

BEHAVE
Internet-Draft
Intended status: Standards Track
Expires: April 18, 2008

D. Wing
J. Rosenberg
Cisco Systems
H. Tschofenig
Nokia Siemens Networks
October 16, 2007

Discovering, Querying, and Controlling Firewalls and NATs
draft-wing-behave-nat-control-stun-usage-05

Status of this Memo

By submitting this Internet-Draft, each author represents that any applicable patent or other IPR claims of which he or she is aware have been or will be disclosed, and any of which he or she becomes aware will be disclosed, in accordance with [Section 6 of BCP 79](#).

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF), its areas, and its working groups. Note that other groups may also distribute working documents as Internet-Drafts.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

The list of current Internet-Drafts can be accessed at <http://www.ietf.org/ietf/lid-abstracts.txt>.

The list of Internet-Draft Shadow Directories can be accessed at <http://www.ietf.org/shadow.html>.

This Internet-Draft will expire on April 18, 2008.

Copyright Notice

Copyright (C) The IETF Trust (2007).

Abstract

A drawback with many NAT UDP hole punching techniques is the keepalive traffic necessary to keep the UDP binding open. It is necessary to send keepalives frequently because it is not possible to determine or modify the NAT's binding lifetime. This keepalive traffic causes server load and additional network traffic, which is especially problematic with battery-operated wireless devices.

Internet-Draft

STUN Control

October 2007

This document describes two mechanisms to discover NATs and firewalls and a mechanism to query and control their binding lifetime. With these mechanisms, UDP binding discovery and UDP keepalive traffic can be reduced to involve only the necessary NATs or firewalls. This eliminates the keepalive traffic to servers, and vastly reduces keepalive traffic across the network. At the same time, backwards compatibility with NATs and firewalls that do not support this specification is retained, which allows for incremental deployment of this mechanism.

This document is discussed on the SAFE mailing list,
<<http://www1.ietf.org/mailman/listinfo/safe>>.

Table of Contents

1.	Introduction	4
2.	Notational Conventions	4
3.	Motivation and Benefits	4
3.1.	Comparison with other NAT Traversal Techniques	6
3.1.1.	Simple Security Model	6
3.1.2.	Incremental Deployment	6
3.2.	Reduce Keepalive Messages	7
3.2.1.	SIP Outbound	7
3.2.2.	IKE/IPsec NAT Traversal	7
3.2.3.	Teredo	8
3.3.	Optimize ICE	8
3.3.1.	Candidate Gathering	8
3.3.2.	Learning STUN Servers without Configuration	8
3.3.3.	Reduce Media Keepalive Messages	9
4.	Overview of Operation	9
5.	Discovery of Middleboxes (NATs and Firewalls)	10
5.1.	Outside-In	10
5.1.1.	Nested NATs	12
5.2.	Tagging	14
6.	Query and Control	15
6.1.	Client Procedures	15
6.2.	Server Procedures	15
7.	New Attributes	16
7.1.	REFRESH-INTERVAL Attribute	16
7.2.	XOR-INTERNAL-ADDRESS Attribute	16
7.3.	PLEASE-TAG Attribute	17
7.4.	TAG Attribute	17

7.5.	BOOTNOUNCE Attribute	18
8.	Limitations of STUN Control	19
8.1.	Overlapping IP Addresses with Nested NATs	19
8.2.	Address Dependent NAT on Path	19
8.3.	Address Dependent Filtering	20

8.4.	Interacting with Legacy NATs	20
9.	Security Considerations	21
9.1.	Authorization	21
9.2.	Resource Exhaustion	21
9.3.	Comparison to Other NAT Control Techniques	21
9.4.	BOOTNOUNCE Attribute	22
10.	Open Issues and Discussion Points	22
11.	IANA Considerations	24
12.	Acknowledgements	24
13.	References	24
13.1.	Normative References	24
13.2.	Informational References	25
Appendix A.	Changes	26
A.1.	Changes in -05	26
A.2.	Changes in -04	27
A.3.	Changes in -03	27
Appendix B.	Implementation Details	27
B.1.	Internal NAT Operation	27
B.2.	Linux specifics	28
Authors' Addresses	29
Intellectual Property and Copyright Statements	31

Internet-Draft

STUN Control

October 2007

1. Introduction

Two common usages of Simple Traversal Underneath NAT (STUN) ([[I-D.ietf-behave-rfc3489bis](#)],[[RFC3489](#)]) are Binding Discovery and NAT Keepalive. The Binding Discovery usage allows a STUN client to learn its public IP address (from the perspective of the STUN server it contacted) and the NAT keepalive usage allows a STUN client to keep an active NAT binding alive. Unlike some other techniques (e.g., UPnP IGD [[UPnP-IGD](#)], MIDCOM [[RFC3303](#)], NAT-PMP [[I-D.cheshire-nat-pmp](#)]), NSIS-NSLP [[I-D.ietf-nsis-nslp-natfw](#)]), STUN does not interact directly with the NAT. Thus, STUN cannot request additional services from the NAT, such as longer lifetimes which would reduce keepalive messages. Furthermore, allocating new NAT bindings (e.g., each phone call) requires communication with a STUN server located somewhere on the Internet.

This document describes three mechanisms for the STUN client to discover NATs and firewalls that are on path with its STUN server. After discovering the NATs and firewalls, the STUN client can query and control those devices using STUN. The STUN client needs to only ask those STUN servers (embedded in the NATs and firewalls) for public IP addresses and UDP ports, thereby offloading that traffic from the STUN server on the Internet. Additionally, the STUN client can ask the NAT's embedded STUN server to extend the NAT binding for the flow, and the STUN client can learn the IP address of the next-outermost NAT. By repeating this procedure with the next-outermost NAT, all of the NATs along that path can have their bindings extended. By learning all of the STUN servers on the path between the public Internet and itself, an endpoint can optimize the path of

peer-to-peer communications.

[2.](#) Notational Conventions

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 [[RFC2119](#)].

[3.](#) Motivation and Benefits

There are a number of problems with existing NAT traversal techniques, such as STUN, UPnP IGD, MIDCOM, NAT-PMP, and NSIS-NSLP:

nested NATs:

Today, many ISPs provide their subscribers with modems that have embedded NATs or within the ISP's network. These subscribers then install NATs behind those devices to provide additional features, such as wireless access. In these situations, UPnP IGD and NAT-PMP no longer function, as those protocols can only control the first NAT closest to the host. STUN continues to function, but is unable to optimize network traffic behind those nested NATs (e.g., traffic that stays within the same house or within the same apartment building).

One technique to avoid nested NATs is to disable one of the NATs, as recommended by [[Vista-cert](#)]. However, this merely sidesteps the problem of nested NATs, as some NATs are installed for a reason (e.g., reduce IP address consumption or provide a modicum of security). Disabling the NAT is also ineffective if the ISP is NATting subscribers within the ISP's network, as ISP NATs do not typically support UPnP.

The technique described in this document allows optimization of the traffic behind those NATs so that the traffic can traverse the fewest NATs possible.

keepalive chatter:

To keep NAT bindings from timing out and to perform its binding discovery, keepalive packets are sent to a server on the Internet. This consumes bandwidth across the user's access network, which in some cases is bandwidth constrained (e.g., wireless, satellite), creates a load on the server, and (for battery-powered devices) consumes battery power. This chattiness can be avoided by using a NAT control mechanism such as UPnP IGD or NAT-PMP. However, relying on such NAT control mechanisms exclusively for NAT traversal is problematic, as they are not universally deployed. Thus, many UDP NAT traversal techniques instead rely on UDP hole punching.

