Behave
Internet-Draft
Intended status: Standards Track
Expires: July 25, 2008

J. Rosenberg
Cisco
R. Mahy
Plantronics
P. Matthews
Avaya

January 22, 2008

Traversal Using Relays around NAT (TURN): Relay Extensions to Session Traversal Utilities for NAT (STUN)
draft-ietf-behave-turn-06

Status of this Memo

Copyright Notice

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

NOTE TO THE READER: This document is a work-in-progress. Please see the list of open and closed issues in Section 17. With only a few exceptions, if there is an open issue the text has NOT been updated in this area pending resolution of this issue - keep this in mind when reading the text. In addition, in the interest of getting the document out quickly in order to make progress on open issues, the authors have elected to release the document is a bit more "raw" state than they would prefer, resulting in some rough spots in the presentation.

Session Traversal Utilities for NAT (STUN) [I-D.ietf-behave-rfc3489bis] 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] 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 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 are from IP addresses belonging to 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 an address on the TURN server, so that the TURN server acts as a relay. This extension defines a handful of new STUN methods. The Allocate method is the most fundamental component of this set of extensions. It is used to provide the client with a transport address that is relayed through the TURN server. A transport address which relays through an intermediary is called a relayed transport address.

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

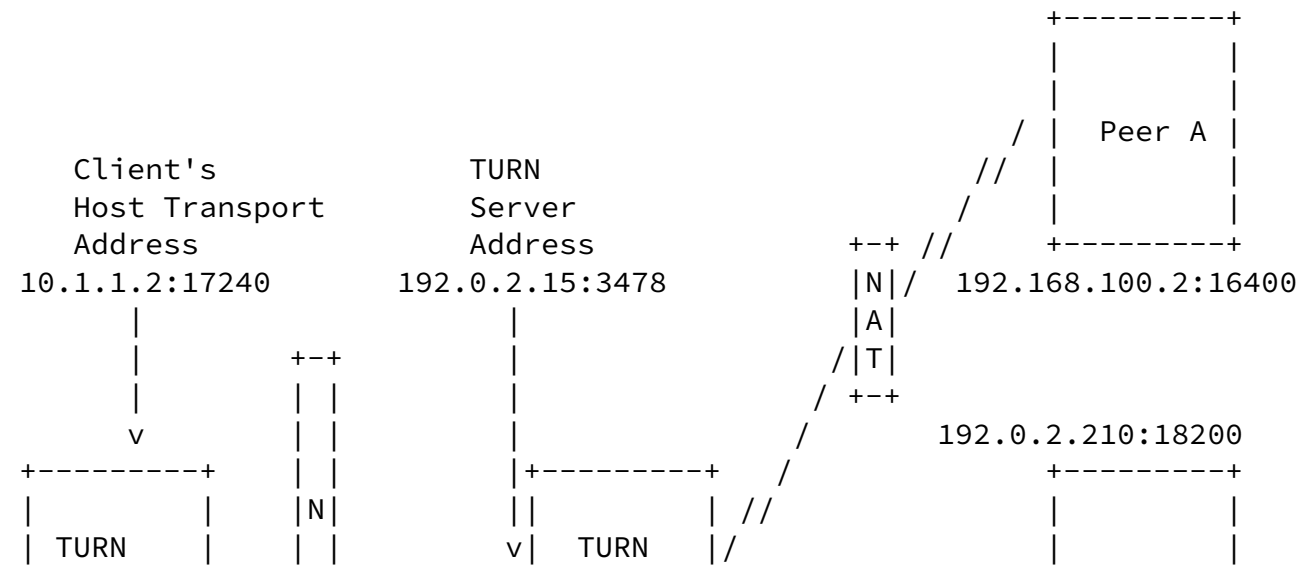
should only be used as a last resort. Protocols using relayed transport addresses should make use of mechanisms to dynamically determine whether such an address is actually needed. One such mechanism, defined for multimedia session establishment protocols based on the offer/answer protocol in RFC 3264 [RFC3264], is Interactive Connectivity Establishment (ICE) [I-D.ietf-mmusic-ice].

Though originally invented for Voice over IP applications, TURN is designed to be a general-purpose relay mechanism for NAT traversal.

2. Overview of Operation

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

In a typical configuration, a TURN client is connected to a private network [RFC1918] and through one or more NATs to the public Internet. On the public Internet is a TURN server. Elsewhere in the Internet are one or more peers that the TURN client wishes to communicate with. These peers may or may not be behind one or more NATs.


Client's
Host Transport
Address
10.1.1.2:17240
TURN
Server
Address
192.0.2.15:3478
Peer A
192.168.100.2:16400
192.0.2.210:18200
TURN
TURN

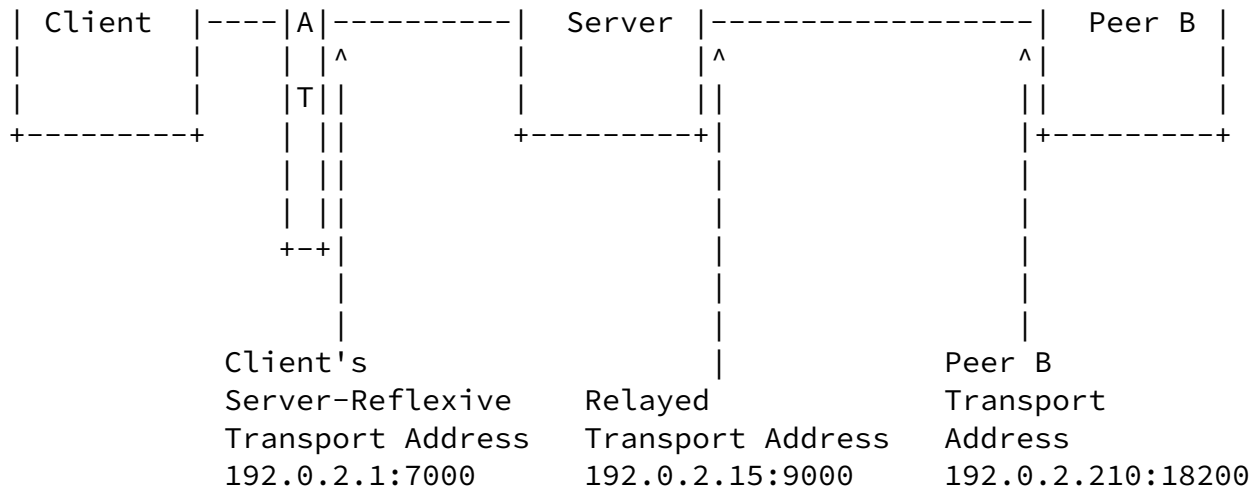


Figure 1

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]) 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 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]). Once this is done, peers can send data packets to the relayed transport address and the server will forward them to the client. In the reverse direction, the client can send data packets to the server (at its TURN server address) and these will be forwarded by the server to the appropriate peer, and the peer will see them as coming from the relayed transport address; in this direction, the client must specify the appropriate peer.

