply attacks, the server should change the nonce regularly.

4. General Behavior

This section contains general TURN processing rules that apply to all TURN messages.

TURN is an extension to STUN. All TURN messages, with the exception of the ChannelData message, are STUN-formatted messages. All the base processing rules described in [RFC5389] apply to STUN-formatted messages. This means that all the message-forming and message-processing descriptions in this document are implicitly prefixed with the rules of [RFC5389].

[RFC5389] specifies an authentication mechanism called the long-term credential mechanism. TURN servers and clients MUST implement this mechanism. The server MUST demand that all requests from the client be authenticated using this mechanism, or that a equally strong or stronger mechanism for client authentication is used.

Note that the long-term credential mechanism applies only to requests and cannot be used to authenticate indications; thus, indications in TURN are never authenticated. If the server requires requests to be authenticated, then the server's administrator MUST choose a realm

value that will uniquely identify the username and password combination that the client must use, even if the client uses multiple servers under different administrations. The server's administrator MAY choose to allocate a unique username to each client, or MAY choose to allocate the same username to more than one client (for example, to all clients from the same department or company). For each allocation, the server SHOULD generate a new random nonce when the allocation is first attempted following the randomness recommendations in [RFC4086] and SHOULD expire the nonce at least once every hour during the lifetime of the allocation.

All requests after the initial Allocate must use the same username as that used to create the allocation, to prevent attackers from hijacking the client's allocation. Specifically, if the server requires the use of the long-term credential mechanism, and if a non-Allocate request passes authentication under this mechanism, and if the 5-tuple identifies an existing allocation, but the request does not use the same username as used to create the allocation, then the request MUST be rejected with a 441 (Wrong Credentials) error.

When a TURN message arrives at the server from the client, the server uses the 5-tuple in the message to identify the associated allocation. For all TURN messages (including ChannelData) EXCEPT an Allocate request, if the 5-tuple does not identify an existing allocation, then the message MUST either be rejected with a 437 Allocation Mismatch error (if it is a request) or silently ignored (if it is an indication or a ChannelData message). A client receiving a 437 error response to a request other than Allocate MUST assume the allocation no longer exists.

[RFC5389] defines a number of attributes, including the SOFTWARE and FINGERPRINT attributes. The client SHOULD include the SOFTWARE attribute in all Allocate and Refresh requests and MAY include it in any other requests or indications. The server SHOULD include the SOFTWARE attribute in all Allocate and Refresh responses (either success or failure) and MAY include it in other responses or indications. The client and the server MAY include the FINGERPRINT attribute in any STUN-formatted messages defined in this document.

TURN does not use the backwards-compatibility mechanism described in [RFC5389].

TURN, as defined in this specification, supports both IPv4 and IPv6. IPv6 support in TURN includes IPv4-to-IPv6, IPv6-to-IPv6, and IPv6-to-IPv4 relaying. The REQUESTED-ADDRESS-FAMILY attribute allows a client to explicitly request the address type the TURN server will allocate (e.g., an IPv4-only node may request the TURN server to allocate an IPv6 address). The ADDITIONAL-ADDRESS-FAMILY attribute

allows a client to request the server to allocate one IPv4 and one IPv6 relay address in a single Allocate request. This saves local ports on the client and reduces the number of messages sent between the client and the TURN server.

By default, TURN runs on the same ports as STUN: 3478 for TURN over UDP and TCP, and 5349 for TURN over (D)TLS. However, TURN has its own set of Service Record (SRV) names: "turn" for UDP and TCP, and "turns" for (D)TLS. Either the SRV procedures or the ALTERNATE-SERVER procedures, both described in Section 6, can be used to run TURN on a different port.

To ensure interoperability, a TURN server MUST support the use of UDP transport between the client and the server, and SHOULD support the use of TCP and (D)TLS transport.

When UDP transport is used between the client and the server, the client will retransmit a request if it does not receive a response within a certain timeout period. Because of this, the server may receive two (or more) requests with the same 5-tuple and same transaction id. STUN requires that the server recognize this case and treat the request as idempotent (see [RFC5389]). Some implementations may choose to meet this requirement by remembering all received requests and the corresponding responses for 40 seconds. Other implementations may choose to reprocess the request and arrange that such reprocessing returns essentially the same response. To aid implementors who choose the latter approach (the so-called "stateless stack approach"), this specification includes some implementation notes on how this might be done. Implementations are free to choose either approach or choose some other approach that gives the same results.

When TCP transport is used between the client and the server, it is possible that a bit error will cause a length field in a TURN packet to become corrupted, causing the receiver to lose synchronization with the incoming stream of TURN messages. A client or server that detects a long sequence of invalid TURN messages over TCP transport SHOULD close the corresponding TCP connection to help the other end detect this situation more rapidly.

To mitigate either intentional or unintentional denial-of-service attacks against the server by clients with valid usernames and passwords, it is RECOMMENDED that the server impose limits on both the number of allocations active at one time for a given username and on the amount of bandwidth those allocations can use. The server should reject new allocations that would exceed the limit on the allowed number of allocations active at one time with a 486

(Allocation Quota Exceeded) (see Section 6.2), and should discard application data traffic that exceeds the bandwidth quota.

5. Allocations

All TURN operations revolve around allocations, and all TURN messages are associated with an allocation. An allocation conceptually consists of the following state data:

- the relayed transport address;
- the 5-tuple: (client's IP address, client's port, server IP address, server port, transport protocol);
- the authentication information;
- the time-to-expiry;
- a list of permissions;
- a list of channel to peer bindings.

The relayed transport address is the transport address allocated by the server for communicating with peers, while the 5-tuple describes the communication path between the client and the server. On the client, the 5-tuple uses the client's host transport address; on the server, the 5-tuple uses the client's server-reflexive transport address.

Both the relayed transport address and the 5-tuple MUST be unique across all allocations, so either one can be used to uniquely identify the allocation.

The authentication information (e.g., username, password, realm, and nonce) is used to both verify subsequent requests and to compute the message integrity of responses. The username, realm, and nonce values are initially those used in the authenticated Allocate request that creates the allocation, though the server can change the nonce value during the lifetime of the allocation using a 438 (Stale Nonce) reply. Note that, rather than storing the password explicitly, for security reasons, it may be desirable for the server to store the key value, which is an MD5 hash over the username, realm, and password (see [RFC5389]).

Editor's Note: Remove MD5 based on the changes in STUN bis draft.

The time-to-expiry is the time in seconds left until the allocation expires. Each Allocate or Refresh transaction sets this timer, which

then ticks down towards 0. By default, each Allocate or Refresh transaction resets this timer to the default lifetime value of 600 seconds (10 minutes), but the client can request a different value in the Allocate and Refresh request. Allocations can only be refreshed using the Refresh request; sending data to a peer does not refresh an allocation. When an allocation expires, the state data associated with the allocation can be freed.

The list of permissions is described in Section 8 and the list of channels is described in Section 11.

6. Creating an Allocation

An allocation on the server is created using an Allocate transaction.

6.1. Sending an Allocate Request

The client forms an Allocate request as follows.

The client first picks a host transport address. It is RECOMMENDED that the client pick a currently unused transport address, typically by allowing the underlying OS to pick a currently unused port for a new socket.

The client then picks a transport protocol to use between the client and the server. The transport protocol MUST be one of UDP, TCP, TLS-over-TCP or DTLS-over-UDP. Since this specification only allows UDP between the server and the peers, it is RECOMMENDED that the client pick UDP unless it has a reason to use a different transport. One reason to pick a different transport would be that the client believes, either through configuration or by experiment, that it is unable to contact any TURN server using UDP. See Section 2.1 for more discussion.

The client also picks a server transport address, which SHOULD be done as follows. The client receives (perhaps through configuration) a domain name for a TURN server. The client then uses the DNS procedures described in [RFC5389], but using an SRV service name of "turn" (or "turns" for TURN over (D)TLS) instead of "stun" (or "stuns"). For example, to find servers in the example.com domain, the client performs a lookup for ’_turn._udp.example.com’, ’_turn._tcp.example.com’, and ’_turns._tcp.example.com’ if the client wants to communicate with the server using UDP, TCP, TLS-over-TCP, or DTLS-over-UDP, respectively.

The client MUST include a REQUESTED-TRANSPORT attribute in the request. This attribute specifies the transport protocol between the server and the peers (note that this is NOT the transport protocol

that appears in the 5-tuple). In this specification, the REQUESTED-TRANSPORT type is always UDP. This attribute is included to allow future extensions to specify other protocols.

If the client wishes to obtain a relayed transport address of a specific address type then it includes a REQUESTED-ADDRESS-FAMILY attribute in the request. This attribute indicates the specific address type the client wishes the TURN server to allocate. Clients MUST NOT include more than one REQUESTED-ADDRESS-FAMILY attribute in an Allocate request. Clients MUST NOT include a REQUESTED-ADDRESS-FAMILY attribute in an Allocate request that contains a RESERVATION-TOKEN attribute, for the reasons outlined in [RFC6156].

If the client wishes to obtain one IPv6 and one IPv4 relayed transport addresses then it includes an ADDITIONAL-ADDRESS-FAMILY attribute in the request. This attribute specifies that the server must allocate both address types. The attribute value in the ADDITIONAL-ADDRESS-FAMILY MUST be set to 0x02 (IPv6 address family). Clients MUST NOT include REQUESTED-ADDRESS-FAMILY and ADDITIONAL-ADDRESS-FAMILY attributes in the same request. Clients MUST NOT include ADDITIONAL-ADDRESS-FAMILY attribute in a Allocate request that contains a RESERVATION-TOKEN attribute. Clients MUST NOT include ADDITIONAL-ADDRESS-FAMILY attribute in a Allocate request that contains a EVEN-PORT attribute with the R bit set to 1.

If the client wishes the server to initialize the time-to-expiry field of the allocation to some value other than the default lifetime, then it MAY include a LIFETIME attribute specifying its desired value. This is just a hint, and the server may elect to use a different value. Note that the server will ignore requests to initialize the field to less than the default value.

If the client wishes to later use the DONT-FRAGMENT attribute in one or more Send indications on this allocation, then the client SHOULD include the DONT-FRAGMENT attribute in the Allocate request. This allows the client to test whether this attribute is supported by the server.

If the client requires the port number of the relayed transport address be even, the client includes the EVEN-PORT attribute. If this attribute is not included, then the port can be even or odd. By setting the R bit in the EVEN-PORT attribute to 1, the client can request that the server reserve the next highest port number (on the same IP address) for a subsequent allocation. If the R bit is 0, no such request is made.

The client MAY also include a RESERVATION-TOKEN attribute in the request to ask the server to use a previously reserved port for the

allocation. If the RESERVATION-TOKEN attribute is included, then the client MUST omit the EVEN-PORT attribute.

Once constructed, the client sends the Allocate request on the 5-tuple.

6.2. Receiving an Allocate Request

When the server receives an Allocate request, it performs the following checks:

1. The server MUST require that the request be authenticated. This authentication MUST be done using the long-term credential mechanism of [RFC5389] unless the client and server agree to use another mechanism through some procedure outside the scope of this document.

2. The server checks if the 5-tuple is currently in use by an existing allocation. If yes, the server rejects the request with a 437 (Allocation Mismatch) error.

3. The server checks if the request contains a REQUESTED-TRANSPORT attribute. If the REQUESTED-TRANSPORT attribute is not included or is malformed, the server rejects the request with a 400 (Bad Request) error. Otherwise, if the attribute is included but specifies a protocol other that UDP, the server rejects the request with a 442 (Unsupported Transport Protocol) error.

4. The request may contain a DONT-FRAGMENT attribute. If it does, but the server does not support sending UDP datagrams with the DF bit set to 1 (see Section 13), then the server treats the DONT-FRAGMENT attribute in the Allocate request as an unknown comprehension-required attribute.

5. The server checks if the request contains a RESERVATION-TOKEN attribute. If yes, and the request also contains an EVEN-PORT or REQUESTED-ADDRESS-FAMILY or ADDITIONAL-ADDRESS-FAMILY attribute, the server rejects the request with a 400 (Bad Request) error. Otherwise, it checks to see if the token is valid (i.e., the token is in range and has not expired and the corresponding relayed transport address is still available). If the token is not valid for some reason, the server rejects the request with a 508 (Insufficient Capacity) error.

6. The server checks if the request contains both REQUESTED-ADDRESS-FAMILY and ADDITIONAL-ADDRESS-FAMILY attributes, then the server rejects the request with a 400 (Bad Request) error.

7. If the server does not support the address family requested by the client in REQUESTED-ADDRESS-FAMILY or is disabled by local policy, it MUST generate an Allocate error response, and it MUST include an ERROR-CODE attribute with the 440 (Address Family not Supported) response code. If the REQUESTED-ADDRESS-FAMILY attribute is absent, the server MUST allocate an IPv4 relayed transport address for the TURN client.

8. The server checks if the request contains an EVEN-PORT attribute with the R bit set to 1. If yes, and the request also contains an ADDITIONAL- ADDRESS-FAMILY attribute, the server rejects the request with a 400 (Bad Request) error. Otherwise, the server checks if it can satisfy the request (i.e., can allocate a relayed transport address as described below). If the server cannot satisfy the request, then the server rejects the request with a 508 (Insufficient Capacity) error.

9. The server checks if the request contains an ADDITIONAL-ADDRESS-FAMILY attribute. If yes, and the attribute value is 0x01 (IPv4 address family), then the server rejects the request with a 400 (Bad Request) error. Otherwise, and the server checks if it can allocate relayed transport addresses of both address types. If the server cannot satisfy the request, then the server rejects the request with a 508 (Insufficient Capacity) error. If the server can partially meet the request, i.e. if it can only allocate one relayed transport address of a specific address type, then it includes ADDRESS-ERROR-CODE attribute in the response to inform the client the reason for partial failure of the request. The error code value signaled in the ADDRESS-ERROR-CODE attribute could be 440 (Address Family not Supported) or 508 (Insufficient Capacity).

10. At any point, the server MAY choose to reject the request with a 486 (Allocation Quota Reached) error if it feels the client is trying to exceed some locally defined allocation quota. The server is free to define this allocation quota any way it wishes, but SHOULD define it based on the username used to authenticate the request, and not on the client’s transport address.

11. Also at any point, the server MAY choose to reject the request with a 300 (Try Alternate) error if it wishes to redirect the client to a different server. The use of this error code and attribute follow the specification in [RFC5389].

If all the checks pass, the server creates the allocation. The 5-tuple is set to the 5-tuple from the Allocate request, while the list of permissions and the list of channels are initially empty.

The server chooses a relayed transport address for the allocation as follows:

- If the request contains a RESERVATION-TOKEN attribute, the server uses the previously reserved transport address corresponding to the included token (if it is still available). Note that the reservation is a server-wide reservation and is not specific to a particular allocation, since the Allocate request containing the RESERVATION-TOKEN uses a different 5-tuple than the Allocate request that made the reservation. The 5-tuple for the Allocate request containing the RESERVATION-TOKEN attribute can be any allowed 5-tuple; it can use a different client IP address and port, a different transport protocol, and even different server IP address and port (provided, of course, that the server IP address and port are ones on which the server is listening for TURN requests).

- If the request contains an EVEN-PORT attribute with the R bit set to 0, then the server allocates a relayed transport address with an even port number.

- If the request contains an EVEN-PORT attribute with the R bit set to 1, then the server looks for a pair of port numbers N and N+1 on the same IP address, where N is even. Port N is used in the current allocation, while the relayed transport address with port N+1 is assigned a token and reserved for a future allocation. The server MUST hold this reservation for at least 30 seconds, and MAY choose to hold longer (e.g., until the allocation with port N expires). The server then includes the token in a RESERVATION-TOKEN attribute in the success response.

- Otherwise, the server allocates any available relayed transport address.

In all cases, the server SHOULD only allocate ports from the range 49152 - 65535 (the Dynamic and/or Private Port range [Port-Numbers]), unless the TURN server application knows, through some means not specified here, that other applications running on the same host as the TURN server application will not be impacted by allocating ports outside this range. This condition can often be satisfied by running the TURN server application on a dedicated machine and/or by arranging that any other applications on the machine allocate ports before the TURN server application starts. In any case, the TURN server SHOULD NOT allocate ports in the range 0 - 1023 (the Well-Known Port range) to discourage clients from using TURN to run standard services.

NOTE: The use of randomized port assignments to avoid certain types of attacks is described in [RFC6056]. It is RECOMMENDED that a TURN server implement a randomized port assignment algorithm from [RFC6056]. This is especially applicable to servers that choose to pre-allocate a number of ports from the underlying OS and then later assign them to allocations; for example, a server may choose this technique to implement the EVEN-PORT attribute.

The server determines the initial value of the time-to-expiry field as follows. If the request contains a LIFETIME attribute, then the server computes the minimum of the client's proposed lifetime and the server's maximum allowed lifetime. If this computed value is greater than the default lifetime, then the server uses the computed lifetime as the initial value of the time-to-expiry field. Otherwise, the server uses the default lifetime. It is RECOMMENDED that the server use a maximum allowed lifetime value of no more than 3600 seconds (1 hour). Servers that implement allocation quotas or charge users for allocations in some way may wish to use a smaller maximum allowed lifetime (perhaps as small as the default lifetime) to more quickly remove orphaned allocations (that is, allocations where the corresponding client has crashed or terminated or the client connection has been lost for some reason). Also, note that the time-to-expiry is recomputed with each successful Refresh request, and thus the value computed here applies only until the first refresh.

Once the allocation is created, the server replies with a success response. The success response contains:

- An XOR-RELAYED-ADDRESS attribute containing the relayed transport address.
- A LIFETIME attribute containing the current value of the time-to-expiry timer.
- A RESERVATION-TOKEN attribute (if a second relayed transport address was reserved).
- An XOR-MAPPED-ADDRESS attribute containing the client's IP address and port (from the 5-tuple).

  NOTE: The XOR-MAPPED-ADDRESS attribute is included in the response as a convenience to the client. TURN itself does not make use of this value, but clients running ICE can often need this value and can thus avoid having to do an extra Binding transaction with some STUN server to learn it.

The response (either success or error) is sent back to the client on the 5-tuple.

NOTE: When the Allocate request is sent over UDP, section 7.3.1 of [RFC5389] requires that the server handle the possible retransmissions of the request so that retransmissions do not cause multiple allocations to be created. Implementations may achieve this using the so-called "stateless stack approach" as follows. To detect retransmissions when the original request was successful in creating an allocation, the server can store the transaction id that created the request with the allocation data and compare it with incoming Allocate requests on the same 5-tuple. Once such a request is detected, the server can stop parsing the request and immediately generate a success response. When building this response, the value of the LIFETIME attribute can be taken from the time-to-expiry field in the allocate state data, even though this value may differ slightly from the LIFETIME value originally returned. In addition, the server may need to store an indication of any reservation token returned in the original response, so that this may be returned in any retransmitted responses.

For the case where the original request was unsuccessful in creating an allocation, the server may choose to do nothing special. Note, however, that there is a rare case where the server rejects the original request but accepts the retransmitted request (because conditions have changed in the brief intervening time period). If the client receives the first failure response, it will ignore the second (success) response and believe that an allocation was not created. An allocation created in this matter will eventually timeout, since the client will not refresh it. Furthermore, if the client later retries with the same 5-tuple but different transaction id, it will receive a 437 (Allocation Mismatch), which will cause it to retry with a different 5-tuple. The server may use a smaller maximum lifetime value to minimize the lifetime of allocations "orphaned" in this manner.

6.3. Receiving an Allocate Success Response

If the client receives an Allocate success response, then it MUST check that the mapped address and the relayed transport address are part of an address family that the client understands and is prepared to handle. If these two addresses are not part of an address family which the client is prepared to handle, then the client MUST delete the allocation (Section 7) and MUST NOT attempt to create another allocation on that server until it believes the mismatch has been fixed.

Otherwise, the client creates its own copy of the allocation data structure to track what is happening on the server. In particular, the client needs to remember the actual lifetime received back from the server, rather than the value sent to the server in the request. The client must also remember the 5-tuple used for the request and the username and password it used to authenticate the request to ensure that it reuses them for subsequent messages. The client also needs to track the channels and permissions it establishes on the server.

The client will probably wish to send the relayed transport address to peers (using some method not specified here) so the peers can communicate with it. The client may also wish to use the server-reflexive address it receives in the XOR-MAPPED-ADDRESS attribute in its ICE processing.

6.4. Receiving an Allocate Error Response

If the client receives an Allocate error response, then the processing depends on the actual error code returned:

- (Request timed out): There is either a problem with the server, or a problem reaching the server with the chosen transport. The client considers the current transaction as having failed but MAY choose to retry the Allocate request using a different transport (e.g., TCP instead of UDP).

- 300 (Try Alternate): The server would like the client to use the server specified in the ALTERNATE-SERVER attribute instead. The client considers the current transaction as having failed, but SHOULD try the Allocate request with the alternate server before trying any other servers (e.g., other servers discovered using the SRV procedures). When trying the Allocate request with the alternate server, the client follows the ALTERNATE-SERVER procedures specified in [RFC5389].

- 400 (Bad Request): The server believes the client’s request is malformed for some reason. The client considers the current transaction as having failed. The client MAY notify the user or operator and SHOULD NOT retry the request with this server until it believes the problem has been fixed.

- 401 (Unauthorized): If the client has followed the procedures of the long-term credential mechanism and still gets this error, then the server is not accepting the client’s credentials. In this case, the client considers the current transaction as having failed and SHOULD notify the user or operator. The client SHOULD

NOT send any further requests to this server until it believes the problem has been fixed.

- 403 (Forbidden): The request is valid, but the server is refusing to perform it, likely due to administrative restrictions. The client considers the current transaction as having failed. The client MAY notify the user or operator and SHOULD NOT retry the same request with this server until it believes the problem has been fixed.

- 420 (Unknown Attribute): If the client included a DONT-FRAGMENT attribute in the request and the server rejected the request with a 420 error code and listed the DONT-FRAGMENT attribute in the UNKNOWN-ATTRIBUTES attribute in the error response, then the client now knows that the server does not support the DONT-FRAGMENT attribute. The client considers the current transaction as having failed but MAY choose to retry the Allocate request without the DONT-FRAGMENT attribute.

- 437 (Allocation Mismatch): This indicates that the client has picked a 5-tuple that the server sees as already in use. One way this could happen is if an intervening NAT assigned a mapped transport address that was used by another client that recently crashed. The client considers the current transaction as having failed. The client SHOULD pick another client transport address and retry the Allocate request (using a different transaction id). The client SHOULD try three different client transport addresses before giving up on this server. Once the client gives up on the server, it SHOULD NOT try to create another allocation on the server for 2 minutes.

- 438 (Stale Nonce): See the procedures for the long-term credential mechanism [RFC5389].

- 440 (Address Family not Supported): The server does not support the address family requested by the client. If the client receives an Allocate error response with the 440 (Unsupported Address Family) error code, the client MUST NOT retry the request.

- 441 (Wrong Credentials): The client should not receive this error in response to a Allocate request. The client MAY notify the user or operator and SHOULD NOT retry the same request with this server until it believes the problem has been fixed.

- 442 (Unsupported Transport Address): The client should not receive this error in response to a request for a UDP allocation. The client MAY notify the user or operator and SHOULD NOT reattempt

the request with this server until it believes the problem has been fixed.

- o 486 (Allocation Quota Reached): The server is currently unable to create any more allocations with this username. The client considers the current transaction as having failed. The client SHOULD wait at least 1 minute before trying to create any more allocations on the server.

- o 508 (Insufficient Capacity): The server has no more relayed transport addresses available, or has none with the requested properties, or the one that was reserved is no longer available. The client considers the current operation as having failed. If the client is using either the EVEN-PORT or the RESERVATION-TOKEN attribute, then the client MAY choose to remove or modify this attribute and try again immediately. Otherwise, the client SHOULD wait at least 1 minute before trying to create any more allocations on this server.

An unknown error response MUST be handled as described in [RFC5389].

7. Refreshing an Allocation

A Refresh transaction can be used to either (a) refresh an existing allocation and update its time-to-expiry or (b) delete an existing allocation.

If a client wishes to continue using an allocation, then the client MUST refresh it before it expires. It is suggested that the client refresh the allocation roughly 1 minute before it expires. If a client no longer wishes to use an allocation, then it SHOULD explicitly delete the allocation. A client MAY refresh an allocation at any time for other reasons.

7.1. Sending a Refresh Request

If the client wishes to immediately delete an existing allocation, it includes a LIFETIME attribute with a value of 0. All other forms of the request refresh the allocation. The client MUST NOT include any REQUESTED-ADDRESS-FAMILY attribute in its Refresh Request.

When refreshing a dual allocation, the client includes ADDITIONAL-ADDRESS-FAMILY attribute indicating the address family type that should be refreshed. If no ADDITIONAL-ADDRESS-FAMILY is included then the request should be treated as applying to all current allocations. The client MUST only include family types it previously allocated and has not yet deleted. This process can also be used to delete an allocation of a specific address type, by setting the

lifetime of that refresh request to 0. Deleting a single allocation destroys any permissions or channels associated with that particular allocation; it MUST NOT affect any permissions or channels associated with allocations for the other address family.

The Refresh transaction updates the time-to-expiry timer of an allocation. If the client wishes the server to set the time-to-expiry timer to something other than the default lifetime, it includes a LIFETIME attribute with the requested value. The server then computes a new time-to-expiry value in the same way as it does for an Allocate transaction, with the exception that a requested lifetime of 0 causes the server to immediately delete the allocation.

7.2. Receiving a Refresh Request

When the server receives a Refresh request, it processes it as per Section 4 plus the specific rules mentioned here.

If the server receives a Refresh Request with an ADDITIONAL-ADDRESS-FAMILY attribute and the attribute value does not match the address family of the allocation, the server MUST reply with a 443 (Peer Address Family Mismatch) Refresh error response.

The server computes a value called the "desired lifetime" as follows: if the request contains a LIFETIME attribute and the attribute value is 0, then the "desired lifetime" is 0. Otherwise, if the request contains a LIFETIME attribute, then the server computes the minimum of the client's requested lifetime and the server's maximum allowed lifetime. If this computed value is greater than the default lifetime, then the "desired lifetime" is the computed value. Otherwise, the "desired lifetime" is the default lifetime.

Subsequent processing depends on the "desired lifetime" value:

- If the "desired lifetime" is 0, then the request succeeds and the allocation is deleted.

- If the "desired lifetime" is non-zero, then the request succeeds and the allocation's time-to-expiry is set to the "desired lifetime".

If the request succeeds, then the server sends a success response containing:

- A LIFETIME attribute containing the current value of the time-to-expiry timer.

NOTE: A server need not do anything special to implement idempotency of Refresh requests over UDP using the "stateless stack approach". Retransmitted Refresh requests with a non-zero "desired lifetime" will simply refresh the allocation. A retransmitted Refresh request with a zero "desired lifetime" will cause a 437 (Allocation Mismatch) response if the allocation has already been deleted, but the client will treat this as equivalent to a success response (see below).

7.3. Receiving a Refresh Response

If the client receives a success response to its Refresh request with a non-zero lifetime, it updates its copy of the allocation data structure with the time-to-expiry value contained in the response.

If the client receives a 437 (Allocation Mismatch) error response to a request to delete the allocation, then the allocation no longer exists and it should consider its request as having effectively succeeded.

8. Permissions

For each allocation, the server keeps a list of zero or more permissions. Each permission consists of an IP address and an associated time-to-expiry. While a permission exists, all peers using the IP address in the permission are allowed to send data to the client. The time-to-expiry is the number of seconds until the permission expires. Within the context of an allocation, a permission is uniquely identified by its associated IP address.

By sending either CreatePermission requests or ChannelBind requests, the client can cause the server to install or refresh a permission for a given IP address. This causes one of two things to happen:

- If no permission for that IP address exists, then a permission is created with the given IP address and a time-to-expiry equal to Permission Lifetime.
- If a permission for that IP address already exists, then the time-to-expiry for that permission is reset to Permission Lifetime.

The Permission Lifetime MUST be 300 seconds (= 5 minutes).

Each permission’s time-to-expiry decreases down once per second until it reaches 0; at which point, the permission expires and is deleted.

CreatePermission and ChannelBind requests may be freely intermixed on a permission. A given permission may be initially installed and/or

refreshed with a CreatePermission request, and then later refreshed with a ChannelBind request, or vice versa.

When a UDP datagram arrives at the relayed transport address for the allocation, the server extracts the source IP address from the IP header. The server then compares this address with the IP address associated with each permission in the list of permissions for the allocation. If no match is found, relaying is not permitted, and the server silently discards the UDP datagram. If an exact match is found, then the permission check is considered to have succeeded and the server continues to process the UDP datagram as specified elsewhere (Section 10.3). Note that only addresses are compared and port numbers are not considered.

The permissions for one allocation are totally unrelated to the permissions for a different allocation. If an allocation expires, all its permissions expire with it.

> NOTE: Though TURN permissions expire after 5 minutes, many NATs deployed at the time of publication expire their UDP bindings considerably faster. Thus, an application using TURN will probably wish to send some sort of keep-alive traffic at a much faster rate. Applications using ICE should follow the keep-alive guidelines of ICE [RFC5245], and applications not using ICE are advised to do something similar.