The technique described in this document provides a significant reduction of keepalive traffic. The keepalive traffic can be reduced in frequency and can even be sent to just the necessary NAT or firewall (rather than the server).

lack of incremental deployment:

Many other NAT traversal techniques require the endpoint and its NAT to both support the same NAT traversal technique or else NAT traversal is not possible at all. Examples include NSIS-NSLP, NAT-PMP, UPnP IGD, and MIDCOM.

The technique described in this document allows incremental

deployment of local endpoints and NATs that support STUN Control. If the local endpoint, or its NATs, does not support the STUN Control functionality, then STUN (see [\[I-D.ietf-behave-rfc3489bis\]](#)), [\[I-D.ietf-sip-outbound\]](#), and ICE [\[I-D.ietf-mmusic-ice\]](#) procedures are used to traverse the NATs without the optimizations described in this document.

The protocol described in this document retains the positive features of STUN -- incremental deployment and support of nested NATs -- without introducing drawbacks inherent in other NAT traversal techniques. The protocol optimizes the operation of STUN clients when those STUN clients are behind a NAT that supports the protocol described in this document. STUN clients that are behind a NAT that doesn't support the protocol described in this document continue to function as they do today, without those optimizations.

[3.1.](#) Comparison with other NAT Traversal Techniques

STUN Control offers the following benefits over other NAT traversal and NAT control techniques such as NSIS-NSLP, MIDCOM, NAT-PMP, and UPnP IGD.

[3.1.1.](#) Simple Security Model

Unlike other middlebox control techniques which have relatively complex security models because a separate control channel is used, STUN Control's is simple. It is simple because only flows originating from the same source IP and UDP port can be controlled (i.e., have its NAT timeout queried or extended). Other flows cannot be created, queried, or controlled via STUN Control.

[3.1.2.](#) Incremental Deployment

STUN Control can be incrementally deployed. If the outer-most NAT does not support it, the STUN client behaves as normal -- it merely isn't able to optimize its keepalive (see also [Section 8.4](#)). If the outer-most NAT does support STUN Control, the STUN client can gain some significant optimizations as described in the following sections.

Likewise, there is no change required to applications if NATs are deployed which support STUN Control: such applications will be unaware of the additional functionality in the NAT, and will not be subject to any worse security risks due to the additional functionality in the NAT.

[3.2.](#) Reduce Keepalive Messages

The primary value of the protocol and technique described in this document is the reduction of UDP keepalive messages. This is helpful for several protocols.

For each of the protocols below, STUN Control as described in this document enables two optimizations:

1. all of the on-path NATs can explicitly indicate their timeouts, reducing the frequency of keepalive messages, and;
2. STUN keepalive messages need only be sent to the outer-most NAT, rather than across the access link to the SIP proxy, which vastly reduces the traffic to the SIP proxy.

[3.2.1.](#) SIP Outbound

In SIP outbound [[I-D.ietf-sip-outbound](#)], the SIP proxy is also the STUN server. Through the initial STUN request/response exchange with that server, the STUN client learns it is behind a NAT, and learns that NAT's public IP address. Once it has learned the NAT's public IP address, it can query and control that NAT by following the procedures in [Section 6](#).

[3.2.2.](#) IKE/IPsec NAT Traversal

In both the NAT traversal for IKEv1 [[RFC3947](#)] and IKEv2 ([Section 2.23 of \[RFC4306\]](#)) the IKE endpoints can only learn that a NAT is present, but cannot learn the IP address of that NAT because the IP address is hashed. Thus, IKE itself isn't usable to learn the IP address of the outer-most NAT. STUN can be used to learn the IP address of the outer-most NAT, and STUN Control can then be used to extend the binding lifetime for the UDP port that is being used by IKE. Once this is done, the IPsec NAT keepalive interval can be reduced ([Section 4 of \[RFC3948\]](#)).

With IKE/IPsec NAT traversal, there are two ways to use STUN to learn the outer-most NAT:

- o STUN packets can be sent between the IKE peers on the same port as IKE. IKE, IPsec ESP, and STUN can be demultiplexed. However, this does require changing software in both IKE peers.
- o STUN packets can be sent to STUN port of the IKE peer's IP address. This does not require changing software on the remote IKE peer, but requires a separate server process running on the remote peer.

[3.2.3.](#) Teredo

Endpoints that implement Teredo [[RFC4380](#)] learn their outer-most NAT's address as their Teredo Mapped Address. Once learned, the Teredo client can utilize STUN Control to query and control that NAT's (and nested NAT's) UDP keepalive timeout, and thus reduce the refresh interval.

In contrast, Teredo's existing refresh interval determination procedure ([Section 5.2.7 of \[RFC4380\]](#)) allows the Teredo host to learn (but not adjust) the NAT's binding lifetime. There is also a small risk that the NAT will use different refresh intervals for different ports (e.g., due to resource constraints), which contributes to some brittleness.

[3.3.](#) Optimize ICE

The STUN Control usage provides several opportunities to optimize ICE [[I-D.ietf-mmusic-ice](#)], as described in this section.

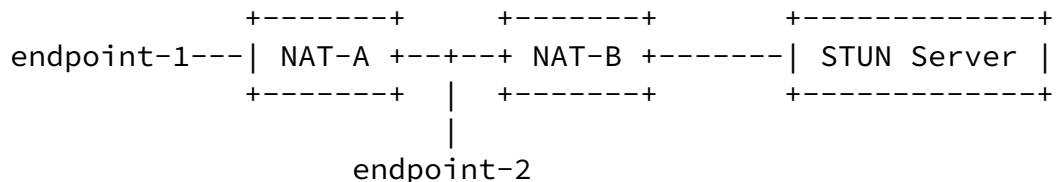
[3.3.1.](#) Candidate Gathering

During its candidate gathering phase, an ICE endpoint normally contacts a STUN server on the Internet. If an ICE endpoint discovers that its outer-most NAT runs a STUN server, the ICE endpoint can use the outer-most NAT's STUN server rather than using the STUN server on the Internet. This saves access bandwidth and reduces the reliance on the STUN server on the Internet -- the STUN server on the Internet need only be contacted once -- when the ICE endpoint first initializes.

[3.3.2.](#) Learning STUN Servers without Configuration

ICE allows endpoints to have multiple STUN servers, but it is difficult to configure all of the STUN servers in the ICE endpoint -- it requires some awareness of network topology. By using the 'walk backward' technique described in this document, all the on-path NATs and their embedded STUN servers can be learned without additional configuration. By knowing the STUN servers at each address domain, ICE endpoints can optimize the network path between two peers.

For example, if endpoint-1 is only configured with the IP address of the STUN server on the left, endpoint-1 can learn about NAT-B and NAT-A. Utilizing the STUN server in NAT-A, endpoint-1 and endpoint-2 can optimize their media path so they make the optimal path from endpoint-1 to NAT-A to endpoint-2:



[3.3.3.](#) Reduce Media Keepalive Messages

While very minor, STUN Control makes it possible to optimize media keepalives. This is useful if a video or audio stream is placed on 'hold' or 'mute', but is expected to be resumed in the future. ICE uses STUN Indications as its primary media stream keepalive mechanism. This document enables two optimizations of ICE's keepalive technique:

1. STUN keepalive messages need only be sent to the outer-most NAT, rather than across the access link to the remote peer, and;
2. all of the on-path NATs can explicitly indicate their timeouts, which allows reducing the keepalive frequency.

