Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: November 14, 2015

B. Schwartz
J. Uberti
Google

May 13, 2015

Recursively Encapsulated TURN (RETURN) for Connectivity and Privacy in WebRTC
draft-ietf-rtcweb-return-00

Abstract

In the context of WebRTC, the concept of a local TURN proxy has been suggested, but not reviewed in detail. WebRTC applications are already using TURN to enhance connectivity and privacy. This document explains how local TURN proxies and WebRTC applications can work together.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at http://datatracker.ietf.org/drafts/current/.

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."

This Internet-Draft will expire on November 14, 2015.

Copyright Notice

Table of Contents

1. Introduction

TURN [RFC5766] is a protocol for communication between a client and a TURN server, in order to route UDP traffic to and from one or more peers. As noted in [RFC5766], the TURN relay server "typically sits in the public Internet". In a WebRTC context, if a TURN server is to be used, it is typically provided by the application, either to provide connectivity between users whose NATs would otherwise prevent

it, or to obscure the identity of the participants by concealing their IP addresses from one another.

In many enterprises, direct UDP transmissions are not permitted between clients on the internal networks and external IP addresses, so media must flow over TCP. To enable WebRTC services in such a situation, clients must use TURN-TCP, or TURN-TLS. These configurations are not ideal: they send all traffic over TCP, which leads to higher latency than would otherwise be necessary, and they force the application provider to operate a TURN server because WebRTC endpoints behind NAT cannot typically act as TCP servers. These configurations may result in especially bad behaviors when operating through TCP or HTTP proxies that were not designed to carry real-time media streams.

To avoid forcing WebRTC media streams through a TCP stage, enterprise network operators may operate a TURN server for their network, which can be discovered by clients using TURN Auto-Discovery [I-D.ietf-tram-turn-server-discovery], or through a proprietary mechanism. This TURN server may be placed inside the network, with a firewall configuration allowing it to communicate with the public internet, or it may be operated by a third party outside the network, with a firewall configuration that allows hosts inside the network to communicate with it. Use of the specified TURN server may be the only way for clients on the network to achieve a high quality WebRTC experience. This scenario is required to be supported by the WebRTC requirements document [I-D.ietf-rtcweb-use-cases-and-requirements] Section 3.3.5.1.

When the application intends to use a TURN server for identity cloaking, and the enterprise network administrator intends to use a TURN server for connectivity, there is a conflict. In current WebRTC implementations, TURN can only be used on a single-hop basis in each candidate, but using only the enterprise’s TURN server reveals information about the user (e.g. organizational affiliation), and using only the application’s TURN server may be blocked by the network administrator, or may require using TURN-TCP or TURN-TLS, resulting in a significant sacrifice in latency.

To resolve this conflict, we introduce Recursively Encapsulated TURN, a procedure that allows a WebRTC endpoint to route traffic through multiple TURN servers, and get improved connectivity and privacy in return.

2. Visual Overview of RETURN

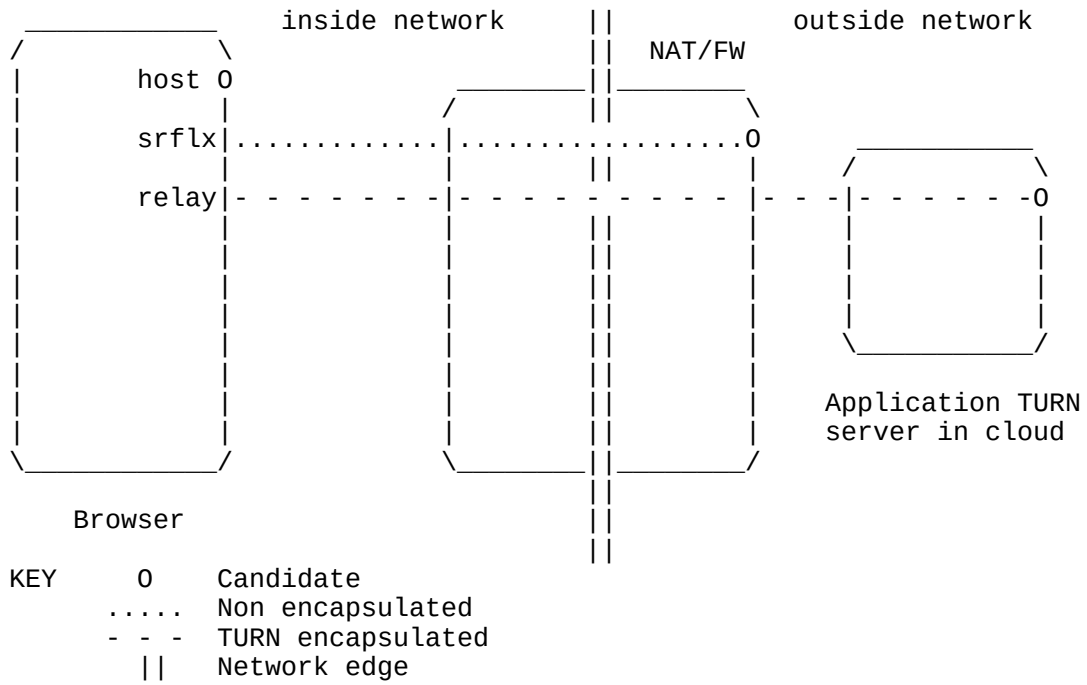


Figure 1: Basic WebRTC ICE Candidates with TURN Server

Figure 1 shows a browser located inside a home or enterprise network which connects to the Internet through a Network Address Translator and Firewall (NAT/FW). A TURN server in the Internet cloud is also shown, which is provided by the WebRTC application via the JavaScript IceServers object.

A WebRTC application can use a TURN server to provide NAT traversal, but also to provide privacy, routing optimizations, logging, or possibly other functionality. The application can accomplish this by forcing all traffic to flow through the TURN server using the JavaScript RTCIceTransportPolicy object [I-D.ietf-rtcweb-jsep]. Since this TURN server is injected by the application, we will refer to it as an Application TURN server.

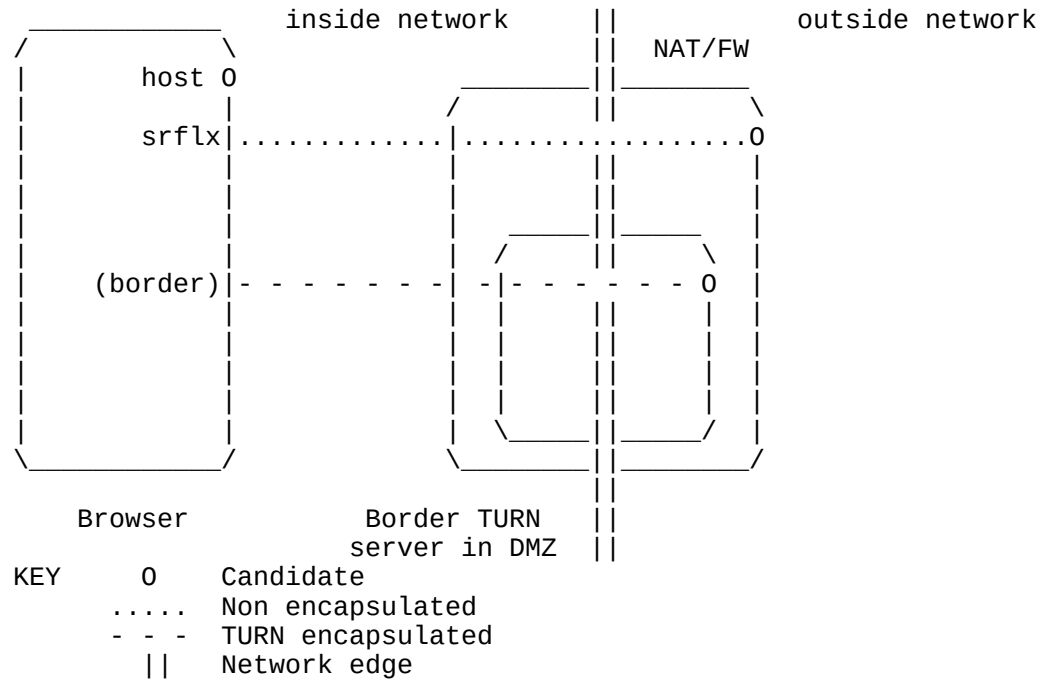


Figure 2: WebRTC ICE Candidates with DMZ TURN Server

Figure 2 shows a TURN server co-resident with the NAT/FW, i.e. in the DMZ of the FW. This TURN server might be used by an enterprise, ISP, or home network to enable WebRTC media flows that would otherwise be blocked by the firewall, or to improve quality of service on flows that pass through this TURN server. This TURN server is not part of a particular application, and is managed as part of the border control system, so we call it a Border TURN Server.

Figure 2 shows the port allocated on this TURN server as "(border)", not any particular candidate type, to distinguish it from the other ports, which have been represented as ICE candidates in accordance with the WebRTC specifications. This case is different, because unlike an Application TURN server, there is not yet any specification for how WebRTC should interact with a Border TURN server. Under what conditions should WebRTC allocate a port on a Border TURN server? How should WebRTC represent that port as an ICE candidate? This draft serves to answer these two questions.

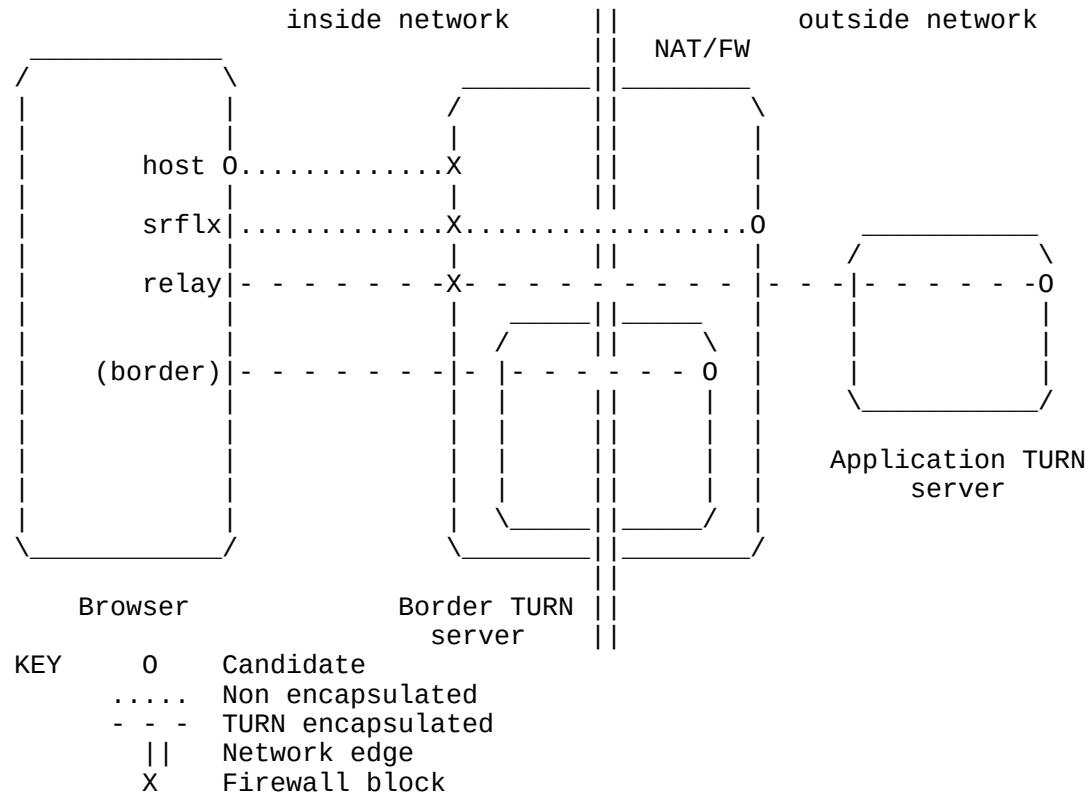


Figure 3: WebRTC ICE Candidates with Application and Border TURN Servers

In Figure 3, there is both an Application TURN server and a Border TURN server. The Firewall is blocking UDP traffic except for UDP traffic to/from the Border TURN server, so only the "(border)" port allocation will work. However, there is no specified way for WebRTC to use this port as a candidate. Moreover, this port on its own would not be sufficient to satisfy the user's needs. Both TURN servers provide important functionality, so we need a way for WebRTC to select a candidate that uses both TURN servers.

The solution proposed in this draft is for the browser to implement RETURN, which provides a candidate that traverses both TURN servers, as shown in Figure 4.

