[MS-TURN]:
Traversal Using Relay NAT (TURN) Extensions

Intellectual Property Rights Notice for Open Specifications Documentation

- **Technical Documentation.** Microsoft publishes Open Specifications documentation for protocols, file formats, languages, standards as well as overviews of the interaction among each of these technologies.
- **Copyrights.** This documentation is covered by Microsoft copyrights. Regardless of any other terms that are contained in the terms of use for the Microsoft website that hosts this documentation, you may make copies of it in order to develop implementations of the technologies described in the Open Specifications and may distribute portions of it in your implementations using these technologies or your documentation as necessary to properly document the implementation. You may also distribute in your implementation, with or without modification, any schema, IDL's, or code samples that are included in the documentation. This permission also applies to any documents that are referenced in the Open Specifications.
- **No Trade Secrets.** Microsoft does not claim any trade secret rights in this documentation.
- **Patents.** Microsoft has patents that may cover your implementations of the technologies described in the Open Specifications. Neither this notice nor Microsoft's delivery of the documentation grants any licenses under those or any other Microsoft patents. However, a given Open Specification may be covered by Microsoft's Open Specification Promise (available here: http://www.microsoft.com/interop/osp) or the Community Promise (available here: http://www.microsoft.com/interop/cp/default.mspx). If you would prefer a written license, or if the technologies described in the Open Specifications are not covered by the Open Specifications Promise or Community Promise, as applicable, patent licenses are available by contacting iplg@microsoft.com.
- **Trademarks.** The names of companies and products contained in this documentation may be covered by trademarks or similar intellectual property rights. This notice does not grant any licenses under those rights.
- **Fictitious Names.** The example companies, organizations, products, domain names, e-mail addresses, logos, people, places, and events depicted in this documentation are fictitious. No association with any real company, organization, product, domain name, email address, logo, person, place, or event is intended or should be inferred.

Reservation of Rights. All other rights are reserved, and this notice does not grant any rights other than specifically described above, whether by implication, estoppel, or otherwise.

Tools. The Open Specifications do not require the use of Microsoft programming tools or programming environments in order for you to develop an implementation. If you have access to Microsoft programming tools and environments you are free to take advantage of them. Certain Open Specifications are intended for use in conjunction with publicly available standard specifications and network programming art, and assumes that the reader either is familiar with the aforementioned material or has immediate access to it.

Revision Summary

Date	Revision History	Revision Class	Comments
04/04/2008	0.1		Initial version
04/25/2008	0.2		Updated the technical content.
06/27/2008	1.0		Updated and revised the technical content.
08/15/2008	1.01		Revised and edited the technical content.
12/12/2008	2.0		Updated and revised the technical content.
02/13/2009	2.01		Revised and edited the technical content.
03/13/2009	2.02		Revised and edited the technical content.
07/13/2009	2.03	Major	Revised and edited the technical content
08/28/2009	2.04	Editorial	Revised and edited the technical content
11/06/2009	2.05	Editorial	Revised and edited the technical content
02/19/2010	2.06	Editorial	Revised and edited the technical content
03/31/2010	2.07	Major	Updated and revised the technical content
04/30/2010	2.08	Editorial	Revised and edited the technical content
06/07/2010	2.09	Editorial	Revised and edited the technical content
06/29/2010	2.10	Editorial	Changed language and formatting in the technical content.
07/23/2010	2.10	No change	No changes to the meaning, language, or formatting of the technical content.
09/27/2010	3.0	Major	Significantly changed the technical content.
11/15/2010	3.0	No change	No changes to the meaning, language, or formatting of the technical content.
12/17/2010	3.0	No change	No changes to the meaning, language, or formatting of the technical content.
03/18/2011	3.0	No change	No changes to the meaning, language, or formatting of the technical content.
06/10/2011	3.0	No change	No changes to the meaning, language, or formatting of the technical content.

Table of Contents

1 Introduction

This document specifies proprietary extensions to the Traversal Using Relay NAT (TURN) protocol. TURN is an Internet Engineering Task Force (IETF) draft proposal designed to provide a mechanism to enable a user behind a network address translation (NAT) to acquire a transport address from a public server and to use the allocated transport address to receive data from a selected peer.

This protocol is used as part of the Interactive Connectivity Establishment (ICE) Extensions protocol, as described in [MS-ICE] and [MS-ICE2].

1.1 Glossary

The following terms are defined in [MS-GLOS]:

Coordinated Universal Time (UTC)
Internet Protocol version 4 (IPv4)
network address translation (NAT)
nonce
Secure Sockets Layer (SSL)
Transmission Control Protocol (TCP)
type-length-value (TLV)
User Datagram Protocol (UDP)

The following terms are defined in [MS-OFCGLOS]:

200 OK
digest
Interactive Connectivity Establishment (ICE)
INVITE
long-term credentials
response message
Session Description Protocol (SDP)
Session Initiation Protocol (SIP)
transport address
Transport Layer Security (TLS)
Traversal Using Relay NAT (TURN)
TURN client
TURN server

The following terms are specific to this document:

allocated transport address: A transport address that is allocated by a Traversal Using Relay NAT (TURN) server in response to an Allocate request from a TURN client. The TURN server obtains the transport address from a network interface that is connected to the Internet. The transport address has the same transport protocol over which the Allocate request was received; a request that is received over TCP returns a TCP allocated transport address. Also referred to as an allocated address.

error response message: A Traversal Using Relay NAT (TURN) message that is sent from a protocol server to a protocol client in response to a request message. It is sent when an error occurs during processing of a request message.

public address: An IPv4 address that is on the Internet.

reflexive transport address: A transport address that is given to a protocol client and identifies the public address of that client as seen by a protocol server. The address is communicated to

the protocol client through the XOR MAPPED ADDRESS attribute (1) in an allocate response message.

request message: A Traversal Using Relay NAT (TURN) message that is sent from a protocol client to a protocol server.

MAY, SHOULD, MUST, SHOULD NOT, MUST NOT: These terms (in all caps) are used as described in [RFC2119]. All statements of optional behavior use either MAY, SHOULD, or SHOULD NOT.

1.2 References

1.2.1 Normative References

We conduct frequent surveys of the normative references to assure their continued availability. If you have any issue with finding a normative reference, please contact dochelp@microsoft.com. We will assist you in finding the relevant information. Please check the archive site, http://msdn2.microsoft.com/en-us/library/E4BD6494-06AD-4aed-9823-445E921C9624, as an additional source.

[IETFDRAFT-STUN-02] Rosenberg, J., Huitema, C., and Mahy, R., "Simple Traversal of UDP Through Network Address Translators (NAT) (STUN)", draft-ietf-behave-rfc3489bis-02, July 2005, http://tools.ietf.org/html/draft-ietf-behave-rfc3489bis-02

[IETFDRAFT-TURN-08] Rosenberg, J., Mahy, R., and Huitema, C., "Traversal Using Relay NAT (TURN)", draft-rosenberg-midcom-turn-08, September 2005, http://tools.ietf.org/html/draft-rosenberg-midcom-turn-08

[RFC1321] Rivest, R., "The MD5 Message-Digest Algorithm", RFC 1321, April 1992, http://www.ietf.org/rfc/rfc1321.txt

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997, http://www.rfc-editor.org/rfc/rfc2119.txt

1.2.2 Informative References

[MS-AVEDGEA] Microsoft Corporation, "Audio Video Edge Authentication Protocol Specification"

[MS-GLOS] Microsoft Corporation, "Windows Protocols Master Glossary".

[MS-ICE] Microsoft Corporation, "Interactive Connectivity Establishment (ICE) Extensions"

[MS-ICE2] Microsoft Corporation, "Interactive Connectivity Establishment (ICE) Extensions 2.0"

[MS-OFCGLOS] Microsoft Corporation, "Microsoft Office Master Glossary".

[RFC2246] Dierks, T., and Allen, C., "The TLS Protocol Version 1.0", RFC 2246, January 1999, http://www.ietf.org/rfc/rfc2246.txt

[RFC3261] Rosenberg, J., Schulzrinne, H., Camarillo, G., Johnston, A., Peterson, J., Sparks, R., Handley, M., and Schooler, E., "SIP: Session Initiation Protocol", RFC 3261, June 2002, http://www.ietf.org/rfc/rfc3261.txt

[RFC4566] Handley, M., Jacobson, V., and Perkins, C., "SDP: Session Description Protocol", RFC 4566, July 2006, http://www.ietf.org/rfc/rfc4566.txt

[RFC768] Postel, J., "User Datagram Protocol", STD 6, RFC 768, August 1980, http://www.ietf.org/rfc/rfc768.txt

[RFC793] Postel, J., "Transmission Control Protocol", STD 7, RFC 793, September 1981, http://www.ietf.org/rfc/rfc0793.txt

[TURN-01] Rosenberg, J., Mahy, R., and Huitema, C., "Obtaining Relay Addresses from Simple Traversal of UDP Through NAT (STUN)", draft-ietf-behave-turn-01, February 2006, http://tools.ietf.org/wg/behave/draft-ietf-behave-turn/draft-ietf-behave-turn-01.txt

[TURN-05] Rosenberg, J., Mahy, R., Matthews, P., and Wing, D., "Traversal Using Relays around NAT (TURN): Relay Extensions to Session Traversal Utilities for NAT (STUN)", draft-ietf-behave-turn-05, November 2007, http://tools.ietf.org/wg/behave/draft-ietf-behave-turn/draft-ietf-behave-turn-05.txt

1.3 Protocol Overview (Synopsis)

The **Traversal Using Relay NAT (TURN)** protocol, as described in [IETFDRAFT-TURN-08], enables a **TURN client** located on a private network behind one or more **network address translation (NAT)** to allocate a **transport address** from a **TURN server** which is sitting on the internet. This **allocated transport address** can be used for receiving data from a peer. The peer itself could be on a private network behind a NAT or it could have a **public address**.

A typical deployment, supported by the TURN protocol and this extension, where a protocol client is behind a NAT and is communicating with a peer on the internet is shown in the following figure.

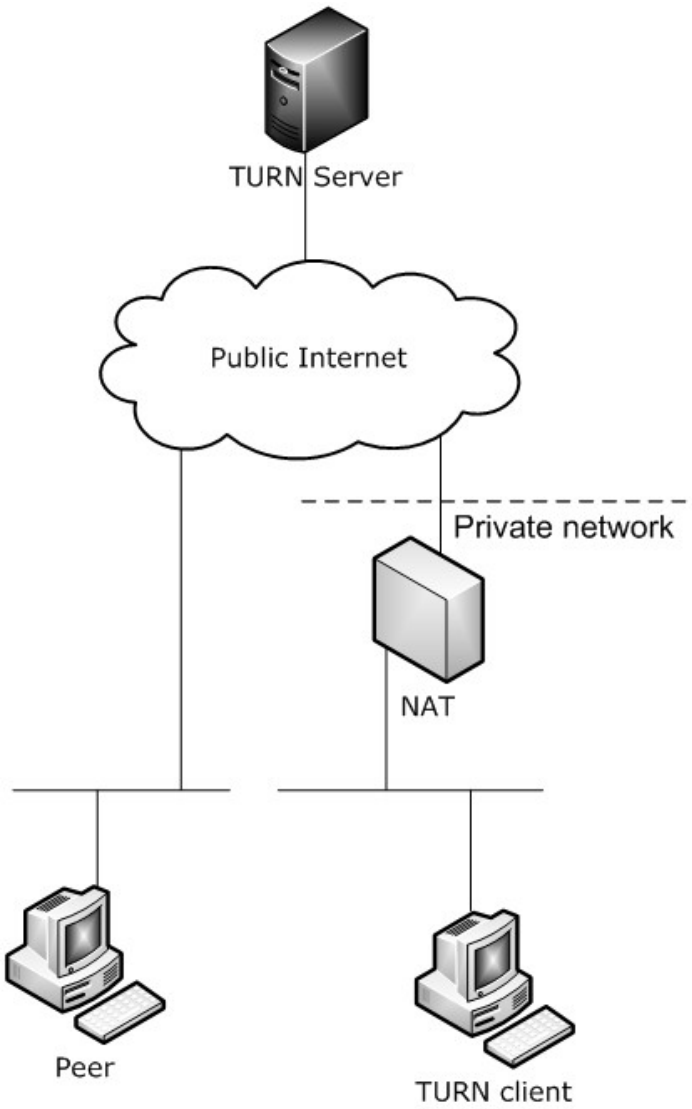


Figure 1: A TURN client communicating with a public peer

When a protocol client needs a public address to send data to or receive data from a peer, it sends an Allocate **request message** to the server. This request is authenticated by the server through a **digest** challenge mechanism. Once the server has authenticated the Allocate request, it returns an allocated address to the protocol client in an Allocate **response message**.

At this point the allocated address has been reserved by the protocol client. It cannot be used to receive data from a peer until the protocol client attempts to send data to the peer by encapsulating the data in a Send request message. The Send request message serves two functions:

- The server relays the data contained in the message to the peer identified by the **Destination** attribute.
- Permissions are set on the allocated address such that data arriving on the allocated address from the peer is relayed to the protocol client in a Data Indication message.

If the protocol client needs to communicate with more than one peer, it can send an additional Send request message to each peer.

Once the permissions have been set for a peer, any data received on the allocated address from that peer is relayed back to the protocol client encapsulated in a Data Indication message. This message includes the **Remote Address** attribute that identifies the peer that originated the data.