9. CreatePermission

TURN supports two ways for the client to install or refresh permissions on the server. This section describes one way: the CreatePermission request.

A CreatePermission request may be used in conjunction with either the Send mechanism in Section 10 or the Channel mechanism in Section 11.

9.1. Forming a CreatePermission Request

The client who wishes to install or refresh one or more permissions can send a CreatePermission request to the server.

When forming a CreatePermission request, the client MUST include at least one XOR-PEER-ADDRESS attribute, and MAY include more than one such attribute. The IP address portion of each XOR-PEER-ADDRESS attribute contains the IP address for which a permission should be installed or refreshed. The port portion of each XOR-PEER-ADDRESS attribute will be ignored and can be any arbitrary value. The various XOR-PEER-ADDRESS attributes can appear in any order. The client MUST only include XOR-PEER-ADDRESS attributes with addresses

of the same address family as that of the relayed transport address for the allocation. For dual allocations obtained using the ADDITIONAL-FAMILY-ADDRESS attribute, the client can include XOR-PEER-ADDRESS attributes with addresses of IPv4 and IPv6 address families.

9.2. Receiving a CreatePermission Request

When the server receives the CreatePermission request, it processes as per Section 4 plus the specific rules mentioned here.

The message is checked for validity. The CreatePermission request MUST contain at least one XOR-PEER-ADDRESS attribute and MAY contain multiple such attributes. If no such attribute exists, or if any of these attributes are invalid, then a 400 (Bad Request) error is returned. If the request is valid, but the server is unable to satisfy the request due to some capacity limit or similar, then a 508 (Insufficient Capacity) error is returned.

If an XOR-PEER-ADDRESS attribute contains an address of an address family that is not the same as that of the relayed transport address for the allocation, the server MUST generate an error response with the 443 (Peer Address Family Mismatch) response code.

The server MAY impose restrictions on the IP address allowed in the XOR-PEER-ADDRESS attribute -- if a value is not allowed, the server rejects the request with a 403 (Forbidden) error.

If the message is valid and the server is capable of carrying out the request, then the server installs or refreshes a permission for the IP address contained in each XOR-PEER-ADDRESS attribute as described in Section 8. The port portion of each attribute is ignored and may be any arbitrary value.

The server then responds with a CreatePermission success response. There are no mandatory attributes in the success response.

> NOTE: A server need not do anything special to implement idempotency of CreatePermission requests over UDP using the "stateless stack approach". Retransmitted CreatePermission requests will simply refresh the permissions.

9.3. Receiving a CreatePermission Response

If the client receives a valid CreatePermission success response, then the client updates its data structures to indicate that the permissions have been installed or refreshed.

10. Send and Data Methods

TURN supports two mechanisms for sending and receiving data from peers. This section describes the use of the Send and Data mechanisms, while Section 11 describes the use of the Channel mechanism.

10.1. Forming a Send Indication

The client can use a Send indication to pass data to the server for relaying to a peer. A client may use a Send indication even if a channel is bound to that peer. However, the client MUST ensure that there is a permission installed for the IP address of the peer to which the Send indication is being sent; this prevents a third party from using a TURN server to send data to arbitrary destinations.

When forming a Send indication, the client MUST include an XOR-PEER-ADDRESS attribute and a DATA attribute. The XOR-PEER-ADDRESS attribute contains the transport address of the peer to which the data is to be sent, and the DATA attribute contains the actual application data to be sent to the peer.

The client MAY include a DONT-FRAGMENT attribute in the Send indication if it wishes the server to set the DF bit on the UDP datagram sent to the peer.

10.2. Receiving a Send Indication

When the server receives a Send indication, it processes as per Section 4 plus the specific rules mentioned here.

The message is first checked for validity. The Send indication MUST contain both an XOR-PEER-ADDRESS attribute and a DATA attribute. If one of these attributes is missing or invalid, then the message is discarded. Note that the DATA attribute is allowed to contain zero bytes of data.

The Send indication may also contain the DONT-FRAGMENT attribute. If the server is unable to set the DF bit on outgoing UDP datagrams when this attribute is present, then the server acts as if the DONT-FRAGMENT attribute is an unknown comprehension-required attribute (and thus the Send indication is discarded).

The server also checks that there is a permission installed for the IP address contained in the XOR-PEER-ADDRESS attribute. If no such permission exists, the message is discarded. Note that a Send indication never causes the server to refresh the permission.

The server MAY impose restrictions on the IP address and port values allowed in the XOR-PEER-ADDRESS attribute -- if a value is not allowed, the server silently discards the Send indication.

If everything is OK, then the server forms a UDP datagram as follows:

- the source transport address is the relayed transport address of the allocation, where the allocation is determined by the 5-tuple on which the Send indication arrived;
- the destination transport address is taken from the XOR-PEER-ADDRESS attribute;
- the data following the UDP header is the contents of the value field of the DATA attribute.

The handling of the DONT-FRAGMENT attribute (if present), is described in Section 13.

The resulting UDP datagram is then sent to the peer.

10.3. Receiving a UDP Datagram

When the server receives a UDP datagram at a currently allocated relayed transport address, the server looks up the allocation associated with the relayed transport address. The server then checks to see whether the set of permissions for the allocation allow the relaying of the UDP datagram as described in Section 8.

If relaying is permitted, then the server checks if there is a channel bound to the peer that sent the UDP datagram (see Section 11). If a channel is bound, then processing proceeds as described in Section 11.7.

If relaying is permitted but no channel is bound to the peer, then the server forms and sends a Data indication. The Data indication MUST contain both an XOR-PEER-ADDRESS and a DATA attribute. The DATA attribute is set to the value of the 'data octets' field from the datagram, and the XOR-PEER-ADDRESS attribute is set to the source transport address of the received UDP datagram. The Data indication is then sent on the 5-tuple associated with the allocation.

10.4. Receiving a Data Indication

When the client receives a Data indication, it checks that the Data indication contains both an XOR-PEER-ADDRESS and a DATA attribute, and discards the indication if it does not. The client SHOULD also check that the XOR-PEER-ADDRESS attribute value contains an IP

address with which the client believes there is an active permission, and discard the Data indication otherwise. Note that the DATA attribute is allowed to contain zero bytes of data.

NOTE: The latter check protects the client against an attacker who somehow manages to trick the server into installing permissions not desired by the client.

If the Data indication passes the above checks, the client delivers the data octets inside the DATA attribute to the application, along with an indication that they were received from the peer whose transport address is given by the XOR-PEER-ADDRESS attribute.

11. Channels

Channels provide a way for the client and server to send application data using ChannelData messages, which have less overhead than Send and Data indications.

The ChannelData message (see Section 11.4) starts with a two-byte field that carries the channel number. The values of this field are allocated as follows:

0x0000 through 0x3FFF: These values can never be used for channel numbers.

0x4000 through 0x7FFF: These values are the allowed channel numbers (16,384 possible values).

0x8000 through 0xFFFF: These values are reserved for future use.

Because of this division, ChannelData messages can be distinguished from STUN-formatted messages (e.g., Allocate request, Send indication, etc.) by examining the first two bits of the message:

0b00: STUN-formatted message (since the first two bits of a STUN-formatted message are always zero).

0b01: ChannelData message (since the channel number is the first field in the ChannelData message and channel numbers fall in the range 0x4000 - 0x7FFF).

0b10: Reserved

0b11: Reserved

The reserved values may be used in the future to extend the range of channel numbers. Thus, an implementation MUST NOT assume that a TURN message always starts with a 0 bit.

Channel bindings are always initiated by the client. The client can bind a channel to a peer at any time during the lifetime of the allocation. The client may bind a channel to a peer before exchanging data with it, or after exchanging data with it (using Send and Data indications) for some time, or may choose never to bind a channel to it. The client can also bind channels to some peers while not binding channels to other peers.

Channel bindings are specific to an allocation, so that the use of a channel number or peer transport address in a channel binding in one allocation has no impact on their use in a different allocation. If an allocation expires, all its channel bindings expire with it.

A channel binding consists of:

- a channel number;
- a transport address (of the peer); and
- A time-to-expiry timer.

Within the context of an allocation, a channel binding is uniquely identified either by the channel number or by the peer's transport address. Thus, the same channel cannot be bound to two different transport addresses, nor can the same transport address be bound to two different channels.

A channel binding lasts for 10 minutes unless refreshed. Refreshing the binding (by the server receiving a ChannelBind request rebinding the channel to the same peer) resets the time-to-expiry timer back to 10 minutes.

When the channel binding expires, the channel becomes unbound. Once unbound, the channel number can be bound to a different transport address, and the transport address can be bound to a different channel number. To prevent race conditions, the client MUST wait 5 minutes after the channel binding expires before attempting to bind the channel number to a different transport address or the transport address to a different channel number.

When binding a channel to a peer, the client SHOULD be prepared to receive ChannelData messages on the channel from the server as soon as it has sent the ChannelBind request. Over UDP, it is possible for

the client to receive ChannelData messages from the server before it receives a ChannelBind success response.

In the other direction, the client MAY elect to send ChannelData messages before receiving the ChannelBind success response. Doing so, however, runs the risk of having the ChannelData messages dropped by the server if the ChannelBind request does not succeed for some reason (e.g., packet lost if the request is sent over UDP, or the server being unable to fulfill the request). A client that wishes to be safe should either queue the data or use Send indications until the channel binding is confirmed.

11.1. Sending a ChannelBind Request

A channel binding is created or refreshed using a ChannelBind transaction. A ChannelBind transaction also creates or refreshes a permission towards the peer (see Section 8).

To initiate the ChannelBind transaction, the client forms a ChannelBind request. The channel to be bound is specified in a CHANNEL-NUMBER attribute, and the peer's transport address is specified in an XOR-PEER-ADDRESS attribute. Section 11.2 describes the restrictions on these attributes. The client MUST only include an XOR-PEER-ADDRESS attribute with an address of the same address family as that of the relayed transport address for the allocation. For dual allocations obtained using the ADDITIONAL-FAMILY-ADDRESS attribute, the client can include XOR-PEER-ADDRESS attributes with addresses of IPv4 and IPv6 address families. When using dual allocation, the peer addresses of those channels may be of different families. Thus, a single 5-tuple session may create several IPv4 channels and several IPv6 channels.

Rebinding a channel to the same transport address that it is already bound to provides a way to refresh a channel binding and the corresponding permission without sending data to the peer. Note however, that permissions need to be refreshed more frequently than channels.

11.2. Receiving a ChannelBind Request

When the server receives a ChannelBind request, it processes as per Section 4 plus the specific rules mentioned here.

The server checks the following:

- The request contains both a CHANNEL-NUMBER and an XOR-PEER-ADDRESS attribute;

- The channel number is in the range 0x4000 through 0x7FFE (inclusive);
- The channel number is not currently bound to a different transport address (same transport address is OK);
- The transport address is not currently bound to a different channel number.
- If the XOR-PEER-ADDRESS attribute contains an address of an address family that is not the same as that of the relayed transport address for the allocation, the server MUST generate an error response with the 443 (Peer Address Family Mismatch) response code.

If any of these tests fail, the server replies with a 400 (Bad Request) error.

The server MAY impose restrictions on the IP address and port values allowed in the XOR-PEER-ADDRESS attribute -- if a value is not allowed, the server rejects the request with a 403 (Forbidden) error.

If the request is valid, but the server is unable to fulfill the request due to some capacity limit or similar, the server replies with a 508 (Insufficient Capacity) error.

Otherwise, the server replies with a ChannelBind success response. There are no required attributes in a successful ChannelBind response.

If the server can satisfy the request, then the server creates or refreshes the channel binding using the channel number in the CHANNEL-NUMBER attribute and the transport address in the XOR-PEER-ADDRESS attribute. The server also installs or refreshes a permission for the IP address in the XOR-PEER-ADDRESS attribute as described in Section 8.

> NOTE: A server need not do anything special to implement idempotency of ChannelBind requests over UDP using the "stateless stack approach". Retransmitted ChannelBind requests will simply refresh the channel binding and the corresponding permission. Furthermore, the client must wait 5 minutes before binding a previously bound channel number or peer address to a different channel, eliminating the possibility that the transaction would initially fail but succeed on a retransmission.

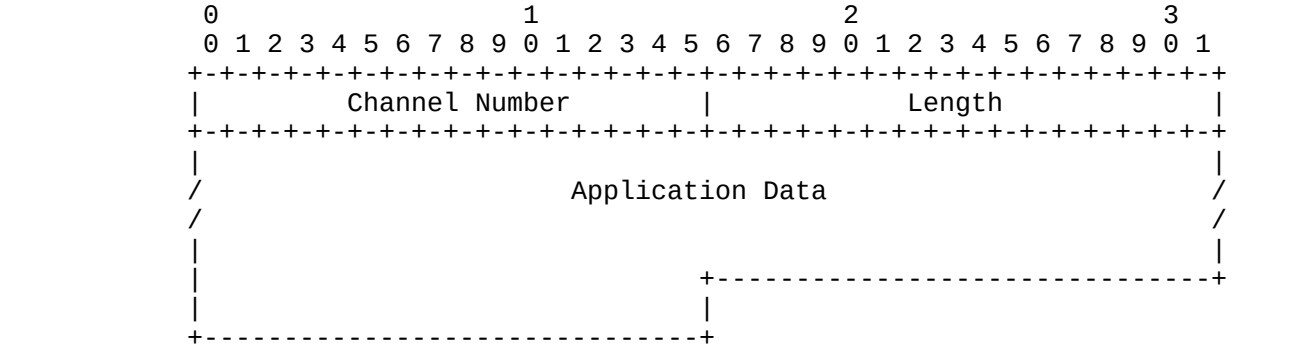
11.3. Receiving a ChannelBind Response

When the client receives a ChannelBind success response, it updates its data structures to record that the channel binding is now active. It also updates its data structures to record that the corresponding permission has been installed or refreshed.

If the client receives a ChannelBind failure response that indicates that the channel information is out-of-sync between the client and the server (e.g., an unexpected 400 "Bad Request" response), then it is RECOMMENDED that the client immediately delete the allocation and start afresh with a new allocation.

11.4. The ChannelData Message

The ChannelData message is used to carry application data between the client and the server. It has the following format:



The Channel Number field specifies the number of the channel on which the data is traveling, and thus the address of the peer that is sending or is to receive the data.

The Length field specifies the length in bytes of the application data field (i.e., it does not include the size of the ChannelData header). Note that 0 is a valid length.

The Application Data field carries the data the client is trying to send to the peer, or that the peer is sending to the client.

11.5. Sending a ChannelData Message

Once a client has bound a channel to a peer, then when the client has data to send to that peer it may use either a ChannelData message or a Send indication; that is, the client is not obligated to use the channel when it exists and may freely intermix the two message types

when sending data to the peer. The server, on the other hand, MUST use the ChannelData message if a channel has been bound to the peer.

The fields of the ChannelData message are filled in as described in Section 11.4.

Over TCP and TLS-over-TCP, the ChannelData message MUST be padded to a multiple of four bytes in order to ensure the alignment of subsequent messages. The padding is not reflected in the length field of the ChannelData message, so the actual size of a ChannelData message (including padding) is (4 + Length) rounded up to the nearest multiple of 4. Over UDP, the padding is not required but MAY be included.

The ChannelData message is then sent on the 5-tuple associated with the allocation.

11.6. Receiving a ChannelData Message

The receiver of the ChannelData message uses the first two bits to distinguish it from STUN-formatted messages, as described above. If the message uses a value in the reserved range (0x8000 through 0xFFFF), then the message is silently discarded.

If the ChannelData message is received in a UDP datagram, and if the UDP datagram is too short to contain the claimed length of the ChannelData message (i.e., the UDP header length field value is less than the ChannelData header length field value + 4 + 8), then the message is silently discarded.

If the ChannelData message is received over TCP or over TLS-over-TCP, then the actual length of the ChannelData message is as described in Section 11.5.

If the ChannelData message is received on a channel that is not bound to any peer, then the message is silently discarded.

On the client, it is RECOMMENDED that the client discard the ChannelData message if the client believes there is no active permission towards the peer. On the server, the receipt of a ChannelData message MUST NOT refresh either the channel binding or the permission towards the peer.

On the server, if no errors are detected, the server relays the application data to the peer by forming a UDP datagram as follows:

- the source transport address is the relayed transport address of the allocation, where the allocation is determined by the 5-tuple on which the ChannelData message arrived;

- the destination transport address is the transport address to which the channel is bound;

- the data following the UDP header is the contents of the data field of the ChannelData message.

The resulting UDP datagram is then sent to the peer. Note that if the Length field in the ChannelData message is 0, then there will be no data in the UDP datagram, but the UDP datagram is still formed and sent.

11.7. Relaying Data from the Peer

When the server receives a UDP datagram on the relayed transport address associated with an allocation, the server processes it as described in Section 10.3. If that section indicates that a ChannelData message should be sent (because there is a channel bound to the peer that sent to the UDP datagram), then the server forms and sends a ChannelData message as described in Section 11.5.

12. Packet Translations

As discussed in Section 2.6, translations in TURN are designed so that a TURN server can be implemented as an application that runs in userland under commonly available operating systems and that does not require special privileges. The translations specified in the following sections follow this principle.

The descriptions below have two parts: a preferred behavior and an alternate behavior. The server SHOULD implement the preferred behavior. Otherwise, the server MUST implement the alternate behavior and MUST NOT do anything else for the reasons detailed in [RFC6145].

12.1. IPv4-to-IPv6 Translations

Traffic Class

Preferred behavior: As specified in Section 4 of [RFC6145].

Alternate behavior: The relay sets the Traffic Class to the default value for outgoing packets.

Flow Label

Preferred behavior: The relay sets the Flow label to 0. The relay can choose to set the Flow label to a different value if it supports the IPv6 Flow Label field[RFC3697].

Alternate behavior: the relay sets the Flow label to the default value for outgoing packets.

Hop Limit

Preferred behavior: As specified in Section 4 of [RFC6145].

Alternate behavior: The relay sets the Hop Limit to the default value for outgoing packets.

Fragmentation

Preferred behavior: As specified in Section 4 of [RFC6145].

Alternate behavior: The relay assembles incoming fragments. The relay follows its default behavior to send outgoing packets.

For both preferred and alternate behavior, the DONT-FRAGMENT attribute MUST be ignored by the server.

Extension Headers

Preferred behavior: The relay sends outgoing packet without any IPv6 extension headers, with the exception of the Fragmentation header as described above.

Alternate behavior: Same as preferred.

12.2. IPv6-to-IPv6 Translations

Flow Label

The relay should consider that it is handling two different IPv6 flows. Therefore, the Flow label [RFC3697] SHOULD NOT be copied as part of the translation.

Preferred behavior: The relay sets the Flow label to 0. The relay can choose to set the Flow label to a different value if it supports the IPv6 Flow Label field[RFC3697].

Alternate behavior: The relay sets the Flow label to the default value for outgoing packets.

Hop Limit

Preferred behavior: The relay acts as a regular router with respect to decrementing the Hop Limit and generating an ICMPv6 error if it reaches zero.

Alternate behavior: The relay sets the Hop Limit to the default value for outgoing packets.

Fragmentation

Preferred behavior: If the incoming packet did not include a Fragment header and the outgoing packet size does not exceed the outgoing link's MTU, the relay sends the outgoing packet without a Fragment header.

If the incoming packet did not include a Fragment header and the outgoing packet size exceeds the outgoing link's MTU, the relay drops the outgoing packet and send an ICMP message of type 2 code 0 ("Packet too big") to the sender of the incoming packet. If the packet is being sent to the peer, the relay reduces the MTU reported in the ICMP message by 48 bytes to allow room for the overhead of a Data indication.

If the incoming packet included a Fragment header and the outgoing packet size (with a Fragment header included) does not exceed the outgoing link's MTU, the relay sends the outgoing packet with a Fragment header. The relay sets the fields of the Fragment header as appropriate for a packet originating from the server.

If the incoming packet included a Fragment header and the outgoing packet size exceeds the outgoing link's MTU, the relay MUST fragment the outgoing packet into fragments of no more than 1280 bytes. The relay sets the fields of the Fragment header as appropriate for a packet originating from the server.

Alternate behavior: The relay assembles incoming fragments. The relay follows its default behavior to send outgoing packets.

For both preferred and alternate behavior, the DONT-FRAGMENT attribute MUST be ignored by the server.

Extension Headers

Preferred behavior: The relay sends outgoing packet without any IPv6 extension headers, with the exception of the Fragmentation header as described above.

Alternate behavior: Same as preferred.

12.3. IPv6-to-IPv4 Translations

Type of Service and Precedence

Preferred behavior: As specified in Section 5 of [RFC6145].

Alternate behavior: The relay sets the Type of Service and Precedence to the default value for outgoing packets.

Time to Live

Preferred behavior: As specified in Section 5 of [RFC6145].

Alternate behavior: The relay sets the Time to Live to the default value for outgoing packets.

Fragmentation

Preferred behavior: As specified in Section 5 of [RFC6145]. Additionally, when the outgoing packet's size exceeds the outgoing link's MTU, the relay needs to generate an ICMP error (ICMPv6 Packet Too Big) reporting the MTU size. If the packet is being sent to the peer, the relay SHOULD reduce the MTU reported in the ICMP message by 48 bytes to allow room for the overhead of a Data indication.

Alternate behavior: The relay assembles incoming fragments. The relay follows its default behavior to send outgoing packets.

For both preferred and alternate behavior, the DONT-FRAGMENT attribute MUST be ignored by the server.

13. IP Header Fields

This section describes how the server sets various fields in the IP header when relaying between the client and the peer or vice versa. The descriptions in this section apply: (a) when the server sends a UDP datagram to the peer, or (b) when the server sends a Data indication or ChannelData message to the client over UDP transport. The descriptions in this section do not apply to TURN messages sent over TCP or TLS transport from the server to the client.

The descriptions below have two parts: a preferred behavior and an alternate behavior. The server SHOULD implement the preferred behavior, but if that is not possible for a particular field, then it SHOULD implement the alternative behavior.

Time to Live (TTL) field

Preferred Behavior: If the incoming value is 0, then the drop the incoming packet. Otherwise, set the outgoing Time to Live/Hop Count to one less than the incoming value.

Alternate Behavior: Set the outgoing value to the default for outgoing packets.

Differentiated Services Code Point (DSCP) field [RFC2474]

Preferred Behavior: Set the outgoing value to the incoming value, unless the server includes a differentiated services classifier and marker [RFC2474].

Alternate Behavior: Set the outgoing value to a fixed value, which by default is Best Effort unless configured otherwise.

In both cases, if the server is immediately adjacent to a differentiated services classifier and marker, then DSCP MAY be set to any arbitrary value in the direction towards the classifier.

Explicit Congestion Notification (ECN) field [RFC3168]

Preferred Behavior: Set the outgoing value to the incoming value, UNLESS the server is doing Active Queue Management, the incoming ECN field is ECT(1) (=0b01) or ECT(0) (=0b10), and the server wishes to indicate that congestion has been experienced, in which case set the outgoing value to CE (=0b11).

Alternate Behavior: Set the outgoing value to Not-ECT (=0b00).

IPv4 Fragmentation fields

Preferred Behavior: When the server sends a packet to a peer in response to a Send indication containing the DONT-FRAGMENT attribute, then set the DF bit in the outgoing IP header to 1. In all other cases when sending an outgoing packet containing application data (e.g., Data indication, ChannelData message, or DONT-FRAGMENT attribute not included in the Send indication), copy the DF bit from the DF bit of the incoming packet that contained the application data.

Set the other fragmentation fields (Identification, More Fragments, Fragment Offset) as appropriate for a packet originating from the server.

Alternate Behavior: As described in the Preferred Behavior, except always assume the incoming DF bit is 0.

In both the Preferred and Alternate Behaviors, the resulting packet may be too large for the outgoing link. If this is the case, then the normal fragmentation rules apply [RFC1122].

IPv4 Options

Preferred Behavior: The outgoing packet is sent without any IPv4 options.

Alternate Behavior: Same as preferred.

14. New STUN Methods

This section lists the codepoints for the new STUN methods defined in this specification. See elsewhere in this document for the semantics of these new methods.

```
0x003  :  Allocate          (only request/response semantics defined)
0x004  :  Refresh           (only request/response semantics defined)
0x006  :  Send              (only indication semantics defined)
0x007  :  Data              (only indication semantics defined)
0x008  :  CreatePermission  (only request/response semantics defined
0x009  :  ChannelBind       (only request/response semantics defined)
```

15. New STUN Attributes

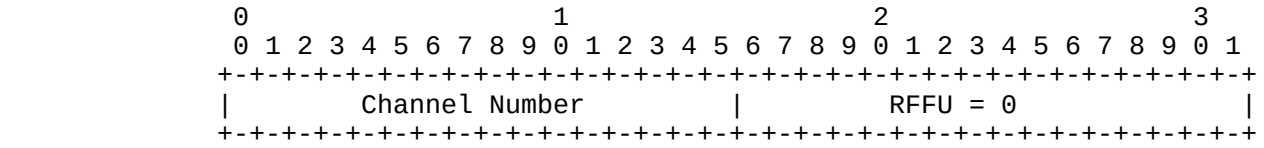
This STUN extension defines the following new attributes:

```
0x000C: CHANNEL-NUMBER
0x000D: LIFETIME
0x0010: Reserved (was BANDWIDTH)
0x0012: XOR-PEER-ADDRESS
0x0013: DATA
0x0016: XOR-RELAYED-ADDRESS
0x0017: REQUESTED-ADDRESS-FAMILY
0x0018: EVEN-PORT
0x0019: REQUESTED-TRANSPORT
0x001A: DONT-FRAGMENT
0x0021: Reserved (was TIMER-VAL)
0x0022: RESERVATION-TOKEN
TBD-CA: ADDITIONAL-ADDRESS-FAMILY
TBD-CA: ADDRESS-ERROR-CODE
```

Some of these attributes have lengths that are not multiples of 4. By the rules of STUN, any attribute whose length is not a multiple of 4 bytes MUST be immediately followed by 1 to 3 padding bytes to ensure the next attribute (if any) would start on a 4-byte boundary (see [RFC5389]).

15.1. CHANNEL-NUMBER

The CHANNEL-NUMBER attribute contains the number of the channel. The value portion of this attribute is 4 bytes long and consists of a 16-bit unsigned integer, followed by a two-octet RFFU (Reserved For Future Use) field, which MUST be set to 0 on transmission and MUST be ignored on reception.



15.2. LIFETIME

The LIFETIME attribute represents the duration for which the server will maintain an allocation in the absence of a refresh. The value portion of this attribute is 4-bytes long and consists of a 32-bit unsigned integral value representing the number of seconds remaining until expiration.

15.3. XOR-PEER-ADDRESS

The XOR-PEER-ADDRESS specifies the address and port of the peer as seen from the TURN server. (For example, the peer's server-reflexive transport address if the peer is behind a NAT.) It is encoded in the same way as XOR-MAPPED-ADDRESS [RFC5389].

15.4. DATA

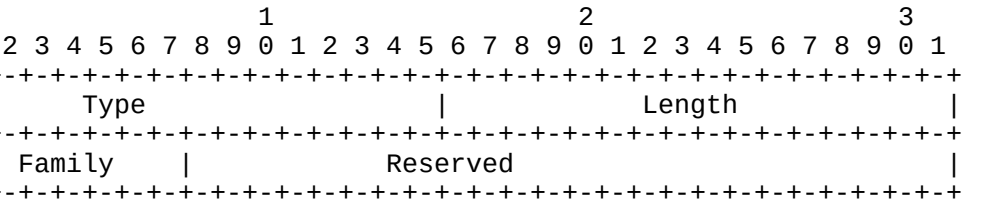
The DATA attribute is present in all Send and Data indications. The value portion of this attribute is variable length and consists of the application data (that is, the data that would immediately follow the UDP header if the data was been sent directly between the client and the peer). If the length of this attribute is not a multiple of 4, then padding must be added after this attribute.

15.5. XOR-RELAYED-ADDRESS

The XOR-RELAYED-ADDRESS is present in Allocate responses. It specifies the address and port that the server allocated to the client. It is encoded in the same way as XOR-MAPPED-ADDRESS [RFC5389].

15.6. REQUESTED-ADDRESS-FAMILY

This attribute is used by clients to request the allocation of a specific address type from a server. The following is the format of the REQUESTED-ADDRESS-FAMILY attribute. Note that TURN attributes are TLV (Type-Length-Value) encoded, with a 16-bit type, a 16-bit length, and a variable-length value.



Type: the type of the REQUESTED-ADDRESS-FAMILY attribute is 0x0017. As specified in [RFC5389], attributes with values between 0x0000 and 0x7FFF are comprehension-required, which means that the client or server cannot successfully process the message unless it understands the attribute.

Length: this 16-bit field contains the length of the attribute in bytes. The length of this attribute is 4 bytes.

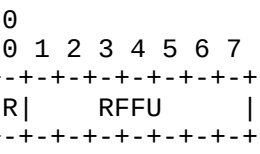
Family: there are two values defined for this field and specified in [RFC5389], Section 15.1: 0x01 for IPv4 addresses and 0x02 for IPv6 addresses.