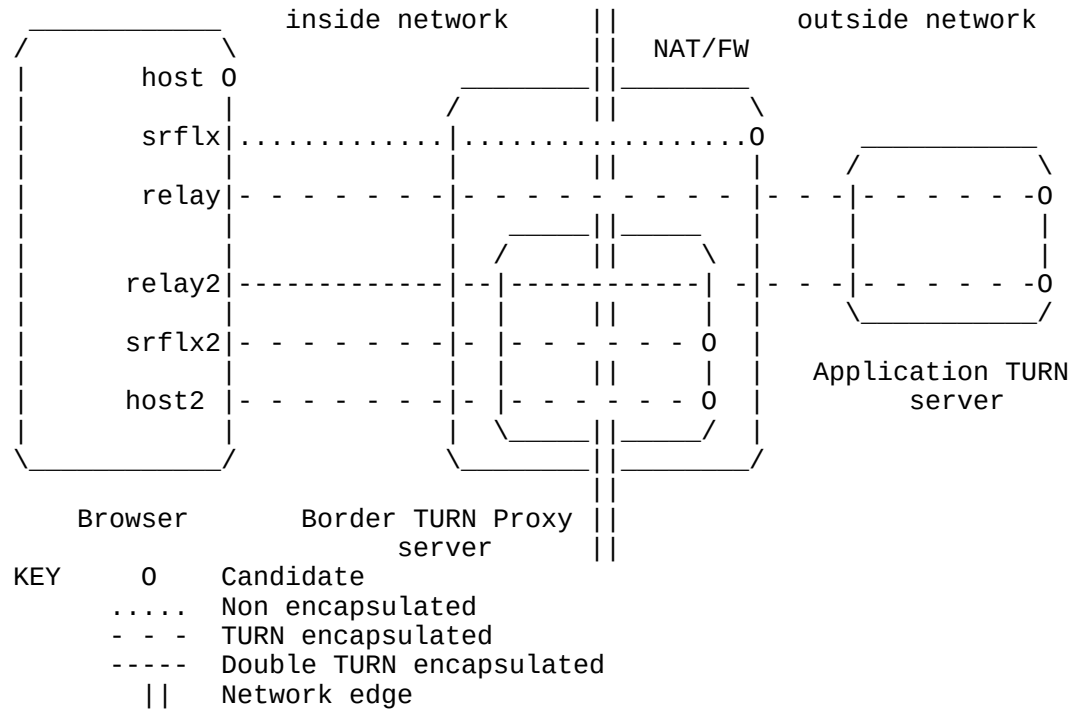


Figure 4: WebRTC ICE Candidates with Application TURN and Border TURN Proxy Servers

The Browser in Figure 4 implements RETURN, so it allocates a port on the Border TURN server, now referred to as a Border TURN Proxy by analogy to an HTTP CONNECT or SOCKS Proxy (see Figure 5), and then runs STUN and TURN over this allocation, resulting in three candidates: relay2, srflx2, and host2. The relay2 candidate causes traffic to flow through both TURN servers by encapsulating TURN within TURN - hence the name Recursively Encapsulated TURN (RETURN).

The host2 and srflx2 candidates are probably identical, so one will be dropped by ICE. If the NAT/FW blocks UDP and the application uses only relay candidates, then the relay2 candidate will be selected. Otherwise, the other candidates will be used, in accordance with the usual ICE procedure.

Only the browser needs to implement the RETURN behavior - both the Border TURN Proxy and Application TURN servers' TURN protocol usage is unchanged.

Note that this arrangement preserves the end-to-end security and privacy features of WebRTC media flows. The ability to steer the

media flows through multiple TURN servers while still allowing end-to-end encryption and authentication is a key benefit of RETURN.

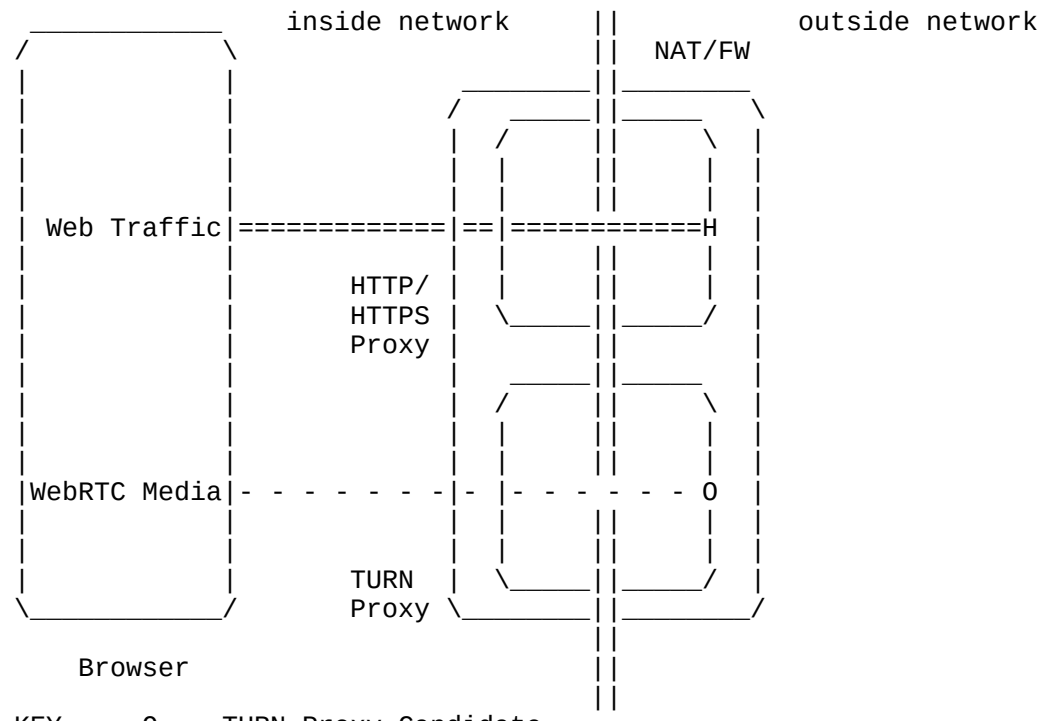


Figure 5: Similarity between HTTP/HTTPS Proxy and TURN Proxy

3. Goals

These goals are requirements on this document (not on implementations of the specification).

3.1. Connectivity

As noted in [I-D.ietf-rtcweb-use-cases-and-requirements] Section 3.3.5.1 and requirement F20, a WebRTC browser endpoint MUST be able to direct UDP connections through a designated TURN server configured by enterprise policy (a "proxy").

It MUST be possible to configure a WebRTC endpoint that supports proxies to achieve connectivity no worse than if the endpoint were operating at the proxy's address.

For efficiency, network administrators SHOULD be able to prevent browsers from attempting to send traffic through routes that are already known to be blocked.

3.2. Independent Path Control

Both network administrators and application developers may wish to direct all their UDP flows through a particular TURN server. There are many goals that might motivate such a choice, including

- improving quality of service by tunneling packets through a network that is faster than the public internet,
- monitoring the usage of UDP services,
- troubleshooting and debugging problematic services,
- logging connection metadata for legal or auditing reasons,
- recording the entire contents of all connections, or
- providing partial IP address anonymization (as described in [I-D.ietf-rtcweb-security] Section 4.2.4).

4. Concepts

To achieve our goals, we introduce the following new concepts:

4.1. Proxy

In this document a "proxy" is any TURN server that was provided by any mechanism other than through the standard WebRTC-application ICE candidate provisioning API [I-D.ietf-rtcweb-jsep]. We call it a "proxy" by analogy with SOCKS proxies and similar network services, because it performs a similar function and can be configured in a similar fashion.

If a proxy is to be used, it will be the destination of traffic generated by the client. (There is no analogue to the transparent/ intercepting HTTP proxy configuration, which modifies traffic at the network layer.) Mechanisms to configure a proxy include Auto-Discovery [I-D.ietf-tram-turn-server-discovery] and local policy ([I-D.ietf-rtcweb-jsep], "ICE candidate policy").

In an application context, a proxy may be "active" (producing candidates) or "inactive" (not in use, having no effect on the context).

4.2. Virtual interface

A typical WebRTC browser endpoint may have multiple network interfaces available, such as wired ethernet, wireless ethernet, and WAN. In this document, a "virtual interface" is a procedure for generating ICE candidates that are not simply generated by a particular physical interface. A virtual interface can produce "host", "server-reflexive", and "relay" candidates, but may be restricted to only some type of candidate (e.g. UDP-only).

4.3. Proxy configuration leakiness

"Leakiness" is an attribute of a proxy configuration. This document defines two values for the "leakiness" of a proxy configuration: "leaky" and "sealed". Proxy configuration, including leakiness, may be set by local policy ([I-D.ietf-rtcweb-jsep], "ICE candidate policy") or other mechanisms.

A leaky configuration adds a proxy and also allows the browser to use routes that transit directly via the endpoint’s physical interfaces (not through the proxy). In a leaky configuration, setting a proxy augments the available set of ICE candidates. Multiple leaky-configuration proxies may therefore be active simultaneously.

A sealed proxy configuration requires the browser to route all WebRTC traffic through the proxy, eliminating all ICE candidates that do not go through the proxy. Only one sealed proxy may be active at a time.

Leaky proxy configurations allow more efficient routes to be selected. For example, two peers on the same LAN can connect directly (peer to peer) if a leaky proxy is enabled, but must "hairpin" through the TURN proxy if the configuration is sealed. However, sealed proxy configurations can be faster to connect, especially if many of the peer-to-peer routes that ICE will try first are blocked by the network’s firewall policies.

4.4. Sealed proxy rank

In some configurations, an endpoint may be subject to multiple sealed proxy settings at the same time. In that case, one of those settings will have highest rank, and it will be the active proxy. In a given application context (e.g. a webpage), there is at most one active sealed proxy. This document does not specify a representation for rank.

5. Requirements

5.1. ICE candidates produced in the presence of a proxy

When a proxy is configured, by Auto-Discovery or a proprietary means, the browser MUST NOT report a "relay" candidate representing the proxy. Instead, the browser MUST connect to the proxy and then, if the connection is successful, treat the TURN tunnel as a UDP-only virtual interface.

For a virtual interface representing a TURN proxy, this means that the browser MUST report the public-facing IP address and port acquired through TURN as a "host" candidate, the browser MUST perform STUN through the TURN proxy (if STUN is configured), and it MUST perform TURN by recursive encapsulation through the TURN proxy, resulting in TURN candidates whose "raddr" and "rport" attributes match the acquired public-facing IP address and port on the proxy.

Because the virtual interface has some additional overhead due to indirection, it SHOULD have lower priority than the physical interfaces if physical interfaces are also active. Specifically, even host candidates generated by a virtual interface SHOULD have priority 0 when physical interfaces are active (similar to [RFC5245] Section 4.1.2.2, "the local preference for host candidates from a VPN interface SHOULD have a priority of 0").

5.2. Leaky proxy configuration

If the active proxy for an application is leaky, the browser should undertake the standard ICE candidate discovery mechanism [RFC5245] on the available physical and virtual interfaces.

5.3. Sealed proxy configuration

If the active proxy for an application is sealed, the browser MUST NOT gather or produce any candidates on physical interfaces. The WebRTC implementation MUST direct its traffic from those interfaces only to the proxy, and perform ICE candidate discovery only on the single virtual interface representing the active proxy.

5.4. Proxy rank

Any browser mechanism for specifying a proxy SHOULD allow the caller to indicate a higher rank than the proxy provided by Auto-Discovery [I-D.ietf-tram-turn-server-discovery].

5.5. Multiple physical interfaces

Some operating systems allow the browser to use multiple interfaces to contact a single remote IP address. To avoid producing an excessive number of candidates, WebRTC endpoints MUST NOT use multiple physical interfaces to connect to a single proxy simultaneously. (If this were violated, it could produce a number of virtual interfaces equal to the product of the number of physical interfaces and the number of active proxies.)

For strategies to choose the best interface for communication with a proxy, see [I-D.reddy-mmusic-ice-best-interface-pcp]. Similar considerations apply when connecting to an application-specified TURN server in the presence of physical and virtual interfaces.

5.6. IPv4 and IPv6

A proxy MAY have both an IPv4 and an IPv6 address (e.g. if the proxy is specified by DNS and has both A and AAAA records). The client MAY try both of these addresses, but MUST select one, preferring IPv6, before allocating any remote addresses. This corresponds to the the Happy Eyeballs [RFC6555] procedure for dual-stack clients.

A proxy MAY provide both IPv4 and IPv6 remote addresses to clients [RFC6156]. A client SHOULD request both address families. If both requests are granted, the client SHOULD treat the two addresses as host candidates on a dual-stack virtual interface.

5.7. Unspecified leakiness

If a proxy configuration mechanism does not specify leakiness, browsers SHOULD treat the proxy as leaky. This is similar to current WebRTC implementations' behavior in the presence of SOCKS and HTTP proxies: the candidate allocation code continues to generate UDP candidates that do not transit through the proxy.

5.8. Interaction with SOCKS5-UDP

The SOCKS5 proxy standard [RFC1928] permits compliant SOCKS proxies to support UDP traffic. However, most implementations of SOCKS5 today do not support UDP. Accordingly, WebRTC browsers MUST by default (i.e. unless deliberately configured otherwise) treat SOCKS5 proxies as leaky and having lower rank than any configured TURN proxies.

5.9. Encapsulation overhead, fragmentation, and Path MTU

Encapsulating a link in TURN adds overhead on the path between the client and the TURN server, because each packet must be wrapped in a TURN message. This overhead is sometimes doubled in RETURN proxying. To avoid excessive overhead, client implementations SHOULD use ChannelBind and ChannelData messages to connect and send data through proxies and application TURN servers when possible. Clients MAY buffer messages to be sent until the ChannelBind command completes (requiring one round trip to the proxy), or they MAY use CreatePermission and Send messages for the first few packets to reduce startup latency at the cost of higher overhead.

Adding overhead to packets on a link decreases the effective Maximum Transmissible Unit on that link. Accordingly, clients that support proxying MUST NOT rely on the effective MTU complying with the Internet Protocol's minimum MTU requirement.

ChannelData messages have constant overheard, enabling consistent effective PMTU, but Send messages do not necessarily have constant overhead. TURN messages may be fragmented and reassembled if they are not marked with the Don't Fragment (DF) IP bit or the DONT-FRAGMENT TURN attribute. Client implementors should keep this in mind, especially if they choose to implement PMTU discovery through the proxy.

5.10. Interaction with alternate TURN server fallback

As per [RFC5766], a TURN server MAY respond to an Allocate request with an error code of 300 and an ALTERNATE-SERVER indication. When connecting to proxies or application TURN servers, clients SHOULD attempt to connect to the specified alternate server in accordance with [RFC5766]. The client MUST route a connection to the alternate server through the proxy if and only if the original connection attempt was routed through the proxy.

5.11. Reusing the same TURN server

It is possible that the same TURN server may appear more than once in the network path. For example, if both endpoints configure the same sealed proxy, then each peer will only provide candidates on this proxy. This is not a problem, and will work as expected.

