BEHAVE WG	J. Rosenberg	
Internet-Draft	Cisco	
Intended status: Standards Track	R. Mahy	
Expires: August 28, 2008	Plantronics	
	P. Matthews	
	Avaya	
	February 25, 2008	

TOC

Traversal Using Relays around NAT (TURN): Relay Extensions to Session Traversal Utilities for NAT (STUN)
draft-ietf-behave-turn-07

Status of this Memo

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) located behind other NATs. 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.

The TURN protocol can be used in isolation, but is more properly used as part of the ICE (Interactive Connectivity Establishment) approach to NAT traversal.

Table of Contents

---

1. Introduction

TOC

Session Traversal Utilities for NAT (STUN) [I-D.ietf-behave-rfc3489bis] (Rosenberg, J., Mahy, R., Matthews, P., and D. Wing, "Session Traversal Utilities for (NAT) (STUN)," July 2008.) provides a suite of tools for facilitating the traversal of NAT. Specifically, it defines the Binding method, which is used by a client to determine its reflexive transport address towards the STUN server. The reflexive transport address can be used by the client for receiving packets from peers, but only when the client is behind "good" NATs. In particular, if a client is behind a NAT whose mapping behavior [RFC4787] (Audet, F. and C. Jennings, "Network Address Translation (NAT) Behavioral Requirements for Unicast UDP," January 2007.) is address or address and port dependent (sometimes called "bad" NATs), the reflexive transport address will not be usable for communicating with a peer.

The only reliable way to obtain a UDP transport address that can be used for corresponding with a peer through such a NAT is to make use of a relay. The relay sits on the public side of the NAT, and allocates transport addresses to clients reaching it from behind the private side of the NAT. These allocated transport addresses, called relayed transport address, are IP addresses and ports on the relay. When the

relay receives a packet on one of these allocated addresses, the relay forwards it toward the client.
This specification defines an extension to STUN, called TURN, that allows a client to request a relayed transport address on a TURN server.
Though a relayed transport address is highly likely to work when corresponding with a peer, it comes at high cost to the provider of the relay service. As a consequence, relayed transport addresses should only be used as a last resort. Protocols using relayed transport addresses should make use of mechanisms to dynamically determine whether such an address is actually needed. One such mechanism, defined for multimedia session establishment protocols based on the offer/answer protocol in RFC 3264 (Rosenberg, J. and H. Schulzrinne, "An Offer/Answer Model with Session Description Protocol (SDP)," June 2002.) [RFC3264], is Interactive Connectivity Establishment (ICE) [I-D.ietf-mmusic-ice] (Rosenberg, J., "Interactive Connectivity Establishment (ICE): A Protocol for Network Address Translator (NAT) Traversal for Offer/Answer Protocols," October 2007.).
TURN was originally invented to support multimedia sessions signaled using SIP. Since SIP supports forking, TURN supports multiple peers per client; a feature not supported by other approaches (e.g., SOCKS [RFC1928] (Leech, M., Ganis, M., Lee, Y., Kuris, R., Koblas, D., and L. Jones, "SOCKS Protocol Version 5," March 1996.)). However, care has been taken in the later stages of its development to make sure that TURN is suitable for other types of applications.

---

2. Overview of Operation

TOC

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 (Rekhter, Y., Moskowitz, R., Karrenberg, D., Groot, G., and E. Lear, "Address Allocation for Private Internets," February 1996.) [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 that the TURN client wishes to communicate with. These peers may or may not be behind one or more NATs.

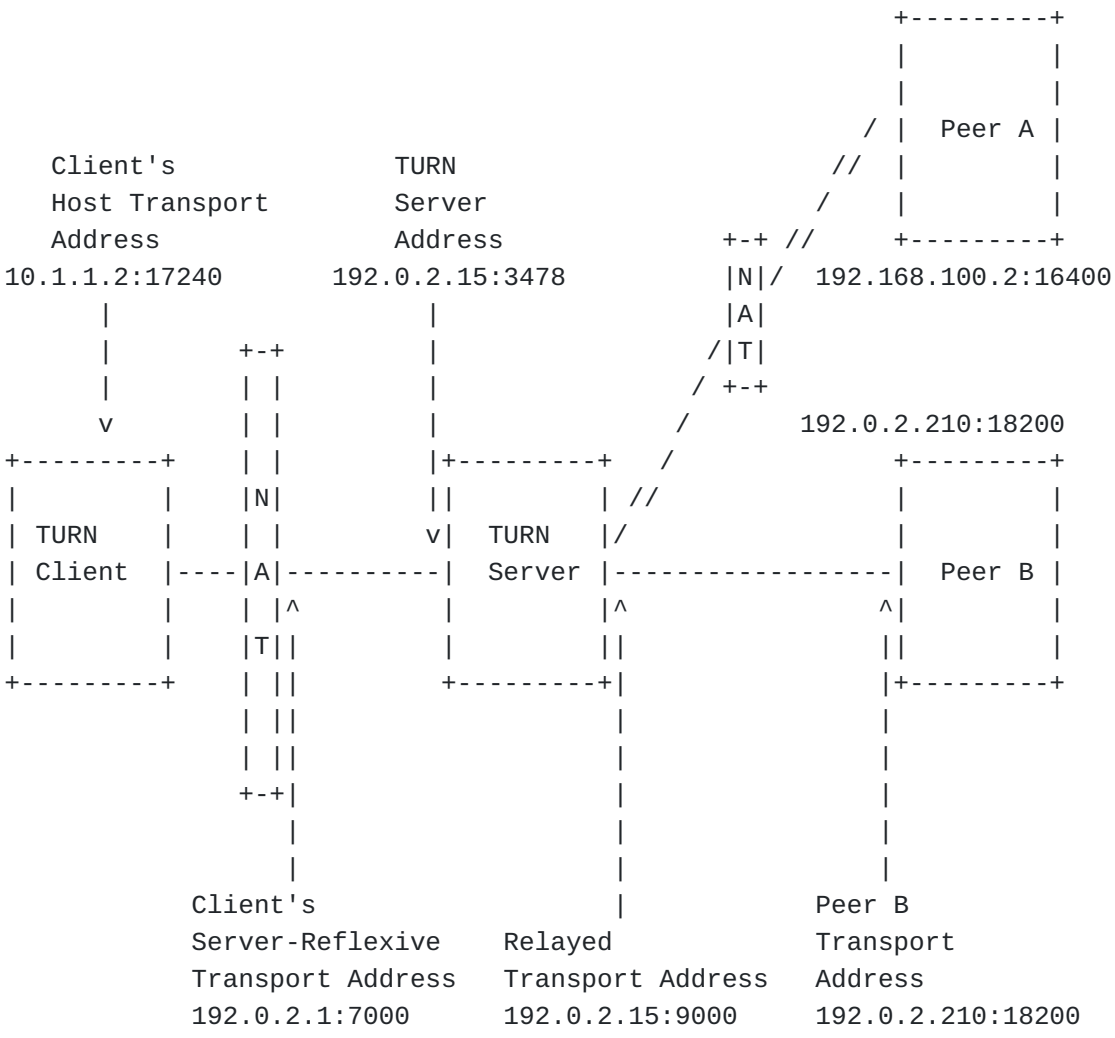


Figure 1

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] (Audet, F. and C. Jennings, “Network Address Translation (NAT) Behavioral Requirements for Unicast UDP,” January 2007.)) for a description of what this means).

The client has allocated a local port on one of its addresses for use in communicating with the server. The combination of an IP address and a port is called a TRANSPORT ADDRESS and since this (IP address, port) combination is located on the client and not on the NAT, it is called the client’s HOST transport address.

The client sends TURN messages from its host transport address to a transport address on the TURN server which is known as the TURN SERVER

ADDRESS. The client learns the server's address through some unspecified means (e.g., configuration), and this address is typically used by many clients simultaneously. The TURN server address is used by the client to send both commands and data to the server; the commands are processed by the TURN server, while the data is relayed on to the peers.

Since the client is behind a NAT, the server sees these packets 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 allocate a RELAYED TRANSPORT ADDRESS, which is an transport address located on the TURN server. The server ensures that there is a one-to-one relationship between the client's server-reflexive transport address and the relayed transport address; thus a packet received at the relayed transport address can be unambiguously relayed by the server to the client.

The client will typically communicate this relayed transport address to one or more peers through some mechanism not specified here (e.g., an ICE offer or answer [I-D.ietf-mmusic-ice] (Rosenberg, J., "Interactive Connectivity Establishment (ICE): A Protocol for Network Address Translator (NAT) Traversal for Offer/Answer Protocols," October 2007.)). Once this is done, the client can send data to the server to relay towards its peers. In the reverse direction, peers can send data to the the relayed transport address of the client. The server will relay this data to the client as long as the client explicitly created a permission (see Section 2.5 (Permissions)) for the IP address of the peer.