If the protocol client decides to communicate with a preferred peer, it can send a Set Active Destination request message to the server. The server acknowledges the protocol client's request by responding with a Set Active Destination response message. This allows the protocol client and server to stop using Send request and Data Indication messages to encapsulate data flowing end-to-end for this peer, thus making the data communication channel more efficient. The results are that all data that the server receives from the protocol client that is not a TURN control message is relayed directly to the active peer. All data that the server receives on the allocated address from the active peer is relayed directly to the protocol client. If the server receives data from a peer other than the active peer but for which it has permissions, as set by the protocol client through an earlier Send request message, the server relays the data encapsulating it in a Data Indication message.

The basic flow of TURN messages between a protocol client and a server is shown in the following figure.

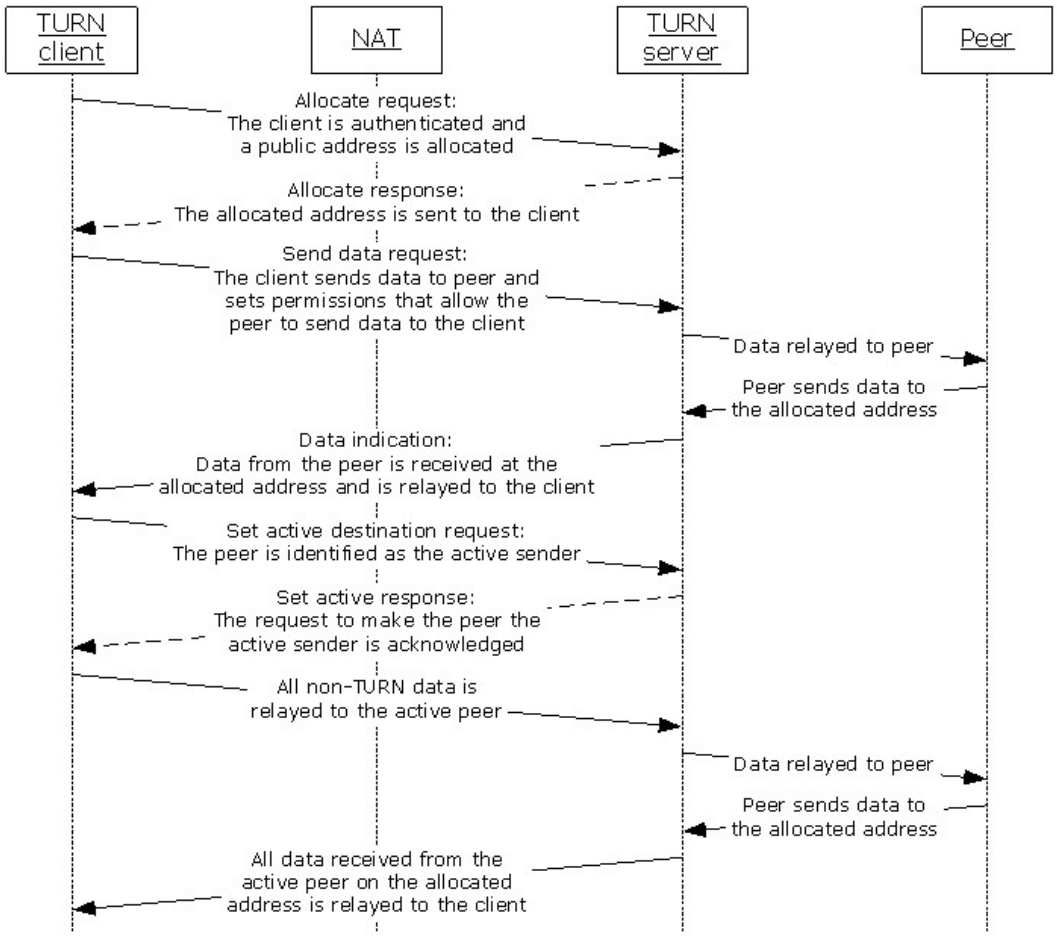


Figure 2: Basic flow of TURN messages

This document specifies proprietary extensions to the TURN protocol. These extensions include:

Authentication: This protocol does not utilize the Shared Secret request/Shared Secret response messages, as described in [IETFDRAFT-TURN-08] section 7.1, for authentication of the protocol client. Instead, this protocol uses **long-term credentials** and has extended the Allocate request and Allocate **error response message** processing, as described in [IETFDRAFT-TURN-08] section 7.2, to incorporate a **digest** challenge mechanism. This is specified in section 3.1.2. This extension is from the TURN draft version described in [TURN-05].

TCP Framing Header: A header has been added to this protocol for stream-based transports, so that TURN control messages are uniquely identifiable from end-to-end data. This is specified in section 2.1.2. This extension is from the TURN draft version described in [TURN-01].

Pseudo-TLS Session Establishment: This protocol includes a pseudo **Transport Layer Security (TLS)** Client Hello/Server Hello message exchange at the beginning of a **Transmission Control Protocol (TCP)** session to allow session establishment over TCP port 443, where a proxy or firewall might inspect the initial traffic for TLS packets. This is specified in section 2.1.1.

Client Versioning: An attribute has been added to this protocol to request messages to allow a protocol client to identify the protocol version it is using. This is specified in section 2.2.2.16.

Request Message Sequencing: An attribute has been added to the protocol that contains a sequence number to request messages. This attribute helps prevent replay attacks and is specified in section 2.2.2.17.

No Support for Send Response or Send Error Response Messages: Support has been dropped for any response to the Send request message, as described in [IETFDRAFT-TURN-08] section 7.3, to streamline the Send request phase of a TURN session. This extension is from the TURN draft version described in [TURN-01].

XOR Mapped Address: This protocol uses the **XOR-MAPPED-ADDRESS** attribute from [IETFDRAFT-STUN-02] section 10.2.12 to inform the protocol client of the protocol client's **reflexive transport address**. This keeps intrusive NATs from rewriting the binary encoded IP address and port. This is specified in section 2.2.2.15.

Alternate Server Attribute: This protocol extends the use of the **Alternate Server** attribute and includes it in the Allocate error response message, error response code of 401 Unauthorized, to convey the IP address of the server to the protocol client. In some deployments, for example, a pool of TURN servers behind a load balancer that presents a virtual IP address, the protocol client needs to know the direct address of the server with which it established a TURN session. This is specified in section 2.2.2.7.

1.4 Relationship to Other Protocols

This protocol does not introduce any new protocol relationships beyond those described in [IETFDRAFT-TURN-08]. The TURN protocol, as described in [IETFDRAFT-TURN-08], is used to provide network connectivity and relies on either **User Datagram Protocol (UDP)**, as described in [RFC768], or TCP, as described in [RFC793], as a transport.

1.5 Prerequisites/Preconditions

It is assumed that the TURN client and server have an **Internet Protocol version 4 (IPv4)** address with either UDP or TCP connectivity and that the protocol client knows the IPv4 address and port of the TURN server and a peer which it wants to communicate with. The server is assumed to be ready to receive datagrams, in the case of UDP, or incoming connections, in the case of TCP, on the configured port.

It is also assumed that the TURN client has long-term credentials that it can use to authenticate with the TURN server. These credentials are acquired by communicating with a protocol server that has implemented the protocol described in [MS-AVEDGEA].

1.6 Applicability Statement

This protocol does not change the applicability of the TURN protocol as it is described in [IETFDRAFT-TURN-08] section 4.

1.7 Versioning and Capability Negotiation

This protocol has the following versioning constraints:

Supported Transports: This protocol can be implemented over either TCP IPv4 or UDP IPv4, as discussed in section 2.2.1.

Protocol Versions: This protocol specifies a mechanism by which the protocol client can explicitly indicate to the server what version of the protocol it is using. This is done by including the **MS-VERSION** attribute in an authenticated Allocate request message. The **MS-VERSION** attribute is specified in section 2.2.2.15.

Security and Authentication Methods: This protocol supports authentication through long-term credentials supplied in the Allocate request message. This is specified in section 3.1.12.

Capability Negotiation: This protocol does not have any capability negotiation constraints.

1.8 Vendor-Extensible Fields

None.

1.9 Standards Assignments

This protocol uses the standard UDP and TCP ports from [IETFDRAFT-STUN-02]. It has no additional standard assignments.

Parameter	Value	Reference
UDP Port	3478	[IETFDRAFT-STUN-02]
TCP Port	3478	[IETFDRAFT-STUN-02]

2 Messages

2.1 Transport

This protocol can use either UDP or TCP as a transport protocol. All message formats are specified as a UDP datagram and do not require any additional framing when sent over UDP. Transport over TCP requires additional framing, as specified in section 2.1.1 and section 2.1.2.

2.1.1 Pseudo-TLS over TCP

When TCP is used as a transport, the server is deployed to listen on port 443, the **Secure Sockets Layer (SSL)**/TLS port. If a protocol client is attempting to communicate with a server deployed in this fashion, it sends a pseudo-TLS message to the server to begin the session. The pseudo-TLS messages are useful if a firewall or Web proxy, doing packet inspection for TLS messages, is sitting between the protocol client and server. The server MUST support pseudo-TLS.

The protocol client begins the exchange by sending the pseudo-TLS "ClientHello" message. If the protocol client sends this message, it MUST be the first message and the protocol client MUST NOT send any additional messages until the server has responded with a pseudo-TLS "ServerHello" message followed by a pseudo-TLS "ServerHelloDone" message. If the server receives a pseudo-TLS "ClientHello" message, it MUST respond with a "ServerHello" followed by a "ServerHelloDone" message. The "ServerHello" and "ServerHelloDone" messages MUST be sent in the same TLS record. These messages appear next in this protocol.

The "ClientHello", "ServerHello", and "ServerHelloDone" messages passed in the exchange are known as Handshake messages within the TLS Record Protocol. The TLS Record Protocol is described in [RFC2246] section 6 while the Handshake messages are described in [RFC2246] section 7.3.

Pseudo-TLS record containing Client Hello message

<table>
<tr><th>0</th><th>1</th><th>2</th><th>3</th><th>4</th><th>5</th><th>6</th><th>7</th><th>8</th><th>9</th><th>10</th><th>1</th><th>2</th><th>3</th><th>4</th><th>5</th><th>6</th><th>7</th><th>8</th><th>9</th><th>20</th><th>1</th><th>2</th><th>3</th><th>4</th><th>5</th><th>6</th><th>7</th><th>8</th><th>9</th><th>30</th><th>1</th></tr>
<tr><td colspan="8">Content Type</td><td colspan="8">Record Version (Major)</td><td colspan="8">Record Version (Minor)</td><td colspan="8">Record Length …</td></tr>
<tr><td colspan="8">…</td><td colspan="8">Handshake Type</td><td colspan="16">Handshake Length …</td></tr>
<tr><td colspan="8">…</td><td colspan="8">Handshake Version (Major)</td><td colspan="8">Handshake Version (Minor)</td><td colspan="8">Time Stamp …</td></tr>
<tr><td colspan="24">…</td><td colspan="8">Random Value (28 bytes)</td></tr>
<tr><td colspan="32">Random Value …</td></tr>
<tr><td colspan="24">…</td><td colspan="8">Session ID Length</td></tr>
<tr><td colspan="16">Cipher Suites Length</td><td colspan="16">Cipher Suites</td></tr>
</table>

Compression Methods Length	Compression Methods

Content Type (1 byte): The Record Layer protocol type. This field MUST be set to 0x16 for the Handshake.

Record Version Major (1 byte): The Major version of TLS for this record. This field MUST be set to 0x03 (TLS 1.0).

Record Version Minor (1 byte): The Minor version of TLS for this record. This field MUST be set to 0x01 (TLS 1.0).

Record Length (2 bytes): The length of the TLS record. This field MUST be set to 0x00 0x2D.

Handshake Type (1 byte): The Handshake message type. This field MUST be set to 0x01 for a Client Hello message.

Handshake Length (3 bytes): The length of the Handshake message. This field MUST be set to 0x00 0x00 0x29.

Handshake Version Major (1 byte): The Major version of TLS for the message. This field MUST be set to 0x03 (TLS 1.0).

Handshake Version Minor (1 byte): The Minor version of TLS for the message. This field MUST be set to 0x01 (TLS 1.0).

Time Stamp (4 bytes): The current time and date in seconds since midnight starting January 1, 1970, **Coordinated Universal Time (UTC)**, ignoring leap seconds. The protocol client SHOULD fill this field with the correct time. The server SHOULD ignore this field.

Random Value (28 bytes): 28 bytes of randomly generated data.

Session ID Length (1 byte): The length of the session ID vector. This field MUST be set to 0x00.

Cipher Suites Length (2 bytes): The length of the cipher suite vector. This field MUST be set to 0x00 0x02.

Cipher Suites (2 bytes): The cipher suite the protocol client is requesting. This field MUST be set to 0x00 0x18.

Compression Methods Length (1 byte): The length of the compression method vector. This field MUST be set to 0x01.

Compression Methods (1 byte): The compression methods that the protocol client is requesting. This field MUST be set to 0x00.

Pseudo-TLS record containing Server Hello and Server Hello Done messages

<table>
<tr><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>10</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>20</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>30</td><td>1</td></tr>
<tr><td colspan="8">Content Type</td><td colspan="8">Record Version (Major)</td><td colspan="8">Record Version (Minor)</td><td colspan="8">Record Length …</td></tr>
<tr><td colspan="8">…</td><td colspan="8">Handshake Type</td><td colspan="16">Handshake Length …</td></tr>
</table>

…	Handshake Version (Major)	Handshake Version (Minor)	Time Stamp …
…			Random Value … (28 bytes)
Random Value …			
…			Session ID Length
Session ID (32 bytes)			
Cipher Suite		Compression Method	Handshake Type
Handshake Length			

Content Type (1 byte): The Record Layer protocol type. This field MUST be set to 0x16 for the Handshake.