It is also possible that the same TURN server could be used by both the enterprise and the application. It might appear attractive to connect to this server only once, rathering connecting to it through itself, in order to avoid imposing unnecessary server load. However,

a RETURN client MUST connect to the server twice, even when this appears redundant, to ensure correct session attribution.

For example, consider a TURN service operator that issues different authentication credentials to different customers, and then allows each customer to observe the source and destination IP addresses used with their credentials. Suppose the application and enterprise both have accounts on this service: the application uses it to prevent the enterprise from learning its peers' IP addresses, and the enterprise uses it to prevent the application from learning its employees' IP addresses. If the client only connects to the service once, then either the enterprise or the application will learn IP address information (via the TURN provider's metadata reporting) that was meant to be kept secret.

As a result of this requirement, it is possible for the same TURN server to appear up to four times in a RETURN network path: once as each peer's application's TURN server, and once as each peer's sealed proxy.

6. Examples

6.1. Firewalled enterprise network with a basic application

In this example, an enterprise network is configured with a firewall that blocks all UDP traffic, and a TURN server is advertised for Auto-Discovery in accordance with [I-D.ietf-tram-turn-server-discovery]. The proxy leakiness of the TURN server is unspecified, so the browser treats it as leaky.

The application specifies a STUN and TURN server on the public net. In accordance with the ICE candidate gathering algorithm RFC 5245 [RFC5245], it receives a set of candidates like:

1. A host candidate acquired from one interface.
    * e.g. candidate:1610808681 1 udp 2122194687 [internal ip addr for interface 0] 63555 typ host generation 0
2. A host candidate acquired from a different interface.
    * e.g. candidate:1610808681 1 udp 2122194687 [internal ip addr for interface 1] 54253 typ host generation 0
3. The proxy, as a host candidate.
    * e.g. candidate:3458234523 1 udp 24584191 [public ip addr for the proxy] 54606 typ host generation 0

4. The virtual interface also generates a STUN candidate, but it is eliminated because it is redundant with the host candidate, as noted in [RFC5245] Sec 4.1.2..

5. The application-provided TURN server as seen through the virtual interface. (Traffic through this candidate is recursively encapsulated.)

   * e.g. candidate:702786350 1 udp 24583935 [public ip addr of the application TURN server] 52631 typ relay raddr [public ip addr for the proxy] rport 54606 generation 0

There are no STUN or TURN candidates on the physical interfaces, because the application-specified STUN and TURN servers are not reachable through the firewall.

If the remote peer is within the same network, it may be possible to establish a direct connection using both peers' host candidates. If the network prevents this kind of direct connection, the path will instead take a "hairpin" route through the enterprise's proxy, using one peer's physical "host" candidate and the other's virtual "host" candidate, or (if that is also disallowed by the network configuration) a "double hairpin" using both endpoints' virtual "host" candidates.

6.2. Conflicting proxies configured by Auto-Discovery and local policy

Consider an enterprise network with TURN and HTTP proxies advertised for Auto-Discovery with unspecified leakiness (thus defaulting to leaky). The browser endpoint configures an additional TURN proxy by a proprietary local mechanism.

If the locally configured proxy is leaky, then the browser MUST produce candidates representing any physical interfaces (including SSLTCP routes through the HTTP proxy), plus candidates for both UDP-only virtual interfaces created by the two TURN servers.

There MUST NOT be any candidate that uses both proxies. Multiple configured proxies are not chained recursively.

If the locally configured proxy is "sealed", then the browser MUST produce only candidates from the virtual interface associated with that proxy.

If both proxies are configured for "sealed" use, then the browser MUST produce only candidates from the virtual interface associated with the proxy with higher rank.

7. Security Considerations

A RETURN proxy can capture, block, and otherwise interfere with all of its clients' WebRTC network activity. Therefore, browsers and other WebRTC endpoints MUST NOT use RETURN proxies that are provided by untrusted sources. For example, endpoints MUST NOT implement a configuration based on unauthenticated network multicast (e.g. mDNS) unless the endpoint will only be used on networks where all other users are fully trusted to intercept all WebRTC traffic. In contrast, endpoints MAY implement mechanisms to configure RETURN proxies by system-wide policy, which can only be modified by trusted system administrators.

This document describes web browser behaviors that, if implemented correctly, allow users to achieve greater identity-confidentiality during WebRTC calls under certain configurations.

If a site administrator offers the site's users a TURN proxy, websites running in the users' browsers will be able to initiate a UDP-based WebRTC connection to any UDP transport address via the proxy. Websites' connections will quickly terminate if the remote endpoint does not reply with a positive indication of ICE consent, but no such restriction applies to other applications that access the TURN server. Administrators should take care to provide TURN access credentials only to the users who are authorized to have global UDP network access.

TURN proxies and application TURN servers can provide some privacy protection by obscuring the identity of one peer from the other. However, unencrypted TURN provides no additional privacy from an observer who can monitor the link between the TURN client and server, and even encrypted TURN ([I-D.ietf-tram-stun-dtls] Section 4.6) does not provide significant privacy from an observer who sniff traffic on both legs of the TURN connection, due to packet timing correlations.

8. IANA Considerations

This document requires no actions from IANA.

9. Acknowledgements

Thanks to Harald Alvestrand, Philipp Hancke, Tirumaleswar Reddy, Alan Johnston, John Yoakum, and Cullen Jennings for suggestions to improve the content and presentation. Special thanks to Alan Johnston for contributing the visual overview in Section 2.

10. References

10.1. Normative References

[I-D.ietf-rtcweb-jsep]
Uberti, J. and C. Jennings, "Javascript Session Establishment Protocol", draft-ietf-rtcweb-jsep-06 (work in progress), February 2014.

[RFC1928] Leech, M., Ganis, M., Lee, Y., Kuris, R., Koblas, D., and L. Jones, "SOCKS Protocol Version 5", RFC 5766, March 1996.

[RFC5245] Rosenberg, J., "Interactive Connectivity Establishment (ICE): A Protocol for Network Address Translator (NAT) Traversal for Offer/Answer Protocols", RFC 5245, April 2010.

[RFC5766] Mahy, R., Matthews, P., and J. Rosenberg, "Traversal Using Relays around NAT (TURN): Relay Extensions to Session Traversal Utilities for NAT (STUN)", RFC 5766, April 2010.

[RFC6156] Camarillo, G., Novo, O., and S. Perreault, "Traversal Using Relays around NAT (TURN) Extension for IPv6", RFC 6156, April 2011.

[RFC6555] Wing, D. and A. Yourtchenko, "Happy Eyeballs: Success with Dual-Stack Hosts", RFC 6555, April 2012.

10.2. Informative References

[I-D.ietf-rtcweb-security]
Rescorla, E., "Security Considerations for WebRTC", ietf-rtcweb-security-07 (work in progress), July 2014.

[I-D.ietf-rtcweb-use-cases-and-requirements]
Holmberg, C., Hakansson, S., and G. Eriksson, "Web Real-Time Communication Use-cases and Requirements", ietf-rtcweb-use-cases-and-requirements-14 (work in progress), February 2014.

[I-D.ietf-tram-stun-dtls]
Petit-Huguenin, M. and G. Salgueiro, "Datagram Transport Layer Security (DTLS) as Transport for Session Traversal Utilities for NAT (STUN)", ietf-rtcweb-use-cases-and-requirements-14 (work in progress), June 2014.

[I-D.ietf-tram-turn-server-discovery]
Patil, P., Reddy, T., and D. Wing, "TURN Server Auto Discovery", draft-ietf-tram-turn-server-discovery-00 (work in progress), July 2014.

[I-D.reddy-mmusic-ice-best-interface-pcp]
Reddy, T., Wing, D., VerSteeg, B., Penno, R., and V. Singh, "Improving ICE Interface Selection Using Port Control Protocol (PCP) Flow Extension", draft-ietf-tram-turn-server-discovery-00 (work in progress), October 2013.

Authors' Addresses

Benjamin M. Schwartz
Google, Inc.
111 8th Ave
New York, NY 10011
USA

Email: bemasc@webrtc.org

Justin Uberti
Google, Inc.
747 6th Street South
Kirkland, WA 98033
USA

Email: justin@uberti.name

TRAM Working Group
Internet-Draft
Intended status: Standards Track
Expires: August 23, 2015

A. Johnston
Avaya
J. Uberti
Google
J. Yoakum
K. Singh
Avaya

February 19, 2015

An Origin Attribute for the STUN Protocol
draft-ietf-tram-stun-origin-05

Abstract

STUN, or Session Traversal Utilities for NAT, is a protocol used to assist other protocols traverse Network Address Translators or NATs. This specification defines an ORIGIN attribute for STUN that can be used in similar ways to the HTTP header field of the same name. WebRTC browsers utilizing STUN and TURN would include this attribute which would provide servers with additional information about the STUN and TURN requests they receive. This specification defines the usage of the STUN ORIGIN attribute for web, SIP, and XMPP contexts.

Table of Contents

1. Introduction

STUN, or Session Traversal Utilities for NAT, is a protocol used to assist other protocols traverse Network Address Translators or NATs. TURN, or Traversal Using Relays around NAT [RFC5766], is a STUN extension [RFC5389] that allows endpoints to acquire a relayed address for media flows. It is most commonly used in conjunction with ICE, Interactive Connectivity Establishment [RFC5245], which is used to establish peer-to-peer flows between endpoints through NATs and firewalls.

STUN defines three authentication modes, depending on the STUN usage. For STUN binding requests sent between peers, such as for ICE connectivity checks, a short term authentication method is recommended. Each peer contributes random strings which are exchanged over signaling and used to authenticate the connectivity checks. For TURN, a usage of STUN used to acquire and refresh relay addresses, a long term authentication method is recommended. This authentication is similar to SIP Digest [RFC3261], which involves an authentication challenge for each request. A server, upon receipt of a TURN request, generates an authentication challenge that includes a realm and nonce. The client resends the TURN request supplying a user name and password based on the realm indicated by the server. For a STUN binding request sent to a STUN server, no authentication is recommended, as generating the response is less work for a server than the server utilizing the short term or long term authentication approach. The server responds statelessly, so the only resources used are those to process each request, which are minimal.

WebRTC, Web Real-Time Communications, adds peer-to-peer real-time, interactive voice and video media capabilities and data channels to browsers [I-D.ietf-rtcweb-overview] without a plugin or download, and allows web developers to access this functionality using JavaScript API calls [WebRTC-API]. WebRTC includes STUN, TURN, and ICE client functionality built into browsers. For a session established between two browsers, if either browser is behind a NAT, a STUN server is necessary. Public STUN servers are currently available and a web application can suggest a particular STUN server be used. In cases where a particular combination of NAT mapping and filtering and/or firewalling is present, a TURN server is needed to establish a peer connection. In this case, TURN credentials need to be available to the browser for the long term authentication approach. A TURN server for WebRTC might serve a number of different domains and realms.

From the perspective of the web application provider, providing service for a number of different domains and realms, it is useful to know something about the source of the STUN request when processing the request. For a web application provider STUN or TURN server, the

server will have no idea which web pages or sites are sending binding requests to the service. In conventional applications, the SOFTWARE attribute would provide some identifying information to the service, but that no longer works when the browser is the application. For a web application provider TURN server, the TURN server does not know which realm to include in an authentication challenge.

In the web world, HTTP requests have the concept of origin. The origin of a web page, as defined in [RFC6454], is defined by the URI's scheme, host or IP address, and port portions. The HTTP Origin header field inserted by the web browser carries this information and is useful information for servers that receive HTTP requests generated via JavaScript. For example, Cross Origin Resource Sharing, CORS, allows an HTTP server to serve HTTP requests from multiple origins.

This specification proposes extending the origin concept to STUN requests. STUN requests generated by a web browser would include the origin of the HTTP page that is initiating the Peer Connection. Using this extra information, a STUN server could use the origin to determine which STUN binding requests to respond to, reducing the load on a STUN server. Using this information, a TURN server could use the origin to determine which realm to include in the authentication challenge. A TURN server can also use the origin information for logging and analytics, and also as additional information after authentication for providing service. This specification also defines an origin for SIP and XMPP users of STUN and TURN.

An important use case that the STUN Origin helps solve is the operation of a multi-tenanted TURN server (i.e. a TURN server that serves multiple, perhaps tens of thousands of different domains). The problem associated with this use case is described in Section 4.5 of [RFC7376]. While it is possible for a TURN server to use the same authentication credentials across many domains, a more likely (and more manageable) scenario is to have separate credentials for each domain, and hence a different realm for each domain. With the TURN server configured with a mapping between a domain (conveyed in the Origin) and the realm string (to be used in the authentication challenge), a single TURN URI could be used across all domains, and the resulting JavaScript code would be portable.

Note that the origin information is most useful as a hint in initial STUN and TURN requests as received by a server. However, origin information still has value throughout the session even after authentication for logging and other purposes.

The following sections of this document define the STUN ORIGIN

attribute and define its usage.

1.1. Requirements Language

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

2. STUN ORIGIN attribute

This specification defines how to apply the web origin concept and syntax of [RFC6454] to the STUN protocol.

This specification defines a new Attribute to the STUN protocol [RFC5389]. The attribute is called ORIGIN and uses the syntax defined in Section 15 of [RFC5389]. The number used for this in the type field is 0x802F, chosen in the comprehension optional range. The value of ORIGIN is a variable-length value. It MUST contain a UTF-8 [RFC3629] encoded sequence of characters. Senders MAY include multiple ORIGIN attributes in a request, and receivers MUST support parsing and receiving multiple ORIGIN attributes. The size of ORIGINs included in a STUN message can have a major impact on the size of a STUN packet, and could potentially cause UDP fragmentation. HTTP origins are less than 267 characters (the maximum 253 character domain name plus 8 characters for the URI scheme plus 5 characters for the port number).