Reserved: at this point, the 24 bits in the Reserved field MUST be set to zero by the client and MUST be ignored by the server.

The REQUEST-ADDRESS-TYPE attribute MAY only be present in Allocate requests.

15.7. EVEN-PORT

This attribute allows the client to request that the port in the relayed transport address be even, and (optionally) that the server reserve the next-higher port number. The value portion of this attribute is 1 byte long. Its format is:



The value contains a single 1-bit flag:

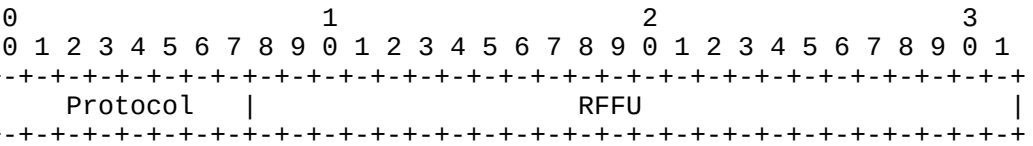
R: If 1, the server is requested to reserve the next-higher port number (on the same IP address) for a subsequent allocation. If 0, no such reservation is requested.

The other 7 bits of the attribute's value must be set to zero on transmission and ignored on reception.

Since the length of this attribute is not a multiple of 4, padding must immediately follow this attribute.

15.8. REQUESTED-TRANSPORT

This attribute is used by the client to request a specific transport protocol for the allocated transport address. The value of this attribute is 4 bytes with the following format:



The Protocol field specifies the desired protocol. The codepoints used in this field are taken from those allowed in the Protocol field in the IPv4 header and the NextHeader field in the IPv6 header [Protocol-Numbers]. This specification only allows the use of codepoint 17 (User Datagram Protocol).

The RFFU field MUST be set to zero on transmission and MUST be ignored on reception. It is reserved for future uses.

15.9. DONT-FRAGMENT

This attribute is used by the client to request that the server set the DF (Don't Fragment) bit in the IP header when relaying the application data onward to the peer. This attribute has no value part and thus the attribute length field is 0.

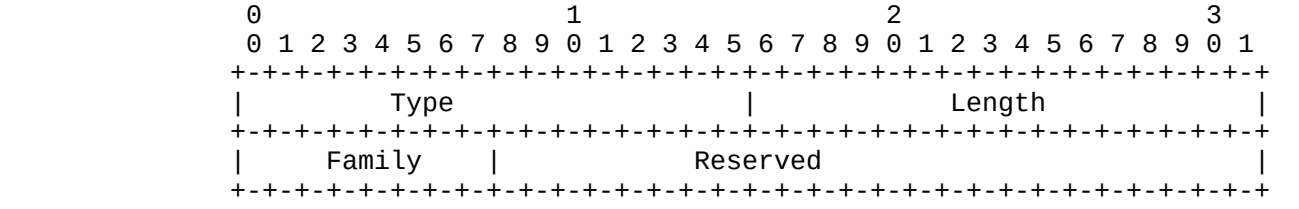
15.10. RESERVATION-TOKEN

The RESERVATION-TOKEN attribute contains a token that uniquely identifies a relayed transport address being held in reserve by the server. The server includes this attribute in a success response to tell the client about the token, and the client includes this attribute in a subsequent Allocate request to request the server use that relayed transport address for the allocation.

The attribute value is 8 bytes and contains the token value.

15.11. ADDITIONAL-ADDRESS-FAMILY

This attribute is used by clients to request the allocation of a IPv4 and IPv6 address type from a server. The following is the format of the ADDITIONAL-ADDRESS-FAMILY attribute.



Type: the type of the ADDITIONAL-ADDRESS-FAMILY attribute is TBD-CA. As specified in [RFC5389], attributes with values between 0x8000 and 0xFFFF are comprehension-optional, which means that the client or server can safely ignore the attribute if they don’t understand it.

Length: this 16-bit field contains the length of the attribute in bytes. The length of this attribute is 4 bytes.

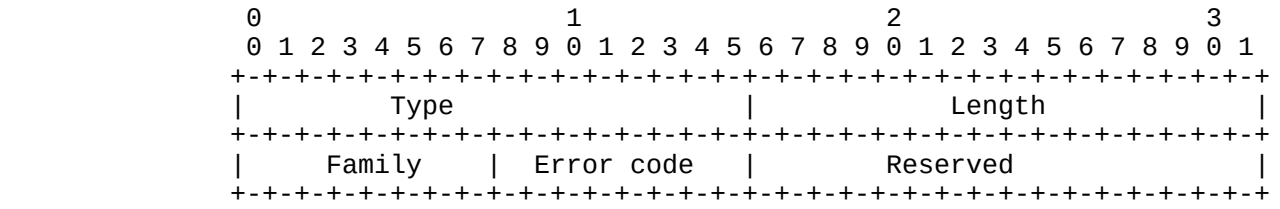
Family: there are two values defined for this field and specified in [RFC5389], Section 15.1: 0x01 for IPv4 addresses and 0x02 for IPv6 addresses.

Reserved: at this point, the 24 bits in the Reserved field MUST be set to zero by the client and MUST be ignored by the server.

The ADDITIONAL-ADDRESS-FAMILY attribute MAY be present in Allocate or Refresh requests. The attribute value of 0x02 (IPv6 address) is the only valid value in Allocate request.

15.12. ADDRESS-ERROR-CODE Attribute

This attribute is used by servers to signal the reason for not allocating the requested address family. The following is the format of the ADDRESS-ERROR-CODE attribute.



Type: the type of the ADDRESS-ERROR-CODE attribute is TBD-CA. As specified in [RFC5389], attributes with values between 0x8000 and 0xFFFF are comprehension-optional, which means that the client or server can safely ignore the attribute if they don't understand it.

Length: this 16-bit field contains the length of the attribute in bytes. The length of this attribute is 4 bytes.

Family: there are two values defined for this field and specified in [RFC5389], Section 15.1: 0x01 for IPv4 addresses and 0x02 for IPv6 addresses.

Error code: this 8-bit field contains the reason server cannot allocate one of the requested address types. The error code values could be either 440 (unsupported address family) or 508 (insufficient capacity).

Reserved: at this point, the 16 bits in the Reserved field MUST be set to zero by the client and MUST be ignored by the server.

The ADDRESS-ERROR-CODE attribute MAY only be present in Allocate responses.

16. New STUN Error Response Codes

This document defines the following new error response codes:

403 (Forbidden): The request was valid but cannot be performed due to administrative or similar restrictions.

437 (Allocation Mismatch): A request was received by the server that requires an allocation to be in place, but no allocation exists,

or a request was received that requires no allocation, but an allocation exists.

440 (Address Family not Supported): The server does not support the address family requested by the client.

441 (Wrong Credentials): The credentials in the (non-Allocate) request do not match those used to create the allocation.

442 (Unsupported Transport Protocol): The Allocate request asked the server to use a transport protocol between the server and the peer that the server does not support. NOTE: This does NOT refer to the transport protocol used in the 5-tuple.

443 (Peer Address Family Mismatch). A peer address is part of a different address family than that of the relayed transport address of the allocation.

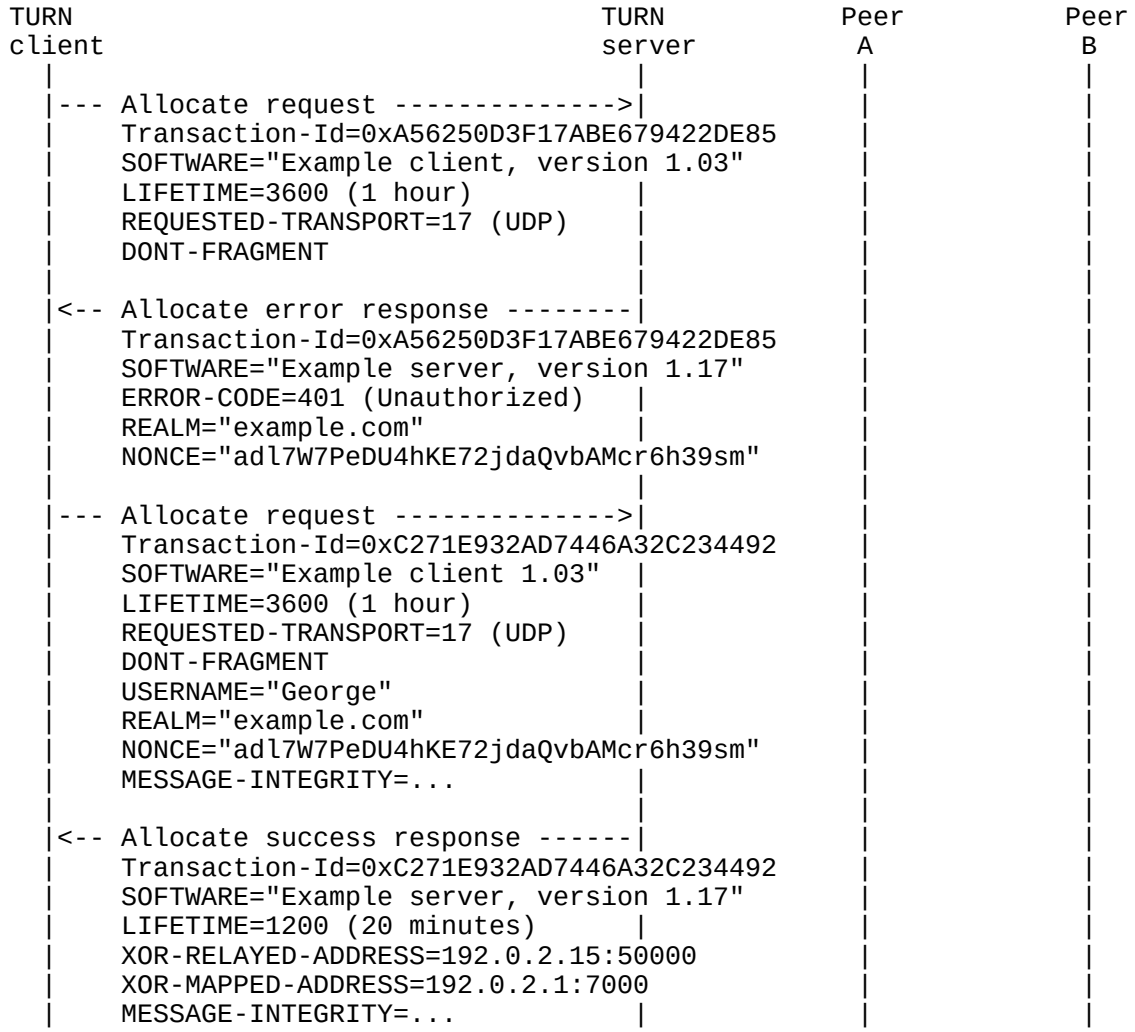
486 (Allocation Quota Reached): No more allocations using this username can be created at the present time.

508 (Insufficient Capacity): The server is unable to carry out the request due to some capacity limit being reached. In an Allocate response, this could be due to the server having no more relayed transport addresses available at that time, having none with the requested properties, or the one that corresponds to the specified reservation token is not available.

17. Detailed Example

This section gives an example of the use of TURN, showing in detail the contents of the messages exchanged. The example uses the network diagram shown in the Overview (Figure 1).

For each message, the attributes included in the message and their values are shown. For convenience, values are shown in a human-readable format rather than showing the actual octets; for example, "XOR-RELAYED-ADDRESS=192.0.2.15:9000" shows that the XOR-RELAYED-ADDRESS attribute is included with an address of 192.0.2.15 and a port of 9000, here the address and port are shown before the xor-ing is done. For attributes with string-like values (e.g., SOFTWARE="Example client, version 1.03" and NONCE="adl7W7PeDU4hKE72jdaQvbAMcr6h39sm"), the value of the attribute is shown in quotes for readability, but these quotes do not appear in the actual value.



The client begins by selecting a host transport address to use for the TURN session; in this example, the client has selected 10.1.1.2:49721 as shown in Figure 1. The client then sends an Allocate request to the server at the server transport address. The client randomly selects a 96-bit transaction id of 0xA56250D3F17ABE679422DE85 for this transaction; this is encoded in the transaction id field in the fixed header. The client includes a SOFTWARE attribute that gives information about the client's software; here the value is "Example client, version 1.03" to indicate that this is version 1.03 of something called the Example client. The client includes the LIFETIME attribute because it wishes the allocation to have a longer lifetime than the default of 10

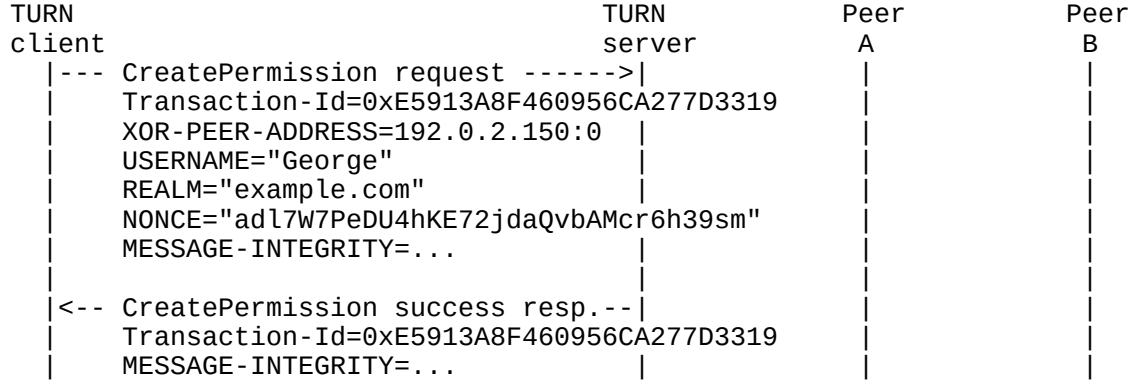
minutes; the value of this attribute is 3600 seconds, which corresponds to 1 hour. The client must always include a REQUESTED-TRANSPORT attribute in an Allocate request and the only value allowed by this specification is 17, which indicates UDP transport between the server and the peers. The client also includes the DONT-FRAGMENT attribute because it wishes to use the DONT-FRAGMENT attribute later in Send indications; this attribute consists of only an attribute header, there is no value part. We assume the client has not recently interacted with the server, thus the client does not include USERNAME, REALM, NONCE, or MESSAGE-INTEGRITY attribute. Finally, note that the order of attributes in a message is arbitrary (except for the MESSAGE-INTEGRITY and FINGERPRINT attributes) and the client could have used a different order.

Servers require any request to be authenticated. Thus, when the server receives the initial Allocate request, it rejects the request because the request does not contain the authentication attributes. Following the procedures of the long-term credential mechanism of STUN [RFC5389], the server includes an ERROR-CODE attribute with a value of 401 (Unauthorized), a REALM attribute that specifies the authentication realm used by the server (in this case, the server's domain "example.com"), and a nonce value in a NONCE attribute. The server also includes a SOFTWARE attribute that gives information about the server's software.

The client, upon receipt of the 401 error, re-attempts the Allocate request, this time including the authentication attributes. The client selects a new transaction id, and then populates the new Allocate request with the same attributes as before. The client includes a USERNAME attribute and uses the realm value received from the server to help it determine which value to use; here the client is configured to use the username "George" for the realm "example.com". The client also includes the REALM and NONCE attributes, which are just copied from the 401 error response. Finally, the client includes a MESSAGE-INTEGRITY attribute as the last attribute in the message, whose value is a Hashed Message Authentication Code - Secure Hash Algorithm 1 (HMAC-SHA1) hash over the contents of the message (shown as just "..." above); this HMAC-SHA1 computation includes a password value. Thus, an attacker cannot compute the message integrity value without somehow knowing the secret password.

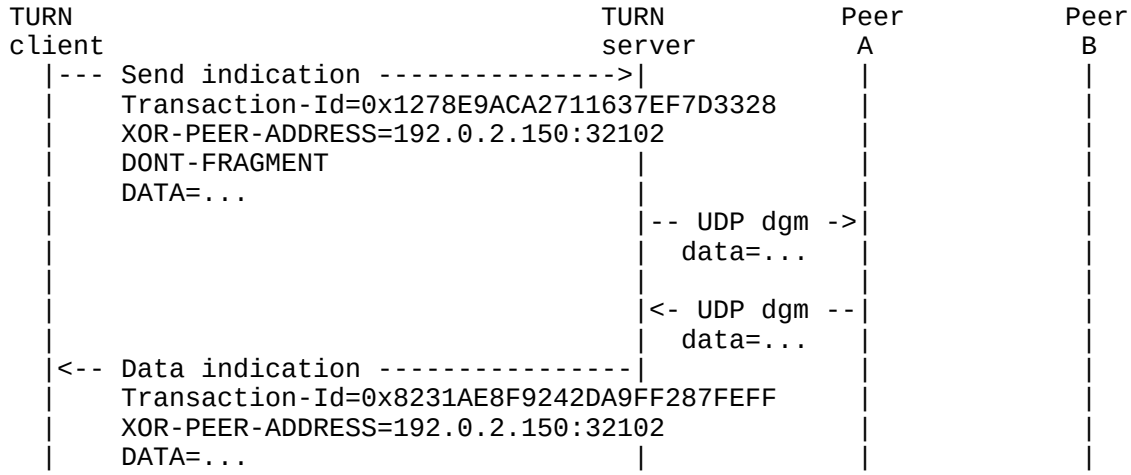
The server, upon receipt of the authenticated Allocate request, checks that everything is OK, then creates an allocation. The server replies with an Allocate success response. The server includes a LIFETIME attribute giving the lifetime of the allocation; here, the server has reduced the client's requested 1-hour lifetime to just 20 minutes, because this particular server doesn't allow lifetimes

longer than 20 minutes. The server includes an XOR-RELAYED-ADDRESS attribute whose value is the relayed transport address of the allocation. The server includes an XOR-MAPPED-ADDRESS attribute whose value is the server-reflexive address of the client; this value is not used otherwise in TURN but is returned as a convenience to the client. The server includes a MESSAGE-INTEGRITY attribute to authenticate the response and to ensure its integrity; note that the response does not contain the USERNAME, REALM, and NONCE attributes. The server also includes a SOFTWARE attribute.



The client then creates a permission towards Peer A in preparation for sending it some application data. This is done through a CreatePermission request. The XOR-PEER-ADDRESS attribute contains the IP address for which a permission is established (the IP address of peer A); note that the port number in the attribute is ignored when used in a CreatePermission request, and here it has been set to 0; also, note how the client uses Peer A's server-reflexive IP address and not its (private) host address. The client uses the same username, realm, and nonce values as in the previous request on the allocation. Though it is allowed to do so, the client has chosen not to include a SOFTWARE attribute in this request.

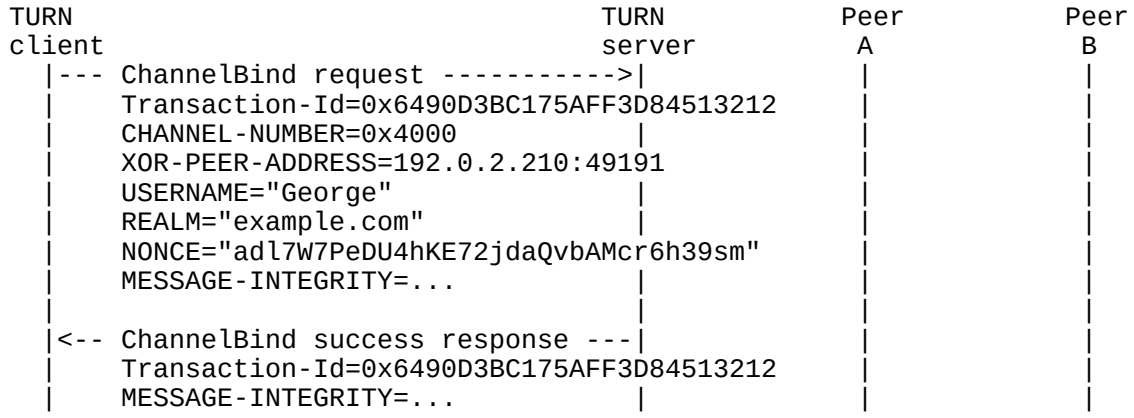
The server receives the CreatePermission request, creates the corresponding permission, and then replies with a CreatePermission success response. Like the client, the server chooses not to include the SOFTWARE attribute in its reply. Again, note how success responses contain a MESSAGE-INTEGRITY attribute (assuming the server uses the long-term credential mechanism), but no USERNAME, REALM, and NONCE attributes.



The client now sends application data to Peer A using a Send indication. Peer A's server-reflexive transport address is specified in the XOR-PEER-ADDRESS attribute, and the application data (shown here as just "...") is specified in the DATA attribute. The client is doing a form of path MTU discovery at the application layer and thus specifies (by including the DONT-FRAGMENT attribute) that the server should set the DF bit in the UDP datagram to send to the peer. Indications cannot be authenticated using the long-term credential mechanism of STUN, so no MESSAGE-INTEGRITY attribute is included in the message. An application wishing to ensure that its data is not altered or forged must integrity-protect its data at the application level.

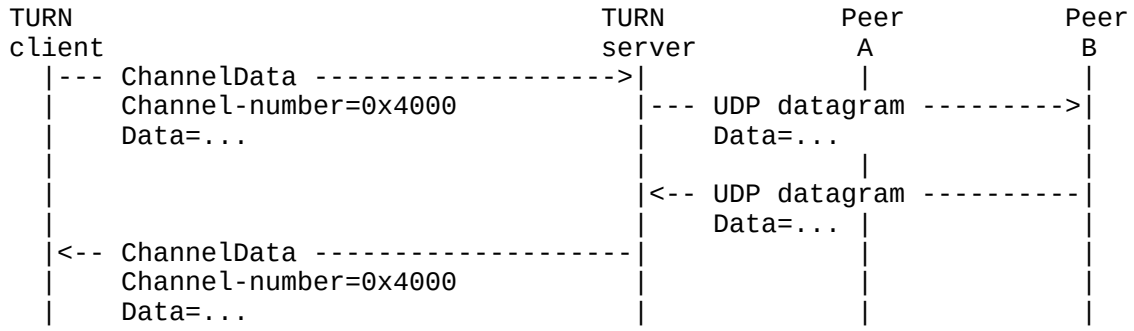
Upon receipt of the Send indication, the server extracts the application data and sends it in a UDP datagram to Peer A, with the relayed transport address as the source transport address of the datagram, and with the DF bit set as requested. Note that, had the client not previously established a permission for Peer A's server-reflexive IP address, then the server would have silently discarded the Send indication instead.

Peer A then replies with its own UDP datagram containing application data. The datagram is sent to the relayed transport address on the server. When this arrives, the server creates a Data indication containing the source of the UDP datagram in the XOR-PEER-ADDRESS attribute, and the data from the UDP datagram in the DATA attribute. The resulting Data indication is then sent to the client.



The client now binds a channel to Peer B, specifying a free channel number (0x4000) in the CHANNEL-NUMBER attribute, and Peer B's transport address in the XOR-PEER-ADDRESS attribute. As before, the client re-uses the username, realm, and nonce from its last request in the message.

Upon receipt of the request, the server binds the channel number to the peer, installs a permission for Peer B's IP address, and then replies with ChannelBind success response.



The client now sends a ChannelData message to the server with data destined for Peer B. The ChannelData message is not a STUN message, and thus has no transaction id. Instead, it has only three fields: a channel number, data, and data length; here the channel number field is 0x4000 (the channel the client just bound to Peer B). When the server receives the ChannelData message, it checks that the channel is currently bound (which it is) and then sends the data onward to Peer B in a UDP datagram, using the relayed transport address as the source transport address and 192.0.2.210:49191 (the value of the XOR-PEER-ADDRESS attribute in the ChannelBind request) as the destination transport address.

Later, Peer B sends a UDP datagram back to the relayed transport address. This causes the server to send a ChannelData message to the client containing the data from the UDP datagram. The server knows to which client to send the ChannelData message because of the relayed transport address at which the UDP datagram arrived, and knows to use channel 0x4000 because this is the channel bound to 192.0.2.210:49191. Note that if there had not been any channel number bound to that address, the server would have used a Data indication instead.

Sometime before the 20 minute lifetime is up, the client refreshes the allocation. This is done using a Refresh request. As before, the client includes the latest username, realm, and nonce values in the request. The client also includes the SOFTWARE attribute, following the recommended practice of always including this attribute in Allocate and Refresh messages. When the server receives the Refresh request, it notices that the nonce value has expired, and so replies with 438 (Stale Nonce) error given a new nonce value. The

client then reattempts the request, this time with the new nonce value. This second attempt is accepted, and the server replies with a success response. Note that the client did not include a LIFETIME attribute in the request, so the server refreshes the allocation for the default lifetime of 10 minutes (as can be seen by the LIFETIME attribute in the success response).

18. Security Considerations

This section considers attacks that are possible in a TURN deployment, and discusses how they are mitigated by mechanisms in the protocol or recommended practices in the implementation.

Most of the attacks on TURN are mitigated by the server requiring requests be authenticated. Thus, this specification requires the use of authentication. The mandatory-to-implement mechanism is the long-term credential mechanism of STUN. Other authentication mechanisms of equal or stronger security properties may be used. However, it is important to ensure that they can be invoked in an inter-operable way.

18.1. Outsider Attacks

Outsider attacks are ones where the attacker has no credentials in the system, and is attempting to disrupt the service seen by the client or the server.

18.1.1. Obtaining Unauthorized Allocations

An attacker might wish to obtain allocations on a TURN server for any number of nefarious purposes. A TURN server provides a mechanism for sending and receiving packets while cloaking the actual IP address of the client. This makes TURN servers an attractive target for attackers who wish to use it to mask their true identity.

An attacker might also wish to simply utilize the services of a TURN server without paying for them. Since TURN services require resources from the provider, it is anticipated that their usage will come with a cost.

These attacks are prevented using the long-term credential mechanism, which allows the TURN server to determine the identity of the requestor and whether the requestor is allowed to obtain the allocation.

18.1.2. Offline Dictionary Attacks

The long-term credential mechanism used by TURN is subject to offline dictionary attacks. An attacker that is capable of eavesdropping on a message exchange between a client and server can determine the password by trying a number of candidate passwords and seeing if one of them is correct. This attack works when the passwords are low entropy, such as a word from the dictionary. This attack can be mitigated by using strong passwords with large entropy. In situations where even stronger mitigation is required, (D)TLS transport between the client and the server can be used.

18.1.3. Faked Refreshes and Permissions

An attacker might wish to attack an active allocation by sending it a Refresh request with an immediate expiration, in order to delete it and disrupt service to the client. This is prevented by authentication of refreshes. Similarly, an attacker wishing to send CreatePermission requests to create permissions to undesirable destinations is prevented from doing so through authentication. The motivations for such an attack are described in Section 18.2.

18.1.4. Fake Data

An attacker might wish to send data to the client or the peer, as if they came from the peer or client, respectively. To do that, the attacker can send the client a faked Data Indication or ChannelData message, or send the TURN server a faked Send Indication or ChannelData message.

Since indications and ChannelData messages are not authenticated, this attack is not prevented by TURN. However, this attack is generally present in IP-based communications and is not substantially worsened by TURN. Consider a normal, non-TURN IP session between hosts A and B. An attacker can send packets to B as if they came from A by sending packets towards A with a spoofed IP address of B. This attack requires the attacker to know the IP addresses of A and B. With TURN, an attacker wishing to send packets towards a client using a Data indication needs to know its IP address (and port), the IP address and port of the TURN server, and the IP address and port of the peer (for inclusion in the XOR-PEER-ADDRESS attribute). To send a fake ChannelData message to a client, an attacker needs to know the IP address and port of the client, the IP address and port of the TURN server, and the channel number. This particular combination is mildly more guessable than in the non-TURN case.

These attacks are more properly mitigated by application-layer authentication techniques. In the case of real-time traffic, usage of SRTP [RFC3711] prevents these attacks.

In some situations, the TURN server may be situated in the network such that it is able to send to hosts to which the client cannot directly send. This can happen, for example, if the server is located behind a firewall that allows packets from outside the firewall to be delivered to the server, but not to other hosts behind the firewall. In these situations, an attacker could send the server a Send indication with an XOR-PEER-ADDRESS attribute containing the transport address of one of the other hosts behind the firewall. If the server was to allow relaying of traffic to arbitrary peers, then this would provide a way for the attacker to attack arbitrary hosts behind the firewall.

To mitigate this attack, TURN requires that the client establish a permission to a host before sending it data. Thus, an attacker can only attack hosts with which the client is already communicating, unless the attacker is able to create authenticated requests. Furthermore, the server administrator may configure the server to restrict the range of IP addresses and ports to which it will relay data. To provide even greater security, the server administrator can require that the client use (D)TLS for all communication between the client and the server.

18.1.5. Impersonating a Server

When a client learns a relayed address from a TURN server, it uses that relayed address in application protocols to receive traffic. Therefore, an attacker wishing to intercept or redirect that traffic might try to impersonate a TURN server and provide the client with a faked relayed address.