Record Version Major (1 byte): The Major version of TLS for this record. This field MUST be set to 0x03 (TLS 1.0).

Record Version Minor (1 byte): The Minor version of TLS for this record. This field MUST be set to 0x01 (TLS 1.0).

Record Length (2 bytes): The length of the TLS record. This field MUST be set to 0x00 0x4E.

Handshake Type (1 byte): The Handshake message type. This field MUST be set to 0x02 for a Server Hello message.

Handshake Length (3 bytes): The length of the Handshake message. This field MUST be set to 0x00 0x00 0x46.

Handshake Version Major (1 byte): The Major version of TLS for the message. This field MUST be set to 0x03 (TLS 1.0).

Handshake Version Minor (1 byte): The Minor version of TLS for the message. This field MUST be set to 0x01 (TLS 1.0).

Time Stamp (4 bytes): The current time and date in seconds since midnight starting January 1, 1970, UTC, ignoring leap seconds. The server SHOULD fill this field with the correct time. The protocol client SHOULD ignore this field.

Random Value (28 bytes): 28 bytes of randomly generated data.

Session ID Length (1 byte): The length of the session ID vector. This field MUST be set to 0x20.

Session ID (32 bytes): 32 bytes used to identify the TLS session. The server does not track the TLS session id so the protocol client SHOULD ignore this field.

Cipher Suite (2 bytes): The cipher suite the server has selected. This field MUST be set to 0x00 0x16.

Compression Methods (1 byte): The compression method that the server has selected. This field MUST be set to 0x00.

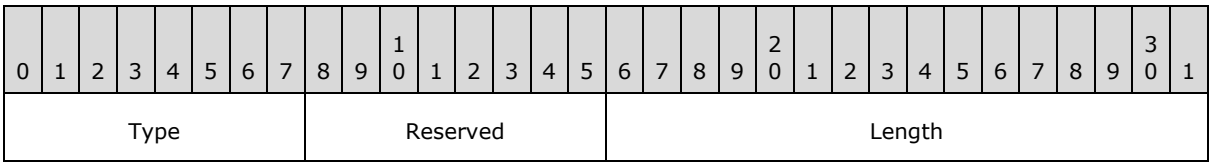
Handshake Type (1 byte): The Handshake message type. This field MUST be set to 0x0E for a Server Hello Done message.

Handshake Length (3 bytes): The length of the Handshake message. This field MUST be set to 0x00 0x00 0x00.

2.1.2 TCP

When TCP is used as a transport for this protocol, it requires some additional framing so that the TURN control messages can be identified within the TCP data stream. This additional framing consists of a header followed by the TURN datagram. This framing header MUST be used for all TURN messages and data sent to the TURN server. The framing header MUST NOT be used for the pseudo-TLS session establishment messages.

TCP Framing Header



Type (1 byte): The data contained in this frame is a TURN control message or end-to-end data. This MUST be set to 2 to identify a TURN control message or it MUST be set to 3 to identify end-to-end data.

Reserved (1 byte): Not used and MUST be set to zero.

Length (2 bytes): The number of bytes of the frame following immediately after the **Length** field itself.

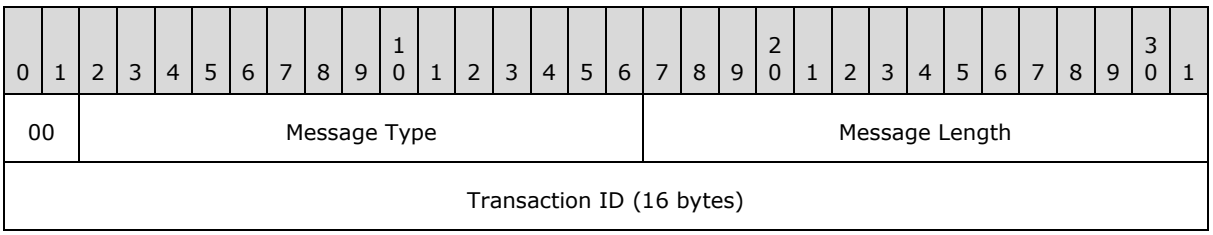
2.1.3 UDP

When UDP is used as a transport for this protocol no transport specific framing is required.

2.2 Message Syntax

2.2.1 Message Header

All TURN messages consist of a 20 byte TURN header followed by 1 or more TURN attributes. The TURN attributes are **type-length-value (TLV)** encoded. The TURN message header is the same as the message header specified in [IETFDRAFT-STUN-02] section 10.1. All TURN messages begin with any necessary transport specific framing, as specified in section 2.1, followed by this header.



Message Type (16 bits): The type of TURN message. The most significant two bits of this field MUST be set to zero so that TURN packets can be differentiated from other protocols. The TURN message types are specified in [IETFDRAFT-TURN-08] section 9.1. The TURN message types supported in this extension:

- 0x0003: Allocate request
- 0x0103: Allocate response
- 0x0113: Allocate error response
- 0x0004: Send request
- 0x0115: Data Indication
- 0x0006: Set Active Destination request
- 0x0106: Set Active Destination response
- 0x0116: Set Active Destination error response

The following TURN message types are not supported by this extension and the server MUST NOT send them:

- 0x0104: Send request response
- 0x0114: Send request error response

In addition, this extension does not support the shared secret authentication mechanism. The following shared secret messages MUST NOT be used by either the protocol client or server:

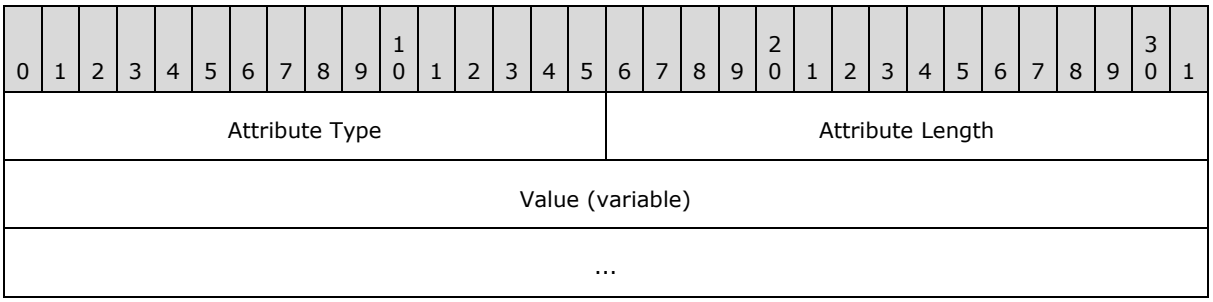
- 0x0002: Shared Secret request
- 0x0102: Shared Secret response
- 0x0112: Share Secret error response

Message Length (16 bits): The length, in bytes, of the message. This length does not include the 20 byte header.

Transaction ID (16 bytes): A 128 bit identifier used to uniquely identify the TURN transaction. A Transaction ID, created by the protocol client and used in a request message, is echoed from the server back to the protocol client in the subsequent response or error response message. The protocol client MUST choose a new transaction ID for each new transaction. A new transaction ID SHOULD be uniformly and randomly distributed between 0 and 2^128 -1. If the protocol client is retransmitting a request message, it MUST use the same Transaction ID as it used in the original request message.

2.2.2 Message Attribute

After the TURN header, all TURN messages consist of 1 or more attributes. All attributes MUST be type-length-value (TLV) encoded and have the same format as specified in [IETFDRAFT-STUN-02] section 10.2. The **Magic Cookie** attribute MUST be the first attribute in all TURN messages.



Attribute Type (2 bytes): The TURN attributes are specified in [IETFDRAFT-STUN-02] section 10.2 and [IETFDRAFT-TURN-08] section 9.2.

The following mandatory attribute types are supported in this extension. Any other attributes from the mandatory attribute space MUST generate an error response with an error response code of Unknown Attribute.

0x0001: **Mapped Address**

0x0006: **Username**

0x0008: **Message Integrity**

0x0009: **Error Code**

0x000A: **Unknown Attributes**

0x000D: **Lifetime**

0x000E: **Alternate Server**

0x000F: **Magic Cookie**

0x0010: **Bandwidth**

0x0011: **Destination Address**

0x0012: **Remote Address**

0x0013: **Data**

0x0014: **Nonce**

0x0015: **Realm**

The following optional attributes are also supported in this extension. Any other attributes from the optional attribute space SHOULD be ignored.

0x8008: **MS-Version**

0x8020: **XOR Mapped Address**

0x8050: **MS-Sequence Number**

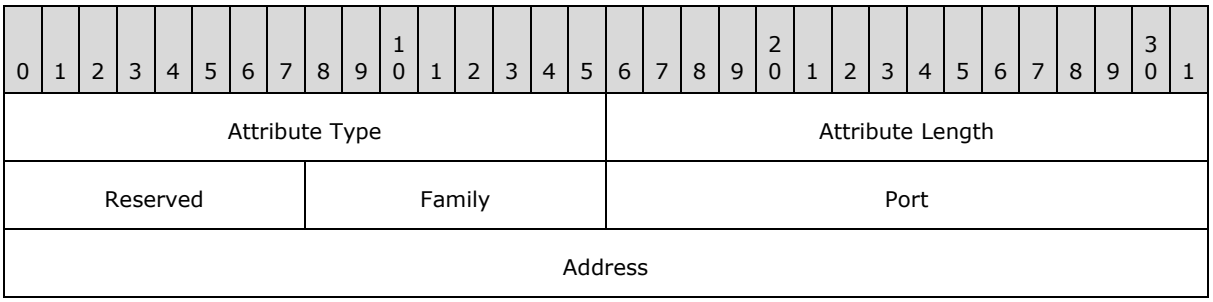
0x8055: **MS-Service Quality**<1>

Attribute Length (2 bytes): The length of bytes of the **Value** data following the **Attribute Length** field itself.

Value (variable): Variable-length field that contains information dependent on the attribute type.

2.2.2.1 Mapped Address

The **Mapped Address** attribute is specified in [IETFDRAFT-STUN-02] section 10.2.1. This protocol supports IPv4 addresses. This attribute is used to identify the public transport address allocated by the server on behalf of the protocol client.



Attribute Type (2 bytes): The TURN attributes are specified in [IETFDRAFT-STUN-02] section 10.2 and [IETFDRAFT-TURN-08] section 9.2. Set to 0x0001.

Attribute Length (2 bytes): Set to 0x0008 (8).

Reserved (1 byte): The first 8 bits are used for alignment purposes and are ignored.

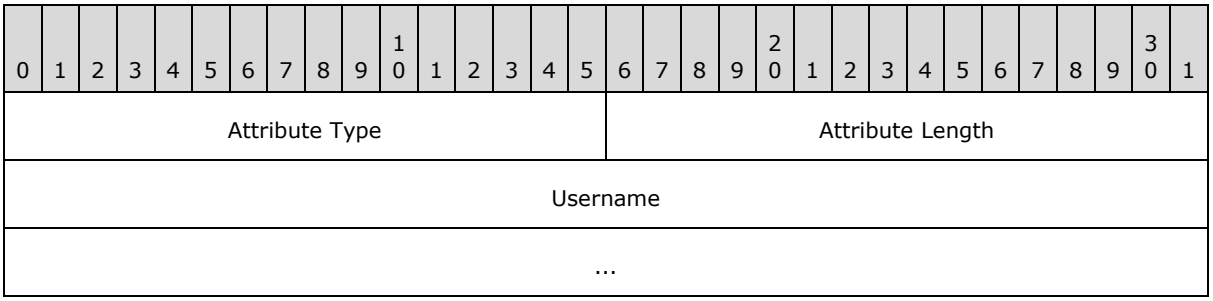
Family (1 byte): The address family of the attribute.

Port (2 bytes): A network byte ordered representation of the mapped port.

Address (4 bytes): A network byte ordered 32-bit IPv4 address.

2.2.2.2 Username

The **Username** attribute is specified in [IETFDRAFT-STUN-02] section 10.2.6. This attribute is used to identify the user name part of the protocol client's long-term credentials with the server. The server MUST know how to validate this user name and it MUST be able to retrieve the password associated with this user name. If the server does not know the user name, it MUST fail the authenticated request with an appropriate error response message that includes an **Error Code** attribute with an error response value of 436 Unknown User.



Attribute Type (2 bytes): The TURN attributes are specified in [IETFDRAFT-STUN-02] section 10.2 and [IETFDRAFT-TURN-08] section 9.2. Set to 0x0006.

Attribute Length (2 bytes): The length of **Username** in bytes.

Username (variable): The value of the opaque, variable length **Username**.

2.2.2.3 Message Integrity

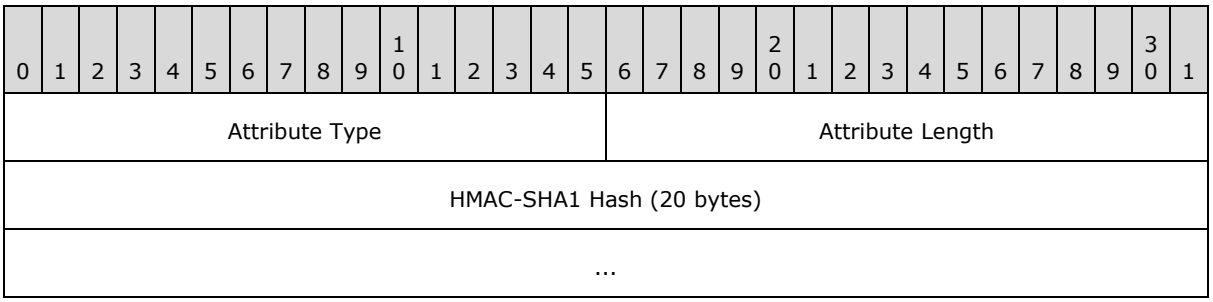
The **Message Integrity** attribute is specified in [IETFDRAFT-STUN-02] section 10.2.8. This protocol uses this attribute in all authenticated request and response messages. This attribute MUST be the last attribute in a TURN message.