For a web browser (HTTP User Agent), the contents of the ORIGIN attribute is the unicode-serialization of an origin defined in Section 6.1 of [RFC6454]. The origin value included is the same as the Origin header field for an HTTP request generated from the web page that is creating the Peer Connection. It does not include any string terminating (\x00) character in the serialization.

For a SIP User Agent [RFC3261] using STUN and TURN, the ORIGIN attribute is set to be the URI of the registrar server used by the User Agent (i.e. the Request-URI of a REGISTER method). This is the full Request-URI component of the SIP ABNF defined in [RFC3261]. For example, a SIP user "sip:bob@biloxi.example.com" might register with the Request-URI of "sip:registrar.biloxi.example.com".

For an XMPP client [RFC6120] using STUN and TURN, the ORIGIN attribute is an XMPP URI [RFC5122] representing the full domainpart of the client’s Jabber ID (JID) as defined in the ABNF of [RFC6122]; for example, if the client’s JID is "juliet@im.example.com/balcony" then the ORIGIN attribute would be "xmpp:im.example.com".

Other contexts can define a usage of the ORIGIN attribute to use an appropriate URI or URL.

If an ORIGIN attribute is not present in a request, it is up to the server how to handle the request. For example, it could assume a default Origin.

2.1. STUN Usage

For STUN requests sent without authentication to a STUN server (i.e. STUN binding requests sent to a STUN server), the STUN client MUST include the ORIGIN attribute when the origin is a web origin. For other origins, such as SIP or XMPP, the STUN client SHOULD include the ORIGIN attribute.

> Note that for privacy reasons, an empty ORIGIN attribute can be sent, as described in the Security Considerations.

A STUN server can derive additional information for logging and analytics about the request through the ORIGIN attribute, such as the source of the request. For example, an enterprise STUN server might only reply to STUN binding requests from certain domains.

2.2. TURN Usage

When the origin is a web origin, TURN [RFC5766] Allocate requests MUST include the ORIGIN attribute. For all other TURN requests, including the Send method, the use of the ORIGIN attribute is NOT RECOMMENDED. For other origins, such as SIP or XMPP, TURN Allocate requests SHOULD include the ORIGIN attribute. A TURN server can use the ORIGIN attribute to determine which REALM to include in the authentication challenge. A TURN server can also use the ORIGIN attribute after authentication to provide appropriate service.

2.3. NAT Behavior Discovery Usage

For the NAT Behavior Discovery Usage in [RFC5780], the rules for STUN usage apply.

2.4. ICE Usage

ICE [RFC5245] connectivity checks MUST NOT include the ORIGIN attribute. Including the ORIGIN attribute leaks information about the site used by one peer to the other peer in the session.

2.5. Media Keep-Alive Usage

For media keep-alive STUN requests described in Section 20 of [RFC5245], the rules for ICE usage apply.

2.6. SIP Keep-Alive Usage

For SIP keep-alive STUN requests described in [RFC5626], the use of the ORIGIN attribute is NOT RECOMMENDED. No valid use cases for the ORIGIN attribute have been identified to date, and as a result the ORIGIN attribute will be ignored.

2.7. Multiple Origins

Multiple Origins for HTTP Requests are described in Section 7.2 of [RFC6454]. Multiple origins can occur when the same resource is fetched by multiple origins at the same time (e.g. multiple tabs, windows, frames, etc.). In the context of WebRTC, it doesn't make sense for a STUN binding or TURN allocation to be shared across origins (e.g. Peer Connections). Based on their definitions, multiple SIP and XMPP origins also do not apply here. Therefore, if a STUN request contains multiple origins, the first origin MUST be used and the remaining origins ignored.

3. IANA Considerations

This specification, if approved, adds a new value to the IANA "STUN Attributes Registry" created by [RFC5389]. The ORIGIN attribute value is 0x802F.

4. Security Considerations

The security considerations of [RFC6454] apply to this extension. Servers using the information present in the STUN ORIGIN attribute need to realize that this attribute could be set arbitrarily by a non-browser client or modified by an intermediary. The method proposed in this document is not meant to replace existing STUN authentication mechanisms but to provide additional information to the server for logging and analytics and how to handle the request after authentication.

Just as browsers do not allow a web application to set the HTTP Origin header field via JavaScript, web browsers (HTTP User Agents) MUST NOT allow a web application to set the STUN ORIGIN attribute through JavaScript.

While the STUN MESSAGE-INTEGRITY attribute can provide integrity protection for all attributes present in a STUN request, MESSAGE-INTEGRITY is not present in the initial STUN message sent. As a result, an ORIGIN attribute could be modified or removed from a STUN request without the server knowing. DTLS or TLS transport SHOULD be used when integrity protection for the ORIGIN attribute is important.

The STUN ORIGIN attribute has privacy implications. As a result, TLS or DTLS transport SHOULD be used. If STUN requests containing the origin attribute are not encrypted with TLS or DTLS, an on-path attacker could determine this information by inspecting STUN messages between the STUN client and STUN server. This information is often available in other messages sent by the browser, such as DNS or HTTP requests. However, in cases where secure HTTP is used, including the ORIGIN attribute over an unencrypted transport could leak this information. In cases where privacy is paramount, but TLS or DTLS transport is not available, the ORIGIN attribute MAY be sent with an empty (null) origin value. It is up to the server what to do when receiving such an empty origin. The server could fail the request, or the server could assume an origin and attempt to process the request.

The STUN ORIGIN attribute also has privacy implications in that the origin information is shared with a STUN or TURN server which otherwise might not know this information. This information could be used to track usage of real-time communication services. A STUN or TURN server will always know the public IP address of each user, but the ORIGIN attribute provides more information about which service or provider is being used. The particular STUN and TURN servers used are usually selected by the real-time communications service provider (i.e. the web provider for WebRTC or the SIP or XMPP service provider). In addition, they are usually also run by the same provider, or by a trusted partner, especially for TURN. However, a service or provider using an untrusted third party STUN or TURN server needs to recognize that the operator of the third party STUN or TURN server will learn the identity of the service or provider through this extension. In cases where the STUN or TURN client does not wish to share origin information with the server for privacy reasons, the ORIGIN attribute MAY be sent with an empty (null) origin value. It is up to the server what to do when receiving such an empty origin. The server could fail the request, or the server could assume an origin and attempt to process the request.

5. Implementation Status

Note to RFC Editor: Please remove this entire section prior to publication, including the reference to RFC 6982.

This section records the status of known implementations of the protocol defined by this specification at the time of posting of this Internet-Draft, and is based on a proposal described in [RFC6982]. The description of implementations in this section is intended to assist the IETF in its decision processes in progressing drafts to RFCs. Please note that the listing of any individual implementation here does not imply endorsement by the IETF. Furthermore, no effort has been spent to verify the information presented here that was supplied by IETF contributors. This is not intended as, and must not be construed to be, a catalog of available implementations or their features. Readers are advised to note that other implementations may exist.

According to [RFC6982], "this will allow reviewers and working groups to assign due consideration to documents that have the benefit of running code, which may serve as evidence of valuable experimentation and feedback that have made the implemented protocols more mature. It is up to the individual working groups to use this information as they see fit".

Two proof-of-concept implementations have been created in support of this proposed standard. One provides a WebRTC enabled browser that includes the appropriate STUN ORIGIN Attribute with the Origin insight known to the browser in STUN/TURN messages sent to servers. The other provides an example of a multiple realms capable TURN server that takes advantage of Origin insight provided in the STUN ORIGIN Attribute.

A Chrome browser implementation has been created by Graham Yoakum and Ryan Yoakum (Skobalt LLC) and is freely licensed under the standard terms of the open source Chromium and WebRTC projects. This proof-of-concept version of the Google Chrome browser (nicknamed ’Chromeo’) sends Origin insight in STUN and TURN messages using the proposed new STUN ORIGIN attribute with a value of 0x802F (as initially proposed, however that value is easily changed in a single line of code). ’Chromeo’ includes a Chrome flag to enable and disable this unique feature (and is by default disabled to prevent any non-intentional use of this feature until the standard is finalized). This implementation is based on is draft-johnston-tram-stun-origin-02.

Coordinated changes to both the WebRTC and Chromium open source projects have been formally submitted for consideration. The two submitted change lists together implement the complete browser proof-of-concept. ’Chromeo’ has been built for Linux and STUN protocol behavior has been verified using WireShark traces illustrating that proper STUN Origin attributes are being included in STUN/TURN messages sent by the browser to servers (screen captures of STUN messages illustrating the Origin attribute and content are

available).

The WebRTC and Chromium open source projects can be found at: http://www.webrtc.org/ and http://www.chromium.org/

Google can choose to accept or modify the changes proposed for Chrome and other browser vendors can access and take advantage of the publicly available WebRTC and Chromium open source submissions as desired. Hopefully this will enable browsers to quickly implement STUN Origin enhancements.

A multiple realms capable advanced open source Origin enabled TURN server (named 'Coturn') has been created by Oleg Moskalenko and is freely licensed under the New BSD license. This reference implementation and proof-of-concept provides a clone (a spin-off) of the rfc5766-turn-server project adding Origin-based multiple realms support.

'Coturn' is backward-compatible with rfc5766-turn-server project but the code is more complex and it uses a different (also more complex) database structure. It is the intent to add all IETF TRAM TURN server related capabilities to this project as they mature. 'Coturn' is publicly available and can be found at: https://code.google.com/p/coturn/

6. Acknowledgements

Thanks to John Selbie, Tirumaleswar Reddy, Simon Perreault, Marc Petit-Huguenin, Andy Hutton, and Oleg Moskalenko for their feedback and reviews. Special thanks to Graham Yoakum and Ryan Yoakum of Skobalt LLC and Oleg Moskalenko of rfc5766-turn-server project for contributing open source proof-of-concept implementations for a Chrome web browser and a multiple realms capable TURN server, quickly demonstrating feasibility.

7. References

7.1. Normative References

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

[RFC3261] Rosenberg, J., Schulzrinne, H., Camarillo, G., Johnston, A., Peterson, J., Sparks, R., Handley, M., and E. Schooler, "SIP: Session Initiation Protocol", RFC 3261, June 2002.

[RFC3629] Yergeau, F., "UTF-8, a transformation format of ISO 10646", STD 63, RFC 3629, November 2003.

[RFC5122] Saint-Andre, P., "Internationalized Resource Identifiers (IRIs) and Uniform Resource Identifiers (URIs) for the Extensible Messaging and Presence Protocol (XMPP)", RFC 5122, February 2008.

[RFC5245] Rosenberg, J., "Interactive Connectivity Establishment (ICE): A Protocol for Network Address Translator (NAT) Traversal for Offer/Answer Protocols", RFC 5245, April 2010.

[RFC5389] Rosenberg, J., Mahy, R., Matthews, P., and D. Wing, "Session Traversal Utilities for NAT (STUN)", RFC 5389, October 2008.

[RFC5626] Jennings, C., Mahy, R., and F. Audet, "Managing Client-Initiated Connections in the Session Initiation Protocol (SIP)", RFC 5626, October 2009.

[RFC5766] Mahy, R., Matthews, P., and J. Rosenberg, "Traversal Using Relays around NAT (TURN): Relay Extensions to Session Traversal Utilities for NAT (STUN)", RFC 5766, April 2010.

[RFC6120] Saint-Andre, P., "Extensible Messaging and Presence Protocol (XMPP): Core", RFC 6120, March 2011.

[RFC6122] Saint-Andre, P., "Extensible Messaging and Presence Protocol (XMPP): Address Format", RFC 6122, March 2011.

[RFC6454] Barth, A., "The Web Origin Concept", RFC 6454, December 2011.

[RFC7350] Petit-Huguenin, M. and G. Salgueiro, "Datagram Transport Layer Security (DTLS) as Transport for Session Traversal Utilities for NAT (STUN)", RFC 7350, August 2014.

7.2. Informative References

[I-D.ietf-rtcweb-overview]
Alvestrand, H., "Overview: Real Time Protocols for Browser-based Applications", draft-ietf-rtcweb-overview-13 (work in progress), November 2014.

[RFC5780] MacDonald, D. and B. Lowekamp, "NAT Behavior Discovery Using Session Traversal Utilities for NAT (STUN)", RFC 5780, May 2010.

[RFC6982] Sheffer, Y. and A. Farrel, "Improving Awareness of Running Code: The Implementation Status Section", RFC 6982, July 2013.

[RFC7376] Reddy, T., Ravindranath, R., Perumal, M., and A. Yegin, "Problems with Session Traversal Utilities for NAT (STUN) Long-Term Authentication for Traversal Using Relays around NAT (TURN)", RFC 7376, September 2014.

[WebRTC-API] Bergkvist, A., Burnett, D., Jennings, C., and A. Narayanan, "WebRTC 1.0: Real-time Communication Between Browsers", W3C Working Draft http://www.w3.org/TR/webrtc/, 2013, <http://www.w3.org/TR/2013/WD-webrtc-20130910/>.

Authors' Addresses

Alan Johnston
Avaya
St. Louis, MO
USA

Phone:
Email: alan.b.johnston@gmail.com

Justin Uberti
Google
Kirkland, WA
USA

Phone:
Email: justin@uberti.name

John Yoakum
Avaya
Cary, NC
USA

Phone:
Email: yoakum@avaya.com

Kundan Singh
Avaya
San Francisco, CA
USA

Phone:
Email: kundan10@gmail.com

rtcweb
Internet-Draft
Intended status: Informational
Expires: January 7, 2016

C. Jennings
S. Nandakumar
J. Rosenberg
Cisco

July 06, 2015

Firewall Traversal for WebRTC
draft-jennings-behave-rtcweb-firewall-00

Abstract

Traversal of RTP through corporate firewalls has traditionally been complex, requiring the deployment of Session Border Controllers (SBCs) or wide open pinholes. This draft proposes a simple technique that allows WebRTC based RTP traffic to traverse firewalls without complex firewall configuration and without deployment of SBCs or other middleboxes.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at http://datatracker.ietf.org/drafts/current/.

