BEHAVE
Internet-Draft
Intended status: Standards Track
Expires: December 2, 2007

D. Wing
J. Rosenberg
Cisco Systems

May 31, 2007

Discovering, Querying, and Controlling Firewalls and NATs using STUN
draft-wing-behave-nat-control-stun-usage-02

Status of this Memo

Copyright Notice

Abstract

Simple Traversal Underneath NAT (STUN) is a mechanism for traversing NATs. STUN requests are transmitted through a NAT to external STUN servers. While this works very well, its two primary drawbacks are the inability to modify the properties of a NAT binding and the need to query a public STUN server for every new NAT binding (e.g., every phone call). These drawbacks require frequent messages which present a load on servers (like SIP servers and STUN servers) and are bad for low speed access networks, such as cellular access.

This document describes several mechanisms to discover NATs and firewalls and a method to query and control them. With these changes, binding discovery and keepalive traffic can be reduced to involve only the necessary NATs or firewalls. At the same time, backwards compatibility with NATs and firewalls that do not support this document is retrained.

This document is discussed on the BEHAVE mailing list, <https://www1.ietf.org/mailman/listinfo/behave>, in anticipation of a BoF at IETF69 in Chicago.

Table of Contents

1. Introduction

Two common usages of 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 [UPnP], MIDCOM [RFC3303], Bonjour [Bonjour]), STUN does not interact directly with the NAT. Because STUN doesn't interact directly with the NAT, STUN cannot request additional services from the NAT such as longer lifetimes (which would reduce keepalive messages), and each new NAT binding (e.g., each phone call) requires communicating with the STUN server on the Internet.

This paper 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. Motivation

There are a number of problems with existing NAT traversal techniques such as STUN [RFC3489], [UPnP], and [Bonjour]):

nested NATs:
: Today, many ISPs provide their subscribers with modems that have embedded NATs, often with only one physical Ethernet port. These subscribers then install NATs behind those devices to provide additional features, such as wireless access. Another example is a NAT in the basement of an apartment building or a campus dormitory, which combined with a NAT within the home or dormitory room also result in nested NATs. In both of these situations, UPnP and Bonjour no longer function at all, 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 if it obtains an RFC1918 address on its WAN interface. This merely sidesteps the problem. This technique is also ineffective if the ISP is NATting its subscribers and the ISP restricts each subscriber to one IP address.

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

chattiness:
: To perform its binding discovery, a STUN client communicates 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). STUN's chattiness is often cited as a reason to use other NAT traversal techniques such as UPnP or Bonjour.

The technique described in this paper provides a significant reduction in STUN's chattiness, to the point that the only time a

STUN client needs to communicate with the STUN server on the public Internet is when the STUN client is initialized.

incremental deployment:
Many other NAT traversal techniques require the endpoint and its NAT to both support the new feature or else NAT traversal isn't possible at all.

The technique described in this paper allows incremental deployment of local endpoints and their NATs that support STUN Control. If the local endpoint, or its NATs, don't support the STUN Control functionality, normal STUN and ICE [I-D.ietf-mmusic-ice] procedures are used to traverse the NATs without the optimizations described in this paper.

3. Conventions Used in this Document

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].

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 three techniques to find NATs or firewalls. The first technique uses STUN to find the outer-most NAT and works itself towards the host. The second technique requests on-path firewalls or NATs to append their IP address to a STUN packet. The third technique sends a STUN packet to the default router (which, in a small network, is often your NAT). This is described in Section 5.

Querying Discovered Middleboxes:
After discovering a NAT or a firewall, it's 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.

Discussion Point: After discovering NATs and firewalls, querying those devices might also be done with a middlebox control protocol (e.g., by using standard or slightly modified versions of SIMCO [RFC4540], UPnP, MIDCOM, or Bonjour). This is open for discussion as this document is scoped within the IETF.

Controlling Discovered Middleboxes

A NAT or 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]). This is described in Section 6.

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, MIDCOM, or Bonjour). This is open for discussion as this document is scoped within the IETF.

5. Discovery of Middleboxes

There are three techniques to discover a middlebox (a NAT or a firewall): outside-in, inside-out (useful when the outer-most NAT device doesn't support STUN Control), or by tagging (useful for firewalls).

These techniques can be combined in useful ways. For example, if your inner-most NAT supports STUN Control, and your public STUN server returns the same IP address and port as your inner-most NAT, you know you don't have a NAT between your inner-most NAT and the STUN server. Otherwise, you know there is a NAT between your inner-most NAT and the STUN server (e.g., an ISP-supplied device or whoever is providing your Internet service). Knowing this allows optimizing NAT keepalives.