[4.](#) Overview of Operation

This document describes three functions, which are all implemented using the STUN protocol:

Discovery of Middleboxes (NATs and Firewalls):

This document describes two techniques for finding NATs or firewalls (see [Section 5](#)). These two approaches are:

Outside-In:

Uses STUN or Teredo to find the outer-most NAT. Then STUN is used to communicate with that NAT and discover the other nested NATs (if any) along that path towards the host by repeated use of STUN with each of those NATs.

Tagging:

Send a STUN Request packet to your STUN server, and asks for compliant firewalls along the path to indicate their presence

by adding an IP address to the STUN Response packet.

Querying Discovered Middleboxes:

After discovering a NAT or a firewall, it is useful to determine characteristics of the NAT binding or the firewall pinhole. Two of the most useful things to learn is the duration the NAT binding or firewall pinhole will remain open if there is no traffic, and the filtering applied to that binding or pinhole. This is described in [Section 6](#).

Controlling Discovered Middleboxes:

A NAT or a firewall might default to a more restrictive behavior than desired by an application (e.g., aggressive timeout, filtering). Requesting the NAT or firewall to change its default behavior is useful for traffic optimization (e.g., reduce keepalive traffic) and network optimization (e.g., adjust filters to eliminate the need for a media relay device [[I-D.ietf-behave-turn](#)]). A discussion of this functionality can be found in [Section 6](#).

[5](#). Discovery of Middleboxes (NATs and Firewalls)

This section describes two techniques to discover a NAT and a firewall: outside-in and by tagging.

Ideally, a single technique could be selected as an outcome of the standardization process. However, it is possible to combine these two techniques.

[5.1](#). Outside-In

The endpoint must first discover its outer-most NAT. This can be accomplished using STUN or Teredo.

STUN: When a STUN client sends a STUN Request to a STUN server, it receives a STUN Response that indicates the IP address and UDP port seen by the STUN server. If the IP address and UDP port differs from the IP address and UDP port of the socket used to send the request, the STUN client knows there is at least one NAT between itself and the STUN server. The STUN client also learns

the 'public' IP address (and port) allocated by the outermost NAT.

Teredo: As part of the Teredo qualification procedure, the Teredo client learns the IP address of its outer-most NAT. With that information, the Teredo client can proceed to the next step.

After learning the public IP address of its outer-most NAT, the endpoint sends a STUN packet to the STUN port (UDP/3478) of its outer-most NAT's public IP address. The NAT will return a STUN

Binding Response message including two important STUN attributes:

XOR-MAPPED-ADDRESS, which indicates the public IP address and UDP port for the mapping. As the endpoint just learned this information via STUN or Teredo, this isn't terribly interesting to the endpoint at this time. However, if the endpoint wants to create a new UDP mapping (e.g., for a new UDP flow), the endpoint need only send a STUN request to this outer-most NAT rather than to a host on the Internet.

XOR-INTERNAL-ADDRESS, which indicates the IP address and UDP port seen on the *internal* side of the NAT for that translation (see Figure 13). This allows the endpoint to discover, query, and control multiple NATs (nested NATs) along that path.

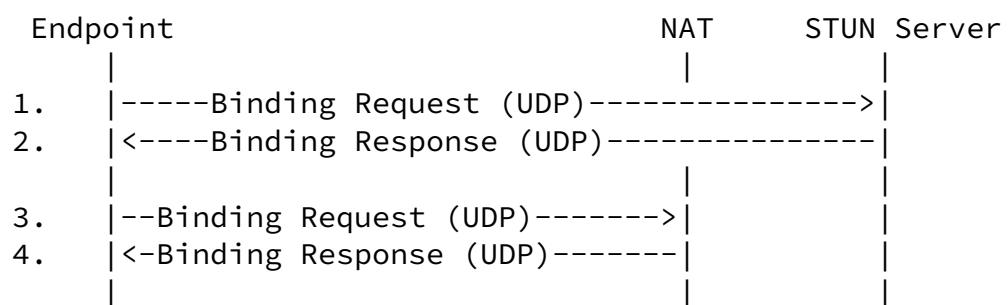


Figure 2: Communication Flow

In the message flow above, steps 1 and 2 correspond to the STUN behavior described in [[I-D.ietf-behave-rfc3489bis](#)]:

1: The STUN client sends a UDP Binding Request to its STUN server

that is located on the Internet.

- 2: The STUN server on the Internet responds with a UDP Binding Response.

After steps 1 and 2, the endpoint has learned the IP address of its outer-most NAT. The endpoint could also have used Teredo to learn that IP address.

The next steps are the additional steps performed by the endpoint implementing STUN Control:

- 3: The endpoint sends a STUN Binding Request to the IP address of its outer-most NAT. This will be received by the STUN server embedded in that outer-most NAT.
- 4: The STUN server (embedded in the NAT) responds with a STUN Binding Response.

The response obtained in message (4) contains the XOR-MAPPED-ADDRESS attribute, which will have the same value as when the STUN server on the Internet responded (in step 2). Thereafter, so long as the BOOTNOUNCE value doesn't change, the STUN client can perform steps (3) and (4) for any new UDP communication, without needing to repeat steps (1) and (2). This meets the desire to reduce chattiness. The STUN client also only needs to send keepalives towards the outer-most NAT's IP address, as well (reduces chatter for SIP outbound [[I-D.ietf-sip-outbound](#)]).

The response obtained in message (4) will also contain the XOR-INTERNAL-ADDRESS, which allows the STUN client to repeat steps (3) and (4) in order to query or control those on-path NATs between itself and its STUN server on the Internet. This is described in detail in [Section 5.1.1](#). This functionality allows ICE to learn more NAT bindings [Section 3.3.2](#) and gives ICE the opportunity to optimize traffic between nested NATs, without requiring configuration of intermediate STUN servers.

The STUN client can request each NAT to increase the binding lifetime for that source IP address and source UDP port, as described in [Section 7.1](#). The STUN client receives positive confirmation that the binding lifetime has been extended, allowing the STUN client to significantly reduce its NAT keepalive traffic. Additionally, as long as the NAT complies with [\[RFC4787\]](#) (which is indicated by its support of this document), the STUN client's keepalive traffic need only be sent to the outer-most NAT's IP address. This functionality meets the need to reduce STUN's chattiness.

[5.1.1](#). Nested NATs

Nested NATs are controlled individually. The nested NATs are discovered, from outer-most NAT to the inner-most NAT, using the XOR-INTERNAL-ADDRESS attribute.

If there is only one NAT between an endpoint and the Internet, XOR-INTERNAL-ADDRESS will return the same IP address and UDP port the endpoint is using. If there are multiple NATs between an endpoint and the Internet, XOR-INTERNAL-ADDRESS will return a different IP address than the endpoint is using, which points towards the NAT closer to the endpoint. By repeating this procedure, the endpoint

can discover all of the NATs. Note, however, the limitation described in [Section 8.1](#).