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."

This Internet-Draft will expire on January 7, 2016.

Copyright Notice

Table of Contents

1. Problem Statement

WebRTC [I-D.ietf-rtcweb-overview] based voice and video communications systems are becoming far more common inside enterprises, which often need voice and video media to traverse the enterprise firewall. This can happen when a device inside the firewall such as a web browser or phone is exchanging media with a conference bridge or gateway outside the firewall, or it can happen when a device inside the firewall is talking to a device in another enterprise or behind a different firewall.

This problem is not unique to WebRTC media of course. It is common practice for enterprise administrators to block outbound UDP through the corporate firewall. This is done for several reasons:

1. The lack of any kind of return messages means that there is no way to know that the recipient of the UDP traffic really wants it. Infected computers within the enterprise could utilize UDP

as the source of a DDoS attack. If the firewall permitted such outbound traffic, the enterprise could in effect be a contributing source to such an attack. By blocking UDP, the enterprise IT admin ensures that this cannot happen - at least not to external targets.

2. There have been prior attacks that have utilized UDP as a command and control channel for orchestrating DDoS attacks. At the time, UDP had little usage within enterprises (most VoIP was internal to the enterprise when it existed at all). Consequently, infosec departments have deemed it safer to block UDP outright in order to prevent such further incidents.

3. Many IT administrators enable various packet inspection operations on traffic flowing through the firewall. High volume UDP traffic - such as voice or video - can be costly to inspect. As such, in cases where there is a need for traversal of such traffic, IT has preferred to deploy an SBC that, in essence, verifies that the traffic is VoIP and authorizes its egress. The IT administrator then enables traffic to/from the SBC through the firewall. In other words, VoIP authorization is delegated to an outsourced SBC.

As more and more IP communications services move to the cloud, there is an increased need for VoIP traffic to traverse the enterprise firewall. At the same time, the entire point of a cloud service is that it does not require the deployment of on premises infrastructure, making SBC-based solutions less desirable. An alternative solution that has been historically used is to enable outbound UDP in the firewall to specific IP addresses, corresponding to the external service (TURN servers or conference servers) that the enterprise wishes to authorize. With more applications running on virtual machines within cloud compute platforms like Amazon EC2, IP addresses are decreasingly usable as identifiers for a service. VMs running TURN servers or conferencing servers may be established and torn down by the day, hour or even minute, with continuously changing IP addresses. Given the multitenant nature of such providers, IT departments are unwilling to whitelist the IP addresses for the entire block used by such providers.

Consequently, there is a growing need for solutions that allow VoIP traversal through the corporate firewall that alleviate the concerns above. This issue is further exacerbated by the growing adoption of WebRTC by enterprise applications, which provide a ready source of RTP traffic which often needs to traverse the firewall.

2. Solution Requirements

We believe the solution must meet the following requirements:

REQ-1: The solution must enable traversal of real-time media without requiring deployment of additional media intermediaries on premise (e.g., no SBC required)

REQ-2: The solution must not require the whitelisting of specific external IP addresses

REQ-3: The solution must enable the enterprise to be sure that the receiving party of the traffic desires the traffic

REQ-4: The solution must work with P2P calls between users in different enterprises without requiring a TURN server

REQ-5: The solution must work with cloud services external to the enterprise which terminate media on servers, such as conference servers, voicemail servers, recording servers, and so on.

REQ-6: The solution must not require decryption of either signalling or media traffic at the firewall or at any other intermediary

REQ-7: The solution must allow the IT department to easily make policy decisions about which applications are allowed, or not allowed, to traverse the firewall

REQ-8: The solution must not require DPI of every single UDP packet that traverses the firewall

REQ-9: The solution must provide a minimum level of proof that the traffic is RTP and not something else

REQ-10: The solution must work with WebRTC traffic. Note that solving this for non-WebRTC is a non-requirement.

3. Solution Overview

Many of the reasons for blocking UDP at the corporate firewall have their origins in the lack of a three-way handshake for UDP traffic. TCP’s three-way handshake ensures that the receiving party of the connection desires the traffic. Similarly, HTTP traffic easily traverses the firewall since it provides application identification information in the URL.

Consequently, the solution proposed here relies on the ICE connectivity checks, which provide a similar handshake and ensure

consent of the remote party. When a firewall sees an outbound UDP packet on a 5-tuple which is not yet authorized, it begins looking for STUN packets (as identified by the STUN magic cookie). Any outbound packet that is not a STUN packet is discarded. Once an outbound STUN packet is identified, the 5-tuple is put in a pending state, and the firewall begins looking for STUN packets among inbound UDP packets. When it sees one, it matches the transaction IDs to ensure that they are correlated. Once matched, the 5-tuple is placed into an authorized state, and UDP traffic is allowed to freely traverse.

In addition, the initial outbound STUN packets contain the STUN ORIGIN field which the firewall can use to make an authorization decision on the application.

4. Firewall Processing

The firewall processing is broken into three stages: recognizing STUN packets, making a policy decision as to whether each STUN packet should trigger a pinhole to be created, and managing the lifetime of any pinholes that are created.

4.1. Recognizing STUN packets

STUN messages all have a magic cookie value of 0x2112A442 in the 4th to 8th byte. This can be used to quickly filter nearly all UDP packets that are not STUN packets. Many firewalls are capable of doing this in hardware. STUN supports an optional FINGERPRINT attribute that provides a 32 bit CRC over the message.

Firewalls SHOULD look at outbound UDP packets and if they have the correct magic cookie they can classify them as STUN packets. Firewalls that desire fewer false positives MAY also check that the FINGERPRINT attribute is correct.

4.2. Policy decision

Once the firewall has received a STUN packet from inside the firewall, it needs to decide if the packet is acceptable. For most situations the firewall SHOULD accept all outbound STUN packets. This is similar to allowing all outbound TCP flows. Some firewalls may choose to look at other factors including the outside UDP port and the ORIGIN attribute in the STUN packet.

In general WebRTC media can be sent on a wide range of UDP ports but the two ports that are commonly used are the the RTP port (5004) and TURN port (3478). Some firewalls MAY choose to only allow flows

where the destination port on the outside of the firewall is one of these.

The STUN ORIGIN attribute [I-D.ietf-tram-stun-origin] carries the origin of the web page that caused the various STUN requests. So for example, if a browser was on a page such as example.com and that page used the WebRTC calls to set up a connection, the STUN request's ORIGIN attribute would include example.com. This allows the firewall to see the web application (in this case, example.com) that is requesting the pinhole be opened. The firewall MAY have a white list or black list for domains in STUN ORIGIN.

4.3. Creating the pinhole rules

Once a STUN packet is accepted, the firewall MUST create a temporary rule that allows incoming and outgoing packets for that 5-tuple for at least 5 seconds. If in that 5 seconds, a response is received to the STUN request, the lifetime of the rule must be extended to at least 30 seconds from last accepted STUN packet from inside the firewall. Once the rule has been extended to 30 seconds the first time, any additional UDP packets from inside the firewall MUST extend the lifetime of the rule by at least 30 seconds from the time that packet was received. The procedures in [I-D.ietf-rtcweb-stun-consent-freshness] will ensure that an outbound packet is sent at least every 30 seconds.

4.4. Tracking media vs data

WebRTC can send audio and video as well as carry a data channel. Confidential data could leave an enterprise by a video camera being pointed at a document, but IT departments are often more concerned about the data channel. It is easy for the firewall to separately track the amount of RTP media and non-media data for each WebRTC flow. If the first byte of the UDP message is 23, it is non-media data; if it is in the range 127 to 192 it is audio or video data. More information about this can be found in [I-D.ietf-avtcore-rfc5764-mux-fixes]. Network management systems on the firewall can track these two separately which can help identify unusual usage.

5. WebRTC Browsers

This specification would require browsers to include the FINGERPRINT and ORIGIN attributes in STUN for this to work correctly.

Open Issue: Does adding the ORIGIN reduce user privacy? Consider the following case. The user goes to https://facebook.com and initiates a call with another Facebook user. The domain facebook.com will

appear (unencrypted) in the STUN packets sent from the browser to Facebook's TURN server. Anyone along the network path could tell that the user is using Facebook's TURN server. However, when the original TLS connection for the HTTP was made, the Server Name Indication (SNI) in the TLS of the HTTPS connection also revealed facebook.com, largely for the same reasons - so that the firewall would be able to see which applications are using the network.

6. Blocking Media Hiding in HTTP

The IETF is designing systems to send interactive audio and video such that it looks like HTTPS and HTTP to the proxies and firewalls. The reasons for doing this is that sometimes the proxies and firewalls allow this to work when the mechanisms and channels designed for sending audio and video data have been explicitly disabled by the firewall administrators. Many firewall administrators feel this circumvents the policy they are trying to enforce and desire a way to prevent this. Any scheme for preventing this has some risk of impacting normal HTTP traffic, so there is a desire to provide guidance around ways to do that in this draft.

Any HTTP or HTTPS connection that sends more than 10 requests per second for longer than 10 seconds should be paused for 1 second, and any HTTP/S requests from that client's IP address in the 1 second pause time should be buffered or simply dropped. This strategy ensures there is no impact to clients other than the one exceeding the rate limit and minimizes the impact to other applications on the device while still reducing the incentive to try and run calls this way.

7. Deployment Advice

7.1. WebRTC Servers

WebRTC media servers and TURN servers with public IP address(es) that can receive incoming packets from anywhere on the Internet are suggested to listen for UDP on ports 53 (DNS), 123 (NTP), and 5004 for RTP media servers and 3478 for TURN servers. UDP destined for port 53 or 123 if often allowed by firewalls that otherwise block UDP.

7.2. Firewall Admins

Often the approach has been to lock down everything, so that all UDP is blocked. This simply causes applications to do things like embed the data in normal looking HTTP or HTTPS requests. Malware and viruses use similar approaches. Just turning off all UDP results in a poor user experience some of the time, which results in users

moving to applications and devices outside the firewall. The IT department loses the visibility into what is going on and can no longer protect its users when their computers become compromised. Allowing things that users want to use to work and monitoring them to detect when things have gone wrong is very valuable.

8. Design Consideration

8.1. Why not just use TCP?

TODO

9. Security Concerns

Enterprises have a range of concerns around WebRTC traffic traversal of the firewall. The major concerns that are raised include:

1. Unlike TCP, UDP does not have a connection where a device inside the firewall has confirmed that it wants to talk to the thing outside.

2. Incoming UDP pinholes allow out of band packets to be spoofed into connecting as there is no equivalent of a TCP sequence number to check.

3. UDP has been used by malware command and control protocols so we block it.

4. We do not want enable ways for data to be exfiltrated outside the firewall with no monitoring.

5. An encrypted data channel in WebRTC can be used to bring malware into the company.

6. An encrypted media or data channel in WebRTC can be used as a command and control channel for malware inside the firewall.

7. An encrypted data channel in WebRTC can be used by an outside attacker to exfiltrate private files from inside the firewall.

10. Alternate Approaches

10.1. Firewall Auth Tokens

[I-D.reddy-rtcweb-stun-auth-fw-traversal] attempts to solve a similar problem by proposing a new comprehension-optional FW-FLOWDATA STUN attribute as part of ICE Connectivity checks enabling the firewall to permit outgoing UDP flows across the firewall. FW-FLOWDATA STUN

provides necessary information, such as lifetime, and candidate information, enabling a firewall to apply the required policy rules. However, [I-D.reddy-rtcweb-stun-auth-fw-traversal] requires establishing shared keys across the firewall(s) and the WebRTC server for successfully verifying the authenticity of the FW-FLOWDATA information. In summary, we believe [I-D.reddy-rtcweb-stun-auth-fw-traversal] to have following short-comings

1. Requiring a tight coupling between the application server (WebRTC server) and firewall(s)

2. Requiring additional efforts for Firewall Admins within an enterprise to distribute and maintain the shared authentication keys needed to generate authentication tags for the FW-FLOWDATA attribute.

3. [I-D.reddy-rtcweb-stun-auth-fw-traversal] doesn’t apply for distributing keys across firewalls in different administrative domains.

10.2. Any Cast Whitelist

Deploying media or TURN servers on a single any-cast IP address also makes it easier for firewall administrators to whitelist the address. Concerns have been raised that two packets sent from the same host to a given any-cast address may get delivered to different servers. This is certainly possible in theory but in practice it does not seem be happen in limited experiments done so far.

11. Acknowledgements

Many thanks to Shaun Cooley and Alissa Cooper.

12. References

12.1. Normative References

[I-D.ietf-avtcore-rfc5764-mux-fixes]
Petit-Huguenin, M. and G. Salgueiro, "Multiplexing Scheme Updates for Secure Real-time Transport Protocol (SRTP) Extension for Datagram Transport Layer Security (DTLS)", draft-ietf-avtcore-rfc5764-mux-fixes-02 (work in progress), March 2015.

[I-D.ietf-tram-stun-origin]
Johnston, A., Uberti, J., Yoakum, J., and K. Singh, "An Origin Attribute for the STUN Protocol", draft-ietf-tram-stun-origin-05 (work in progress), February 2015.

12.2. Informative References

[I-D.ietf-rtcweb-overview]
Alvestrand, H., "Overview: Real Time Protocols for Browser-based Applications", draft-ietf-rtcweb-overview-14 (work in progress), June 2015.

[I-D.ietf-rtcweb-stun-consent-freshness]
Perumal, M., Wing, D., R, R., Reddy, T., and M. Thomson, "STUN Usage for Consent Freshness", draft-ietf-rtcweb-stun-consent-freshness-15 (work in progress), June 2015.

[I-D.reddy-rtcweb-stun-auth-fw-traversal]
Reddy, T., Perumal, M., and D. Wing, "STUN Extensions for Authenticated Firewall Traversal", draft-reddy-rtcweb-stun-auth-fw-traversal-00 (work in progress), July 2012.

