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

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 7, 2015.

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.

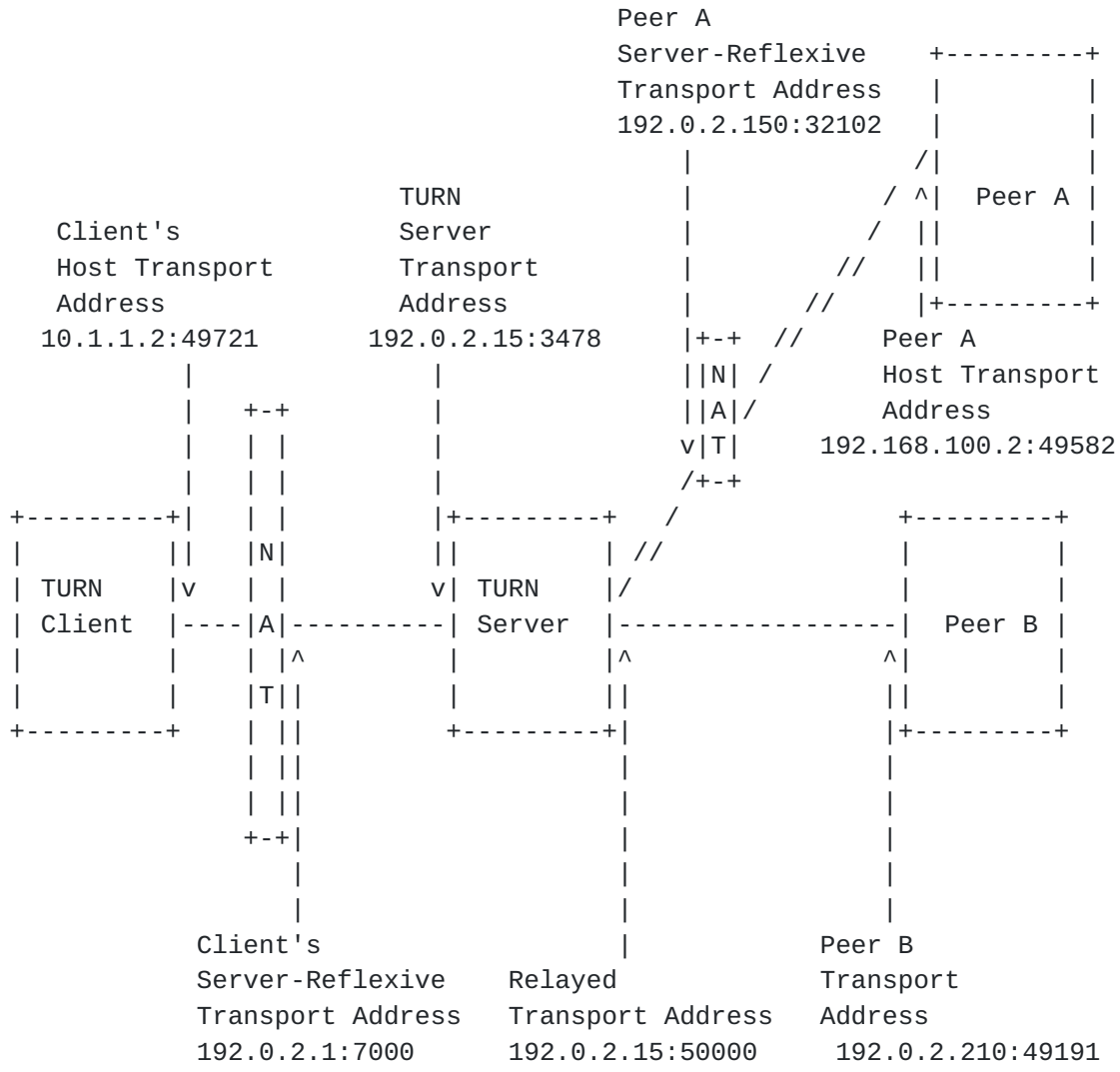


Figure 1

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.