The algorithm for using HMAC-SHA1 to create the hash value is specified as part of [IETFDRAFT-TURN-08] section 7.1. This algorithm is used to create the hash for outbound messages and to verify the hash of inbound messages. The algorithm summary is as follows:

- The text used as input to the HMAC is the TURN message, including the TURN message header, up to and including the attribute preceding the **Message Integrity** attribute. The text is padded with zeros to be a multiple of 64 bytes.
- As shown in the following example, the key used in the HMAC is the 128-bit **digest** resulting from using the MD5 message-digest algorithm [RFC1321] on the long term user name, the value of the **Realm** attribute and the long term password, each separated by a ":".

Key = MD5(Username ":" Realm ":" password)

Hash = HMAC-SHA1(Key, Text)



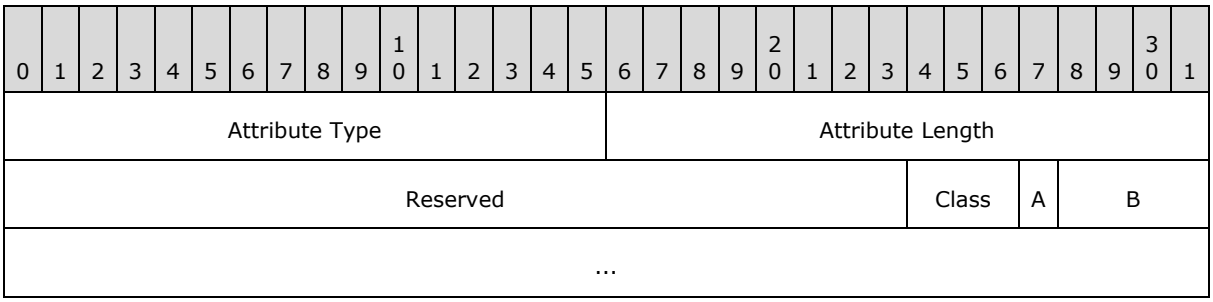
Attribute Type (2 bytes): The TURN attributes are specified in [IETFDRAFT-STUN-02] section 10.2 and [IETFDRAFT-TURN-08] section 9.2. Set to 0x0008.

Attribute Length (2 bytes): This field contains the length, in bytes, of the **HMAC-SHA1 Hash** field. Set to 0x0014 (20).

HMAC-SHA1 Hash (20 bytes): The output of the HMAC-SHA1 algorithm.

2.2.2.4 Error Code

The **Error Code** attribute is specified in [IETFDRAFT-STUN-02] section 10.2.9. For error response codes and suggested text for the associated Reason Phrase, see [IETFDRAFT-STUN-02] section 10.2.9 and [IETFDRAFT-TURN-08] section 9.2.10.



Attribute Type (2 bytes): The TURN attributes are specified in [IETFDRAFT-STUN-02] section 10.2 and [IETFDRAFT-TURN-08] section 9.2. Set to 0x0009.

Attribute Length (2 bytes): This field contains the length, in bytes, of the following fields.

Reserved (3 bytes): Set to zero.

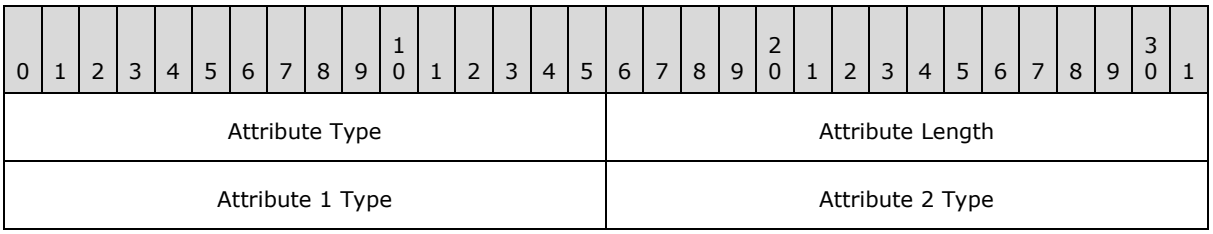
Class (3 bits): The 100s digit of the response code. The value MUST be between 1 and 6, as specified in [IETFDRAFT-STUN-02] section 10.2.9. The supported Error Class values are specified in [IETFDRAFT-STUN-02] section 10.2.9 and [IETFDRAFT-TURN-08] section 9.2.10.

A - *Number (1 bit): The response code modulo 100. The value MUST be between 0 and 99, as specified in [IETFDRAFT-STUN-02] section 10.2.9. The supported Error Numbers are specified in [IETFDRAFT-STUN-02] section 10.2.9 and [IETFDRAFT-TURN-08] section 9.2.10.

B - Reason Phrase (variable): Textual description of the error that has occurred. The phrase is encoded in UTF-8, as specified in [IETFDRAFT-STUN-02] section 10.2.9. Recommended reason phrases for various errors are specified in [IETFDRAFT-STUN-02] section 10.2.9 and [IETFDRAFT-TURN-08] section 9.2.10.

2.2.2.5 Unknown Attributes

The **Unknown Attributes** attribute is specified in [IETFDRAFT-STUN-02] section 10.2.10. This attribute is present only in an error response message that contains an error code of 420. The attribute contains a list of 16-bit values, each representing the mandatory attribute type that was not understood by the server.



Attribute Type (2 bytes): The TURN attributes are specified in [IETFDRAFT-STUN-02] section 10.2 and [IETFDRAFT-TURN-08] section 9.2. Set to 0x000A.

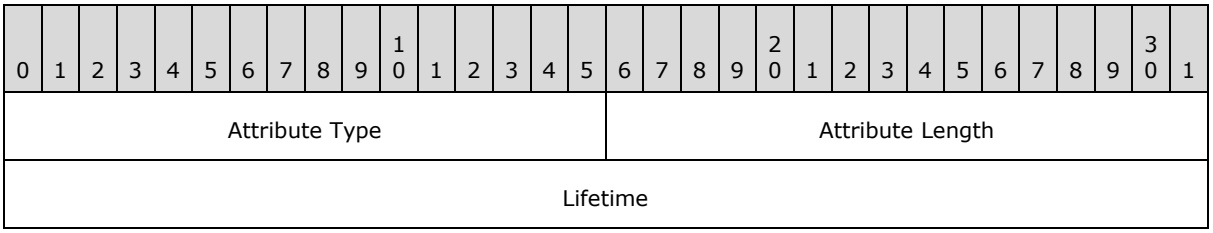
Attribute Length (2 bytes): This field contains the length, in bytes, of the following fields.

Attribute 1 Type (2 bytes): Type of first attribute.

Attribute 2 Type (2 bytes): Type of second attribute.

2.2.2.6 Lifetime

The **Lifetime** attribute is specified in [IETFDRAFT-TURN-08] section 9.2.1.



Attribute Type (2 bytes): The TURN attributes are specified in [IETFDRAFT-STUN-02] section 10.2 and [IETFDRAFT-TURN-08] section 9.2. Set to 0x000D.

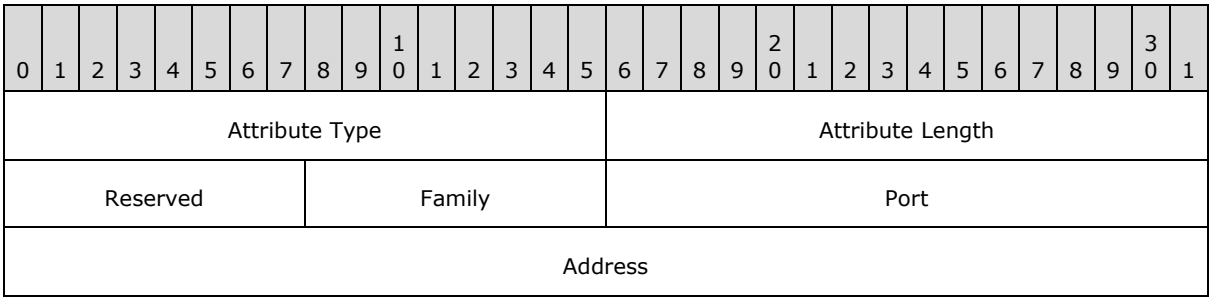
Attribute Length (2 bytes): This field contains the length, in bytes, of the **Lifetime** field. Set to 0x0004 (4).

Lifetime (4 bytes): Number of seconds for which the server maintains an allocated address in the absence of data traffic from the protocol client to the server. If the value is zero in a subsequent Allocate request message, the TURN session associated with this protocol client MUST be torn down.

2.2.2.7 Alternate Server

The **Alternate Server** attribute is specified in [IETFDRAFT-TURN-08] section 9.2.2. The alternate server is used in two error response messages:

- An error response with an error code of 401 Unauthorized. In this case the value of the **Alternate Server** attribute SHOULD be the public transport address of the TURN server from which the response originated. If the transport is UDP, the protocol client MUST use the transport address from the **Alternate Server** attribute as the destination for the next Allocate request message.
- An error response with an error code of 300 Try Alternate, which occurs when the TURN server does not have resources to satisfy an Allocate request. In this case the value of the **Alternate Server** attribute is another TURN server that had available resources for the Allocate request.



Attribute Type (2 bytes): The TURN attributes are specified in [IETFDRAFT-STUN-02] section 10.2 and [IETFDRAFT-TURN-08] section 9.2.

Attribute Length (2 bytes): This field contains the length, in bytes, of the following fields. Set to 0x0008 (8).

Reserved (1 byte): The first 8 bits are used for alignment purposes and are ignored.

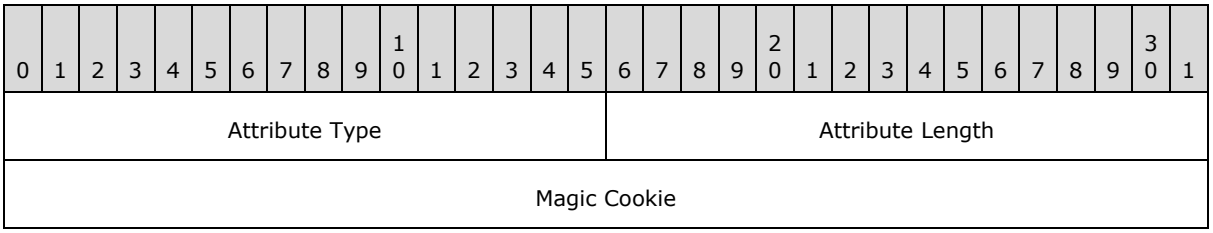
Family (1 byte): The address family of the attribute. Set to 0x01.

Port (2 bytes): A network-byte-ordered representation of the mapped port.

Address (4 bytes): A network-byte-ordered 32-bit IPv4 address.

2.2.2.8 Magic Cookie

The **Magic Cookie** attribute is specified in [IETFDRAFT-TURN-08] section 9.2.3. This attribute MUST be the first attribute following the TURN message header in all TURN messages. It is used to disambiguate TURN messages from data traffic.



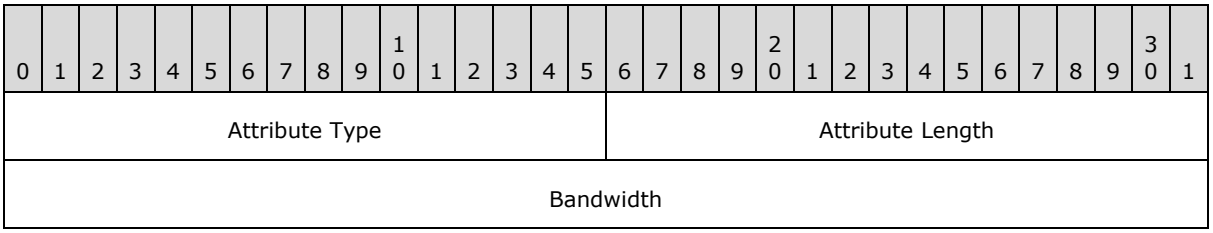
Attribute Type (2 bytes): The TURN attributes are specified in [IETFDRAFT-STUN-02] section 10.2 and [IETFDRAFT-TURN-08] section 9.2. Set to 0x000F.

Attribute Length (2 bytes): This field contains the length, in bytes, of the **Magic Cookie** field. Set to 0x0004 (4).

Magic Cookie (4 bytes): Set to 0x72c64bc6.

2.2.2.9 Bandwidth

The **Bandwidth** attribute is specified in [IETFDRAFT-TURN-08] section 9.2.4.