Authors' Addresses

Cullen Jennings
Cisco

Email: fluffy@iii.ca

Suhas Nandakumar
Cisco

Email: snandaku@cisco.com

Jonathan Rosenberg
Cisco

Email: jdrosen@cisco.com

TRAM
Internet-Draft
Intended status: Standards Track
Expires: December 3, 2015

P. Martinsen
D. Wing
Cisco

June 1, 2015

STUN Traceroute
draft-martinsen-tram-stuntrace-01

Abstract

After a UDP protocol such as RTP determines a network path is experiencing problems, a traceroute is often useful to determine which router or which link is contributing to the problem. However, operating system traceroute commands follow a different path than the actual UDP flow which complicates troubleshooting. A superior method is shown which is absolutely path-congruent with the UDP protocol itself, works on IPv4 and IPv6, and does not require administrative privileges on most operating systems.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at http://datatracker.ietf.org/drafts/current/.

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."

This Internet-Draft will expire on December 3, 2015.

Table of Contents

1. Introduction

Traceroute [RFC1393] is a simple tool available on most operating systems and is popular to debug the network by simply getting round-trip time along each hop to a remote IP address. More advanced tools, such as MTR, provide more metrics such as packet loss and round trip time to each hop over several seconds or minutes.

To simplify network debugging when dealing with bi-directional real time media it is often useful to get as much information as possible regarding the network path. In this specification probe packets are sent using the same 5-tuple where (S)RTP media is flowing. This will provide the most accurate results, as probe packets sent on a different 5-tuple may take another path due to Equal-Cost Multipath (ECMP, [RFC2992]), policy-based routing, and similar techniques.

To avoid those problems, the probe packets need to be sent from the same socket and with the same DiffServ code point the normal (S)RTP

media packets. As shown in Appendix A, most operating systems can pass the ICMP "Time to Live Exceeded" error to the application, so the application can perform the diagnostics over that network path.

This specifications uses STUN [RFC5389] packets as probes. STUN packets are designed to be multiplexed together with RTP [RFC3550] (and SRTP [RFC3711]) and are unlikely to cause any "problems" for the (S)RTP receiver. To differentiate each hop count, classic traceroute uses different UDP port numbers (e.g., TTL=1 uses UDP port 55001, TTL=2 uses UDP port 55002, etc.). The mechanism described here uses the same UDP port number (so that the trace is path-congruent with the (S)RTP packets), and uses different length UDP packets to differentiate each hop count (e.g., TTL=1 uses length 501, TTL=2 uses length 502, etc.).

Using a technique based on ICMP replies avoids a forklift upgrade of the network to provide host applications with useful information. ICMP is already supported in most network and application stacks.

Additional network characteristics like MTU and bandwidth availability can be discovered by using [I-D.petithuguenin-behave-stun-pmtud] and [I-D.martinsen-tram-turnbandwidthprobe].

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. Overview of Operation

An application using (S)RTP to send and receive media like audio and video following the guidelines in [RFC4961] uses symmetric send and receive ports. The application opens one socket that it uses to both send and receive media on.

It is important to note that the functionality described here can be done on most OSes without any administrative privileges.

Figure 1 depicts the various components needed for this to work. The application opens up its media socket as it would in normal cases where media is to be sent and received. It also opens up a ICMP socket or installs an error listener on the media socket.

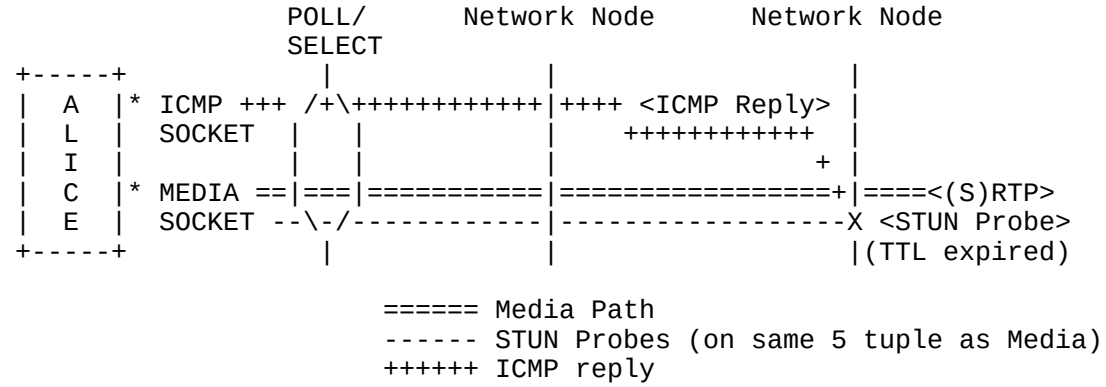


Figure 1

The application also need to listen on the sockets for any incoming ICMP packets or socket error messages. This is usually done with the socket calls select() or poll(). How to actually receive the ICMP messages will vary from OS to OS. See Appendix A for implementation details on various OSes.

Once the application have media running and is listening for ICMP replies it can start sending probes to detect networks nodes in the media path. This is done by sending STUN messages and setting the TTL/MAX_HOP limit in the IPv4/IPv6 header. Appendix A.1 explains how to set this on various platforms.

The STUN packet is sent on the same socket as the media packet are sent and received on. Mixing (S)RTP and STUN is well known behavior and should not cause any problems.

Along the path, every layer 3 network node (a.k.a. router) decreases the IPv4 TTL or IPv6 HOP_LIMIT field. If the field becomes 0 the network node responds with a ICMP error "Time to Live Exceeded" (TTL Exceeded) or "Hop Limit Exceeded in Transit" (Time Exceeded Message).

The application will receive a ICMP error in response to the offending probe packet. The source IP address of the ICMP packet will be the sending network node. This enables the application to trace the path towards the destination. The ICMP reply contains at least 8 bytes of the offending packet. The IP fragment of the offending packet in the ICMP reply can be used to determining if this ICMP reply actually was a reply to an offending packet the application did send out.

4. New STUN Attributes

This STUN extension defines the following new attribute:

0xXXX0: PATH-NODE-PROBE

4.1. PATH-NODE-PROBE

This attribute have a length of 8. Padding is needed to hit the required STUN 32 bit STUN attribute boundary.

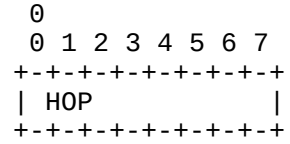


Figure 2: PATH-NODE-PROBE Attribute

The HOP field indicates what hop in the network path (relative to the application) the application is trying to learn the IP address of. This field should be set to the same value as the TTL/HOP_LIMIT field in the IPv4/IPv6 header of the probe packet leaving the application. Note that the TTL/HOP_LIMIT field in the IPv4/IPv6 header will decrease as the packet traverses the path. The HOP field in the attribute will remain unchanged.

This attribute is useful for clients when receiving the whole offending IP packet in the ICMP reply. The attribute will be reflected back in a STUN response if the remote application supports is. This makes it easier to correlate sent probe packets and ICMP responses.

5. Base Protocol Procedures

The procedures are simple; send a probe packet that may or may not trigger a reply from one of the nodes in the network path and then listen and parse any incoming replies. The reply might be an ICMP Time To Live Exceeded (from an intermediate hop), a STUN response (from the (S)RTP peer), or any other ICMP error message.

5.1. Forming STUN Packet Probes

To reduce chances of a STUN traceroute probe being stopped by various middle-boxes it is RECOMMENDED to use a STUN binding request as described in ICE [RFC5245].

Since the STUN packet can traverse the whole media-path and reach the remote peer it is RECOMMENDED the agent follows the guidelines for sending connectivity checks defined in ICE [RFC5245]. Adding a USERNAME attribute and integrity protecting the STUN message enables the remote peer to authenticate the STUN message and create an appropriate response. If the remote peer is unable to authenticate the STUN request it will not send any response. Getting a response from the remote peer is useful as it is an indication the probe have traveled the whole network path.

When forming the STUN packet probe the agent SHOULD add the PATH-NODE-PROBE attribute and MAY add a PADDING attribute as described in [RFC5780] Section 7.6. The PATH-NODE-PROBE attribute is useful for STUN servers receiving the STUN probe and it can be used to correlate any ICMP replies if the reply contains the complete offending packet. Adding the PADDING attribute is useful for clients that needs to have several outstanding probe packets on the same 5-tuple. The length of the offending packet reported back in any ICMP reply will make it possible to correlate this to the correct probe.

The agent sending the STUN packet probe MUST store the length of the UDP packet (as reported in the IP header) containing the STUN probe.

Before sending the probe on the wire it is important to set the appropriate TTL or HOP_LIMIT field in the IPv4 or IPv6 header before the packet is sent. How to do this on various OSes are described in Appendix A.1.

The probe MUST also be sent with the same DSCP value as the (S)RTP packets. This is normally not a problem as the STUN probes and (S)RTP packets are sent on the same socket.

5.2. Receiving a STUN Packet Probe

An agent that listens for STUN requests (a.k.a STUN server) that receives a STUN request with a PATH-NODE-PROBE attribute, MUST include a PATH-NODE-PROBE attribute with the same value in the generated response.

Any PADDING attributes as defined in [RFC5780] SHOULD be ignored by the STUN server.

5.3. Receiving ICMP Messages

After an agent sends a STUN probe it must be ready to receive a ICMP reply or a STUN reply. Details on how to do this on various OSes are described in Appendix A.2.

To prevent ICMP spoofing attacks [RFC5927] , the received ICMP packet MUST be validated by port number and length in the IP fragment of the offending packet contained in the ICMP payload. Port number validation checks that the port number in the offending IP fragment of the probe packet contained in the ICMP payload corresponds to the (S)RTP media (and STUN probe) 5-tuple. The length validation checks IP packet length field in the IP fragment of the offending packet received in the ICMP reply. This value MUST correspond to any length stored when the agent sent the STUN probe. If the agent uses the PADDING (Defined in [RFC5780]) attribute to generate different length on the STUN probes it is possible to have several outstanding probes, thus speeding up the trace.

6. IPv4 and IPv6 Differences

Core functionality is the same. In IPv6 the IPv4 TTL field is renamed to HOP_LIMIT to better reflect what it actually represent.

7. IANA Considerations

The code-point for the new STUN attribute defined in this specification is described in Section 4.

8. Security Considerations

ICMP messages does leak network topology, which is a well-known threat to networks and mitigations have long existed in routers and firewalls so that networks can be configured to not leak this topology information beyond their borders.

ICMP spoofing and DOS attack prevention exist in routers deployed on the Internet today.

No new threats have been added in this specification.

9. Acknowledgements

Trond Andersen for actually implementing this and Wilson Chen for helping out with different OS behavior testing.

10. References

10.1. Normative References

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

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

[RFC3711] Baugher, M., McGrew, D., Naslund, M., Carrara, E., and K. Norrman, "The Secure Real-time Transport Protocol (SRTP)", RFC 3711, March 2004.

[RFC4961] Wing, D., "Symmetric RTP / RTP Control Protocol (RTCP)", BCP 131, RFC 4961, July 2007.

[RFC5245] Rosenberg, J., "Interactive Connectivity Establishment (ICE): A Protocol for Network Address Translator (NAT) Traversal for Offer/Answer Protocols", RFC 5245, April 2010.

[RFC5389] Rosenberg, J., Mahy, R., Matthews, P., and D. Wing, "Session Traversal Utilities for NAT (STUN)", RFC 5389, October 2008.

[RFC5780] MacDonald, D. and B. Lowekamp, "NAT Behavior Discovery Using Session Traversal Utilities for NAT (STUN)", RFC 5780, May 2010.

10.2. Informative References

[I-D.martinsen-tram-turnbandwidthprobe] Martinsen, P., Andersen, T., Salgueiro, G., and M. Petit-Huguenin, "Traversal Using Relays around NAT (TURN) Bandwidth Probe", draft-martinsen-tram-turnbandwidthprobe-00 (work in progress), May 2015.

[I-D.petithuguenin-behave-stun-pmtud] Petit-Huguenin, M., "Path MTU Discovery Using Session Traversal Utilities for NAT (STUN)", draft-petithuguenin-behave-stun-pmtud-03 (work in progress), March 2009.

[ICMPTest] "ICMP test github repo", <https://github.com/palerikm/ICMPTest/>.

[RFC1393] Malkin, G., "Traceroute Using an IP Option", RFC 1393, January 1993.

[RFC2992] Hopps, C., "Analysis of an Equal-Cost Multi-Path Algorithm", RFC 2992, November 2000.

[RFC5927] Gont, F., "ICMP Attacks against TCP", RFC 5927, July 2010.

Appendix A. Platform Implementation Details

This section provides examples and hint on how probe packets can be sent and ICMP messages received on various OSes. For a complete example please refer to [ICMPTest].

A.1. Setting TTL or HOP_LIMIT on Probes

Setting the appropriate value in the IPv4 or IPv6 header is the same for most platforms. Use

```
setsockopt(sockHandle, IPPROTO_IP, IP_TTL, &sock_ttl,
sizeof(sock_ttl));
```

for IPv4 or

```
setsockopt(sockHandle,
         IPPROTO_IPV6, IPV6_UNICAST_HOPS, &sock_ttl,
         sizeof(sock_ttl));
```

for IPv6.

Sending the probes on the same socket as media is flowing requires the implementations to only set this when sending the probe packet. Remember to set it back to initial value when sending media. Most OSes seems to handle the setsockopt call correctly and not set the value in the IP header of any buffered packets.

A.2. Receiving ICMP Messages

A.2.1. OS-X and iOS

Creating a socket to listen for incoming ICMP messages can be done as:

```
icmpSocket=socket(config.remoteAddr.ss_family, SOCK_DGRAM,
                  IPPROTO_ICMP); <<<
```

This is done in addition to the normal socket used to send media on (RTP) and probes. (Yes, even if the probe are sent on the media socket the ICMP reply will be on the ICMP sockets..)