5.1. Outside-In

When a STUN client sends a STUN Request to a STUN server, it receives a STUN Response which 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, and knows the 'public' IP address of that NAT. For example, in the following diagram, the STUN client learns the public IP address of its NAT is 192.0.2.1:

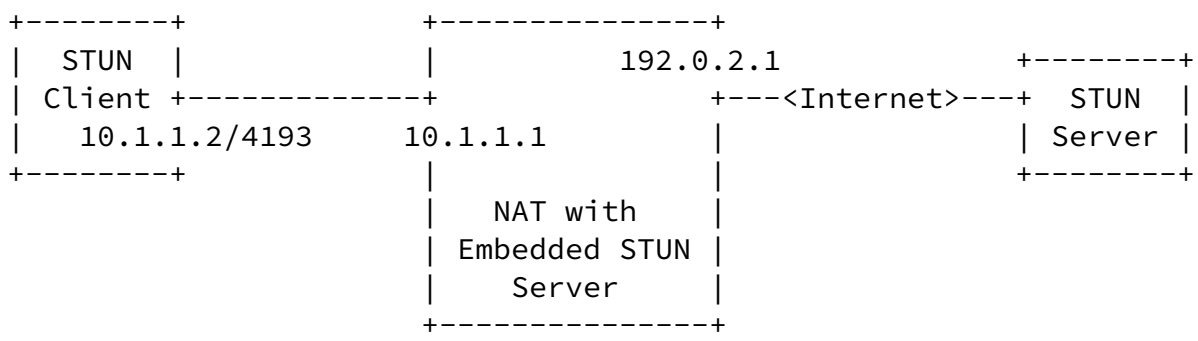


Figure 1: One NAT with embedded STUN server

Internally, the NAT can be diagrammed to function like this, where

the NAT operation occurs before the STUN server:

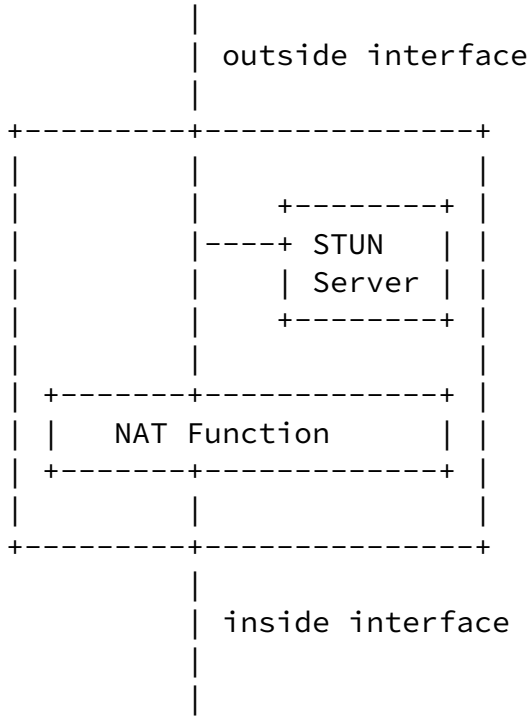


Figure 2: Block Diagram of Internal NAT Operation

After learning the public IP address of its outer-most NAT, the STUN client attempts to communicate with the STUN server embedded in that outer-most NAT. The STUN client does this by first obtaining a shared secret, over a TLS connection, to the NAT's public IP address (192.0.2.1 in the figure above). After obtaining a shared secret, it sends a STUN Binding Request to the NAT's public IP address. The NAT will return a STUN Binding Response message including the XOR-INTERNAL-ADDRESS attribute, which will indicate the IP address and UDP port seen on the *internal* side of the NAT for that translation. In the example above, the IP address and UDP port indicated in XOR-INTERNAL-ADDRESS are the same as that used by the STUN client (10.1.1.2/4193), which indicates there are no other NATs between the STUN client and that outer-most NAT.