The following figure shows two nested NATs:

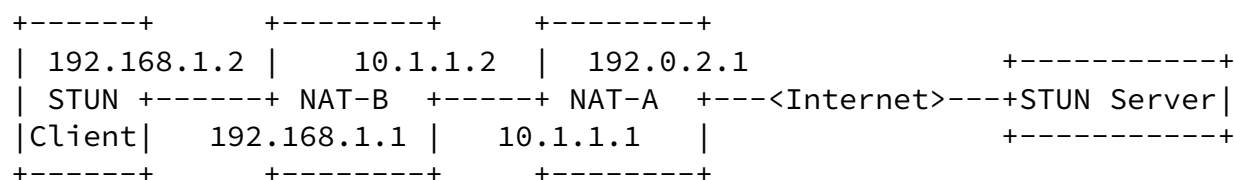


Figure 3: Two nested NATs with embedded STUN servers

First, the endpoint would learn the outer-most NAT's IP address via STUN or Teredo. The endpoint will then send a STUN binding request to that outer-most NAT. With nested NATs, however, the IP address and UDP port indicated by the XOR-INTERNAL-ADDRESS will not be the

STUN client's own IP address and UDP port -- rather, it is the IP address and UDP port on the inside of NAT-A, which are the same as the IP address and UDP port on the outside of the NAT-B -- 10.1.1.2.

Because of this, the STUN client repeats the procedure and sends another STUN Binding Request to that newly-learned address (the *outer* side of NAT-B). NAT-B will respond with a STUN Binding Response containing the XOR-INTERNAL-ADDRESS attribute, which will match the STUN client's IP address and UDP port. The STUN client then knows there are no other NATs between itself and NAT-B, and finishes.

The message flow with two nested NATs is shown below:

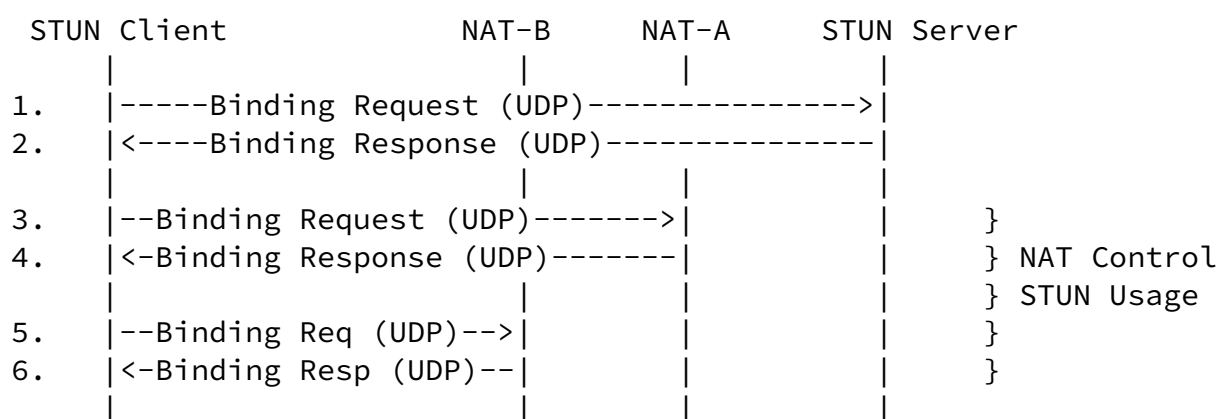


Figure 4: Message Flow for Outside-In with Two NATs

A BOOTNOUNCE value is obtained from each of these NATs, and is

validated whenever a subsequent STUN Binding Request is sent to any of those learned NATs.

5.2. Tagging

To discover an on-path firewall, the PLEASE-TAG attribute is used with a STUN Binding Request (a STUN packet sent to UDP/3478) message. A firewall would inspect bypassing Binding Request messages and determine whether there is a PLEASE-TAG attribute. When the firewall sees the associated Binding Response, the firewall appends a TAG attribute as the last attribute of the Binding Response. This TAG

attribute contains the firewall's management IP address and UDP port. Each on-path firewall would be able to insert its own TAG attribute. In this way, the STUN Response would contain a pointer to each of the on-path firewalls between the client and that STUN server.

Motivation for developing the Tagging mechanism: The Outside-In discovery technique ([Section 5.1](#)) uses the public IP address of the NAT to find the outer-most NAT that supports STUN Control. Firewalls do not translate packets and hence a different technique is needed to identify firewalls.

Note that tagging is similar to how NSIS-NSLP [[I-D.ietf-nsis-nslp-natfw](#)], TIST [[I-D.shore-tist-prot](#)], and NLS [[I-D.shore-nls-tl](#)] function.

This figure shows how tagging functions.

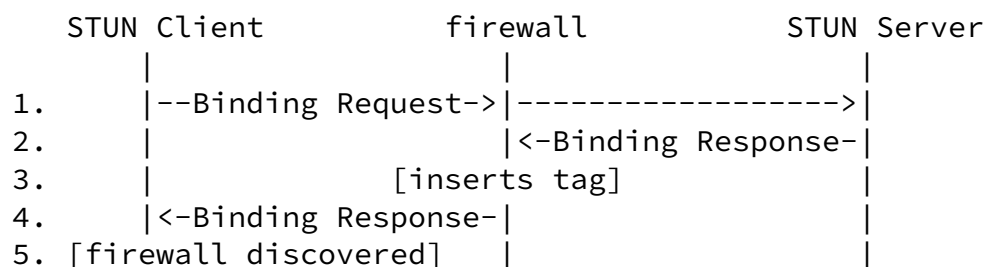


Figure 5: Tagging Message Flow

1. A Binding Request, containing the PLEASE-TAG attribute, is sent to the IP address of the STUN server that is located somewhere on the Internet. This is seen by the firewall, and the firewall remembers the STUN transaction id, and permits the STUN Binding Request packet.
2. When the firewall observes a STUN Binding Response packet it checks its cache for the previously stored STUN transaction id. If a previous STUN transaction id was found then the firewall inserts the TAG attribute, which contains the firewall's

management address.

3. The firewall sends the (modified) STUN Binding Response towards the STUN client.

4. The STUN client has now discovered the firewall, and can query it or control it.

6. Query and Control

This section describes how to use STUN to query and control a NAT that was discovered using the technique described in [Section 5](#).

6.1. Client Procedures

After discovering on-path NATs and firewalls with the procedure described in [Section 5](#), the STUN client begins querying and controlling those devices.

To modify an existing NAT mapping's attributes, or to request a new NAT mapping for a new UDP port, the STUN client can now send a STUN Binding Request to the IP address of address of the respective NAT or firewall (using the STUN UDP port, 3478).

Client produces for handling the BOOTNOUNCE attribute can be found in [Section 7.5](#).

6.2. Server Procedures

When receiving a STUN Binding Request the STUN controlled NAT will respond with a STUN Binding Response containing an XOR-MAPPED-ADDRESS attribute (which points at the NAT's public IP address and port -- just as if the STUN Binding Request had been sent to a STUN server on the public Internet) and an XOR-INTERNAL-ADDRESS attribute (which points to the source IP address and UDP port the packet STUN Binding Request packet had prior to being NATted). See Figure 13 which depicts how this might be implemented in a NAT.