2.1. Transports

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

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]. 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

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 may require 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. This is done by the client periodically doing a Refresh transaction with the server, where the client includes the allocated relayed transport address in the Refresh Request. 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 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

The client can use the relayed transport address to exchange data with its peers by using Send and Data indications. A Send Indication is sent from a client to the TURN server and contains, in attributes inside the message, 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.

Note that a client can use a single relayed transport address to exchange data with multiple peers at the same time.

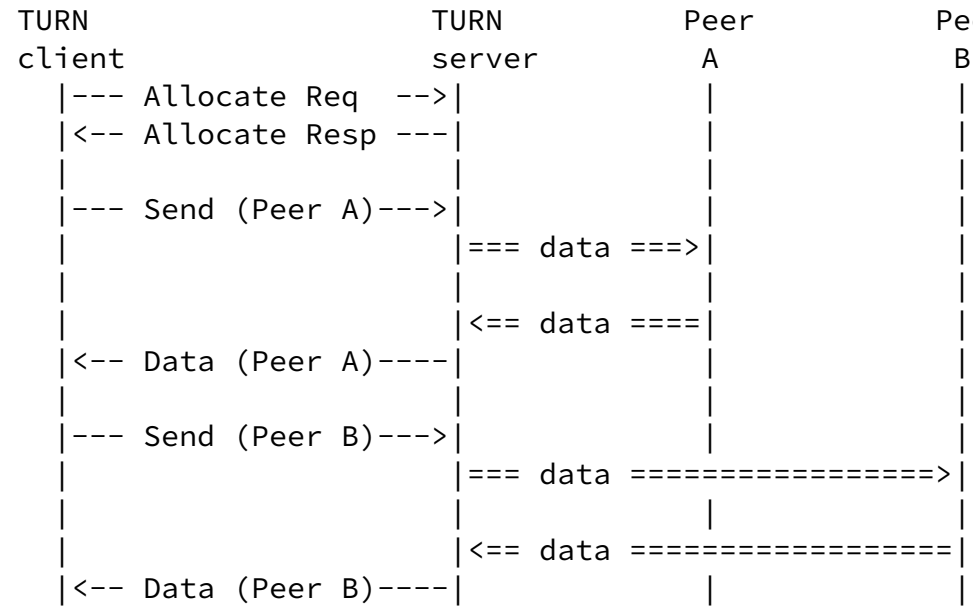


Figure 2

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. Permissions

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

A TURN server will drop a UDP datagram arriving at a relayed transport address from a peer unless the client has recently sent data to a peer with the same IP address (the port numbers can differ). See the normative description for the precise definition of "recently".

A permission will timeout if not refreshed periodically. The client refreshes a permission by sending data to the corresponding peer. Data received from the peer DOES NOT refresh the permission.

2.5. Channels

In some applications, the overhead of using Send and Data indications can be substantial. For example, for applications like VoIP which utilize small packets, Send and Data Indications, with 36 bytes of overhead, can have a substantial impact on overall bandwidth usage. To remedy this, TURN clients can assign a CHANNEL to a peer. Data to and from such a peer can then be sent using an alternate packet format that adds only 4 bytes per packet of overhead.

The alternate packet format is 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.

To create a channel, the client sends a ChannelBind request to the server, and includes an unallocated channel number and the transport address of the peer. Once the client receives the response to the

ChannelBind request, it can send data to that peer using a ChannelData message. Similarly, once the server has received the request, it can relay data from that peer towards the client using a ChannelData message. There is no way to modify channel bindings, so once a channel is bound to a peer, it remains bound for the lifetime of the allocation.

When the server receives a ChannelData message from the client, it uses the channel number to determine the destination peer and then

forwards the data inside a UDP datagram to the peer. In the reverse direction, when a UDP datagram arives at the relayed transport address from that peer, the server inserts it into a ChannelData message containing the channel number bound to that peer; in this way the client can determine the peer that send the UDP datagram.

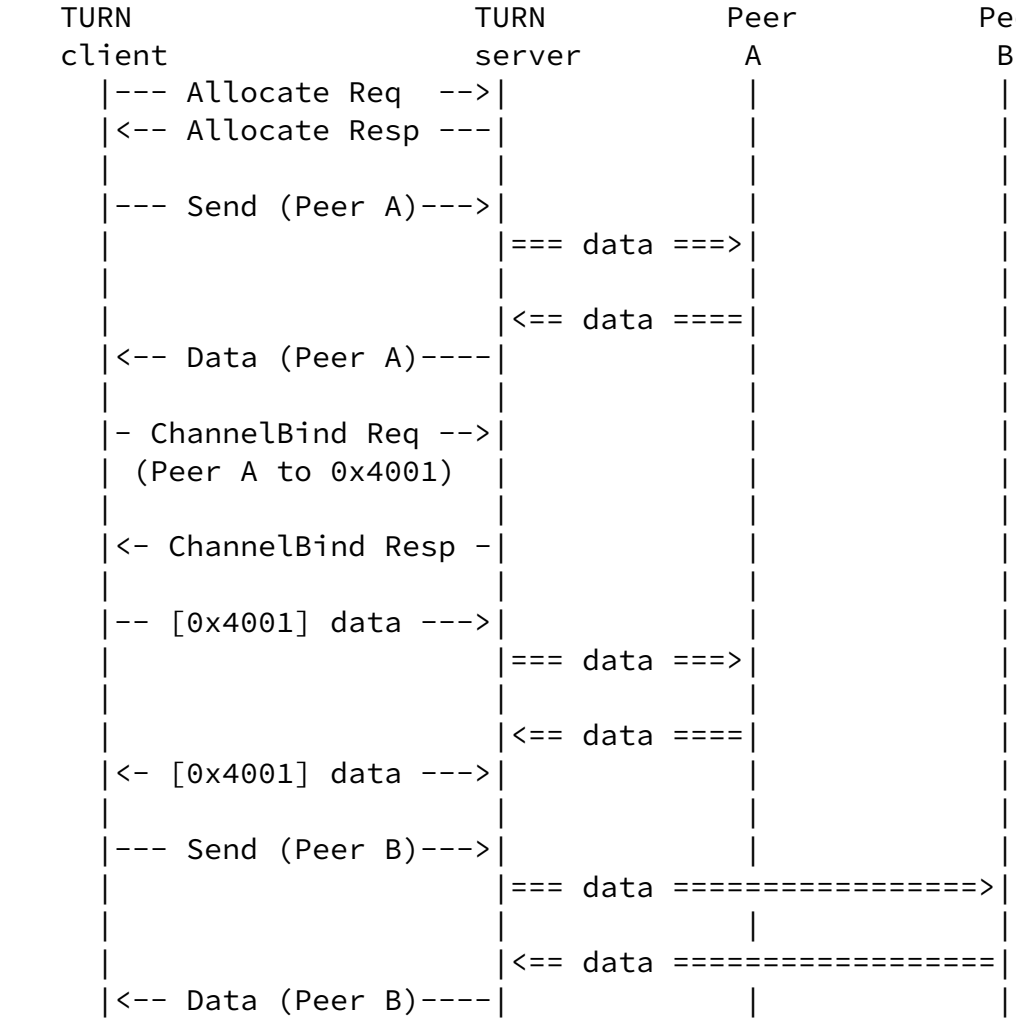


Figure 3

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.