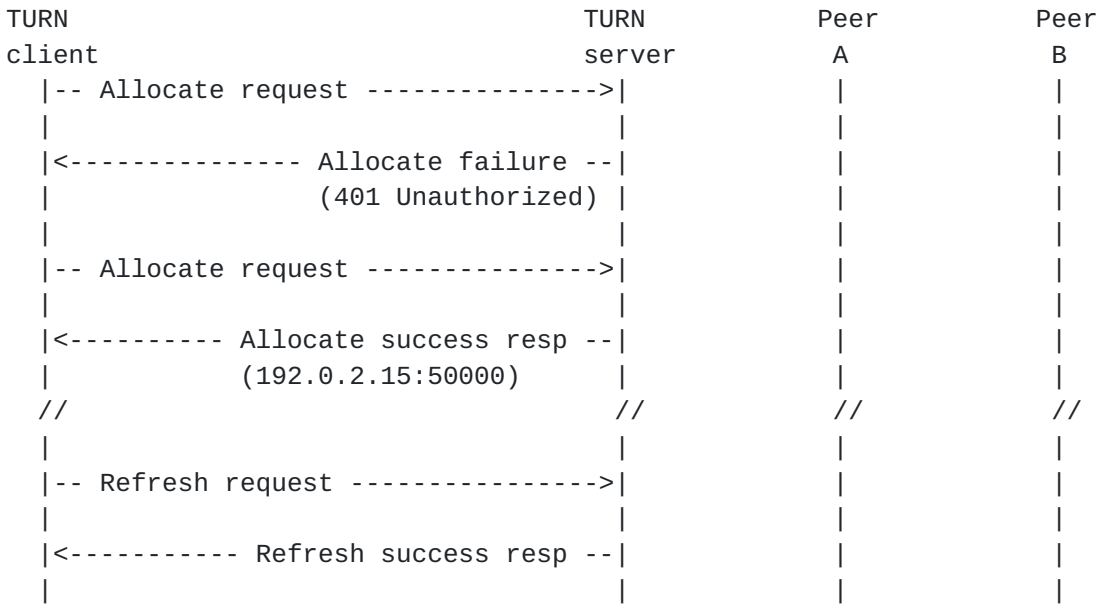


Figure 2

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.

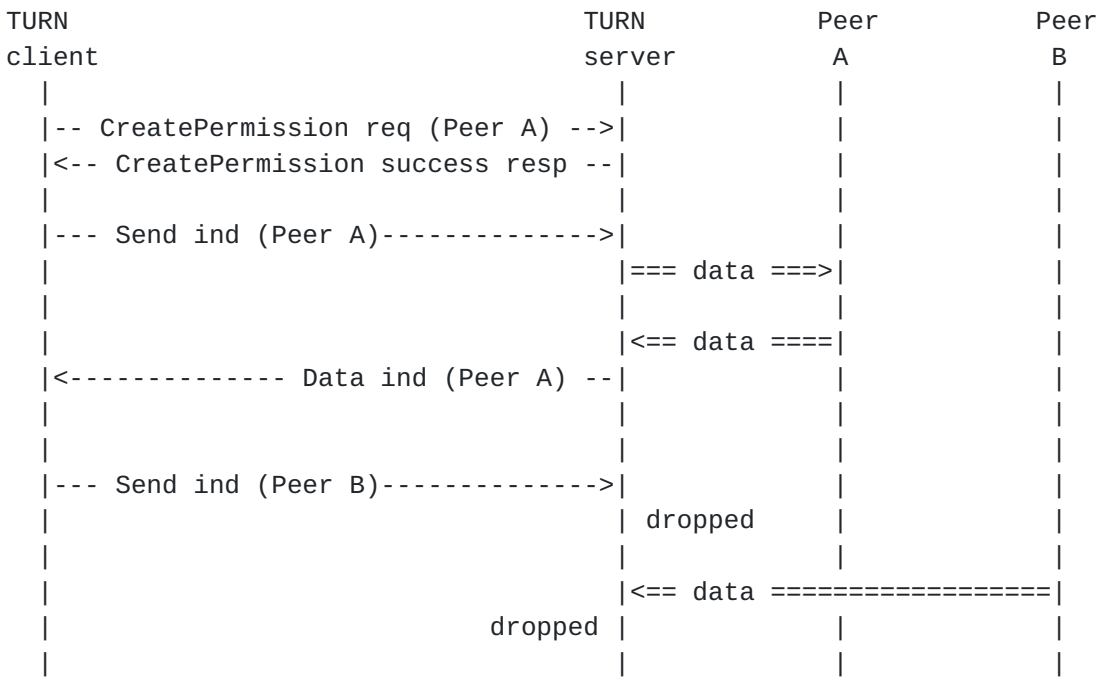


Figure 3

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.

