< draft-ietf-tram-turnbis-27.txt

```
TRAM WG                                                  T. Reddy, Ed.
Internet-Draft                                                  McAfee
Obsoletes: 5766, 6156 (if approved)                  A. Johnston, Ed.
Intended status: Standards Track                  Villanova University
Expires: December 29, 2019                                P. Matthews
                                                        Alcatel-Lucent
                                                          J. Rosenberg
                                                           jdrosen.net
                                                         June 27, 2019
```

```
 Traversal Using Relays around NAT (TURN): Relay Extensions to Session
                  Traversal Utilities for NAT (STUN)
                      draft-ietf-tram-turnbis-27
```

```
Abstract

   If a host is located behind a NAT, then in certain situations it can
   be impossible for that host to communicate directly with other hosts
   (peers).  In these situations, it is necessary for the host to use
```

draft-ietf-tram-turnbis-28.txt >

```
TRAM WG                                                  T. Reddy, Ed.
Internet-Draft                                                  McAfee
Obsoletes: 5766, 6156 (if approved)                  A. Johnston, Ed.
Intended status: Standards Track                  Villanova University
Expires: January 17, 2020                                 P. Matthews
                                                        Alcatel-Lucent
                                                          J. Rosenberg
                                                           jdrosen.net
                                                         July 16, 2019
```

```
 Traversal Using Relays around NAT (TURN): Relay Extensions to Session
                  Traversal Utilities for NAT (STUN)
                      draft-ietf-tram-turnbis-27
```

```
Abstract

   If a host is located behind a NAT, then in certain situations it can
   be impossible for that host to communicate directly with other hosts
   (peers).  In these situations, it is necessary for the host to use
```

skipping to change at *page 2, line 4* | **skipping to change at *page 2, line 4***

skipping to change at *page 2, line 45* | **skipping to change at *page 2, line 45***

draft-ietf-tram-turnbis-27.txt	draft-ietf-tram-turnbis-28.txt
21.3.1. DoS against TURN Server 79	21.3.1. DoS against TURN Server 80
21.3.2. Anonymous Relaying of Malicious Traffic 79	21.3.2. Anonymous Relaying of Malicious Traffic 80
21.3.3. Manipulating Other Allocations 80	21.3.3. Manipulating Other Allocations 80
21.4. Tunnel Amplification Attack 80	21.4. Tunnel Amplification Attack 80
21.5. Other Considerations 81	21.5. Other Considerations 82
22. IANA Considerations 81	22. IANA Considerations 82
23. IAB Considerations 82	23. IAB Considerations 83
24. Changes since RFC 5766 84	24. Changes since RFC 5766 84
25. Updates to RFC 6156 84	25. Updates to RFC 6156 85
26. Acknowledgements 84	26. Acknowledgements 85
27. References 85	27. References 85
27.1. Normative References 85	27.1. Normative References 85
27.2. Informative References 87	27.2. Informative References 87
Authors' Addresses 91	Authors' Addresses 92
1. Introduction	1. Introduction
A host behind a NAT may wish to exchange packets with other hosts, some of which may also be behind NATs. To do this, the hosts involved can use "hole punching" techniques (see [RFC5128]) in an attempt discover a direct communication path; that is, a communication path that goes from one host to another through intervening NATs and routers, but does not traverse any relays.	A host behind a NAT may wish to exchange packets with other hosts, some of which may also be behind NATs. To do this, the hosts involved can use "hole punching" techniques (see [RFC5128]) in an attempt discover a direct communication path; that is, a communication path that goes from one host to another through intervening NATs and routers, but does not traverse any relays.
skipping to change at *page 4, line 46*	**skipping to change at *page 4, line 46***
For example, if both hosts are behind NATs that have a mapping behavior of "address-dependent mapping" or "address- and port-dependent mapping" (Section 4.1 in [RFC4787]), then hole punching techniques generally fail.	For example, if both hosts are behind NATs that have a mapping behavior of "address-dependent mapping" or "address- and port-dependent mapping" (Section 4.1 in [RFC4787]), then hole punching techniques generally fail.
When a direct communication path cannot be found, it is necessary to use the services of an intermediate host that acts as a relay for the packets. This relay typically sits in the public Internet and relays packets between two hosts that both sit behind NATs.	When a direct communication path cannot be found, it is necessary to use the services of an intermediate host that acts as a relay for the packets. This relay typically sits in the public Internet and relays packets between two hosts that both sit behind NATs.
	In many enterprise networks, direct UDP transmissions are not permitted between clients on the internal networks and external IP addresses. To permit media sessions in such a situation to use UDP and to avoid forcing the media sessions through TCP, an Enterprise Firewall can be configured to allow UDP traffic relayed through an Enterprise relay server. This scenario is required to be supported by the WebRTC requirements (Section 2.3.5.1 in [RFC7478]). In addition, in a SIP or WebRTC call, if the user wants IP location privacy from the peer then the client can select a relay server offering IP location privacy and only convey the relayed candidates to the peer for ICE connectivity checks (see Section 4.2.4 in [I-D.ietf-rtcweb-security]).
This specification defines a protocol, called TURN, that allows a host behind a NAT (called the TURN client) to request that another host (called the TURN server) act as a relay. The client can arrange for the server to relay packets to and from certain other hosts (called peers) and can control aspects of how the relaying is done. The client does this by obtaining an IP address and port on the server, called the relayed transport address. When a peer sends a packet to the relayed transport address, the server relays the transport protocol data from the packet to the client. The client knows the peer from which the transport protocol data was relayed by	This specification defines a protocol, called TURN, that allows a host behind a NAT (called the TURN client) to request that another host (called the TURN server) act as a relay. The client can arrange for the server to relay packets to and from certain other hosts (called peers) and can control aspects of how the relaying is done. The client does this by obtaining an IP address and port on the server, called the relayed transport address. When a peer sends a packet to the relayed transport address, the server relays the transport protocol data from the packet to the client. The client knows the peer from which the transport protocol data was relayed by

skipping to change at *page 6, line 19*	**skipping to change at *page 6, line 31***
messages are STUN-formatted messages. A reader of this document should be familiar with STUN.	messages are STUN-formatted messages. A reader of this document should be familiar with STUN.
The TURN specification was originally published as [RFC5766], which was updated by [RFC6156] to add IPv6 support. This document supersedes and obsoletes both [RFC5766] and [RFC6156].	The TURN specification was originally published as [RFC5766], which was updated by [RFC6156] to add IPv6 support. This document supersedes and obsoletes both [RFC5766] and [RFC6156].
2. Terminology	2. Terminology
The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [RFC2119].	The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119][RFC8174] when, and only when, they appear in all capitals, as shown here.
Readers are expected to be familiar with [I-D.ietf-tram-stunbis] and the terms defined there.	Readers are expected to be familiar with [I-D.ietf-tram-stunbis] and the terms defined there.
The following terms are used in this document:	The following terms are used in this document:
TURN: The protocol spoken between a TURN client and a TURN server. It is an extension to the STUN protocol [I-D.ietf-tram-stunbis]. The protocol allows a client to allocate and use a relayed transport address.	TURN: The protocol spoken between a TURN client and a TURN server. It is an extension to the STUN protocol [I-D.ietf-tram-stunbis]. The protocol allows a client to allocate and use a relayed transport address.
skipping to change at *page 8, line 6*	**skipping to change at *page 8, line 18***
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.	of a peer that is permitted to send traffic to the TURN server and have that traffic relayed to the TURN client. The TURN server will only forward traffic to its client from peers that match an existing permission.
Realm: A string used to describe the server or a context within the server. The realm tells the client which username and password combination to use to authenticate requests.	Realm: A string used to describe the server or a context within the server. The realm tells the client which username and password combination to use to authenticate requests.
Nonce: A string chosen at random by the server and included in the message-digest. To prevent replay attacks, the server should change the nonce regularly.	Nonce: A string chosen at random by the server and included in the server response. To prevent replay attacks, the server should change the nonce regularly.
(D)TLS: This term is used for statements that apply to both Transport Layer Security [RFC8446] and Datagram Transport Layer Security [RFC6347].	(D)TLS: This term is used for statements that apply to both Transport Layer Security [RFC8446] and Datagram Transport Layer Security [RFC6347].
3. Overview of Operation	3. Overview of Operation
This section gives an overview of the operation of TURN. It is non-normative.	This section gives an overview of the operation of TURN. It is non-normative.
skipping to change at *page 10, line 42*	**skipping to change at *page 10, line 42***
communicate with multiple peers.	communicate with multiple peers.
When the peer is behind a NAT, then the client must identify the peer using its server-reflexive transport address rather than its host transport address. For example, to send application data to Peer A in the example above, the client must specify 192.0.2.150:32102 (Peer A's server-reflexive transport address) rather than 203.0.113.2:49582	When the peer is behind a NAT, then the client must identify the peer using its server-reflexive transport address rather than its host transport address. For example, to send application data to Peer A in the example above, the client must specify 192.0.2.150:32102 (Peer A's server-reflexive transport address) rather than 203.0.113.2:49582

(Peer A's host transport address).	(Peer A's host transport address).
Each allocation on the server belongs to a single client and has exactly one relayed transport address that is used only by that allocation. Thus, when a packet arrives at a relayed transport address on the server, the server knows for which client the data is intended.	Each allocation on the server belongs to a single client and has exactly one or two relayed transport addresses that is used only by that allocation. Thus, when a packet arrives at a relayed transport address on the server, the server knows for which client the data is intended.
The client may have multiple allocations on a server at the same time.	The client may have multiple allocations on a server at the same time.
3.1. Transports	3.1. Transports
TURN, as defined in this specification, always uses UDP between the server and the peer. However, this specification allows the use of	TURN, as defined in this specification, always uses UDP between the server and the peer. However, this specification allows the use of
skipping to change at *page 11, line 49*	**skipping to change at *page 11, line 49***
rules. Also, TCP has explicit connection teardown; while for UDP, the firewall has to use timers to guess when the flow is finished.	rules. Also, TCP has explicit connection teardown; while for UDP, the firewall has to use timers to guess when the flow is finished.
TURN supports TLS-over-TCP transport and DTLS-over-UDP transport between the client and the server because (D)TLS provides additional security properties not provided by TURN's default digest authentication; properties that some clients may wish to take advantage of. In particular, (D)TLS provides a way for the client to ascertain that it is talking to the correct server, and provides for confidentiality of TURN control messages. If (D)TLS transport is used between the TURN client and the TURN server, the cipher suites discussed in Section 6.2.3 of [I-D.ietf-tram-stunbis] and the guidance given in [RFC7525] MUST be followed to avoid attacks on (D)TLS. TURN does not require (D)TLS because the overhead of using (D)TLS is higher than that of digest authentication; for example, using (D)TLS likely means that most application data will be doubly encrypted (once by (D)TLS and once to ensure it is still encrypted in the UDP datagram).	TURN supports TLS-over-TCP transport and DTLS-over-UDP transport between the client and the server because (D)TLS provides additional security properties not provided by TURN's default digest authentication; properties that some clients may wish to take advantage of. In particular, (D)TLS provides a way for the client to ascertain that it is talking to the correct server, and provides for confidentiality of TURN control messages. If (D)TLS transport is used between the TURN client and the TURN server, the cipher suites, server certificate validation and authentication of TURN server are discussed in Section 6.2.3 of [I-D.ietf-tram-stunbis]. The guidance given in [RFC7525] MUST be followed to avoid attacks on (D)TLS. TURN does not require (D)TLS because the overhead of using (D)TLS is higher than that of digest authentication; for example, using (D)TLS likely means that most application data will be doubly encrypted (once by (D)TLS and once to ensure it is still encrypted in the UDP datagram).
There is an extension to TURN for TCP transport between the server and the peers [RFC6062]. For this reason, allocations that use UDP between the server and the peers are known as UDP allocations, while allocations that use TCP between the server and the peers are known as TCP allocations. This specification describes only UDP allocations.	There is an extension to TURN for TCP transport between the server and the peers [RFC6062]. For this reason, allocations that use UDP between the server and the peers are known as UDP allocations, while allocations that use TCP between the server and the peers are known as TCP allocations. This specification describes only UDP allocations.
In some applications for TURN, the client may send and receive packets other than TURN packets on the host transport address it uses	In some applications for TURN, the client may send and receive packets other than TURN packets on the host transport address it uses
skipping to change at *page 16, line 20*	**skipping to change at *page 16, line 20***
of the peer included in an XOR-PEER-ADDRESS attribute and the ICMP type and code in a ICMP attribute. ICMP attribute forwarding always uses Data indications containing the XOR-PEER-ADDRESS and ICMP attributes, even when using the channel mechanism to forward UDP data.	of the peer included in an XOR-PEER-ADDRESS attribute and the ICMP type and code in a ICMP attribute. ICMP attribute forwarding always uses Data indications containing the XOR-PEER-ADDRESS and ICMP attributes, even when using the channel mechanism to forward UDP data.
Send and Data indications cannot be authenticated, since the long-term credential mechanism of STUN does not support authenticating indications. This is not as big an issue as it might first appear, since the client-to-server leg is only half of the total path to the	Send and Data indications cannot be authenticated, since the long-term credential mechanism of STUN does not support authenticating indications. This is not as big an issue as it might first appear, since the client-to-server leg is only half of the total path to the

draft-ietf-tram-turnbis-27.txt	draft-ietf-tram-turnbis-28.txt
peer. Applications that want proper security should encrypt the data sent between the client and a peer.	peer. Applications that want end-to-end security should encrypt the data sent between the client and a peer.
Because Send indications are not authenticated, it is possible for an attacker to send bogus Send indications to the server, which will then relay these to a peer. To partly mitigate this attack, TURN requires that the client install a permission towards a peer before sending data to it using a Send indication. The technique to fully mitigate the attack is discussed in Section 21.1.4.	Because Send indications are not authenticated, it is possible for an attacker to send bogus Send indications to the server, which will then relay these to a peer. To partly mitigate this attack, TURN requires that the client install a permission towards a peer before sending data to it using a Send indication. The technique to fully mitigate the attack is discussed in Section 21.1.4.
TURN client TURN server Peer A Peer B	TURN client TURN server Peer A Peer B
skipping to change at *page 19, line 18*	**skipping to change at *page 19, line 18***
and from Peer B would use the Send mechanism.	and from Peer B would use the Send mechanism.
3.6. Unprivileged TURN Servers	3.6. Unprivileged TURN Servers
This version of TURN is designed so that the server can be implemented as an application that runs in user space under commonly available operating systems without requiring special privileges. This design decision was made to make it easy to deploy a TURN server: for example, to allow a TURN server to be integrated into a peer-to-peer application so that one peer can offer NAT traversal services to another peer.	This version of TURN is designed so that the server can be implemented as an application that runs in user space under commonly available operating systems without requiring special privileges. This design decision was made to make it easy to deploy a TURN server: for example, to allow a TURN server to be integrated into a peer-to-peer application so that one peer can offer NAT traversal services to another peer and to use (D)TLS to secure the TURN connection.
This design decision has the following implications for data relayed by a TURN server:	This design decision has the following implications for data relayed by a TURN server:
o The value of the Diffserv field may not be preserved across the server;	o The value of the Diffserv field may not be preserved across the server;
o The Time to Live (TTL) field may be reset, rather than decremented, across the server;	o The Time to Live (TTL) field may be reset, rather than decremented, across the server;
skipping to change at *page 19, line 42*	**skipping to change at *page 19, line 43***
o There is no end-to-end fragmentation, since the packet is re-assembled at the server.	o There is no end-to-end fragmentation, since the packet is re-assembled at the server.
Future work may specify alternate TURN semantics that address these limitations.	Future work may specify alternate TURN semantics that address these limitations.
3.7. Avoiding IP Fragmentation	3.7. Avoiding IP Fragmentation
For reasons described in [Frag-Harmful], applications, especially those sending large volumes of data, should avoid having their packets fragmented. Applications using TCP can more or less ignore this issue because fragmentation avoidance is now a standard part of TCP, but applications using UDP (and thus any application using this version of TURN) need to avoid IP fragmentation by sending sufficiently small messages or use UDP fragmentation [I-D.ietf-tsvwg-udp-options].	For reasons described in [Frag-Harmful], applications, especially those sending large volumes of data, should avoid having their packets fragmented. [I-D.ietf-intarea-frag-fragile] discusses issues associated with IP fragmentation and proposes alternatives to IP fragmentation. Applications using TCP can more or less ignore this issue because fragmentation avoidance is now a standard part of TCP, but applications using UDP (and thus any application using this version of TURN) need to avoid IP fragmentation by sending sufficiently small messages or use UDP fragmentation [I-D.ietf-tsvwg-udp-options]. Note that the UDP fragmentation option needs to be supported by both endpoints, and at the time of writing of this document, UDP fragmentation support is under discussion and