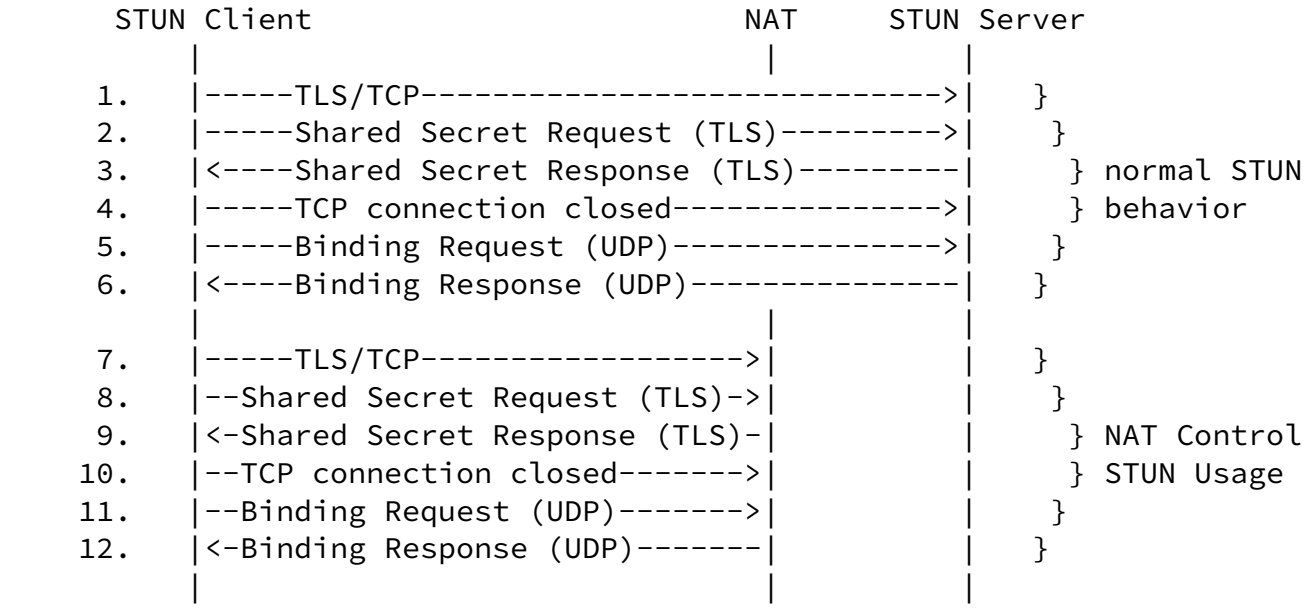


Figure 3: Communication Flow

In the call flow above, steps 1-6 are normal STUN behavior [I-D.ietf-behave-rfc3489bis]:

1: STUN client initiates a TLS-over-TCP connection to its STUN server on the Internet.

2: Using that connection, the STUN client sends Shared Secret Request to that STUN server.

3: Using that connection, the STUN server sends Shared Secret Response. This contains the STUN username the client should use for subsequent queries to this STUN server, and the STUN password that will be used to integrity-protect subsequent STUN messages with this STUN server.

4: TCP connection is closed.

5: STUN client sends UDP Binding Request to its STUN server on the Internet, using the STUN username obtained from that STUN server (from step 3).

6: STUN server responds with UDP Binding Response, integrity protected with the STUN password (from step 3). The STUN client now knows the public IP address of its outer-most NAT. This is used in the next step.

The next steps are the additional steps performed by a STUN client

that has implemented the NAT Control STUN Usage:

7: STUN client initiates a TLS-over-TCP connection to the STUN server embedded in its outer-most NAT.

8: Using that connection, the STUN client sends Shared Secret Request to that STUN server.

9: Using that connection, the STUN server sends Shared Secret Response. This contains the STUN username the client should use for subsequent queries to this STUN server, and the STUN password that will be used to integrity-protect subsequent STUN messages with this STUN server.

10: TCP connection is closed.

11: STUN client sends UDP Binding Request to the STUN server embedded in its outer-most NAT, using the STUN username obtained from that STUN server (from step 10).

12: STUN server responds with UDP Binding Response, integrity protected with the STUN password (from step 10).

The response obtained in the message 12 contains the XOR-MAPPED-ADDRESS attribute which will have the same value as when the STUN server on the Internet responded (in step 6). The STUN client can perform steps 11-12 for any new UDP communication (e.g., for every new phone call), without needing to repeat steps 1-10. This meets the desire to reduce chattiness.

The response obtained in message 12 will also contain the XOR-INTERNAL-ADDRESS, which allows the STUN client to repeat steps 7-12 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 Section 5.1.1. This functionality meets the need 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, as described in Section 6.1. The STUN client receives positive confirmation that the binding lifetime has been extended,

allowing the STUN client to significantly reduces its NAT keepalive traffic. Additionally, as long as the NAT complies with [RFC4787], 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.

The following diagram shows how a STUN client iterates over the NATs to communicate with all of the NATs in the path. First, the STUN client would learn the outer-most NAT's IP address by performing the steps above. In the case below, 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's the IP address and UDP port on the *outer* side 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 knows there are no other NATs between itself and NAT-B, and finishes.