2.1. Transports

TOC

TURN as defined in this specification only allows the use of UDP between the server and the peer. However, this specification allows the use of any one of UDP, TCP, or TLS over TCP 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

If TCP or TLS over TCP is used between the client and the server, then the server will convert between stream transport and UDP transport when relaying data. TURN allows both TCP and TLS over TCP as transports in part because many firewalls are configured to not pass any UDP traffic.

For TURN clients, using TLS over TCP to communicate with the TURN server provides two benefits. First, the client can be assured that the addresses of its peers are not visible to any attackers between it and the server. Second, the client may be able to communicate with TURN servers using TLS when it would not be able to communicate with the same server using TCP or UDP, due to the policy of a firewall between the TURN client and its server. In this second case, TLS between the client and TURN server facilitates traversal.
There is a planned extension to TURN to add support for TCP between the server and the peers [I-D.ietf-behave-turn-tcp] (Perreault, S. and J. Rosenberg, “Traversal Using Relays around NAT (TURN) Extensions for TCP Allocations,” March 2010.). 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.

2.2. Allocations

TOC

To allocate a relayed transport address, the client uses an Allocate transaction. The client sends a Allocate Request to the server, and the server replies with an Allocate 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). And since relaying data can require lots of bandwidth, the server typically requires that the client authenticate itself 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 with the allocated related transport address. 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 client can request a 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. If a client no longer wishes to use an Allocation, it should do a Refresh transaction with a requested lifetime of 0.
Note that sending or receiving data from a peer DOES NOT refresh the allocation.
The server keeps track of the client reflexive transport address and port, the server transport address and port, and the protocol used by the client to communicate with the server. (Together known as a 5-

tuple. The server remembers the 5-tuple used in the Allocate Request. Subsequent transactions between the client and the server use this same 5-tuple. In this way, the server knows which client owns the allocated relayed transport address. If the client wishes to allocate a second relayed transport address, it must use a different 5-tuple for this allocation (e.g., by using a different client host address).,

> 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, many implementations use a file-descriptor in place of a 5-tuple to represent a TCP connection.

2.3. Exchanging Data with Peers

TOC

There are two ways for the client and peers to exchange data using the TURN server. The first way uses Send and Data indications, 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 to forward the data to, and for the server to indicate which peer sent the data.
When using the first way, the client sends a Send indication to the TURN server containing, in attributes inside the indication, the transport address of the peer and the data to be sent to that peer. When the TURN server receives the Send Indication, it extracts the data from the Send Indication and sends it in a UDP datagram to the peer, using the allocated relay address as the source address. In the reverse direction, UDP datagrams arriving at the relay address on the TURN server are converted into Data Indications and sent to the client, with the transport address of the peer included in an attribute in the Data Indication.

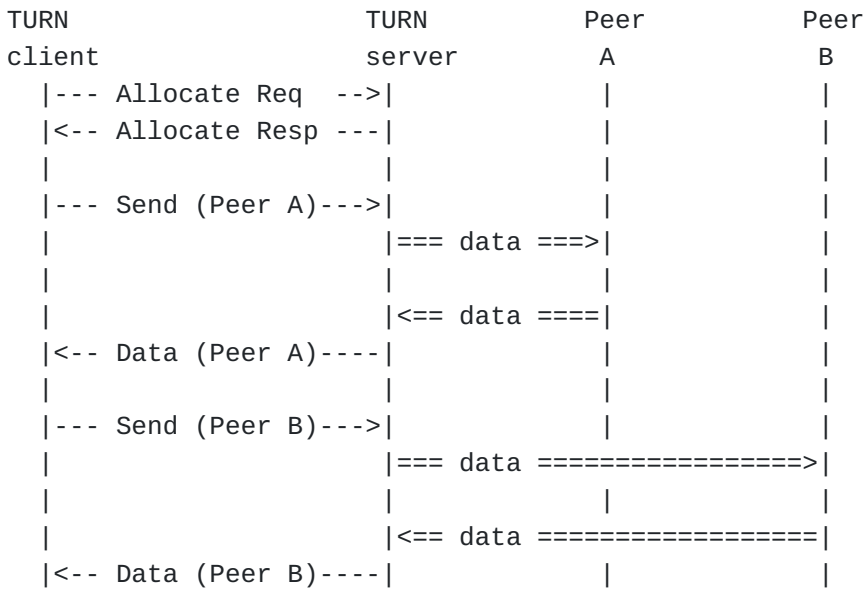


Figure 2

In the figure above, the client first allocates a relayed transport address. It then sends data to Peer A using a Send Indication; at the server, the 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. A similar exchange happens with Peer B.

2.4. Channels

TOC

For some applications (e.g. Voice over IP), the 36 bytes of overhead that a Send 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 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. Channel bindings are refreshed by sending ChannelData messages from the client to the server, or by rebinding the channel to the peer.

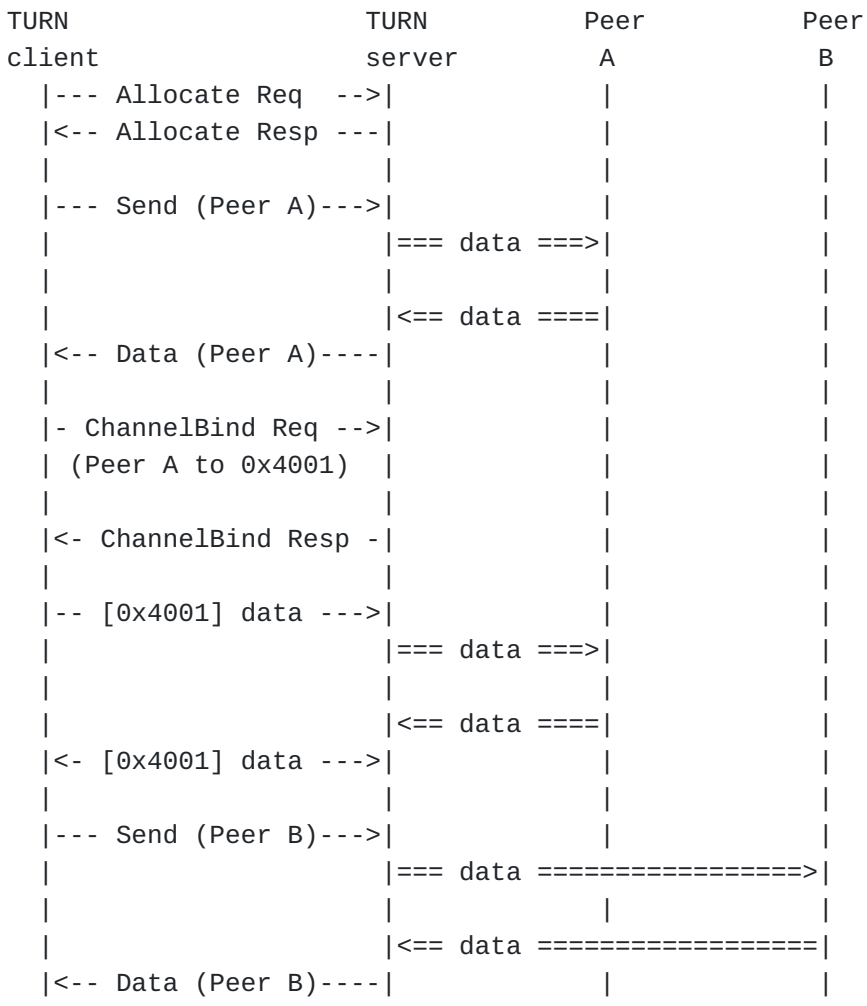


Figure 3

The figure above shows the channel mechanism in use. The client begins by allocating a relayed transport address, and then uses that address to exchange data with Peer A. After a bit, the client decides to bind a channel to Peer A. To do this, it 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.

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 uses Send and Data indications.
Channel bindings are always initiated by the client.

2.5. Permissions

TOC

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] (Audet, F. and C. Jennings, “Network Address Translation (NAT) Behavioral Requirements for Unicast UDP,” January 2007.).
The client can install a permission by sending data to a peer (or by doing certain other things). Once a permission is installed, any peer with the same IP address (the ports numbers can differ) is permitted to send data to the client. After 5 minutes, the permission times out and the server drops any UDP datagrams arriving at the relayed transport from that IP address. Note that permissions are within the context of an allocation, so adding or expiring a permission in one allocation does not affect other allocations.
Data received from the peer DOES NOT refresh the permission.