	is not deployed.
The application running on the client and the peer can take one of two approaches to avoid IP fragmentation until UDP fragmentation support is available. The first uses messages that are limited to a predetermined fixed maximum and the second relies on network feedback to adapt that maximum.	The application running on the client and the peer can take one of two approaches to avoid IP fragmentation until UDP fragmentation support is available. The first uses messages that are limited to a predetermined fixed maximum and the second relies on network feedback to adapt that maximum.
The first approach is to avoid sending large amounts of application data in the TURN messages/UDP datagrams exchanged between the client and the peer. This is the approach taken by most VoIP (Voice-over-	The first approach is to avoid sending large amounts of application data in the TURN messages/UDP datagrams exchanged between the client and the peer. This is the approach taken by most VoIP (Voice-over-
skipping to change at *page 20, line 44*	**skipping to change at *page 20, line 50***
overhead of the ChannelData message is less than Send and Data indications.	overhead of the ChannelData message is less than Send and Data indications.
The second approach the client and peer can take to avoid fragmentation is to use a path MTU discovery algorithm to determine the maximum amount of application data that can be sent without fragmentation. The classic path MTU discovery algorithm defined in [RFC1191] may not be able to discover the MTU of the transmission path between the client and the peer since:	The second approach the client and peer can take to avoid fragmentation is to use a path MTU discovery algorithm to determine the maximum amount of application data that can be sent without fragmentation. The classic path MTU discovery algorithm defined in [RFC1191] may not be able to discover the MTU of the transmission path between the client and the peer since:
- a probe packet with DF bit set to test a path for a larger MTU can be dropped by routers, or	- a probe packet with DF bit in the IPv4 header set to test a path for a larger MTU can be dropped by routers, or
- ICMP error messages can be dropped by middle boxes.	- ICMP error messages can be dropped by middle boxes.
As a result, the client and server need to use a path MTU discovery algorithm that does not require ICMP messages. The Packetized Path MTU Discovery algorithm defined in [RFC4821] is one such algorithm.	As a result, the client and server need to use a path MTU discovery algorithm that does not require ICMP messages. The Packetized Path MTU Discovery algorithm defined in [RFC4821] is one such algorithm.
[I-D.ietf-tram-stun-pmtud] is an implementation of [RFC4821] that uses STUN to discover the path MTU, and so might be a suitable approach to be used in conjunction with a TURN server that supports the DONT-FRAGMENT attribute. When the client includes the DONT-FRAGMENT attribute in a Send indication, this tells the server to set the DF bit in the resulting UDP datagram that it sends to the peer. Since some servers may be unable to set the DF bit, the client should also include this attribute in the Allocate request -- any server that does not support the DONT-FRAGMENT attribute will indicate this by rejecting the Allocate request. If the TURN server carrying out packet translation from IPv4-to-IPv6 cannot get the Don't Fragment (DF) bit in the IPv4 header, it MUST reject the Allocate request with DONT-FRAGMENT attribute.	[I-D.ietf-tram-stun-pmtud] is an implementation of [RFC4821] that uses STUN to discover the path MTU, and so might be a suitable approach to be used in conjunction with a TURN server that supports the DONT-FRAGMENT attribute. When the client includes the DONT-FRAGMENT attribute in a Send indication, this tells the server to set the DF bit in the resulting UDP datagram that it sends to the peer. Since some servers may be unable to set the DF bit, the client should also include this attribute in the Allocate request -- any server that does not support the DONT-FRAGMENT attribute will indicate this by rejecting the Allocate request. If the TURN server carrying out packet translation from IPv4-to-IPv6 is unable to access the state of Don't Fragment (DF) bit in the IPv4 header, it MUST reject the Allocate request with DONT-FRAGMENT attribute.
3.8. RTP Support	3.8. RTP Support
One of the envisioned uses of TURN is as a relay for clients and peers wishing to exchange real-time data (e.g., voice or video) using RTP. To facilitate the use of TURN for this purpose, TURN includes some special support for older versions of RTP.	One of the envisioned uses of TURN is as a relay for clients and peers wishing to exchange real-time data (e.g., voice or video) using RTP. To facilitate the use of TURN for this purpose, TURN includes some special support for older versions of RTP.
Old versions of RTP [RFC3550] required that the RTP stream be on an even port number and the associated RTP Control Protocol (RTCP)	Old versions of RTP [RFC3550] required that the RTP stream be on an even port number and the associated RTP Control Protocol (RTCP)
skipping to change at *page 21, line 47*	**skipping to change at *page 22, line 5***
server's IPv6 path is not working, a dual-stack TURN client can	server's IPv6 path is not working, a dual-stack TURN client can

experience a significant connection delay compared to an IPv4-only TURN client. To overcome these connection setup problems, the TURN client needs to query both A and AAAA records for the TURN server specified using a domain name and try connecting to the TURN server using both IPv6 and IPv4 addresses in a fashion similar to the Happy Eyeballs mechanism defined in [RFC8305]. The TURN client performs the following steps based on the transport protocol being used to connect to the TURN server.	experience a significant connection delay compared to an IPv4-only TURN client. To overcome these connection setup problems, the TURN client needs to query both A and AAAA records for the TURN server specified using a domain name and try connecting to the TURN server using both IPv6 and IPv4 addresses in a fashion similar to the Happy Eyeballs mechanism defined in [RFC8305]. The TURN client performs the following steps based on the transport protocol being used to connect to the TURN server.
o For TCP or TLS-over-TCP, initiate TCP connection to both IP address families as discussed in [RFC8305], and use the first TCP connection that is established. If connections are established on both IP address families then terminate the TCP connection using the IP address family with lower precedence [RFC6724].	o For TCP or TLS-over-TCP, the results of the Happy Eyeballs procedure [RFC8305] are used by the TURN client for sending its TURN messages to the server.
o For clear text UDP, send TURN Allocate requests to both IP address families as discussed in [RFC8305], without authentication information. If the TURN server requires authentication, it will send back a 401 unauthenticated response and the TURN client uses the first UDP connection on which a 401 error response is received. If a 401 error response is received from both IP address families then the TURN client can silently abandon the UDP connection on the IP address family with lower precedence. If the TURN server does not require authentication (as described in	o For clear text UDP, send TURN Allocate requests to both IP address families as discussed in [RFC8305], without authentication information. If the TURN server requires authentication, it will send back a 401 unauthenticated response and the TURN client uses the first UDP connection on which a 401 error response is received. If a 401 error response is received from both IP address families then the TURN client can silently abandon the UDP connection on the IP address family with lower precedence. If the TURN server does not require authentication (as described in
skipping to change at *page 22, line 33*	**skipping to change at *page 22, line 37***
to terminate the DTLS session using the IP address family with lower precedence. If TURN over DTLS server has been configured to require a cookie exchange (Section 4.2 in [RFC6347]) and HelloVerifyRequest is received from the TURN servers on both IP address families then the client can silently abandon the connection on the IP address family with lower precedence.	to terminate the DTLS session using the IP address family with lower precedence. If TURN over DTLS server has been configured to require a cookie exchange (Section 4.2 in [RFC6347]) and HelloVerifyRequest is received from the TURN servers on both IP address families then the client can silently abandon the connection on the IP address family with lower precedence.
4. Discovery of TURN server	4. Discovery of TURN server
Methods of TURN server discovery, including using anycast, are described in [RFC8155]. The syntax of the "turn" and "turns" URIs are defined in Section 3.1 of [RFC7065]. DTLS as a transport protocol for TURN is defined in [RFC7350].	Methods of TURN server discovery, including using anycast, are described in [RFC8155]. If a host with multiple interfaces discovers a TURN server in each interface, the mechanism described in [RFC7982] can be used by the TURN client to influence the TURN server selection. The syntax of the "turn" and "turns" URIs are defined in Section 3.1 of [RFC7065]. DTLS as a transport protocol for TURN is defined in [RFC7350].
4.1. TURN URI Scheme Semantics	4.1. TURN URI Scheme Semantics
The "turn" and "turns" URI schemes are used to designate a TURN server (also known as a relay) on Internet hosts accessible using the TURN protocol. The TURN protocol supports sending messages over UDP, TCP, TLS-over-TCP or DTLS-over-UDP. The "turns" URI scheme MUST be used when TURN is run over TLS-over-TCP or in DTLS-over-UDP, and the "turn" scheme MUST be used otherwise. The required <host> part of the "turn" URI denotes the TURN server host. The <port> part, if	The "turn" and "turns" URI schemes are used to designate a TURN server (also known as a relay) on Internet hosts accessible using the TURN protocol. The TURN protocol supports sending messages over UDP, TCP, TLS-over-TCP or DTLS-over-UDP. The "turns" URI scheme MUST be used when TURN is run over TLS-over-TCP or in DTLS-over-UDP, and the "turn" scheme MUST be used otherwise. The required <host> part of the "turn" URI denotes the TURN server host. The <port> part, if
skipping to change at *page 23, line 19*	**skipping to change at *page 23, line 23***
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-tram-stunbis] apply to	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-tram-stunbis] apply to

STUN-formatted messages. This means that all the message-forming and message-processing descriptions in this document are implicitly prefixed with the rules of [I-D.ietf-tram-stunbis].	STUN-formatted messages. This means that all the message-forming and message-processing descriptions in this document are implicitly prefixed with the rules of [I-D.ietf-tram-stunbis].
[I-D.ietf-tram-stunbis] specifies an authentication mechanism called the long-term credential mechanism. TURN servers and clients MUST implement this mechanism. The server MUST demand that all requests from the client be authenticated using this mechanism, or that a equally strong or stronger mechanism for client authentication is used.	[I-D.ietf-tram-stunbis] specifies an authentication mechanism called the long-term credential mechanism. TURN servers and clients MUST implement this mechanism.
Note that the long-term credential mechanism applies only to requests and cannot be used to authenticate indications; thus, indications in TURN are never authenticated. If the server requires requests to be authenticated, then the server's administrator MUST choose a realm value that will uniquely identify the username and password combination that the client must use, even if the client uses multiple servers under different administrations. The server's administrator MAY choose to allocate a unique username to each client, or MAY choose to allocate the same username to more than one	Note that the long-term credential mechanism applies only to requests and cannot be used to authenticate indications; thus, indications in TURN are never authenticated. If the server requires requests to be authenticated, then the server's administrator MUST choose a realm value that will uniquely identify the username and password combination that the client must use, even if the client uses multiple servers under different administrations. The server's administrator MAY choose to allocate a unique username to each client, or MAY choose to allocate the same username to more than one
skipping to change at *page 25, line 4*	**skipping to change at *page 25, line 6***
To ensure interoperability, a TURN server MUST support the use of UDP transport between the client and the server, and SHOULD support the use of TCP, TLS-over-TCP and DTLS-over-UDP transports.	To ensure interoperability, a TURN server MUST support the use of UDP transport between the client and the server, and SHOULD support the use of TCP, TLS-over-TCP and DTLS-over-UDP transports.
When UDP or DTLS-over-UDP transport is used between the client and the server, the client will retransmit a request if it does not receive a response within a certain timeout period. Because of this, the server may receive two (or more) requests with the same 5-tuple and same transaction id. STUN requires that the server recognize this case and treat the request as idempotent (see	When UDP or DTLS-over-UDP transport is used between the client and the server, the client will retransmit a request if it does not receive a response within a certain timeout period. Because of this, the server may receive two (or more) requests with the same 5-tuple and same transaction id. STUN requires that the server recognize this case and treat the request as idempotent (see
[I-D.ietf-tram-stunbis]). Some implementations may choose to meet this requirement by remembering all received requests and the corresponding responses for 40 seconds. Other implementations may choose to reprocess the request and arrange that such reprocessing returns essentially the same response. To aid implementors who choose the latter approach (the so-called "stateless stack approach"), this specification includes some implementation notes on how this might be done. Implementations are free to choose either approach or choose some other approach that gives the same results.	[I-D.ietf-tram-stunbis]). Some implementations may choose to meet this requirement by remembering all received requests and the corresponding responses for 40 seconds (Section 6.3.1 in [I-D.ietf-tram-stunbis]). Other implementations may choose to reprocess the request and arrange that such reprocessing returns essentially the same response. To aid implementors who choose the latter approach (the so-called "stateless stack approach"), this specification includes some implementation notes on how this might be done. Implementations are free to choose either approach or choose some other approach that gives the same results.
To mitigate either intentional or unintentional denial-of-service attacks against the server by clients with valid usernames and passwords, it is RECOMMENDED that the server impose limits on both the number of allocations active at one time for a given username and on the amount of bandwidth those allocations can use. The server should reject new allocations that would exceed the limit on the allowed number of allocations active at one time with a 486 (Allocation Quota Exceeded) (see Section 7.2), and should discard application data traffic that exceeds the bandwidth quota.	To mitigate either intentional or unintentional denial-of-service attacks against the server by clients with valid usernames and passwords, it is RECOMMENDED that the server impose limits on both the number of allocations active at one time for a given username and on the amount of bandwidth those allocations can use. The server should reject new allocations that would exceed the limit on the allowed number of allocations active at one time with a 486 (Allocation Quota Exceeded) (see Section 7.2), and since UDP does not include a congestion control mechanism, it should discard application data traffic that exceeds the bandwidth quota.
6. Allocations	6. Allocations
All TURN operations revolve around allocations, and all TURN messages are associated with either a single or dual allocation. An	All TURN operations revolve around allocations, and all TURN messages are associated with either a single or dual allocation. An