The following figure shows two nested NATs:

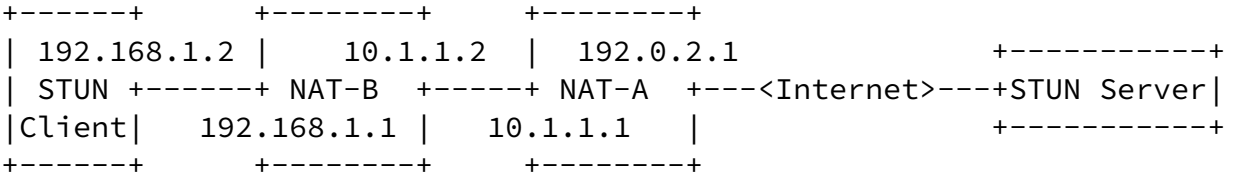


Figure 4: Two nested NATs with embedded STUN servers

The message flow with two nested NATs is shown below:

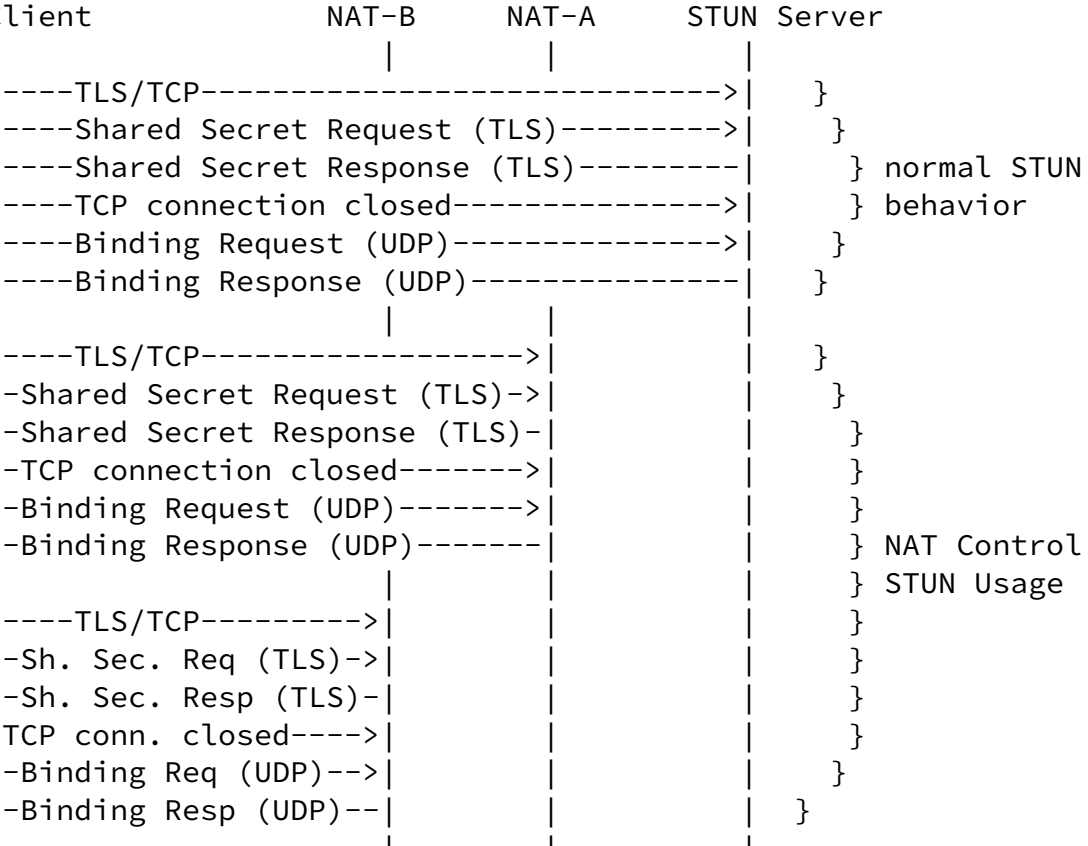


Figure 5: Message Flow for Outside-In with Two NATs

Once a shared secret has been obtained with each of the on-path NATs, the STUN client no longer needs the TLS/TCP connection -- all subsequent bindings for individual UDP streams (that is, for each subsequent call) are obtained by just sending a Binding Request to each of the NATs. By sending a Binding Request to both NAT-A and NAT-B, the STUN client has the opportunity to optimize the packet flow if their UDP peer is also behind the same NAT.

5.1.2. XOR-INTERNAL-ADDRESS Attribute

This attribute MUST be present in a Binding Response and may be used in other responses as well. This attribute is necessary to allow a STUN client to 'walk backwards' and communicate directly with all of the STUN-aware NATs along the path.