3. Terminology

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

Readers are expected to be familar with [I-D.ietf-behave-rfc3489bis] 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]. 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 transport address granted to a client through an Allocate request, along with related state, such as permissions and expiration timers. See also Relayed Transport Address.

5-tuple: A combination of the source IP address and port, destination IP address and port, and transport protocol (UDP or TCP). A 5-tuple uniquely identifies a TCP connection or the bi-directional flow of UDP datagrams.

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

After the initial Allocate transaction, all subsequent TURN transactions need to be sent in the context of a valid allocation. The source and destination IP address and ports for these TURN messages MUST match the those used in the initial Allocate Request.

These are processed using the general server procedures in [I-D.ietf-behave-rfc3489bis] with a few important additions. For requests, if there is no matching allocation, the server MUST generate a 437 (Allocation Mismatch) error response. For indications, if there is no matching allocation, the indication is silently discarded. An Allocate request MUST NOT be sent by a client within the context of an existing allocation. Such a request MUST be rejected by the server with a 437 (Allocation Mismatch) error response.

A subsequent request MUST be authenticated using the same username, password and realm as the one used in the Allocate request that created the allocation. If the request was authenticated but not with the matching credential, the server MUST reject the request with a 401 (Unauthorized) error response.

When a server returns an error response, it MAY include an ALTERNATE-SERVER attribute if it has positive knowledge that the problem reported in the error response will not be a problem on the alternate server. For example, a 443 response (Invalid IP Address) with an ALTERNATE-SERVER means that the other server is responsible for that IP address. A 442 (Unsupported Transport Protocol) with this attribute means that the other server is known to support that transport protocol. A 507 (Insufficient Capacity) means that the other server is known to have sufficient capacity. Using the ALTERNATE-SERVER mechanism in the 507 (Insufficient Capacity) response can only be done if the rejecting server has definitive knowledge of available capacity on the target. This will require some kind of state sharing mechanism between TURN servers, which is beyond the scope of this specification. If a TURN server attempts to

redirect to another server without knowledge of available capacity, it is possible that all servers are in a congested state, resulting in series of rejections that only serve to further increase the load on the system. This can cause congestion collapse.

If a client sends a request to a server and gets a 500 class error response without an ALTERNATE-SERVER, or the STUN transaction times out without a response, and the client was utilizing the SRV procedures of [I-D.ietf-behave-rfc3489bis] to contact the server, the client SHOULD try another server based on those procedures. However, the client SHOULD cache the fact that the request to this server

failed, and not retry that server again for a configurable period of time. Five minutes is RECOMMENDED.

TURN clients and servers MUST NOT include the FINGERPRINT attribute in any of the methods defined in this document.

5. Managing Allocations

Communications between a TURN client and a TURN server begin with an Allocate transaction. All subsequent transactions happen in the context of that allocation, and happen on the same 5-tuple. The client refreshes allocations and deallocates them using a Refresh transaction.

5.1. Client Behavior

5.1.1. Initial Allocate Requests

When a client wishes to obtain a transport address, it sends an Allocate request to the server. This request is constructed and sent using the general procedures defined in [I-D.ietf-behave-rfc3489bis]. Clients MUST implement the long term credential mechanism defined in [I-D.ietf-behave-rfc3489bis], and be prepared for the server to demand credentials for requests.

The client SHOULD include a BANDWIDTH attribute, which indicates the maximum bandwidth that will be used with this binding. If the maximum is unknown, the attribute is not included in the request.

The client MAY request a particular lifetime for the allocation by including it in the LIFETIME attribute in the request.

The client MUST include a REQUESTED-TRANSPORT attribute. In this specification, the REQUESTED-TRANSPORT MUST always be UDP. This attribute is included to allow for future extensions to TURN (e.g., [I-D.ietf-behave-turn-tcp])

The client MAY include a REQUESTED-PORT-PROPS or REQUESTED-IP attribute in the request to obtain specific types of transport addresses, if desired.

Processing of the response follows the general procedures of [I-D.ietf-behave-rfc3489bis]. A successful response will include both a RELAY-ADDRESS and an XOR-MAPPED-ADDRESS attribute, providing both a relayed transport address and a reflexive transport address, respectively, to the client. The value of the LIFETIME attribute in the response indicates the amount of time after which the server will expire the allocation, if not refreshed with a Refresh request. The server will allow the user to send and receive at least the amount of data indicated in the BANDWIDTH attribute per allocation. (At its discretion the server can optionally discard UDP data above this threshold.)

If the response is an error response and contains a 442, 443 or 444 error code, the client knows that its requested properties could not be met. The client MAY retry with different properties, with the same properties (in a hope that something has changed on the server), or give up, depending on the needs of the application. However, if the client retries, it SHOULD wait 500ms, and if the request fails again, wait 1 second, then 2 seconds, and so on, exponentially backing off.

5.1.2. Refresh Requests

TURN permissions are kept alive by traffic flowing through them, and persist for the lifetime of the allocation. However, The allocations themselves have to be kept alive through Refresh Requests.

Before 3/4 of the lifetime of the allocation has passed (the lifetime of the allocation is conveyed in the LIFETIME attribute of the Allocate Response), the client SHOULD refresh the allocation with a Refresh transaction if it wishes to keep the allocation.

To perform a refresh, the client generates a Refresh Request. The client MUST use the same username, realm and password for the Refresh request as it used in its initial Allocate Request. The Refresh request MAY contain a proposed LIFETIME attribute. The client MAY include a BANDWIDTH attribute if it wishes to request more or less bandwidth than in the original request (this might also be the first time the TURN client indicates bandwidth to the TURN server). If the BANDWIDTH attribute is absent, it indicates no change in the requested bandwidth from the Allocate request. The client MUST NOT include a REQUESTED-IP, REQUESTED-TRANSPORT, or REQUESTED-PORT-PROPS attribute in the Refresh request.

In a successful response, the LIFETIME attribute indicates the amount of additional time (the number of seconds after the response is received) that the allocation will live without being refreshed. A successful response will also contain a BANDWIDTH attribute, indicating the bandwidth the server is allowing for this allocation. Note that an error response does not imply that the allocation has expired, just that the refresh has failed.

If a client no longer needs an allocation, it SHOULD perform an explicit deallocation. If the client wishes to explicitly remove the allocation because it no longer needs it, it sends a Refresh request, but sets the LIFETIME attribute to zero. This will cause the server to remove the allocation, and all associated permissions and channel numbers. For connection-oriented transports such as TCP, the client can also remove the allocation (and all associated bindings) by closing the relevant connection with the TURN server.

5.2. Server Behavior

5.2.1. Receiving an Allocate Request

When the server receives an Allocate request, the server attempts to allocate a relayed transport address.