When receiving a STUN Binding Request the STUN controlled firewall will respond with a STUN Binding Response containing an XOR-MAPPED-ADDRESS attribute (which points at the public IP address and port) and an XOR-INTERNAL-ADDRESS attribute (which points to the source IP address of the interface and UDP port where the packet was received, i.e., the internal interface).

Server procedures for handling the BOOTNOUNCE and REFRESH-INTERVAL attributes can be found in [Section 7.5](#) and [Section 7.1](#).

STUN Binding Requests, which arrived from its public interface(s), MAY be handled as if the server is not listening on that port (e.g., return an ICMP error). This specification does not need them.

7. New Attributes

7.1. REFRESH-INTERVAL Attribute

In a STUN request, the REFRESH-INTERVAL attribute indicates the number of milliseconds that the client wants the NAT binding (or firewall pinhole) to be opened. This applies to all bindings that exist in that NAT from that same source IP address and same source UDP port (see also [Appendix B.2](#)). In a STUN response, the REFRESH-INTERVAL attribute indicates the number of milliseconds the STUN server (embedded in the NAT or firewall) will keep the bindings open.

REFRESH-INTERVAL is specified as an unsigned 32 bit integer, and represents an interval measured in milliseconds (thus the maximum value is approximately 50 days). This attribute can be present in Binding Requests and in Binding Responses.

7.2. XOR-INTERNAL-ADDRESS Attribute

This attribute MUST be present in a Binding Response and is necessary to allow a STUN client to perform the outside-in discovery technique, in order to discover all of the STUN Control-aware NATs along the path.

The format of the XOR-INTERNAL-ADDRESS attribute is:

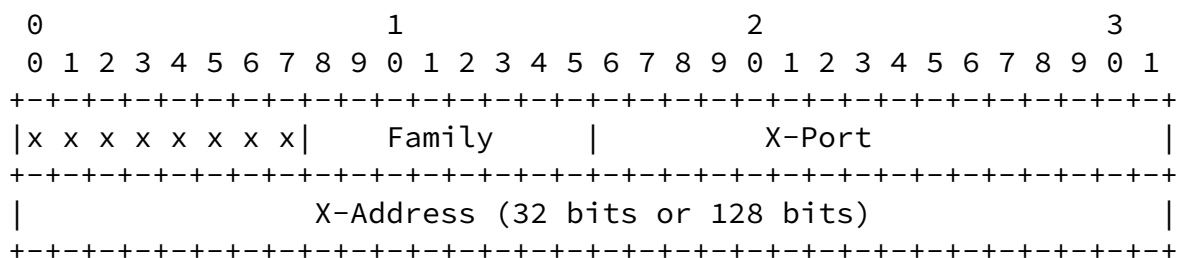


Figure 6: XOR-INTERNAL-ADDRESS Attribute

The meaning of Family, X-Port, and X-Address are exactly as in [\[I-D.ietf-behave-rfc3489bis\]](#). The length of X-Address depends on the address family (IPv4 or IPv6).

[7.3.](#) PLEASE-TAG Attribute

If a STUN client wants to discover on-path firewalls, it **MUST** include this attribute in its Binding Response when performing the Binding Discovery usage.

STUN servers are not expected to understand this attribute; if they return this attribute as an unknown attribute, it does not affect the operation described in this document.

The format of the PLEASE-TAG attribute is:

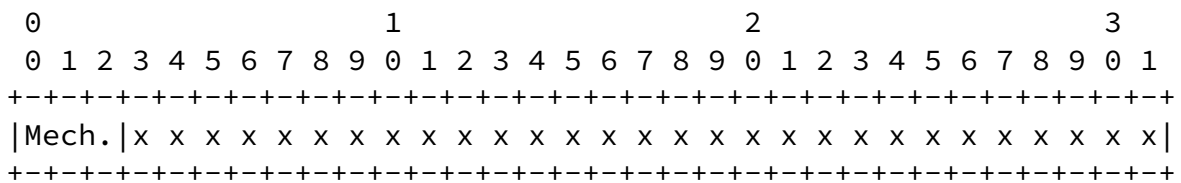


Figure 7: PLEASE-TAG Attribute

The 3-bit Mechanism field indicates the control mechanism desired. Currently, the only defined mechanism is STUN Control, and is indicated with all zeros. The intent of this field is to allow additional control mechanisms (e.g., UPnP IGD, NAT-PMP, MIDCOM).

[7.4.](#) TAG Attribute

The TAG attribute contains the XOR'd management transport address of the middlebox. Typically, a firewall as well as a NAT may find this technique useful as well.

If the associated STUN Request contained the PLEASE-TAG attribute, a middlebox **MUST** append this attribute as the last attribute of the STUN Response (with that same transaction-id). After appending this attribute, the STUN length field **MUST** be also be adjusted.

The format of the TAG attribute is:

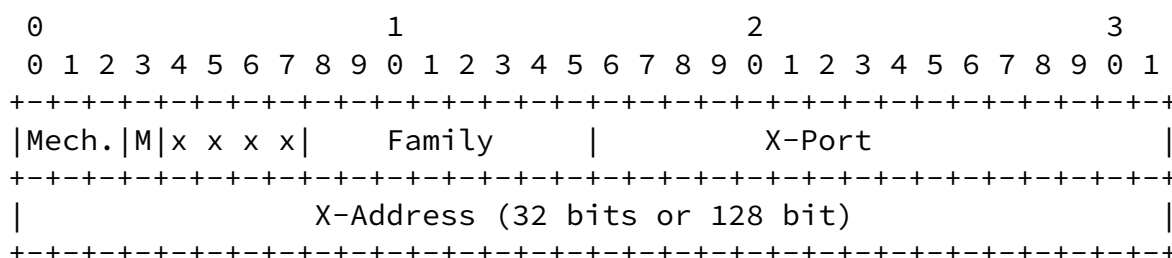


Figure 8: TAG Attribute

Mech: The 3-bit Mechanism field indicates the control mechanism supported on the described port. Currently, the only defined mechanism is STUN Control, and is indicated with 0x0. The intent of this field is to allow additional control mechanisms (e.g., UPnP IGD, NAT-PMP, MIDCOM).

The one-bit M field indicates if this firewall permits Mobility Header packets to flow through it ([\[RFC3775\]](#)).

The meaning of Family, X-Port, and X-Address are exactly as in [\[I-D.ietf-behave-rfc3489bis\]](#). The length of X-Address depends on the address family (IPv4 or IPv6).

[7.5.](#) BOOTNONCE Attribute

The BOOTNONCE attribute protects against the attack described in [Section 9.4](#).

Client procedures: The STUN client expects each NAT to return the same BOOTNONCE value each time that NAT is contacted. If a NAT

returns a different value, the STUN client MUST NOT use any information returned in the Binding Response and MUST re-run the STUN Control procedures from the beginning (i.e., obtain its public IP address from the STUN server on the Internet). This would only occur if an attack is in progress or if the NAT rebooted. If the NAT rebooted, it is good practice to re-run the STUN Control procedures anyway, as the network topology could be different as well.

Server procedures: This attribute's value is a hash of the STUN client's IP address and a value that is randomly-generated each time the NAT is initialized. The STUN client's IP address is included in this hash to thwart an attacker attaching to the NAT's internal network and learning the BOOTNONCE value.