---

3. Terminology

TOC

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 (Bradner, S., “Key words for use in RFCs to Indicate Requirement Levels,” March 1997.) [RFC2119].
Readers are expected to be familar with [I-D.ietf-behave-rfc3489bis] (Rosenberg, J., Mahy, R., Matthews, P., and D. Wing, “Session Traversal Utilities for (NAT) (STUN),” July 2008.) and the terms defined there.
The following terms are used in this document:

TURN: A protocol spoken between a TURN client and a TURN server. It is an extension to the STUN protocol [I-D.ietf-behave-rfc3489bis] (Rosenberg, J., Mahy, R., Matthews, P., and D. Wing, “Session Traversal Utilities for (NAT) (STUN),” July 2008.). 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.

Host Transport Address: A transport address allocated on a host.

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 that exists on a TURN server. If a permission exists, packets that arrive at this address are relayed towards the TURN client.

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 (UDP or TCP)) 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.

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.

---

4. General Behavior

TOC

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 [I-D.ietf-behave-rfc3489bis] (Rosenberg, J., Mahy, R., Matthews, P., and D. Wing, “Session Traversal Utilities for (NAT) (STUN),” July 2008.) apply to STUN-formatted messages. This

means that all the message-forming and -processing descriptions in this document are implicitly prefixed with the rules of [I-D.ietf-behave-rfc3489bis] (Rosenberg, J., Mahy, R., Matthews, P., and D. Wing, “Session Traversal Utilities for (NAT) (STUN),” July 2008.).
In addition, the server SHOULD require that all TURN requests use the Long-Term Credential mechanism described in [I-D.ietf-behave-rfc3489bis] (Rosenberg, J., Mahy, R., Matthews, P., and D. Wing, “Session Traversal Utilities for (NAT) (STUN),” July 2008.), and the client MUST be prepared to authenticate requests if required. 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).
The client and/or the server MAY include the FINGERPRINT attribute in any of the methods defined in this document. However, TURN does not use the backwards-compatibility mechanism described in [I-D.ietf-behave-rfc3489bis] (Rosenberg, J., Mahy, R., Matthews, P., and D. Wing, “Session Traversal Utilities for (NAT) (STUN),” July 2008.).
By default, TURN runs on the same port as STUN. However, either the SRV procedures or the ALTERNATE-SERVER procedures described in Section 6 (Creating an Allocation) may be used to run TURN on a different port.

---

5. Allocations

TOC

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

- Relayed transport address
- The 5-tuple: client IP address, client port, server IP address, server port, transport protocol
- Username
- Transaction ID of the Allocate request
- Bandwidth
- Time-to-expiry
- List of permissions

*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. Both of these MUST be unique across all allocations, so either one can be used to uniquely identify the allocation.

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.

The username and password of the allocation is the username and password of the authenticated Allocate request that creates the allocation. Subsequent requests on an allocation use the same username and password as those 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 438 (Wrong Credentials) error.

The transaction ID of the allocation is the transaction ID used in the Allocate request. This is used to detect retransmissions of the Allocate request over UDP (see Section 6.2 (Receiving an Allocate Request) for details).

The bandwidth is the maximum bandwidth between the client and the server that the client expects to need (in either direction). The server MAY choose to police this value and refuse allocations to ensure that the total bandwidth across all allocations does not exceed the server's capacity. Servers that do so SHOULD require that an allocation's bandwidth lie within two values: the minimum per-allocation bandwidth and the maximum per-allocation bandwidth.

> NOTE: Readers should be aware that the details around bandwidth are still preliminary. The present description is likely to change, perhaps significantly, before the specification is finalized.

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 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 is freed. However the server MUST ensure that neither the relayed transport address nor the client reflexive transport address from the 5-tuple are re-used in other allocations until 2 minutes after the allocation expires; this ensures that any messages that are in transit when the allocation expires are gone before either of these transport addresses are re-used.
The list of permissions is described in Section 8 (Permissions) and the list of channels is described in Section 10 (Channels).

---

6. Creating an Allocation

TOC

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

6.1. Sending an Allocate Request

TOC

The client forms an Allocate request as follows.
The client first needs to pick a host transport address that the server does not think is currently in use, or was recently in use. The client SHOULD pick a currently-unused transport address on the client's host (typically by allowing its OS to pick a currently-unused port for a new socket).
The client needs to pick a transport protocol to use between the client and the server. The transport protocol MUST be one of UDP, TCP, or TLS over TCP. 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 (Transports) for more discussion.
The client must also pick a server transport address. Typically, this is done by the client learning (perhaps through configuration) one or more domain names for TURN servers. In this case, the client uses the DNS procedures described in [I-D.ietf-behave-rfc3489bis] (Rosenberg, J., Mahy, R., Matthews, P., and D. Wing, “Session Traversal Utilities for (NAT) (STUN),” July 2008.), but using an SRV service name of "turn" (or "turns" for TURN over 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, or TLS over TCP, 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: NOT the one in the 5-tuple). In this specification,

the REQUESTED-TRANSPORT type is always UDP. This attribute is included to allow future extensions specify other protocols (e.g., [I-D.ietf-behave-turn-tcp] (Perreault, S. and J. Rosenberg, "Traversal Using Relays around NAT (TURN) Extensions for TCP Allocations," March 2010.)).

The client MAY include a BANDWIDTH attribute, describing the maximum bandwidth that the client expects to exchange between it and the server over this allocation. This is just a request, and the server may elect to use a different value. If the client omits this attribute, the server will pick a bandwidth for the allocation.

If the client wishes the server to initialize the time-to-expire field of the allocation to some value other the default lifetime, then it MAY include a LIFETIME attribute specifying its desired value. This is just a request, 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 communicate with older peers that make certain assumptions about the port numbers that an endpoint uses, then it MAY include either a REQUESTED-PROPS attribute or a RESERVATION-TOKEN attribute (but not both). Using the REQUESTED-PROPS attribute, the client can request:

- That the server allocate a relayed transport address with an even port number, OR

- That the server reserve a pair of relayed transport addresses with adjacent port numbers N and N+1, where N is even and N+1 is odd, and then use port N for the current allocation. In this case, the server returns a RESERVATION-TOKEN attribute in the response which the client can then include in a subsequent Allocate request to create an allocation with port number N+1.

The client then sends the allocation on the 5-tuple.

6.2. Receiving an Allocate Request

TOC

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

1. The server checks the credentials of the request, as per the Long-Term Credential mechanism of [I-D.ietf-behave-rfc3489bis] (Rosenberg, J., Mahy, R., Matthews, P., and D. Wing, "Session Traversal Utilities for (NAT) (STUN)," July 2008.).

2. The server checks if the 5-tuple is currently in use by an existing allocation, or was it in use by another allocation

within the last 2 minutes. If yes, then there are two sub-cases:

- If the transport protocol in the 5-tuple is UDP, and if the 5-tuple is currently in use by an existing allocation, and if the transaction id of the request matches the transaction id stored with the allocation, then the request is a retransmission of the original request. The server replies either with a stored copy of the original response, or with a response rebuilt from the stored state data. If the server chooses to rebuild the response, then (a) it need not parse the request further, but can immediately start building a success response, (b) the value of the LIFETIME attribute can be set to the current value of the time-to-expire timer, and (c) the server may need to include an extra field in the allocation to store the token returned in a RESERVATION-TOKEN attribute.

- Otherwise, the server rejects the request with a 437 (Allocation Mismatch) error.

NOTE: If the request includes credentials that are acceptable to server, but the 5-tuple is already in use, then it is important that the server reject the request with a 437 (Allocation Mismatch) error rather than a 401 (Unauthorized) error. This ensures that the client knows that the problem is with the 5-tuple, rather than (wrongly) believing that the problem lies with its credentials.

3. The server checks if the request contain a REQUESTED-TRANPORT 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 422 (Unsupported Transport Protocol) error.

4. The server checks if the request contains a BANDWIDTH attribute. If yes, but the attribute is malformed or is out of range, the server rejects the request with a 400 (Bad Request) error. Otherwise, the server checks if it is willing to grant the bandwidth request. The details of this check are described below. If the server is not willing, it rejects the request with a 507 (Insufficient Bandwidth Capacity) error.

5. The server checks if the request contains a REQUESTED-PROPS attribute. If yes, then the server checks if it understands the prop-type and can satisfy the request. If the prop-type is not understood, or if the server cannot satisfy the request, then

the server rejects the request with a 508 (Insufficient Port Capacity) error.

6. The server checks if the request contains a RESERVATION-TOKEN attribute. If yes, and the request also contains a REQUESTED-PROPS attribute, then the server rejectes 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 Port Capacity) error.

7. At any point, the server MAY also 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.

If the server rejects the request with one of the error codes 422 (Unsupported Transport Protocol), 486 (Allocation Quota Reached), 507 (Insufficient Bandwidth Capacity) or 508 (Insufficient Port Capacity), it MAY include an ALTERNATE-SERVER attribute in the error response redirecting the client to another server that it believes will accept the request. If the attribute is included, the address MUST be from the same address family as the server's transport address. Note that, if the attribute is included, the client will try this alternate server before trying the other servers given by the SRV procedures.
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.
When allocating a relayed transport address for the allocation, the server MUST allocate an IP address of the same family (e.g, IPv4 vs. IPv6) as the server's transport address.