allocation conceptually consists of the following state data:	allocation conceptually consists of the following state data:
o the relayed transport address or addresses;	o the relayed transport address or addresses;
o the 5-tuple: (client's IP address, client's port, server IP	o the 5-tuple: (client's IP address, client's port, server IP
skipping to change at *page 25, line 51*	**skipping to change at *page 26, line 6***
o a list of channel to peer bindings for each relayed transport address.	o a list of channel to peer bindings for each relayed transport address.
The relayed transport address is the transport address allocated by the server for communicating with peers, while the 5-tuple describes the communication path between the client and the server. On the client, the 5-tuple uses the client's host transport address; on the server, the 5-tuple uses the client's server-reflexive transport address. The relayed transport address MUST be unique across all allocations, so it can be used to uniquely identify the allocation.	The relayed transport address is the transport address allocated by the server for communicating with peers, while the 5-tuple describes the communication path between the client and the server. On the client, the 5-tuple uses the client's host transport address; on the server, the 5-tuple uses the client's server-reflexive transport address. The relayed transport address MUST be unique across all allocations so it can be used to uniquely identify the allocation, and an allocation in this context can be either a single or dual allocation.
Both the relayed transport address and the 5-tuple MUST be unique across all allocations, so either one can be used to uniquely identify the allocation, and an allocation in this context can be either a single or dual allocation.	
The authentication information (e.g., username, password, realm, and nonce) is used to both verify subsequent requests and to compute the message integrity of responses. The username, realm, and nonce values are initially those used in the authenticated Allocate request that creates the allocation, though the server can change the nonce value during the lifetime of the allocation using a 438 (Stale Nonce) reply. For security reasons, the server MUST NOT store the password explicitly and MUST store the key value, which is a secure hash over the username, realm, and password (see Section 16.1.3 in [I-D.ietf-tram-stunbis]).	The authentication information (e.g., username, password, realm, and nonce) is used to both verify subsequent requests and to compute the message integrity of responses. The username, realm, and nonce values are initially those used in the authenticated Allocate request that creates the allocation, though the server can change the nonce value during the lifetime of the allocation using a 438 (Stale Nonce) reply. For security reasons, the server MUST NOT store the password explicitly and MUST store the key value, which is a cryptographic hash over the username, realm, and password (see Section 16.1.3 in [I-D.ietf-tram-stunbis]).
	Note that if the response contains a PASSWORD-ALGORITHMS attribute and this attribute contains both MD5 and SHA-256 algorithms, and the client also supports both the algorithms, the request MUST contain a PASSWORD-ALGORITHM attribute with the SHA-256 algorithm.
The time-to-expiry is the time in seconds left until the allocation expires. Each Allocate or Refresh transaction sets this timer, which then ticks down towards 0. By default, each Allocate or Refresh transaction resets this timer to the default lifetime value of 600 seconds (10 minutes), but the client can request a different value in the Allocate and Refresh request. Allocations can only be refreshed using the Refresh request; sending data to a peer does not refresh an allocation. When an allocation expires, the state data associated with the allocation can be freed.	The time-to-expiry is the time in seconds left until the allocation expires. Each Allocate or Refresh transaction sets this timer, which then ticks down towards 0. By default, each Allocate or Refresh transaction resets this timer to the default lifetime value of 600 seconds (10 minutes), but the client can request a different value in the Allocate and Refresh request. Allocations can only be refreshed using the Refresh request; sending data to a peer does not refresh an allocation. When an allocation expires, the state data associated with the allocation can be freed.
skipping to change at *page 26, line 43*	**skipping to change at *page 26, line 48***
7. Creating an Allocation	7. Creating an Allocation
An allocation on the server is created using an Allocate transaction.	An allocation on the server is created using an Allocate transaction.
7.1. Sending an Allocate Request	7.1. Sending an Allocate Request

The client forms an Allocate request as follows.	The client forms an Allocate request as follows.
The client first picks a host transport address. It is RECOMMENDED that the client pick a currently unused transport address, typically by allowing the underlying OS to pick a currently unused port.	The client first picks a host transport address. It is RECOMMENDED that the client picks a currently unused transport address, typically by allowing the underlying OS to pick a currently unused port.
The client then picks a transport protocol that the client supports to use between the client and the server based on the transport protocols supported by the server. 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 discovery or by experiment, that it is unable to contact any TURN server using UDP.	The client then picks a transport protocol that the client supports to use between the client and the server based on the transport protocols supported by the server. 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 discovery or by experiment, that it is unable to contact any TURN server using UDP.
skipping to change at *page 28, line 35*	**skipping to change at *page 28, line 42***
client MUST omit the EVEN-PORT attribute.	client MUST omit the EVEN-PORT attribute.
Once constructed, the client sends the Allocate request on the 5-tuple.	Once constructed, the client sends the Allocate request on the 5-tuple.
7.2. Receiving an Allocate Request	7.2. Receiving an Allocate Request
When the server receives an Allocate request, it performs the following checks:	When the server receives an Allocate request, it performs the following checks:
1. The server SHOULD require that the request be authenticated. The authentication of the request is optional to allow TURN servers provided by the local or access network to accept Allocation requests from new and/or guest users in the network who do not necessarily possess long term credentials for STUN authentication and its security implications are discussed in [RFC8155]. If the request is authenticated, the authentication MUST be done using the long-term credential mechanism of [I-D.ietf-tram-stunbis] unless the client and server agree to use another mechanism through some procedure outside the scope of this document.	1. The TURN server provided by the local or access network MAY allow unauthenticated request in order to accept Allocation requests from new and/or guest users in the network who do not necessarily possess long term credentials for STUN authentication. Making STUN authentication optional and its security implications are discussed in [RFC8155]. Otherwise, the server MUST require that the request be authenticated. If the request is authenticated, the authentication MUST be done either using the long-term credential mechanism of [I-D.ietf-tram-stunbis] or the STUN Extension for Third-Party Authorization [RFC7635] unless the client and server agree to use another mechanism through some procedure outside the scope of this document.
2. The server checks if the 5-tuple is currently in use by an existing allocation. If yes, the server rejects the request with a 437 (Allocation Mismatch) error.	2. The server checks if the 5-tuple is currently in use by an existing allocation. If yes, the server rejects the request with a 437 (Allocation Mismatch) error.
3. The server checks if the request contains a REQUESTED-TRANSPORT attribute. If the REQUESTED-TRANSPORT attribute is not included or is malformed, the server rejects the request with a 400 (Bad	3. The server checks if the request contains a REQUESTED-TRANSPORT attribute. If the REQUESTED-TRANSPORT attribute is not included or is malformed, the server rejects the request with a 400 (Bad
skipping to change at *page 29, line 35*	**skipping to change at *page 29, line 42***
corresponding relayed transport address is still available). If the token is not valid for some reason, the server rejects the request with a 508 (Insufficient Capacity) error.	corresponding relayed transport address is still available). If the token is not valid for some reason, the server rejects the request with a 508 (Insufficient Capacity) error.
6. The server checks if the request contains both REQUESTED-ADDRESS-FAMILY and ADDITIONAL-ADDRESS-FAMILY attributes. If yes, then the server rejects the request with a 400 (Bad Request) error.	6. The server checks if the request contains both REQUESTED-ADDRESS-FAMILY and ADDITIONAL-ADDRESS-FAMILY attributes. If yes, then the server rejects the request with a 400 (Bad Request) error.

draft-ietf-tram-turnbis-27.txt	draft-ietf-tram-turnbis-28.txt
7. If the server does not support the address family requested by the client in REQUESTED-ADDRESS-FAMILY or is disabled by local policy, it MUST generate an Allocate error response, and it MUST include an ERROR-CODE attribute with the 440 (Address Family not Supported) response code. If the REQUESTED-ADDRESS-FAMILY attribute is absent and the server does not support IPv4 address family, the server MUST include an ERROR-CODE attribute with the 440 (Address Family not Supported) response code. If the REQUESTED-ADDRESS-FAMILY attribute is absent and the server supports IPv4 address family, the server MUST allocate an IPv4 relayed transport address for the TURN client.	7. If the server does not support the address family requested by the client in REQUESTED-ADDRESS-FAMILY or if the allocation of the requested address family is disabled by local policy, it MUST generate an Allocate error response, and it MUST include an ERROR-CODE attribute with the 440 (Address Family not Supported) response code. If the REQUESTED-ADDRESS-FAMILY attribute is absent and the server does not support IPv4 address family, the server MUST include an ERROR-CODE attribute with the 440 (Address Family not Supported) response code. If the REQUESTED-ADDRESS-FAMILY attribute is absent and the server supports IPv4 address family, the server MUST allocate an IPv4 relayed transport address for the TURN client.
8. The server checks if the request contains an EVEN-PORT attribute with the R bit set to 1. If yes, and the request also contains an ADDITIONAL-ADDRESS-FAMILY attribute, the server rejects the request with a 400 (Bad Request) error. Otherwise, the server checks if it can satisfy the request (i.e., can allocate a relayed transport address as described below). If the server cannot satisfy the request, then the server rejects the request with a 508 (Insufficient Capacity) error.	8. The server checks if the request contains an EVEN-PORT attribute with the R bit set to 1. If yes, and the request also contains an ADDITIONAL-ADDRESS-FAMILY attribute, the server rejects the request with a 400 (Bad Request) error. Otherwise, the server checks if it can satisfy the request (i.e., can allocate a relayed transport address as described below). If the server cannot satisfy the request, then the server rejects the request with a 508 (Insufficient Capacity) error.
9. The server checks if the request contains an ADDITIONAL-ADDRESS-FAMILY attribute. If yes, and the attribute value is 0x01 (IPv4 address family), then the server rejects the request with a 400 (Bad Request) error. Otherwise, the server checks if it can allocate relayed transport addresses of both address types. If the server cannot satisfy the request, then the server rejects the request with a 508 (Insufficient Capacity) error. If the server can partially meet the request, i.e. if it can only allocate one relayed transport address of a specific address type, then it includes ADDRESS-ERROR-CODE attribute in the response to inform the client the reason for partial failure of the request. The error code value signaled in the ADDRESS-ERROR-CODE attribute could be 440 (Address Family not Supported) or 508 (Insufficient Capacity). If the server can fully meet the request, then the server allocates one IPv4 and one IPv6 relay address, and returns an Allocate success response containing the relayed transport addresses assigned to the dual allocation in two XOR-RELAYED-ADDRESS attributes.	9. The server checks if the request contains an ADDITIONAL-ADDRESS-FAMILY attribute. If yes, and the attribute value is 0x01 (IPv4 address family), then the server rejects the request with a 400 (Bad Request) error. Otherwise, the server checks if it can allocate relayed transport addresses of both address types. If the server cannot satisfy the request, then the server rejects the request with a 508 (Insufficient Capacity) error. If the server can partially meet the request, i.e. if it can only allocate one relayed transport address of a specific address type, then it includes ADDRESS-ERROR-CODE attribute in the success response to inform the client the reason for partial failure of the request. The error code value signaled in the ADDRESS-ERROR-CODE attribute could be 440 (Address Family not Supported) or 508 (Insufficient Capacity). If the server can fully meet the request, then the server allocates one IPv4 and one IPv6 relay address, and returns an Allocate success response containing the relayed transport addresses assigned to the dual allocation in two XOR-RELAYED-ADDRESS attributes.
10. At any point, the server MAY choose to reject the request with a 486 (Allocation Quota Reached) error if it feels the client is trying to exceed some locally defined allocation quota. The server is free to define this allocation quota any way it wishes, but SHOULD define it based on the username used to authenticate the request, and not on the client's transport address.	10. At any point, the server MAY choose to reject the request with a 486 (Allocation Quota Reached) error if it feels the client is trying to exceed some locally defined allocation quota. The server is free to define this allocation quota any way it wishes, but SHOULD define it based on the username used to authenticate the request, and not on the client's transport address.
skipping to change at *page 32, line 26*	**skipping to change at *page 32, line 35***
remove orphaned allocations (that is, allocations where the corresponding client has crashed or terminated or the client connection has been lost for some reason). Also, note that the time-to-expiry is recomputed with each successful Refresh request, and thus the value computed here applies only until the first refresh.	remove orphaned allocations (that is, allocations where the corresponding client has crashed or terminated or the client connection has been lost for some reason). Also, note that the time-to-expiry is recomputed with each successful Refresh request, and thus the value computed here applies only until the first refresh.
Once the allocation is created, the server replies with a success response. The success response contains:	Once the allocation is created, the server replies with a success response. The success response contains:

draft-ietf-tram-turnbis-27.txt	draft-ietf-tram-turnbis-28.txt
o An XOR-RELAYED-ADDRESS attribute containing the relayed transport address.	o An XOR-RELAYED-ADDRESS attribute containing the relayed transport address or two XOR-RELAYED-ADDRESS attributes containing the relayed transport addresses.
o A LIFETIME attribute containing the current value of the time-to-expiry timer.	o A LIFETIME attribute containing the current value of the time-to-expiry timer.
o A RESERVATION-TOKEN attribute (if a second relayed transport address was reserved).	o A RESERVATION-TOKEN attribute (if a second relayed transport address was reserved).
o An XOR-MAPPED-ADDRESS attribute containing the client's IP address and port (from the 5-tuple).	o An XOR-MAPPED-ADDRESS attribute containing the client's IP address and port (from the 5-tuple).
skipping to change at *page 36, line 23*	**skipping to change at *page 36, line 32***
o 508 (Insufficient Capacity): The server has no more relayed transport addresses available, or has none with the requested properties, or the one that was reserved is no longer available. The client considers the current operation as having failed. If the client is using either the EVEN-PORT or the RESERVATION-TOKEN attribute, then the client MAY choose to remove or modify this attribute and try again immediately. Otherwise, the client SHOULD wait at least 1 minute before trying to create any more allocations on this server.	o 508 (Insufficient Capacity): The server has no more relayed transport addresses available, or has none with the requested properties, or the one that was reserved is no longer available. The client considers the current operation as having failed. If the client is using either the EVEN-PORT or the RESERVATION-TOKEN attribute, then the client MAY choose to remove or modify this attribute and try again immediately. Otherwise, the client SHOULD wait at least 1 minute before trying to create any more allocations on this server.
	Note that the error code values 486 and 508 indicate to a eavesdropper that several other users are using the server at this time, similar to that of the HTTP error response code 503, but does not reveal any information about the users using the TURN server.
An unknown error response MUST be handled as described in [I-D.ietf-tram-stunbis].	An unknown error response MUST be handled as described in [I-D.ietf-tram-stunbis].
8. Refreshing an Allocation	8. Refreshing an Allocation
A Refresh transaction can be used to either (a) refresh an existing allocation and update its time-to-expiry or (b) delete an existing allocation.	A Refresh transaction can be used to either (a) refresh an existing allocation and update its time-to-expiry or (b) delete an existing allocation.
If a client wishes to continue using an allocation, then the client	If a client wishes to continue using an allocation, then the client
skipping to change at *page 36, line 49*	**skipping to change at *page 37, line 15***
8.1. Sending a Refresh Request	8.1. Sending a Refresh Request
If the client wishes to immediately delete an existing allocation, it includes a LIFETIME attribute with a value of 0. All other forms of the request refresh the allocation.	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.
When refreshing a dual allocation, the client includes REQUESTED-ADDRESS-FAMILY attribute indicating the address family type that should be refreshed. If no REQUESTED-ADDRESS-FAMILY is included then the request should be treated as applying to all current allocations. The client MUST only include family types it previously allocated and has not yet deleted. This process can also be used to delete an allocation of a specific address type, by setting the lifetime of that refresh request to 0. Deleting a single allocation destroys any permissions or channels associated with that particular allocation;	When refreshing a dual allocation, the client includes REQUESTED-ADDRESS-FAMILY attribute indicating the address family type that should be refreshed. If no REQUESTED-ADDRESS-FAMILY is included then the request should be treated as applying to all current allocations. The client MUST only include the family type it previously allocated and has not yet deleted. This process can also be used to delete an allocation of a specific address type, by setting the lifetime of that refresh request to 0. Deleting a single allocation destroys any permissions or channels associated with that particular allocation;