This attack is prevented through the long-term credential mechanism, which provides message integrity for responses in addition to verifying that they came from the server. Furthermore, an attacker cannot replay old server responses as the transaction id in the STUN header prevents this. Replay attacks are further thwarted through frequent changes to the nonce value.

18.1.6. Eavesdropping Traffic

TURN concerns itself primarily with authentication and message integrity. Confidentiality is only a secondary concern, as TURN control messages do not include information that is particularly sensitive. The primary protocol content of the messages is the IP

address of the peer. If it is important to prevent an eavesdropper on a TURN connection from learning this, TURN can be run over (D)TLS.

Confidentiality for the application data relayed by TURN is best provided by the application protocol itself, since running TURN over (D)TLS does not protect application data between the server and the peer. If confidentiality of application data is important, then the application should encrypt or otherwise protect its data. For example, for real-time media, confidentiality can be provided by using SRTP.

18.1.7. TURN Loop Attack

An attacker might attempt to cause data packets to loop indefinitely between two TURN servers. The attack goes as follows. First, the attacker sends an Allocate request to server A, using the source address of server B. Server A will send its response to server B, and for the attack to succeed, the attacker must have the ability to either view or guess the contents of this response, so that the attacker can learn the allocated relayed transport address. The attacker then sends an Allocate request to server B, using the source address of server A. Again, the attacker must be able to view or guess the contents of the response, so it can send learn the allocated relayed transport address. Using the same spoofed source address technique, the attacker then binds a channel number on server A to the relayed transport address on server B, and similarly binds the same channel number on server B to the relayed transport address on server A. Finally, the attacker sends a ChannelData message to server A.

The result is a data packet that loops from the relayed transport address on server A to the relayed transport address on server B, then from server B's transport address to server A's transport address, and then around the loop again.

This attack is mitigated as follows. By requiring all requests to be authenticated and/or by randomizing the port number allocated for the relayed transport address, the server forces the attacker to either intercept or view responses sent to a third party (in this case, the other server) so that the attacker can authenticate the requests and learn the relayed transport address. Without one of these two measures, an attacker can guess the contents of the responses without needing to see them, which makes the attack much easier to perform. Furthermore, by requiring authenticated requests, the server forces the attacker to have credentials acceptable to the server, which turns this from an outsider attack into an insider attack and allows the attack to be traced back to the client initiating it.

The attack can be further mitigated by imposing a per-username limit on the bandwidth used to relay data by allocations owned by that username, to limit the impact of this attack on other allocations. More mitigation can be achieved by decrementing the TTL when relaying data packets (if the underlying OS allows this).

18.2. Firewall Considerations

A key security consideration of TURN is that TURN should not weaken the protections afforded by firewalls deployed between a client and a TURN server. It is anticipated that TURN servers will often be present on the public Internet, and clients may often be inside enterprise networks with corporate firewalls. If TURN servers provide a 'backdoor' for reaching into the enterprise, TURN will be blocked by these firewalls.

TURN servers therefore emulate the behavior of NAT devices that implement address-dependent filtering [RFC4787], a property common in many firewalls as well. When a NAT or firewall implements this behavior, packets from an outside IP address are only allowed to be sent to an internal IP address and port if the internal IP address and port had recently sent a packet to that outside IP address. TURN servers introduce the concept of permissions, which provide exactly this same behavior on the TURN server. An attacker cannot send a packet to a TURN server and expect it to be relayed towards the client, unless the client has tried to contact the attacker first.

It is important to note that some firewalls have policies that are even more restrictive than address-dependent filtering. Firewalls can also be configured with address- and port-dependent filtering, or can be configured to disallow inbound traffic entirely. In these cases, if a client is allowed to connect the TURN server, communications to the client will be less restrictive than what the firewall would normally allow.

18.2.1. Faked Permissions

In firewalls and NAT devices, permissions are granted implicitly through the traversal of a packet from the inside of the network towards the outside peer. Thus, a permission cannot, by definition, be created by any entity except one inside the firewall or NAT. With TURN, this restriction no longer holds. Since the TURN server sits outside the firewall, at attacker outside the firewall can now send a message to the TURN server and try to create a permission for itself.

This attack is prevented because all messages that create permissions (i.e., ChannelBind and CreatePermission) are authenticated.

18.2.2. Blacklisted IP Addresses

Many firewalls can be configured with blacklists that prevent a client behind the firewall from sending packets to, or receiving packets from, ranges of blacklisted IP addresses. This is accomplished by inspecting the source and destination addresses of packets entering and exiting the firewall, respectively.

This feature is also present in TURN, since TURN servers are allowed to arbitrarily restrict the range of addresses of peers that they will relay to.

18.2.3. Running Servers on Well-Known Ports

A malicious client behind a firewall might try to connect to a TURN server and obtain an allocation which it then uses to run a server. For example, a client might try to run a DNS server or FTP server.

This is not possible in TURN. A TURN server will never accept traffic from a peer for which the client has not installed a permission. Thus, peers cannot just connect to the allocated port in order to obtain the service.

18.3. Insider Attacks

In insider attacks, a client has legitimate credentials but defies the trust relationship that goes with those credentials. These attacks cannot be prevented by cryptographic means but need to be considered in the design of the protocol.

18.3.1. DoS against TURN Server

A client wishing to disrupt service to other clients might obtain an allocation and then flood it with traffic, in an attempt to swamp the server and prevent it from servicing other legitimate clients. This is mitigated by the recommendation that the server limit the amount of bandwidth it will relay for a given username. This won’t prevent a client from sending a large amount of traffic, but it allows the server to immediately discard traffic in excess.

Since each allocation uses a port number on the IP address of the TURN server, the number of allocations on a server is finite. An attacker might attempt to consume all of them by requesting a large number of allocations. This is prevented by the recommendation that the server impose a limit of the number of allocations active at a time for a given username.

18.3.2. Anonymous Relaying of Malicious Traffic

TURN servers provide a degree of anonymization. A client can send data to peers without revealing its own IP address. TURN servers may therefore become attractive vehicles for attackers to launch attacks against targets without fear of detection. Indeed, it is possible for a client to chain together multiple TURN servers, such that any number of relays can be used before a target receives a packet.

Administrators who are worried about this attack can maintain logs that capture the actual source IP and port of the client, and perhaps even every permission that client installs. This will allow for forensic tracing to determine the original source, should it be discovered that an attack is being relayed through a TURN server.

18.3.3. Manipulating Other Allocations

An attacker might attempt to disrupt service to other users of the TURN server by sending Refresh requests or CreatePermission requests that (through source address spoofing) appear to be coming from another user of the TURN server. TURN prevents this by requiring that the credentials used in CreatePermission, Refresh, and ChannelBind messages match those used to create the initial allocation. Thus, the fake requests from the attacker will be rejected.

18.4. Tunnel Amplification Attack

An attacker might attempt to cause data packets to loop numerous times between a TURN server and a tunnel between IPv4 and IPv6. The attack goes as follows.

Suppose an attacker knows that a tunnel endpoint will forward encapsulated packets from a given IPv6 address (this doesn’t necessarily need to be the tunnel endpoint’s address). Suppose he then spoofs two packets from this address:

1. An Allocate request asking for a v4 address, and

2. A ChannelBind request establishing a channel to the IPv4 address of the tunnel endpoint

Then he has set up an amplification attack:

- The TURN relay will re-encapsulate IPv6 UDP data in v4 and send it to the tunnel endpoint

o The tunnel endpoint will de-encapsulate packets from the v4 interface and send them to v6

So if the attacker sends a packet of the following form...

```
IPv6: src=2001:DB9::1 dst=2001:DB8::2
UDP:  <ports>
TURN: <channel id>
IPv6: src=2001:DB9::1 dst=2001:DB8::2
UDP:  <ports>
TURN: <channel id>
IPv6: src=2001:DB9::1 dst=2001:DB8::2
UDP:  <ports>
TURN: <channel id>
...
```

Then the TURN relay and the tunnel endpoint will send it back and forth until the last TURN header is consumed, at which point the TURN relay will send an empty packet, which the tunnel endpoint will drop.

The amplification potential here is limited by the MTU, so it’s not huge: IPv6+UDP+TURN takes 334 bytes, so a four-to-one amplification out of a 1500-byte packet is possible. But the attacker could still increase traffic volume by sending multiple packets or by establishing multiple channels spoofed from different addresses behind the same tunnel endpoint.

The attack is mitigated as follows. It is RECOMMENDED that TURN relays not accept allocation or channel binding requests from addresses known to be tunneled, and that they not forward data to such addresses. In particular, a TURN relay MUST NOT accept Teredo or 6to4 addresses in these requests.

18.5. Other Considerations

Any relay addresses learned through an Allocate request will not operate properly with IPsec Authentication Header (AH) [RFC4302] in transport or tunnel mode. However, tunnel-mode IPsec Encapsulating Security Payload (ESP) [RFC4303] should still operate.

19. IANA Considerations

Since TURN is an extension to STUN [RFC5389], the methods, attributes, and error codes defined in this specification are new methods, attributes, and error codes for STUN. IANA has added these new protocol elements to the IANA registry of STUN protocol elements.

The codepoints for the new STUN methods defined in this specification are listed in Section 14.

The codepoints for the new STUN attributes defined in this specification are listed in Section 15.

The codepoints for the new STUN error codes defined in this specification are listed in Section 16.

IANA has allocated the SRV service name of "turn" for TURN over UDP or TCP, and the service name of "turns" for TURN over (D)TLS.

IANA has created a registry for TURN channel numbers, initially populated as follows:

- 0x0000 through 0x3FFF: Reserved and not available for use, since they conflict with the STUN header.
- 0x4000 through 0x7FFF: A TURN implementation is free to use channel numbers in this range.
- 0x8000 through 0xFFFF: Unassigned.

Any change to this registry must be made through an IETF Standards Action.

[Paragraphs in braces should be removed by the RFC Editor upon publication]

[The ADDITIONAL-ADDRESS-FAMILY, ADDRESS-ERROR-CODE attributes requires that IANA allocate a value in the "STUN attributes Registry" from the comprehension- optional range (0x8000-0xFFFF), to be replaced for TBD-CA throughout this document]

20. 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]. The TURN extension is an example of a protocol that performs this type of function. The IAB has mandated that any protocols developed for this purpose document a specific set of considerations. These considerations and the responses for TURN are documented in this section.

Consideration 1: 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. Such generalizations lead to the prolonged dependence on and usage of the supposed short-term fix -- meaning that it is no longer accurate to call it "short-term".

Response: TURN is a protocol for communication between a relay (= TURN server) and its client. The protocol allows a client that is behind a NAT to obtain and use a public IP address on the relay. As a convenience to the client, TURN also allows the client to determine its server-reflexive transport address.

Consideration 2: 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.

Response: TURN will no longer be needed once there are no longer any NATs. Unfortunately, as of the date of publication of this document, it no longer seems very likely that NATs will go away any time soon. However, the need for TURN will also decrease as the number of NATs with the mapping property of Endpoint-Independent Mapping [RFC4787] increases.

Consideration 3: 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.

Response: TURN is "brittle" in that it requires the NAT bindings between the client and the server to be maintained unchanged for the lifetime of the allocation. This is typically done using keep-alives. If this is not done, then the client will lose its allocation and can no longer exchange data with its peers.

Consideration 4: Identify requirements for longer-term, sound technical solutions; contribute to the process of finding the right longer-term solution.

Response: The need for TURN will be reduced once NATs implement the recommendations for NAT UDP behavior documented in [RFC4787]. Applications are also strongly urged to use ICE [RFC5245] to communicate with peers; though ICE uses TURN, it does so only as a last resort, and uses it in a controlled manner.

Consideration 5: Discussion of the impact of the noted practical issues with existing deployed NATs and experience reports.

Response: Some NATs deployed today exhibit a mapping behavior other than Endpoint-Independent mapping. These NATs are difficult to work

with, as they make it difficult or impossible for protocols like ICE to use server-reflexive transport addresses on those NATs. A client behind such a NAT is often forced to use a relay protocol like TURN because "UDP hole punching" techniques [RFC5128] do not work.

21. Changes since RFC 5766

This section lists the major changes in the TURN protocol from the original [RFC5766] specification.

- o IPv6 support.
- o REQUESTED-ADDRESS-FAMILY, ADDITIONAL-ADDRESS-FAMILY, AND ADDRESS-ERRR-CODE attributes.
- o 440 (Address Family not Supported) and 443 (Peer Address Family Mismatch) responses.
- o Description of the tunnel amplification attack.
- o DTLS support.
- o More detail on packet translations.

22. Acknowledgements

Most of the text in this note comes from the original TURN specification, [RFC5766]. The authors would like to thank Rohan Mahy co-author of orginal TURN specification and everyone who had contributed to that document.

Thanks to Justin Uberti, Pal Martinsen, Oleg Moskalenko, Aijun Wang and Simon Perreault for their help on SSODA mechanism.

23. References

23.1. Normative References

[RFC5389] Rosenberg, J., Mahy, R., Matthews, P., and D. Wing, "Session Traversal Utilities for NAT (STUN)", RFC 5389, October 2008.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[RFC2474] Nichols, K., Blake, S., Baker, F., and D. Black, "Definition of the Differentiated Services Field (DS Field) in the IPv4 and IPv6 Headers", RFC 2474, December 1998.

[RFC3168] Ramakrishnan, K., Floyd, S., and D. Black, "The Addition of Explicit Congestion Notification (ECN) to IP", RFC 3168, September 2001.

[RFC1122] Braden, R., "Requirements for Internet Hosts - Communication Layers", STD 3, RFC 1122, October 1989.

[RFC6145] Li, X., Bao, C., and F. Baker, "IP/ICMP Translation Algorithm", RFC 6145, April 2011.

[RFC3697] Rajahalme, J., Conta, A., Carpenter, B., and S. Deering, "IPv6 Flow Label Specification", RFC 3697, March 2004.

23.2. Informative References

[RFC1191] Mogul, J. and S. Deering, "Path MTU discovery", RFC 1191, November 1990.

[RFC0791] Postel, J., "Internet Protocol", STD 5, RFC 791, September 1981.

[RFC1918] Rekhter, Y., Moskowitz, R., Karrenberg, D., Groot, G., and E. Lear, "Address Allocation for Private Internets", BCP 5, RFC 1918, February 1996.

[RFC3424] Daigle, L. and IAB, "IAB Considerations for UNilateral Self-Address Fixing (UNSAF) Across Network Address Translation", RFC 3424, November 2002.

[RFC4787] Audet, F. and C. Jennings, "Network Address Translation (NAT) Behavioral Requirements for Unicast UDP", BCP 127, RFC 4787, January 2007.

[RFC5245] Rosenberg, J., "Interactive Connectivity Establishment (ICE): A Protocol for Network Address Translator (NAT) Traversal for Offer/Answer Protocols", RFC 5245, April 2010.

[RFC6062] Perreault, S. and J. Rosenberg, "Traversal Using Relays around NAT (TURN) Extensions for TCP Allocations", RFC 6062, November 2010.

[RFC6156] Camarillo, G., Novo, O., and S. Perreault, "Traversal Using Relays around NAT (TURN) Extension for IPv6", RFC 6156, April 2011.

[RFC6056] Larsen, M. and F. Gont, "Recommendations for Transport-Protocol Port Randomization", BCP 156, RFC 6056, January 2011.

[RFC5128] Srisuresh, P., Ford, B., and D. Kegel, "State of Peer-to-Peer (P2P) Communication across Network Address Translators (NATs)", RFC 5128, March 2008.

[RFC1928] Leech, M., Ganis, M., Lee, Y., Kuris, R., Koblas, D., and L. Jones, "SOCKS Protocol Version 5", RFC 1928, March 1996.

[RFC3550] Schulzrinne, H., Casner, S., Frederick, R., and V. Jacobson, "RTP: A Transport Protocol for Real-Time Applications", STD 64, RFC 3550, July 2003.

[RFC3711] Baugher, M., McGrew, D., Naslund, M., Carrara, E., and K. Norrman, "The Secure Real-time Transport Protocol (SRTP)", RFC 3711, March 2004.

[RFC4302] Kent, S., "IP Authentication Header", RFC 4302, December 2005.

[RFC4303] Kent, S., "IP Encapsulating Security Payload (ESP)", RFC 4303, December 2005.

[RFC4821] Mathis, M. and J. Heffner, "Packetization Layer Path MTU Discovery", RFC 4821, March 2007.

[RFC3261] 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.

[I-D.rosenberg-mmusic-ice-nonsip] Rosenberg, J., "Guidelines for Usage of Interactive Connectivity Establishment (ICE) by non Session Initiation Protocol (SIP) Protocols", draft-rosenberg-mmusic-ice-nonsip-01 (work in progress), July 2008.

[I-D.ietf-tram-turn-server-discovery] Patil, P., Reddy, T., and D. Wing, "TURN Server Auto Discovery", draft-ietf-tram-turn-server-discovery-01 (work in progress), January 2015.

[RFC4086] Eastlake, D., Schiller, J., and S. Crocker, "Randomness Requirements for Security", BCP 106, RFC 4086, June 2005.

[RFC5766] Mahy, R., Matthews, P., and J. Rosenberg, "Traversal Using Relays around NAT (TURN): Relay Extensions to Session Traversal Utilities for NAT (STUN)", RFC 5766, April 2010.

[Port-Numbers] "IANA Port Numbers Registry", 2005, <http://www.iana.org/assignments/port-numbers>.

[Frag-Harmful] "Fragmentation Considered Harmful", <Proc. SIGCOMM '87, vol. 17, No. 5, October 1987>.

[Protocol-Numbers] "IANA Protocol Numbers Registry", 2005, <http://www.iana.org/assignments/protocol-numbers>.

Authors' Addresses

Tirumaleswar Reddy (editor)
Cisco Systems, Inc.
Cessna Business Park, Varthur Hobl
Sarjapur Marathalli Outer Ring Road
Bangalore, Karnataka 560103
India

Email: tireddy@cisco.com

Alan Johnston (editor)
Avaya
St. Louis, MO
USA

Email: alan.b.johnston@gmail.com

Philip Matthews
Alcatel-Lucent
600 March Road
Ottawa, Ontario
Canada

Email: philip_matthews@magma.ca

Jonathan Rosenberg
jdrosen.net
Edison, NJ
USA

Email: jdrosen@jdrosen.net
URI: http://www.jdrosen.net

TRAM
Internet-Draft
Intended status: Standards Track
Expires: August 16, 2015

P. Martinsen
H. Wildfeuer
Cisco

February 12, 2015

Differentiated prIorities and Status Code-points Using Stun Signalling (DISCUSS)
draft-martinsen-tram-discuss-02

Abstract

This draft describes a mechanism for information exchange between an application and the network. The information provided from the application to the network MAY be used by a network element in the path to modify its behavior to improve application quality of experience (QoE). Likewise, the information provided by the network to the application MAY be used by the application to modify its behavior to optimize for QoE.

The information provided from the application to the network can also be useful for middleboxes that are responsible for security at edges of network (e.g. firewalls) or other middleboxes in determining how to treat the packets delivered from this application.

Requirements Language

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [RFC2119].

Status of This Memo

Table of Contents

1. Introduction

In the context of Content, Mobile, Fixed Service, Service Providers, Enterprise and Private networks have a need to prioritize packet flows end-to-end. These flows are often dynamic, time-bound, encrypted, peer-to-peer, possibly asymmetric, and might have different priorities depending on network conditions, direction, time of the day, dynamic user preferences and other factors. These

factors may be time variant, and thus need to be signalled. Moreover, in many cases of peer-to-peer communication, flow information is known only to the endpoint. These considerations, coupled with the trend to use encryption for browser-to-browser communication [I-D.ietf-rtcweb-security-arch], imply that access lists, deep packet inspection and other static prioritization methods cannot be employed successfully to prioritize packet flows.

The lack of congestion control in UDP may lead to problems as described in [I-D.eggert-tsvwg-rfc5405bis]. The mechanism described in this document can be used to introduce fairness and congestion control for UDP streams.

There is a need for a solution that is easy for the application developer to use. That means consistent behavior on all supported platforms and preferably without need of administrative privileges to set and read values. The solution also needs to be able to cross administrative domains without the risk of being rewritten. [[Q1: This draft will only offer tamper detection of some of the values. Further discussion regarding the incentive to lie is needed. --palmarti]]

This draft describes how these problems can be solved by defining a few strictly defined STUN [RFC5389] attributes which can be added to any STUN message the client wants to send. STUN messages are typically sent during the ICE [RFC5245] connectivity check phase when the media session is established, or when keep-alive STUN messages are sent after the session is established. The application is not limited to those two scenarios, if some communication between application and network is needed it can choose to do so at any time.

Devices on the media path can use the information in the STUN attributes to prioritize the flow, perform traffic engineering, provide network analytics or as a gateway to existing methods for prioritizing flows (DSCP [RFC2474]). Applications can use information in network status attribute to influence rate stating points or rate adaption mechanisms.

2. General Usage

This draft defines several attributes that can be added to a STUN message; STREAM-TYPE, BANDWIDTH-USAGE, STREAM-PRIORITY and NETWORK-STATUS. See Section 6 for the formal description. It is RECOMMENDED to add them to a STUN request response pair, especially if the NETWORK-STATUS attribute is in use. This allows the information gathered to be sent back to the requesting agent in the the STUN response.

The STREAM-TYPE, BANDWIDTH-USAGE, STREAM-PRIORITY attributes MUST be added before any INTEGRITY attribute. It is RECOMMENDED to only add these attributes to STUN messages containing a INTEGRITY attribute as this prevents tampering with the content of the attribute.

If the client wants to receive feedback from the network it must add a null NETWORK-STATUS attribute. A null NETWORK_STATUS attribute is created by filling in the all the fields in the attribute with 0x0 values. This attribute MUST be added after the INTEGRITY attribute, as on-path devices may write information into this attribute. Having a readily available attribute to write into will save the the on-path device from growing buffers to add their own attribute. On path devices SHOULD not add their own NETWORK-STATUS attribute (or any other STUN attribute).

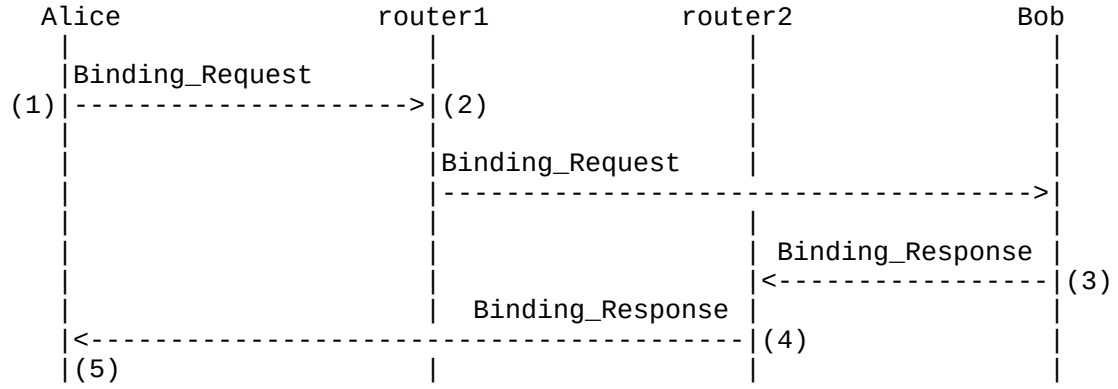
If an agent receives a STUN request with a NETWORK-STATUS attribute after the INTEGRITY attribute, it should copy the content into a new NETWORK-STATUS attribute and add it before the INTEGRITY attribute when sending the STUN response. A new null NETWORK-STATUS attribute can be added after the INTEGRITY attribute. New STUN attributes described in this draft can also be added describing the stream in this direction.

If an agent receives a STUN response with a NETWORK-STATUS attribute before the INTEGRITY attribute, this describes the stream in the upstream direction. A NETWORK-STATUS attribute after the INTEGRITY attribute describes the stream in the downstream direction.

It might make sense to distinguish DISCUSS packets from normal STUN packets. This would prevent unnecessary processing of normal STUN packets on the network nodes.

[[Q2: A few alternatives (Needs discussion): ---1: Alter the STUN magic cookie. (But than i would not be a STUN packet anymore, and that raises a new set of problems) ---2: Add a special this is DISCUSS attribute. This must be the first attribute in the message. This allows for network node to look for DISCUSS packets at a fixed offset without needing to parse the entire packet. ---3: Alter the transaction id. This might be problematic if using it in conjunction with ICE connectivity checks. But probably fine in other scenarios. ---4: Define a new STUN Method. Also brakes ICE, makes it harder to tag onto attributes to already in use messages. --palmarti]]

[[Q3: Do we want to restrict this to req/resp or do we want to allow for the attributes to be added in this fashion for indications as well? --palmarti]]



DISCUSS example flow

1. Alice creates a Binding Request, adds STREAM-TYPE, BANDWIDTH-USAGE, STREAM-PRIORITY attributes before the INTEGRITY attribute and a single null NETWORK-STATUS attribute after the INTEGRITY attribute.

2. Router1 inspects the STUN Request message and reads any STREAM-TYPE, BANDWIDTH-USAGE, or STREAM-PRIORITY attributes and the information they contain. It then updates the NETWORK-STATUS attribute with any information the router have. It then forwards the request.

3. Bob processes the STUN Request. When Bob builds the response, it copies the NETWORK-STATUS attribute into the STUN Response before the INTEGRITY check and adds new null NETWORK-STATUS attribute after the INTEGRITY attribute. Bob then transmits the message.

4. Router2 (first DISCUSS network element for the downstream direction) inspects the Response message, reads the STREAM-TYPE, BANDWIDTH-USAGE, or STREAM-PRIORITY attributes and MAY alter the NETWORK-STATUS attribute located after the INTEGRITY attribute. It then transmits the message.

5. When Alice receives the STUN Response, she can extract the STREAM-TYPE, BANDWIDTH-USAGE, or STREAM-PRIORITY attributes and the two NETWORK-STATUS attributes to get a complete picture of what the remote agent is sending and how the status of both the upstream and downstream path.

3. Network Processing

This section describes the processing of DISCUSS packets by network devices.

3.1. Packet Processing by Network Device

Network devices are said to support DISCUSS if they perform inspection of packets being forward or switched in order to identify an DISCUSS STUN packet. These devices will also be able to read/ write STUN attributes to/from this packet. It is not required that every network device in the path support DISCUSS. It is expected that DISCUSS will have the most value being implemented at certain points in the network (PIN's) such as WAN edge devices, wireless access devices, and Internet gateways.

Network devices that support DISCUSS MAY utilize the information provided by the application in the STUN attributes to modify their behavior. These include the attributes defined in this document with the exception of the NETWORK-STATUS attribute. The NETWORK-STATUS attribute SHOULD NOT be used by the DISCUSS capable network device to modify its behavior. The intent of the NETWORK-STATUS attribute is for the application to modify its behavior.

If the NETWORK-STATUS attributes exists in a DISCUSS packet after an INTEGRITY attribute, the DISCUSS capable network device MUST process it as described in this section. NETWORK-STATUS attributes that exist before the INTEGRITY attribute MUST NOT be modified by the network device. The modifications to the NETWORK-STATUS attribute are:

- Update the Node Cnt field in the attribute. The device SHALL increment this field by one unless it is at its maximum (saturated) value. If the field is at its maximum value, the device SHALL NOT modify this field.

- Overwrite the attribute CS bit if the value at this device is "worst" than the current value. In other words, only write to this bit if the device is experiencing congestion on relevant queues/interfaces for this flow AND the current value of this field is 0 (Off).

The determination of congestion at a device is out of the scope of this document. Setting of CS bit to On by the device is meant to provide direct feedback to the application of potential or current loss of packets in its flow (s). The application can then react to this indication by altering its encoding of information in the packet to deal with congestion/packet loss, e.g. reduce its encoding rate or

switch to embedded encoding. Devices SHOULD ensure that the DISCUSS capable applications that do react to congestion notification by reducing their transmission rate be treated properly to ensure fairness with non reacting applications (i.e. ensure fairness for well behaving applications).

The DISCUSS STUN packet SHOULD experience minimal extra processing delay through the DISCUSS capable network device relative to non-DISCUSS packets in the flow. The DISCUSS STUN packet MAY be placed out of order in the packet flow, but SHOULD NOT be delayed more than a few packet interval times.

3.2. Interaction with DSCP

One of the attributes that may be added to the STUN packet by the application is the STREAM-PRIORITY attribute. This attribute indicates the relative priority of streams inside of an application session. This attribute is compatible with the use of DSCP (or other priority markings) at the networking layer as described in this section.

Since transport layer markings may be modified by middle boxes or devices in the path or at the interface of the application itself due to the lack of support in the OS network stack, the STREAM-PRIORITY attribute can be used as a mechanism for ensuring proper QoS treatment through multiple domains. DISCUSS capable device may use the STREAM-PRIORITY attribute to remark the DSCP value to the appropriate value. DSCP re-marking based on STREAM-PRIORITY attribute may make sense at certain PIN’s, e.g. gateway between network domains (e.g. managed network to/from Internet), access switches in managed network, etc. The translation from the Priority number in the STREAM-PRIORITY attribute to the correct transport layer marking (e.g. DSCP) is implementation specific and out of the scope of this document.