The format of the BOOTNONCE attribute is:

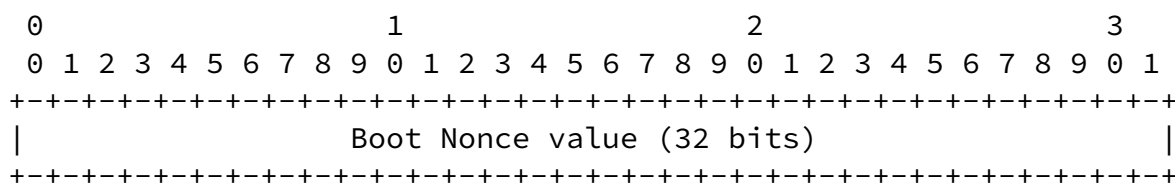
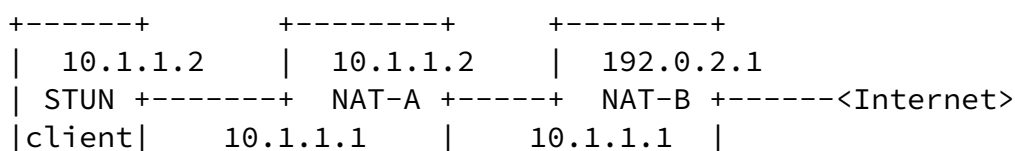


Figure 9: BOOTNONCE Attribute

8. Limitations of STUN Control

8.1. Overlapping IP Addresses with Nested NATs

If nested NATs have overlapping IP address space, there will be undetected NATs on the path. When this occurs, the STUN client will be unable to detect the presence of NAT-A if NAT-A assigns the same UDP port. For example, in the following figure, NAT-A and NAT-B are both using 10.1.1.x as their 'private' network.



+-----+ +-----+ +-----+

Figure 10: Overlapping Addresses with Nested NATs

When this situation occurs, the STUN client can only learn the outer-most address. This is not a problem -- the STUN client is still able to communicate with the outer-most NAT and is still able to avoid consuming access network bandwidth and avoid communicating with the public STUN server. All that is lost is the ability to optimize paths within the private network that has overlapped addresses.

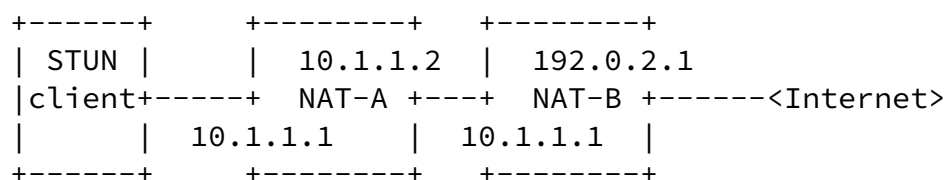
Of course when such an overlap occurs the end host (STUN client) cannot successfully establish bi-directional communication with hosts in the overlapped network, anyway.

8.2. Address Dependent NAT on Path

In order to utilize the mechanisms described in this document, a STUN Request is sent from the same source IP address and source port as the original STUN Binding Discovery message, but is sent to a different destination IP address -- it is sent to the IP address of an on-path NAT. If there is an on-path NAT, between the STUN client

and the STUN server, with 'address dependent' or 'address and port-dependent' mapping behavior (as described in [Section 4.1 of \[RFC4787\]](#)), that NAT will prevent a STUN client from taking advantage of the technique described in this document. When this occurs, the ports indicated by XOR-MAPPED-ADDRESS from the public STUN server and the NAT's embedded STUN server will differ.

An example of such a topology is shown in the following figure:



In this figure, NAT-A is a NAT that has address dependent mapping. Thus, when the STUN client sends a STUN Binding Request to 192.0.2.1 on UDP/3478, NAT-A will choose a new public UDP port for that communication. NAT-B will function normally, returning a different

port in its XOR-MAPPED-ADDRESS, which indicates to the STUN client that a symmetric NAT exists between the STUN client and the STUN server it just queried (NAT-B, in this example).

Figure 11: Address Dependant NAT on Path

[8.3.](#) Address Dependent Filtering

If there is an NAT along the path that has address dependent filtering (as described in [section 5 of \[RFC4787\]](#)), and the STUN client sends a STUN packet directly to any of the on-path NATs public addresses, the address-dependent filtering NAT will filter packets from the remote peer. Thus, after communicating with all of the on-path NATs the STUN client MUST send a UDP packet to the remote peer, if the remote peer is known.

[8.4.](#) Interacting with Legacy NATs

There will be cases where the STUN client attempts to communicate with an on-path NAT, which does not support STUN Control. There are two cases:

- o the NAT does not run a STUN server on its public interface (this will be the most common)
- o the NAT does run a STUN server on its public interface, but does not return the XOR-INTERNAL-ADDRESS attribute defined in this document

In both cases the optimizations described in this section will not be available to the STUN client. This is no worse than the condition today. This allows incremental upgrades of applications and NATs that implement the technique described in this document.

[9.](#) Security Considerations

This security considerations section will be expanded in a subsequent version of this document. So far, the authors have identified the following considerations:

[9.1.](#) Authorization

Only hosts that are 'inside' a NAT, which a NAT is already providing services for, can query or adjust the timeout of a NAT mapping.

A discussion of additional authorization mechanisms that might be needed for firewall traversal can be found at [\[I-D.wing-session-auth\]](#).

[9.2.](#) Resource Exhaustion

A malicious STUN client could ask for absurdly long NAT bindings (days) for many UDP sessions, which would exhaust the resources in the NAT. The same attack is possible (without considering this document and without considering STUN or other UNSAF [\[RFC3424\]](#) NAT traversal techniques) -- a malicious TCP (or UDP) client can open many TCP (or UDP) connections, and keep them open, causing resource exhaustion in the NAT.

[9.3.](#) Comparison to Other NAT Control Techniques

Like UPnP IGD, NAT-PMP, and host-initiated MIDCOM, the STUN usage described in this document allows a host to learn its public IP address and UDP port mapping, and to request a specific lifetime for mappings from that same source IP address and same source UDP port.

However, unlike other NAT traversal technologies, STUN Control described in this document only allows each UDP port on the host to create and adjust the mapping timeout of its own NAT mappings. Specifically, an application on a host can only adjust the duration of a NAT bindings for itself, and not for another application on that same host, and not for other hosts. This provides security advantages over other NAT control mechanisms where malicious software on a host can surreptitiously create NAT mappings to another application or to another host.

[9.4.](#) BOOTNOUNCE Attribute

Using the mechanisms described in this document, a STUN client learns the public IP addresses of its NAT which supports the mechanisms described in this document. However, without the STUN client's

knowledge, that NAT may acquire a new IP address (e.g., due to DHCP lease expiration or network renumbering). When this occurs, the STUN client will send a STUN Binding Request to the NAT's previous public IP address. If an attacker were to run a rogue STUN server on that address, the attacker will have effectively compromised the STUN server, as described in [Section 12.2.1 of \[RFC3489\]](#). The attacker, upon receiving STUN Binding Requests, will reply with STUN Binding Responses indicating an IP address the attacker controls. The attacker will thus have access to the subsequent flow established by the STUN client (e.g., RTP traffic). This attack is possible because the STUN client is unable to distinguish the attacker's replies from replies from the legitimate NAT.