> NOTE: An extension to TURN to allow an address from a different address family is currently in progress [I-D.ietf-behave-turn-ipv6] (Camarillo, G., Novo, O., and S. Perreault, “Traversal Using Relays around NAT (TURN) Extension for IPv6,” March 2010.).

In addition, the server SHOULD only allocate ports from the range 49152 – 65535 (the Dynamic and/or Private Port range [Port-Numbers] (, “IANA Port Numbers Registry,” .)), 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.

If the request contains a REQUESTED-PROPS attribute requesting a pair of ports, 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.

If the request contains a RESERVATION-TOKEN, the server uses the previously-reserved transport address corresponding to the included token (if it is still available).

The server determines the initial value of the allocation's bandwidth as follows. If the BANDWIDTH attribute was not included, or if the requested bandwidth is less than the minimum per-allocation bandwidth, then the server behaves as if the minimum per-allocation bandwidth was requested. Otherwise, if the request bandwidth is greater than the maximum per-allocation bandwidth, then the server behaves as if the maximum per-allocation bandwidth was requested.

The server then check if the (updated) requested bandwidth is available, and if necessary reduces the requested bandwidth to the amount that is willing to grant. If the result less than the minimum per-allocation bandwidth, then the server considers the request to be unsatisfiable, and rejects the request with a 507 (Insufficient Bandwidth Capacity) error. Otherwise, the requested bandwidth becomes the bandwidth of the allocation.

The server determines the initial value of the time-to-expire field as follows. If the request contains a LIFETIME attribute, and the proposed lifetime value is greater than the default lifetime, and the proposed lifetime value is otherwise acceptable to the server, then the server uses that value. Otherwise, the server uses the default value. It is RECOMMENDED that the server impose a maximum lifetime of no more than 3600 seconds (1 hour).

> NOTE: Both the bandwidth and the time-to-expire are recomputed with each successful Refresh request. Thus the values computed here apply only until the first refresh.

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

*A RELAYED-ADDRESS attribute containing the relayed transport address;

*A LIFETIME attribute containing the current value of the time-to-expire timer;

*A BANDWIDTH attribute containing the actual bandwidth of the allocation; and

*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.

6.3. Receiving an Allocate Response

TOC

If the client receives a success response, then it MUST check that the relayed transport address is in an address family that the client understands and is prepared to deal with. This specification only covers the case where the relayed transport address is of the same address family as the client's transport address. If the relayed transport address is not in an address family that the client is prepared to deal with, then the client MUST delete the allocation (Section 7 (Refreshing an Allocation)) 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 and the actual bandwith received back from the server, rather than the values 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.

If the client receives an 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 MAY choose to try again using a different transport (e.g., TCP instead of UDP), or the client MAY try a different server.

*400 (Bad Request): The server believes the client's request is malformed for some reason. 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. The client MAY try a different server.

*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. The client SHOULD notify the user or operator and SHOULD NOT send any further requests to this server until it believes the problem has been fixed. The client MAY try a different server.

*437 (Allocation Mismatch): This indicates that the client has picked a 5-tuple which the server sees as already in use or which was recently in use. One way this could happen is if an intervening NAT assigned a mapped transport address that was recently used by another allocation. 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 (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. The client MAY try a different server.

*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 retry the same request with this server until it believes the problem has been fixed. The client MAY try a different server.

*486 (Allocation Quota Reached): The server is currently unable to create any more allocations with this username. The client SHOULD wait at least 1 minute before trying to create any more allocations on the server. The client MAY try a different server.

*507 (Insufficient Bandwidth Capacity): The server is currently unable to allocate any bandwidth to this allocation. The client SHOULD wait at least 1 minute before trying to create any more allocations on the server. The client MAY try a different server.

*508 (Insufficient Port Capacity): The server has no more relayed transport addresses avaiable, or has none with the requested properties, or the one that was reserved is no longer available. If the client is using either the REQUESTED-PROPS or the RESERVATION-TOKEN attribute, then the client MAY choose to remove 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. The client MAY try a different server.

If the error response contains an ALTERNATE-SERVER attribute, and the client elects to try a different server, the the client SHOULD try the alternate server specified in that attribute (while obeying the rules in [I-D.ietf-behave-rfc3489bis] (Rosenberg, J., Mahy, R., Matthews, P., and D. Wing, "Session Traversal Utilities for (NAT) (STUN)," July 2008.) for avoiding redirection loops) before trying any other servers found using the SRV procedures of [I-D.ietf-behave-rfc3489bis] (Rosenberg, J., Mahy, R., Matthews, P., and D. Wing, "Session Traversal Utilities for (NAT) (STUN)," July 2008.).

---

7. Refreshing an Allocation

TOC

A Refresh transaction can be used to either (a) refresh an existing allocation and update its time-to-expire and bandwidth, 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 also change the bandwidth and/or the time-to-expire of an allocation at any time for other reasons.

7.1. Sending a Refresh Request

TOC

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 Refresh transaction updates the time-to-expire timer of an allocation. If the client wishes the server to set the time-to-expire

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-expire 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.
The Refresh transaction also updates the bandwidth of an allocation. If the client wishes the server to update the bandwidth to something other than the mimimum per-allocation bandwidth, it includes the BANDWIDTH attribute with the requested value.
The Refresh transaction is sent on the 5-tuple for the allocation.

7.2. Receiving a Refresh Request

TOC

When the server receives a Refresh request, it processes it as follows. If, during processing, an error in the request is detected (for example, a syntax error in the request which causes a 400 error), then the request is rejected with an error response but the allocation is NOT deleted (but note that a 437 error will indicate that the allocation was not found).
The server determines the new value for the time-to-expire field as follows. If the request contains a LIFETIME attribute, and the attribute value is 0, then the server uses a value of 0, which causes the allocation to expire. Otherwise, if the request contains a LIFETIME attribute and the attribute value is greater than the default lifetime, and the attribute value is otherwise acceptable to the server, then the server uses the attribute value. Otherwise, the server uses the default value. It is RECOMMENDED that the server impose a maximum lifetime of no more than 3600 seconds (1 hour).
Assuming the allocation is not now expired, the server then determines a new value for the bandwidth as follows. If the request contains a BANDWIDTH attribute, or if the requested bandwidth is less than the minimum per-allocation bandwidth, then the server behaves as if the minimum per-allocation bandwidth was requested. Otherwise, if the request bandwidth is greater than the maximum per-allocation bandwidth, then the server behaves as if the maximum per-allocation bandwidth was requested.
The server then compares the requested allocation bandwidth with the current allocation bandwidth. If the requested bandwidth is smaller, the current allocation bandwidth is updated. If the requested bandwidth is larger, then the current allocation bandwidth is increased to either the requested bandwidth or to the maximum currently available, whichever is smaller.
The server then constructs a success response containing:

*A LIFETIME attribute containing the current value of the time-to-expire timer; and

*A BANDWIDTH attribute containing the actual bandwidth of the allocation.

The response is then sent on the 5-tuple.

7.3. Receiving a Refresh Response

TOC

If the client receives a success response to its Refresh request, it updates its copy of the allocation data structure with the bandwidth and time-to-expire values contained in the response.
If the client receives an 437 (Allocation Mismatch) error response to its Refresh request, then it must consider the allocation as having expired, as described in Section 4 (General Behavior). All other errors indicate a software error on the part of either the client or the server.

---

8. Permissions

TOC

For each allocation, the server keeps a list of zero or more permissions. Each permission consists an IP address which uniquely identifies the permission, and an associated time-to-expiry. The IP address describes a peer that is allowed to send data to the client, and the time-to-expiry is the number of seconds until the permission expires.
Various events, as described in subsequent sections, can cause a permission for a given IP address to be installed or refreshed. 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 the default permission lifetime.

*If a permission for that IP address already exists, then the lifetime for that permission is reset to the default permission lifetime.

The default permission lifetime MUST be 300 seconds (= 5 minutes). Each permission’s time-to-expire decreases down once per second until it reaches 0, at which point the permission expires and is deleted.
When a UDP datagram arrives at the relayed transport address for the allocation, the server checks the list of permissions for that allocation. If there is a permission with an IP address that is equal to the source IP address of the UDP datagram, then the UDP datagram can be relayed to the client. Otherwise, the UDP datagram is silently

discarded. Note that only IP addresses are compared; port numbers are irrelevant.
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 [I-D.ietf-mmusic-ice] (Rosenberg, J., "Interactive Connectivity Establishment (ICE): A Protocol for Network Address Translator (NAT) Traversal for Offer/Answer Protocols," October 2007.), and applications not using ICE are advised to do something similar.