Code in the while(1) loop of poll would look something like:

```
       for(i=0;i<numSockets;i++){
           if (ufds[i].revents & POLLIN) {
               if(i == rtpSock){
                   //Handle "normal" data here.
               }
               if(i == icmpSock){//This is the ICMP socket
                  //Handle ICMP packets here.
               }
           }
       }
```

A.2.2. Linux and Android

For unprivileged recipient of the ICMP messages an error handler must be installed. This can be done like:

```
   setsockopt (config.sockfd, SOL_IP,
               IP_RECVERR, &val, sizeof(val)) < 0);
```

In the poll() section of the code something like this needs to be there:

```
       struct msghdr msg;

       if (ufds[dataSock].revents & POLLERR) {
           if (recvmsg(sockfd, &msg, MSG_ERRQUEUE ) == -1) {
               //Ignore for now. Will get it later..
               continue;
           }
           //possible ICMP message
           //use cmsg to read the structures in msg
       }
```

Failing to call rcvmsg seems to let the msg fall through to the kernel. Looks like it will close down the socket because of the received error. So be careful!

For application with the right administrative privileges it is possible create a separate ICMP listen socket as described in the previous section. The socket() call would then look like:

```
               icmpSocket=socket(config.remoteAddr.ss_family, SOCK_RAW,
               IPPROTO_ICMP);
```

The poll() loop will be as described for OS-X and iOS. No need for a error handler.

A.2.3. Windows

The following code in select() or poll() will read and detect any incoming ICMP messages on the send socket.

```
            if (FD_ISSET(sendsocket, &read_flags)) {
                cc = recvfrom(sendsocket, receivepacket,
                    sizeof(receivepacket), 0,
                    (struct sockaddr *)&receiveaddr, (int*)&fromlen);
                if (cc < 0 && GETERRORCODE == WSAENETRESET) {
                   //ICMP packet handling here
                   //Do:
                   //inet_ntoa(receiveaddr.sin_addr));
                   //to get the address of the router sending the
                   //ICMP reply
                }
            }
```

Authors' Addresses

Paal-Erik Martinsen
Cisco Systems, Inc.
Philip Pedersens Vei 22
Lysaker, Akershus 1325
Norway

Email: palmarti@cisco.com

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

Email: dwing@cisco.com

TRAM
Internet-Draft
Intended status: Informational
Expires: November 30, 2015

P. Martinsen
T. Andersen
G. Salgueiro
Cisco
M. Petit-Huguenin
Impedance Mismatch

May 29, 2015

Traversal Using Relays around NAT (TURN) Bandwidth Probe draft-martinsen-tram-turnbandwidthprobe-00

Abstract

Performing pre-call probing to discover a reasonable value for the available bandwidth, is useful information that can be utilized by bandwidth sensitive or bandwidth intensive network devices (e.g., video encoders). The method described herein is intended to produce an initial bandwidth value. Applications using this mechanism should also employ appropriate rate adaptation techniques. In addition to bandwidth, latency and bufferbloat can also be measured. No modification is needed on the server side.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at http://datatracker.ietf.org/drafts/current/.

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."

This Internet-Draft will expire on November 30, 2015.

Copyright Notice

Table of Contents

1. Introduction

When Interactive Connectivity Establishment (ICE) [RFC5245] and Traversal Using Relays around NAT (TURN) [RFC5766] are used by an endpoint as a firewall/NAT traversal mechanism, the TURN relay can also be used to measure bandwidth and latency prior to call setup.

In normal ICE behavior the client first sends a message (allocate request) to the TURN server to allocate a RELAY address. This address can be used by the endpoint to receive media from other endpoints. The media stream is then received by the TURN server and then relayed back to the endpoint behind the firewall/NAT. For security reasons the endpoint must first set the correct permissions on the TURN server to only allow media from remote participants it wants to communicate with (i.e., addresses taken from the Session Initiation Protocol (SIP) [RFC3261] Session Description Protocol (SDP) [RFC4566] offer/answer exchange [RFC3264]) . The endpoint will also learn its reflexive address on the firewall/NAT when talking to the TURN server.

Combining this with a TCP transfer on the same TURN server can be used to also measure bufferbloat, an important metric for multimedia applications.

Note that only the maximum bandwidth, maximum latency and maximum bufferbloat of the aggregation of both uplink and downlink can be measured. It is not possible with this technique to get the metrics of only one. For most multimedia applications using TURN that is not an issue as they are generally symmetrical, but some other use cases (like conferencing) may need other techniques to measure these metrics separately.

No modification to the TURN server is necessary.

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. Overview of Operation

Prior to the call (upon registering with the call control server, receiving a configuration, loading application, or a similar event) the endpoint can measure bandwidth and latency between the endpoint and the TURN server.

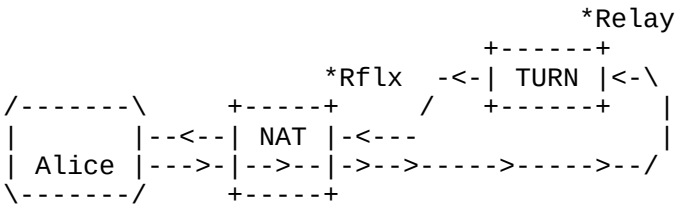


Figure 1

The agent allocates a TURN Relay port on its designated TURN server as described in TURN RFC [RFC5766]. In the process the agent will also learn the outermost NAT address. This is called a reflexive address (Rflx). For more information see Section 2.1 of the TURN RFC regarding candidate gathering in ICE.

The agent must set the permissions on the allocated RELAY port as described in Section 8 of the TURN RFC to allow traffic from the discovered reflexive address.

When sending packets to the allocated RELAY port on the TURN server, the packets will be forwarded back to the agent in a data indication packet. See Section 10 of the TURN RFC for details on how the TURN server can relay packets back to the allocating agent. Available bandwidth can be measured by sending varying number of packets and detecting the amount of packet loss. Each packet sent affects both upstream and downstream links.

To make it easier to calculate the available bandwidth a TIMESTAMP attribute is defined in this document (see Section Section 6.1) and can be added to the Session Traversal Utilities for NAT (STUN) [RFC5389] probe packets. The PADDING attribute from the NAT Behavior Discovery Using STUN RFC [RFC5780] can be used to vary the packet size.

Discovering the MTU and network path (using the STUN-PMTUD [I-D.petithuguenin-tram-stun-pmtud] and STUN Traceroute [I-D.martinsen-tram-stuntrace] mechanisms) can also be performed when probing for the bandwidth available between the client and the TURN server.

4. Base Protocol Procedures

In order to perform the STUN bandwidth probing mechanism described in this document, the client MUST take the following general steps (explained in greater detail in the following subsections).

- Allocate TURN RELAY address
- Set correct permissions on the allocated TURN RELAY address
- Originating client sends data to itself through the TURN server and measures bandwidth throughput and latency

When initiating a bandwidth probe it is important to not do so when a device powers up or some similar initiating events. If a power failure has happened and all devices within an area are rebooted concurrently the bandwidth probing of all the devices can have a DDOS-like effect. Measures should be taken to avoid such scenarios (e.g., random delays to initiate bandwidth probing, etc).

Discovery of the TURN server as well as the determination of what TURN server to uses is entirely at the discretion of the client and outside the scope of this document. A client MUST be prepared to be

redirected to another TURN server if it receives an ALTERNATE-SERVER response.

While allocating the TURN RELAY port the client will learn its outermost NAT address or reflexive address. This is the address the TURN server will receive the bandwidth probing packets from.

The bandwidth mechanism can use either a UDP transport or a TCP. Secure transports (i.e. TLS or DTLS) may be used to discover if an intermediary network element tries to process flows differently when they are secured.

4.1. UDP Procedures

The client allocates a TURN RELAY port as described in the TURN RFC. The client then use a CreatePermission request with the obtained reflexive address encoded in a XOR-PEER-ADDRESS attribute as described in Section 9.1 of the TURN RFC.

It is recommended to create a TURN channel as soon as possible to lower the overhead of the packets exchanged.

If the transport address used to send the UDP packets to the TURN relay is identical to the transport address used to create the TURN allocation, then a TURN Channel can be created immediately by using the reflexive transport address learned during the Allocate.

If not, the TURN Channel can be created as soon the first Data indication is received.

The client can then send UDP packets to the relay transport address and receive then over the TURN Channel.

Immediately after this the client can send UDP packets over the TURN channel and receive them directly, as an additional way of averaging the impact of the difference of encapsulation for the packets. Note that the client still need to periodically send packets over the TURN Channel to persist eventual NAT bindings.

Note that the client cannot use a TCP transport to the server with a UDP allocation because there would be no way to retrieve the UDP reflexive address for the CreatePermission request.

4.2. TCP Procedures

The client allocates a TURN RELAY port as described in TURN Extensions for TCP Allocations [RFC6062]. The client then use a CreatePermission request with the obtained reflexive address encoded

in a XOR-PEER-ADDRESS attribute as described in Section 9.1 of the TURN RFC.

The client then establishes a TCP connection to the relay transport address. The client will receive a ConnectAttempt indication that will trigger a new TCP connection to the TURN server, and the sending of a ConnectBind.

After completion of this procedure, data sent over the direct TCP connection will be received over the bound TURN connection, and vice-versa, although there is no difference of overhead in that case.

4.3. Sending Data to Measure Available Bandwidth and Latency

The specific calculation and measurement of the bandwidth is client dependent and implementation-specific and is thus outside the scope of this document.

If the client want to use STUN packets as the basis for the probing packets, then a TIMESTAMP attribute is defined in this specification (see Section Section 6.1) to simplify measurement of round-trip time (RTT) and available bandwidth. A PADDING attribute is already defined in RFC 5780 [RFC5780] that makes it easy to vary the size of the STUN probing packet.

The probing packet will be sent upstream to the TURN server and later received downstream from the TURN server. Available bandwidth would typically be determined to be the lowest of the bandwidth values calculated for the upstream and downstream directions.

If the RTP [RFC3550] loop-back mechanism described in RFC 6849 [RFC6849] is in use the method described here can extend the use-cases mentioned in RFC6849 Section 1.1 to enable the "loopback source" and "loopback mirror" to be running on the same device. Using RTP would permit to reuse the standards RTP tools for calculating latency, jitter and other metrics. It may also permit to get better results if some intermediary network element has preferential treatment for media packets.

The client should take care to reuse the same congestion control mechanisms it uses when sending media to avoid unnecessary strain on the network.

5. IANA Considerations

This specification defines a new STUN attribute. IANA added this new attribute to the STUN Attributes sub-registry of the Session

Traversal Utilities for NAT (STUN) Parameters registry. (This is still an ID draft so not assignment yet)

6. New STUN Attribute

This STUN extension defines the following new attribute:

0xXXX0: TIMESTAMP

6.1. TIMESTAMP

The TIMESTAMP attribute has a length of 80 bits. Padding is needed to hit the required 32 bit STUN attribute boundary.

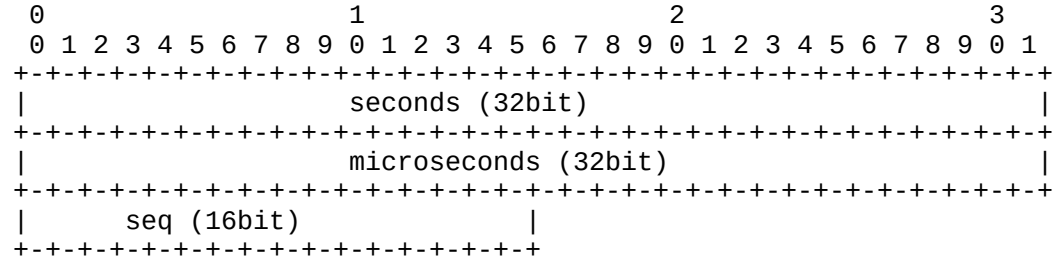


Figure 2: TIMESTAMP Attribute

The seconds and microseconds fields reflect what would be returned in the struct timeval when calling getTimeofDay() function. Note that the size of that struct may vary based on platform, but 32 bits is more than sufficient to obtain the required accuracy for the feature described in this document. It is RECOMMENDED to initialize these fields with a random value that later can be subtracted to get the right timing.

The seq field is a 16 bit sequence number. It is increased by one for each bandwidth probe STUN packet sent. It is RECOMMENDED to choose a random starting value.

7. Implementation Status

[[Note to RFC Editor: Please remove this section and the reference to [RFC6982] before publication.]]

This section records the status of known implementations of the protocol defined by this specification at the time of posting of this Internet-Draft, and is based on a proposal described in RFC 6982

[RFC6982]. The description of implementations in this section is intended to assist the IETF in its decision processes in progressing drafts to RFCs. Please note that the listing of any individual implementation here does not imply endorsement by the IETF. Furthermore, no effort has been spent to verify the information presented here that was supplied by IETF contributors. This is not intended as, and must not be construed to be, a catalog of available implementations or their features. Readers are advised to note that other implementations may exist.

According to RFC 6982 [RFC6982], "this will allow reviewers and working groups to assign due consideration to documents that have the benefit of running code, which may serve as evidence of valuable experimentation and feedback that have made the implemented protocols more mature. It is up to the individual working groups to use this information as they see fit"

7.1. Cisco Collaboration Endpoint (CE)

Organization: Cisco

Name: Cisco Collaboration Endpoints (CE) software

Description: Hard video endpoint part of the Cisco collaboration portfolio

Level of maturity: In released products

Coverage Implementation of base procedures of the functionality described in this specification

Licensing: Proprietary

Implementation experience: Straight forward, but implementation was don prior to writing up the spec

Contact: Paal-Erik Martinsen (palmarti@cisco.com)

8. Security Considerations

When setting permissions this is done on a per IP address basis. Port number is not part of the permission. This is necessary limitation of the TURN protocol [RFC5766] and not something introduced by this specification.