it MUST NOT affect any permissions or channels associated with allocations for the other address family.	it MUST NOT affect any permissions or channels associated with allocations for the other address family.
The Refresh transaction updates the time-to-expiry timer of an allocation. If the client wishes the server to set the time-to-expiry timer to something other than the default lifetime, it includes a LIFETIME attribute with the requested value. The server	The Refresh transaction updates the time-to-expiry timer of an allocation. If the client wishes the server to set the time-to-expiry timer to something other than the default lifetime, it includes a LIFETIME attribute with the requested value. The server
skipping to change at *page 43, line 35*	**skipping to change at *page 43, line 50***
If the Data indication passes the above checks, the client delivers the data octets inside the DATA attribute to the application, along with an indication that they were received from the peer whose transport address is given by the XOR-PEER-ADDRESS attribute.	If the Data indication passes the above checks, the client delivers the data octets inside the DATA attribute to the application, along with an indication that they were received from the peer whose transport address is given by the XOR-PEER-ADDRESS attribute.
11.5. Receiving an ICMP Packet	11.5. Receiving an ICMP Packet
When the server receives an ICMP packet, the server verifies that the type is either 3 or 11 for an ICMPv4 [RFC0792] packet or either 1, 2, or 3 for an ICMPv6 [RFC4443] packet. It also verifies that the IP packet in the ICMP packet payload contains a UDP header. If either of these conditions fail, then the ICMP packet is silently dropped.	When the server receives an ICMP packet, the server verifies that the type is either 3 or 11 for an ICMPv4 [RFC0792] packet or either 1, 2, or 3 for an ICMPv6 [RFC4443] packet. It also verifies that the IP packet in the ICMP packet payload contains a UDP header, and if a UDP header is present, the server extracts the UDP source and destination IP address and port information. If either of these conditions fail, then the ICMP packet is silently dropped.
The server looks up the allocation whose relayed transport address corresponds to the encapsulated packet's source IP address and UDP port. If no such allocation exists, the packet is silently dropped. The server then checks to see whether the set of permissions for the allocation allows the relaying of the ICMP packet. For ICMP packets, the source IP address MUST NOT be checked against the permissions list as it would be for UDP packets. Instead, the server extracts the destination IP address from the encapsulated IP header. The server then compares this address with the IP address associated with	The server looks up the allocation whose relayed transport address corresponds to the encapsulated packet's source IP address and UDP port. If no such allocation exists, the packet is silently dropped. The server then checks to see whether the set of permissions for the allocation allows the relaying of the ICMP packet. For ICMP packets, the source IP address MUST NOT be checked against the permissions list as it would be for UDP packets. Instead, the server extracts the destination IP address from the encapsulated IP header. The server then compares this address with the IP address associated with
skipping to change at *page 44, line 19*	**skipping to change at *page 44, line 35***
address portion of XOR-PEER-ADDRESS attribute is set to the destination IP address in the encapsulated IP header. At the time of writing of this specification, Socket APIs on some operating systems do not deliver the destination port in the encapsulated UDP header to applications without superuser privileges. If destination port in the encapsulated UDP header is available to the server then the port portion of XOR-PEER-ADDRESS attribute is set to the destination port otherwise the port portion is set to 0. The Data indication is then sent on the 5-tuple associated with the allocation.	address portion of XOR-PEER-ADDRESS attribute is set to the destination IP address in the encapsulated IP header. At the time of writing of this specification, Socket APIs on some operating systems do not deliver the destination port in the encapsulated UDP header to applications without superuser privileges. If destination port in the encapsulated UDP header is available to the server then the port portion of XOR-PEER-ADDRESS attribute is set to the destination port otherwise the port portion is set to 0. The Data indication is then sent on the 5-tuple associated with the allocation.
	Implementation Note: New ICMP types or codes can be defined in future specifications. If the server receives an ICMP error packet, and the new type or code field can help the client to make use of the ICMP error notification and generate feedback to the application layer, the server sends the Data indication with an ICMP attribute conveying the new ICMP type or code.
11.6. Receiving a Data Indication with an ICMP attribute	11.6. Receiving a Data Indication with an ICMP attribute
When the client receives a Data indication with an ICMP attribute, it checks that the Data indication contains an XOR-PEER-ADDRESS attribute, and discards the indication if it does not. The client SHOULD also check that the XOR-PEER-ADDRESS attribute value contains	When the client receives a Data indication with an ICMP attribute, it checks that the Data indication contains an XOR-PEER-ADDRESS attribute, and discards the indication if it does not. The client SHOULD also check that the XOR-PEER-ADDRESS attribute value contains

```
   an IP address with an active permission, and discard the Data          an IP address with an active permission, and discard the Data
   indication otherwise.                                                  indication otherwise.

   If the Data indication passes the above checks, the client signals    If the Data indication passes the above checks, the client signals

              skipping to change at page 45, line 5                              skipping to change at page 45, line 31

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

      0x4000 through 0x4FFF: These values are the allowed channel           0x4000 through 0x4FFF: These values are the allowed channel
      numbers (4096 possible values).                                       numbers (4096 possible values).

      0x5000-0xFFFF: Reserved (For DTLS-SRTP multiplexing collision         0x5000-0xFFFF: Reserved (For DTLS-SRTP multiplexing collision
      avoidance, see [RFC7983].                                             avoidance, see [RFC7983].

                                                                         Note that the channel number range is not backwards compatible with
                                                                         [RFC5766], and cannot be used in environments where such
                                                                         compatibility is required.

   According to [RFC7983], ChannelData messages can be distinguished     According to [RFC7983], ChannelData messages can be distinguished
   from other multiplexed protocols by examining the first byte of the   from other multiplexed protocols by examining the first byte of the
   message:                                                              message:

   +------------+------------------------------+                          +------------+------------------------------+
   | [0..3]     | STUN                         |                          | [0..3]     | STUN                         |
   |            |                              |                          |            |                              |
   +-------------------------------------------+                          +-------------------------------------------+
   | [16..19]   | ZRTP                         |                          | [16..19]   | ZRTP                         |
   |            |                              |                          |            |                              |

              skipping to change at page 47, line 32                             skipping to change at page 48, line 30

   o  The channel number is in the range 0x4000 through 0x4FFF          o  The channel number is in the range 0x4000 through 0x4FFF
      (inclusive);                                                         (inclusive);

   o  The channel number is not currently bound to a different transport o  The channel number is not currently bound to a different transport
      address (same transport address is OK);                              address (same transport address is OK);

   o  The transport address is not currently bound to a different       o  The transport address is not currently bound to a different
      channel number.                                                      channel number.

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

   If any of these tests fail, the server replies with a 400 (Bad       If any of these tests fail, the server replies with a 400 (Bad
   Request) error.                                                       Request) error.  If the XOR-PEER-ADDRESS attribute contains an
                                                                         address of an address family that is not the same as that of a
                                                                         relayed transport address for the allocation, the server MUST
                                                                         generate an error response with the 443 (Peer Address Family
                                                                         Mismatch) response code.

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

If the request is valid, but the server is unable to fulfill the request due to some capacity limit or similar, the server replies with a 508 (Insufficient Capacity) error.	If the request is valid, but the server is unable to fulfill the request due to some capacity limit or similar, the server replies with a 508 (Insufficient Capacity) error.
Otherwise, the server replies with a ChannelBind success response.	Otherwise, the server replies with a ChannelBind success response.
skipping to change at *page 49, line 35*	**skipping to change at *page 50, line 32***
peer.	peer.
The fields of the ChannelData message are filled in as described in Section 12.4.	The fields of the ChannelData message are filled in as described in Section 12.4.
Over TCP and TLS-over-TCP, the ChannelData message MUST be padded to a multiple of four bytes in order to ensure the alignment of subsequent messages. The padding is not reflected in the length field of the ChannelData message, so the actual size of a ChannelData message (including padding) is (4 + Length) rounded up to the nearest multiple of 4. Over UDP, the padding is not required but MAY be included.	Over TCP and TLS-over-TCP, the ChannelData message MUST be padded to a multiple of four bytes in order to ensure the alignment of subsequent messages. The padding is not reflected in the length field of the ChannelData message, so the actual size of a ChannelData message (including padding) is (4 + Length) rounded up to the nearest multiple of 4 (See section 14 in [I-D.ietf-tram-stunbis]). 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.	The ChannelData message is then sent on the 5-tuple associated with the allocation.
12.6. Receiving a ChannelData Message	12.6. Receiving a ChannelData Message
The receiver of the ChannelData message uses the first byte to distinguish it from other multiplexed protocols, as described in Figure 5. If the message uses a value in the reserved range (0x5000 through 0xFFFF), then the message is silently discarded.	The receiver of the ChannelData message uses the first byte to distinguish it from other multiplexed protocols, as described in Figure 5. If the message uses a value in the reserved range (0x5000 through 0xFFFF), then the message is silently discarded.
skipping to change at *page 50, line 36*	**skipping to change at *page 51, line 34***
o the destination transport address is the transport address to which the channel is bound;	o the destination transport address is the transport address to which the channel is bound;
o the data following the UDP header is the contents of the data field of the ChannelData message.	o the data following the UDP header is the contents of the data field of the ChannelData message.
The resulting UDP datagram is then sent to the peer. Note that if the Length field in the ChannelData message is 0, then there will be no data in the UDP datagram, but the UDP datagram is still formed and sent.	The resulting UDP datagram is then sent to the peer. Note that if the Length field in the ChannelData message is 0, then there will be no data in the UDP datagram, but the UDP datagram is still formed and sent (Section 4.1 in [RFC6263]).
12.7. Relaying Data from the Peer	12.7. Relaying Data from the Peer
When the server receives a UDP datagram on the relayed transport address associated with an allocation, the server processes it as described in Section 11.3. If that section indicates that a ChannelData message should be sent (because there is a channel bound to the peer that sent to the UDP datagram), then the server forms and sends a ChannelData message as described in Section 12.5.	When the server receives a UDP datagram on the relayed transport address associated with an allocation, the server processes it as described in Section 11.3. If that section indicates that a ChannelData message should be sent (because there is a channel bound to the peer that sent to the UDP datagram), then the server forms and sends a ChannelData message as described in Section 12.5.
When the server receives an ICMP packet, the server processes it as described in Section 11.5. A Data indication MUST be sent regardless of whether there is a channel bound to the peer that was the destination of the UDP datagram that triggered the reception of the ICMP packet.	When the server receives an ICMP packet, the server processes it as described in Section 11.5.

13. Packet Translations	13. Packet Translations
This section addresses IPv4-to-IPv6, IPv6-to-IPv4, and IPv6-to-IPv6 translations. Requirements for translation of the IP addresses and port numbers of the packets are described above. The following sections specify how to translate other header fields.	This section addresses IPv4-to-IPv6, IPv6-to-IPv4, and IPv6-to-IPv6 translations. Requirements for translation of the IP addresses and port numbers of the packets are described above. The following sections specify how to translate other header fields.
As discussed in Section 3.6, translations in TURN are designed so that a TURN server can be implemented as an application that runs in user space under commonly available operating systems and that does not require special privileges. The translations specified in the following sections follow this principle.	As discussed in Section 3.6, translations in TURN are designed so that a TURN server can be implemented as an application that runs in user space under commonly available operating systems and that does not require special privileges. The translations specified in the following sections follow this principle.
The descriptions below have two parts: a preferred behavior and an alternate behavior. The server SHOULD implement the preferred behavior. Otherwise, the server MUST implement the alternate behavior and MUST NOT do anything else for the reasons detailed in [RFC7915]. The TURN server solely relies on the DF bit in the IPv4 header and Fragmentation header in IPv6 header to handle fragmentation and does not rely on the DONT-FRAGMENT attribute.	The descriptions below have two parts: a preferred behavior and an alternate behavior. The server SHOULD implement the preferred behavior, but if that is not possible for a particular field, the server MUST implement the alternate behavior and MUST NOT do anything else for the reasons detailed in [RFC7915]. The TURN server solely relies on the DF bit in the IPv4 header and Fragmentation header in IPv6 header to handle fragmentation and does not rely on the DONT-FRAGMENT attribute.
13.1. IPv4-to-IPv6 Translations	13.1. IPv4-to-IPv6 Translations
Time to Live (TTL) field	Time to Live (TTL) field
Preferred Behavior: As specified in Section 4 of [RFC7915].	Preferred Behavior: As specified in Section 4 of [RFC7915].
Alternate Behavior: Set the outgoing value to the default for outgoing packets.	Alternate Behavior: Set the outgoing value to the default for outgoing packets.
skipping to change at *page 53, line 22*	**skipping to change at *page 54, line 19***
Preferred behavior: If the incoming packet did not include a Fragment header and the outgoing packet size does not exceed the outgoing link's MTU, the TURN server sends the outgoing packet without a Fragment header.	Preferred behavior: If the incoming packet did not include a Fragment header and the outgoing packet size does not exceed the outgoing link's MTU, the TURN server sends the outgoing packet without a Fragment header.
If the incoming packet did not include a Fragment header and the outgoing packet size exceeds the outgoing link's MTU, the TURN server drops the outgoing packet and send an ICMP message of type 2 code 0 ("Packet too big") to the sender of the incoming packet. If the packet is being sent to the peer, the TURN server reduces the MTU reported in the ICMP message by 48 bytes to allow room for the overhead of a Data indication.	If the incoming packet did not include a Fragment header and the outgoing packet size exceeds the outgoing link's MTU, the TURN server drops the outgoing packet and send an ICMP message of type 2 code 0 ("Packet too big") to the sender of the incoming packet. If the ICMPv6 packet ("Packet too big") is being sent to the peer, the TURN server reduces the MTU reported in the ICMP message by 48 bytes to allow room for the overhead of a Data indication.
If the incoming packet included a Fragment header and the outgoing packet size (with a Fragment header included) does not exceed the outgoing link's MTU, the TURN server sends the outgoing packet with a Fragment header. The TURN server sets the fields of the Fragment header as appropriate for a packet originating from the server.	If the incoming packet included a Fragment header and the outgoing packet size (with a Fragment header included) does not exceed the outgoing link's MTU, the TURN server sends the outgoing packet with a Fragment header. The TURN server sets the fields of the Fragment header as appropriate for a packet originating from the server.
If the incoming packet included a Fragment header and the outgoing packet size exceeds the outgoing link's MTU, the TURN server MUST	If the incoming packet included a Fragment header and the outgoing packet size exceeds the outgoing link's MTU, the TURN server MUST

skipping to change at *page 54, line 28*

```
      Preferred behavior: As specified in Section 5 of [RFC7915].