When the server receives the Allocate Request, it begins by processing it according to the base protocol procedures described in [I-D.ietf-behave-rfc3489bis], plus the Long-Term Credential Mechanism procedures if the server is using this mechanism.

It then checks if the 5-tuple used for the Allocate Request matches the 5-tuple used for an existing allocation. If there is a match, then:

- If the transport protocol is UDP, and the transaction id in the request message matches the transaction id used for the original allocation, then the server treats this as a retransmission of the original request, and replies with the same response as it did to the original request. The server may do this by either storing its original response and resending it, or by rebuilding its original response from other state data.

- If the transport protocol is not UDP, or if the transaction id in the request message does not match the transaction id used for the original allocation, then the server replies with an error response containing the error code 437 Allocation Mismatch.

If the 5-tuple does not match an existing allocation, then processing

continues as described below.

5.2.1.1. BANDWIDTH

The server checks for the BANDWIDTH attribute in the request. If present, the server determines whether or not it has sufficient capacity to handle a binding that will generate the requested bandwidth.

If it does, the server attempts to allocate a transport address for the client. The Allocate Request can contain several additional attributes that allow the client to request specific characteristics of the transport address. If it doesn't, it sends an error response.

5.2.1.2. REQUESTED-TRANSPORT

The server checks for the REQUESTED-TRANSPORT attribute. This indicates the transport protocol requested by the client. This specification defines a value for UDP only, but support for TCP allocations is planned in [I-D.ietf-behave-turn-tcp].

> As a consequence of the REQUESTED-TRANSPORT attribute, it is possible for a client to connect to the server over TCP or TLS over TCP and request a UDP transport address. In this case, the server will relay data between the transports.

If the requested transport is supported, the server allocates a port using the requested transport protocol. If the REQUESTED-TRANSPORT attribute contains a value of the transport protocol unknown to the server, or known to the server but not supported by the server in the context of this request, the server MUST reject the request and include a 442 (Unsupported Transport Protocol) in the response. If the request did not contain a REQUESTED-TRANSPORT attribute, the server MUST use the same transport protocol as the request arrived on.

5.2.1.3. REQUESTED-IP

The server checks for the REQUESTED-IP attribute. If present, it indicates a specific IP address from which the client would like its transport address allocated. (The client could do this if it requesting the second address in a specific port pair). If this IP

address is not a valid one for allocations on the server, the server MUST reject the request and include a 443 (Invalid IP Address) error code in the response, or else redirect the request to a server that is known to support this IP address. If the IP address is one that is valid for allocations (presumably, the server is configured to know the set of IP addresses from which it performs allocations), the server MUST provide an allocation from that IP address. If the attribute is not present, the selection of an IP address is at the

discretion of the server.

5.2.1.4. REQUESTED-PORT-PROPS

The server checks for the REQUESTED-PORT-PROPS attribute. If present, it indicates specific port properties desired by the client. This attribute is split into two portions: one portion for port behavior and the other for requested port alignment (whether the allocated port is odd, even, reserved as a pair, or at the discretion of the server).

If the port behavior requested is for a Specific Port, the server MUST attempt to allocate that specific port for the client. If the specific port is not available (in use or reserved), the server MUST reject the request with a 444 (Invalid Port) response. For example, the STUN server could reject a request for a Specific Port because the port is temporarily reserved as part of an adjacent pair of ports, or because the requested port is a well-known port (1-1023).

If the client requests "even" port alignment, the server MUST attempt to allocate an even port for the client. If an even port cannot be obtained, the server MUST reject the request with a 444 (Invalid Port) response or redirect to an alternate server. If the client requests odd port alignment, the server MUST attempt to allocate an odd port for the client. If an odd port cannot be obtained, the server MUST reject the request with a 444 (Invalid Port) response or redirect to an alternate server. Finally, the "Even port with hold of the next higher port" alignment is similar to requesting an even port. It is a request for an even port, and MUST be rejected by the server if an even port cannot be provided, or redirected to an alternate server. However, it is also a hint from the client that the client will request the next higher port with a separate Allocate request. As such, it is a request for the server to allocate an even

port whose next higher port is also available, and furthermore, a request for the server to not allocate that one higher port to any other request except for one that asks for that port explicitly. The server can honor this request for adjacency at its discretion. The only constraint is that the allocated port number MUST be even.

> Port alignment requests exist for compatibility with implementations of RTP which predate [RFC3550]. These implementations use the port numbering conventions in (now obsolete) [RFC1889].

5.2.1.5. Lifetime

The server checks for a LIFETIME attribute. If present, it indicates the lifetime the client would like the server to assign to the allocation.

If the LIFETIME attribute is malformed, or if the requested lifetime value is less than 32 seconds, the server replies with an error response with an error code of XXX Lifetime Malformed or Invalid.

5.2.1.6. Creating the Allocation

If any of the requested or desired constraints cannot be met, whether it be bandwidth, transport protocol, IP address or port, the server can redirect the client to a different server that may be able to fulfill the request. This is accomplished using the 300 error response and ALTERNATE-SERVER attribute. If the server does not redirect and cannot service the request because the server has reached capacity, it sends a 507 (Insufficient Capacity) response. The server can also reject the request with a 486 (Allocation Quota Reached) if the user or client is not authorized to request additional allocations.

The server SHOULD only allocate ports from the range 49152 - 65535 (the Dynamic and/or Private Port range [Port-Numbers]), unless the TURN server application knows, through some means not specified here,

that other applications running on the same host as the TURN server application will not be impacted by allocating ports outside this range. This condition can often be satisfied by running the TURN server application on a dedicated machine and/or by arranging that any other applications on the machine allocate ports before the TURN server application starts. In any case, the TURN server SHOULD NOT allocate ports in the range 0 - 1023 (the Well-Known Port range) to discourage clients from using TURN to run standard services.

Once a port is allocated, the server associates the allocation with the 5-tuple used to communicate between the client and the server. For TCP, this amounts to associating the TCP connection from the TURN client with the allocated transport address.

The new allocation MUST also be associated with the username, password and realm used to authenticate the request. These credentials are used in all subsequent requests to ensure that only the same client can use or modify the allocation it was given.

In addition, the allocation created by the server is associated with a set of permissions and a set of channel bindings. Each set is initially empty.

If the LIFETIME attribute was present in the request, and the value is larger than the maximum duration the server is willing to use for the lifetime of the allocation, the server MAY lower it to that maximum. However, the server MUST NOT increase the duration requested in the LIFETIME attribute. If there was no LIFETIME attribute, the server may choose a duration at its discretion. Ten minutes is RECOMMENDED. In either case, the resulting duration is added to the current time, and a timer, called the allocation expiration timer, is set to expire at or after that time. Note that the LIFETIME attribute in an Allocate request can be zero, though this is effectively a no-op, since it will create and destroy the allocation in one transaction.