[I-D.dhesikan-tsvwg-rtcweb-qos] provides the recommended DSCP values for webrtc enabled browsers to use for various classes of traffic.

4. Interaction with ICE

An ICE connectivity check is performed by sending a STUN Binding indication. Prior to sending the agent can add one STREAM-TYPE attribute. If added, only one MUST be added. This is to avoid unnecessary large STUN packets during the connectivity checks. If the connectivity check is sent on a 5-tuple that multiplexes different types of media and more detailed information wants to be signalled it should be done after the connectivity check phase is finished.

This limits the information the STUN messages are able to convey during the connectivity checks, but also avoids adding network confusion with BANDWIDTH-USAGE attributes describing different paths that never going to be utilized.

[[Q4: Problem with consent freshness if not based on STUN. --palmarti]]

5. Multiplexed Streams

In some scenarios a 5-tuple can be used to transport several media streams. BUNDLE [I-D.ietf-mmusic-sdp-bundle-negotiation] describes such a mechanism.

At times, the different "streams" carried in this bundle require very different treatment from the network, including the ability to prioritize some of these "streams" over others. For example, the application my bundle video and audio in the same 5-tuple flow, but would like the network to prioritize the delivery of audio over that of video in the case of congestion. Another example is the use of embedded (or scalable) coding for video. Per RFC 6190 [RFC6190], using Multi-session Transmission (MST) the layers are transported in seperate sub-flows (RTP sessions) within the bundled flow. Using the STREAM-TYPE attribute with the extension to identify the sub-flow and its priority would allow network elements, if capable, to provide differentiated services even in the case of bundling.

For RTP/SRTP based flows, the existence and attributes for sub-flows in the flow MAY be indicated by the application via the SUB-STREAM-xxx attributes. These attributes MUST only be included if the equivalent STREAM-xxxx attributes are included. It is expected that only a sub-set of network elements representing bottleneck Points in Network (PIN) will be able to inspect the higher layer protocols to differentiate sub-flows, so it is important to describe the agregate flow, and then the sub-flows. The SUB-STREAM-xxxx attributes are similar to the corresponding STREAM-xxxx attributes with the addition of the application layer identifier field. For the case of RTP/SRTP, this field is the SSRC assigned to the flow. Note that this will only work for non-header encrypted SRTP.

When describing the aggregate stream with a STREAM-TYPE attribute there are two possibilities to describe the streams that are multiplexed. Adding one attribute for each type (Audio, Video,++), or to save a few bits on the wire it is also possible to construct the STREAM-TYPE so a one type value describes several types. For example audio have the value of 1 and application data have the value of 4. If the STREAM_TYPE value is set to 5 the only combination that gives that is audio and application data. As previously discussed,

in the case of bundling, the agregate stream attribute MUST be included before the optional sub-stream attributes

The other attributes BANDWIDTH-USAGE, STREAM-PRIORITY and NETWORK-STATUS SHOULD only be added once as they describe the behavior of the 5-tuple and not individual streams.

6. New STUN attributes

This STUN extension defines the following new attributes:

0xXXX0: STREAM-TYPE
0xXXX1: BANDWIDTH-USAGE
0xXXX2: STREAM-PRIORITY
0xXXX8: SUB-STREAM-TYPE
0xXXX9: SUB-BANDWIDTH-USAGE
0xXXXA: SUB-STREAM-PRIORITY
0xYYYY: NETWORK-STATUS

6.1. STREAM-TYPE

This attribute have a length that are multiples of 4 (32) so no padding is necessary.

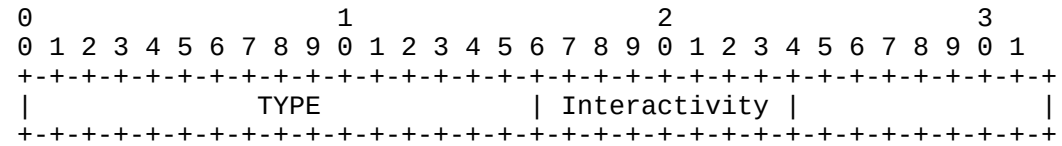


Figure 1: STREAM TYPE Attribute

TYPE: STREAM TYPE is a 16 bit value defined in Figure 2 below describing the flow.

0x0001 Audio
0x0002 Video
0x0004 Application Data
0x0008 Other

Figure 2: STREAM Types

Interactivity: Is a 8 bit value defined in Figure 3 below describing the flow.

```
0x00 Undef
0x01 Stream (Broadcast? Oneway?)
0x02 Interactive
```

Figure 3: Interactivity Types

It is possible to combine the stream types if a stream contains more than one type.

If a 5-tuple is used to send both a audio and video stream, the stream type can be set to 0x0006. This can be useful if the application wants to hint that the 5-tuple contains several streams, This is useful if the attribute is added to STUN binding requests during ICE connectivity checks. If more information regarding multiplexed streams is needed it is possible to add more than one attribute to a STUN message (See section ??). This can be done to STUN messages that are being sent after the connectivity check phase is finished (Keep-alive, consent freshness). During this phase the added size of the STUN messages pose no security threat.

6.2. BANDWIDTH-USAGE

This attribute have a length that are multiples of 4 (32) so no padding is necessary.

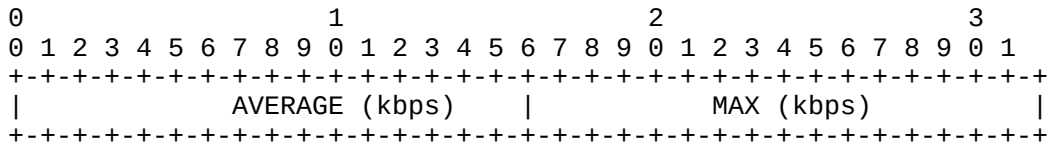


Figure 4: BANDWIDTH USAGE Attribute

AVERAGE: Expected sustained bandwidth usage for this stream. Note that for elastic types of streams like video, sudden large movements in the picture my lead to this value being inaccurate.

MAX: The maximum bandwidth usage for this stream. If the sustained and max value differ greatly it might be safe to assume that an elastic encoder is in use. (Would it be useful to say something about expected BURST lengths?)

6.3. STREAM-PRIORITY

This attribute have a length that are multiples of 4 (32) so no padding is necessary.

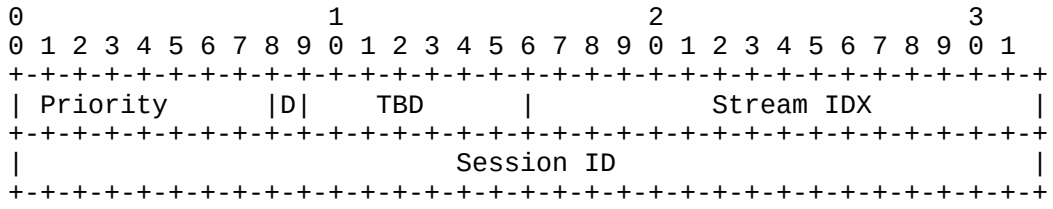


Figure 5: STREAM PRIORITY Attribute

Priority: Describes this streams priority among other streams coming from this endpoint/application (With same session ID). Values range from 255 (0xFF) to 0.

D: Delay sensitive. The application can set this bit as a hint to the network element that the stream is delay sensitive. (Unsure if this is useful)

Stream IDX: Application can choose set this to ease debugging in the network. A reasonable value can foe example be the index have in the SDP.

Session ID: Identification to distinguish what session this stream is part of. This MUST have the same value for all the media streams the application wants to give differentiated services. (Note that this ID may overlap with other streams that originates from a different IP address. The network element MUST only prioritize among streams with the same Session Id originating from the same IP)

6.4. NETWORK-STATUS

This attribute have a length that are multiples of 4 (32) so no padding is necessary. The values are kept in the same attribute to make it easier for the network element to process it. Only one attribute, with static placement of the fields. [[Q5: Does this matter? Could we have several attributes with possible different ordering without any problem for the network element? --palmarti]]

This attribute MUST be added after any INTEGRITY attribute in the STUN message. Values in this attribute can be updated along the network path by nodes that are not able to regenerate a correct INTEGRITY attribute.

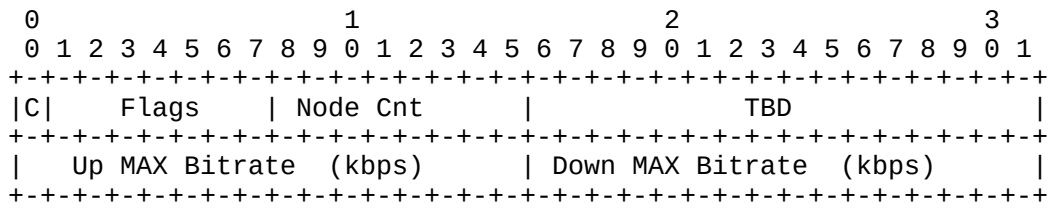


Figure 6: NETWORK-STATUS Attribute

C: Congestion Status. This bit is set to indicate that there is congestion at the network device's relevant queues/interfaces for this flow. The network element should set this bit to 1 (On) if it is experiencing congestion. This bit is set to 0 (off) when the attribute is created by the application. The application that sees this bit set might act on it by doing some rate adaption or similar action.

Flags: 7 more bits available for flags.

Node Cnt: Numbers of nodes that supports DISCUSS in the network path. Any router on the path that understands DISCUSS should increase this number. This field is set to 0 when the attribute is created by the application.

TBD: 16 more bits available for useful info.

Up MAX Bitrate: Available MAX bit-rate the router is able to handle for the 5-tuple in the UP direction. (Same direction as the packet is moving)

Down MAX Bitrate: Available MAX bit-rate the router is able to handle for the 5-tuple in the DOWN direction. (Opposite direction as the packet is moving)

6.5. SUB-STREAM-TYPE, SUB-STREAM-PRIORITY

These attributes are identical format to their aggregate stream version (STREAM-TYPE, STREAM-PRIORITY) with the addition of a transport layer identifier. The transport layer identifier is a 64 bit field which contains the unique identifier of the sub-stream for which the attribute applies.

Currently, only RTP transport is supported with the identifier being the SSRC currently used by the sub-stream.

7. IANA Considerations

IANA is requested to add the following attributes to the STUN attribute registry [iana-stun],

- 0xXXX0: STREAM-TYPE (0xXXX0, in the comprehension-optional range)
- 0xXXX1: BANDWIDTH-USAGE (0xXXX1, in the comprehension-optional range)
- 0xXXX2: STREAM-PRIORITY (0xXXX2, in the comprehension-optional range)
- 0xYYYY: NETWORK-STATUS (0xYYYY, in the comprehension-optional range)

8. Implementation Status

[Note to RFC Editor: Please remove this section and reference to [RFC6982] prior to publication.]

This section records the status of known implementations of the protocol defined by this specification at the time of posting of this Internet-Draft, and is based on a proposal described in [RFC6982]. The description of implementations in this section is intended to assist the IETF in its decision processes in progressing drafts to RFCs. Please note that the listing of any individual implementation here does not imply endorsement by the IETF. Furthermore, no effort has been spent to verify the information presented here that was supplied by IETF contributors. This is not intended as, and must not be construed to be, a catalog of available implementations or their features. Readers are advised to note that other implementations may exist.

According to [RFC6982], "this will allow reviewers and working groups to assign due consideration to documents that have the benefit of running code, which may serve as evidence of valuable experimentation and feedback that have made the implemented protocols more mature. It is up to the individual working groups to use this information as they see fit".

8.1. NATtools

Organization: Cisco

Description: Open-Source ICE, TURN and STUN implementation.

Implementation: https://github.com/cisco/NATTools

Level of maturity: Code is stable. Tests being run to learn more on how to leverage the information shared between client and network.

Coverage: Implements the DISCUSS attributes

Licensing: BSD

Implementation experience: Draft was implemented with internal video test clients. Wireless access point implemented STUN detection in the media path and acted on the information in the DISCUSS attributes. After running tests in different congestion scenarios it is clear that sharing information between endpoint and network can help with congestion and end-user experience. This approach required little effort to implement on the client side.

Contact: Paal-Erik Martinsen <palmarti@cisco.com>.

9. Security Considerations

Due to the security implications described in [I-D.thomson-mmusic-ice-webrtc] where large STUN packet are used to amplify an attack, keeping the added STUN attributes small is a important design consideration.

To avoid unwanted information leakage the new defined STUN attributes defined in this draft are strictly defined. No more information should be leaked that an on-path device could learn by observing the stream over time or do some deep packet analysis. This draft would benefit from more discussions on this topic.

It is also worth noticing that the STUN attributes defined should be treated as hints, and more work is needed regarding how to deal with misbehaving clients or network devices.

10. Acknowledgements

Authors would like to thank Dan Wing, Anca Zamfir, Jon Snyder and Cullen Jennings for their comments and review.

11. References

11.1. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[RFC2474] Nichols, K., Blake, S., Baker, F., and D. Black, "Definition of the Differentiated Services Field (DS Field) in the IPv4 and IPv6 Headers", RFC 2474, December 1998.

[RFC5389] Rosenberg, J., Mahy, R., Matthews, P., and D. Wing, "Session Traversal Utilities for NAT (STUN)", RFC 5389, October 2008.

[RFC5245] Rosenberg, J., "Interactive Connectivity Establishment (ICE): A Protocol for Network Address Translator (NAT) Traversal for Offer/Answer Protocols", RFC 5245, April 2010.

[RFC6190] Wenger, S., Wang, Y., Schierl, T., and A. Eleftheriadis, "RTP Payload Format for Scalable Video Coding", RFC 6190, May 2011.

[RFC6982] Sheffer, Y. and A. Farrel, "Improving Awareness of Running Code: The Implementation Status Section", RFC 6982, July 2013.

11.2. Informational References

[I-D.ietf-rtcweb-security-arch]
Rescorla, E., "WebRTC Security Architecture", draft-ietf-rtcweb-security-arch-09 (work in progress), February 2014.

[I-D.thomson-mmusic-ice-webrtc]
Thomson, M., "Using Interactive Connectivity Establishment (ICE) in Web Real-Time Communications (WebRTC)", draft-thomson-mmusic-ice-webrtc-01 (work in progress), October 2013.

[I-D.dhesikan-tsvwg-rtcweb-qos]
Dhesikan, S., Druta, D., Jones, P., and J. Polk, "DSCP and other packet markings for RTCWeb QoS", draft-dhesikan-tsvwg-rtcweb-qos-06 (work in progress), March 2014.

[I-D.ietf-mmusic-sdp-bundle-negotiation]
Holmberg, C., Alvestrand, H., and C. Jennings, "Multiplexing Negotiation Using Session Description Protocol (SDP) Port Numbers", draft-ietf-mmusic-sdp-bundle-negotiation-05 (work in progress), October 2013.

[I-D.eggert-tsvwg-rfc5405bis]
Eggert, L., Fairhurst, G., and G. Shepherd, "UDP Usage Guidelines", draft-eggert-tsvwg-rfc5405bis-01 (work in progress), June 2014.

[iana-stun]
IANA, , "IANA: STUN Attributes", April 2011, <http://www.iana.org/assignments/stun-parameters/stun-parameters.xml>.

Authors' Addresses

Paal-Erik Martinsen
Cisco Systems, Inc.
Philip Pedersens vei 20
Lysaker, Akershus 1366
Norway

Email: palmarti@cisco.com

Herb Wildfeuer
Cisco Systems, Inc.
821 Alder Drive
Milpitas, California 95035
United States

Email: hwildfeu@cisco.com

TRAM
Internet-Draft
Intended status: Standards Track
Expires: August 31, 2015

P. Martinsen
D. Wing
Cisco

February 27, 2015

STUN Traceroute
draft-martinsen-tram-stuntrace-00

Abstract

After a UDP protocol such as RTP determines a network path is experiencing problems, a traceroute is often useful to determine which router or which link is contributing to the problem. However, operating system traceroute commands follow a different path than the actual UDP flow which complicates troubleshooting. A superior method is shown which is absolutely path-congruent with the UDP protocol itself, works on IPv4 and IPv6, and does not require administrative privileges on most operating systems.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at http://datatracker.ietf.org/drafts/current/.

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."

This Internet-Draft will expire on August 31, 2015.

Table of Contents

1. Introduction

Traceroute [RFC1393] is a simple tool available on most operating systems and is popular to debug the network by simply getting round-trip time along each hop to a remote IP address. More advanced tools, such as MTR, provide more metrics such as packet loss and round trip time to each hop over several seconds or minutes.

To simplify network debugging when dealing with bi-directional real time media it is often useful to get as much information as possible regarding the network path. In this specification probe packets are sent using the same 5-tuple where (S)RTP media is flowing. This will provide the most accurate results, as probe packets sent on a

different 5-tuple may take another path due to Equal-Cost Multipath (ECMP, [RFC2992]), policy-based routing, and similar techniques.

To avoid those problems, the probe packets need to be sent from the same socket and with the same DiffServ code point the normal (S)RTP media packets. As shown in the examples below, most operating systems can pass the ICMP "Time to Live Exceeded" error to the application, so the application can perform the diagnostics over that network path.

This specifications uses STUN [RFC5389] packets as probes. STUN packets are designed to be multiplexed together with RTP [RFC3550] (and SRTP [RFC3711]) and are unlikely to cause any "problems" for the (S)RTP receiver. To differentiate each hop count, classic traceroute uses different UDP port numbers (e.g., TTL=1 uses UDP port 55001, TTL=2 uses UDP port 55002, etc.). The mechanism described here uses the same UDP port number (so that the trace is path-congruent with the (S)RTP packets), and uses different length UDP packets to differentiate each hop count (e.g., TTL=1 uses length 501, TTL=2 uses length 502, etc.).

To receive richer information from each hop, we also describe how to use ICMP extensions [RFC4884].

Using a technique based on ICMP replies avoids a forklift upgrade of the network to provide host applications with useful information. ICMP is already supported in most network and application stacks. It also provides a solution for routers and application to share a richer set of information through the STUN probes itself and embedding STUN responses with additional information to the application in the ICMP reply.

2. Notational Conventions

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119].

3. Overview of Operation

An application using (S)RTP to send and receive media like audio and video following the guidelines in [RFC4961] uses symmetric send and receive ports. The application opens one socket that it uses to both send and receive media on.

Multiplexing media-streams as described in Bundle [I-D.ietf-mmusic-sdp-bundle-negotiation] would limit the number of media-paths needing tracing for trouble shooting purposes. The

method described here can be used to trace numerous paths simultaneously and does not rely on any multiplexing to work.

It is important to note that the functionality described here can be done on most OSes without any administrative privileges.

Figure 1 depicts the various components needed for this to work. The application opens up its media socket as it would in normal cases where media is to be sent and received. It also opens up a ICMP socket or installs an error listener on the media socket.

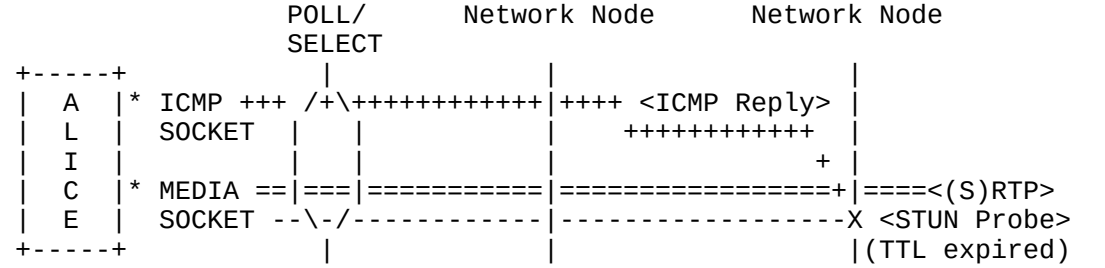


Figure 1

The application also need to listen on the sockets for any incoming ICMP packets or socket error messages. This is usually done with the socket calls select() or poll(). How to actually receive the ICMP messages will vary from OS to OS. See Appendix A for implementation details on various OSes.

Once the application have media running and is listening for ICMP replies it can start sending probes to detect networks nodes in the media path. This is done by sending STUN messages and setting the TTL/MAX_HOP limit in the IPv4/IPv6 header. Appendix A.1 explains how to set this on various platforms.

The STUN packet is sent on the same socket as the media packet are sent and received on. Mixing (S)RTP and STUN is well known behavior and should not cause any problems.

Along the path, every layer 3 network node (a.k.a. router) decreases the IPv4 TTL or IPv6 HOP_LIMIT field. If the field becomes 0 the

network node responds with a ICMP error "Time to Live Exceeded" (TTL Exceeded) or "Hop Limit Exceeded in Transit" (Time Exceeded Message).

The application will receive a ICMP error in response to the offending probe packet. The source IP address of the ICMP packet will be the sending network node. This enables the application to trace the path towards the destination. The ICMP reply contains at least 8 bytes of the offending packet. The IP fragment of the offending packet in the ICMP reply can be used to determining if this ICMP reply actually was a reply to an offending packet the application did send out.

4. New STUN Attributes

This STUN extension defines the following new attribute:

0xXXX0: PATH-NODE-PROBE

4.1. PATH-NODE-PROBE

This attribute have a length of 8. Padding is needed to hit the required STUN 32 bit STUN attribute boundary.

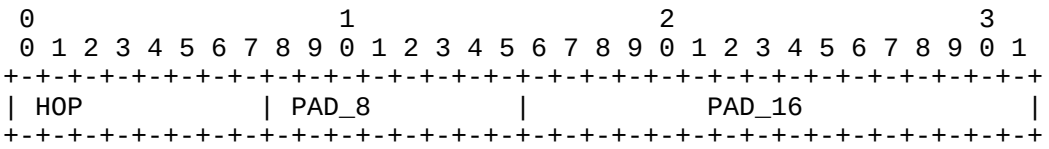


Figure 2: PATH-NODE-PROBE Attribute

The HOP field indicates what hop in the network path (relative to the application) the application is trying to learn the IP address of. This field should be set to the same value as the TTL/HOP_LIMIT field in the IPv4/IPv6 header of the probe packet leaving the application. Note that the TTL/HOP_LIMIT field in the IPv4/IPv6 header will decrease as the packet traverses the path. The HOP field in the attribute will remain unchanged.

This attribute is useful for clients when receiving the whole offending IP packet in the ICMP reply or when the rich ICMP feedback mechanism described in Section 6 is in use. The attribute will also be reflected back in a STUN response if the remote application supports is. This makes it easier to correlate sent probe packets and ICMP responses.

5. Base Protocol Procedures

The procedures are simple; send a probe packet that may or may not trigger a reply from one of the nodes in the network path and then listen and parse any incoming replies. The reply might be an ICMP Time To Live Exceeded (from an intermediate hop), a STUN response (from the (S)RTP peer), or any other ICMP error message.

5.1. Forming STUN Packet Probes

To reduce chances of a STUN traceroute probe being stopped by various middle-boxes it is RECOMMENDED to use a STUN binding request as described in ICE [RFC5245].

Since the STUN packet can traverse the whole media-path and reach the remote peer it is RECOMMENDED the agent follows the guidelines for sending connectivity checks defined in the ICE [RFC5245]. Adding a USERNAME attribute and integrity protecting the STUN message enables the remote peer to authenticate the STUN message and create an appropriate response. If the remote peer is unable to authenticate the STUN request it will not send any response. Getting a response from the remote peer is useful as it is an indication the probe have traveled the whole network path.

When forming the STUN packet probe the agent SHOULD add the PATH-NODE-PROBE attribute and MAY add a PADDING attribute as described in [RFC5780] Section 7.6. The PATH-NODE-PROBE attribute is useful for STUN servers receiving the STUN probe and it can be used to correlate any ICMP replies if the reply contains the complete offending packet. Adding the PADDING attribute is useful for clients that needs to have several outstanding probe packets on the same 5-tuple. The length of the offending packet reported back in any ICMP reply will make it possible to correlate this to the correct probe.

The agent sending the STUN packet probe MUST store the length of the UDP packet (as reported in the IP header) containing the STUN probe.

Before sending the probe on the wire it is important to set the appropriate TTL or HOP_LIMIT field in the IPv4 or IPv6 header before the packet is sent. How to do this on various OSes are described in Appendix A.1.

The probe MUST also be sent with the same DSCP value as the (S)RTP packets. This is normally not a problem as the STUN probes and (S)RTP packets are sent on the same socket.

5.2. Receiving a STUN Packet Probe

An agent that listens to for STUN requests (a.k.a STUN server) that receives a STUN request with a PATH-NODE-PROBE attribute, MUST include a PATH-NODE-PROBE attribute with the same value in the generated response.

Any PADDING attributes as defined in [RFC5780] SHOULD be ignored by the STUN server.

5.3. Receiving ICMP Messages

After an agent sends a STUN probe it must be ready to receive a ICMP reply or a STUN reply. Details on how to do this on various OSes are described in Appendix A.2.

To prevent ICMP spoofing attacks [RFC5927] , the received ICMP packet MUST be validated by port number and length in the IP fragment of the offending packet contained in the ICMP payload. Port number validation checks that the port number in the offending IP fragment of the probe packet contained in the ICMP payload corresponds to the (S)RTP media (and STUN probe) 5-tuple. The length validation checks IP packet length field in the IP fragment of the offending packet received in the ICMP reply. This value MUST correspond to any length stored when the agent sent the STUN probe. If the agent uses the PADDING (Defined in [RFC5780]) attribute to generate different length on the STUN probes it is possible to have several outstanding probes, thus speeding up the trace.

6. Rich Network Feedback

ICMP can be extended to include Multi part messages as described in [RFC4884]. In the Discuss document [I-D.martinsen-tram-discuss], we explored the possibility of using STUN as a transport protocol to get network feedback back to the client. If the network element understands STUN it is possible to create a STUN response and add that to the ICMP reply, in addition to the mandatory 8 bytes. This STUN response can carry additional information from the network to the client.

6.1. Forming and Sending Request

When forming the STUN packet probe the agent can add any of the attributes described in the discuss draft [I-D.martinsen-tram-discuss].

6.2. Receiving and Responding to Request

If the network element understand the STUN message probe containing discuss attributes, it creates a STUN response as described in the discuss draft and put it into the ICMP response.

6.3. Receiving a Response

If the agent receives a ICMP reply that contains a STUN response, it MUST be processed as an ordinary STUN response. This will end the STUN transaction.

7. IPv4 and IPv6 Differences

Core functionality is the same. Some code point differences. (more to be added later)

8. IANA Considerations

The codepoint for the new STUN attribute defined in this specification is described in Section 4.

9. Security Considerations

ICMP messages does leak network topology, which is a well-known threat to networks and mitigations have long existed in routers and firewalls so that networks can be configured to not leak this topology information beyond their borders.

ICMP spoofing and DOS attack prevention exist in routers deployed on the Internet today.

No new threats have been added in this specification.

10. Acknowledgements

Trond Andersen for actually implementing this and Wilson Chen for helping out with different OS behavior testing.

11. References

11.1. Normative References

[I-D.martinsen-tram-discuss]
Martinsen, P. and H. Wildfeuer, "Differentiated prIorities and Status Code-points Using Stun Signalling (DISCUSS)", draft-martinsen-tram-discuss-02 (work in progress), February 2015.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[RFC3550] Schulzrinne, H., Casner, S., Frederick, R., and V. Jacobson, "RTP: A Transport Protocol for Real-Time Applications", STD 64, RFC 3550, July 2003.

[RFC3711] Baugher, M., McGrew, D., Naslund, M., Carrara, E., and K. Norrman, "The Secure Real-time Transport Protocol (SRTP)", RFC 3711, March 2004.

[RFC4884] Bonica, R., Gan, D., Tappan, D., and C. Pignataro, "Extended ICMP to Support Multi-Part Messages", RFC 4884, April 2007.

[RFC4961] Wing, D., "Symmetric RTP / RTP Control Protocol (RTCP)", BCP 131, RFC 4961, July 2007.

[RFC5245] Rosenberg, J., "Interactive Connectivity Establishment (ICE): A Protocol for Network Address Translator (NAT) Traversal for Offer/Answer Protocols", RFC 5245, April 2010.

[RFC5389] Rosenberg, J., Mahy, R., Matthews, P., and D. Wing, "Session Traversal Utilities for NAT (STUN)", RFC 5389, October 2008.

[RFC5780] MacDonald, D. and B. Lowekamp, "NAT Behavior Discovery Using Session Traversal Utilities for NAT (STUN)", RFC 5780, May 2010.

11.2. Informative References

[I-D.ietf-mmusic-sdp-bundle-negotiation]
Holmberg, C., Alvestrand, H., and C. Jennings, "Multiplexing Negotiation Using Session Description Protocol (SDP) Port Numbers", draft-ietf-mmusic-sdp-bundle-negotiation-05 (work in progress), October 2013.

[ICMPTest]
"ICMP test github repo", <https://github.com/palerikm/ICMPTest/>.

[RFC1393] Malkin, G., "Traceroute Using an IP Option", RFC 1393, January 1993.

[RFC2992] Hopps, C., "Analysis of an Equal-Cost Multi-Path Algorithm", RFC 2992, November 2000.

[RFC5927] Gont, F., "ICMP Attacks against TCP", RFC 5927, July 2010.