---

9. Send and Data Indications

TOC

TURN supports two ways to send and receive data from peers. This section describes the use of Send and Data indications, while Section 10 (Channels) describes the use of the Channel Mechanism.

9.1. Sending a Send Indication

TOC

A client can use a Send Indication to pass data to the server for relaying to a peer. A client can also use a Send Indication without a DATA attribute to install or refresh a permission for the specified IP address. A client may use a Send indication to send data to a peer even if a channel is bound to that peer.
When forming a Send Indication, the client MUST include a PEER-ADDRESS attribute and MAY include a DATA attribute. If the DATA attribute is included, then the DATA attribute contains the actual application data to be sent to the peer, and the PEER-ADDRESS attribute contains the transport address of the peer to which the data is to be sent. If the DATA attribute is not present, then the PEER-ADDRESS attribute contains the IP address for which a permission is to be installed or refreshed; in this case the port specified in the attribute is ignored.
Note that no authentication attributes are included, since indications cannot be authenticated using the Long-Term Credential mechanism.
The Send Indication MUST be sent using the same 5-tuple used for the original allocation.

9.2. Receiving a Send Indication

TOC

When the server receives a Send indication, it processes it as follows. If the received Send indication contains a DATA attribute, then it 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 PEER-ADDRESS attribute;
- the data following the UDP header is the contents of the value field of the DATA attribute.

The resulting UDP datagram is then sent to the peer. If any errors are detected during this process (e.g., the Send indication does not contain a PEER-ADDRESS attribute), the received indication is silently discarded and no UDP datagram is sent.
When the server receives a valid Send Indication, either with or without a DATA attribute, it also installs or refreshes a permission for the IP address contained in the PEER-ADDRESS attribute (see Section 8 (Permissions)).

9.3. Receiving a UDP Datagram

TOC

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. It then checks to see if relaying is permitted, as described in section Section 8 (Permissions)).
If relaying is permitted, and there is no channel bound to the peer that sent the UDP datagram (see ISection 10 (Channels)), then the server forms and sends a Data indication. The Data indication MUST contain both a PEER-ADDRESS and a DATA attribute. The DATA attribute is set to the value of the ‘data octets’ field from the datagram, and the 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.

TOC

9.4. Receiving a Data Indication

When the client receives a Data indication, it checks that the Data indication contains both a PEER-ADDRESS and a DATA attribute. It then 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 PEER-ADDRESS attribute.

---

10. Channels

TOC

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.

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 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 a binding in one allocation has no relationship to a binding in any other 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);
- 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 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 last for 10 minutes unless refreshed. Refreshing the binding (by the server receiving either a ChannelBind request rebinding the channel to the same peer, or by the server receiving a ChannelData message on that channel) resets the time-to-expire timer back to 10 minutes. When the channel binding expires, the channel becomes unbound and available for binding to a different transport address.

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.

10.1. Sending a ChannelBind Request

TOC

A channel binding is created using a ChannelBind transaction. A channel binding can also be refreshed using a ChannelBind transaction.
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 a PEER-ADDRESS attribute. Section 10.2 (Receiving a ChannelBind Request) describes the restrictions on these attributes.
Note that rebinding a channel to the same transport address that it is already bound to provides a way to refresh a channel binding without sending data to the peer.
Once formed, the ChannelBind Request is sent using the 5-tuple for the allocation.

10.2. Receiving a ChannelBind Request

TOC

When the server receives a ChannelBind request, it checks the following:

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

*The channel number is in the range 0x4000 to 0xFFFE (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 any of these tests fail, the server replies with an error response with error code 400 "Bad Request". Otherwise, the ChannelBind request is valid and the server replies with a ChannelBind success response. If ChannelBind request is valid, then the server creates or refreshes the channel binding using the channel number in the CHANNEL-ADDRESS attribute and the transport address in the PEER-ADDRESS attribute. The server also installs or refreshes a permission for the IP address in the PEER-ADDRESS attribute.

10.3. Receiving a ChannelBind Response

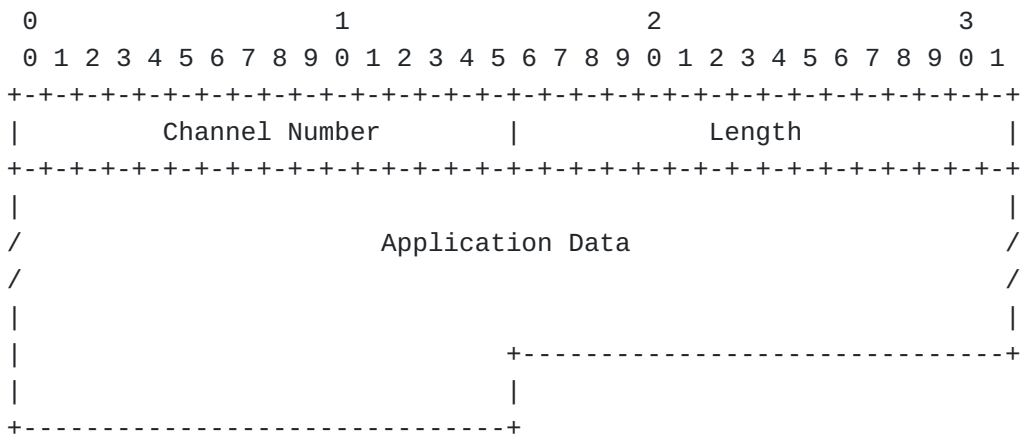
TOC

When the client receives a successful ChannelBind response, it updates its data structures to record that the channel binding is now active.

10.4. The ChannelData Message

TOC

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 channel number MUST be in the range 0x4000 – 0xFFFF, with channel number 0xFFFF being reserved for possible future extensions.
Channel numbers 0x0000 – 0x3FFF cannot be used because bits 0 and 1 are used to distinguish ChannelData messages from STUN-formatted messages (i.e., Allocate, Send, Data, ChannelBind, etc). STUN-formatted messages always have bits 0 and 1 as “00”, while ChannelData messages use combinations “01”, “10”, and “11”.

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.

10.5. Sending a ChannelData Message

TOC

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 10.4 (The ChannelData Message).
Over stream transports, 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.

10.6. Receiving a ChannelData Message

TOC

The receiver of the ChannelData message uses bits 0 and 1 to distinguish it from STUN-formatted messages, as described in Section 10.4 (The ChannelData Message).
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 10.5 (Sending a ChannelData Message).
If the ChannelData message is received on a channel which is not bound to any peer, then the message is silently discarded.

TOC

10.7. Relaying

When a server receives a ChannelData message, it first processes it as described in the previous section. If no errors are detected, it 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.
If the ChannelData message is valid, then the server refreshes the channel binding, and also installs or refreshes a permission for the IP address part of the transport address to which the UDP datagram is sent (see Section 8 (Permissions)).
In the other direction, when the server receives a UDP datagram on the relayed transport address associated with an allocation, then it first checks to see if it is permitted to relay the datagram. This check is done as described in Section 8 (Permissions). If relaying is permitted, then the server checks to see if there is a channel bound to the peer that sent the UDP datagram. If there is, then it SHOULD form and send a ChannelData message as described in Section 10.5 (Sending a ChannelData Message). If no channel is bound to the peer, then it MUST form and send a Data indication as described in Section 9.3 (Receiving a UDP Datagram).

---

11. IP Header Fields and Path MTU

TOC

This section describes how the server should set various fields in the IP header when relaying application data. The requirements here document the desired behavior of the server, but it is recognized that some of these requirements may be impossible to implement in certain environments.

> NOTE: The recommendations in this section are the result of much discussion, and are a compromise between the perfect relaying solution and one that can be implemented easily. In particular, these recommendations takes into account the following:
>
> - TURN allows a TCP, or a TLS over TCP, connection between the client and the server, while using a UDP connection between

the server and a peer. For this reason, the notion of a single end-to-end connection does not always exist.

*Many people want to run a TURN server as a process in user-space under common operating systems, without requiring the server process to have special privileges (such as those required to use RAW sockets). One motivation for this is the desire to implement a TURN server in a peer application in a peer-to-peer overlay to provide relaying functions to other peers which reside behind 'bad' NATs; such applications are often downloaded by users with very little knowledge of computers and networking.

*A process in user-space under many common operating systems is rather restricted in which fields in the IP header it can set and (even worse) read.

*TURN is the relay solution of last resort. It is intended to be used only when a direct connection between the TURN client and the peer cannot be established.