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

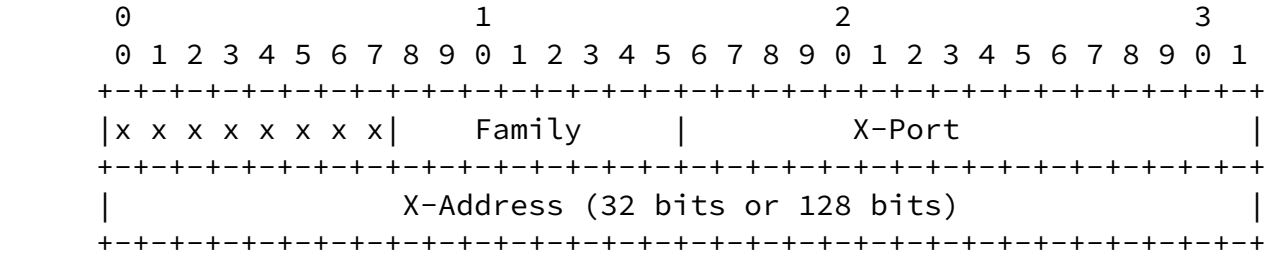


Figure 6: XOR-INTERNAL-ADDRESS Attribute

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).

5.1.3. Interacting with Legacy NATs

There will be cases where the STUN client attempts to communicate with an on-path NAT which does not support the outside-in usage described in Section 5.1. There are two cases:

- the NAT does not run a STUN server on its public interface (this will be the most common)

- the NAT does run a STUN server on its public interface, but doesn't return the XOR-INTERNAL-ADDRESS attribute defined in this document

In both cases the optimizations described in this section won't 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. In such a situation, the STUN client might implement the inside-out technique, described in Section 5.2.

5.2. Inside-Out

[[Discussion Point: This is being included as a discussion item. Traditional traceroute provides similar functionality, and in many cases traceroute survives traversing over a NAT that doesn't support STUN Control. However, traceroute has significant disadvantages (induces a load on intermediate devices to return ICMP error messages, and those ICMP messages are routinely or inadvertently filtered). Unlike the Inside-Out technique described below, traceroute doesn't rely on the default route.]]

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.

5.2.1. DEFAULT-ROUTE Attribute

The DEFAULT-ROUTE attribute contains the XOR'd default route, which is useful for finding the next-outer device.

The format of the DEFAULT-ROUTE attribute is:

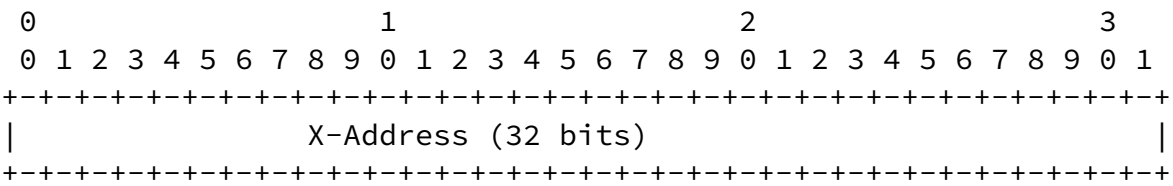


Figure 7: DEFAULT-ROUTE Attribute

The meaning of X-Address is exactly as in [I-D.ietf-behave-rfc3489bis].

5.3. Tagging

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 normally put their IP address into packets, so a different technique is needed to identify firewalls.

To discover an on-path firewall, the PLEASE-TAG attribute is used with a normal STUN Binding Request usage. A firewall sees the normal Binding Request usage (a STUN packet sent to UDP/3478) with the 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 pointer to each of the on-path firewalls between the client and that STUN server.

> Note: Tagging is similar to how NSIS [I-D.ietf-nsis-nslp-natfw], TIST [I-D.shore-tist-prot], and NLS [I-D.shore-nls-tl] function.
>
> 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.

This figure shows how tagging functions.

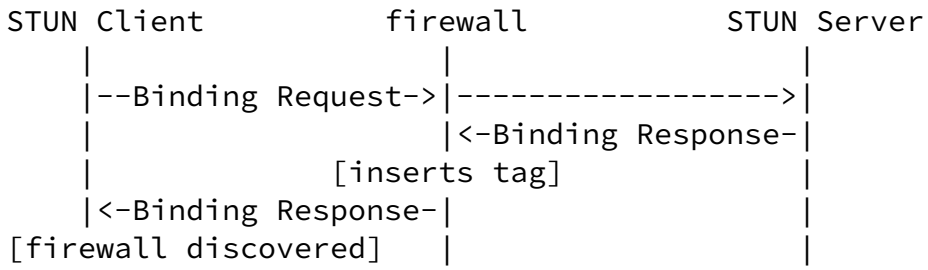


Figure 8: Tagging Message Flow

1. Binding Request, containing PLEASE-TAG attribute, is sent to the IP address of the STUN server. This is seen by the firewall, and the firewall remembers the STUN transaction id, and permits the STUN Binding Request packet.