Appendix A. Platform Implementation Details

This section provides examples and hint on how probe packets can be sent and ICMP messages received on various OSes. For a complete example please refer to [ICMPTest].

A.1. Setting TTL or HOP_LIMIT on Probes

Setting the appropriate value in the IPv4 or IPv6 header is the same for most platforms. Use

```
setsockopt(sockHandle, IPPROTO_IP, IP_TTL, &sock_ttl,
sizeof(sock_ttl));
```

for IPv4 or

```
setsockopt(sockHandle,
         IPPROTO_IPV6, IPV6_UNICAST_HOPS, &sock_ttl,
         sizeof(sock_ttl));
```

for IPv6.

Sending the probes on the same socket as media is flowing requires the implementations to only set this when sending the probe packet. Remember to set it back to initial value when sending media. Most OSes seems to handle the setsockopt call correctly and not set the value in the IP header of any buffered packets.

A.2. Receiving ICMP Messages

A.2.1. OS-X and iOS

Creating a socket to listen for incoming ICMP messages can be done as:

```
icmpSocket=socket(config.remoteAddr.ss_family, SOCK_DGRAM,
                  IPPROTO_ICMP); <<<
```

This is done in addition to the normal socket used to send media on (RTP) and probes. (Yes, even if the probe are sent on the media socket the ICMP reply will be on the ICMP sockets..)

Code in the while(1) loop of poll would look something like:

```
       for(i=0;i<numSockets;i++){
           if (ufds[i].revents & POLLIN) {
               if(i == rtpSock){
                   //Handle "normal" data here.
               }
               if(i == icmpSock){//This is the ICMP socket
                  //Handle ICMP packets here.
               }
           }
       }
```

A.2.2. Linux and Android

For unprivileged recipient of the ICMP messages an error handler must be installed. This can be done like:

```
   setsockopt (config.sockfd, SOL_IP,
               IP_RECVERR, &val, sizeof(val)) < 0);
```

In the poll() section of the code something like this needs to be there:

```
       struct msghdr msg;

       if (ufds[dataSock].revents & POLLERR) {
           if (recvmsg(sockfd, &msg, MSG_ERRQUEUE ) == -1) {
               //Ignore for now. Will get it later..
               continue;
           }
           //possible ICMP message
           //use cmsg to read the structures in msg
       }
```

Failing to call rcvmsg seems to let the msg fall through to the kernel. Looks like it will close down the socket because of the received error. So be careful!

For application with the right administrative privileges it is possible create a separate ICMP listen socket as described in the previous section. The socket() call would then look like:

```
               icmpSocket=socket(config.remoteAddr.ss_family, SOCK_RAW,
               IPPROTO_ICMP);
```

The poll() loop will be as described for OS-X and iOS. No need for a error handler.

A.2.3. Windows

You might be able to get the IP address of the router returning the ICMP error by using an unconnected UDP socket, and looking at the 'from' sockaddr returned by recvfrom() or WSARecvMsg() when it returns an error.

That is, we think that when an ICMP error is received and the embedded header matches what the socket uses, recvfrom/WSARecvMsg will complete with an error, and put the ICMP sender's sockaddr in the 'from' field.

Authors' Addresses

Paal-Erik Martinsen
Cisco Systems, Inc.
Philip Pedersens Vei 22
Lysaker, Akershus 1325
Norway

Email: palmarti@cisco.com

Dan Wing
Cisco Systems, Inc.
170 West Tasman Drive
San Jose, California 95134
USA

Email: dwing@cisco.com

TRAM
Internet-Draft
Intended status: Standards Track
Expires: July 24, 2015

T. Reddy
D. Wing
P. Martinsen
Cisco
V. Singh
callstats.io

January 20, 2015

Discovery of path characteristics using STUN
draft-reddy-tram-stun-path-data-01

Abstract

A host with multiple interfaces needs to choose the best interface for communication. Oftentimes, this decision is based on a static configuration and does not consider the path characteristics, which may affect the user experience.

This document describes a mechanism for an endpoint to discover the path characteristics using Session Traversal Utilities for NAT (STUN) messages. The measurement information can then be used to influence the endpoint's Interactive Connectivity Establishment (ICE) candidate pair selection algorithm.

Table of Contents

1. Introduction

The ICE [RFC5245] mechanism uses a prioritization formula to order the candidate pairs and perform connectivity checks, in which the most preferred address pairs are tested first and when a sufficiently good pair is discovered, that pair is used for communications and further connectivity tests are stopped. This approach works well for an endpoint with a single interface, but is too simplistic for endpoints with multiple interfaces, wherein a candidate pair with a lower priority might infact have better path characteristics (e.g., round-trip time, loss, etc.). The ICE connectivity checks can assist in measuring the path characteristics, but as currently defined, the STUN responses to re-transmitted requests are indistinguishable from each other.

This draft extends STUN [RFC5389] to distinguish STUN responses to re-transmitted requests and this assists the client in determining the path characteristics like round-trip time (RTT) and packet loss in each direction between endpoints. These metrics can then be used by the controlling agent to influence the ICE candidate pair priorities.

The PATH-CHARACTERISTICS attribute introduced in this specification can be used in ICE connectivity checks (STUN Binding request and response). When multiple TURN servers are discovered then this new attribute can also be used with Allocate request to determine the priority amongst the relayed candidates.

This specification can be used with the regular nomination procedure defined in ICE, wherein ICE connectivity checks need to be performed on all or subset of the chosen candidate pairs. Finalizing an approporiate candidate pair based on the path characteristics depends on the number of candidate pairs, time interval for pacing ICE connectivity checks and the corresponding RTO values. By picking appropriate values the endpoints will not observe any noticeable impact to the media setup time.

TODO: Add details of ICE continous nomination procedure discussed in mmusic WG which allows picking better candidate pairs as and when they appear. http://juberti.github.io/draughts/nombis/draft-uberti-mmusic-nombis.html explains simplifying and improving the procedures for candidate nomination in ICE to make dynamic decisions.

2. Notational Conventions

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119].

This note uses terminology defined in ICE [RFC5245] and STUN [RFC5389].

3. Path characteristics determination mechanism

When multiple paths are available for communication, the endpoint sends ICE connectivity checks across each path (candidate pair) and perhaps chooses the path with the lowest round trip time. Choosing the path with the lowest round trip time is a reasonable approach, but re-transmits can cause an otherwise-good path to appear flawed. However, STUN’s retransmission algorithm [RFC5389] cannot determine the round-trip time (RTT) if a STUN request packet is re-transmitted, because each request and retransmission packet is identical. Further, several STUN requests may be sent before the connectivity between candidate pairs is ascertained (see Section 16 of [RFC5245]). To resolve the issue of identical request and response packets in a STUN transaction, this document changes that retransmission behavior for idempotent packets. In addition to determining RTT, it is also desirable to detect which path direction caused packet loss, described as "bi-directional path characteristics," below. This is

achieved by defining a new STUN attribute and requires compliant STUN (TURN, ICE) endpoints to count request packets.

This specification defines a new comprehension-optional STUN attribute PATH-CHARACTERISTIC. PATH-CHARACTERISTIC will have a STUN Type TBD-CA. This type is in the comprehension-optional range, which means that STUN agents can safely ignore the attribute if they do not understand it.

If a client wishes to determine the path characteristics, it inserts the PATH-CHARACTERISTIC attribute in a STUN request. In the PATH-CHARACTERISTIC attribute client sends the number of times the STUN request is retransmitted with the same Transaction ID. The server would echo back the retransmission count in the response so that client can distinguish STUN responses from the re-transmitted requests. Hence, the endpoint can use the STUN requests and responses to determine the round-trip time (RTT). The server may also convey the number of times it received the request with the same Transaction ID and the number of responses it has sent for the STUN request to the client. Further, this information enables the client to determine packet loss in each direction.

3.1. The PATH-CHARACTERISTIC attribute in request

The PATH-CHARACTERISTIC attribute in a STUN request takes a 1-byte Value, which means that the Length is 1 and 3 bytes of padding are required after the value (Section 15 of [RFC5389]). When sending a STUN request, the PATH-CHARACTERISTIC attribute allows a client to indicate to the server that it wants to determine path characteristics. If the client receives a STUN response with error code 420 (Unknown Attribute) and PATH-CHARACTERISTIC is listed in the UNKNOWN-ATTRIBUTE attribute of the message, the client SHOULD retransmit the original request without the PATH-CHARACTERISTIC attribute. However this case is not expected to occur, due to the use of the comprehension-optional attribute type.

This specification updates one the STUN message structuring rules explained in Section 6 of [RFC5389] that resends of the same request reuse the same transaction ID and are bit-wise identical to the previous request. For idempotent packets the ReTransCnt in the PATH-CHARACTERISTIC attribute will be incremented by 1 for every re-transmission and the re-transmitted STUN request MUST be bit-wise identical to the previous request except for the ReTransCnt value.

The format of the value in PATH-CHARACTERISTIC attribute in the request is:

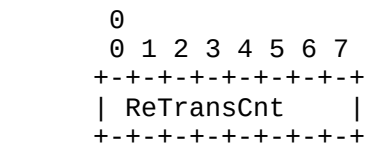


Figure 1: PATH-CHARACTERISTIC attribute in request

The field is described below:

ReTransCnt: Number of times request is re-transmitted with the same transaction ID to the server.

3.2. The PATH-CHARACTERISTIC attribute in response

When a server receives a STUN request that includes a PATH-CHARACTERISTIC attribute, it processes the request as per the STUN specification [RFC5389] plus the specific rules mentioned here. The server checks the following:

- If the PATH-CHARACTERISTIC attribute is not recognized, ignore the attribute because its type indicates that it is comprehension-optional. This should be the existing behavior as explained in section 3.1 of [RFC5389].

- The server that supports PATH-CHARACTERISTIC attribute MUST echo back ReTransCnt in the response.

- If the server is stateless or does not want to remember the transaction ID then it would populate value 0 for the ReqTransCnt and RespTransCnt fields in PATH-CHARACTERISTIC attribute sent in the response .If the server is stateful then it populates ReqTransCnt with the number of times it received the STUN request with the same transaction ID and RespTransCnt with the number of responses it has sent for the STUN request.

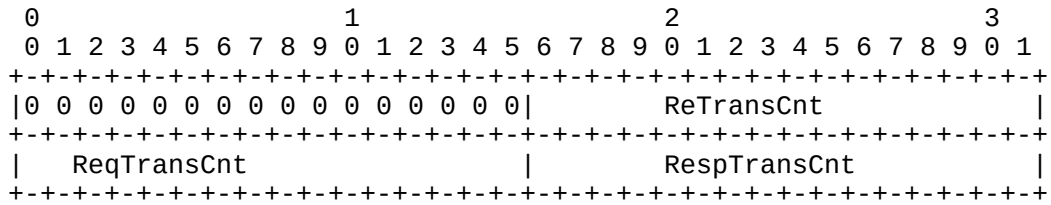


Figure 2: PATH-CHARACTERISTIC attribute in response

The fields are described below:

ReTransCnt: Copied from request.

ReqTransCnt: Number of times request is received from the client with the same transaction ID.

RespTransCnt: Number of responses sent to the client for the same transaction ID.

3.3. Example Operation

The operation is described in Figure 3. In the first case, all the requests and responses are received correctly. In the upstream loss case, the first request is lost, but the second one is received correctly, the client on receiving the response notes that while 2 requests were sent, only one was received by the server, also the server realizes that the RespTransCnt does not match the ReTransCnt, therefore 1 request was lost. This may also occur at startup in the presence firewalls or NATs that block unsolicited incoming traffic. In the downstream loss case, the responses get lost, client expecting multiple response notes that while the server responded to 3 requests but only 1 response was received. In the both loss case, requests and responses get lost in tandem, the server notes one request packet was not received, while the client expecting 3 responses received only one, it notes that one request and response packets were lost.

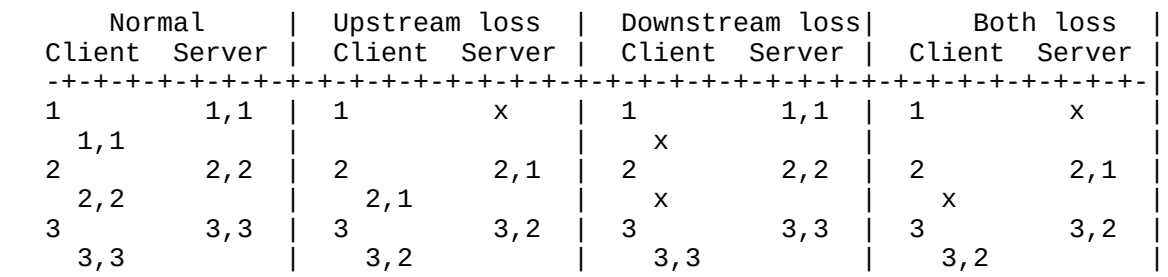


Figure 3: Retransmit Operation between client and Server

4. Usecases

The STUN attribute defined in this specification can be used by applications in the following scenarios:

- When an endpoint has multiple interfaces (for example 3G, 4G, WiFi, VPN, etc.), an ICE agent can choose the interfaces for media streams according to the path characteristics. After STUN responses to STUN checks are received, the ICE agent using regular nomination can sort the ICE candidate pairs according to the path characteristics (loss and RTT) discovered using STUN. The

controlling agent can then assign the highest priority to candidate pair which best fulfills the desired path characteristics. However, it should be noted that the path capacity or throughput is not determined by these STUN checks. If an endpoint needs to pick paths based on capacity, it would have to send media on those paths.

- o When a host has multiple interfaces available an MPRTP [I-D.ietf-avtcore-mprtp] application can choose the interfaces for the corresponding subflows according to the path characteristics discovered using STUN. For example, the scheduling algorithm described in [ACM-MPRTP] uses path capacity, latency, and loss rate for choosing the most suitable subset of paths.

- o The STUN extension proposed in this specification can also be used to choose a TURN server that provides the best user experience (section 3.1 of [I-D.patil-tram-turn-serv-selection]).

5. IANA Considerations

[Paragraphs in braces should be removed by the RFC Editor upon publication]

[The PATH-CHARACTERISTIC attribute requires that IANA allocate a value in the "STUN attributes Registry" from the comprehension-optional range (0x8000-0xFFFF), to be replaced for TBD-CA throughout this document]

This document defines the PATH-CHARACTERISTIC STUN attribute, described in Section 3. IANA has allocated the comprehension-optional codepoint TBD-CA for this attribute.

6. Security Considerations

Security considerations discussed in [RFC5389] are to be taken into account. STUN requires the 96 bits transaction ID to be uniformly and randomly chosen from the interval 0 .. 2**96-1, and be cryptographically strong. This is good enough security against an off-path attacker. An on-path attacker can either inject a fake response or modify the values in PATH-CHARACTERISTIC attribute to mislead the client and server, this attack can be mitigated using STUN authentication. As PATH-CHARACTERISTIC is expected to be used between peers using ICE, and ICE uses STUN short-term credential mechanism the risk of on-path attack influencing the messages is minimal. However, an attacker could corrupt, remove, or delay an ICE request or response, in order to discourage that path from being used. Unauthenticated STUN message MUST NOT include the PATH-

CHARACTERISTIC attribute in order to prevent on-path attacker from influencing decision-making.

7. Acknowledgements

Thanks to Brandon Williams, Simon Perreault, Aijun Wang for valuable inputs and comments.

8. References

8.1. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[RFC5245] Rosenberg, J., "Interactive Connectivity Establishment (ICE): A Protocol for Network Address Translator (NAT) Traversal for Offer/Answer Protocols", RFC 5245, April 2010.

[RFC5389] Rosenberg, J., Mahy, R., Matthews, P., and D. Wing, "Session Traversal Utilities for NAT (STUN)", RFC 5389, October 2008.

8.2. Informative References

[ACM-MPRTP] Singh, V., Ahsan, S., and J. Ott, "MPRTP: multipath considerations for real-time media", in Proc. of ACM Multimedia Systems, MMSys, 2013.

[I-D.ietf-avtcore-mprtp] Singh, V., Karkkainen, T., Ott, J., Ahsan, S., and L. Eggert, "Multipath RTP (MPRTP)", draft-ietf-avtcore-mprtp-00 (work in progress), December 2014.

[I-D.patil-tram-turn-serv-selection] Patil, P., Reddy, T., and G. Salgueiro, "Traversal Using Relays around NAT (TURN) Server Selection", draft-patil-tram-turn-serv-selection-00 (work in progress), October 2014.

Authors' Addresses

Tirumaleswar Reddy
Cisco Systems, Inc.
Cessna Business Park, Varthur Hobli
Sarjapur Marathalli Outer Ring Road
Bangalore, Karnataka 560103
India

Email: tireddy@cisco.com

Dan Wing
Cisco Systems, Inc.
170 West Tasman Drive
San Jose, California 95134
USA

Email: dwing@cisco.com

Paal-Erik Martinsen
Cisco Systems, Inc.
Philip Pedersens vei 22
Lysaker, Akershus 1325
Norway

Email: palmarti@cisco.com

Varun Singh
Nemu Dialogue System Oy
Espoo 02235
Finland

Email: varun@callstats.io

Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: September 6, 2015

B. Schwartz
J. Uberti
Google

March 5, 2015

TURN by name: an extension to TURN for contacting an endpoint by its DNS name.
draft-schwartz-tram-turnbyname-00

Abstract

When tunneling traffic through TURN, a client may sometimes desire to contact a remote endpoint that it knows by its DNS name, not its IP address. This document describes an extension to TURN that allows such a client to contact a named endpoint.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at http://datatracker.ietf.org/drafts/current/.

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."

This Internet-Draft will expire on September 6, 2015.

Copyright Notice

Table of Contents

1. Introduction

The TURN standard [RFC5766] extends STUN to allow proxying connections directly through the server. Clients send messages to the server in order to request the allocation of ports on the server, and identify the remote peers with whom they want to exchange packets. These remote peers are identified by an XOR-PEER-ADDRESS attribute, which includes the remote peer's IP address and port.

TURN is most commonly used as a component of an ICE [RFC5245] implementation, to allow communication between endpoints that each send their own transport address to the other in the form of an ICE candidate. These candidates are typically constructed using STUN, or

by direct interrogation of the network interfaces, so they normally contain IP addresses, although domain names are also allowed.

However, TURN is now attracting a wider range of use cases, especially on enterprise networks and in conjunction with WebRTC. Some use cases employ TURN as an "escape hatch" in an otherwise tightly restricted network, with the intention that users would tunnel much or all of their UDP traffic through the TURN server. On some restricted networks, DNS access is also restricted, which may prevent users from determining the IP address of a domain (e.g. an application-specified TURN server, for RETURN [I-D.schwartz-rtcweb-return], or an ICE candidate that contains a domain name) that they wish to contact through the escape-hatch TURN server.

Extending TURN to support named peers allows TURN to work for clients who are attempting to contact an endpoint by name, on networks where resolving those names is not otherwise possible.

2. New Address Family for DNS names

The Address Family 0x03 is defined to indicate that the specified address is a DNS name. This family is only permitted under certain circumstances, detailed below.

3. Extension to the XOR-PEER-ADDRESS attribute format

STUN/TURN attributes are Type-Length-Value encoded ([RFC5389], Section 15). For both of the existing Families, the attribute's encoded length is a known constant, because the length of the address is constant. For the newly defined Family 0x03, the length is variable, and the indicated length from the TLV encoding is necessary in order to parse the attribute.

The DNS name is transmitted in the X-Address field, and is encoded by the following procedure:

1. Define the "legacy transaction ID" as a 128-bit value consisting of the 32-bit magic cookie followed by the 96-bit transaction ID.

2. Define the DNS name as a standard dot-separated UTF-8 byte-string (not null-terminated).

3. Compute the encoded address (X-Address) by XOR'ing each byte of the DNS name with the corresponding byte of the legacy transaction ID.

* If the DNS name is longer than 128 bits, the corresponding byte with which to XOR wraps around to the beginning of the legacy transaction ID.

This procedure is an extension of the encoding for families 0x01 and 0x02, so all three XOR-PEER-ADDRESS families can be encoded and parsed by a single procedure, without any special cases.

4. Changes to TURN server behavior

4.1. Servers that do not support the extension

Servers are NOT REQUIRED to support this extension. No change is required to servers that do not support the extension. Upon receiving a message containing an XOR-PEER-ADDRESS attribute with Family 0x03, existing compliant servers MUST reply with Error 440 (Address Family not Supported).

Servers that do support this extension MUST comply with the requirements that follow in this section.

4.2. Supported attributes

Address Family 0x03 is only permitted in the context of the XOR-PEER-ADDRESS attribute. All other attributes that use address families remain restricted to families 0x01 and 0x02. The server MUST respond with Error 440 (Address Family not Supported) when encountering this address family in an attribute where it is not supported.

4.3. Supported messages

The XOR-PEER-ADDRESS attribute may only have family 0x03 in the context of a CreatePermission, Send, Data, or ChannelBind message. If the server encounters this address family in the context of any other message type, it MUST respond with Error 440 (Address Family not Supported).

If a TURN server supports address family 0x03 in one of these messages, it MUST support it in all of these messages.

4.4. Name mapping storage

Baseline TURN servers must store two kinds of state for each Allocation: Permissions and Channel Bindings. This extension adds a third kind of state: Name Mappings. Each DNS Name Mapping consists of:

o a DNS name

- an IP address
- a reference count, which is always either 1 or 2

Name Mappings do not have an expiration time, but the server MUST delete them if their reference count falls to zero. Like Permissions and Channel Bindings, Name Mappings are scoped to a single Allocation.

Each IP address appears in only one Name Mapping for an Allocation. The requirements for CreatePermission and ChannelBind are structured to maintain this invariant.

Server implementations SHOULD implement Name Mappings in a way that enables fast bidirectional lookup.

4.5. Lookup behavior

When a DNS name lookup is required, the server's behavior depends on the current Allocation. Each supported message is associated with an Allocation, whose address family is IPv4, IPv6 [RFC6156], or Both (via ADDITIONAL-ADDRESS-FAMILY [I-D.ietf-tram-turnbis]).

If the address family is IPv4, then the server MUST search for an A record for the name, and similarly if the address family is IPv6, the server MUST search for a AAAA record. The server MUST handle errors as follows:

- If resolution fails due to a server error (e.g. DNS SERVFAIL), reply with error code 500 (Server Error).
- If the resolution fails because there is no record of the required type (e.g. DNS NOERROR), respond with error code 443 (Peer Address Family Mismatch).
- For all other DNS errors, return error code 447 (Connection timeout or failure).

The TURN server implementation MAY use a high-level DNS resolution API, such as gethostbyname or getaddrinfo, to perform the lookup.

If the Allocation has both address families, then it MUST look for an IPv6 address, and fall back to IPv4 only if a AAAA record is not found.

4.6. CreatePermission

In baseline TURN, each CreatePermission message creates or renews a Permission to send and receive messages to some specified IP address. With this extension, a Permission may indicate either an IP address or a DNS name. Both types of Permissions are subject to the same expiration policy.

At any given time, there is at most one Permission that specifies any IP address, or any DNS name, but there may be a Permission specifying a DNS name that resolves to an IP address that is specified in another Permission.

Upon receiving a CreatePermission message on an Allocation, the server MUST perform these steps:

1. If the CreatePermission message contains a peer address of family 0x01 or 0x02, create or update a Permission for the given address. (No change from baseline.)

2. If the CreatePermission message contains peer address of family 0x03:

    A. Look for an existing Permission with the given DNS name. If one exists, refresh its expiration time and return success.

    B. Otherwise, check if there is a Name Mapping for the DNS name.

        i. If one exists, increment its reference count.

        ii. Otherwise, perform a DNS lookup for the name. If it succeeds, add a DNS name mapping for the name and the resolved address, with reference count 1.

    C. Install a new permission for the DNS name.

When a Permission containing a DNS name expires, the server MUST decrement the reference count on the Name Mapping for this DNS name, and delete the Name Mapping if its reference count falls to zero.

4.6.1. Implications of this permission model

As long as a permission is regularly refreshed with the same DNS name, the effective IP address will not change.

Permission refreshes for an IP address do not extend the lifetime of DNS resolutions to that address.

Permission requests for an IP address are not sufficient to allow Send requests to a DNS name that resolves to that IP address, and vice versa.

4.7. Send

Upon receiving a Send message on an Allocation, the server MUST perform these steps:

1. If the Send message contains a peer address of family 0x01 or 0x02, check for a Permission that indicates that IP address. (There will be at most one.) If a Permission matches, send the packet; otherwise silently drop it. (No change from baseline.)

2. If the Send message contains a peer address of family 0x03, check if there is a Permission for the given DNS name. (There will be at most one.) If one exists, send the packet to the IP address indicated for that DNS name in its Name Mapping; otherwise silently drop it.

4.8. Channel Binding

In baseline TURN, each ChannelBind message creates or renews a channel binding, which consists of a transport ID, a peer's IP address, and a port on that address. It also creates or renews a permission for the peer's IP address, exactly as if a CreatePermission message had been received for that IP address.

In this extension, each channel binding includes either an IP address or a DNS name.

Upon receiving a ChannelBind message on an Allocation, the server MUST perform these steps:

1. If the ChannelBind indicates a peer address of family 0x01 or 0x02

   A. If a binding already exists with the specified transport ID, IP address, and port, refresh the binding.

   B. If a binding already exists for the specified transport ID with a different or unspecified IP address or port, report Error 400 (Bad Request).

   C. If a binding already exists with this port and this IP address, or a DNS name that maps to this IP address, report Error 400 (Bad Request) and include a CHANNEL-NUMBER

attribute that indicates the number of the conflicting channel.

D. Otherwise, create a binding.

E. Install or refresh a permission for the originally indicated peer IP address.

2. If the ChannelBind indicates a peer address of family 0x03

A. If a binding already exists with the specified transport ID, DNS name, and port, refresh the binding, including the IP address.

B. Otherwise, resolve the DNS name to an IP address, using the name mapping table if it exists, and performing a DNS lookup only if no name mapping exists for this DNS name.

i. If a binding already exists for the specified transport ID with a different IP address or port, report Error 400 (Bad Request).

ii. If a binding already exists with a different transport ID, for this port, and this IP address or a DNS name that is mapped to this IP address, report Error 400 (Bad Request) and include a CHANNEL-NUMBER attribute that indicates the number of the conflicting channel.

C. Install a channel binding with the specified transport ID, DNS name, and port.

D. Increment the name mapping's reference count, or Install a new name mapping if one does not already exist for this DNS name.

E. Perform the steps required when receiving a CreatePermission message for this DNS name.

When a channel binding that indicates a DNS name expires, the server MUST decrement the reference count on the matching name mapping, and delete the mapping if the reference count falls to zero.

4.8.1. Implications of this channel binding model

As long as a channel is refreshed before it times out, it will continue to resolve to a constant address.

There can never be two channels bound to the same remote transport address. If that were possible, it would result in traffic amplification (sending each received packet to all matching channels) or other strange behaviors (e.g. selecting one arbitrary channel to receive the packet).

Each time a new channel is bound for a DNS name, it checks for a Name Mapping before doing any external resolution, so the resolved IP address is guaranteed to be consistent with the active Permission for this DNS name, if one exists. As a result, DNS resolution results can persist indefinitely within an Allocation, longer than the DNS TTL or any individual connection, if they are maintained by ChannelBind or CreatePermission calls to different ports on the same remote peer that overlap in time.

If two ChannelBind requests are received for the same port on two different DNS names that resolve to the same IP address, the second request will fail with a generic error code (400), but will also let the client know which existing channel to use instead. The same is true of collisions between IP and DNS channel binding requests.

Installing a channel binding to a DNS name also enables Send messages to the DNS name, but not to the resolved IP address.

4.9. Receiving Data

Upon receiving an incoming packet on an Allocation, the server MUST perform these steps:

1. Check if there is a channel binding to this source port and IP address, or a DNS name that is mapped to this IP address. (There will be at most one such channel.) If there is, let the channel handle the packet.

2. Otherwise, check if there is any DNS permission that is mapped to the source IP address. If there is, produce a Data message with that DNS name.

3. Otherwise, check if there is any IP permission that matches the source IP address. If there is, produce a Data message with the source IP address; otherwise discard the packet.

4.9.1. Implications of this data receipt model

If a name mapping exists for an IP address, all packets received from that address will be labeled with the DNS name, not the IP address. Clients never learn the IP address for a DNS name unless they provoke a conflict, similar to the naming model used by SOCKS5 [RFC1928].

If a channel is bound for a port on a peer, all packets from that port will be routed to the channel exclusively.

5. Changes to TURN client behavior

Clients are NOT REQUIRED to support this extension. No change is required to existing clients. The requirements in this section only apply to clients that opt to support the extension.

5.1. When to use this extension

When the client receives a request to contact an endpoint that is identified by its DNS name, the client SHOULD attempt to use this extension to reach that endpoint, and SHOULD NOT attempt to perform a local DNS lookup for the name, so that connections may succeed even if the local DNS server fails to return a correct result.

If the TURN server responds with Error 440 (Address Family Not Supported), then the TURN client application SHOULD attempt to perform a local DNS lookup for the name, and retry the connection by IP address. (This functionality is logically separable from the TURN protocol itself, and might best be implemented by having a TURN client library that indicates the error, leaving the DNS lookup to be the responsibility of the application that uses the library.)