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

   Fragmentation

      Preferred behavior: As specified in Section 5 of [RFC7915].
      Additionally, when the outgoing packet's size exceeds the outgoing
      link's MTU, the TURN server needs to generate an ICMP error
      (ICMPv6 Packet Too Big) reporting the MTU size.  If the packet is
      being sent to the peer, the TURN server SHOULD reduce the MTU
      reported in the ICMP message by 48 bytes to allow room for the
      overhead of a Data indication.

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

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

14.  UDP-to-UDP relay
```

skipping to change at *page 55, line 26*

```
      Preferred behavior: As specified in Section 5 of [RFC7915].

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

   Fragmentation

      Preferred behavior: As specified in Section 5 of [RFC7915].
      Additionally, when the outgoing packet's size exceeds the outgoing
      link's MTU, the TURN server needs to generate an ICMP error
      (ICMPv6 Packet Too Big) reporting the MTU size.  If the ICMPv4
      packet (Destination Unreachable (Type 3) with Code 4) is being
      sent to the peer, the TURN server SHOULD reduce the MTU reported
      in the ICMP message by 48 bytes to allow room for the overhead of
      a Data indication.

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

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

14.  UDP-to-UDP relay
```

skipping to change at *page 56, line 24*

```
      Alternate Behavior: Same as preferred.

15.  TCP-to-UDP relay

   This section describes how the server sets various fields in the IP
   header for TCP-to-UDP relay from the client to the peer.  The
   descriptions in this section apply when the server sends a UDP
   datagram to the peer.  Note that the server does not perform per-
   packet translation for TCP-to-UDP relaying.

   TCP multi-path [RFC6824] is not supported by this version of TURN
   because TCP multi-path is not used by both SIP and WebRTC protocols
   [RFC7478] for media and non-media data.  TCP connection between the
   TURN client and server can use TCP-AO [RFC5925] but UDP does not
   provide a similar type of authentication until UDP supports a similar
   authentication option [I-D.ietf-tsvwg-udp-options].  Even if both
   TCP-AO and UDP authentication would be used between TURN client and
   server, it would not change the end-to-end security properties of the
   UDP payload being relayed.  Therefore applications using TURN will
   need to secure their application data end-to-end appropriately, e.g.,
   SRTP for RTP applications.  Note that TCP-AO option obsoletes TCP MD5
   option.  Unlike UDP, TCP without the TCP Fast Open extension
   [RFC7413] does not support 0-RTT session resumption.  The TCP user
   timeout [RFC5482] equivalent for application data relayed by the TURN
   is the use of RTP control protocol (RTCP).  As a reminder, RTCP is a
   fundamental and integral part of RTP.

   The descriptions below have two parts: a preferred behavior and an
   alternate behavior.  The server SHOULD implement the preferred
```

skipping to change at *page 57, line 20*

```
      Alternate Behavior: Same as preferred.

15.  TCP-to-UDP relay

   This section describes how the server sets various fields in the IP
   header for TCP-to-UDP relay from the client to the peer.  The
   descriptions in this section apply when the server sends a UDP
   datagram to the peer.  Note that the server does not perform per-
   packet translation for TCP-to-UDP relaying.

   Multipath TCP [I-D.ietf-mptcp-rfc6824bis] is not supported by this
   version of TURN because TCP multi-path is not used by both SIP and
   WebRTC protocols [RFC7478] for media and non-media data.  TCP
   connection between the TURN client and server can use TCP-AO
   [RFC5925] but UDP does not provide a similar type of authentication
   until UDP supports a similar authentication option
   [I-D.ietf-tsvwg-udp-options].  Even if both TCP-AO and UDP
   authentication would be used between TURN client and server, it would
   not change the end-to-end security properties of the application
   payload being relayed.  Therefore applications using TURN will need
   to secure their application data end-to-end appropriately, e.g., SRTP
   for RTP applications.  Note that TCP-AO option obsoletes TCP MD5
   option.  Unlike UDP, TCP without the TCP Fast Open extension
   [RFC7413] does not support 0-RTT session resumption.  The TCP user
   timeout [RFC5482] equivalent for application data relayed by the TURN
   is the use of RTP control protocol (RTCP).  As a reminder, RTCP is a
   fundamental and integral part of RTP.

   The descriptions below have two parts: a preferred behavior and an
   alternate behavior.  The server SHOULD implement the preferred
```

draft-ietf-tram-turnbis-27.txt	draft-ietf-tram-turnbis-28.txt
behavior, but if that is not possible for a particular field, then it SHOULD implement the alternative behavior.	behavior, but if that is not possible for a particular field, then it SHOULD implement the alternative behavior.
skipping to change at *page 57, line 42*	**skipping to change at *page 58, line 38***
all other cases when sending an outgoing UDP packet containing application data (e.g., Data indication, ChannelData message, or DONT-FRAGMENT attribute not included in the Send indication), set the DF bit in the outgoing IP header to 0.	all other cases when sending an outgoing UDP packet containing application data (e.g., Data indication, ChannelData message, or DONT-FRAGMENT attribute not included in the Send indication), set the DF bit in the outgoing IP header to 0.
Alternate Behavior: Same as preferred.	Alternate Behavior: Same as preferred.
IPv6 Fragmentation fields	IPv6 Fragmentation fields
Preferred Behavior: If the TCP traffic arrives over IPv6, the server relies on the presence of DON'T-FRAGMENT attribute in the send indication to set the outgoing UDP packet to not fragment.	Preferred Behavior: If the TCP traffic arrives over IPv6, the server relies on the presence of DONT-FRAGMENT attribute in the send indication to set the outgoing UDP packet to not fragment.
Alternate Behavior: Same as preferred.	Alternate Behavior: Same as preferred.
IPv4 Options	IPv4 Options
Preferred Behavior: The outgoing packet uses the system defaults for IPv4 options.	Preferred Behavior: The outgoing packet uses the system defaults for IPv4 options.
Alternate Behavior: Same as preferred.	Alternate Behavior: Same as preferred.
16. UDP-to-TCP relay	16. UDP-to-TCP relay
This section describes how the server sets various fields in the IP header for UDP-to-TCP relay from the peer to the client. The descriptions in this section apply when the server sends a Data	This section describes how the server sets various fields in the IP header for UDP-to-TCP relay from the peer to the client. The descriptions in this section apply when the server sends a Data
skipping to change at *page 58, line 47*	**skipping to change at *page 59, line 43***
Preferred Behavior: Ignore, ECN signals are dropped in the TURN server for the incoming UDP datagrams from the peer.	Preferred Behavior: Ignore, ECN signals are dropped in the TURN server for the incoming UDP datagrams from the peer.
Alternate Behavior: Same as preferred.	Alternate Behavior: Same as preferred.
Fragmentation	Fragmentation
Preferred Behavior: Any fragmented packets are reassembled in the server and then forwarded to the client over the TCP connection. ICMP messages resulting from the UDP datagrams sent to the peer MUST be forwarded to the client using TURN's mechanism for relevant ICMP types and codes.	Preferred Behavior: Any fragmented packets are reassembled in the server and then forwarded to the client over the TCP connection. ICMP messages resulting from the UDP datagrams sent to the peer are processed by the server as described in Section 11.5 and forwarded to the client using TURN's mechanism for relevant ICMP types and codes.
Alternate Behavior: Same as preferred.	Alternate Behavior: Same as preferred.
Extension Headers	Extension Headers
Preferred behavior: The outgoing packet uses the system defaults for IPv6 extension headers.	Preferred behavior: The outgoing packet uses the system defaults for IPv6 extension headers.
Alternate behavior: Same as preferred.	Alternate behavior: Same as preferred.
IPv4 Options	IPv4 Options

Preferred Behavior: The outgoing packet uses the system defaults for IPv4 options.	Preferred Behavior: The outgoing packet uses the system defaults for IPv4 options.
skipping to change at *page 61, line 14*	**skipping to change at *page 61, line 40***
18.3. XOR-PEER-ADDRESS	18.3. XOR-PEER-ADDRESS
The XOR-PEER-ADDRESS specifies the address and port of the peer as seen from the TURN server. (For example, the peer's server-reflexive transport address if the peer is behind a NAT.) It is encoded in the same way as XOR-MAPPED-ADDRESS [I-D.ietf-tram-stunbis].	The XOR-PEER-ADDRESS specifies the address and port of the peer as seen from the TURN server. (For example, the peer's server-reflexive transport address if the peer is behind a NAT.) It is encoded in the same way as XOR-MAPPED-ADDRESS [I-D.ietf-tram-stunbis].
18.4. DATA	18.4. DATA
The DATA attribute is present in all Send and Data indications. The value portion of this attribute is variable length and consists of the application data (that is, the data that would immediately follow the UDP header if the data was been sent directly between the client and the peer). The application data is equivalent to the "UDP user data" and does not include the "surplus area" defined in Section 4 of [I-D.ietf-tsvwg-udp-options]. If the length of this attribute is not a multiple of 4, then padding must be added after this attribute.	The DATA attribute is present in all Send indications. If the ICMP attribute is not present in a Data indication, it contains a DATA attribute. The value portion of this attribute is variable length and consists of the application data (that is, the data that would immediately follow the UDP header if the data was been sent directly between the client and the peer). The application data is equivalent to the "UDP user data" and does not include the "surplus area" defined in Section 4 of [I-D.ietf-tsvwg-udp-options]. If the length of this attribute is not a multiple of 4, then padding must be added after this attribute.
18.5. XOR-RELAYED-ADDRESS	18.5. XOR-RELAYED-ADDRESS
The XOR-RELAYED-ADDRESS is present in Allocate responses. It specifies the address and port that the server allocated to the client. It is encoded in the same way as XOR-MAPPED-ADDRESS [I-D.ietf-tram-stunbis].	The XOR-RELAYED-ADDRESS is present in Allocate responses. It specifies the address and port that the server allocated to the client. It is encoded in the same way as XOR-MAPPED-ADDRESS [I-D.ietf-tram-stunbis].
18.6. REQUESTED-ADDRESS-FAMILY	18.6. REQUESTED-ADDRESS-FAMILY
skipping to change at *page 62, line 26*	**skipping to change at *page 63, line 5***
+-+-+-+-+-+-+-+-+	+-+-+-+-+-+-+-+-+
The value contains a single 1-bit flag:	The value contains a single 1-bit flag:
R: If 1, the server is requested to reserve the next-higher port number (on the same IP address) for a subsequent allocation. If 0, no such reservation is requested.	R: If 1, the server is requested to reserve the next-higher port number (on the same IP address) for a subsequent allocation. If 0, no such reservation is requested.
RFFU: Reserved For Future Use.	RFFU: Reserved For Future Use.
The other 7 bits of the attribute's value must be set to zero on transmission and ignored on reception.	The RFFU field must be set to zero on transmission and ignored on reception.
Since the length of this attribute is not a multiple of 4, padding must immediately follow this attribute.	Since the length of this attribute is not a multiple of 4, padding must immediately follow this attribute.
18.8. REQUESTED-TRANSPORT	18.8. REQUESTED-TRANSPORT
This attribute is used by the client to request a specific transport protocol for the allocated transport address. The value of this attribute is 4 bytes with the following format:	This attribute is used by the client to request a specific transport protocol for the allocated transport address. The value of this attribute is 4 bytes with the following format:

skipping to change at *page 63, line 33*

```
18.11.  ADDITIONAL-ADDRESS-FAMILY

   This attribute is used by clients to request the allocation of a IPv4
   and IPv6 address type from a server.  It is encoded in the same way
   as REQUESTED-ADDRESS-FAMILY Section 18.6.  The ADDITIONAL-ADDRESS-
   FAMILY attribute MAY be present in Allocate request.  The attribute
   value of 0x02 (IPv6 address) is the only valid value in Allocate
   request.

18.12.  ADDRESS-ERROR-CODE Attribute

   This attribute is used by servers to signal the reason for not
   allocating the requested address family.  The value portion of this
   attribute is variable length with the following format:

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

   Family:  there are two values defined for this field and specified in
      [I-D.ietf-tram-stunbis], Section 14.1: 0x01 for IPv4 addresses and
      0x02 for IPv6 addresses.

   Reserved:  at this point, the 13 bits in the Reserved field MUST be
      set to zero by the client and MUST be ignored by the server.

   Class:  The Class represents the hundreds digit of the error code and
      is defined in section 14.8 of [I-D.ietf-tram-stunbis].

   Number:  this 8-bit field contains the reason server cannot allocate
      one of the requested address types.  The error code values could
      be either 440 (unsupported address family) or 508 (insufficient
      capacity).  The number representation is defined in section 14.8
      of [I-D.ietf-tram-stunbis].

   Reason Phrase:  The recommended reason phrases for error codes 440
      and 508 are explained in Section 19.  The reason phrase MUST be a
      UTF-8 [RFC3629] encoded sequence of less than 128 characters
      (which can be as long as 509 bytes when encoding them or 763 bytes
      when decoding them).

18.13.  ICMP Attribute

   This attribute is used by servers to signal the reason an UDP packet
   was dropped.  The following is the format of the ICMP attribute.

        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
       +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
       |  Reserved                     |  ICMP Type  |  ICMP Code      |
       +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
       |                          Error Data                           |
```

skipping to change at *page 64, line 14*

```
18.11.  ADDITIONAL-ADDRESS-FAMILY

   This attribute is used by clients to request the allocation of a IPv4
   and IPv6 address type from a server.  It is encoded in the same way
   as REQUESTED-ADDRESS-FAMILY Section 18.6.  The ADDITIONAL-ADDRESS-
   FAMILY attribute MAY be present in Allocate request.  The attribute
   value of 0x02 (IPv6 address) is the only valid value in Allocate
   request.

18.12.  ADDRESS-ERROR-CODE

   This attribute is used by servers to signal the reason for not
   allocating the requested address family.  The value portion of this
   attribute is variable length with the following format:

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

   Family:  there are two values defined for this field and specified in
      [I-D.ietf-tram-stunbis], Section 14.1: 0x01 for IPv4 addresses and
      0x02 for IPv6 addresses.

   Reserved:  at this point, the 13 bits in the Reserved field MUST be
      set to zero by the server and MUST be ignored by the client.

   Class:  The Class represents the hundreds digit of the error code and
      is defined in section 14.8 of [I-D.ietf-tram-stunbis].

   Number:  this 8-bit field contains the reason server cannot allocate
      one of the requested address types.  The error code values could
      be either 440 (unsupported address family) or 508 (insufficient
      capacity).  The number representation is defined in section 14.8
      of [I-D.ietf-tram-stunbis].

   Reason Phrase:  The recommended reason phrases for error codes 440
      and 508 are explained in Section 19.  The reason phrase MUST be a
      UTF-8 [RFC3629] encoded sequence of less than 128 characters
      (which can be as long as 509 bytes when encoding them or 763 bytes
      when decoding them).

18.13.  ICMP

   This attribute is used by servers to signal the reason an UDP packet
   was dropped.  The following is the format of the ICMP attribute.

        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
       +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
       |  Reserved                     |  ICMP Type  |  ICMP Code      |
       +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
       |                          Error Data                           |
       +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+

                    skipping to change at page 66, line 10
   diagram shown in the Overview (Figure 1).