To defend against this attack, the STUN server embedded in the NAT returns a BOOTNONSE value. The STUN client validates that it receives the same BOOTNONSE value in each STUN Binding Response from that NAT. If the STUN client receives a new BOOTNONSE value, the STUN client discards information about NATs it has learned through the procedures in this document, and restarts the procedure described in this document.

A weakness of this approach is that an attacker can learn the BOOTNONSE value if the attacker is able to connect to the NAT's internal network prior to initiating the attack. This is plausible if the internal network has no security (e.g., public WiFi network). For this reason, it is RECOMMENDED that the BOOTNONSE value is hashed with the STUN client's IP address. Doing so means that a successful attacker must acquire both the same IP address as the victim from behind the NAT (to learn the BOOTNONSE), and must also acquire the NAT's previous public IP address, or needs to be on-path between the victim and its NAT (in which case the attacker has no incentive to redirect traffic elsewhere to observe such traffic; however, the attacker might be interested in redirecting traffic towards another endpoint on the Internet. To thwart that attack, the STUN client MUST only honor STUN responses that have an X-MAPPED-ADDRESS that matches the public IP address of the NAT-embedded STUN server.

[10](#). Open Issues and Discussion Points

- o Discussion Point: After discovering NATs and firewalls, controlling those devices might also be done with a middlebox control protocol (e.g., by using standard or slightly modified

versions of SIMCO, UPnP IGD, MIDCOM, or NAT-PMP). This is open for discussion as this document is scoped within the IETF.

- o Discussion Point: Tagging would also be useful for the Connectivity Check usage (which is used by ICE), especially considering that a different firewall may be traversed for media than for the initial Binding Discovery usage. In such a situation, the new on-path firewall's policy might not allow a binding request to leave the network or allow a binding response to return. In this case, the firewall would need to indicate its presence to the STUN client in another way. An ICMP error message may be appropriate, and an ICMP extension [[RFC4884](#)] could indicate the firewall is controllable.
- o Open issue: We could resolve the problem of address dependant NATs along the path by introducing a new STUN attribute which indicates the UDP port the STUN client wants to control. However, this changes the security properties of STUN Control, so this seems undesirable.

Open issue: When the STUN client detects an address dependant NAT, should we recommend it abandon the STUN Control usage, and revert to operation as if it doesn't support the STUN Control usage?

- o Open issue: How many filter entries are in address dependent filtering NATs? If only one, this does become a real limitation if NATs are nested; if they're not nested, the outer-most NAT can avoid overwriting its own address in its address dependent filter.
- o Discussion: One way to thwart a resource consumption attack is to challenge the STUN client. This would allow the STUN server to delay the establishment of resources before a return-routability test is performed. This functionality is currently not provided by this specification. The NONCE attribute [[I-D.ietf-behave-rfc3489bis](#)] could be useful to provide this function. However, the mere sending of a UDP packet across a NAT creates a binding (for ~2 minutes), and there isn't a return-routability check for that.
- o The inside-out discovery technique was removed with version -03 of this document. The procedure worked as follows: The STUN client sends a STUN request to UDP/3478 of the IP address of its default router. If there is a STUN server listening there, it will respond, and will indicate its default route via the new DEFAULT-ROUTE attribute. With that information, the STUN client can discover the next-outermost NAT by repeating the procedure. More feedback is needed to determine whether the functionality is

Internet-Draft

STUN Control

October 2007

needed.

[11.](#) IANA Considerations

This section registers new STUN attributes per the procedures in [\[I-D.ietf-behave-rfc3489bis\]](#):

Mandatory range:

0x0029 XOR-INTERNAL-ADDRESS
0x00.. BOOTNONCE

Optional range:

0x8024 REFRESH-INTERVAL
0x80.. PLEASE-TAG
0x80.. TAG

[12.](#) Acknowledgements

Thanks to Remi Denis-Courmont, Christian Dickmann, Bajko Gabor, Markus Isomaki, Cullen Jennings, and Philip Matthews for their suggestions which have improved this document.

Thanks to Christian Dickmann and Yan Sun for their initial implementations of STUN Control.

[13.](#) References

[13.1.](#) Normative References

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

[I-D.ietf-behave-rfc3489bis]
Rosenberg, J., Huitema, C., Mahy, R., Matthews, P., and D. Wing, "Session Traversal Utilities for (NAT) (STUN)", [draft-ietf-behave-rfc3489bis-11](#) (work in progress), October 2007.

- [RFC4787] Audet, F. and C. Jennings, "Network Address Translation (NAT) Behavioral Requirements for Unicast UDP", [BCP 127](#), [RFC 4787](#), January 2007.
- [RFC3489] Rosenberg, J., Weinberger, J., Huitema, C., and R. Mahy, "STUN - Simple Traversal of User Datagram Protocol (UDP)

Wing, et al.

Expires April 18, 2008

[Page 24]

Internet-Draft

STUN Control

October 2007

Through Network Address Translators (NATs)", [RFC 3489](#), March 2003.

- [RFC3775] Johnson, D., Perkins, C., and J. Arkko, "Mobility Support in IPv6", [RFC 3775](#), June 2004.

[13.2.](#) Informational References

[I-D.ietf-behave-turn]

Rosenberg, J., "Traversal Using Relays around NAT (TURN): Relay Extensions to Session Traversal Utilities for NAT (STUN)", [draft-ietf-behave-turn-04](#) (work in progress), July 2007.

[UPnP-IGD]

UPnP Forum, "Universal Plug and Play (UPnP) Internet Gateway Device (IGD)", November 2001, <<http://www.upnp.org/standardizeddcps/igd.asp>>.

[Vista-cert]

Microsoft, "Windows Logo Program Device Requirements", 2006, <http://download.microsoft.com/download/d/e/1/de1e0c8f-a222-47bc-b78b-1656d4cf3cf7/WLP-DeviceReqs_309.pdf>.

[I-D.cheshire-nat-pmp]

Cheshire, S., "NAT Port Mapping Protocol (NAT-PMP)", [draft-cheshire-nat-pmp-02](#) (work in progress), October 2006.

- [RFC3303] Srisuresh, P., Kuthan, J., Rosenberg, J., Molitor, A., and A. Rayhan, "Middlebox communication architecture and framework", [RFC 3303](#), August 2002.

[I-D.ietf-mmusic-ice]

Rosenberg, J., "Interactive Connectivity Establishment (ICE): A Protocol for Network Address Translator (NAT) Traversal for Offer/Answer Protocols", [draft-ietf-mmusic-ice-18](#) (work in progress), September 2007.

[I-D.ietf-sip-outbound]
Jennings, C. and R. Mahy, "Managing Client Initiated Connections in the Session Initiation Protocol (SIP)", [draft-ietf-sip-outbound-10](#) (work in progress), July 2007.

[I-D.ietf-nsis-nslp-natfw]
Stiemerling, M., "NAT/Firewall NSIS Signaling Layer

Wing, et al. Expires April 18, 2008 [Page 25]

Internet-Draft STUN Control October 2007

Protocol (NSLP)", [draft-ietf-nsis-nslp-natfw-15](#) (work in progress), July 2007.

[RFC4884] Bonica, R., Gan, D., Tappan, D., and C. Pignataro, "Extended ICMP to Support Multi-Part Messages", [RFC 4884](#), April 2007.