5.2.1.7. Sending the Allocate Response

Once the port has been obtained and the allocation expiration timer has been started, the server generates an Allocate Response using the general procedures defined in [I-D.ietf-behave-rfc3489bis], including the ones for long term authentication. The transport address

allocated to the client MUST be included in the RELAY-ADDRESS attribute in the response. In addition, this response MUST contain the XOR-MAPPED-ADDRESS attribute. This allows the client to determine its reflexive transport address in addition to a relayed transport address, from the same Allocate request.

The server MUST add a LIFETIME attribute to the Allocate Response. This attribute contains the duration, in seconds, of the allocation expiration timer associated with this allocation.

The server MUST add a BANDWIDTH attribute to the Allocate Response. This MUST be equal to the attribute from the request, if one was present. Otherwise, it indicates a per-allocation limit that the server is placing on the bandwidth usage on each binding. Such limits are needed to prevent against denial-of-service attacks (see Section 12).

5.2.2. Refresh Requests

A Refresh request is processed using the general server and long term authentication procedures in [I-D.ietf-behave-rfc3489bis]. It is used to refresh and extend an allocation, or to cause an immediate deallocation. It is processed as follows.

First, the request MUST be authenticated using the same shared secret as the one associated with the allocation. If the request was authenticated but not with such a matching credential, the server MUST generate a Refresh Error Response with a 401 response.

If the Refresh request contains a BANDWIDTH attribute, the server checks that it can relay the requested volume of traffic.

Finally, a Refresh Request will set a new allocation expiration timer for the allocation, effectively canceling the previous allocation expiration timer. As with an Allocate request, the server MAY utilize a shorter allocation lifetime, but MUST NOT utilize a longer lifetime.

A success Refresh response MUST contain a LIFETIME attribute. If its associated Allocate request contained the BANDWIDTH attribute, or this Refresh request contained a new BANDWIDTH attribute, the

response MUST also contain the BANDWIDTH attribute.

6. Send and Data Indications

TURN supports two ways to send and receive data from peers. This section describes the use of Send and Data indications, while Section 7 describes the use of the Channel Mechanism.

6.1. Forming and Sending an Indication

When the client has data to send to a peer, it uses a Send Indication to pass the data to the server. When the server has data to send to the client, it uses a Data Indication to pass the data to the client. A client can also use a Send Indication without a DATA attribute to install or refresh a permission for the specified IP address. Both indications are formed following the general rules described in [ref 3489bis] with the extra considerations described below.

A Send Indication MUST contain a PEER-ADDRESS attribute and MAY contain a DATA attribute, while a Data Indication MUST contain both attributes. The PEER-ADDRESS attribute contains the transport address of the peer to which the data is to be sent (in the case of a Send Indication) or from which the data was received (in the case of a Data Indication). This peer address is the transport address of the peer as seen by the server, which may not be the same as the host transport address of the peer. The DATA attribute contains the actual application data. Note that the application data may need to be padded to ensure the DATA attribute length is a multiple of 4.

No other attributes are included. For example, neither the FINGERPRINT attribute nor any authentication attributes are included. The latter holds even if the server is using the Long-Term Credential Mechanism, since indications cannot be authenticated using this mechanism.

Both the Send and Data indications MUST be sent using the 5-tuple of the original allocation. Thus, in the case of the Send Indication, the source transport address is the client's host transport address, the destination transport address is the TURN server address, and the transport protocol is the same as was used for the Allocate request.

For the Data Indication, the source and destination transport addresses are the reverse.

6.2. Receiving an Indication

When a Send Indication is received at the server, or a Data Indication is received at the client, the receiver first does the basic indication processing described in [3489bis]. Once this is done, it does the processing specific to the Send and Data methods described below.

A Send Indication MUST contain a PEER-ADDRESS attribute and MAY contain a DATA attribute, while a Data Indication MUST contain both attributes. Any other attributes appearing in the message are treated as unexpected.

> TODO: Add check that Send or Data indication arrives with appropriate 5-tuple. Since this check applies to all STUN messages, not just Send and Data indications, perhaps this goes under the general processing section.

6.3. Relaying

When the server receives a valid Send Indication contains a DATA attribute, 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 Length field in the UDP header is set to the Length field of the DATA attribute;
- the Checksum field in the UDP header is computed as described in [RFC 768].

The resulting UDP datagram is then sent to the peer.

When the server receives a valid Send Indication (with or without a DATA attribute), it also updates the permission associated with the IP address contained in the PEER-ADDRESS attribute. For a certain interval after the permission is updated, UDP datagrams received from peers with source IP address equal to the IP address contained in the PEER-ADDRESS attribute can be forwarded to the client. Note that only the IP addresses are considered and the port numbers are irrelevent. This permission is specific to the allocation and has no affect on any other allocation. The recommended length of time is 60 seconds from when the Send Indication is received.

When the server receives a UDP datagram with a destination transport address corresponding to an active (i.e., still alive) allocation, then it first checks to see if it is permitted to relay the datagram. If it is not permitted, the UDP datagram MUST be discarded.

If relaying is permitted, the server forms and send a Data Indication as described in Section 6.1, using the data following the UDP header as the application data.

7. Channel Mechanism

As described in the overview, channel mechanism provides a way for a 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.

Once a channel is bound to a peer, the channel binding cannot be changed. There is no way to unbind a channel or bind it to a different peer.

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.

7.1. Forming and Sending a ChannelBind Request

When a client wishes to bind a channel to a peer in an allocation, it forms a ChannelBind Request. The Request formed following the

general rules described in [I-D.ietf-behave-rfc3489bis] with the extra considerations described below.

A ChannelBind Request MUST contain both a CHANNEL-NUMBER attribute and a PEER-ADDRESS attribute. The CHANNEL-NUMBER attribute specifies the number of the channel that the client wishes to bind to the peer. The channel number MUST be in the range 0x4000 to 0xFFFE (inclusive) and the channel MUST NOT be already bound to a different peer. It is acceptable to rebind a channel to the peer it is already bound to. The PEER-ADDRESS attribute specifies the peer address to bind the channel to.

Once formed, the ChannelBind Request is sent using the 5-tuple for the allocation.

The client SHOULD be prepared to receive ChannelData messages on the channel as soon as it has sent the ChannelBind Request. Over UDP, it is possible for the client to receive these before it receives a ChannelBind Success Response.

Over UDP, the client SHOULD NOT send ChannelData messages on the channel until it has received a ChannelBind Success Response for the binding attempt. Sending them before the success response is received risks having them dropped by the server if he ChannelBind Request was lost.

7.2. Receiving a ChannelBind Request and Sending a Response

When the server receives a ChannelBind Request, it first does the basic request processing described in [I-D.ietf-behave-rfc3489bis]. Once this is done, it does the processing specific to the ChannelBind method described below.