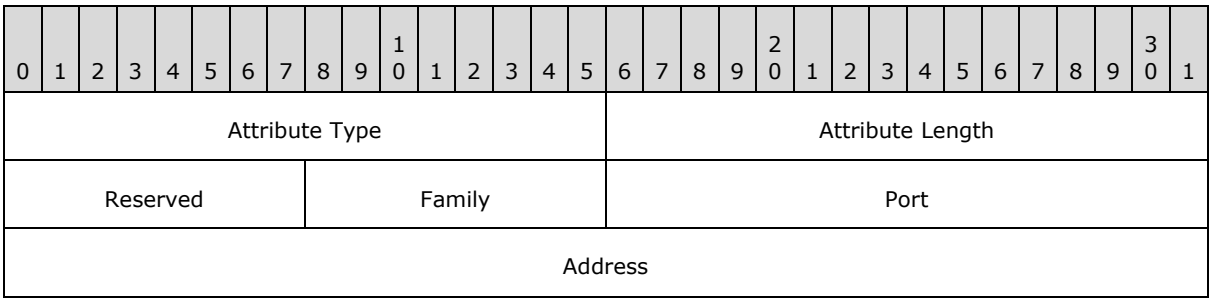
Attribute Type (2 bytes): The TURN attributes are specified in [IETFDRAFT-STUN-02] section 10.2 and [IETFDRAFT-TURN-08] section 9.2. Set to 0x0010.

Attribute Length (2 bytes): This field contains the length, in bytes, of the **Bandwidth** field. Set to 0x0004 (4).

Bandwidth (4 bytes): The **Bandwidth** value represents the peak bandwidth, in kilobits per second.

2.2.2.10 Destination Address

The **Destination Address** attribute is specified in [IETFDRAFT-TURN-08] section 9.2.5.



Attribute Type (2 bytes): The TURN attributes are specified in [IETFDRAFT-STUN-02] section 10.2 and [IETFDRAFT-TURN-08] section 9.2. Set to 0x0011.

Attribute Length (2 bytes): This field contains the length, in bytes, of the following fields. Set to 0x0008 (8).

Reserved (1 byte): The first 8 bits are used for alignment purposes and are ignored.

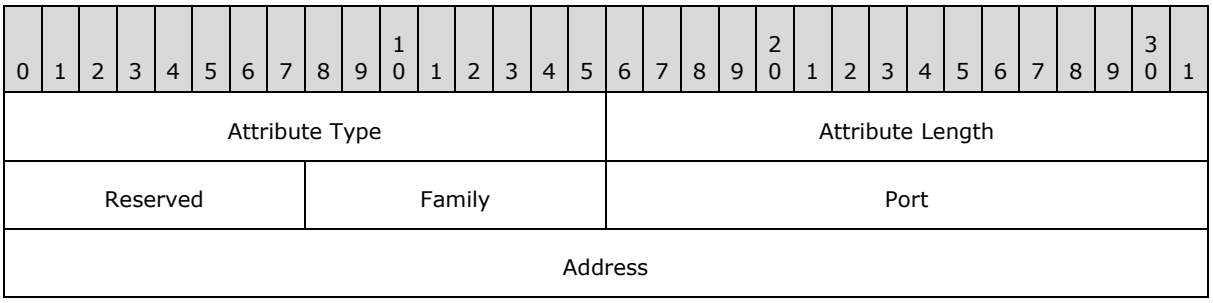
Family (1 byte): The address family of the attribute. This extension only supports IPv4, so this field MUST be set to 0x01.

Port (2 bytes): A network byte ordered representation of the mapped port.

Address (4 bytes): A network byte ordered 32-bit IPv4 address.

2.2.2.11 Remote Address

The **Remote Address** attribute is specified in [IETFDRAFT-TURN-08] section 9.2.6.



Attribute Type (2 bytes): The TURN attributes are specified in [IETFDRAFT-STUN-02] section 10.2 and [IETFDRAFT-TURN-08] section 9.2. Set to 0x0012.

Attribute Length (2 bytes): This field contains the length, in bytes, of the following field. Set to 0x0008 (8).

Reserved (1 byte): The first 8 bits are used for alignment purposes and are ignored.

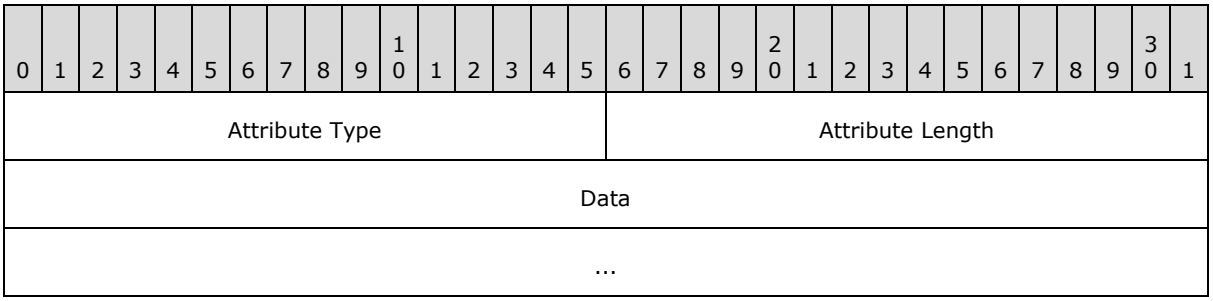
Family (1 byte): The address family of the attribute. This extension only supports IPv4, so this field MUST be set to 0x01.

Port (2 bytes): A network-byte-ordered representation of the mapped port.

Address (4 bytes): A network-byte-ordered 32-bit IPv4 address.

2.2.2.12 Data

The **Data** attribute is specified in [IETFDRAFT-TURN-08] section 9.2.7.



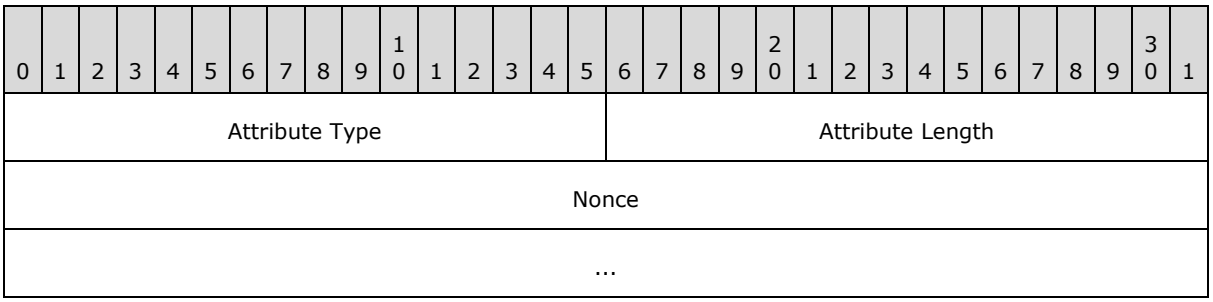
Attribute Type (2 bytes): The TURN attributes are specified in [IETFDRAFT-STUN-02] section 10.2 and [IETFDRAFT-TURN-08] section 9.2. Set to 0x0013.

Attribute Length (2 bytes): This field contains the length, in bytes, of the **Data** field.

Data (variable): Raw data that is to be relayed between a protocol client and a peer.

2.2.2.13 Nonce

The **Nonce** attribute is specified in [IETFDRAFT-TURN-08] section 9.2.8. The value of the **Nonce** attribute is used for replay protection and SHOULD be encoded by the server in such a way as to indicate duration of validity or the protocol client identity for which it is valid. This protocol uses the attribute in the Digest Challenge extension specified in section 3.1.12.



Attribute Type (2 bytes): The TURN attributes are specified in [IETFDRAFT-STUN-02] section 10.2 and [IETFDRAFT-TURN-08] section 9.2. Set to 0x0014.

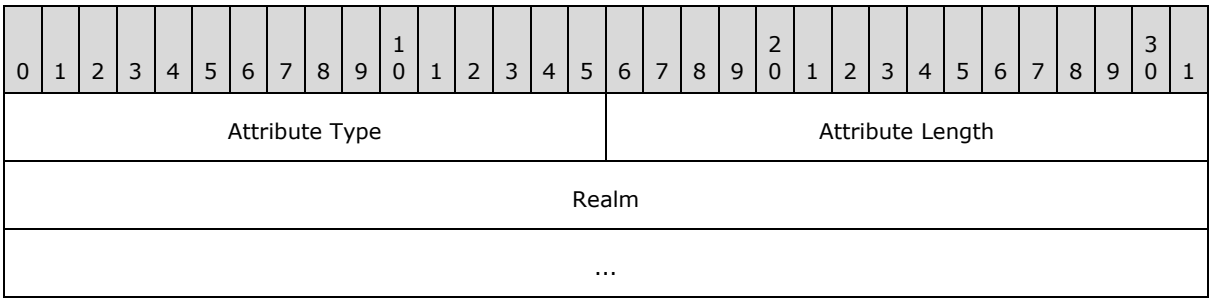
Attribute Length (2 bytes): The length of bytes of the **Nonce** field. The **Nonce** field length MUST NOT exceed 128 bytes.

Nonce (variable): Variable length of data used as a **nonce** value. This nonce value length MUST NOT exceed 128 bytes.

2.2.2.14 Realm

The **Realm** attribute is specified in [IETFDRAFT-TURN-08] section 9.2.9. The value of the **Realm** attribute SHOULD be the domain name of the provider of the TURN server. This protocol uses the attribute in the Digest Challenge extension specified in section 3.1.12. If the protocol client includes this attribute the server SHOULD use the specified Realm value in the Digest Challenge extension. If the client does not include this attribute in the request message the server will use a default Realm

value. The server MUST include this attribute in the associated response and the Realm value MUST be the value that the server used in the Digest Challenge extension.



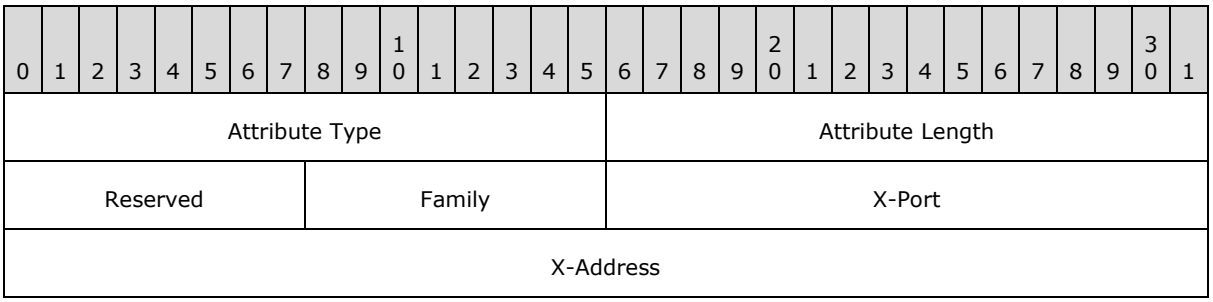
Attribute Type (2 bytes): The TURN attributes are specified in [IETFDRAFT-STUN-02] section 10.2 and [IETFDRAFT-TURN-08] section 9.2. Set to 0x0015.

Attribute Length (2 bytes): The length of bytes of the **Realm** field. The **Realm** length MUST NOT exceed 128 bytes.

Realm (variable): Variable length of data used as the realm value. **Realm** MUST NOT exceed 128 bytes.

2.2.2.15 XOR Mapped Address

The **XOR Mapped Address** attribute is specified in [IETFDRAFT-STUN-02] section 10.2.12. This protocol uses the **XOR Mapped Address** attribute to indicate to the protocol client its reflexive transport address. The protocol client can use this to help identify the type of NAT it is behind.



Attribute Type (2 bytes): The TURN attributes are specified in [IETFDRAFT-STUN-02] section 10.2 and [IETFDRAFT-TURN-08] section 9.2. Set to 0x8020.

Attribute Length (2 bytes): The length of bytes of following fields. Set to 0x0008 (8).

Reserved (1 byte): The first 8 bits are used for alignment purposes and are ignored.

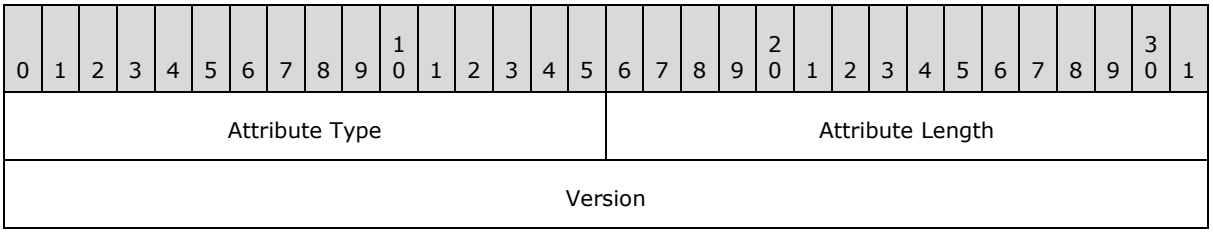
Family (1 byte): The address family of the attribute. This extension only supports IPv4, so this field MUST be set to 0x01.

X-Port (2 bytes): A network byte ordered representation of the source port that the Allocate request message was received from. This value is created from the exclusive-or of the source port with the most significant 16 bits of the Transaction ID. If the source port was 0x1122 (network byte order) and the most significant 16 bits of the Transaction ID was 0x4455 (network byte order), the resulting X-Port is 0x1122 ^ 0x4455 = 0x5577.

X-Address (4 bytes): A network byte ordered 32-bit IPv4 address of the source address that the Allocate request message was received from. This value is created from the exclusive-or of the source IP address with the most significant 32 bits of the Transaction ID. If the source IPv4 address was 0x11223344 and the most significant 32 bits of the Transaction ID was 0xaabbccdd, the resulting X-Address is 0x11223344 ^ 0xaabbccdd = 0xbb99ff99.

2.2.2.16 MS-Version Attribute

The **MS-Version** attribute is used to convey the TURN protocol version. This attribute SHOULD be included in the Allocate Request message from the protocol client. The format of this attribute is as follows.