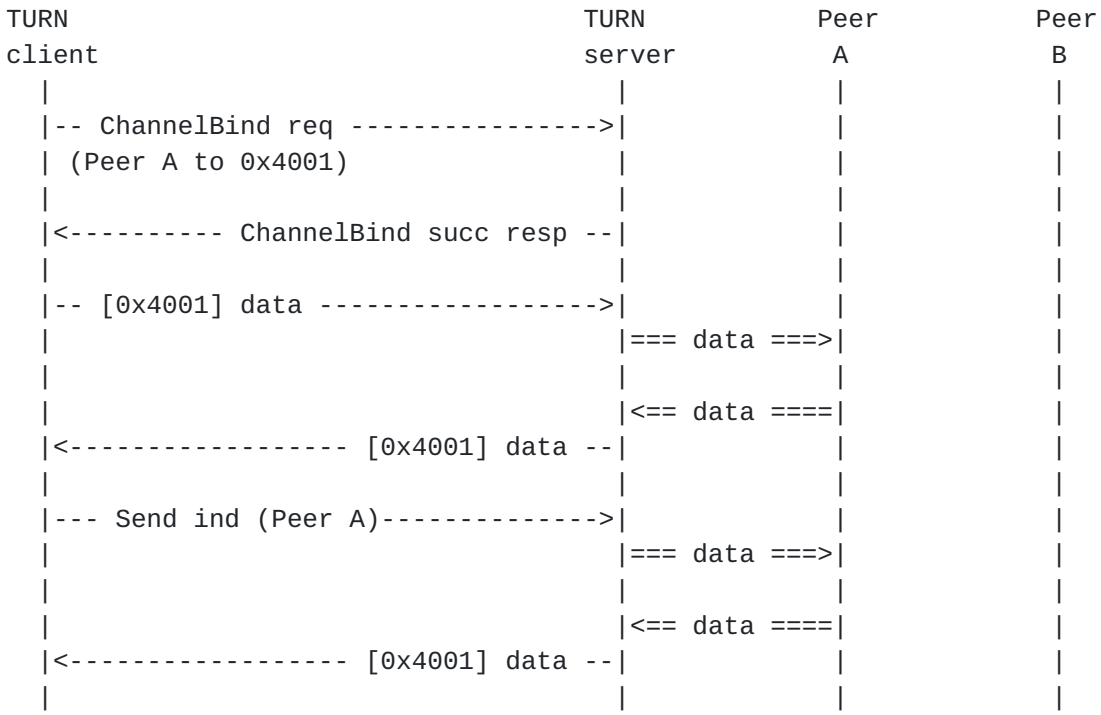


Figure 4

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;

- 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.

- 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.

- 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.