11.1. DiffServ Code Point (DSCP)

TOC

If the client-server connection uses UDP, then the server SHOULD read the DSCP from the IP header of the received Data indication or ChannelData message and use that DSCP for the corresponding outgoing UDP datagram. In the reverse direction, the server SHOULD read the DSCP from the arriving UDP datagram and use that DSCP for the corresponding outgoing Data indication or ChannelData message.
If the client-server connection uses TCP (or TLS over TCP), then to the extent possible, the server SHOULD read the DSCP from the TCP connection whenever it reads a Data indication or a ChannelData message from the TCP socket, and use that DSCP for the corresponding outgoing UDP datagram. In the reverse direction, the server SHOULD read the DSCP from the IP header of the received UDP datagram, and set the DSCP of the TCP connection to the same value.
If, for efficiency or other reasons, the server is unable to read the DSCP for every message, then it SHOULD read these values at frequent intervals and use the DSCP learned for all outgoing packets (in the appropriate direction and on this allocation) until the next time it reads the DSCP.

NOTE: By copying the DSCP, the server ensures that the application data gets consistent QoS treatment along the entire path from the client to the peer.

11.2. Don't Fragment (DF) bit

TOC

When the client sends a Data indication or ChannelData message to the server using UDP IPv4, it SHOULD NOT set the DF (Don't Fragment) bit unless the application explicitly requests the bit to be set.
When the server sends a UDP datagram to a peer over IPv4, or when sends a Data indication or a ChannelData message to the client using UDP over IPv4, the server SHOULD NOT set the DF bit.
When using TCP or TLS over TCP, the client and the server MAY let the setting of the DF bit be determined by the TCP/IP stack.

> NOTE: By not setting the DF bit over UDP, the server maximizes the chances that the UDP datagram, Data indication, or ChannelData message will be delivered. This is consistent with the view that TURN is a relay solution of last resort.

11.3. Other IP Header Fields

TOC

The server SHOULD NOT preserve the ECN (Explicit Congestion Notification) field, and MAY preserve thee TTL (Time-To-Live) fields when relaying application data.

> NOTE: The ECN field is not preserved because the view is that there are two connections here: one between the client and the server, and a second between the server and a peer. For example, if the client-server connection uses TCP, then the ECN field conveys useful information between the two TCP stacks, but is meaningless outside that TCP connection.
>
> The TTL field need not be preserved because there seems to be little chance of a forwarding loop, and because reading the TTL field is impossible without using RAW sockets in most situations.

11.4. Path MTU

TOC

Applications using TURN SHOULD follow the guidelines in [I-D.ietf-tsvwg-udp-guidelines] (Eggert, L. and G. Fairhurst, “Unicast UDP Usage Guidelines for Application Designers,” October 2008.), but use the algorithm of [RFC4821] (Mathis, M. and J. Heffner,

"Packetization Layer Path MTU Discovery," March 2007.) rather than the algorithm of [RFC1191] (Mogul, J. and S. Deering, "Path MTU discovery," November 1990.) to determine the Path MTU. This algorithm should be run at the application level (and not at the TURN layer or below) and used to discovery the maximum size of a application PDU that can be successfully delivered to the far end application.

> NOTE: According to [I-D.ietf-tsvwg-udp-guidelines] (Eggert, L. and G. Fairhurst, "Unicast UDP Usage Guidelines for Application Designers," October 2008.), applications using UDP should do Path MTU Discovery. If they do not do Path MTU Discovery, then they must restrict their packet size to 576 (over IPv4) or 1280 (over IPv6).
>
> The original Path MTU Discovery algorithm [RFC1191] (Mogul, J. and S. Deering, "Path MTU discovery," November 1990.) will not work because a TURN server does not relay ICMP packets.
>
> The Path MTU Discover algorithm described in [RFC4821] (Mathis, M. and J. Heffner, "Packetization Layer Path MTU Discovery," March 2007.) will work. However, when run over a path that goes through a TURN server, it will not discover the Path MTU (because the DF bit is not set by the server), but intead will discover the maximum size of an application PDU that can be delivered between the client and the peer. Applications that limit themselves to this discovered size WILL be able to communicate effectively, though the application PDU may end up being fragmented on the section of the path after the server.
>
> Applications that instead restrict their packet size to 576 or 1280 may suffer from the fact that TURN adds some overhead between the client and the server. Thus in some situations, these applications will see their maximum-sized packets dropped. However, this overhead is only 4 bytes when channels are used, so the chances of this happening are small.

---

12. New STUN Methods

TOC

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.

```
Request/Response Transactions
  0x003  :  Allocate
  0x004  :  Refresh
  0x009  :  ChannelBind

Indications
  0x006  :  Send
  0x007  :  Data
```

---

13. New STUN Attributes

TOC

This STUN extension defines the following new attributes:

```
0x000C: CHANNEL-NUMBER
0x000D: LIFETIME
0x0010: BANDWIDTH
0x0012: PEER-ADDRESS
0x0013: DATA
0x0016: RELAY-ADDRESS
0x0018: REQUESTED-PROPS
0x0019: REQUESTED-TRANSPORT
0x0022: RESERVATION-TOKEN
```