Attribute Type (2 bytes): The TURN attributes are specified in [IETFDRAFT-STUN-02] section 10.2 and [IETFDRAFT-TURN-08] section 9.2. Set to 0x8008.

Attribute Length (2 bytes): The length of bytes of **Version** field. Set to 0x0004 (4).

Version (4 bytes): This field contains the version of the TURN protocol in use.

The following versions are currently defined:

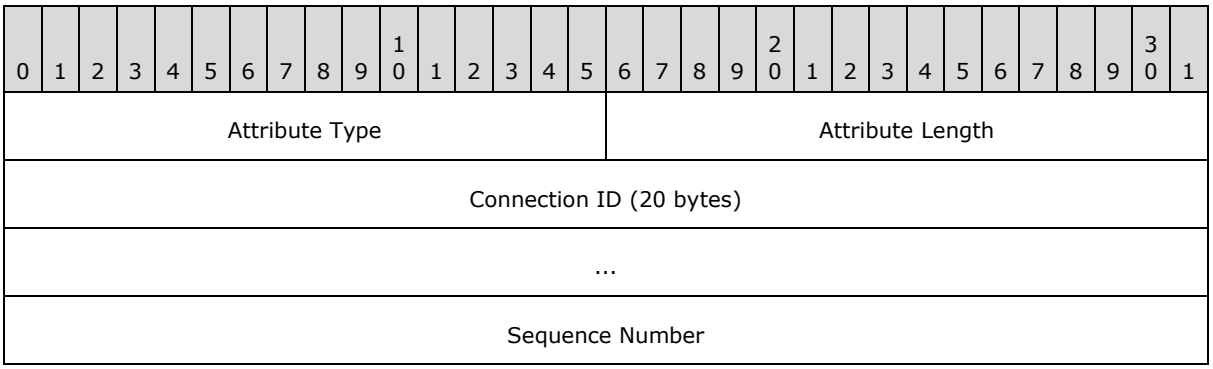
0x00000001 – Used by a protocol client implementing the **Interactive Connectivity Establishment (ICE)** protocol specified in [MS-ICE].

0x00000002 – Used by a protocol client implementing the ICE protocol specified in [MS-ICE2].<2>

2.2.2.17 MS-Sequence Number Attribute

The **MS-Sequence Number** attribute is used to provide sequence information for all authenticated request messages sent from the protocol client to the server. This can help against replay attacks. The server can include this attribute in the initial successfully authenticated Allocate response it sends to the protocol client. The Connection ID and initial Sequence Number are generated by the server. The server MUST use 20 bytes of random data for the Connection ID. The Connection ID SHOULD be unique per connection on the server. The initial Sequence Number SHOULD be zero. If the server includes this attribute in the Allocate response, the protocol client MUST include this attribute in all subsequent authenticated request messages. The protocol client MUST echo the Connection ID that it received from the server in each request message. The protocol client MUST increment the sequence number monotonically for each request message it sends.

If the server supports this attribute, it SHOULD use an algorithm that is tolerant of out of order packet reception and dropped packets.



Attribute Type (2 bytes): The TURN attributes are specified in [IETFDRAFT-STUN-02] section 10.2 and [IETFDRAFT-TURN-08] section 9.2. Set to 0x8050.

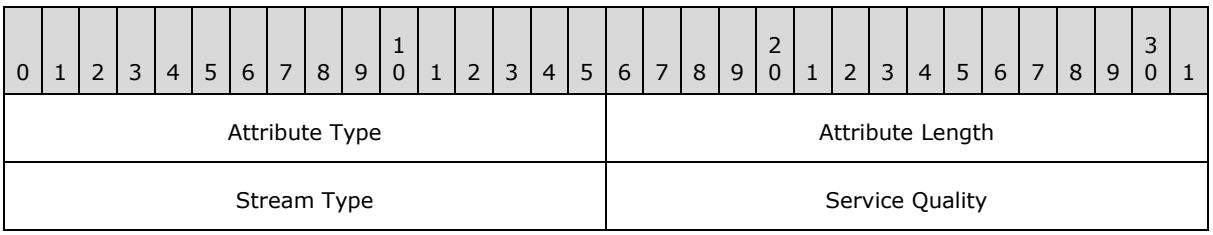
Attribute Length (2 bytes): The length of bytes of following fields. Set to 0x0018 (24).

Connection ID (20 bytes): A 20-byte connection identifier generated by the server.

Sequence Number (4 bytes): A 32-bit sequence number that is monotonically incremented by the protocol client for each request message it sends to the server.

2.2.2.18 MS-Service Quality Attribute

The **MS-Service Quality** attribute is used to convey information about the data stream that the protocol client is intending to transfer over an allocated port. The protocol client SHOULD<3> include this attribute as part of an Allocate Request message. A server SHOULD use the information in this attribute to make decisions about resource allocation, bandwidth prioritization, and data delivery methods. If the attribute is not present in the Allocate Request message, the server SHOULD assume that the data stream is audio with best effort delivery. The format of this attribute is as follows.



Attribute Type (2 bytes): The TURN attributes are specified in [IETFDRAFT-STUN-02] section 10.2 and [IETFDRAFT-TURN-08] section 9.2. Set to 0x8055.

Attribute Length (2 bytes): The length of bytes of following fields. Set to 0x0004 (4).

Stream Type (2 bytes): The type of data to be transferred over the allocated port.

The following stream types are supported in this extension. All other stream types are reserved for future use.

0x0001: Audio

0x0002: Video

0x0003: Supplemental Video

0x0004: Data

Service Quality (2 bytes): The service quality level required by the protocol client for the stream.

The following service quality levels are supported in this extension. All other service quality levels are reserved for future use.

0x0000: Best effort delivery.

0x0001: Reliable delivery.

3 Protocol Details

3.1 Common Details

3.1.1 Abstract Data Model

None.

3.1.2 Timers

None.

3.1.3 Initialization

None.

3.1.4 Higher-Layer Triggered Events

None.

3.1.5 Message Processing Events and Sequencing Rules

None.

3.1.6 Timer Events

None.

3.1.7 Other Local Events

None.

3.1.8 Forming Outbound TURN Messages

A TURN message MUST begin with the transport specific header, as specified in section 2.1. The TURN message header MUST immediately follow the transport specific header, as specified in section 2.2. The **Magic Cookie** attribute, encoded as specified in section 2.2.2.8, MUST be the first attribute after the TURN header.

3.1.9 Forming Raw Data

All data sent between the protocol client and the server that is not encapsulated in either a Send request or a Data Indication MUST begin with the transport specific header as specified in section 2.1.

3.1.10 Verifying Inbound TURN Messages

A TURN message received by either the protocol client or server MUST begin with a properly formed transport specific header, as specific in section 2.1. The TURN message header MUST immediately follow the transport specific header, as specified in section 2.2.2. The **Magic Cookie** attribute, encoded as specified in section 2.2.2.8, MUST be the first attribute after the TURN message header. If any of these conditions are not met, the message is considered an improperly formed message and MUST be ignored. If the message transport is TCP, the connection SHOULD be disconnected.

3.1.11 Message Authentication

An authenticated TURN message MUST include a **Message Integrity** attribute as the last attribute of the message. This attribute and the algorithm used to authenticate the message are specified in section 2.2.2.3.

3.1.12 Digest Challenge Extension

This protocol does not use the Shared Secret authentication mechanism specified in [IETFDRAFT-TURN-08] sections 7.1 and 8.2. Instead, it uses long-term credentials that consist of a user name and password that are pre-configured on the protocol client. The server MUST be able to verify the user name and discover the associated password. These credentials are used in place of the short-term shared secrets specified in [IETFDRAFT-TURN-08] section 7.2.2. The Allocate request and Allocate error response messages have been extended to use long-term credentials in a **digest** challenge/response exchange. These messages are used in the following procedure:

1. The protocol client MUST form an initial Allocate request message, as specified in [IETFDRAFT-TURN-08] section 8.3, with the following exceptions:
2. The request MUST be formed as specified in section 3.1.8.
3. The request SHOULD include the **MS-Version** attribute as specified in section 2.2.2.16.
4. The request SHOULD include the **MS-Service Quality** attribute as specified in section 2.2.2.18.<4>
5. The request MUST NOT include a **Message Integrity** attribute.
6. Upon reception of an Allocate request message, the server does processing as specified in [IETFDRAFT-TURN-08] section 7.2, with the following exceptions:
7. The server MUST do basic message verification as specified in section 3.1.10.
8. If this message is an initial message from the protocol client and it does not contain a **Message Integrity** attribute, the server MUST respond with an Allocate error response code 401 Unauthorized.
   - The response MUST be formed as specified in section 3.1.8.
   - The response MUST include an **Error Code** attribute with the supplied error response code.
   - The response MUST include a **Realm** attribute, as specified in section 2.2.2.14.
   - The response MUST include a **Nonce** attribute, as specified in section 2.2.2.13.
   - The response SHOULD include the **Alternate Server** attribute, as specified in section 2.2.2.7.
   - The response MUST NOT include the **Message Integrity** attribute.
9. If this message does contain a **Message Integrity** attribute, it MUST be processed as specified in Step 4.
10. When the protocol client receives the Allocate error response message, it does processing as specified in [IETFDRAFT-TURN-08] section 8.4, with the following exceptions:
11. The protocol client MUST do basic message verification, as specified in section 3.1.10.

12. Continued processing of the Allocate error response depends on the response code in the **Error Code** attribute. If the error response code is 401, 431, 432, 434, 435, or 438, the protocol client SHOULD retry the Allocate request. The request MUST be formed as specified in [IETFDRAFT-TURN-08] section 8.3, with the following exceptions:

    - The request MUST be formed as specified in section 3.1.8.
    - The request SHOULD include the **MS-Version** attribute, as specified in section 2.2.2.16.
    - The request MUST include the **Username** attribute, as specified in section 2.2.2.2.
    - The request MUST include the **Realm** attribute, as specified in section 2.2.2.14.
    - The request MUST include the **Nonce** attribute, as specified in section 2.2.2.13. The **Nonce** value MUST be equal to what the server sent in the 401 error response message.
    - The request SHOULD include the **MS-Service Quality** attribute, as specified in section 2.2.2.18.<5>
    - The request MUST be authenticated as specified in section 3.1.11.

13. Error response codes other than those listed above MUST be processed as specified in [IETFDRAFT-TURN-08] section 8.4.

14. Upon receipt of the second Allocate request message, the server does processing as specified in [IETFDRAFT-TURN-08] section 7.2, with the following exceptions:

15. The server MUST do basic message verification as specified in section 3.1.10.

16. The request MUST contain the **Message Integrity** attribute.

    - If the request does not contain the **Message Integrity** attribute, it MUST be processed as specified in Step 2 using an error response code of 401 Unauthorized.

17. The request MUST contain the **Username** attribute, as specified in section 2.2.2.2.

    - If the request does not contain a **Username** attribute, it MUST be processed as specified in Step 2 using an error response code of 432 Missing Username.
    - If the request contained a **Username** attribute, but the value of the attribute was not understood by the server, the request MUST be processed as specified in Step 2 using an error response code of 436 Unknown User.

18. The request MUST contain the **Realm** attribute, as specified in section 2.2.2.14.

    - If the request does not contain a **Realm** attribute, it MUST be processed as specified in Step 2 using an error response code of 434 Missing Realm.

19. The request MUST contain the **Nonce** attribute, as specified in section 2.2.2.13.

    - If the request does not contain a **Nonce** value, it MUST be processed as specified in Step 2, using an error response code of 435 Missing Nonce.
    - If the request contained a **Nonce** attribute, but the Nonce value was not valid, the request MUST be processed as specified in Step2 using an error response code of 438 Stale Nonce.

20. If all of the required attributes are present and valid, the server MUST authenticate the Allocate request message as specified in section 3.1.11.

21. If authentication fails, the request MUST be processed as specified in Step 2 using an error response code of 431 Integrity Check Failure.

22. If authentication succeeds, the server MUST attempt to allocate a public transport address on behalf of the protocol client. If the allocation is successful, the server MUST respond with an Allocate response.

23. The response MUST be formed as specified in section 3.1.8.

24. The response MUST contain a **Mapped Address** attribute, as specified in section 2.2.2.1. The value of the attribute MUST be that of the transport address allocated by the server.

25. The response MUST contain the **XOR Mapped Address** attribute, as specified in section 2.2.2.15.

26. The response can contain the **MS-Sequence Number** attribute, as specified in section 2.2.2.17.

27. The response MUST be authenticated as specified in section 3.1.11.

3.2 Client Details

3.2.1 Abstract Data Model

This protocol uses the abstract data model specified in [IETFDRAFT-TURN-08] section 8.

3.2.2 Timers

Retransmission Timer: This timer SHOULD be used by the protocol client for retransmission of the Allocate request and Set Active Destination request messages when the protocol client fails to receive a response from the server. The protocol client SHOULD start this timer when the request message has been sent to the wire. The protocol client SHOULD retransmit these request messages at a fixed interval of 650 milliseconds. The protocol client SHOULD retransmit a maximum of 9 times before assuming the transaction and TURN session are no longer valid.

3.2.3 Initialization

The protocol client MUST know the transport address of the TURN server and a peer with which it wants to communicate. The protocol client also MUST have long-term credentials that it can use to authenticate with the server.