The server checks that the ChannelBind Request contains both a CHANNEL-NUMBER attribute and a PEER-ADDRESS attribute. If the PEER-ADDRESS attribute is missing or malformed, then the server rejects the request with an Error Response containing the error code XXX "Peer address missing or invalid". If the CHANNEL-NUMBER attribute is missing or malformed, or the channel number is not in the range 0x4000 to 0xFFFE (inclusive), or the channel is already bound to another peer (already bound to the same peer is OK) the server rejects the request with an Error Response containing the error code

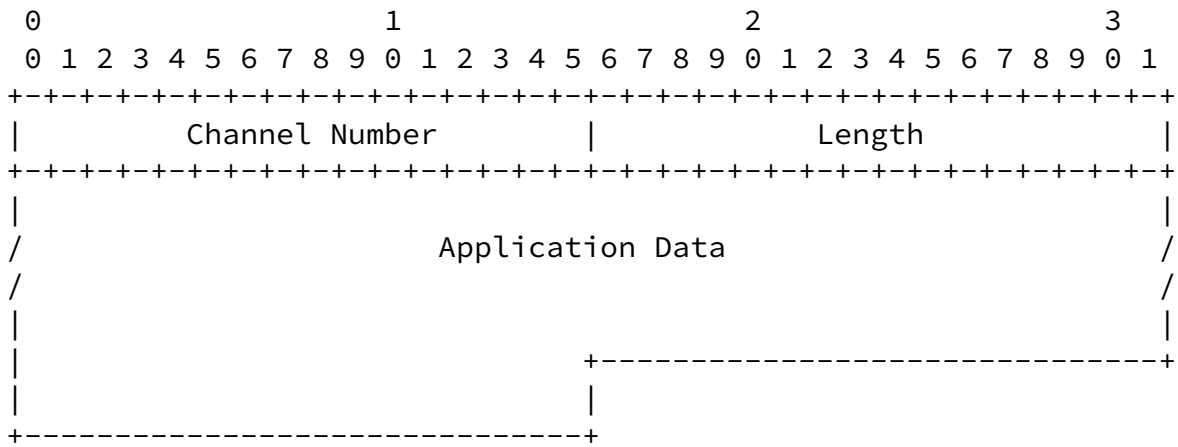
XXX "Channel number missing or invalid". Otherwise, if no errors are detected, the server replies with a ChannelBind Success Response.

7.3. Receiving a ChannelBind Response

When the client receives a ChannelBind response (either success or error), it processes it as specified in [3489bis]. Any additional processing is implementation specific.

7.4. The ChannelData Message

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



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

7.5. Forming and Sending a ChannelData Message

Once a client has bound a channel to a peer, then when the client has data to send to that peer it may use either a ChannelData message or a Send Indication; that is, the client is not obligated to use the

channel when it exists and may freely intermix the two message types when sending data to the peer. The server, on the other hand, SHOULD 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 7.4.

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.

7.6. Receiving a ChannelData Message

The receiver of the ChannelData message uses bits 0 and 1 to distinguish it from STUN-formatted messages, as described in Section 7.4.

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 7.5.

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

7.7. Relaying

When the server receives a valid ChannelData message, 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 ChannelData message arrived; the destination transport address is the peer 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 Length field in the UDP header is set to the Length field of the ChannelData message + 8; and the Checksum field in the UDP header is computed as described in

[RFC 768]. The resulting UDP datagram is then sent to the peer.

The server also updates the permission associated with the IP address part of the peer address to which the UDP datagram is sent.

When the server receives a UDP datagram with a destination transport address corresponding to an active (i.e., still alive) allocation, then it first checks to see if it is permitted to relay the datagram. If the allocation contains an active permission for the source IP address (from the IP header) of the received UDP datagram, then the UDP datagram is permitted. Otherwise, the UDP datagram MUST be discarded.

To relay the UDP datagram, the server forms and send a ChannelData message as described in Section 7.5

8. New STUN Methods

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

of these new methods.

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

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

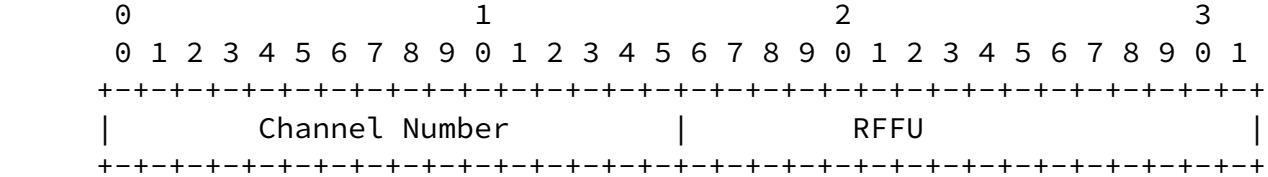
9. New STUN Attributes

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-PORT-PROPS
  0x0019: REQUESTED-TRANSPORT
  0x0022: REQUESTED-IP
```


9.1. CHANNEL-NUMBER

The CHANNEL-NUMBER attribute contains the number of the channel. It is a 16-bit unsigned integer, followed by a two-octet RFFU field which MUST be set to 0 on transmission and ignored on reception.



9.2. 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.

9.3. BANDWIDTH

The bandwidth attribute represents the peak bandwidth, measured in kilobits per second, that the client expects to use on the allocation in each direction.

9.4. PEER-ADDRESS

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.

9.5. DATA

The DATA attribute is present in most Send Indications and Data 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).

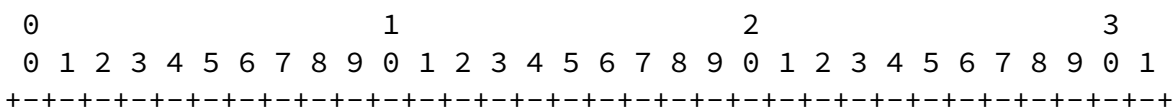
9.6. RELAY-ADDRESS

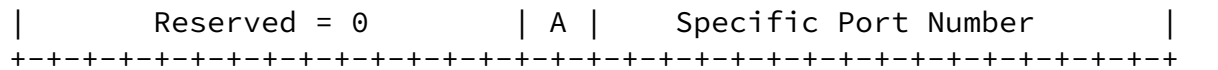
The RELAY-ADDRESS is present in Allocate responses. It specifies the address and port that the server allocated to the client. It is encoded in the same way as XOR-MAPPED-ADDRESS.

9.7. REQUESTED-PORT-PROPS

This attribute allows the client to request certain properties for the port that is allocated by the server. The attribute can be used

with any transport protocol that has the notion of a 16 bit port space (including TCP and UDP). The attribute is 32 bits long. Its format is:





The two bits labeled A in the diagram above are for requested port alignment and have the following meaning:

```
00  no specific port alignment
01  odd port number
10  even port number
11  even port number; reserve next higher port
```

If the value of the A field is 00 (no specific port alignment), then the Specific Port Number field can either be 0 or some non-zero port number. If the Specific Port Number field is 0, then the client is not putting any restrictions on the port number it would like allocated. If the Specific Port Number is some non-zero port number, then the client is requesting that the server allocate the specified port and the server MUST provide that port.

If the value of the A field is 01 (odd port number), then the Specific Port Number field MUST be zero, and the client is requesting the server allocate an odd-numbered port. The server MUST provide an odd port number.

If the value of the A field is 10 (even port number), then the Specific Port number field MUST be zero, and the client is requesting the server allocate an even-numbered port. The server MUST provide an even port number.

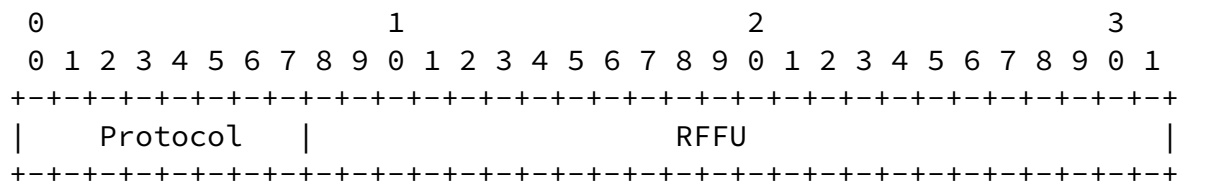
If the value of the A field is 11 (even port number; reserve next higher port), then the Specific Port Number field MUST be zero, and the client is requesting the server allocate an even-numbered port. The server MUST return an even port number. In addition, the client is requesting the server reserve the next higher port (i.e., N+1 if the server allocates port N). The server SHOULD only allocate the N+1 port number if it is explicitly requested (with a subsequent request specifying that exact port number by the same TURN client, over a different alllocation).

In all cases, if a port with the requested properties cannot be allocated, the server MUST respond with a error response with an

error code of 444 (Invalid Port).

9.8. REQUESTED-TRANSPORT

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]. 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.

9.9. REQUESTED-IP

The REQUESTED-IP attribute is used by the client to request that a specific IP address be allocated by the TURN server. This attribute is needed since it is anticipated that TURN servers will be multi-homed so as to be able to allocate more than 64k transport addresses. As a consequence, a client needing a second transport address on the same interface as a previous one can use this attribute to request a remote address from the same TURN server interface as the TURN client's previous remote address.

The format of this attribute is identical to XOR-MAPPED-ADDRESS. However, the port component of the attribute MUST be ignored by the server. If a client wishes to request a specific IP address and port, it uses both the REQUESTED-IP and REQUESTED-PORT-PROPS attributes.

10. New STUN Error Response Codes

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.

442 (Unsupported Transport Protocol): The Allocate request asked for a transport protocol to be allocated that is not supported by the server. If the server is aware of another server that supports the requested protocol, it SHOULD include the other server's address in an ALTERNATE-SERVER attribute in the error response.

443 (Invalid IP Address): The Allocate request asked for a transport address to be allocated from a specific IP address that is not valid on the server.

444 (Invalid Port): The Allocate request asked for a port to be allocated that is not available on the server.

486 (Allocation Quota Reached): The user or client is not authorized to request additional allocations.

(tbd) (Channel Number Missing or Invalid): The request requires a channel number, but the CHANNEL-NUMBER attribute is missing, or the specified channel number is invalid in some way.

(tbd) (Peer Address Missing or Invalid): The request requires a peer transport address, but the PEER-ADDRESS attribute is missing, or the specified peer transport address is invalid in some way.

(tbd) (Lifetime Malformed or Invalid): The LIFETIME attribute is malformed or the specified lifetime is invalid in some way.

507 (Insufficient Capacity): The server cannot allocate a new port for this client as it has exhausted its relay capacity.

11. Client Discovery of TURN Servers

The STUN extensions introduced by TURN differ from the binding requests defined in [I-D.ietf-behave-rfc3489bis] in that they are sent with additional framing and demand substantial resources from the TURN server. In addition, it seems likely that administrators might want to block connections from clients to the TURN server for

relaying separately from connections for the purposes of binding discovery. As a consequence, TURN runs on a separate port from STUN. The client discovers the address and port of the TURN server using the same DNS procedures defined in [I-D.ietf-behave-rfc3489bis], but using an SRV service name of "turn" (or "turns" for TURN over TLS)

instead of just "stun".

For example, to find TURN servers in the example.com domain, the TURN client performs a lookup for '_turn._udp.example.com', '_turn._tcp.example.com', and '_turns._tcp.example.com' if the STUN client wants to communicate with the TURN server using UDP, TCP, or TLS over TCP, respectively.

12. Security Considerations

TURN servers allocate bandwidth and port resources to clients, in contrast to the Binding method defined in [I-D.ietf-behave-rfc3489bis]. Therefore, a TURN server requires 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] [[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 cannot be used by clients for subverting firewall policies. 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 web servers, email servers, SIP 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.

One TURN-specific DoS attack bears extra discussion. An attacker who can corrupt, drop, or cause the loss of a Send or Data indication sent over UDP, and then forge a Channel Confirmation indication for the corresponding channel number, can cause a TURN client (server) to start sending unencapsulated data that the server (client) will discard. Since indications are not integrity protected, this attack is not prevented by cryptographic means. However, any attacker who can generate this level of network disruption could simply prevent a large fraction of the data from arriving at its destination, and therefore protecting against this attack does not seem important. The ChannelConfirmation forging attack is not possible when the client to server communication is over TCP or TLS over TCP.

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) [RFC4302] in transport or tunnel mode. However, tunnel-mode IPSec ESP [RFC4303] should still operate.

13. IANA Considerations

Since TURN is an extension to STUN [I-D.ietf-behave-rfc3489bis], 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 8.

The codepoints for the new STUN attributes defined in this specification are listed in Section 9.

The codepoints for the new STUN error codes defined in this specification are listed in Section 10.

Extensions to TURN can be made through IETF consensus.

14. IAB Considerations

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]. 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 [I-D.ietf-mmusic-ice].

15. Example

TBD

16. Changes from Previous Versions

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.

16.1. Changes from -05 to -06

- 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.

16.2. Changes from -04 to -05

- 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.

17. Issues

NOTE to RFC Editor: Please remove this section prior to publication of this document as an RFC.

This section lists the open and now closed issues in this document. The descriptions here are brief, and the reader should consult the corresponding thread on the mailing list for a more in-depth description of the issue and the resolutions being considered.

17.1. Open Issues

1. Bandwidth: What should we do with the BANDWIDTH attribute, which is currently ill-specified? Should we remove it? Or should we try to come up with a good specification, perhaps using ideas from RSVP?

2. Permission Policy: What should the permission policy be? Address-restricted, as is currently specified in the document? Or address-and-port-restricted, as many firewalls implement today? Or should we leave this open to the implementor, under the assumption that the IT administrator will only allow clients to contact those servers that implement whatever permission policy the IT administrator can accept?