[I-D.shore-tist-prot]
Shore, M., "The TIST (Topology-Insensitive Service Traversal) Protocol", [draft-shore-tist-prot-00](#) (work in progress), May 2002.

[I-D.shore-nls-tl]
Shore, M., "Network-Layer Signaling: Transport Layer", [draft-shore-nls-tl-05](#) (work in progress), June 2007.

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

[RFC3948] Huttunen, A., Swander, B., Volpe, V., DiBurro, L., and M. Stenberg, "UDP Encapsulation of IPsec ESP Packets", [RFC 3948](#), January 2005.

[I-D.wing-session-auth]
Wing, D., "Media Session Authorization", [draft-wing-session-auth-00](#) (work in progress), February 2006.

- [RFC3947] Kivinen, T., Swander, B., Huttunen, A., and V. Volpe, "Negotiation of NAT-Traversal in the IKE", [RFC 3947](#), January 2005.
- [RFC4306] Kaufman, C., "Internet Key Exchange (IKEv2) Protocol", [RFC 4306](#), December 2005.
- [RFC4380] Huitema, C., "Teredo: Tunneling IPv6 over UDP through Network Address Translations (NATs)", [RFC 4380](#), February 2006.

[Appendix A](#). Changes

[A.1](#). Changes in -05

- o Teredo is another mechanism to learn outer-most NAT, and Teredo also benefits from STUN Control with reduced frequency of keepalives.

Wing, et al.

Expires April 18, 2008

[Page 26]

Internet-Draft

STUN Control

October 2007

- o Provided more detail in how IKE/IPsec-over-UDP would operate with STUN Control.

[A.2](#). Changes in -04

- o Clarified that all existing bindings, for that source IP address and UDP port, are controlled with STUN Control.
- o Introduction now concentrates on the primary purpose of STUN Control, namely reducing keepalive traffic for SIP-Outbound.

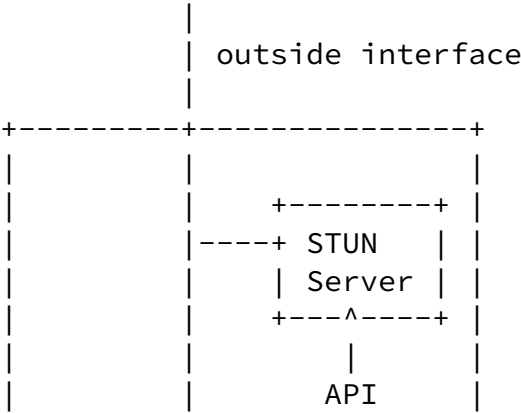
[A.3](#). Changes in -03

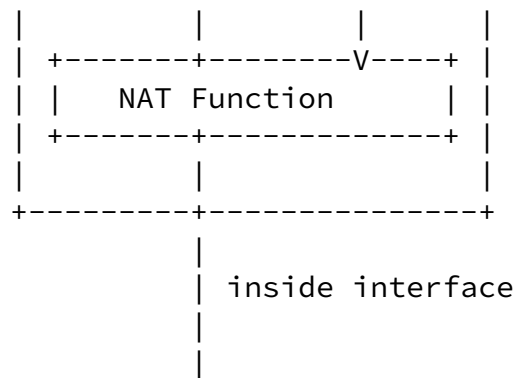
- o Removed TLS from normal STUN operation (as few use it, and ICE makes it unnecessary anyway)
- o BOOTNOUNCE attribute replaces STUN Control's previous use of TLS.
- o Added "MIP-capable" bit to TAG attribute
- o Removed "inside-out" discovery technique.

[Appendix B](#). Implementation Details

[B.1](#). Internal NAT Operation

Internally, the NAT can be diagrammed to function like this, where the NAT operation occurs before the STUN server:





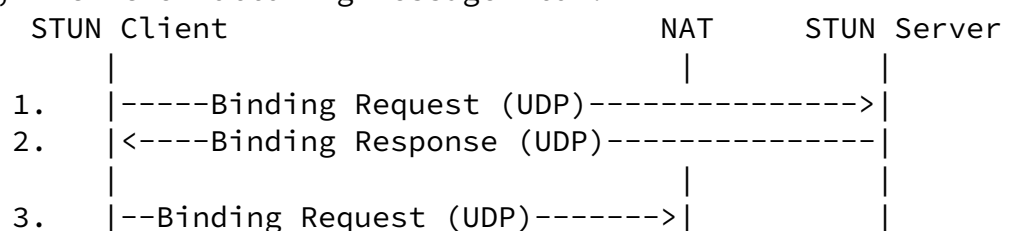
The host on the 'inside' interface of the NAT sends packets to the NAT's public interface, where the STUN server is listening. This STUN server returns the same public IP address (XOR-MAPPED-ADDRESS) as a STUN server that resides on a separate server on the 'outside' interface. In order to query and to control the NAT binding lifetimes, the STUN server uses an API with the NAT function.

Figure 13: Block Diagram of Internal NAT Operation

B.2. Linux specifics

The Linux NAT implementation maintains a separate connection table entry for every binding. When STUN Control is used to control the binding lifetime (e.g., extend the lifetime), the binding lifetime for each of those connection table entries is modified to the new value.

For example, with the following message flow:



4. |<-Binding Response (UDP)-----| |

the following two connection table entries are created:

```
udp      17 24 src=10.7.2.4 dst=10.7.1.2 sport=1024
         dport=3478 packets=1 bytes=64 src=10.7.1.2
         dst=10.7.1.3 sport=3478 dport=1024 packets=1
         bytes=84 mark=0 use=1
udp      17 25 src=10.7.2.4 dst=10.7.1.3 sport=1024
         dport=3478 packets=2 bytes=64 src=10.7.1.3
         dst=10.7.2.4 sport=3478 dport=1024 packets=2
         bytes=208 mark=0 use=1
```

the first src/dst/sport/dport combination is the internal and the second one is the external version. Both are equal in the second connection, as the NAT function wasn't active for the "internal" message.

s

Authors' Addresses

Dan Wing
Cisco Systems, Inc.
170 West Tasman Drive
San Jose, CA 95134
USA

Email: dwing@cisco.com

Jonathan Rosenberg
Cisco Systems, Inc.
Edison, NJ 07054
USA

Email: jdrosen@cisco.com

Hannes Tschofenig
Nokia Siemens Networks
Otto-Hahn-Ring 6
Munich, Bavaria 81739
Germany

Email: Hannes.Tschofenig@nsn.com
URI: <http://www.tschofenig.com>

Internet-Draft

STUN Control

October 2007

Full Copyright Statement

Copyright (C) The IETF Trust (2007).

This document is subject to the rights, licenses and restrictions contained in [BCP 78](#), and except as set forth therein, the authors retain all their rights.

This document and the information contained herein are provided on an "AS IS" basis and THE CONTRIBUTOR, THE ORGANIZATION HE/SHE REPRESENTS OR IS SPONSORED BY (IF ANY), THE INTERNET SOCIETY, THE IETF TRUST AND THE INTERNET ENGINEERING TASK FORCE DISCLAIM ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO ANY WARRANTY THAT THE USE OF THE INFORMATION HEREIN WILL NOT INFRINGE ANY RIGHTS OR ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

Intellectual Property

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

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

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

Acknowledgment

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

Wing, et al.

Expires April 18, 2008

[Page 31]