5.2. Issuing Send, CreatePermission, and ChannelBind requests for DNS names

When attempting to contact an endpoint by its DNS name, the client SHOULD transmit a CreatePermission or ChannelBind request whose XOR-PEER-ADDRESS attribute contains family 0x03, conveying the DNS name formatted as described above.

If the server responds with Error 440 (Address family not supported), then the client SHOULD abandon all requests using DNS, because the server does not support this extension.

If a ChannelBind request fails with Error 400, but includes a CHANNEL-NUMBER attribute, then that channel is already bound to the remote transport address.

5.3. Receiving Data indications

Clients MAY send CreatePermission requests for both an IP address and a DNS name that maps to that IP address, and both requests will succeed. However, all Data messages from the remote peer will be marked as being received from the DNS name. Therefore, clients MUST

NOT assume that replies from a Send to an IP address are labeled with that IP address.

5.4. Handling dynamic addressing

The IP address to which a DNS name resolves is not a constant. It may change occasionally due to address reassignment, or it may even change on every lookup, in the case of round-robin DNS.

The TURN server ensures that the IP address associated with a permission or channel binding does not change as long as the permission or binding is refreshed before it expires. Therefore, clients that need to send messages to a stable IP address MUST refresh their DNS name permissions and channel bindings even while they are not in use, to ensure that they do not expire and later resolve to a different IP address.

If the client has previously connected to a DNS name on an Allocation, and wishes to connect again to the same DNS name with an up-to-date IP address resolution, it SHOULD request a new Allocation, and connect to the DNS name on the new Allocation.

5.5. Dual-stack behavior

If a specific address family is not indicated for the remote endpoint, and the server does not support dual allocation (e.g. ADDITIONAL-ADDRESS-FAMILY [I-D.ietf-tram-turnbis]]), then the client's behavior is implementation-defined. For example, when processing a request to send the first packet to a DNS name, the client MAY use an approach inspired by Happy Eyeballs [RFC6555]:

- Create an Allocation for the system's preferred address family (e.g. IPv6).
- Attempt to connect to the DNS name on this Allocation using a ChannelBind message.
  - If the server replies with error code 443 (Peer Address Family Mismatch), immediately discard the Allocation and try again with an Allocation of the other family.
  - If a response message is received before some timeout (e.g. 300 ms), use this Allocation
  - If no response message is received before some timeout (e.g. 300 ms), attempt to connect using a new Allocation of the other address family, and use whichever Allocation receives a response first. Discard the other Allocation.

6. Examples

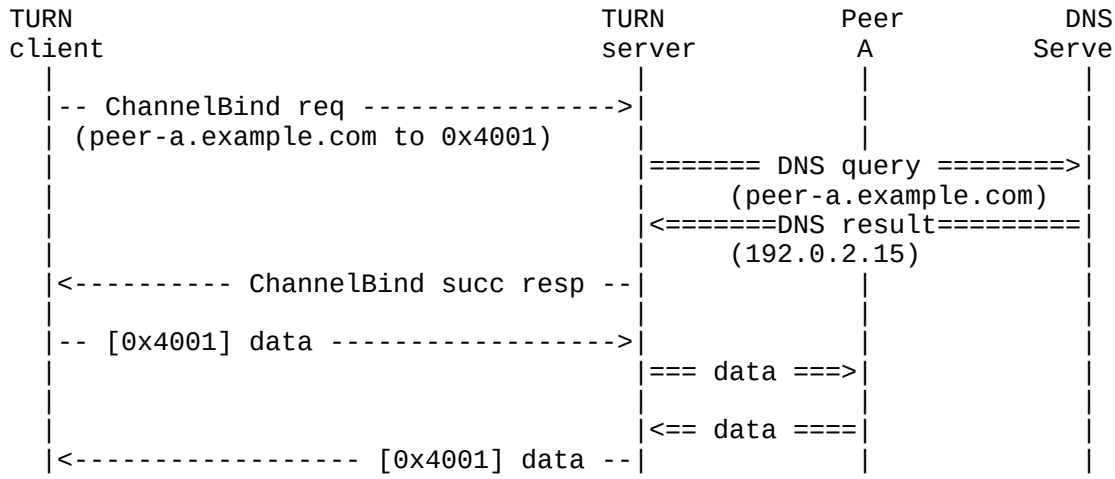


Figure 1: Using DNS names with ChannelBind

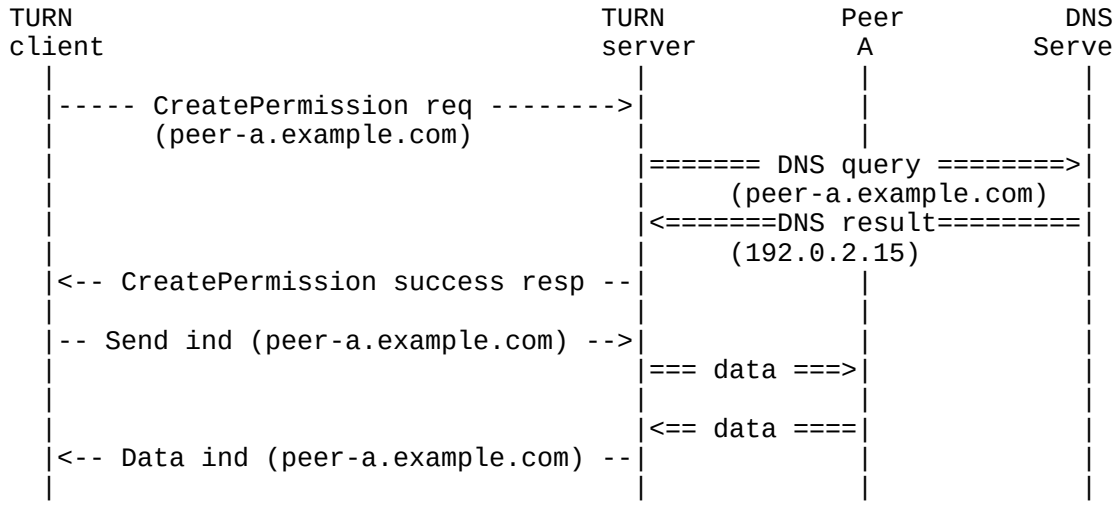


Figure 2: Using DNS names with CreatePermission and Send

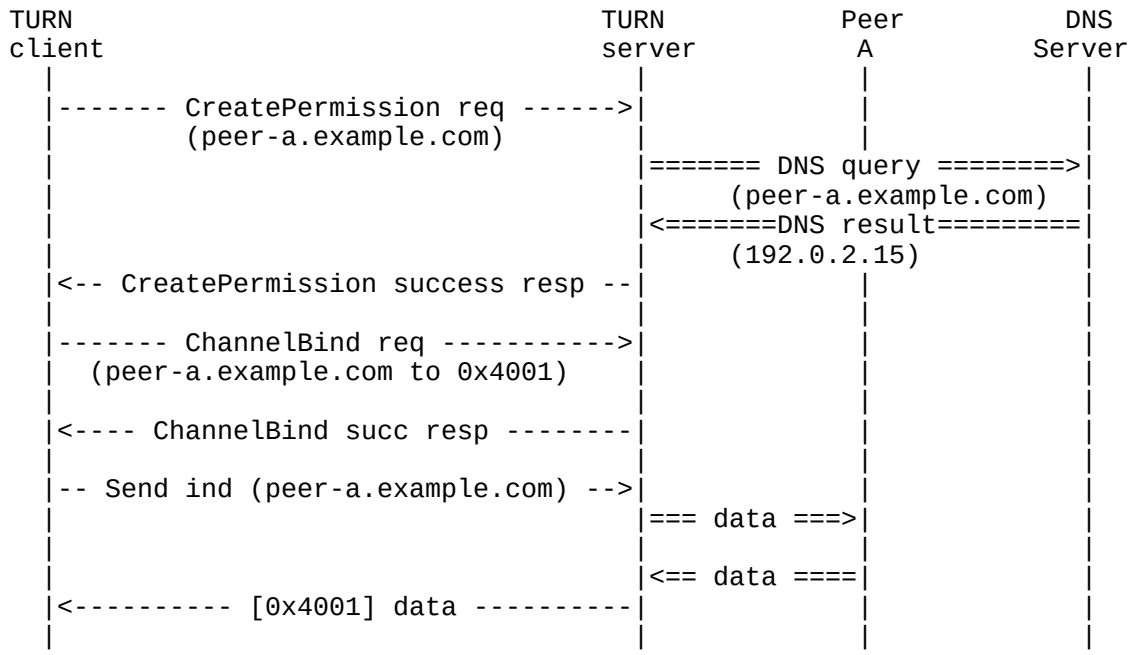


Figure 3: Sharing DNS names between CreatePermission and ChannelBind

7. Security Considerations

TURN servers that implement this specification can be made to parse arbitrary DNS records. They should make sure to use secure, well-tested DNS client implementations.

Clients can cause the TURN server to perform an arbitrary number of DNS lookups. Server implementations MAY limit the rate at which an individual client can trigger lookups, and return Error 508 (Insufficient Capacity) when a client exceeds the limit.

A malicious server could forward messages to the wrong IP address for a specified domain name, but this does not represent a change in security relative to the basic TURN standard.

To provide this functionality, the server is required to store a number of DNS Name Mappings that is at most the number of active permissions or channels. Implementers should take care to avoid resource leaks in the DNS mapping implementation, to maintain this bound.

8. IANA Considerations

This draft adds a new STUN address family, 0x03 (DNS name).

9. Acknowledgements

Thanks to Warren Kumari for his early review.

10. References

10.1. Normative References

[I-D.ietf-tram-turnbis]
Reddy, T., Johnston, A., Matthews, P., and J. Rosenberg, "Traversal Using Relays around NAT (TURN): Relay Extensions to Session Traversal Utilities for NAT (STUN)", draft-ietf-tram-turnbis-02 (work in progress), February 2015.

[RFC5245] Rosenberg, J., "Interactive Connectivity Establishment (ICE): A Protocol for Network Address Translator (NAT) Traversal for Offer/Answer Protocols", RFC 5245, April 2010.

[RFC5389] Rosenberg, J., Mahy, R., Matthews, P., and D. Wing, "Session Traversal Utilities for NAT (STUN)", RFC 5389, October 2008.

[RFC5766] Mahy, R., Matthews, P., and J. Rosenberg, "Traversal Using Relays around NAT (TURN): Relay Extensions to Session Traversal Utilities for NAT (STUN)", RFC 5766, April 2010.

[RFC6156] Camarillo, G., Novo, O., and S. Perreault, "Traversal Using Relays around NAT (TURN) Extension for IPv6", RFC 6156, April 2011.

10.2. Informative References

[I-D.schwartz-rtcweb-return]
Schwartz, B. and J. Uberti, "Recursively Encapsulated TURN (RETURN) for Connectivity and Privacy in WebRTC", draft-schwartz-return-04 (work in progress), November 2014.

[RFC1928] Leech, M., Ganis, M., Lee, Y., Kuris, R., Koblas, D., and L. Jones, "SOCKS Protocol Version 5", RFC 5766, March 1996.

[RFC6555] Wing, D. and A. Yourtchenko, "Happy Eyeballs: Success with Dual-Stack Hosts", RFC 6555, April 2012.

Authors' Addresses

Benjamin M. Schwartz
Google, Inc.
111 8th Ave
New York, NY 10011
USA

Email: bemasc@webrtc.org

Justin Uberti
Google, Inc.
747 6th Street South
Kirkland, WA 98033
USA

Email: justin@uberti.name

tram
Internet-Draft
Intended status: Standards Track
Expires: September 10, 2015

A. Wang
China Telecom
B. Liu
Huawei Technologies

March 9, 2015

A Lightweight TURN Architecture and Specification (TURN-Lite) draft-wang-tram-turnlite-02

Abstract

This document proposes a new relay-based NAT traversal architecture called TURN-Lite which could simplify the data communication process between two hosts that locates behind some non-BEHAVE compliant [RFC4787] [RFC5382] NAT devices. The key mechanism in TURN-Lite is the newly defined "Couple" operation (using STUN [RFC5389] message format) which allows the TURN-Lite servers to be easily incorporated into existing CGN devices/CDN nodes which are already deployed within the network in a distributed manner.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at http://datatracker.ietf.org/drafts/current/.

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."

This Internet-Draft will expire on September 10, 2015.

Copyright Notice

Table of Contents

1. Introduction

1.1. Motivations

This document proposes a new relay-based NAT traversal architecture called TURN-Lite based on the following motivations.

1) Leverage ISPs' infrastructures

Currently, the deployment of TURN [RFC5766] is very limited and most of the application providers use their own platform to transfer the data between two hosts that behind NATs and to translate the

communication packets between two hosts in different address families.

The relay devices deployed centrally by various application providers often lead to inefficient data transmit between two hosts. The relay devices must deal with complex network layer problems which the application providers are not familiar with.

On the other hand, service providers have deployed many CGN devices/ CDN nodes in a distributed manner within their networks. If the service providers can use these CGN devices/CDN nodes as the relay devices for communication between two hosts behind NATs or that from different address family, and open their data translation/forwarding capability to the application providers, the host to host communication will be more efficient. Given most of the CGNs are capable of translating packets between IPv4 and IPv6, the adoption of IPv6 technology will also be accelerated.

2) Simplify the communication procedures

TURN-Lite needs less communication procedures than TURN of which the procedures are considered very complex to be integrated into the ISPs' infrastructure, for example:

o TURN solution has to closely interact with ICE

Within current TURN solution, there are scenarios where the ICE or other NAT-hole punching procedures must be included for the success of communication via TURN servers. The key point is that TURN allocates different relay transport address-port pairs for different hosts.

Each client must first use TURN allocation request to get their transport relay address-port pairs, and then must use ICE procedure (connectivity check) or other similar signaling to punch holes for these transport relay addresses on the alongside NAT devices. Or else the relayed UDP/TCP packet will be blocked.

Even with the above procedures, there still exists some risks that the packets be rejected by TURN server due to the permission list that created by client via "CreatePermission Request" before it sending data to the peer. In order to mitigate such issues, current TURN solution requires the TURN servers only check the IP address part of the relay transport address, and ignore the port portion. But this will again introduce some attack risks because different host may share

one public IP address when the CGN device is deployed within network.

- IPv4/IPv6 Relay Address/Port Reservation and Allocation

  Another drawback of different relay transport addresses for different host is that the TURN server must reserve some IPv4/IPv6 address block for the allocation and plan the TCP/UDP port usage for each host. When TURN servers are deployed in a distribute manner (For example when they are incorporated into the CGN devices), there will be much coordination work to do for the address/port reservation and allocation on the TURN servers.

- Simultaneous TCP/UDP connections burden on TURN server

  Current TURN solution requires the TURN servers to open and listen on many TCP/UDP ports at the same time, and the number of sessions is proportional to the number of the hosts under one relay server.(Under TURN solution for TCP [RFC6062], each host requires two connections to the TURN server). This will increase the burden on TURN server and the complexity to incorporate them into the CGN/CDN devices.

- Different procedures for TCP/UDP communication

  Current TURN solution adopts different procedures for the TCP and UDP communication channel. So for one TURN server to provide the TCP/UDP relay function, it must implement two different procedures. This again increases the complexity of the TURN server implementation, especially in CGN devices.

- Communication complexity between two different TURN servers

  Current TURN solution cannot assure two hosts select the same TURN server, and then it must deal with the communication situation between two different TURN servers. This scenario has not been covered by the current TURN related drafts. The communication complexity of different TURN server arises from the communication procedures, the transfer of the relay addresses of the two TURN servers and the permission list creation etc.

  On the other hand, because the hosts select their own TURN server, there is no mechanism to assure the relayed path is most optimal for them. The application latency will be increased when this occurs.

TURN-Lite architecture and solution will simplify the above mentioned complexity and make the TCP/UDP data relay function be easily incorporated into the existing distributed CGN devices or other kinds distributed devices i.e. the CDN nodes etc.

1.2. Relationship with TURN

This document doesn't intent to replace TURN with TURN-Lite, but consider TURN-Lite as a complementary solution along with TURN for some specific scenarios.

If one SP wants to open its infrastructure to accelerate their customers' (mainly regarding to application providers) client-to-client communications within the SP's domain, TURN-Lite could be a good candidate.

2. Requirements Language and Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119] when they appear in ALL CAPS. When these words are not in ALL CAPS (such as "should" or "Should"), they have their usual English meanings, and are not to be interpreted as [RFC2119] key words.

- Application Provider: the service providers who provide client to client communications through the Internet. E.g. VoIP service providers, instant message service providers etc.

- Relay Selector: which is in charge of selecting a proper relay device (CGN or CDN nodes) for the communicating hosts behind NATs. Normally, the RS is a function located in the network's management plane and possibly a part of the NMS server

- TURN-Lite: lightweight TURN architecture. The word "lightweight" is in the perspective of an application provider.

- TURN-Lite Client: the TURN-Lite entity that deployed in the application providers' networks; be responsible for TURN-Lite signaling/control interactions with the TURN-Lite servers.

- TURN-Lite Server: the TURN-Lite entity that deployed in the ISP's networks; be mainly responsible for the data relay between an application providers' clients. Normally, the TURN-Lite servers collated with the CGNs (Carrier Grade NATs) within the service provider.

3. Solution Overview

3.1. Reference Architecture of TURN-Lite

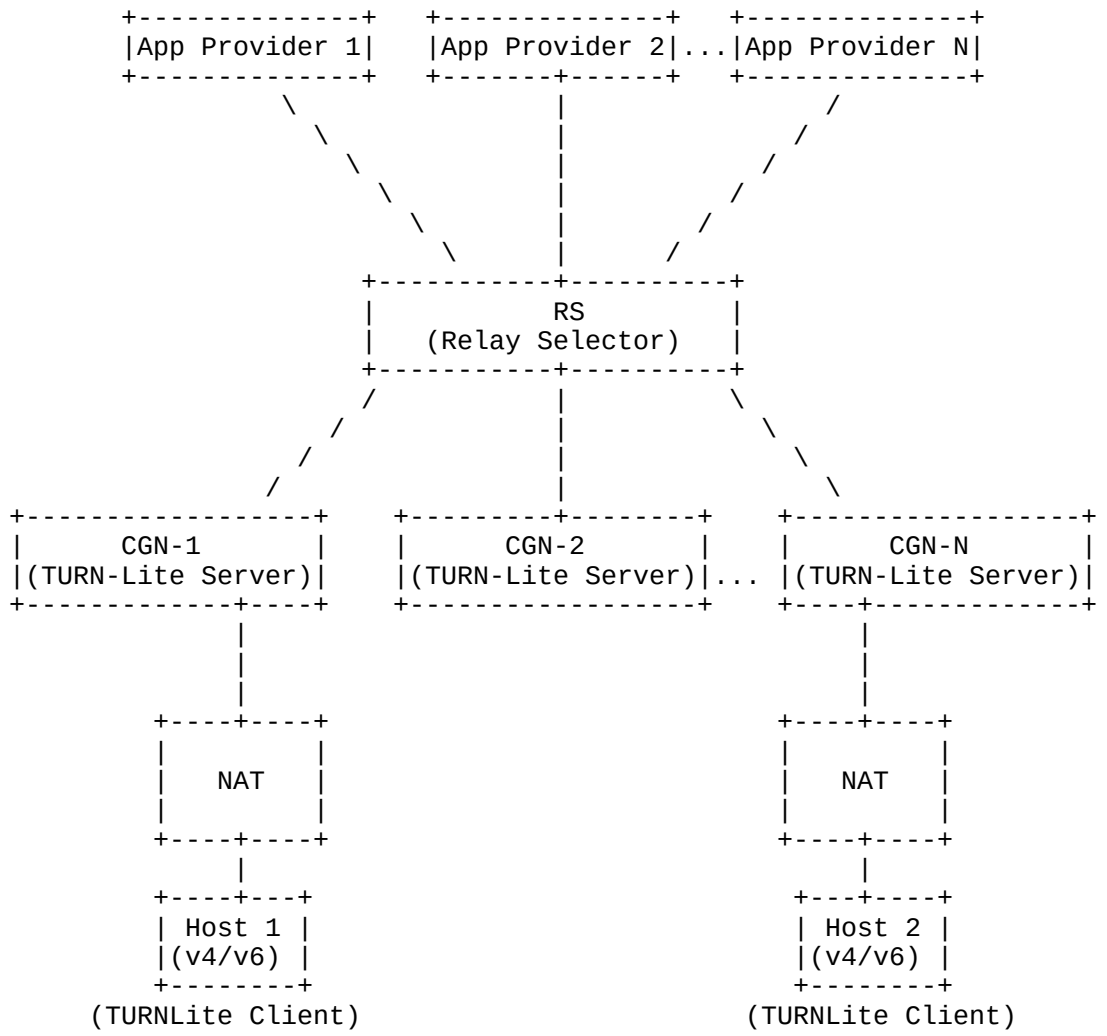


Fig. 3-1: TURN-Lite Arhitecture

As depicted in above figure, the application clients that located on hosts act as the TURN-Lite clients while the CGNs act as TURN-Lite Servers. There is a Relay Selector (RS) for choosing a proper CGN to relay traffic between the two hosts. In practice, the RS could be a dedicated server or a function located in the management plane servers such as NMS server.

RS has the intelligent route selection capability to choose a proper CGN for a given host pair. RS sends the data relay indication to the selected CGN devices/CDN node via the newly defined "Couple" method.

BEHAVE compliant and non-BEHAVE compliant NAT traverse [RFC4787] [RFC5382] is supported in TURN-Lite. IPv6 and IPv4 host communication is also supported.

3.2. Solution Rationale

The solution could be briefly described in the following sections.

3.2.1. Relay Selector Reflection and Selection

Each host that wants to communicate with the other host should send STUN message to the RS (Relay Selector), and get their reflex addresses to the RS (here we refer to REFLX-RS).

The application provider needs to select a suitable RS and informs it to the hosts (e.g. via application specific client-server protocol). The detailed RS selection mechanism and criteria are out of the scope of this document, but some general considerations are as the following.

- If the hosts locate in the same ISP/administrative domain, then the RS selection is fairly easy since normally there is only one RS in one ISP; even there are multiple RSes in one ISP, the application provider should also be able to select a suitable RS based on the addresses of the two hosts.

- If the hosts locate in two ISPs/administrative domains (assuming both of the ISPs providing TURN-Lite service to the application provider), the application provider can select one RS based on pre-defined policies (the simplest way is just arbitrarily choosing one RS in one of the ISPs).

3.2.2. Relay Selection

Each host will report its REFLX-RS address to its application server. If two hosts want to communicate, the application server will send the two hosts' REFLX-RS addresses to the selected RS, to let the RS select one appropriate relay device to relay the traffic.

Generally, the RS can select the appropriate relay device based solely on the REFLX-RS addresses of these two hosts, for example, select one relay device that locate in the middle of the communication path. This approach is possible since the relay

behavior is within one ISP's domain that the RS could be possible to learn the knowledge of all CGNs (relays) within that domain.

The selection criteria can also be expanded to include other factors, such as the privacy concern of the communication peers, the linkage usage information between the host and the relay device etc. For example, RS can select one relay device that locates far from the communication peer to hide the location of the peer. This might sacrifice the communication efficiency but increase the protection of the host privacy. Anyway, RS has more flexible control over the relay selection, upon the requirement of communication hosts, or the consideration of relay service provider.

After the relay device selection, the RS will respond the IP address of the selected relay device to the communication peer, together with the well known port that used by every relay device. The combination of this relay IP address and the well-known port form the relay transport address of the communication peers, each peer will use this relay transport address to communicate.

When two hosts located within one administration domain, the centralized relay point selection and control architecture can easily achieve one low latency communication path because it knows the whole network condition of its own. When two hosts located within different administration domains, the TURN-Lite solution will also work except that some end-to-end communication efficiency might be sacrificed unless there is some coordination between these two administration domains.

3.2.3. Forming "Couple" Command

Each host will send again one STUN message to the selected relay transport address, get the new reflex address(here we refer to REFLX-Relay) to the selected relay device, and reports it to the RS, together with the previous reflex address to the RS (which is REFLX-RS).

The RS will use the REFLX-RS addresses to find out which two peers will communicate (such communication pair information is gotten from Section 3.2.2). RS will retrieve the corresponding REFLX-Relay address of the communication peer, forms the "Couple" command based on such information, and sends the "Couple" command to the selected relay transport address.

Upon receiving the "Couple" command, the relay device will add one item to its forwarding table. The forwarding table will bind the reflex addresses of the two peers, the required transport protocol and other additional information.

3.2.4. Data Relay

Each host will then send the data traffic directly to the unique relay transport address. The source address of this packet will be changed by the alongside NAT devices that located between the host and the relay device.

When this packet arrives to the relay address, its source address will be one of the RFLEX-Relay addresses. The relay device will search the forwarding table that formed in Section 3.2.3. If the REFLX-Relay address in one item match the source address of the received packet, then the other REFLX-Relay address will be retrieved and be used as the destination address of the application packet, the packet's source address will be changed to the relay transport address.

After the conversion, the packet will be sent by the relay device. This packet will be routed to the corresponding peer, according to its REFLX-Relay address.

The application return packet will be sent again back to the same relay device via the relay transport address. The similar search process and convert work will be done by the relay device. The converted return packet will then be routed to the packet originator.

4. New STUN Method Definition

In order to let the CGN device to build one Couple item upon the request of RS, it is needed to define one general Couple message to transfer the related information.

4.1. Couple Operation

The Couple request defines the relationship between two TCP or UDP half-connections, the translation rule that converts both the source address and destination address of pass through packet in both directions.

Couple Opcode: It defines how to bind two half-connections that ends at the CGN's well-known relay transport address together. When CGN device receives the Couple request, it will create one map table item that includes the reflex IP address/port [REFLX-Relay] of both hosts that lies behind the NAT device and the protocol that the host will use to communicate.

When the CGN device receives the packet from one host, it will use the reflex source address/port to lookup the map table item; converts the source address/port of this packet to the relay transport address

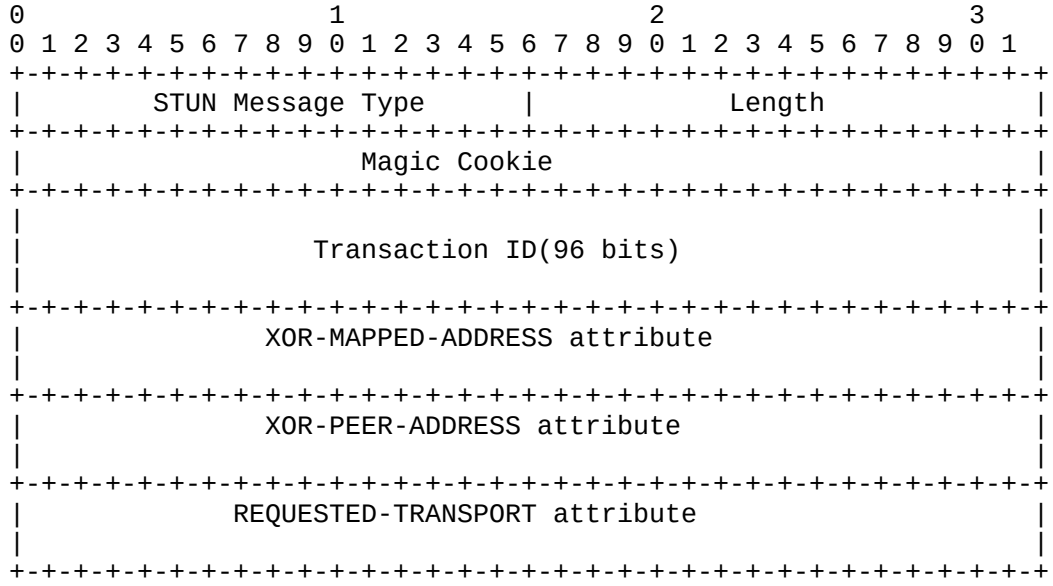
of the CGN device and converts the destination address/port of this packet to the reflex address [REFLX-Relay] that results from the map table lookup action.

The converted packet will be routed to NAT side of the other host, converted by the NAT device and then to the other host. The return packet will be delivered to the relay transport address of CGN/CDN device and be converted in reverse accordingly.

4.2. Couple Operation Packet Format

The Couple Opcode allows RS to create a new explicit couple table on the CGN device(TURN-Lite Server), instructs the CGN device to accomplish the related translation work.

The following diagram shows the Opcode layout for the Couple Opcode request/response format.



STUN Message Type — Couple method: value TBD. only request/response semantics

Decouple method: value TBD. only request/response semantics

Length — The same definition as STUN prococol [RFC5389]

Magic Cookie	The same definition as STUN prococol [RFC5389]
Transaction ID	The same definition as STUN prococol [RFC5389]
XOR-MAPPED-ADDRESS	The same definition as STUN prococol [RFC5389]. The value should be the RFLX-Relay address of the host.
XOR-PEER-ADDRESS	The same definition as TURN prococol [RFC5766]. The value should be the RFLX-Relay address of the peer.
REQUESTED-TRANSPORT	The same definition as TURN prococol [RFC5766]. the value of the "protocol" fiel should be TCP or UDP.