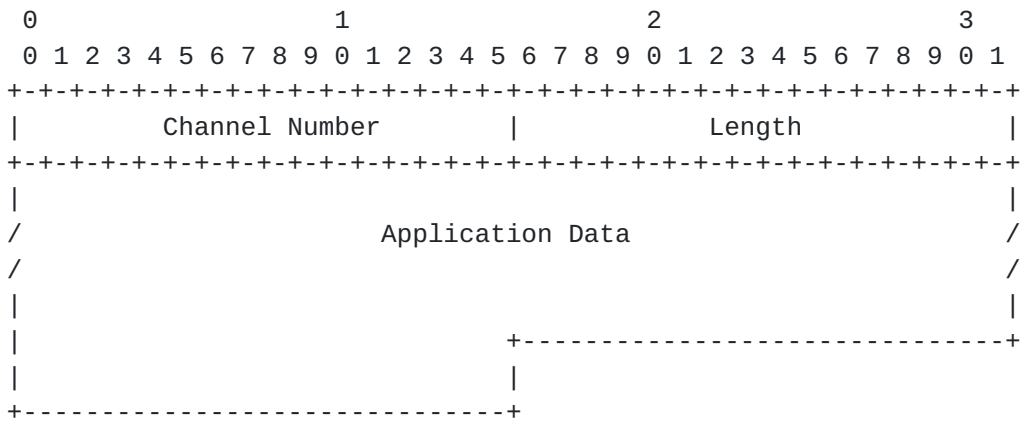
11.3. Receiving a ChannelBind Response

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

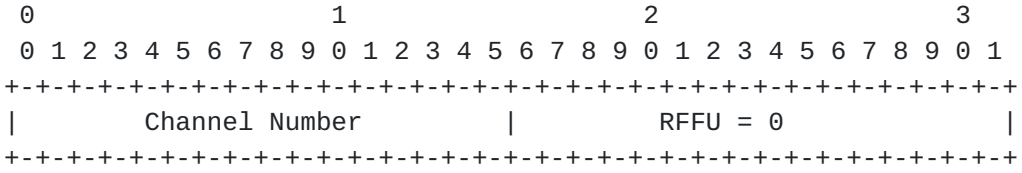
This STUN extension defines the following new attributes:

```
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

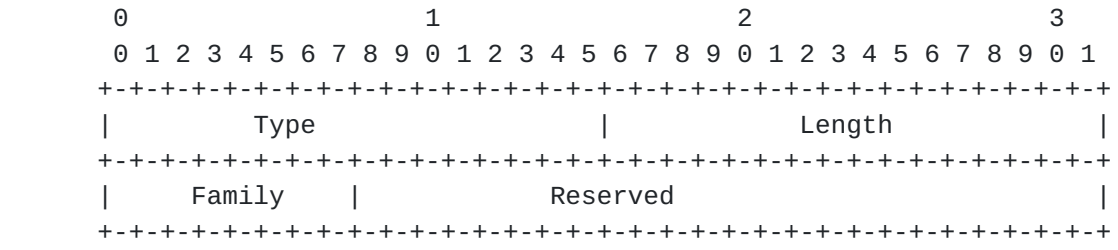
The DATA attribute is present in all Send and Data indications. The value portion of this attribute is variable length and consists of the application data (that is, the data that would immediately follow the UDP header if the data was been sent directly between the client and the peer). If the length 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:

```
 0
 0 1 2 3 4 5 6 7
+-+-+-+-+-+-+-+-+
|R|    RFFU     |
+-+-+-+-+-+-+-+-+
```

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:

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|    Protocol   |                    RFFU                       |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

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.

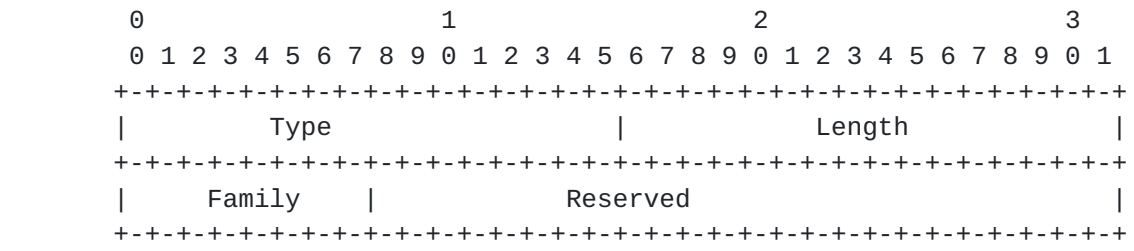
15.10. RESERVATION-TOKEN

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.

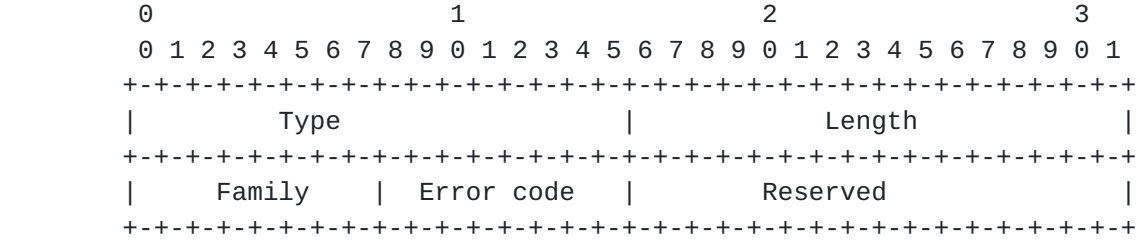
Family: there are two values defined for this field and specified in [RFC5389], Section 15.1: 0x01 for IPv4 addresses and 0x02 for IPv6 addresses.

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.

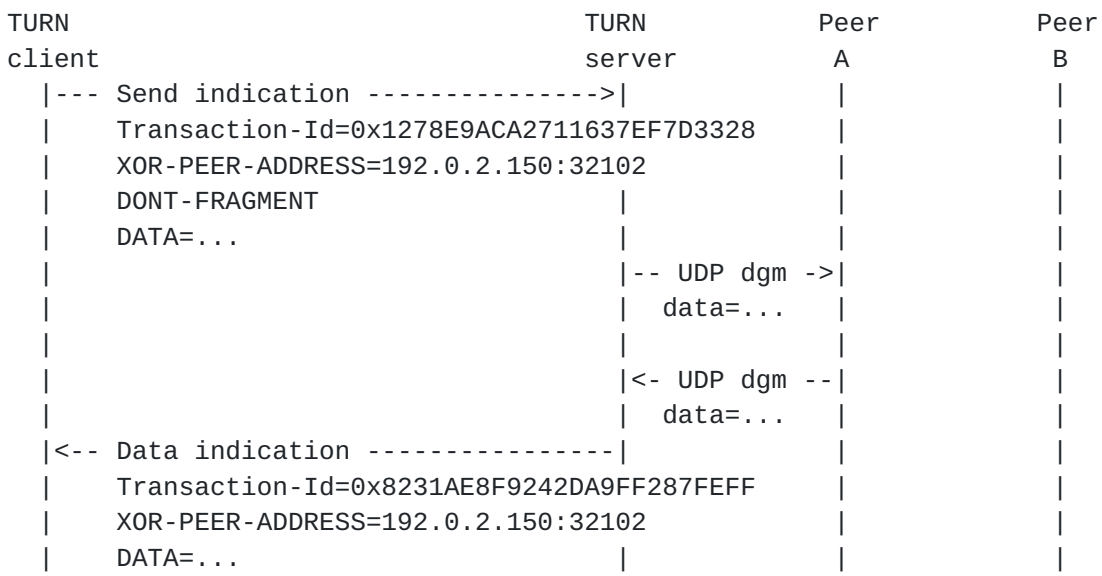
The client, upon receipt of the 401 error, re-attempts the Allocate request, this time including the authentication attributes. The client selects a new transaction id, and then populates the new Allocate request with the same attributes as before. The client includes a USERNAME attribute and uses the 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.