   For each message, the attributes included in the message and their
   values are shown.  For convenience, values are shown in a human-
   readable format rather than showing the actual octets; for example,
   "XOR-RELAYED-ADDRESS=192.0.2.15:9000" shows that the XOR-RELAYED-
   ADDRESS attribute is included with an address of 192.0.2.15 and a
   port of 9000, here the address and port are shown before the xor-ing
   is done.  For attributes with string-like values (e.g.,
   SOFTWARE="Example client, version 1.03" and
   NONCE="obMatJos2AAABadl7W7PeDU4hKE72jda"), the value of the attribute
   is shown in quotes for readability, but these quotes do not appear in
   the actual value.

  TURN                                 TURN           Peer          Peer
  client                               server          A             B
    |                                    |             |             |
    |--- Allocate request -------------->|             |             |
    |    Transaction-Id=0xA56250D3F17ABE679422DE85     |             |
    |    SOFTWARE="Example client, version 1.03"       |             |
    |    LIFETIME=3600 (1 hour)          |             |             |
    |    REQUESTED-TRANSPORT=17 (UDP)    |             |             |
    |    DONT-FRAGMENT                   |             |             |
    |                                    |             |             |
    |<-- Allocate error response --------|             |             |
    |    Transaction-Id=0xA56250D3F17ABE679422DE85     |             |
    |    SOFTWARE="Example server, version 1.17"       |             |
    |    ERROR-CODE=401 (Unauthorized)   |             |             |
    |    REALM="example.com"             |             |             |
    |    NONCE="obMatJos2AAABadl7W7PeDU4hKE72jda"      |             |
    |    PASSWORD-ALGORITHMS=MD5 and SHA256            |             |
    |                                    |             |             |
    |--- Allocate request -------------->|             |             |
    |    Transaction-Id=0xC271E932AD7446A32C234492     |             |
    |    SOFTWARE="Example client 1.03"  |             |             |
    |    LIFETIME=3600 (1 hour)          |             |             |
    |    REQUESTED-TRANSPORT=17 (UDP)    |             |             |
    |    DONT-FRAGMENT                   |             |             |
    |    USERNAME="George"               |             |             |
    |    REALM="example.com"             |             |             |
    |    NONCE="obMatJos2AAABadl7W7PeDU4hKE72jda"      |             |
    |    PASSWORD-ALGORITHMS=MD5 and SHA256            |             |
    |    PASSWORD-ALGORITHM=SHA256       |             |             |
    |    MESSAGE-INTEGRITY=...           |             |             |
    |    MESSAGE-INTEGRITY-SHA256=...    |             |             |
    |                                    |             |             |
    |<-- Allocate success response ------|             |             |
    |    Transaction-Id=0xC271E932AD7446A32C234492     |             |
    |    SOFTWARE="Example server, version 1.17"       |             |
    |    LIFETIME=1200 (20 minutes)      |             |             |
    |    XOR-RELAYED-ADDRESS=192.0.2.15:50000          |             |
    |    XOR-MAPPED-ADDRESS=192.0.2.1:7000             |             |
    |    MESSAGE-INTEGRITY=...           |             |             |

   The client begins by selecting a host transport address to use for
   the TURN session; in this example, the client has selected
```

```
       +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+

                    skipping to change at page 66, line 41
   diagram shown in the Overview (Figure 1).

   For each message, the attributes included in the message and their
   values are shown.  For convenience, values are shown in a human-
   readable format rather than showing the actual octets; for example,
   "XOR-RELAYED-ADDRESS=192.0.2.15:9000" shows that the XOR-RELAYED-
   ADDRESS attribute is included with an address of 192.0.2.15 and a
   port of 9000, here the address and port are shown before the xor-ing
   is done.  For attributes with string-like values (e.g.,
   SOFTWARE="Example client, version 1.03" and
   NONCE="obMatJos2gAAAadl7W7PeDU4hKE72jda"), the value of the attribute
   is shown in quotes for readability, but these quotes do not appear in
   the actual value.

  TURN                                 TURN           Peer          Peer
  client                               server          A             B
    |                                    |             |             |
    |--- Allocate request -------------->|             |             |
    |    Transaction-Id=0xA56250D3F17ABE679422DE85     |             |
    |    SOFTWARE="Example client, version 1.03"       |             |
    |    LIFETIME=3600 (1 hour)          |             |             |
    |    REQUESTED-TRANSPORT=17 (UDP)    |             |             |
    |    DONT-FRAGMENT                   |             |             |
    |                                    |             |             |
    |<-- Allocate error response --------|             |             |
    |    Transaction-Id=0xA56250D3F17ABE679422DE85     |             |
    |    SOFTWARE="Example server, version 1.17"       |             |
    |    ERROR-CODE=401 (Unauthorized)   |             |             |
    |    REALM="example.com"             |             |             |
    |    NONCE="obMatJos2gAAAadl7W7PeDU4hKE72jda"      |             |
    |    PASSWORD-ALGORITHMS=MD5 and SHA256            |             |
    |                                    |             |             |
    |--- Allocate request -------------->|             |             |
    |    Transaction-Id=0xC271E932AD7446A32C234492     |             |
    |    SOFTWARE="Example client 1.03"  |             |             |
    |    LIFETIME=3600 (1 hour)          |             |             |
    |    REQUESTED-TRANSPORT=17 (UDP)    |             |             |
    |    DONT-FRAGMENT                   |             |             |
    |    USERNAME="George"               |             |             |
    |    REALM="example.com"             |             |             |
    |    NONCE="obMatJos2gAAAadl7W7PeDU4hKE72jda"      |             |
    |    PASSWORD-ALGORITHMS=MD5 and SHA256            |             |
    |    PASSWORD-ALGORITHM=SHA256       |             |             |
    |    MESSAGE-INTEGRITY=...           |             |             |
    |    MESSAGE-INTEGRITY-SHA256=...    |             |             |
    |                                    |             |             |
    |<-- Allocate success response ------|             |             |
    |    Transaction-Id=0xC271E932AD7446A32C234492     |             |
    |    SOFTWARE="Example server, version 1.17"       |             |
    |    LIFETIME=1200 (20 minutes)      |             |             |
    |    XOR-RELAYED-ADDRESS=192.0.2.15:50000          |             |
    |    XOR-MAPPED-ADDRESS=192.0.2.1:7000             |             |
    |    MESSAGE-INTEGRITY-SHA256=...    |             |             |

   The client begins by selecting a host transport address to use for
   the TURN session; in this example, the client has selected
   198.51.100.2:49721 as shown in Figure 1.  The client then sends an
   Allocate request to the server at the server transport address.  The
   client randomly selects a 96-bit transaction id of
   0xA56250D3F17ABE679422DE85 for this transaction; this is encoded in
   the transaction id field in the fixed header.  The client includes a
   SOFTWARE attribute that gives information about the client's
   software; here the value is "Example client, version 1.03" to
```

```
   198.51.100.2:49721 as shown in Figure 1.  The client then sends an
   Allocate request to the server at the server transport address.  The
   client randomly selects a 96-bit transaction id of
   0xA56250D3F17ABE679422DE85 for this transaction; this is encoded in
   the transaction id field in the fixed header.  The client includes a
   SOFTWARE attribute that gives information about the client's
   software; here the value is "Example client, version 1.03" to
```

skipping to change at *page 68, line 51* **skipping to change at *page 68, line 51***

```
   The client, upon receipt of the 401 error, re-attempts the Allocate
   request, this time including the authentication attributes.  The
   client selects a new transaction id, and then populates the new
   Allocate request with the same attributes as before.  The client
   includes a USERNAME attribute and uses the realm value received from
   the server to help it determine which value to use; here the client
   is configured to use the username "George" for the realm
   "example.com".  The client includes the PASSWORD-ALGORITHM attribute
   indicating the algorithm that the server must use to derive the long-
   term password.  The client also includes the REALM and NONCE
   attributes, which are just copied from the 401 error response.
   Finally, the client includes MESSAGE-INTEGRITY and MESSAGE-INTEGRITY-
   SHA256 attributes as the last attributes in the message, whose values
   are Hashed Message Authentication Code - Secure Hash Algorithm 1
   (HMAC-SHA1) hash and Hashed Message Authentication Code - Secure Hash
   Algorithm 2 (HMAC-SHA2) hash over the contents of the message (shown
   as just "..." above); this HMAC-SHA1 and HMAC-SHA2 computation
   includes a password value.  Thus, an attacker cannot compute the
   message integrity value without somehow knowing the secret password.
```

```
   The client, upon receipt of the 401 error, re-attempts the Allocate
   request, this time including the authentication attributes.  The
   client selects a new transaction id, and then populates the new
   Allocate request with the same attributes as before.  The client
   includes a USERNAME attribute and uses the realm value received from
   the server to help it determine which value to use; here the client
   is configured to use the username "George" for the realm
   "example.com".  The client includes the PASSWORD-ALGORITHM attribute
   indicating the algorithm that the server must use to derive the long-
   term password.  The client also includes the REALM, PASSWORD-
   ALGORITHMS and NONCE attributes, which are just copied from the 401
   error response.  Finally, the client includes MESSAGE-INTEGRITY-
   SHA256 attribute as the last attributes in the message, whose value
   is Hashed Message Authentication Code - Secure Hash Algorithm 2
   (HMAC-SHA2) hash over the contents of the message (shown as just
   "..." above); this HMAC-SHA2 computation includes a password value.
   Thus, an attacker cannot compute the message integrity value without
   somehow knowing the secret password.
```

```
   The server, upon receipt of the authenticated Allocate request,
   checks that everything is OK, then creates an allocation.  The server
   replies with an Allocate success response.  The server includes a
   LIFETIME attribute giving the lifetime of the allocation; here, the
   server has reduced the client's requested 1-hour lifetime to just 20
   minutes, because this particular server doesn't allow lifetimes
   longer than 20 minutes.  The server includes an XOR-RELAYED-ADDRESS
   attribute whose value is the relayed transport address of the
   allocation.  The server includes an XOR-MAPPED-ADDRESS attribute
   whose value is the server-reflexive address of the client; this value
   is not used otherwise in TURN but is returned as a convenience to the
   client.  The server includes either a MESSAGE-INTEGRITY or MESSAGE-
   INTEGRITY-SHA256 attribute to authenticate the response and to ensure
   its integrity; note that the response does not contain the USERNAME,
   REALM, and NONCE attributes.  The server also includes a SOFTWARE
   attribute.
```

```
   The server, upon receipt of the authenticated Allocate request,
   checks that everything is OK, then creates an allocation.  The server
   replies with an Allocate success response.  The server includes a
   LIFETIME attribute giving the lifetime of the allocation; here, the
   server has reduced the client's requested 1-hour lifetime to just 20
   minutes, because this particular server doesn't allow lifetimes
   longer than 20 minutes.  The server includes an XOR-RELAYED-ADDRESS
   attribute whose value is the relayed transport address of the
   allocation.  The server includes an XOR-MAPPED-ADDRESS attribute
   whose value is the server-reflexive address of the client; this value
   is not used otherwise in TURN but is returned as a convenience to the
   client.  The server includes a MESSAGE-INTEGRITY-SHA256 attribute to
   authenticate the response and to ensure its integrity; note that the
   response does not contain the USERNAME, REALM, and NONCE attributes.
   The server also includes a SOFTWARE attribute.
```

```
  TURN                                 TURN          Peer          Peer
  client                               server         A             B
    |--- CreatePermission request ------>|             |             |
    |    Transaction-Id=0xE5913A8F460956CA277D3319     |             |
    |    XOR-PEER-ADDRESS=192.0.2.150:0  |             |             |
    |    USERNAME="George"               |             |             |
    |    REALM="example.com"             |             |             |
    |    NONCE="obMatJos2AAABadl7W7PeDU4hKE72jda"      |             |
    |    MESSAGE-INTEGRITY=...           |             |             |

    |                                    |             |             |
```

```
  TURN                                 TURN          Peer          Peer
  client                               server         A             B
    |--- CreatePermission request ------>|             |             |
    |    Transaction-Id=0xE5913A8F460956CA277D3319     |             |
    |    XOR-PEER-ADDRESS=192.0.2.150:0  |             |             |
    |    USERNAME="George"               |             |             |
    |    REALM="example.com"             |             |             |
    |    NONCE="obMatJos2gAAAadl7W7PeDU4hKE72jda"      |             |
    |    PASSWORD-ALGORITHMS=MD5 and SHA256            |             |
    |    PASSWORD-ALGORITHM=SHA256       |             |             |
    |    MESSAGE-INTEGRITY-SHA256=...    |             |             |
    |                                    |             |             |
    |<-- CreatePermission success resp.--|             |              |
    |    Transaction-Id=0xE5913A8F460956CA277D3319     |              |
    |    MESSAGE-INTEGRITY=...           |             |              |
```

The client then creates a permission towards Peer A in preparation for sending it some application data. This is done through a CreatePermission request. The XOR-PEER-ADDRESS attribute contains the IP address for which a permission is established (the IP address of peer A); note that the port number in the attribute is ignored when used in a CreatePermission request, and here it has been set to 0; also, note how the client uses Peer A's server-reflexive IP address and not its (private) host address. The client uses the same username, realm, and nonce values as in the previous request on the allocation. Though it is allowed to do so, the client has chosen not to include a SOFTWARE attribute in this request.

The server receives the CreatePermission request, creates the corresponding permission, and then replies with a CreatePermission success response. Like the client, the server chooses not to include the SOFTWARE attribute in its reply. Again, note how success responses contain a MESSAGE-INTEGRITY or MESSAGE-INTEGRITY-SHA256 attribute (assuming the server uses the long-term credential mechanism), but no USERNAME, REALM, and NONCE attributes.

```
  TURN                                 TURN           Peer          Peer
  client                               server          A             B
    |--- Send indication ---------------->|             |              |
    |    Transaction-Id=0x1278E9ACA2711637EF7D3328     |              |
    |    XOR-PEER-ADDRESS=192.0.2.150:32102            |              |
    |    DONT-FRAGMENT                   |             |              |
    |    DATA=...                        |             |              |
    |                                    |-- UDP dgm ->|              |
    |                                    |  data=...   |              |
```

```
    |<-- CreatePermission success resp.--|             |              |
    |    Transaction-Id=0xE5913A8F460956CA277D3319     |              |
    |    MESSAGE-INTEGRITY-SHA256=...    |             |              |
```

The client then creates a permission towards Peer A in preparation for sending it some application data. This is done through a CreatePermission request. The XOR-PEER-ADDRESS attribute contains the IP address for which a permission is established (the IP address of peer A); note that the port number in the attribute is ignored when used in a CreatePermission request, and here it has been set to 0; also, note how the client uses Peer A's server-reflexive IP address and not its (private) host address. The client uses the same username, realm, and nonce values as in the previous request on the allocation. Though it is allowed to do so, the client has chosen not to include a SOFTWARE attribute in this request.

The server receives the CreatePermission request, creates the corresponding permission, and then replies with a CreatePermission success response. Like the client, the server chooses not to include the SOFTWARE attribute in its reply. Again, note how success responses contains a MESSAGE-INTEGRITY-SHA256 attribute (assuming the server uses the long-term credential mechanism), but no USERNAME, REALM, and NONCE attributes.