13.1. CHANNEL-NUMBER

TOC

The CHANNEL-NUMBER attribute contains the number of the channel. It is 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 ignored on reception.

```
 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
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|         Channel Number        |         Reserved = 0          |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

TOC

13.2. LIFETIME

The lifetime attribute represents the duration for which the server will maintain an allocation in the absence of a refresh. It is a 32-bit unsigned integral value representing the number of seconds remaining until expiration.

13.3. BANDWIDTH

TOC

The bandwidth attribute represents the peak bandwidth that the client expects to use on the client to server connection. It is a 32-bit unsigned integral value and is measured in kilobits per second.

13.4. PEER-ADDRESS

TOC

The PEER-ADDRESS specifies the address and port of the peer as seen from the TURN server. It is encoded in the same way as XOR-MAPPED-ADDRESS.

13.5. DATA

TOC

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

13.6. RELAY-ADDRESS

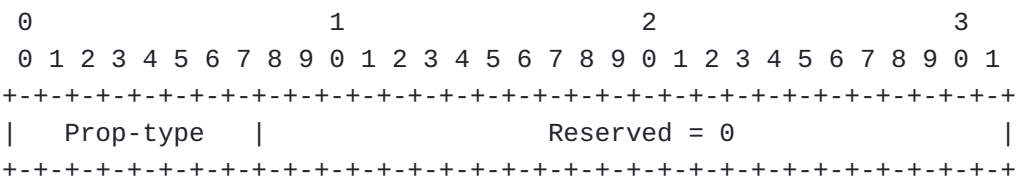
TOC

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

TOC

13.7. REQUESTED-PROPS

This attribute allows the client to request certain properties for the relayed transport address that is allocated by the server. The attribute is 32 bits long. Its format is:



The field labeled "Prop-type" is an 8-bit field specifying the desired property. The rest of the attribute is RFFU (Reserved For Future Use) and MUST be set to 0 on transmission and ignored on reception. The values of the "Prop-type" field are:

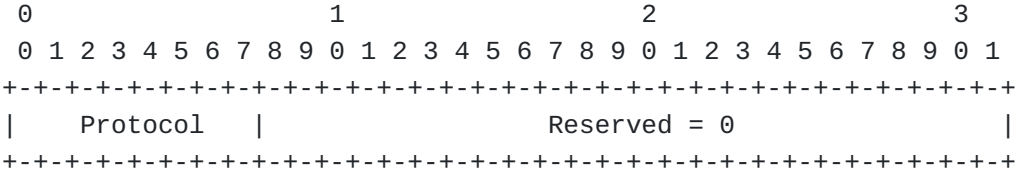
0x00 (Reserved)
0x01 Even port number
0x02 Pair of ports

If the value of the "Prop-type" field is 0x01, then the client is requesting the server allocate an even-numbered port for the relayed transport address.
If the value of the "Prop-type" field is 0x02, then client is requesting the server allocate an even-numbered port for the relayed transport address, and in addition reserve the next-highest port for a subsequent allocation.
All other values of the "Prop-type" field are reserved.

13.8. REQUESTED-TRANSPORT

TOC

This attribute is used by the client to request a specific transport protocol for the allocated transport address. It has the following format:



The Protocol field specifies the desired protocol. The codepoints used in this field are taken from those allowed in the Protocol field in the IPv4 header and the NextHeader field in the IPv6 header [Protocol-Numbers] (, “IANA Protocol Numbers Registry,” 2005.). This

specification only allows the use of codepoint 17 (User Datagram Protocol).
The RFFU field is set to zero on transmission and ignored on receiption. It is reserved for future uses.

13.9. RESERVATION-TOKEN

TOC

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 a 64-bit-long field containing the token value.

---

14. New STUN Error Response Codes

TOC

This document defines the following new error response codes:

437 (Allocation Mismatch): A request was received by the server that requires an allocation to be in place, but there is none, or a request was received which requires no allocation, but there is one.

438 (Wrong Credentials): The credentials in the (non-Allocate) request, though otherwise acceptable to the server, 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.

486 (Allocation Quota Reached): No more allocations using this username can be created at the present time.

507 (Insufficient Bandwidth Capacity): The server cannot create an allocation with the requested bandwidth right now as it has exhausted its capacity.

508 (Insufficient Port Capacity): The server has no more relayed transport addresses available right now, or has none with the requested properties, or the one that corresponds to the specified token is not available.

15. Security Considerations

TOC

TURN servers allocate bandwidth and port resources to clients, in contrast to the Binding method defined in [I-D.ietf-behave-rfc3489bis] (Rosenberg, J., Mahy, R., Matthews, P., and D. Wing, "Session Traversal Utilities for (NAT) (STUN)," July 2008.). Therefore, a TURN server may require the authentication and authorization of STUN requests. This authentication is provided by mechanisms defined in the STUN specification itself, in particular digest authentication.

Because TURN servers allocate resources, they can be susceptible to denial-of-service attacks. All Allocate transactions are authenticated, so that an unknown attacker cannot launch an attack. An authenticated attacker can generate multiple Allocate Requests, however. To prevent a single malicious user from allocating all of the resources on the server, it is RECOMMENDED that a server implement a modest per user limit on the amount of bandwidth that can be allocated. Such a mechanism does not prevent a large number of malicious users from each requesting a small number of allocations. Attacks such as these are possible using botnets, and are difficult to detect and prevent. Implementors of TURN should keep up with best practices around detection of anomalous botnet attacks.

A client will use the transport address learned from the RELAY-ADDRESS attribute of the Allocate Response to tell other users how to reach them. Therefore, a client needs to be certain that this address is valid, and will actually route to them. Such validation occurs through the message integrity checks provided in the Allocate response. They can guarantee the authenticity and integrity of the allocated addresses. Note that TURN is not susceptible to the attacks described in Section 12.2.3, 12.2.4, 12.2.5 or 12.2.6 of [I-D.ietf-behave-rfc3489bis] (Rosenberg, J., Mahy, R., Matthews, P., and D. Wing, "Session Traversal Utilities for (NAT) (STUN)," July 2008.) [[TODO: Update section number references to 3489bis]]. These attacks are based on the fact that a STUN server mirrors the source IP address, which cannot be authenticated. STUN does not use the source address of the Allocate Request in providing the RELAY-ADDRESS, and therefore, those attacks do not apply.

TURN attempts to adhere as closely as possible to common firewall policies, consistent with allowing data to flow. TURN has fairly limited applicability, requiring a user to explicitly authorize permission to receive data from a peer, one IP address at a time. Thus, it does not provide a general technique for externalizing sockets. Rather, it has similar security properties to the placement of an address-restricted NAT in the network, allowing messaging in from a peer only if the internal client has sent a packet out towards the IP address of that peer. This limitation means that TURN cannot be used to

run, for example, SIP servers, NTP servers, FTP servers or other network servers that service a large number of clients. Rather, it facilitates rendezvous of NATted clients that use some other protocol, such as SIP, to communicate IP addresses and ports for communications. Confidentiality of the transport addresses learned through Allocate transactions does not appear to be that important. If required, it can be provided by running TURN over TLS.

TURN does not and cannot guarantee that UDP data is delivered in sequence or to the correct address. As most TURN clients will only communicate with a single peer, the use of a single channel number will be very common. Consider an enterprise where Alice and Bob are involved in separate calls through the enterprise NAT to their corporate TURN server. If the corporate NAT reboots, it is possible that Bob will obtain the exact NAT binding originally used by Alice. If Alice and Bob were using identical channel numbers, Bob will receive unencapsulated data intended for Alice and will send data accidentally to Alice's peer. This is not a problem with TURN. This is precisely what would happen if there was no TURN server and Bob and Alice instead provided a (STUN) reflexive transport address to their peers. If detecting this misdelivery is a problem, the client and its peer need to use message integrity on their data.

Relay servers are useful even for users not behind a NAT. They can provide a way for truly anonymous communications. A user can cause a call to have its media routed through a TURN server, so that the user's IP addresses are never revealed.

Any relay addresses learned through an Allocate request will not operate properly with IPSec Authentication Header (AH) (Kent, S., "IP Authentication Header," December 2005.) [RFC4302] in transport or tunnel mode. However, tunnel-mode IPSec ESP (Kent, S., "IP Encapsulating Security Payload (ESP)," December 2005.) [RFC4303] should still operate.

---

16. IANA Considerations

TOC

Since TURN is an extension to STUN [I-D.ietf-behave-rfc3489bis] (Rosenberg, J., Mahy, R., Matthews, P., and D. Wing, "Session Traversal Utilities for (NAT) (STUN)," July 2008.), the methods, attributes and error codes defined in this specification are new method, attributes, and error codes for STUN. This section directs IANA to add 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 12 (New STUN Methods).

The codepoints for the new STUN attributes defined in this specification are listed in Section 13 (New STUN Attributes).

The codepoints for the new STUN error codes defined in this specification are listed in Section 14 (New STUN Error Response Codes).

Extensions to TURN can be made through IETF consensus.

---

17. IAB Considerations

TOC

The IAB has studied the problem of "Unilateral Self Address Fixing", which is the general process by which a client attempts to determine its address in another realm on the other side of a NAT through a collaborative protocol reflection mechanism [RFC3424] (Daigle, L. and IAB, “IAB Considerations for UNilateral Self-Address Fixing (UNSAF) Across Network Address Translation,” November 2002.). 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.
TURN is an extension of the STUN protocol. As such, the specific usages of STUN that use the TURN extensions need to specifically address these considerations. Currently the only STUN usage that uses TURN is ICE (Rosenberg, J., “Interactive Connectivity Establishment (ICE): A Protocol for Network Address Translator (NAT) Traversal for Offer/ Answer Protocols,” October 2007.) [I-D.ietf-mmusic-ice].

---

18. Example

TOC

TBD

---

19. Changes from Previous Versions

TOC

Note to RFC Editor: Please remove this section prior to publication of this document as an RFC.
This section lists the changes between the various versions of this specification.

19.1. Changes from -06 to -07

TOC

*Rewrote the General Behavior section, making various changes in the process.

*Changed the usage of authentication from MUST to SHOULD.

*Changed the requirement that subsequent requests use the same username and password from MUST to SHOULD to allow for the possibility of changing the credentials using some unspecified mechanism.

*Introduced a 438 (Wrong Credentials) error which is used when a non-Allocate request authenticates but does not use the same username and password as the Allocate request. Having a separate error code for this case avoids the client being confused over what the error actually is.

*The server must now prevent the relayed transport address and the 5-tuple from being reused in different allocations for 2 minutes after the allocation expires.

*Changed the usage of FINGERPRINT from MUST NOT to MAY, to allow for the possible multiplexing of TURN with some other protocol.