2. The STUN Binding Response packet is seen by the firewall.

3. The firewall inserts the TAG attribute, which contains the firewall's management address.

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

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

5.3.1. 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:

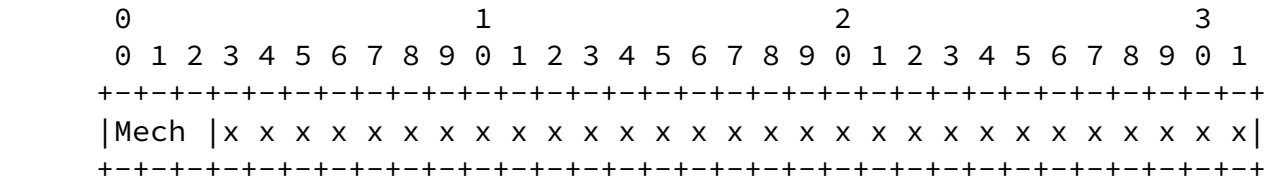


Figure 9: PLEASE-TAG Attribute

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, Bonjour, MIDCOM).

5.3.2. TAG Attribute

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

A middlebox MUST append this attribute as the last attribute of a STUN response, and only if the associated STUN request (with the same transaction-id) contained the PLEASE-TAG attribute.

The format of the TAG attribute is:

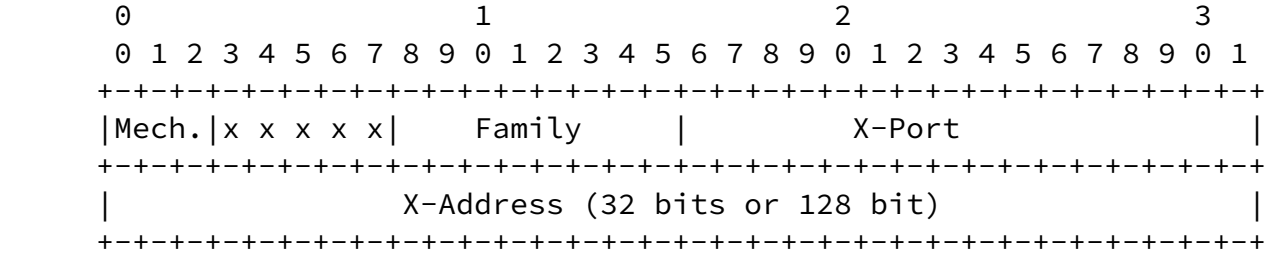


Figure 10: TAG Attribute

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, Bonjour, MIDCOM).

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).

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. 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. In a STUN response, the same attribute indicates the length of time of that NAT binding (or firewall pinhole).

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. In a request, the value 0xFFFF means it's a query and the refresh interval isn't actually changed.

6.2. Client Procedures

After discovering on-path NATs and firewalls, the STUN client begins querying and controlling those devices. The STUN client first performs the Shared Secret Usage (as described in [I-D.ietf-behave-rfc3489bis]) with the NAT or firewall it discovered. After performing that usage, the STUN client now has a STUN USERNAME and PASSWORD. The username and password are used, thereafter, for all subsequent messages between the STUN client and this NAT's STUN server. This procedure might be done, for example, when the STUN client first initializes such as upon bootup or initialization of the application.

If subsequent messages from that STUN server fail authentication, the STUN client MUST re-obtain its IP address from a public STUN server, not from its outer-most NAT (see section Section 9.3).

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 in its outer-most NAT's

STUN UDP port (3478). The NAT's STUN server 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).

6.3. Server Procedures

The server should listen for STUN Shared Secret Requests and STUN Binding Requests on the STUN UDP and TCP ports (UDP/3478, TCP/3478) on its public IP address(es) and its private IP address(es), and accept such STUN packets from hosts connected to its private interface(s). STUN Binding Requests which arrived from its public interface(s) MAY be handled as if the server isn't listening on that port (e.g., return an ICMP error) -- this specification does not need them.