3.2.4 Higher-Layer Triggered Events

3.2.4.1 Allocating a Public Address

When a protocol client is ready to allocate a public address, it MUST follow the procedure as specified in this section. This procedure replaces [IETFDRAFT-TURN-08] section 8.2 and supplements section 8.3 and is explained in detail in section 3.1.12.

The protocol client MUST send an initial Allocate request message to the server.

- The request MUST be formed as specified in section 3.1.8.
- The request SHOULD include the **MS-Version** attribute, as specified in section 2.2.2.16.
- The request SHOULD include the **MS-Service Quality** attribute, as specified in section 2.2.2.18.<6>

- The request MUST NOT include a **Message Integrity** attribute.

3.2.4.2 Sending TURN Encapsulated Data to the Peer

When the protocol client needs to send encapsulated data to a peer and set permissions with the server to allow data from that peer to be relayed to the protocol client, it MUST follow the procedure specified in [IETFDRAFT-TURN-08] section 8.6 with the following exceptions:

- The Send request message MUST be formed as specified in section 3.1.8.
- The request SHOULD include the **MS-Version** attribute, as specified in section 2.2.2.16.
- The request MUST include the **MS-Sequence Number** attribute, as specified in section 2.2.2.17.
- The request MUST be authenticated using the procedure specified in section 3.1.11.

3.2.4.3 Set the Peer as the Active Destination

When the protocol client selects a peer that it wants to use as the destination for all non-TURN encapsulated data, it MUST follow the procedures in [IETFDRAFT-TURN-08] section 8.8, with the following exceptions:

- The Set Active Destination request message MUST be formed as specified in section 3.1.8.
- The request SHOULD include the **MS-Version** attribute, as specified in section 2.2.2.16.
- The request MUST include the **MS-Sequence Number** attribute, as specified in section 2.2.2.17.
- The request MUST be authenticated using the procedure specified in section 3.1.11.
- The protocol client SHOULD NOT implement the state computer from [IETFDRAFT-TURN-08] section 8.8 controlling the transition from one active peer to another. This mechanism has been removed from more recent versions of the draft.

3.2.4.4 Tearing Down an Allocation

When the protocol client is done with the allocated address, it MUST follow the procedure specified in [IETFDRAFT-TURN-08] section 8.9, with the following exceptions:

- The Allocate request message MUST be formed as specified in section 3.1.8.
- The request SHOULD contain the **MS-Version** attribute, as specified in section 2.2.2.16.
- The request MUST contain the **MS-Sequence Number** attribute, as specified in section 2.2.2.17.
- The request SHOULD<7> include the **MS-Service Quality** attribute, as specified in section 2.2.2.18.
- The request MUST be authenticated using the procedure specified in section 3.1.11.

3.2.4.5 Sending Non-TURN Data to the Peer

When the protocol client is sending data to a peer that has been set as the active destination with the server, it MUST follow the procedure specified in [IETFDRAFT-TURN-08] section 8.10, with the exception that the data MUST be formed as specified in section 3.1.9.

3.2.5 Message Processing Events and Sequencing Rules

3.2.5.1 Receiving Allocate Response Messages

When a protocol client receives an Allocate response message, it MUST follow the procedure specified in [IETFDRAFT-TURN-08] section 8.4, with the following exceptions:

- The response MUST be verified as specified in section 3.1.10.
- The response MUST be authenticated as specified in section 3.1.11.
- The response MUST contain a **Mapped Address** attribute, as specified in section 2.2.2.1. This attribute identifies the public transport address allocated by the TURN server.
- The response MUST contain the **XOR Mapped Address** attribute, as specified in section 2.2.2.15.
- The response can contain the **MS-Sequence Number** attribute, as specified in section 2.2.2.17.

The protocol client can advertise the public transport address contained in the **Mapped Address** attribute as a destination address to receive data over. The protocol client can use the transport address contained in the **XOR Mapped Address** to identify its public address as seen by the TURN server.

3.2.5.2 Receiving Allocate Error Response Messages

When a protocol client receives an Allocate error response, it MUST follow the procedure specified in [IETFDRAFT-TURN-08] section 8.4, with the exception that the response MUST be verified as specified in section 3.1.10.

If the error response code is 401, 431, 432, 434, 435, or 438, the protocol client SHOULD retry the Allocate request as follows:

- The request MUST be formed as specified in section 3.1.8.
- The request MUST include the **Username** attribute, as specified in section 2.2.2.2.
- The request MUST include the **Realm** attribute, as specified in section 2.2.2.14.
- The request MUST include the **Nonce** attribute, as specified in section 2.2.2.13. The Nonce value MUST be equal to what the server sent in the previous 401 error response message.
- The request SHOULD<8> include the **MS-Service Quality** attribute, as specified in section 2.2.2.18.
- The request MUST be authenticated as specified in section 3.1.11.

Processing for other error response codes MUST be done as specified in [IETFDRAFT-TURN-08] section 8.4.

3.2.5.3 Receiving Set Active Destination Response Messages

When a protocol client receives a Set Active Destination response message, it MUST follow the procedure specified in [IETFDRAFT-TURN-08] section 8.8, with the following exceptions:

- The response MUST be verified as specified in section 3.1.10.

- The response MUST be authenticated as specified in section 3.1.11.
- The protocol client SHOULD NOT implement the state computer from [IETFDRAFT-TURN-08] section 8.8 controlling the transition from one active peer to another. This mechanism has been removed from more recent versions of the draft. When the protocol client receives the set Active Destination response message, it SHOULD assume that the server has set the active destination.

3.2.5.4 Receiving Set Active Destination Error Response Messages

When a protocol client receives a Set Active Destination error response message, it MUST follow the procedure specified in [IETFDRAFT-TURN-08] section 8.8, with the following exceptions:

- The response MUST be verified as specified in section 3.1.10.
- The response MUST be authenticated as specified in section 3.1.11.
- The client SHOULD NOT implement the state computer from [IETFDRAFT-TURN-08] section 8.8 controlling the transition from one active peer to another. This mechanism has been removed from more recent versions of the draft. When the client receives the Set Active Destination error response message, it SHOULD assume that the server has not set an active destination.

3.2.5.5 Receiving Data Indication Messages

When a protocol client receives a Data Indication message, it MUST follow the procedure specified in [IETFDRAFT-TURN-08] section 8.7, with the exception that the indication MUST be verified as specified in section 3.1.10.

3.2.5.6 Receiving Non-TURN Data from the Server

Once the protocol client has set a peer as the active destination, it can receive non-TURN framed data from the server. This data originates from the active peer and is relayed through the server to the protocol client. When the protocol client receives this data, it MUST follow the procedure specified in [IETFDRAFT-TURN-08] section 8.10, with the exception that the data MUST be formed as specified in section 3.1.9.

3.2.6 Timer Events

Retransmission Timer Expiration: Upon expiration of the retransmission timer, the protocol client SHOULD retransmit the outstanding request message for which the timer was originally set. The protocol client SHOULD restart the timer when the retransmitted message has been sent to the wire. The protocol client SHOULD track the number of retransmit attempts it makes, and stop retransmitting after 9 attempts. If the protocol client does not receive a response after 9 attempts, it SHOULD consider the transaction to have failed.

3.2.7 Other Local Events

None.

3.3 Server Details

3.3.1 Abstract Data Model

This protocol uses the abstract data model specified in [IETFDRAFT-TURN-08] section 7.

3.3.2 Timers

Lifetime Timer: The lifetime timer MUST be implemented as specified in [IETFDRAFT-TURN-08] section 7.7.

3.3.3 Initialization

The server MUST be initialized to receive request messages over TCP and/or UDP. It MUST be ready to receive messages on the default UDP TURN port 3478. It SHOULD be listening on TCP port 443.

3.3.4 Higher-Layer Triggered Events

None.

3.3.5 Message Processing Events and Sequencing Rules

3.3.5.1 Receiving Allocate Request Messages

Upon receipt of an Allocate request message, the server does processing as specified in [IETFDRAFT-TURN-08] section 7.2, with the following exceptions:

1. The server MUST do basic message verification as specified in section 3.1.10.
2. If the request does not contain a **Message Integrity** attribute, the server MUST respond with an Allocate error response message with an error response value of 401 Unauthorized. The message MUST be formed as follows:
   - The response MUST be formed as specified in section 3.1.8.
   - The response MUST include an **Error Code** attribute with the appropriate error response code.
   - The response MUST include a **Realm** attribute, as specified in section 2.2.2.14.
   - The response MUST include a **Nonce** attribute, as specified in section 2.2.2.13.
   - The response SHOULD include the **Alternate Server** attribute, as specified in section 2.2.2.7.
   - The response MUST NOT include the **Message Integrity** attribute.
3. If the request does contain a **Message Integrity** attribute, it MUST be processed as follows:
4. The request MUST contain the Username attribute, as specified in section 2.2.2.2.
   - If the request does not contain a **Username** attribute, the server MUST respond with an Allocate error response, as specified in Step 2, with an error response code of 432 Missing Username.
   - If the request contained a **Username** attribute, but the value of the attribute was not understood by the server, the server MUST respond with an Allocate error response, as specified in Step 2, with an error response code of 436 Unknown User.
5. The request MUST contain the **Realm** attribute, as specified in section 2.2.2.14.
   - If the request does not contain a **Realm** attribute, the server MUST respond with an Allocate error response, as specified in Step 2, with an error response code of 434 Missing Realm.
6. The request MUST contain the **Nonce** attribute, as specified in section 2.2.2.13.

- If the request does not contain a **Nonce** attribute, the server MUST respond with an Allocate error response, as specified in Step 2, with an error response code of 435 Missing Nonce.
- If the request contained a **Nonce** attribute, but the value was not valid, the server MUST respond with an Allocate error response, as specified in Step 2, with an error response code of 438 Stale Nonce.

7. If all of the required attributes are present and valid, the server MUST authenticate the Allocate request message as specified in section 3.1.11.
8. If authentication fails, the server MUST respond with an Allocate error response, as specified in Step 2, with an error response code of 431 Integrity Check Failure.
9. If authentication succeeds, the server MUST attempt to allocate a public transport address on behalf of the protocol client.
10. If allocation of a transport address fails for any reason, the server MUST respond with an Allocate error response, as specified in Step 2, with an error response code of either 300 Alternate Server or 500 Server Error. The server SHOULD use an error response code of Alternate Server if it is configured such that it knows about other servers in the deployment that implement this protocol. Otherwise the server MUST use an error response code of Server Error .
11. If the allocation of the transport address is successful, the server MUST respond with an Allocate response.
12. The response MUST be formed as specified in section 3.1.8.
13. The response MUST contain a **Mapped Address** attribute, as specified in section 2.2.2.1.The value of the attribute MUST be that of the transport address allocated by the server.
14. The response MUST contain the **XOR Mapped Address** attribute, as specified in section 2.2.2.15.
15. The response can contain the **MS-Sequence Number** attribute, as specified in section 2.2.2.17.
16. The response MUST be authenticated as specified in section 3.1.11.

3.3.5.2 Receiving Send Request Messages

Processing of a Send request message is done as specified in [IETFDRAFT-TURN-08] section 7.3 with the following exceptions:

- The request MUST be verified as specified in section 3.1.10. If the request fails verification it MUST be silently dropped by the server.
- The request MUST be authenticated as specified in section 3.1.11. If the request fails authentication it MUST be silently dropped by the server.
- The server MUST NOT respond to a protocol client with either a Send response or a Send error response.

3.3.5.3 Receiving Set Active Destination Request Messages

Processing of a Set Active Destination request message is done as specified in [IETFDRAFT-TURN-08] section 7.5, with the following exceptions:

- The request MUST be verified as specified in section 3.1.10.

- The request MUST be authenticated as specified in section 3.1.11.

Any response message sent to the protocol client after processing the request is formed as specified in [IETFDRAFT-TURN-08] section 7.5, with the following exceptions:

- The response MUST be formed as specified in section 3.1.8.
- The response MUST be authenticated as specified in section 3.1.11.

The server SHOULD NOT implement the state computer from [IETFDRAFT-TURN-08] section 7.5 controlling the transition from one active peer to another. This mechanism has been removed from more recent versions of the draft. If the server successfully processed the request, it SHOULD set the active destination before it sends the Set Active Destination response message. If an error occurred while the server was processing the request, it SHOULD NOT change the current active destination. If this is the first Set Active Destination request, the server SHOULD NOT set an active destination. If the active destination has been set through an earlier Set Active Destination request, the server SHOULD NOT change the active destination.

3.3.5.4 Receiving Data and Connections on the Allocated Transport Address

Processing of incoming data or connection requests on the allocated transport address is done as specified in [IETFDRAFT-TURN-08] section 7.4, with the following exceptions:

- If the received data results in a Data Indication message sent to the protocol client, the Data Indication message MUST be formed as specified in section 3.1.8.
- If the received data is from a peer that has been identified as the active peer through a Set Active Destination request, it MUST be formed as specified in section 3.1.9.

3.3.5.5 Receiving Non-TURN Data from the Client

Once the protocol client has set a peer as the active destination, it can send non-TURN framed data to the server. This data is relayed through the server to the active peer. When the server receives this data, it MUST follow the procedure specified in [IETFDRAFT-TURN-08] section 7.6, with the exception that the data MUST be formed as specified in section 3.1.9.

3.3.6 Timer Events

Lifetime Expiration: When the lifetime timer fires, the server MUST process it as specified in [IETFDRAFT-TURN-08] section 7.7.

3.3.7 Other Local Events

None.