```
  TURN                                 TURN           Peer          Peer
  client                               server          A             B
    |--- Send indication ---------------->|             |              |
    |    Transaction-Id=0x1278E9ACA2711637EF7D3328     |              |
    |    XOR-PEER-ADDRESS=192.0.2.150:32102            |              |
    |    DONT-FRAGMENT                   |             |              |
    |    DATA=...                        |             |              |
    |                                    |-- UDP dgm ->|              |
    |                                    |  data=...   |              |
```

skipping to change at *page 71, line 22*

The resulting Data indication is then sent to the client.

```
  TURN                                 TURN           Peer          Peer
  client                               server          A             B
    |--- ChannelBind request ----------->|             |              |
    |    Transaction-Id=0x6490D3BC175AFF3D84513212     |              |
    |    CHANNEL-NUMBER=0x4000           |             |              |
    |    XOR-PEER-ADDRESS=192.0.2.210:49191            |              |
    |    USERNAME="George"               |             |              |
    |    REALM="example.com"             |             |              |
    |    NONCE="obMatJos2AAABadl7W7PeDU4hKE72jda"      |              |
    |    MESSAGE-INTEGRITY=...           |             |              |

    |                                    |             |              |
    |<-- ChannelBind success response ---|             |              |
    |    Transaction-Id=0x6490D3BC175AFF3D84513212     |              |
    |    MESSAGE-INTEGRITY=...           |             |              |
```

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

skipping to change at *page 71, line 22*

The resulting Data indication is then sent to the client.

```
  TURN                                 TURN           Peer          Peer
  client                               server          A             B
    |--- ChannelBind request ----------->|             |              |
    |    Transaction-Id=0x6490D3BC175AFF3D84513212     |              |
    |    CHANNEL-NUMBER=0x4000           |             |              |
    |    XOR-PEER-ADDRESS=192.0.2.210:49191            |              |
    |    USERNAME="George"               |             |              |
    |    REALM="example.com"             |             |              |
    |    NONCE="obMatJos2gAAAadl7W7PeDU4hKE72jda"      |              |
    |    PASSWORD-ALGORITHMS=MD5 and SHA256            |              |
    |    PASSWORD-ALGORITHM=SHA256       |             |              |
    |    MESSAGE-INTEGRITY-SHA256=...    |             |              |
    |                                    |             |              |
    |<-- ChannelBind success response ---|             |              |
    |    Transaction-Id=0x6490D3BC175AFF3D84513212     |              |
    |    MESSAGE-INTEGRITY-SHA256=...    |             |              |
```

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

```
   Upon receipt of the request, the server binds the channel number to
   the peer, installs a permission for Peer B's IP address, and then
   replies with ChannelBind success response.
```

skipping to change at *page 72, line 35*

```
   indication instead.

  TURN                                 TURN           Peer          Peer
  client                               server          A             B
    |--- ChannelBind request ----------->|            |             |
    |     Transaction-Id=0xE5913A8F46091637EF7D3328   |             |
    |     CHANNEL-NUMBER=0x4000          |            |             |
    |     XOR-PEER-ADDRESS=192.0.2.210:49191          |             |
    |     USERNAME="George"              |            |             |
    |     REALM="example.com"            |            |             |
    |     NONCE="obMatJos2AAABadl7W7PeDU4hKE72jda"    |             |
    |     MESSAGE-INTEGRITY=...          |            |             |

    |                                    |            |             |
    |<-- ChannelBind success response ---|            |             |
    |     Transaction-Id=0xE5913A8F46091637EF7D3328   |             |
    |     MESSAGE-INTEGRITY=...          |            |             |

   The channel binding lasts for 10 minutes unless refreshed.  The TURN
   client refreshes the binding by sending ChannelBind request rebinding
   the channel to the same peer (Peer B's IP address).  The server
   processes the ChannelBind request, rebinds the channel to the same
   peer and resets the time-to-expiry timer back to 10 minutes.

  TURN                                 TURN           Peer          Peer
  client                               server          A             B
    |--- Refresh request --------------->|            |             |
    |     Transaction-Id=0x0864B3C27ADE9354B4312414   |             |
    |     SOFTWARE="Example client 1.03" |            |             |
    |     USERNAME="George"              |            |             |
    |     REALM="example.com"            |            |             |
    |     NONCE="obMatJos2AAABadl7W7PeDU4hKE72jda"    |             |
    |     MESSAGE-INTEGRITY=...          |            |             |

    |                                    |            |             |
    |<-- Refresh error response ---------|            |             |
    |     Transaction-Id=0x0864B3C27ADE9354B4312414   |             |
    |     SOFTWARE="Example server, version 1.17"     |             |
    |     ERROR-CODE=438 (Stale Nonce)   |            |             |
    |     REALM="example.com"            |            |             |
    |     NONCE="obMatJos2AAABnpSw1Xw239bBwGYhjN"     |             |
    |     PASSWORD-ALGORITHMS=MD5 and SHA256          |             |
    |                                    |            |             |
    |--- Refresh request --------------->|            |             |
    |     Transaction-Id=0x427BD3E625A85FC731DC4191   |             |
    |     SOFTWARE="Example client 1.03" |            |             |
    |     USERNAME="George"              |            |             |
    |     REALM="example.com"            |            |             |
    |     NONCE="obMatJos2AAABnpSw1Xw239bBwGYhjNj"    |             |
    |     PASSWORD-ALGORITHMS=MD5 and SHA256          |             |
    |     PASSWORD-ALGORITHM=SHA256      |            |             |
```

```
   Upon receipt of the request, the server binds the channel number to
   the peer, installs a permission for Peer B's IP address, and then
   replies with ChannelBind success response.
```

skipping to change at *page 72, line 35*

```
   indication instead.

  TURN                                 TURN           Peer          Peer
  client                               server          A             B
    |--- ChannelBind request ----------->|            |             |
    |     Transaction-Id=0xE5913A8F46091637EF7D3328   |             |
    |     CHANNEL-NUMBER=0x4000          |            |             |
    |     XOR-PEER-ADDRESS=192.0.2.210:49191          |             |
    |     USERNAME="George"              |            |             |
    |     REALM="example.com"            |            |             |
    |     NONCE="obMatJos2gAAAadl7W7PeDU4hKE72jda"    |             |
    |     PASSWORD-ALGORITHMS=MD5 and SHA256          |             |
    |     PASSWORD-ALGORITHM=SHA256      |            |             |
    |     MESSAGE-INTEGRITY-SHA256=...   |            |             |
    |                                    |            |             |
    |<-- ChannelBind success response ---|            |             |
    |     Transaction-Id=0xE5913A8F46091637EF7D3328   |             |
    |     MESSAGE-INTEGRITY-SHA256=...   |            |             |

   The channel binding lasts for 10 minutes unless refreshed.  The TURN
   client refreshes the binding by sending ChannelBind request rebinding
   the channel to the same peer (Peer B's IP address).  The server
   processes the ChannelBind request, rebinds the channel to the same
   peer and resets the time-to-expiry timer back to 10 minutes.

  TURN                                 TURN           Peer          Peer
  client                               server          A             B
    |--- Refresh request --------------->|            |             |
    |     Transaction-Id=0x0864B3C27ADE9354B4312414   |             |
    |     SOFTWARE="Example client 1.03" |            |             |
    |     USERNAME="George"              |            |             |
    |     REALM="example.com"            |            |             |
    |     NONCE="oobMatJos2gAAAadl7W7PeDU4hKE72jda"   |             |
    |     PASSWORD-ALGORITHMS=MD5 and SHA256          |             |
    |     PASSWORD-ALGORITHM=SHA256      |            |             |
    |     MESSAGE-INTEGRITY-SHA256=...   |            |             |
    |                                    |            |             |
    |<-- Refresh error response ---------|            |             |
    |     Transaction-Id=0x0864B3C27ADE9354B4312414   |             |
    |     SOFTWARE="Example server, version 1.17"     |             |
    |     ERROR-CODE=438 (Stale Nonce)   |            |             |
    |     REALM="example.com"            |            |             |
    |     NONCE="obMatJos2gAAAadl7W7PeDU4hKE72jda"    |             |
    |     PASSWORD-ALGORITHMS=MD5 and SHA256          |             |
    |                                    |            |             |
    |--- Refresh request --------------->|            |             |
    |     Transaction-Id=0x427BD3E625A85FC731DC4191   |             |
    |     SOFTWARE="Example client 1.03" |            |             |
    |     USERNAME="George"              |            |             |
    |     REALM="example.com"            |            |             |
    |     NONCE="obMatJos2gAAAadl7W7PeDU4hKE72jda"    |             |
    |     PASSWORD-ALGORITHMS=MD5 and SHA256          |             |
    |     PASSWORD-ALGORITHM=SHA256      |            |             |
  |     MESSAGE-INTEGRITY=...                 |                 |                 |
  |                                           |                 |                 |
  |<-- Refresh success response -------|                 |                 |
  |     Transaction-Id=0x427BD3E625A85FC731DC4191  |                 |
  |     SOFTWARE="Example server, version 1.17"    |                 |
  |     LIFETIME=600 (10 minutes)             |                 |                 |

   Sometime before the 20 minute lifetime is up, the client refreshes
   the allocation.  This is done using a Refresh request.  As before,
   the client includes the latest username, realm, and nonce values in
   the request.  The client also includes the SOFTWARE attribute,
```

```
  |     MESSAGE-INTEGRITY-SHA256=...          |                 |                 |
  |                                           |                 |                 |
  |<-- Refresh success response -------|                 |                 |
  |     Transaction-Id=0x427BD3E625A85FC731DC4191  |                 |
  |     SOFTWARE="Example server, version 1.17"    |                 |
  |     LIFETIME=600 (10 minutes)             |                 |                 |

   Sometime before the 20 minute lifetime is up, the client refreshes
   the allocation.  This is done using a Refresh request.  As before,
   the client includes the latest username, realm, and nonce values in
   the request.  The client also includes the SOFTWARE attribute,
```

skipping to change at *page 75, line 30* | **skipping to change at *page 75, line 32***

```
   they came from the peer or client, respectively.  To do that, the
   attacker can send the client a faked Data Indication or ChannelData
   message, or send the TURN server a faked Send Indication or
   ChannelData message.

   Since indications and ChannelData messages are not authenticated,
   this attack is not prevented by TURN.  However, this attack is
   generally present in IP-based communications and is not substantially
   worsened by TURN.  Consider a normal, non-TURN IP session between
   hosts A and B.  An attacker can send packets to B as if they came
   from A by sending packets towards A with a spoofed IP address of B.
   This attack requires the attacker to know the IP addresses of A and
   B.  With TURN, an attacker wishing to send packets towards a client
   using a Data indication needs to know its IP address (and port), the
   IP address and port of the TURN server, and the IP address and port
   of the peer (for inclusion in the XOR-PEER-ADDRESS attribute).  To
   send a fake ChannelData message to a client, an attacker needs to
   know the IP address and port of the client, the IP address and port
   of the TURN server, and the channel number.  This particular
   combination is mildly more guessable than in the non-TURN case.
```

```
   they came from the peer or client, respectively.  To do that, the
   attacker can send the client a faked Data Indication or ChannelData
   message, or send the TURN server a faked Send Indication or
   ChannelData message.

   Since indications and ChannelData messages are not authenticated,
   this attack is not prevented by TURN.  However, this attack is
   generally present in IP-based communications and is not substantially
   worsened by TURN.  Consider a normal, non-TURN IP session between
   hosts A and B.  An attacker can send packets to B as if they came
   from A by sending packets towards B with a spoofed IP address of A.
   This attack requires the attacker to know the IP addresses of A and
   B.  With TURN, an attacker wishing to send packets towards a client
   using a Data indication needs to know its IP address (and port), the
   IP address and port of the TURN server, and the IP address and port
   of the peer (for inclusion in the XOR-PEER-ADDRESS attribute).  To
   send a fake ChannelData message to a client, an attacker needs to
   know the IP address and port of the client, the IP address and port
   of the TURN server, and the channel number.  This particular
   combination is mildly more guessable than in the non-TURN case.
```

skipping to change at *page 76, line 36* | **skipping to change at *page 76, line 38***

```
   This attack is prevented through the long-term credential mechanism,
   which provides message integrity for responses in addition to
   verifying that they came from the server.  Furthermore, an attacker
   cannot replay old server responses as the transaction id in the STUN
   header prevents this.  Replay attacks are further thwarted through
   frequent changes to the nonce value.

21.1.6.  Eavesdropping Traffic
```

```
   This attack is prevented through the long-term credential mechanism,
   which provides message integrity for responses in addition to
   verifying that they came from the server.  Furthermore, an attacker
   cannot replay old server responses as the transaction id in the STUN
   header prevents this.  Replay attacks are further thwarted through
   frequent changes to the nonce value.

21.1.6.  Eavesdropping Traffic

   If the TURN client and server use the STUN Extension for Third-Party
   Authorization [RFC7635] (for example it is used in WebRTC), the
   username does not reveal the real user's identity, the USERNAME
   attribute carries an ephemeral and unique key identifier.  If the
   TURN client and server use the STUN long-term credential mechanism
   and the username reveals the real user's identity, the client MUST
   either use the USERHASH attribute instead of the USERNAME attribute
   to anonynmize the username or use (D)TLS transport between the client
   and the server.