3. Port Adjacency: The spec currently allows a client to request that the server allocate a port and also reserve the next higher port number for a possible future allocation (on a different 5-tuple). However, the exact behavior of the server in this case is ill-specified. For example, must the next-higher-port be available for the allocation of the lower port number to succeed? How long is the next-higher-port reserved? 30 seconds? For the lifetime of the lower-numbered-port's allocation? Or should we just ditch this feature, since it is difficult to implement, it is at odds with port randomization, and paired port numbers applications don't work well with NATs anyway?

4. Demuxing ChannelData messages: How does a client or server demux STUN-formatted messages from ChannelData messages? Does it use the first two bits (as currently specified) or just one bit? And how many channels do we need anyway? Some people are questioning the need for any more than 200 channels. If we don't need many channels, then the demux algorithm might become simpler.

5. Deallocating Channels: Do we need a mechanism for deallocating channels? Some have argued for this feature, because a TURN server administrator will want a way to recover resources for

channels no longer in active use. If yes, then what is the mechanism? For example, should a channel binding expire when the corresponding permission expires?

6. Permissions and Channel Allocations: Should allocating a channel for a peer automatically install a permission for that peer's IP address?

7. Permission and Allocation Lifetimes: What should the default permission lifetime be? Should there be a minumum value? Should there be a way for the client to modify the permission lifetime? Should there be a way for the client to learn the current permission lifetime? And what is the relationship of the permission lifetime to the allocation lifetime? Does it make sense for the allocation lifetime to be less than the permission lifetime?

8. Preserving bits in the IP header: What bits (if any) should be preserved in the IP header when a packet is relayed by the server? The bits under consideration are currently the Don't Fragment (DF) bit, the Explicit Congestion Notification (ECN) bits, and the DiffServ (DS) bits.

9. Exceeding the Path MTU Size: TURN adds an overhead of 4 bytes (ChannelData msg) or 36 bytes (Send or Data Indication), thus potentially exceeding the path MTU between the client and server. This could either cause IP fragmentation, or cause the packet to be dropped if the DF bit is set. Who handles this problem? Does TURN need to handle this, or is this left up to the application to handle?

10. Allowed PEER-ADDRESS values: Should there be any restrictions on the IP address the client can specify in the PEER-ADDRESS attribute? Are multicast addresses allowed? What about 0.0.0.0? Any other restrictions?

11. Discarding UDP datagrams: If the server discards a received UDP datagram on the relayed transport address (because there is no corresponding permission), then does the server send an ICMP response? If so, what error code does it use? (What does RFC 4787 say about the corresponding situation in NATs? I believe many NATs silently discard these packets by default, or have a "stealth mode" that enables this behavior.)

12. Authentication: Is the use of STUN's Long-Term Authentication Mechanism by a TURN server mandatory? The document currently

implicitly assumes "yes", but what about someone who wants to operate a public TURN server?

13. Re-using the 5-tuple: If an allocation expires, is there any reason a client should not be able to immediately create a new allocation using the same 5-tuple?

14. Password change: Is it possible to change the password for the Long-Term Authentication mechanism during the lifetime of an allocation? If so, how is it done?

15. IPv6: TURN probably works fine in an all IPv6 environment, but there are a number of mixed IPv4/IPv6 cases that are ill-specified. As an example, the server needs to check that the PEER-ADDRESS in a Send Indication is of the same address family as the relayed transport address. Should we carefully work through all these cases and make sure we have caught them all, or should we just state that this document covers the IPv4 case only, and punt the specification of IPv6 and mixed IPv4/IPv6 operation to draft-ietf-behave-turn-ipv6? Does the current interest in resurecting IPv4-to-IPv6 NATs have any impact on TURN?

17.2. Closed Issues

1. Channel Allocation: Should TURN use the mechanism proposed by EKR to allocate channels? RESOLUTION: Yes. Document now reflects this.

2. Stateful Allocations: Does a TURN server need to distinguish between the case where the client retransmits the initial Allocate Request because the Allocate Response was lost and the case where the client sends an Allocate Request because it thinks the allocation does not exist? RESOLUTION: Yes. Document now reflects this.

3. Port Range: From what range of port numbers should a TURN server allocate ports? RESOLUTION: The server SHOULD allocate from the Dynamic and/or Private Port range unless it is sure it will not interfere with other apps on the same machine. Document now reflects this.

4. Framing Header for STUN-formatted messages: Should TURN use the framing mechanism for STUN-formatted messages? RESOLUTION: NO. Document now reflects this. However, see related issues.

5. Length field in ChannelData header: Over UDP, the length of the application data field in the ChannelData message can be determined from the length field in the UDP header. So should the length field in the ChannelData header be set to zero in this case? RESOLUTION: No, the ChannelData length field should have

the same semantics over both TCP and UDP. Document now reflects this.

18. Acknowledgements

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, Cullen Jennings, Lars Eggert, Magnus Westerlund, and Eric Rescorla have been particularly helpful, with Eric also suggesting the channel allocation 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 his huge help in restarting progress on this draft after work had stalled.

19. References

19.1. Normative References

[I-D.ietf-behave-rfc3489bis]
Rosenberg, J., Mahy, R., Matthews, P., and D. Wing, "Session Traversal Utilities for (NAT) (STUN)", draft-ietf-behave-rfc3489bis-13 (work in progress), November 2007.

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

19.2. Informative References

[RFC3550] Schulzrinne, H., Casner, S., Frederick, R., and V. Jacobson, "RTP: A Transport Protocol for Real-Time Applications", STD 64, RFC 3550, July 2003.

[RFC1889] Schulzrinne, H., Casner, S., Frederick, R., and V. Jacobson, "RTP: A Transport Protocol for Real-Time Applications", RFC 1889, January 1996.

[RFC1918] Rekhter, Y., Moskowitz, R., Karrenberg, D., Groot, G., and E. Lear, "Address Allocation for Private Internets", BCP 5, RFC 1918, February 1996.

[RFC3264] Rosenberg, J. and H. Schulzrinne, "An Offer/Answer Model with Session Description Protocol (SDP)", RFC 3264, June 2002.

[RFC4302] Kent, S., "IP Authentication Header", RFC 4302, December 2005.

[RFC4303] Kent, S., "IP Encapsulating Security Payload (ESP)", RFC 4303, December 2005.

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

[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.

[RFC4787] Audet, F. and C. Jennings, "Network Address Translation (NAT) Behavioral Requirements for Unicast UDP", BCP 127, RFC 4787, January 2007.

[I-D.ietf-behave-turn-tcp] Rosenberg, J. and R. Mahy, "Traversal Using Relays around NAT (TURN) Extensions for TCP Allocations", draft-ietf-behave-turn-tcp-00 (work in progress), November 2007.

[Port-Numbers]
"IANA Port Numbers Registry",
<http://www.iana.org/assignments/port-numbers>.

[Protocol-Numbers]
"IANA Protocol Numbers Registry", 2005,
<http://www.iana.org/assignments/protocol-numbers>.

Authors' Addresses

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 x223
Fax:
Email: philip_matthews@magma.ca
URI:

Full Copyright Statement

THE INTERNET ENGINEERING TASK FORCE DISCLAIM ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO ANY WARRANTY THAT THE USE OF THE INFORMATION HEREIN WILL NOT INFRINGE ANY RIGHTS OR ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

Intellectual Property

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

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

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

Acknowledgment

Funding for the RFC Editor function is provided by the IETF Administrative Support Activity (IASA).