After receiving a STUN Shared Secret Request, the NAT follows the procedures described in the Short-Term Usage section of [I-D.ietf-behave-rfc3489bis]. The embedded STUN server MUST store that username and password so long as any NAT bindings, created or adjusted with that same STUN username, have active mappings on the NAT, and for 60 seconds thereafter (to allow the STUN client to obtain a new binding).

After receiving a STUN Binding Request containing the REFRESH-INTERVAL attribute, the server SHOULD authenticate the request using the USERNAME attribute and the previously-shared STUN password (this is to defend against resource starvation attacks, see Section 9.1). If authenticated, the new binding's lifetime can be maximized against the NAT's configured sanity check and the lifetime indicated in the REFRESH-INTERVAL attribute of the STUN Binding Response.

In addition to its other attributes, the Binding Response MUST contain the XOR-MAPPED-ADDRESS and XOR-INTERNAL-ADDRESS attributes. The XOR-MAPPED-ADDRESS contains the public IP address and UDP port for this binding, which is shared with the intended peer. The XOR-INTERNAL-ADDRESS contains the IP address and UDP port of the STUN Binding Request prior to the NAT translation, which is used by the

STUN client to walk backwards through nested NATs (Section 5.1)

For example, looking at Figure 1, the XOR-INTERNAL-ADDRESS is the IP address and UDP port prior to the NAPT operation. If there is only one NAT, as shown in Figure 1, XOR-INTERNAL-ADDRESS would contain the STUN client's own IP address and UDP port. If there are multiple NATs, XOR-INTERNAL-ADDRESS would indicate the IP address of the next NAT (that is, the next NAT closer to the STUN client). Iterating over this procedure allows the STUN client to find all of the NATs along the path.

7. Benefits

7.1. 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's simple because only the flow being used can be controlled (e.g., have its NAT timeout queried or extended). Other flows cannot be created, queried, or controlled via STUN Control.

7.2. Incremental Deployment

NAT Control can be incrementally deployed. If the outer-most NAT does not support it, the STUN client behaves as normal. In this case, the STUN client might benefit from attempting inside-out procedure described in Section 5.2, and still gain some optimizations. Where the outer-most STUN NAT does support it, 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 NAT 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.

7.3. Optimize SIP-Outbound

In sip-outbound [I-D.ietf-sip-outbound], the SIP proxy is also the STUN server. STUN Control as described in this document enables two optimizations of SIP-Outbound's keepalive mechanism:

1. 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, and;

2. all of the on-path NATs can explicitly indicate their timeouts, reducing the frequency of keepalive messages.

7.4. Optimize ICE

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

7.4.1. 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.

7.4.2. Keepalive

ICE uses STUN Indications as its primary media stream keepalive mechanism. This document enables two optimizations of ICE's keepalive technique:

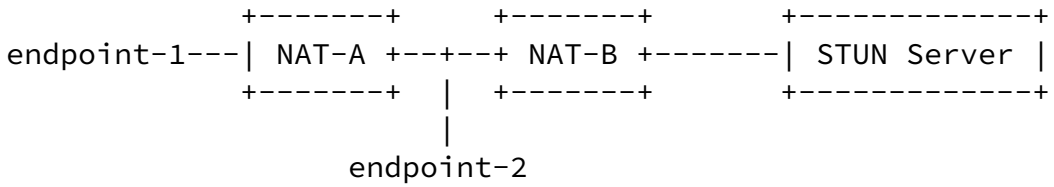
1. STUN keepalive messages need only be sent to the outer-most NAT, rather than across the access link to the remote peer, and;

2. all of the on-path NATs can explicitly indicate their timeouts, which allows reducing the keepalive frequency.

7.4.3. 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.

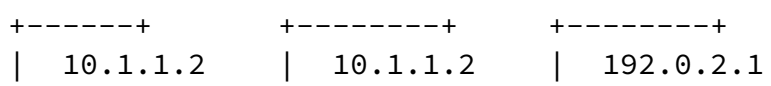
For example, if endpoint-1 is only configured with the IP address of the STUN server on the left, endpoint-1 can learn about NAT-B and NAT-A. Utilizing the STUN server in NAT-A, endpoint-1 and endpoint-2 can optimize their media path so they make the optimal path from endpoint-1 to NAT-A to endpoint-2:



8. Limitations

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.