4 Protocol Examples

In the following figure, a TURN protocol client is behind a NAT and is communicating with a peer using **Session Initiation Protocol (SIP)**, as described in [RFC3261]. The protocol client and peer wish to establish a media flow between them. Since the protocol client is behind a NAT, it must allocate a public transport address which it can include in the **Session Description Protocol (SDP)** of the SIP **INVITE** sent to the peer, as described in [RFC4566]. The details of the SIP message exchange are not included in the example; only the basic message flow used to communicate the public address of the protocol client and peer to each other is included.

The TURN protocol client has a private transport address of 10.0.0.1 which it uses for network connectivity. The NAT on the protocol client's private network has a public transport address of 192.0.2.10. The TURN server has a public transport address of 192.0.2.20. The peer is connected directly to the internet and has a transport address of 192.0.2.30. The following figure shows the flow of TURN messages used to allocate a public transport address.

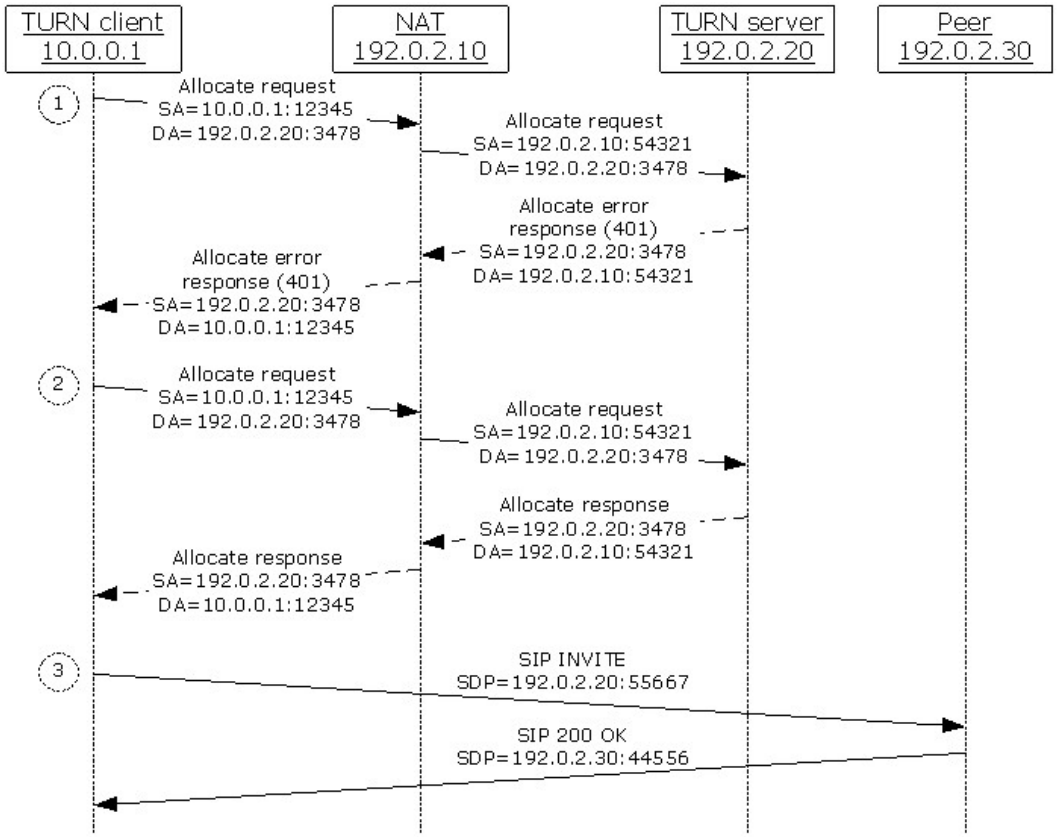


Figure 3: Example of TURN message flow

1. The protocol client sends an initial Allocate request message to the server. This request message does not include a **Message Integrity** attribute and begins the **digest** authentication exchange specified in section 3.1.12. The source address for the request is 10.0.0.1:12345 and the destination address is 192.0.2.20:3478. The request passes through the NAT which allocates a new port, 54321, and creates a binding between the internal address 10.0.0.1:12345 and 192.0.2.10:54321. The NAT translates the source address to 192.0.2.10:54321 and sends the request on to the TURN server. The TURN server checks the request for a **Message Integrity**

attribute. Since the **Message Integrity** attribute is missing, the server challenges the protocol client for credentials by responding with an Allocate error response or with an error response code of 401Unauthorized. The server sends the response message to the protocol client, through the NAT binding, with the NAT translating the destination address.

2. When the protocol client receives the Allocate error response message, it retries the Allocate request using the **Username**, **Nonce**, and **Realm** attributes specified in section 3.1.12. The request is sent through the NAT binding to the server, with the NAT translating the source address discussed in Step 1. The server validates and authenticates the new Allocate request and allocates transport address 192.0.2.20: 55667. It forms an Allocate response message and includes the **Mapped Address** attribute with a value of 192.0.2.20:55667 and the **XOR Mapped Address** attribute with a value of 192.0.2.10:54321 xor'd with the transaction ID, as specified in section 2.2.2.15. The response is sent to the protocol client through the NAT binding, with the NAT again doing the required address translation.

3. The protocol client receives the Allocate response and uses the Mapped Address, 192.0.2.20:55667, in the SDP of the SIP INVITE to signal to the peer the address it should send data to. The peer responds to the SIP INVITE with a SIP **200 OK** and includes its address of 192.0.2.30:44556 in the SDP.

At this point, both the protocol client and the peer have a transport address that they can use to receive data. However, until the protocol client has set permission on the allocated port, the server does not allow any data to be received on the allocated port. The following figure shows the messages used to set permissions on an allocated port and the subsequent data flow.

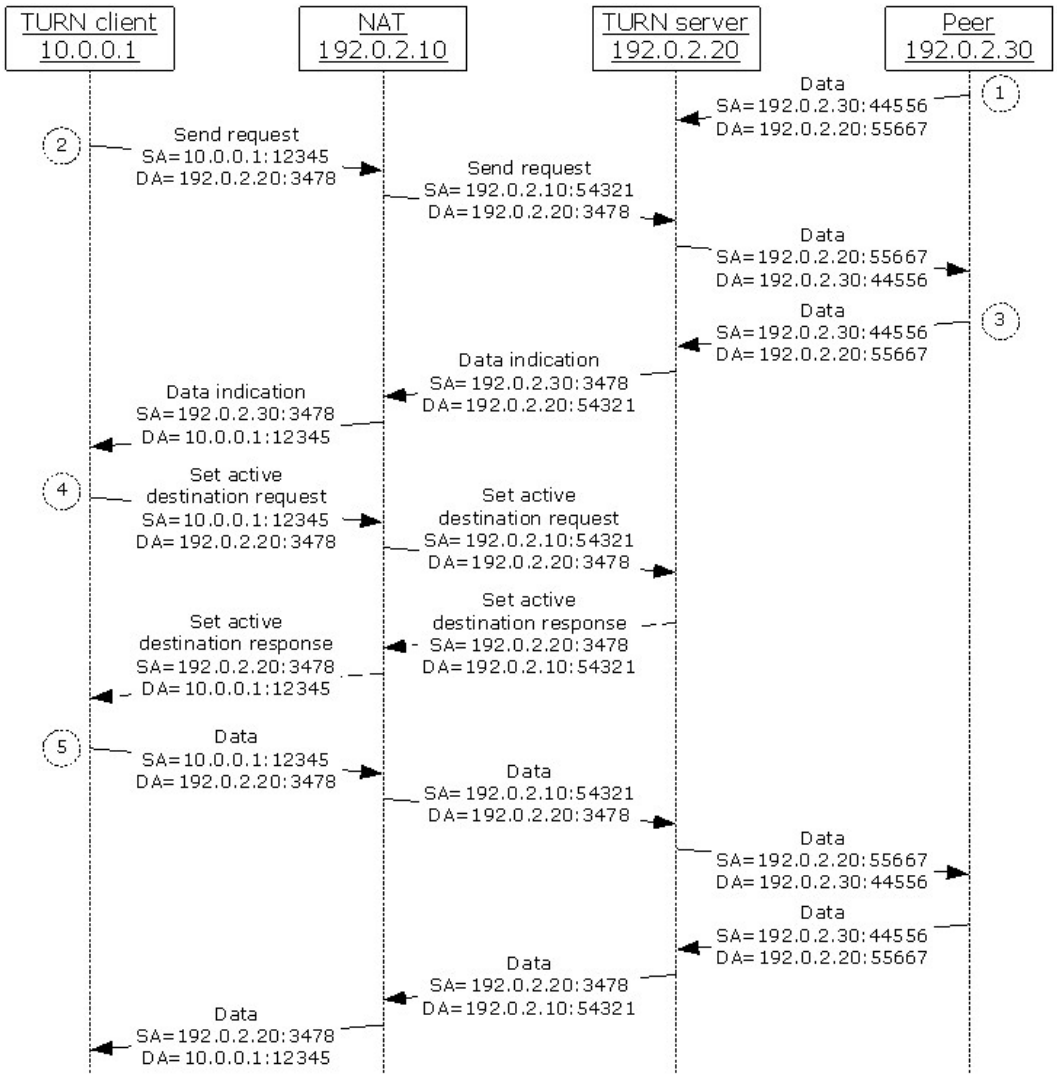


Figure 4: Using TURN messages to set permissions

1. Once the peer has the public transport address of the protocol client, it can start to send data. When the data arrives at the allocated port on the server, the server checks to see if the protocol client has permissions to receive data from the peer, 192.0.2.30:44556. Permissions are set when the protocol client does a Send request to the server with the peer's transport address in the **Destination Address** attribute. Since the protocol client has not sent a Send request, the server drops the data.

2. Once the protocol client has the public transport address of the peer, it can start to send data. It does this by sending a Send request message to the server with the data to be sent in the **Data** attribute and the address of the peer, 192.0.2.30:44556, in the **Destination Address** attribute. The Send request is sent to the server through the NAT binding. When the server receives the Send request, it adds the peer's IPv4 address to the permissions list for the allocated address. It then forwards the data contained in the **Data** attribute on to the peer. The data is sent using the

allocated address, 192.0.2.20:55667, as the source address and the address in the **Destination Address** attribute, 192.0.2.30:44556, as the destination address.

3. The peer again attempts to send data to the allocated address. The server checks the permissions list and finds that the peer now has permissions to send data to the protocol client. The server forwards the data to the protocol client using a Data Indication message, encapsulating the data in the **Data** attribute and identifying the peer as the source of the data by including a **Remote Address** attribute with the peer's address. The Data Indication message is sent to the protocol client through the NAT binding.

4. The protocol client is now ready to make the peer the active destination for all non-TURN encapsulated data. It sends a Set Active Destination request message to the server with the peer's address in the **Destination Address** attribute. The request is sent to the server through the NAT binding. When the server receives the request, it identifies the peer as the active destination and sends a Set Active Destination response back to the protocol client.

5. Now that the protocol client has established the peer as the active destination, all non-TURN data sent by either the protocol client or the peer is relayed between the two with non-TURN message encapsulation. Only transport specific framing is required. This is a more efficient mechanism for relaying the data.

5 Security

5.1 Security Considerations for Implementers

The security considerations for this protocol are the same as described in [IETFDRAFT-TURN-08] section 10.

The long-term credentials, which are used for protocol client authentication with the server, are valid for an extended period of time. Since the credentials are valid for this extended period, replay prevention is provided through the use of a **digest** challenge as described in section 3.1.12.

The long-term credential mechanism is also susceptible to offline dictionary attacks so it is recommended that deployments utilize strong passwords.

5.2 Index of Security Parameters

Security Parameter	Section
Use of long-term credentials in a **digest** challenge/response exchange.	3.1.12

6 Appendix A: Product Behavior

The information in this specification is applicable to the following Microsoft products or supplemental software. References to product versions include released service packs:

- Microsoft® Office Communications Server 2007
- Microsoft® Office Communications Server 2007 R2
- Microsoft® Office Communicator 2007
- Microsoft® Office Communicator 2007 R2
- Microsoft® Lync™ Server 2010
- Microsoft® Lync™ 2010

Exceptions, if any, are noted below. If a service pack or Quick Fix Engineering (QFE) number appears with the product version, behavior changed in that service pack or QFE. The new behavior also applies to subsequent service packs of the product unless otherwise specified. If a product edition appears with the product version, behavior is different in that product edition.

Unless otherwise specified, any statement of optional behavior in this specification that is prescribed using the terms SHOULD or SHOULD NOT implies product behavior in accordance with the SHOULD or SHOULD NOT prescription. Unless otherwise specified, the term MAY implies that the product does not follow the prescription.

<1> Section 2.2.2: Office Communications Server 2007, Office Communicator 2007: This behavior is not supported.

<2> Section 2.2.2.16: Office Communications Server 2007, Office Communicator 2007: This behavior is not supported.

<3> Section 2.2.2.18: Office Communications Server 2007, Office Communicator 2007: This behavior is not supported.

<4> Section 3.1.12: Office Communications Server 2007, Office Communicator 2007: This behavior is not supported.

<5> Section 3.1.12: Office Communications Server 2007, Office Communicator 2007: This behavior is not supported.

<6> Section 3.2.4.1: Office Communications Server 2007, Office Communicator 2007: This behavior is not supported.

<7> Section 3.2.4.4: Office Communications Server 2007, Office Communicator 2007: This behavior is not supported.

<8> Section 3.2.5.2: Office Communications Server 2007, Office Communicator 2007: This behavior is not supported. For all other products, this attribute can be included.

7 Change Tracking

No table of changes is available. The document is either new or has had no changes since its last release.

8 Index

V