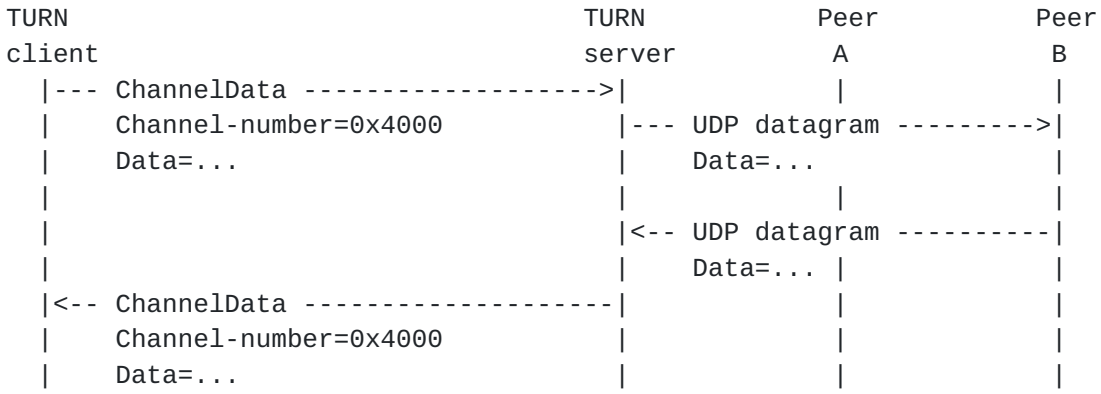
The client now sends application data to Peer A using a Send indication. Peer A's server-reflexive transport address is specified in the XOR-PEER-ADDRESS attribute, and the application data (shown here as just "...") is specified in the DATA attribute. The client is doing a form of path MTU discovery 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.

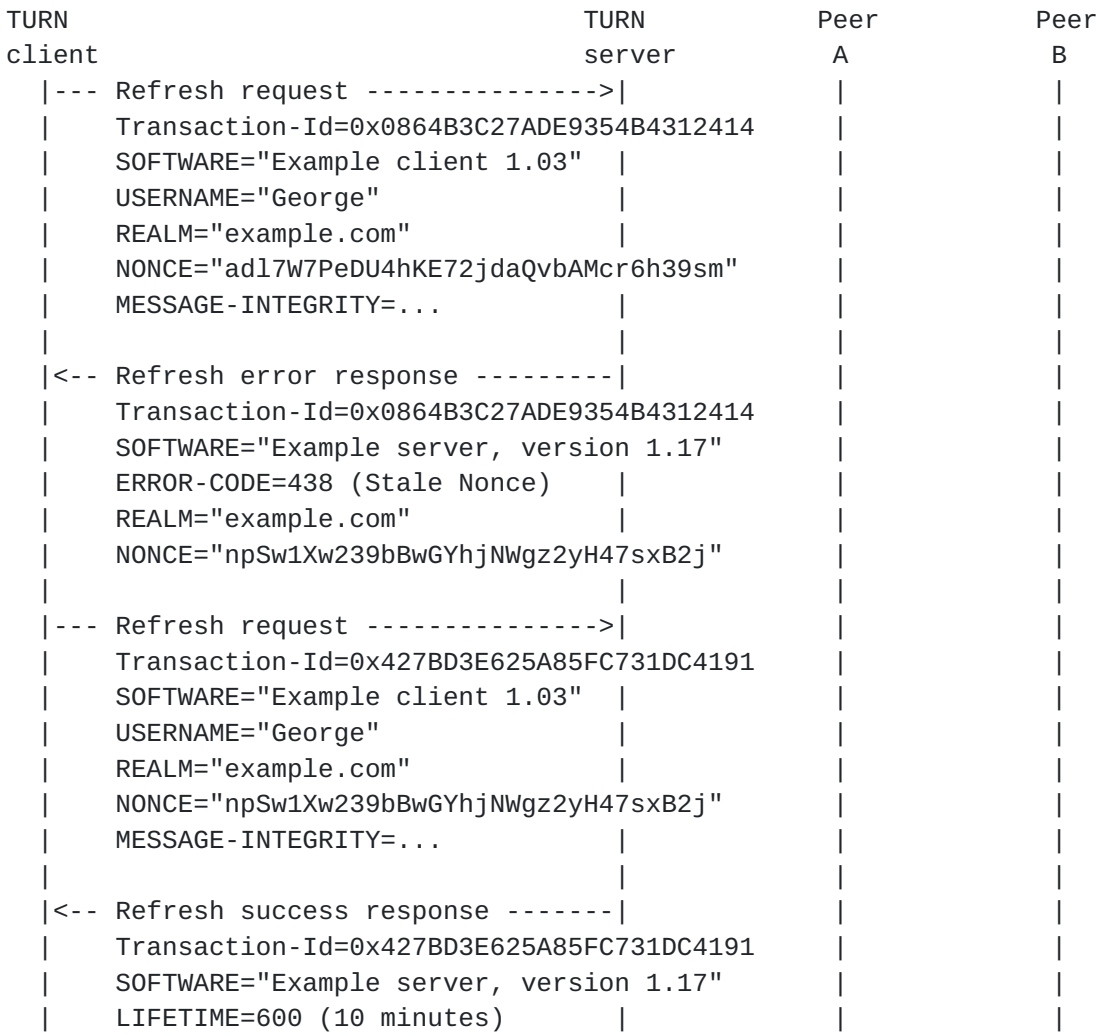
The client now binds a channel to Peer B, specifying a free channel number (0x4000) in the CHANNEL-NUMBER attribute, and Peer B's transport address in the XOR-PEER-ADDRESS attribute. As before, the client re-uses the username, realm, and nonce from its last request in the message.

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

- 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.

- IPv6 support.
- REQUESTED-ADDRESS-FAMILY, ADDITIONAL-ADDRESS-FAMILY, AND ADDRESS-ERRR-CODE attributes.
- 440 (Address Family not Supported) and 443 (Peer Address Family Mismatch) responses.
- Description of the tunnel amplification attack.
- DTLS support.
- 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