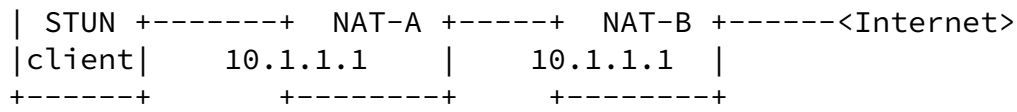


Figure 12: Overlapping Addresses with Nested NATs

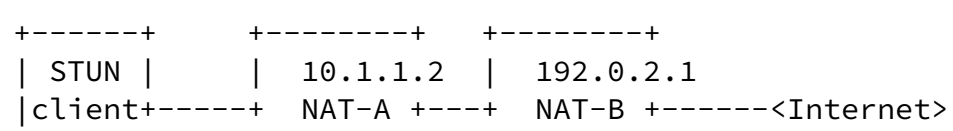
When this situation occurs, the STUN client can only learn the outer-most address. This isn't a problem -- the STUN client is still able to communicate with the outer-most NAT and is still able to avoid consuming access network bandwidth and avoid communicating with the public STUN server. All that is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

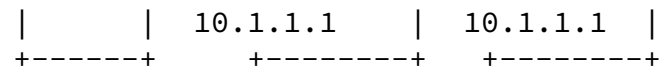
Of course when such an overlap occurs the end host (STUN client) cannot successfully establish bi-directional communication with hosts in the overlapped network, anyway.

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 13: Address Dependant NAT on Path

Open issue: We could resolve this problem by introducing a new STUN attribute which indicates the UDP port the STUN client wants to control. However, this changes the security properties of NAT Control, so this seems undesirable.

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

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.

Discussion: 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.

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 and Resource Exhaustion

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 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 client can open many TCP connections, and keep them open, causing resource exhaustion in the NAT. One way to thwart such an attack is to challenge the STUN client with a nonce, which is already part of the STUN specification. By doing this, a NAT can provide DoS protection similar to what it could do for TCP today.

9.2. Comparison to Other NAT Control Techniques

Like UPnP, Bonjour, 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 that mapping.

However, unlike those technologies, the NAT Control usage 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.

9.3. Rogue STUN Server

As described in Section 7, a STUN client can learn its outer-most NAT runs an embedded STUN server. However, without the STUN client's knowledge, the outer-most NAT may acquire a new IP address. This could occur when the NAT moves to a new mobile network or its DHCP lease expires. When the NAT acquires a new IP address, the STUN client will send a STUN Binding Request to the NAT's prior public IP address, which will be routed to the NAT's previous address.

If an attacker runs a rogue STUN server on that address, the attacker has effectively compromised the STUN server (the attacked described in section 12.2.1 of [RFC3489]). The attacker will send STUN Binding

Responses indicating his IP address, which will be indistinguishable, to the STUN client, from the behavior of the legitimate STUN server.

To defend against this attack, the STUN client and STUN server obtain a short-term password as described in section Section 6.2.

10. IANA Considerations

This section registers one new STUN attribute per the procedures in [I-D.ietf-behave-rfc3489bis]:

Mandatory range:
0x0028 XOR-INTERNAL-ADDRESS

Optional range:
0x80.. PLEASE-TAG
0x80.. TAG

11. Acknowledgements

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

12. References

12.1. 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., "Session Traversal Utilities for (NAT) (STUN)", draft-ietf-behave-rfc3489bis-06 (work in progress), March 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) Through Network Address Translators (NATs)", RFC 3489, March 2003.

12.2. Informational References

[I-D.ietf-behave-turn]
Rosenberg, J., "Obtaining Relay Addresses from Simple Traversal Underneath NAT (STUN)", draft-ietf-behave-turn-03 (work in progress), March 2007.

[UPnP] UPnP Forum, "Universal Plug and Play", 2000, <http://www.upnp.org>.

[Bonjour] Apple Computer, "Bonjour", 2005, <http://www.apple.com/macosx/features/bonjour/>.

[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 Methodology for Network Address Translator (NAT) Traversal for Offer/Answer Protocols", draft-ietf-mmusic-ice-15 (work in progress), March 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-08 (work in progress), March 2007.

[I-D.ietf-nsis-nslp-natfw]
Stiemerling, M., "NAT/Firewall NSIS Signaling Layer Protocol (NSLP)", draft-ietf-nsis-nslp-natfw-14 (work in progress), March 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-04 (work in progress), May 2007.

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

Translation", RFC 3424, November 2002.

[RFC4540] Stiemerling, M., Quittek, J., and C. Cadar, "NEC's Simple Middlebox Configuration (SIMCO) Protocol Version 3.0", RFC 4540, May 2006.

Authors' Addresses

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

Email: dwing@cisco.com

Jonathan Rosenberg
Cisco Systems
600 Lanidex Plaza
Parsippany, NJ 07054
USA

Email: jdrosen@cisco.com

Full Copyright Statement

Intellectual Property

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).