*Rewrote much of the section on Allocations, splitting it into three new sections (one on allocations in general, one on creating an allocation, and one on refreshing an allocation).

*Replaced the mechanism for requesting relayed transport addresses with specific properties. The new mechanism is less powerful: a client can request an even port, or a pair of ports, but cannot request a single odd port or a specific port as was possible under the old mechanism. Nor can the client request a specific IP address.

*Changed the rules for handling ALTERNATE-SERVER, removing the requirement that the referring server have "positive knowledge" about the state of the alternate server. The new rules instead rely on text in STUN to prevent referral loops.

*Changed the rules for allocation lifetimes. Allocations lifetimes are now a minimum of 10 minutes; the client can ask for longer values, but requests for shorter values are ignored. The text now recommends that the client refresh an allocation one minute before it expires.

*Put in temporary procedures for handling the BANDWIDTH attribute, modelled on the LIFETIME attribute. These procedures are mostly placeholders and likely to change in the next revision.

*Added a detailed description of how a client reacts to the various errors it can receive in reply to an Allocate request. This replaces the various descriptions that were previously scattered throughout the document, which were inconsistent and sometimes contradictory.

*Added a new section that gives the normative rules for permissions.

*Changed the rules around permission lifetimes. The text used to recommend a value of one minute; it MUST now be 5 minutes.

*Removed the errors "Channel Missing or Invalid", "Peer Address Missing or Invalid" and "Lifetime Malformed or Invalid" and used 400 "Bad Request" instead.

*Rewrote portions of the section on Send and Data indications and the section on Channels to try to make the client vs. server behavior clearer.

*Channel bindings now expire after 10 minutes, and must be refreshed to keep them alive.

*Binding a channel now installs or refreshes a permission for the IP address of corresponding peer.

*Changed the wording describing the situation when the client sends a ChannelData message before receiving the ChannelBind success response. -06 said that client SHOULD NOT do this; -07 now says that a client MAY, but describes the consequences of doing it.

*Added a section discussing the setting of fields in the IP header.

*Replaced the REQUESTED-PORT-PROPS attribute with the REQUESTED-PROPS attribute that has a different format and semantics, but reuses the same code point.

*Replaced the REQUESTED-IP attribute with the RESERVATION-TOKEN attribute, which has a different format and semantics, but reuses the same code point.

*Removed error codes 443 and 444, and replaced them with 508 (Insufficient Port Capacity). Also changed the error text for code 507 from "Insufficient Capacity" to "Insufficient Bandwidth Capacity".

19.2. Changes from -05 to -06

TOC

*Changed the mechanism for allocating channels to the one proposed by Eric Rescorla at the Dec 2007 IETF meeting.

*Removed the framing mechanism (which was used to frame all messages) and replaced it with the ChannelData message. As part of this change, noted that the demux of ChannelData messages from TURN messages can be done using the first two bits of the message.

*Rewrote the sections on transmitted and receiving data as a result of the above to changes, splitting it into a section on Send and Data Indications and a separate section on channels.

*Clarified the handling of Allocate Request messages. In particular, subsequent Allocate Request messages over UDP with the same transaction id are not an error but a retransmission.

*Restricted the range of ports available for allocation to the Dynamic and/or Private Port range, and noted when ports outside this range can be used.

*Changed the format of the REQUESTED-TRANSPORT attribute. The previous version used 00 for UDP and 01 for TCP; the new version uses protocol numbers from the IANA protocol number registry. The format of the attribute also changed.

*Made a large number of changes to the non-normative portion of the document to reflect technical changes and improve the presentation.

*Added the Issues section.

19.3. Changes from -04 to -05

TOC

*Removed the ability to allocate addresses for TCP relaying. This is now covered in a separate document. However, communication between the client and the server can still run over TCP or TLS/TCP. This resulted in the removal of the Connect method and the TIMER-VAL and CONNECT-STAT attributes.

*Added the concept of channels. All communication between the client and the server flows on a channel. Channels are numbered 0..65535. Channel 0 is used for TURN messages, while the remaining channels are used for sending unencapsulated data to/from a remote peer. This concept adds a new Channel Confirmation method and a new CHANNEL-NUMBER attribute. The new attribute is also used in the Send and Data methods.

*The framing mechanism formally used just for stream-oriented transports is now also used for UDP, and the former Type and Reserved fields in the header have been replaced by a Channel Number field. The length field is zero when running over UDP.

*TURN now runs on its own port, rather than using the STUN port. The use of channels requires this.

*Removed the SetActiveDestination concept. This has been replaced by the concept of channels.

*Changed the allocation refresh mechanism. The new mechanism uses a new Refresh method, rather than repeating the Allocation transaction.

*Changed the syntax of SRV requests for secure transport. The new syntax is "_turns._tcp" rather than the old "_turn._tls". This change mirrors the corresponding change in STUN SRV syntax.

*Renamed the old REMOTE-ADDRESS attribute to PEER-ADDRESS, and changed it to use the XOR-MAPPED-ADDRESS format.

*Changed the RELAY-ADDRESS attribute to use the XOR-MAPPED-ADDRESS format (instead of the MAPPED-ADDRESS format)).

*Renamed the 437 error code from "No Binding" to "Allocation Mismatch".

*Added a discussion of what happens if a client's public binding on its outermost NAT changes.

*The document now consistently uses the term "peer" as the name of a remote endpoint with which the client wishes to communicate.

*Rewrote much of the document to describe the new concepts. At the same time, tried to make the presentation clearer and less repetitive.

---

20. Open Issues

TOC

NOTE to RFC Editor: Please remove this section prior to publication of this document as an RFC.
Bandwidth: How should bandwidth be specified? What are the right rules around bandwidth?
Alternate Server: Do we still want this mechanism? Is the current proposal acceptable? Note that the usage of the ALTERNATE-SERVER

attribute in this document is inconsistent with its usage in STUN. In STUN, if the ALTERNATE-SERVER attribute is used, then the error that the server would otherwise generate is replaced by a 300 (Try Alternate) code. In this document, the 300 error code is not used, and the server returns an appropriate error code and then includes the ALTERNATE-SERVER attribute in the response. In this way, the client can see the actual error code, rather than always seeing error code 300, and can thus make a more intelligent decision on whether it wishes to try the alternate server.
Public TURN servers: The text currently says that a server "SHOULD" use the Long-Term Credential mechanism, with the unstated idea that a public TURN server would not use it. But this really weakens the security of TURN. Is there a better way to allow public servers? Or should we just drop the notion of a public server entirely?

---

21. Acknowledgements

TOC

The authors would like to thank the various participants in the BEHAVE working group for their many comments on this draft. Marc Petit-Huguenin, Remi Denis-Courmont, Derek MacDonald, Cullen Jennings, Lars Eggert, Magnus Westerlund, and Eric Rescorla have been particularly helpful, with Eric also suggesting the channel allocation mechanism, and Cullen suggesting the REQUESTED-PROPS mechanism. Christian Huitema was an early contributor to this document and was a co-author on the first few drafts. Finally, the authors would like to thank Dan Wing for both his contributions to the text and his huge help in restarting progress on this draft after work had stalled.

---

22. References

TOC

22.1. Normative References

TOC

[I-D.ietf-behave-rfc3489bis]	Rosenberg, J., Mahy, R., Matthews, P., and D. Wing, “Session Traversal Utilities for (NAT) (STUN),” draft-ietf-behave-rfc3489bis-18 (work in progress), July 2008 (TXT).
[RFC2119]	Bradner, S., “Key words for use in RFCs to Indicate Requirement Levels,” BCP 14, RFC 2119, March 1997 (TXT, HTML, XML).

22.2. Informative References

TOC

[RFC1918]	Rekhter, Y., Moskowitz, R., Karrenberg, D., Groot, G., and E. Lear, “Address Allocation for Private Internets,” BCP 5, RFC 1918, February 1996 (TXT).
[RFC3264]	Rosenberg, J. and H. Schulzrinne, “An Offer/Answer Model with Session Description Protocol (SDP),” RFC 3264, June 2002 (TXT).
[RFC4302]	Kent, S., “IP Authentication Header,” RFC 4302, December 2005 (TXT).
[RFC4303]	Kent, S., “IP Encapsulating Security Payload (ESP),” RFC 4303, December 2005 (TXT).
[RFC3424]	Daigle, L. and IAB, “IAB Considerations for UNilateral Self-Address Fixing (UNSAF) Across Network Address Translation,” RFC 3424, November 2002 (TXT).
[I-D.ietf-mmusic-ice]	Rosenberg, J., “Interactive Connectivity Establishment (ICE): A Protocol for Network Address Translator (NAT) Traversal for Offer/Answer Protocols,” draft-ietf-mmusic-ice-19 (work in progress), October 2007 (TXT).
[RFC4787]	Audet, F. and C. Jennings, “Network Address Translation (NAT) Behavioral Requirements for Unicast UDP,” BCP 127, RFC 4787, January 2007 (TXT).
[I-D.ietf-behave-turn-tcp]	Perreault, S. and J. Rosenberg, “Traversal Using Relays around NAT (TURN) Extensions for TCP Allocations,” draft-ietf-behave-turn-tcp-06 (work in progress), March 2010 (TXT).
[I-D.ietf-behave-turn-ipv6]	Camarillo, G., Novo, O., and S. Perreault, “Traversal Using Relays around NAT (TURN) Extension for IPv6,” draft-ietf-behave-turn-ipv6-09 (work in progress), March 2010 (TXT).
[I-D.ietf-tsvwg-udp-guidelines]	Eggert, L. and G. Fairhurst, “Unicast UDP Usage Guidelines for Application Designers,” draft-ietf-tsvwg-udp-guidelines-11 (work in progress), October 2008 (TXT).
[RFC1191]	Mogul, J. and S. Deering, “Path MTU discovery,” RFC 1191, November 1990 (TXT).
[RFC4821]	Mathis, M. and J. Heffner, “Packetization Layer Path MTU Discovery,” RFC 4821, March 2007 (TXT).
[RFC1928]	Leech, M., Ganis, M., Lee, Y., Kuris, R., Koblas, D., and L. Jones, “SOCKS Protocol Version 5,” RFC 1928, March 1996 (TXT).
[Port-Numbers]	“IANA Port Numbers Registry.”
[Protocol-Numbers]	“IANA Protocol Numbers Registry,” 2005.

Authors' Addresses

TOC

	Jonathan Rosenberg
	Cisco Systems, Inc.
	Edison, NJ
	USA
Email:	jdrosen@cisco.com
URI:	http://www.jdrosen.net
	Rohan Mahy
	Plantronics, Inc.
Email:	rohan@ekabal.com
	Philip Matthews
	Avaya, Inc.
	1135 Innovation Drive
	Ottawa, Ontario K2K 3G7
	Canada
Phone:	+1 613-592-4343 x224
Fax:	
Email:	philip_matthews@magma.ca
URI:	

Full Copyright Statement

TOC

Intellectual Property

procedures with respect to rights in RFC documents can be found in BCP 78 and BCP 79.

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

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