   If the TURN client and server use the STUN long-term credential
   mechanism and realm information is privacy sensitive, TURN can be run
```

	over (D)TLS. As a reminder, STUN Extension for Third-Party Authorization does not use realm.
	The SOFTWARE attribute can reveal the specific software version of the TURN client and server to eavesdropper and it might possibly allow attacks against vulnerable software that is known to contain security vulnerabilities. If the software version is known to contain security vulnerabilities, TURN SHOULD be run over (D)TLS to prevent leaking the SOFTWARE attribute in clear text. If zero-day vulnerabilities are detected in the software version, the endpoint policy can be modified to mandate the use of (D)TLS till the patch is in place to fix the flaw.
TURN concerns itself primarily with authentication and message integrity. Confidentiality is only a secondary concern, as TURN control messages do not include information that is particularly sensitive. The primary protocol content of the messages is the IP address of the peer. If it is important to prevent an eavesdropper on a TURN connection from learning this, TURN can be run over (D)TLS.	TURN concerns itself primarily with authentication and message integrity. Confidentiality is only a secondary concern, as TURN control messages do not include information that is particularly sensitive, with the exception of USERNAME, REALM and SOFTWARE. The primary protocol content of the messages is the IP address of the peer. If it is important to prevent an eavesdropper on a TURN connection from learning this, TURN can be run over (D)TLS.
Confidentiality for the application data relayed by TURN is best provided by the application protocol itself, since running TURN over (D)TLS does not protect application data between the server and the peer. If confidentiality of application data is important, then the application should encrypt or otherwise protect its data. For example, for real-time media, confidentiality can be provided by using SRTP.	Confidentiality for the application data relayed by TURN is best provided by the application protocol itself, since running TURN over (D)TLS does not protect application data between the server and the peer. If confidentiality of application data is important, then the application should encrypt or otherwise protect its data. For example, for real-time media, confidentiality can be provided by using SRTP.
21.1.7. TURN Loop Attack	21.1.7. TURN Loop Attack
skipping to change at *page 82, line 17*	**skipping to change at *page 82, line 37***
TRANSPORT, DONT-FRAGMENT, Reserved (was TIMER-VAL) and RESERVATION-TOKEN listed in Section 18.]	TRANSPORT, DONT-FRAGMENT, Reserved (was TIMER-VAL) and RESERVATION-TOKEN listed in Section 18.]
[The ADDITIONAL-ADDRESS-FAMILY, ADDRESS-ERROR-CODE and ICMP attributes requires that IANA allocate a value in the "STUN attributes Registry" from the comprehension-optional range (0x8000-0xFFFF), to be replaced for TBD-CA throughout this document]	[The ADDITIONAL-ADDRESS-FAMILY, ADDRESS-ERROR-CODE and ICMP attributes requires that IANA allocate a value in the "STUN attributes Registry" from the comprehension-optional range (0x8000-0xFFFF), to be replaced for TBD-CA throughout this document]
The codepoints for the STUN error codes defined in this specification are listed in Section 19. [IANA is requested to update the reference from [RFC5766] to RFC-to-be for the STUN error codes listed in Section 19.]	The codepoints for the STUN error codes defined in this specification are listed in Section 19. [IANA is requested to update the reference from [RFC5766] and [RFC6156] to RFC-to-be for the STUN error codes listed in Section 19.]
IANA has allocated the SRV service name of "turn" for TURN over UDP or TCP, and the service name of "turns" for TURN over (D)TLS.	IANA has allocated the SRV service name of "turn" for TURN over UDP or TCP, and the service name of "turns" for TURN over (D)TLS.
IANA has created a registry for TURN channel numbers, initially populated as follows:	IANA has created a registry for TURN channel numbers, initially populated as follows:
o 0x0000 through 0x3FFF: Reserved and not available for use, since they conflict with the STUN header.	o 0x0000 through 0x3FFF: Reserved and not available for use, since they conflict with the STUN header.
skipping to change at *page 85, line 12*	**skipping to change at *page 85, line 32***
specification, [RFC5766]. The authors would like to thank Rohan Mahy co-author of original TURN specification and everyone who had	specification, [RFC5766]. The authors would like to thank Rohan Mahy co-author of original TURN specification and everyone who had

contributed to that document. The authors would also like to acknowledge that this document inherits material from [RFC6156].	contributed to that document. The authors would also like to acknowledge that this document inherits material from [RFC6156].
Thanks to Justin Uberti, Pal Martinsen, Oleg Moskalenko, Aijun Wang and Simon Perreault for their help on the ADDITIONAL-ADDRESS-FAMILY mechanism. Authors would like to thank Gonzalo Salgueiro, Simon Perreault, Jonathan Lennox, Brandon Williams, Karl Stahl, Noriyuki Torii, Nils Ohlmeier, Dan Wing, Vijay Gurbani, Joseph Touch, Justin Uberti and Oleg Moskalenko for comments and review. The authors would like to thank Marc for his contributions to the text.	Thanks to Justin Uberti, Pal Martinsen, Oleg Moskalenko, Aijun Wang and Simon Perreault for their help on the ADDITIONAL-ADDRESS-FAMILY mechanism. Authors would like to thank Gonzalo Salgueiro, Simon Perreault, Jonathan Lennox, Brandon Williams, Karl Stahl, Noriyuki Torii, Nils Ohlmeier, Dan Wing, Vijay Gurbani, Joseph Touch, Justin Uberti, Christopher Wood, Roman Danyliw, Eric Vyncke, Adam Roach, Mirja Kuehlewind, Benjamin Kaduk, Noriyuki Torii and Oleg Moskalenko for comments and review. The authors would like to thank Marc for his contributions to the text.
Special thanks to Magnus Westerlund for the detailed AD review.	Special thanks to Magnus Westerlund for the detailed AD review.
27. References	27. References
27.1. Normative References	27.1. Normative References
[I-D.ietf-tram-stunbis] Petit-Huguenin, M., Salgueiro, G., Rosenberg, J., Wing, D., Mahy, R., and P. Matthews, "Session Traversal	[I-D.ietf-tram-stunbis] Petit-Huguenin, M., Salgueiro, G., Rosenberg, J., Wing, D., Mahy, R., and P. Matthews, "Session Traversal
skipping to change at *page 86, line 29*	**skipping to change at *page 86, line 49***
[RFC6347] Rescorla, E. and N. Modadugu, "Datagram Transport Layer Security Version 1.2", RFC 6347, DOI 10.17487/RFC6347, January 2012, <https://www.rfc-editor.org/info/rfc6347>.	[RFC6347] Rescorla, E. and N. Modadugu, "Datagram Transport Layer Security Version 1.2", RFC 6347, DOI 10.17487/RFC6347, January 2012, <https://www.rfc-editor.org/info/rfc6347>.
[RFC6437] Amante, S., Carpenter, B., Jiang, S., and J. Rajahalme, "IPv6 Flow Label Specification", RFC 6437, DOI 10.17487/RFC6437, November 2011, <https://www.rfc-editor.org/info/rfc6437>.	[RFC6437] Amante, S., Carpenter, B., Jiang, S., and J. Rajahalme, "IPv6 Flow Label Specification", RFC 6437, DOI 10.17487/RFC6437, November 2011, <https://www.rfc-editor.org/info/rfc6437>.
[RFC6724] Thaler, D., Ed., Draves, R., Matsumoto, A., and T. Chown, "Default Address Selection for Internet Protocol Version 6 (IPv6)", RFC 6724, DOI 10.17487/RFC6724, September 2012, <https://www.rfc-editor.org/info/rfc6724>.	
[RFC7065] Petit-Huguenin, M., Nandakumar, S., Salgueiro, G., and P. Jones, "Traversal Using Relays around NAT (TURN) Uniform Resource Identifiers", RFC 7065, DOI 10.17487/RFC7065, November 2013, <https://www.rfc-editor.org/info/rfc7065>.	[RFC7065] Petit-Huguenin, M., Nandakumar, S., Salgueiro, G., and P. Jones, "Traversal Using Relays around NAT (TURN) Uniform Resource Identifiers", RFC 7065, DOI 10.17487/RFC7065, November 2013, <https://www.rfc-editor.org/info/rfc7065>.
[RFC7350] Petit-Huguenin, M. and G. Salgueiro, "Datagram Transport Layer Security (DTLS) as Transport for Session Traversal Utilities for NAT (STUN)", RFC 7350, DOI 10.17487/RFC7350, August 2014, <https://www.rfc-editor.org/info/rfc7350>.	[RFC7350] Petit-Huguenin, M. and G. Salgueiro, "Datagram Transport Layer Security (DTLS) as Transport for Session Traversal Utilities for NAT (STUN)", RFC 7350, DOI 10.17487/RFC7350, August 2014, <https://www.rfc-editor.org/info/rfc7350>.
skipping to change at *page 87, line 10*	**skipping to change at *page 87, line 21***
"Recommendations for Secure Use of Transport Layer Security (TLS) and Datagram Transport Layer Security (DTLS)", BCP 195, RFC 7525, DOI 10.17487/RFC7525, May 2015, <https://www.rfc-editor.org/info/rfc7525>.	"Recommendations for Secure Use of Transport Layer Security (TLS) and Datagram Transport Layer Security (DTLS)", BCP 195, RFC 7525, DOI 10.17487/RFC7525, May 2015, <https://www.rfc-editor.org/info/rfc7525>.
[RFC7915] Bao, C., Li, X., Baker, F., Anderson, T., and F. Gont, "IP/ICMP Translation Algorithm", RFC 7915,	[RFC7915] Bao, C., Li, X., Baker, F., Anderson, T., and F. Gont, "IP/ICMP Translation Algorithm", RFC 7915,

DOI 10.17487/RFC7915, June 2016, <https://www.rfc-editor.org/info/rfc7915>.	DOI 10.17487/RFC7915, June 2016, <https://www.rfc-editor.org/info/rfc7915>.
	[RFC7982] Martinsen, P., Reddy, T., Wing, D., and V. Singh, "Measurement of Round-Trip Time and Fractional Loss Using Session Traversal Utilities for NAT (STUN)", RFC 7982, DOI 10.17487/RFC7982, September 2016, <https://www.rfc-editor.org/info/rfc7982>. [RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/info/rfc8174>.
[RFC8200] Deering, S. and R. Hinden, "Internet Protocol, Version 6 (IPv6) Specification", STD 86, RFC 8200, DOI 10.17487/RFC8200, July 2017, <https://www.rfc-editor.org/info/rfc8200>. [RFC8305] Schinazi, D. and T. Pauly, "Happy Eyeballs Version 2: Better Connectivity Using Concurrency", RFC 8305, DOI 10.17487/RFC8305, December 2017, <https://www.rfc-editor.org/info/rfc8305>. [RFC8446] Rescorla, E., "The Transport Layer Security (TLS) Protocol Version 1.3", RFC 8446, DOI 10.17487/RFC8446, August 2018, <https://www.rfc-editor.org/info/rfc8446>. 27.2. Informative References [Frag-Harmful]	[RFC8200] Deering, S. and R. Hinden, "Internet Protocol, Version 6 (IPv6) Specification", STD 86, RFC 8200, DOI 10.17487/RFC8200, July 2017, <https://www.rfc-editor.org/info/rfc8200>. [RFC8305] Schinazi, D. and T. Pauly, "Happy Eyeballs Version 2: Better Connectivity Using Concurrency", RFC 8305, DOI 10.17487/RFC8305, December 2017, <https://www.rfc-editor.org/info/rfc8305>. [RFC8446] Rescorla, E., "The Transport Layer Security (TLS) Protocol Version 1.3", RFC 8446, DOI 10.17487/RFC8446, August 2018, <https://www.rfc-editor.org/info/rfc8446>. 27.2. Informative References [Frag-Harmful]
"Fragmentation Considered Harmful", <Proc. SIGCOMM '87, vol. 17, No. 5, October 1987>.	Kent, C. and J. Mogul, ""Fragmentation Considered Harmful", In Proc. SIGCOMM '87 Workshop on Frontiers in Computer Communications Technology, DOI 10.1145/55483.55524", August 1987, <http://www.hpl.hp.com/techreports/Compaq-DEC/WRL-87-3.pdf>. [I-D.ietf-intarea-frag-fragile] Bonica, R., Baker, F., Huston, G., Hinden, R., Troan, O., and F. Gont, "IP Fragmentation Considered Fragile", draft-ietf-intarea-frag-fragile-15 (work in progress), July 2019.
[I-D.ietf-mmusic-ice-sip-sdp] Petit-Huguenin, M., Nandakumar, S., and A. Keranen, "Session Description Protocol (SDP) Offer/Answer procedures for Interactive Connectivity Establishment (ICE)", draft-ietf-mmusic-ice-sip-sdp-36 (work in progress), June 2019.	[I-D.ietf-mmusic-ice-sip-sdp] Petit-Huguenin, M., Nandakumar, S., and A. Keranen, "Session Description Protocol (SDP) Offer/Answer procedures for Interactive Connectivity Establishment (ICE)", draft-ietf-mmusic-ice-sip-sdp-36 (work in progress), June 2019.
	[I-D.ietf-mptcp-rfc6824bis] Ford, A., Raiciu, C., Handley, M., Bonaventure, O., and C. Paasch, "TCP Extensions for Multipath Operation with Multiple Addresses", draft-ietf-mptcp-rfc6824bis-18 (work in progress), June 2019. [I-D.ietf-rtcweb-security] Rescorla, E., "Security Considerations for WebRTC", draft-ietf-rtcweb-security-12 (work in progress), July 2019.

draft-ietf-tram-turnbis-27.txt	draft-ietf-tram-turnbis-28.txt
[I-D.ietf-tram-stun-pmtud] Petit-Huguenin, M. and G. Salgueiro, "Path MTU Discovery Using Session Traversal Utilities for NAT (STUN)", draft-ietf-tram-stun-pmtud-10 (work in progress), September 2018.	[I-D.ietf-tram-stun-pmtud] Petit-Huguenin, M. and G. Salgueiro, "Path MTU Discovery Using Session Traversal Utilities for NAT (STUN)", draft-ietf-tram-stun-pmtud-10 (work in progress), September 2018.
[I-D.ietf-tsvwg-udp-options] Touch, J., "Transport Options for UDP", draft-ietf-tsvwg-udp-options-07 (work in progress), March 2019.	[I-D.ietf-tsvwg-udp-options] Touch, J., "Transport Options for UDP", draft-ietf-tsvwg-udp-options-07 (work in progress), March 2019.
skipping to change at *page 90, line 10*	**skipping to change at *page 91, line 5***
[RFC6062] Perreault, S., Ed. and J. Rosenberg, "Traversal Using Relays around NAT (TURN) Extensions for TCP Allocations", RFC 6062, DOI 10.17487/RFC6062, November 2010, <https://www.rfc-editor.org/info/rfc6062>.	[RFC6062] Perreault, S., Ed. and J. Rosenberg, "Traversal Using Relays around NAT (TURN) Extensions for TCP Allocations", RFC 6062, DOI 10.17487/RFC6062, November 2010, <https://www.rfc-editor.org/info/rfc6062>.
[RFC6156] Camarillo, G., Novo, O., and S. Perreault, Ed., "Traversal Using Relays around NAT (TURN) Extension for IPv6", RFC 6156, DOI 10.17487/RFC6156, April 2011, <https://www.rfc-editor.org/info/rfc6156>.	[RFC6156] Camarillo, G., Novo, O., and S. Perreault, Ed., "Traversal Using Relays around NAT (TURN) Extension for IPv6", RFC 6156, DOI 10.17487/RFC6156, April 2011, <https://www.rfc-editor.org/info/rfc6156>.
[RFC6824] Ford, A., Raiciu, C., Handley, M., and O. Bonaventure, "TCP Extensions for Multipath Operation with Multiple Addresses", RFC 6824, DOI 10.17487/RFC6824, January 2013, <https://www.rfc-editor.org/info/rfc6824>.	[RFC6263] Marjou, X. and A. Sollaud, "Application Mechanism for Keeping Alive the NAT Mappings Associated with RTP / RTP Control Protocol (RTCP) Flows", RFC 6263, DOI 10.17487/RFC6263, June 2011, <https://www.rfc-editor.org/info/rfc6263>.
[RFC7413] Cheng, Y., Chu, J., Radhakrishnan, S., and A. Jain, "TCP Fast Open", RFC 7413, DOI 10.17487/RFC7413, December 2014, <https://www.rfc-editor.org/info/rfc7413>.	[RFC7413] Cheng, Y., Chu, J., Radhakrishnan, S., and A. Jain, "TCP Fast Open", RFC 7413, DOI 10.17487/RFC7413, December 2014, <https://www.rfc-editor.org/info/rfc7413>.
[RFC7478] Holmberg, C., Hakansson, S., and G. Eriksson, "Web Real-Time Communication Use Cases and Requirements", RFC 7478, DOI 10.17487/RFC7478, March 2015, <https://www.rfc-editor.org/info/rfc7478>.	[RFC7478] Holmberg, C., Hakansson, S., and G. Eriksson, "Web Real-Time Communication Use Cases and Requirements", RFC 7478, DOI 10.17487/RFC7478, March 2015, <https://www.rfc-editor.org/info/rfc7478>.
End of changes. 92 change blocks.	
239 lines changed or deleted	*344 lines changed or added*

This html diff was produced by rfcdiff 1.47. The latest version is available from http://tools.ietf.org/tools/rfcdiff/