Fig.4-1: Couple Opcode Request/Response Format

5. Detailed Example

5.1. Procedures of Communication Traversing Symmetric NATs

When one of the communication hosts located behind the symmetric NAT device, the host-to-host communication must via one relay device. Below are the key procedures of TURN-Lite solution, we use the Fig 3-1 to describe the example.

Please note the communication procedures between the hosts and the application server are out of the scope of this document, we only focus on the key procedure proposed by this document.

1. If Host 1 and Host 2 want to communicate with each other, they will send STUN binding message to the RS IPv4 address/port, get their reflex address to RS[REFLX-RS].

2. RS will select one CGN device to relay the packet, based on the RS addresses information of the two peers. Here we assume it select CGN-1 as the relay device. RS will notify Host 1 and Host 2 of their relay transport address, both will use the same relay IP address/port on CGN-1.

3. Host 1 and Host 2 will send STUN binding message to CGN-1, get their relay address to CGN-1[REFLX-Relay] and report them to RS, together with RS addresses gotten in step 1). Here we assume the [REFLX-Relay] address of Host 1 is 192.0.2.1:7000, and [REFLX-Relay] address of Host 2 is 192.0.2.150:32102.

4. RS will form the "Couple" message, which include mainly the [REFLX-Relay] addresses of Host 1 and Host 2 and their communication protocol, here we assume they use TCP to communicate.

5. Upon receiving the "Couple" message, the CGN-1 device will form one forwarding table that look like below:

Reflextive transport address of Host1	Reflextive transport address of Host2	Transport Protocol
192.0.2.1:7000	192.0.2.150:32102	TCP

Table 5-1: Couple Table Example (symmetric case)

6. Host1 will send the application data to the relay transport address in CGN-1.

7. CGN device will look up the Couple table, use the source address of received packet(192.0.2.1:7000 in this example) to get the reflex IPv4 address of Host 2.

8. It then will change the source address of the packet to the relay transport address in CGN device, the destination address of this packet to the IPv4 reflex address of Host 2. The translated packet will be forwarded to Host 2.

9. The return traffic will also be sent to the same relay transport address in CGN-1, converted by the CGN device, and sent back to Host 1.

5.2. Procedures of IPv4 and IPv6 Host Communication

If Host 1 is one IPv4 node and Host 2 is one IPv6 node. The communication process are similar, except the relay address that is sent to the Host 1 is the IPv4 address of the CGN-1 and the relay address that is sent to the Host 2 is the IPv6 address of the CGN-1. Host 1 and Host 2 will not notice that they are talking to one node that in different address family.

The relay device selection process is same as the above example. Here we describe the procedure from step 3.

3. Host 1 and Host 2 will send STUN binding message to CGN-1, get their relay address to CGN-1[REFLX-Relay] and report them to RS, together with RS addresses gotten in step 1). Here we assume the

[REFLX-Relay] address of Host 1 is 192.0.2.1:7000, and [REFLX-Relay] address of Host 2 is 2001:c68:300:105::1[49191].

4. RS will form the "Couple" message, which include mainly the [REFLX-Relay] addresses of Host 1 and Host 2 and their communication protocol, here we assume they use TCP to communicate.

5. Upon receiving the "Couple" message, the CGN-1 device will form one forwarding table that look like below:

Reflextive transport address of Host1	Reflextive transport address of Host2	Transport Protocol
192.0.2.1:7000	2001:c68:300:105::1[49191]	UDP

Table 5-2: Couple Table Example (different address families case)

6. Host1 will send the application data to the relay transport address in CGN-1.

7. CGN device will look up the Couple table, use the source address of received packet(192.0.2.1:7000 in this example) to get the reflex IPv6 address of Host 2.

8. It then will change the source address of the packet to the relay transport IPv6 address in CGN device, the destination address of this packet to the IPv6 reflex address of Host 2. The translated packet will be forwarded to Host 2.

9. The return traffic will also be sent to the same relay transport address in CGN-1, converted by the CGN device, and sent back to Host 1.

6. TURN-Lite Benefits

TURN-LiteTURN-LiteComparing to TURN, TURN-Lite could provide following benefits:

- Decoupled from ICE

  TURN is tightly coupled with ICE. Operations like NAT punching, create permission .etc all require ICE connectivity check packets.

Benefited from the couple operation, TURN-Lite doesn't necessarily need ICE interaction.

- Avoid the Create Permission issues in TURN

  In the TURN-Lite solution, each communication pair will use the same relay server and the same relay address. The relay permission action required by TURN solution is replaced with the "Couple" command. There is no ambiguity for the relay permission because "Couple" command use the ip address and port information of the communication pair simultaneously. There are also no possible attacks due to the loose control of the current TURN permission treaments.

- Less Relay Address/Port Consumption and Management

  TURN-LiteTURN-Lite doesn't need to allocate different address-port pair for each session initiated from the hosts. Thus, it could obviously reduce the resource consumption and the human burden for planning the resource allocation.

- Simplified Procedures

  Theoretically, it requires only two commands to accomplish the relay function, compared with over eight commands that required by TURN solution. Due to every host communicate with the well-known relay address, there is no additional requirement for punching holes in the NAT devices, which is indispensable for the current TURN solution.

	TURN Solution	TURN-Lite Solution
Required Commands	1. Binding 2. Allocate 3. Send 4. Data 5. Channel Bind 6. Connect 7. ConnectionBind 8. ConnectionAttempt	1. Binding 2. Couple

Table 6-1: Procedures comparison between TURN and TURN-Lite

- Unified solution for TCP/UDP and IPv4(6)-IPv6(4) data relay

TURN-Lite supports the data relay for the communication between two hosts, uses same mechanism for TCP and UDP transport protocol, even for the communication between different address families.

- o Support for optimal relay selection

  Because of the central deployed RS could have the whole network's routing/path knowledge, TURN-Lite is more likely to achieve an optimal relay (TURN-Lite server) selection based on various information such as the reflective transport addresses of the two communicating peers, the link usage information between two peers and the load status of the candidate TURN-Lite servers etc.

  With the RS's knowledge, TURN-Lite is also more likely to achieve better relay selection for some specific requirements. For example, if one peer wants to hide its ip address to protect its privacy, the RS in TURN-Lite architecture could possibly select one TURN-Lite server that located far away from the host.

7. TURN-Lite Deployment Considerations

The TURN-Lite Server can be deployed in distributed manner. The most appropriate devices for incorporating this function are the CGN devices that have been deployed distributed by the service provider. Each distributed TURN-Lite Server has one unique public IPv4/IPv6 transport address.

The RS can select the appropriate TURN-Lite Server based on the proximity of the TURN-Lite server with the communication hosts and can also use other criteria to influence the selection procedure, as described in Section 3.

8. Security Considerations

The additional requirement of TURN-Lite is authenticating the couple operation from the RS. When the communication channel between the RS and the TURN-Lite server is secured, such security risks can be mitigated accordingly.

9. IANA Considerations

This draft requires IANA to allocate following STUN methods:

Couple: value TBD.

Decouple: value TBD.

10. Conclusions

Currently, the communication between two hosts that located behind NAT devices, especially that behind the symmetric NAT devices is emerging. With the development of IPv6 technology, the communication between two hosts that in different address families needs also be considered. Under the TURN-Lite architecture, the communication requests for IPv4/IPv4, IPv4/IPv6 scenario can be met in one general solution. Such solution can alleviate the burden of various CP/SP to deploy the TURN server by themselves, exploit and open the capabilities of CGN device that deployed by service provider to the third party(CP/SP), make the host-to-host communication more efficient.

11. Acknowledgements

Many valuable comments were received from Brandon Williams, Oleg Moskalenko, Jonathan Rosenberg, and Dan Wing etc.

This document was produced using the xml2rfc tool [RFC2629].

12. References

12.1. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[RFC2629] Rose, M., "Writing I-Ds and RFCs using XML", RFC 2629, June 1999.

[RFC5389] Rosenberg, J., Mahy, R., Matthews, P., and D. Wing, "Session Traversal Utilities for NAT (STUN)", RFC 5389, October 2008.

[RFC5766] Mahy, R., Matthews, P., and J. Rosenberg, "Traversal Using Relays around NAT (TURN): Relay Extensions to Session Traversal Utilities for NAT (STUN)", RFC 5766, April 2010.

12.2. Informative References

[RFC4787] Audet, F. and C. Jennings, "Network Address Translation (NAT) Behavioral Requirements for Unicast UDP", BCP 127, RFC 4787, January 2007.

[RFC5382] Guha, S., Biswas, K., Ford, B., Sivakumar, S., and P. Srisuresh, "NAT Behavioral Requirements for TCP", BCP 142, RFC 5382, October 2008.

[RFC6062] Perreault, S. and J. Rosenberg, "Traversal Using Relays around NAT (TURN) Extensions for TCP Allocations", RFC 6062, November 2010.

Authors' Addresses

Aijun Wang
China Telecom
No.118,Xizhimenneidajie,Xicheng District
Beijing, 100035
P.R. China

Email: wangaj@ctbri.com.cn

Bing Liu
Huawei Technologies
Q14, Huawei Campus, No.156 Beiqing Road, Hai-Dian District
Beijing, 100095
P.R. China

Email: leo.liubing@huawei.com

Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: July 3, 2015

B. Williams
Akamai
T. Reddy
Cisco

December 30, 2014

Peer-specific Redirection for Traversal Using Relays around NAT (TURN)
draft-williams-peer-redirect-03

Abstract

This specification describes a peer-specific redirection method that allows the TURN server to redirect a client for the purpose of improving communication with a specific peer without negatively affecting communication with other peers.

Status of this Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at http://datatracker.ietf.org/drafts/current/.

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."

This Internet-Draft will expire on July 3, 2015.

Copyright Notice

Table of Contents

1. Introduction

A Traversal Using Relay around NAT (TURN) [RFC5766] service provider may provide multiple candidate TURN servers for use by a host, but it might not be possible to determine which candidate TURN server will provide the best performance until both peers have been identified. This could be true for a variety of reasons, including:

- Using the selected relay for a specific peer results in a sub-optimal end-to-end Internet path.
- Load conditions on the selected relay have changed since the allocation was established such that it cannot support the new data flow.

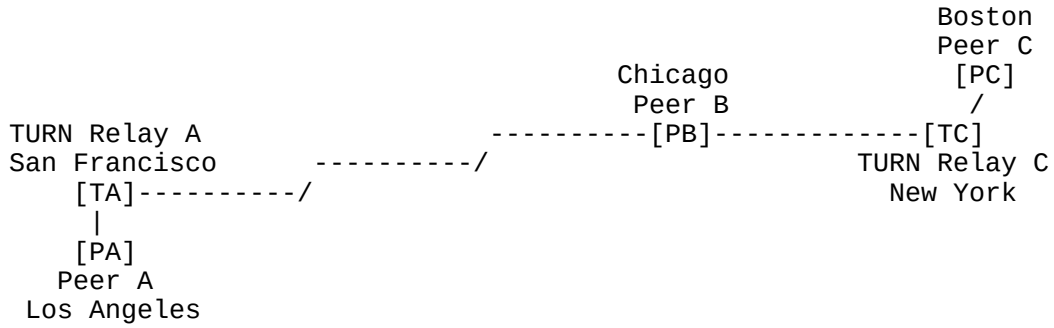
At the same time, the above conditions might apply to one peer but not another, such that it would be best to selectively use the existing relay allocation for peers that will receive reasonable performance and redirect data flows for other peers to an alternate server. These scenarios are discussed in greater detail below.

The Session Traversal Utilities for NAT (STUN) protocol [RFC5389] defines an ALTERNATE-SERVER mechanism with which a server can redirect a client to another server by replying to a request message with an error response with error code 300 (Try Alternate). The TURN protocol describes error code 300 as one of the possible error codes for an Allocate error response.

This specification describes an additional use of the ALTERNATE-SERVER STUN attribute for TURN that allows the TURN server to redirect a client for the purpose of improving communication with a specific peer without negatively affecting communication with other peers. The client application indicates the nature of the desired response, which allows the client to treat the alternate server selection as either a requirement or a suggestion. This flexibility gives the client the option to choose the best way for the Interactive Connectivity Establishment (ICE) protocol [RFC5245] to respond (e.g. discarding the existing relay candidate for communication with this peer versus evaluating the two candidate servers using ICE connectivity checks and selecting the best one).

1.1. Redirection for Performance

Consider the following example:



When Peer B wishes to communicate with either Peer A or Peer C, it performs a DNS lookup and discovers TURN Relay C, the nearest of the candidate TURN servers. Peer B then sends a TURN Allocate request to TURN Relay C to determine the reflexive and relay candidates to offer. After the reflexive candidate has been chosen, Peer B sends a ChannelBind request to TURN Relay C to establish a channel for communication with the peer. If Peer C is the remote peer, the existing allocation will perform reasonably well, but if Peer A is the remote peer, the latency for relayed packets will be nearly twice as long as if TURN Relay A had been selected as the relay candidate. The problem is worse if Peer B wishes to communicate with both Peer A and Peer C, since there is no single relay candidate that would provide optimum performance for both peers.

If TURN Relay C and TURN Relay A are part of a common TURN service, it would be possible for TURN Relay C to determine that TURN Relay A will provide optimal service for communication between Peer B and Peer A. This allows the TURN service to redirect just the data channel between Peer A and Peer B to TURN relay A, thus providing optimal performance for both relay channels.

The above example describes the problem in terms of physical geography instead of network geography in order to help clarify the discussion. However, readers should note that the problem of selecting a relay server to achieve optimal end-to-end routing is much more complicated than the above description suggests, requiring a detailed real-time view of network connectivity characteristics and the peering relationships between autonomous systems. A naive approach based solely on the physical location of the hosts involved is just as likely to produce negative results as positive ones.

That said, a relay service provider with a broadly distributed system for actively monitoring network performance across the relevant parts of the Internet could make use of the resulting data set to select the optimal relay for each peer pair.

1.2. Redirection for Load Balancing

At the point when a relay allocation is first established, it can be difficult to determine how much aggregate concurrent load could eventually be associated with that allocation. The initiating peer could attempt to use that allocation for any number of peer-to-peer data flows over an extended period of time, during which time load conditions on the relay could change substantially, such that quality of service for already established flows would degrade if the relay were to accept additional flows.

Under these conditions, a TURN service provider with multiple relay hosts and distributed capacity could improve service quality by redirecting data flows to a different host that has more available capacity. At the same time, it is desirable to avoid disrupting established data flows by continuing to handle established flows on the current relay and only redirecting new flows elsewhere.

2. Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119].

3. Peer-specific Server Redirect Mechanism

This specification describes two new uses of the existing STUN ALTERNATE-SERVER attribute. In the first case, the ALTERNATE-SERVER attribute is included with either a CreatePermission error response or a ChannelBind error response. In the second case, the ALTERNATE-SERVER attribute is included with either a CreatePermission success response or a ChannelBind success response.

This specification also defines two new comprehension-optional STUN attributes: CHECK-ALTERNATE and XOR-OTHER-ADDRESS. The CHECK-ALTERNATE attribute is used by the client to request that the server perform peer-specific redirection. The XOR-OTHER-ADDRESS is used by the client to provide an alternate peer address for location identification in the event that the XOR-PEER-ADDRESS attribute in the CreatePermission or ChannelBind request is not expected to reliably serve this purpose.

3.1. Attribute Usage

When sending a CreatePermission or a ChannelBind request, the CHECK-ALTERNATE STUN attribute allows a TURN client to indicate support for

peer-specific server redirection. To maintain backward compatibility with [RFC5766] compliant TURN servers that do not support peer-specific redirection, this attribute is defined as comprehension-optional, which allows a TURN server that does not support peer-specific redirection to ignore the attribute. To maintain backward compatibility with [RFC5766] compliant TURN clients that do not support peer-specific redirection, a TURN server only sends the ALTERNATE-SERVER attribute in CreatePermission and ChannelBind responses when the CHECK-ALTERNATE STUN attribute was present in the request. This prevents transmission of the ALTERNATE-SERVER attribute in cases where the receiving client might not consider the usage legitimate.

The CHECK-ALTERNATE STUN attribute's value indicates the expected server response type: error or success. This capability to declare the expected response type allows TURN client implementers greater flexibility during session establishment. For example, a TURN client implementer may wish to maintain the smallest number of permissions possible during session establishment in order keep the internal client implementation simple, in which case an error response would be desirable. On the other hand, a TURN client implementer may wish to optimize for faster session establishment by continuing to use a sub-optimal allocation while setting up the new one, in which case a success response would be desirable. This second case could be achieved with an error response if the client were to send a second request without the CHECK-ALTERNATE attribute, but such an approach would require an extra RTT.

The XOR-OTHER-ADDRESS STUN attribute allows the TURN client to provide an alternate peer address that can be used by the server to identify the network geographic location of the peer when performing the peer-specific redirection check. Use of this attribute is only necessary if the XOR-PEER-ADDRESS already contained in the CreatePermission or ChannelBind request does not adequately serve this purpose, which should only be true when both peers require a TURN relay for end-to-end data flow. In this case, the TURN CreatePermission or ChannelBind request will provide the peer's TURN relay address as the XOR-PEER-ADDRESS value. If the RTT between the peer and its TURN relay server is very small, the TURN relay address might still be an appropriate address to use for the peer-specific redirection check. As the RTT grows, the TURN relay address will become less suitable for this purpose. For this reason, it is generally the case that the peer's public address (i.e. its host or reflexive address) is a better indication of its network geographic location than its TURN relay address.

Even in cases where both peers require a TURN relay, a typical ICE protocol implementation will give higher candidate priority to the

peer's host and reflexive addresses, which means that the first CreatePermission or ChannelBind request will provide the peer's public address as the XOR-PEER-ADDRESS value and no XOR-OTHER-ADDRESS attribute is necessary. However, although ICE recommends this priority, it does not require it, and so the first request may contain the peer's TURN relay address. With such an implementation, the XOR-OTHER-ADDRESS attribute allows the client to provide the peer's reflexive address in a request that populates the XOR-PEER-ADDRESS attribute with the peer's relay address.

3.2. Sending a CreatePermission or ChannelBind Request

A client that supports peer-specific server redirection and desires such redirection to be performed MUST include the CHECK-ALTERNATE attribute in the first CreatePermission or ChannelBind request when that request is expected to form a new permission or binding. A client MUST NOT include the CHECK-ALTERNATE attribute in a CreatePermission or ChannelBind request that is intended to extend the lifetime of an existing permission or binding.

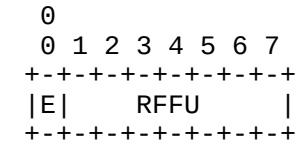
Peer-specific server redirection is only supported for requests that include a single XOR-PEER-ADDRESS attribute. When forming a CreatePermission request with multiple XOR-PEER-ADDRESS attributes, the client MUST NOT include the CHECK-ALTERNATE attribute.

When the CreatePermission or ChannelBind request includes the CHECK-ALTERNATE attribute, the client MAY also include an XOR-OTHER-ADDRESS attribute with a value appropriate for the above described purpose. The XOR-OTHER-ADDRESS attribute SHOULD NOT be included in the request if its value will be identical to the request's XOR-PEER-ADDRESS attribute.

3.2.1. The CHECK-ALTERNATE Attribute

When forming a CHECK-ALTERNATE attribute, the STUN Type is TBD-CA. This type is in the comprehension-optional range, which means that STUN agents can safely ignore the attribute if they do not understand it.

The CHECK-ALTERNATE attribute takes a 1-byte Value, which means that the Length is 1 and 3 bytes of padding are required after the Value. The format of the Value is:



The Value contains a single 1-bit flag:

E: If 1, the server is requested to send a Try Alternate (300) error response when redirection is expected. If 0, the server is request to include an ALTERNATE-SERVER attribute in the success response for the request.

The other 7 bits of the attribute's value must be set to zero on transmission and ignored on reception.

3.2.2. The XOR-OTHER-ADDRESS attribute

When forming an XOR-OTHER-ADDRESS attribute, the STUN Type is TBD-XOA. This type is in the comprehension-optional range, which means that STUN agents can safely ignore the attribute if they do not understand it.

The XOR-OTHER-ADDRESS value specifies an address and port suitable for identification of the peer's network geographic location. It is encoded in the same way as XOR-MAPPED-ADDRESS [RFC5389].

3.3. Receiving a CreatePermission or ChannelBind Request

When a server receives a CreatePermission or ChannelBind request that includes a CHECK-ALTERNATE attribute, it processes as per the TURN specification [RFC5766] plus the specific rules mentioned here.

The server checks the following:

- If the CHECK-ALTERNATE attribute is not recognized, ignore the attribute because its type indicates that it is comprehension-optional. This should be the existing behavior.
- If the message is a CreatePermission request with multiple XOR-PEER-ADDRESS attributes, ignore the CHECK-ALTERNATE attribute if present.
- If peer-specific redirection is not supported by the server, ignore the attribute.
- If the associated permission or binding already exists, ignore the attribute.

If none of the above causes the attribute to be ignored and no other cause for sending an error response has been found, the server attempts to identify an alternate server that will provide better performance for the session based on the criteria supported by the TURN service (e.g. optimal data path and/or load balancing). When an

XOR-OTHER-ADDRESS attribute is found in the request message, the server SHOULD use this address for peer location identification. Otherwise, the server SHOULD use the address provided in the XOR-PEER-ADDRESS attribute.

If no alternate server is identified, the server replies with a success response that does not include an ALTERNATE-SERVER attribute.

If an alternate server is identified and the client requested an error response for redirection, the server rejects the request with a 300 (Try Alternate) error. No new permission or binding is generated on the server in this case.

If an alternate server is identified and the client did not request an error response for redirection, the server creates the permission or binding. The server then replies to the request with a success response, including an ALTERNATE-SERVER attribute in the message.

3.4. Receiving a CreatePermission or ChannelBind Error Response

If the client receives a CreatePermission or ChannelBind error response with error code 420 (Unknown Attribute) and CHECK-ALTERNATE is listed in the UNKNOWN-ATTRIBUTE attribute of the message, the client SHOULD retransmit the original request without the CHECK-ALTERNATE attribute. This case is not expected due to the use of a comprehension-optional attribute type.

If the client receives a CreatePermission or ChannelBind error response with error code 300 (Try Alternate), the client SHOULD attempt to form an allocation to the TURN server indicated in the ALTERNATE-SERVER attribute.

If the alternate server responds to the Allocate request with a success response, the client SHOULD attempt to form a new permission or binding using the new allocation from the alternate server. The CreatePermission or ChannelBind request to the alternate server MAY include a CHECK-ALTERNATE attribute but SHOULD NOT request redirection via an error response. This helps to avoid the possibility of redirection loops.

If the alternate server responds to the Allocate request with an error response, the client MAY resend the original CreatePermission or ChannelBind request, either without the CHECK-ALTERNATE attribute or with a CHECK-ALTERNATE attribute that does not request an error response.

See Section 4 below for discussion of how the client should respond when receiving a Try Alternate error response that was not requested.

3.5. Receiving a CreatePermission or ChannelBind Success Response

If the client receives a CreatePermission or ChannelBind success response, it proceeds with processing according to the TURN specification [RFC5766]. If the message does not include an ALTERNATE-SERVER attribute, no additional processing is required.

If the success response includes an ALTERNATE-SERVER attribute, the client SHOULD attempt to form an allocation to the TURN server indicated in the ALTERNATE-SERVER attribute.

If the alternate server responds to the Allocate request with a success response, the client SHOULD attempt to form a new permission or binding using the new allocation from the alternate server. The CreatePermission or ChannelBind request to the alternate server MAY include a CHECK-ALTERNATE attribute with either attribute value. If this is done, care should be taken in the client implementation to recognize and avoid redirection loops.

While waiting for the new allocation and permission or binding to form via the indicated alternate server, the client SHOULD use the original permission or binding from the request that included the CHECK-ALTERNATE attribute. In this way, peer-specific redirection without an error response can be considered a "hint" that allows the client to establish an alternate path and test its quality before switching to it.

See Section 4 below for discussion of how the client should respond when receiving an ALTERNATE-SERVER attribute that was not requested.

4. Security Considerations

This section considers attacks that are possible in a TURN deployment through the specified protocol extension, and discusses how they are mitigated by mechanisms in the protocol or recommended practices in the implementation.

The specified mechanism affects the use of TURN CreatePermission request messages, ChannelBind request messages, and their respective success and error response messages. Each of these TURN message types requires the MESSAGE-INTEGRITY STUN attribute, which limits attacks that attempt to make use of the specified mechanism to authenticated clients and servers.

4.1. CHECK-ALTERNATE Flood

A compromised TURN client could send a large number of CreatePermission or ChannelBind request messages, which would drive increased load on the TURN server. The CHECK-ALTERNATE attribute does not make such an attack more likely, though it could make it possible to increase the impact of such an attack due to the additional load associated with determining whether an alternate server should be used by the client. The TURN server MAY be configured to ignore the CHECK-ALTERNATE attribute under some conditions in order to limit the associated load. The conditions under which it is appropriate for a TURN server to ignore the CHECK-ALTERNATE attribute are implementation dependent.

4.2. Unsolicited or Invalid ALTERNATE-SERVER

A compromised TURN server could send the "Try Alternate" error code in response to a request message that did not contain the CHECK-ALTERNATE attribute or where the value of the attribute did not request an error response. For client connectivity, this is no worse than any other error response code that could be sent. No matter what the error response code may be, the client is unable to relay data to the remote peer. The client MUST ignore the ALTERNATE-SERVER attribute in error responses when the CHECK-ALTERNATE attribute was not included in the associated request. The client SHOULD ignore the ALTERNATE-SERVER attribute in error responses when the CHECK-ALTERNATE attribute was included in the associated request if the attribute value did not request an error response. The client MAY discontinue use of the associated TURN allocation when an unsolicited Try Alternate error is received.

A compromised TURN server could send an ALTERNATE-SERVER attribute in a success response message for a request message that did not contain the CHECK-ALTERNATE attribute. The client MUST ignore the ALTERNATE-SERVER attribute in success responses when the CHECK-ALTERNATE attribute was not included in the associated request message. The client SHOULD ignore the ALTERNATE-SERVER attribute in success responses when the CHECK-ALTERNATE attribute was included in the associated request if the attribute value requested an error response. The client MAY discontinue use of the associated TURN allocation when an unsolicited ALTERNATE-SERVER attribute is received.

A compromised TURN server could send an invalid ALTERNATE-SERVER attribute value in either an error or a success response message, where the value refers to an unaffiliated TURN server to which the sending TURN server is not allowed to redirect traffic. Such an attack is already allowed by the use of Try Alternate errors in

response to Allocate request messages. Use of the ALTERNATE-SERVER attribute in the context of peer-specific redirection does not make such an attack more likely, though it could make it possible to increase the scale of such an attack by allowing multiple ALTERNATE-SERVER attributes to each client, one per requested permission or binding. A client SHOULD ignore all future ALTERNATE-SERVER attributes received from the TURN server after an authentication failure with any server identified via an ALTERNATE-SERVER attribute. A client MAY discontinue use of the associated TURN allocation after an authentication failure with any server identified via an ALTERNATE-SERVER attribute.

5. IANA Considerations

[Paragraphs below in braces should be removed by the RFC Editor upon publication]

[The CHECK-ALTERNATE attribute requires that IANA allocate a value in the "STUN attributes Registry" from the comprehension-optional range (0x8000-0xFFFF), to be replaced for TBD-CA throughout this document]

This document defines the CHECK-ALTERNATE STUN attribute, described in Section 3.2.1. IANA has allocated the comprehension-optional codepoint TBD-CA for this attribute.

[The XOR-OTHER-ADDRESS attribute requires that IANA allocate a value in the "STUN attributes Registry" from the comprehension-optional range (0x8000-0xFFFF), to be replaced for TBD-XOA throughout this document]

This document defines the XOR-OTHER-ADDRESS STUN attribute, described in Section 3.2.2. IANA has allocated the comprehension-optional codepoint TBD-XOA for this attribute.

6. References

6.1. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

6.2. Informative References

[RFC5245] Rosenberg, J., "Interactive Connectivity Establishment (ICE): A Protocol for Network Address Translator (NAT) Traversal for Offer/Answer Protocols", RFC 5245,

April 2010.

[RFC5389] Rosenberg, J., Mahy, R., Matthews, P., and D. Wing, "Session Traversal Utilities for NAT (STUN)", RFC 5389, October 2008.

[RFC5766] Mahy, R., Matthews, P., and J. Rosenberg, "Traversal Using Relays around NAT (TURN): Relay Extensions to Session Traversal Utilities for NAT (STUN)", RFC 5766, April 2010.

Authors' Addresses

Brandon Williams
Akamai, Inc.
8 Cambridge Center
Cambridge, MA 02142
USA

Email: brandon.williams@akamai.com

Tirumaleswar Reddy
Cisco Systems, Inc.
Cessna Business Park, Varthur Hobli
Sarjapur Marathalli Outer Ring Road
Bangalore, Karnataka 560103
India

Email: tireddy@cisco.com