To prevent replay attacks or other attacks that rely on static sequence number initialization it is important to randomly initialize the seq number in the TIMESTAMP Attribute. Likewise it is important

to hide the time information by assigning a random value to the seconds and microseconds fields. That random value can be added and subtracted by the client when sending and receiving packets to get the correct value. This prevents any information leakage regarding time from the client.

9. Acknowledgements

Thanks to Dan Wing for input.

10. References

10.1. Normative References

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

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

[RFC5245] Rosenberg, J., "Interactive Connectivity Establishment (ICE): A Protocol for Network Address Translator (NAT) Traversal for Offer/Answer Protocols", RFC 5245, April 2010.

[RFC5389] Rosenberg, J., Mahy, R., Matthews, P., and D. Wing, "Session Traversal Utilities for NAT (STUN)", RFC 5389, October 2008.

[RFC5766] Mahy, R., Matthews, P., and J. Rosenberg, "Traversal Using Relays around NAT (TURN): Relay Extensions to Session Traversal Utilities for NAT (STUN)", RFC 5766, April 2010.

[RFC5780] MacDonald, D. and B. Lowekamp, "NAT Behavior Discovery Using Session Traversal Utilities for NAT (STUN)", RFC 5780, May 2010.

[RFC6062] Perreault, S. and J. Rosenberg, "Traversal Using Relays around NAT (TURN) Extensions for TCP Allocations", RFC 6062, November 2010.

[RFC6849] Kaplan, H., Hedayat, K., Venna, N., Jones, P., and N. Stratton, "An Extension to the Session Description Protocol (SDP) and Real-time Transport Protocol (RTP) for Media Loopback", RFC 6849, February 2013.

[RFC6982] Sheffer, Y. and A. Farrel, "Improving Awareness of Running Code: The Implementation Status Section", RFC 6982, July 2013.

10.2. Informative References

[I-D.martinsen-tram-stuntrace]
Martinsen, P. and D. Wing, "STUN Traceroute", draft-martinsen-tram-stuntrace-00 (work in progress), February 2015.

[I-D.petithuguenin-tram-stun-pmtud]
Petit-Huguenin, M., "Path MTU Discovery Using Session Traversal Utilities for NAT (STUN)", draft-petithuguenin-tram-stun-pmtud-00 (work in progress), January 2015.

[RFC3261] Rosenberg, J., Schulzrinne, H., Camarillo, G., Johnston, A., Peterson, J., Sparks, R., Handley, M., and E. Schooler, "SIP: Session Initiation Protocol", RFC 3261, June 2002.

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

[RFC4566] Handley, M., Jacobson, V., and C. Perkins, "SDP: Session Description Protocol", RFC 4566, July 2006.

Authors' Addresses

Paal-Erik Martinsen
Cisco Systems, Inc.
Philip Pedersens Vei 22
Lysaker, Akershus 1325
Norway

Email: palmarti@cisco.com

Trond Andersen
Cisco Systems, Inc.
Philip Pedersens Vei 22
Lysaker, Akershus 1325
Norway

Email: trondand@cisco.com

Gonzalo Salgueiro
Cisco Systems, Inc.
7200-12 Kit Creek Road
Research Triangle Park, NC 27709
US

Email: gsalguei@cisco.com

Marc Petit-Huguenin
Impedance Mismatch

Email: marc@petit-huguenin.org

TRAM
Internet-Draft
Intended status: Standards Track
Expires: January 7, 2016

M. Petit-Huguenin
Impedance Mismatch
G. Salgueiro
Cisco

July 6, 2015

Path MTU Discovery Using Session Traversal Utilities for NAT (STUN) draft-petithuguenin-tram-stun-pmtud-01

Abstract

This document describes a Session Traversal Utilities for NAT (STUN) usage for Path MTU Discovery (PMTUD) between a client and a server.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at http://datatracker.ietf.org/drafts/current/.

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."

This Internet-Draft will expire on January 7, 2016.

Copyright Notice

Table of Contents

1. Introduction

The Packetization Layer Path MTU Discovery specification [RFC4821] describes a method to discover the path MTU but does not describe a practical protocol to do so with UDP.

This document only describes how probing mechanisms are implemented with Session Traversal Utilities for NAT (STUN). The algorithm to find the path MTU is described in [RFC4821].

The STUN usage defined in this document for Path MTU Discovery (PMTUD) between a client and a server simplifies troubleshooting and has multiple applications across a wide variety of technologies.

Additional network characteristics like the network path (using the STUN Traceroute mechanism described in [I-D.martinsen-tram-stuntrace]) and bandwidth availability (using the mechanism described in [I-D.martinsen-tram-turnbandwidthprobe]) can be discovered using complementary techniques.

2. Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119]. When these words are not in ALL CAPS (such as "must" or "Must"), they have their usual English meanings, and are not to be interpreted as RFC 2119 key words.

3. Probing Mechanisms

A client MUST NOT send a probe if it does not have knowledge that the server supports this specification. This is done by an external mechanism specific to each UDP protocol. Section 6 describes some of this mechanisms.

The probe mechanism is used to discover the path MTU in one direction only, from the client to the server.

Two probing mechanisms are described, a simple probing mechanism and a more complete mechanism that can converge quicker.

The simple probing mechanism is implemented by sending a Probe Request with a PADDING [RFC5780] attribute and the DF bit set over UDP. A router on the path to the server can reject this request with an ICMP message or drop it. The client SHOULD cease retransmissions after 3 missing responses.

The complete probing mechanism is implemented by sending one or more Probe Indication with a PADDING attribute and the DF bit set over UDP then a Report Request to the same server. A router on the path to the server can reject this indication with an ICMP message or drop it. The server keeps a time ordered list of identifiers of all packets received (including retransmitted packets) and sends this

list back to the client in the Report Response. The client analyzes this list to find which packets were not received. Because UDP packets does not contain an identifier, the complete probing mechanism needs a way to identify each packet received. While there are other possible packet identification schemes, this document describes two different ways to identify a specific packet.

In the first packet identifier mechanism, the server computes a checksum over each packet received and sends back to the sender the ordered list of checksums. The client compares this list to its own list of checksums.

In the second packet identifier mechanism, the client adds a sequential number in front of each UDP packet sent. The server sends back the ordered list of sequential numbers received that the client compares to its own list

4. Simple Probing Mechanism

4.1. Sending a Probe Request

A client forms a Probe Request by following the rules in Section 7.1 of [RFC5389]. No authentication method is used. The client adds a PADDING [RFC5780] attribute with a length that, when added to the IP and UDP headers and the other STUN components, is equal to the Selected Probe Size, as defined in [RFC4821] section 7.3. The client MUST add the FINGERPRINT attribute.

Then the client sends the Probe Request to the server over UDP with the DF bit set. The client SHOULD stop retransmitting after 3 missing responses.

4.2. Receiving a Probe Request

A server receiving a Probe Request MUST process it as specified in [RFC5389]. The server MUST NOT challenge the client.

The server then creates a Probe Response. The server MUST add the FINGERPRINT attribute. The server then sends the response to the client.

4.3. Receiving a Probe Response

A client receiving a Probe Response MUST process it as specified in [RFC5389]. If a response is received this is interpreted as a Probe Success as defined in [RFC4821] section 7.6.1. If an ICMP packet "Fragmentation needed" is received then this is interpreted as a Probe Failure as defined in [RFC4821] section 7.6.2. If the Probe

transactions fails in timeout, then this is interpreted as a Probe Inconclusive as defined in [RFC4821] section 7.6.4.

5. Complete Probing Mechanism

5.1. Sending the Probe Indications and Report Request

A client forms a Probe Indication by following the rules in [RFC5389] section 7.1. The client adds to the Probe Indication a PADDING attribute with a size that, when added to the IP and UDP headers and the other STUN components, is equal to the Selected Probe Size, as defined in [RFC4821] section 7.3. The client MUST add the FINGERPRINT attribute.

Then the client sends the Probe Indication to the server over UDP with the DF bit set.

Then the client forms a Report Request by following the rules in [RFC5389] section 7.1. No authentication method is used. The client MUST add the FINGERPRINT attribute.

Then the client waits half the RTO if it is known or 50 milliseconds after sending the Probe Indication and sends the Report Request to the server over UDP.

5.2. Receiving an ICMP packet

If an ICMP packet "Fragmentation needed" is received then this is interpreted as a Probe Failure as defined in [RFC4821] section 7.5.

5.3. Receiving a Probe Indication and Report Request

A server supporting this specification and knowing that the client also supports it will keep the identifiers of all packets received in a list ordered by receiving time. The same identifier can appear multiple times in the list because of retransmission. The maximum size of this list is calculated so that when the list is added to the Report Response, the total size of the packet does not exceed the unknown path MTU as defined in [RFC5389] section 7.1. Older identifiers are removed when new identifiers are added to a list already full.

A server receiving a Report Request MUST process it as specified in [RFC5389]. The server MUST NOT challenge the client.

The server creates a Report Response and adds an IDENTIFIERS attribute that contains the list of all identifiers received so far.

The server MUST add the FINGERPRINT attribute. The server then sends the response to the client.

5.4. Receiving a Report Response

A client receiving a Report Response processes it as specified in [RFC5389]. If the response IDENTIFIERS attribute contains the identifier of the Probe Indication, then this is interpreted as a Probe Success for this probe as defined in [RFC4821] Section 7.5. If the Probe Indication identifier cannot be found in the Report Response, this is interpreted as a Probe Failure as defined in [RFC4821] Section 7.5. If the Probe Indication identifier cannot be found in the Report Response but other packets identifier sent before or after the Probe Indication cannot also be found, this is interpreted as a Probe Inconclusive as defined in [RFC4821] Section 7.5. If the Report Transaction fails in timeout, this is interpreted as a Full-Stop Timeout as defined in [RFC4821] Section 3.

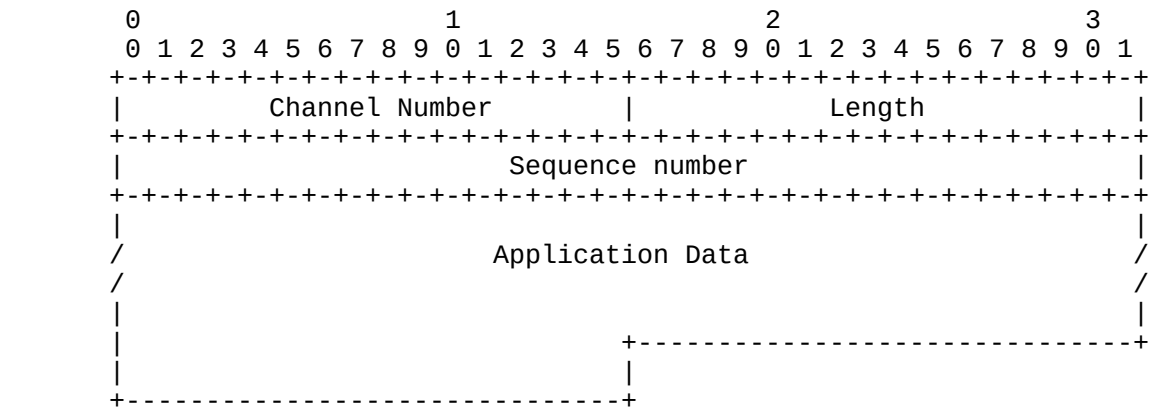
5.5. Using Checksum as Packet Identifiers

When using checksum as packet identifiers, the client calculate the checksum for each packet sent over UDP and keep this checksum in an ordered list. The server does the same thing and send back this list in the Report Response.

It could have been possible to use the checksum generated in the UDP checksum for this, but this value is generally not accessible to applications. Also sometimes the checksum is not calculated or off-loaded to the network card.

5.6. Using Sequential Numbers as Packet Identifiers

When using sequential numbers, a small header similar to the TURN ChannelData header is added in front of all non-STUN packets. The sequential number is incremented for each packet sent. The server collects the sequence number of the packets sent.



The Channel Number is always 0xFFFF.

6. Probe Support Discovery Mechanisms

6.1. Implicit Mechanism

An endpoint acting as a client for the STUN usage described in this specification MUST also act as a server for this STUN usage. This means that a server receiving a probe can assumes that it can acts as a client to discover the path MTU to the IP address and port from which it received the probe.

6.2. Probe Support Discovery with TURN

A TURN client supporting this STUN usage will add a PMTUD-SUPPORTED attribute to the Allocate Request sent to the TURN server. The TURN server can immediately start to send probes to the TURN client on reception of an Allocation Request with a PMTUD-SUPPORTED attribute. The TURN client will then use the Implicit Mechanism described above to send probes.

6.3. Probe Support Discovery with ICE

An ICE [RFC5245] client supporting this STUN usage will add a PMTUD-SUPPORTED attribute to the Binding Request sent during a connectivity check. The ICE server can immediately start to send probes to the ICE client on reception of a Binding Request with a PMTUD-SUPPORTED attributed. Local candidates receiving Binding Request with the PMTUD-SUPPORTED flag must not start PMTUD with the remote candidate if already done so. The ICE client will then use the Implicit Mechanism described above to send probes.

7. New STUN Method

This specification defines the following new STUN methods:

0x801 : Probe

0x802 : Report

8. New STUN Attributes

This specification defines the following new STUN attributes:

0x4001 : IDENTIFIERS

0xC001 : PMTUD-SUPPORTED

8.1. IDENTIFIERS

The IDENTIFIERS attribute is used in Report Response. It contains a list of UDP packet identifiers.

8.2. PMTUD-SUPPORTED

The PMTUD-SUPPORTED attribute is used in STUN usages and extensions to signal the support of this specification. This attribute has no content.

9. Security Considerations

TBD

10. IANA Considerations

TBD

11. Acknowledgements

Thanks to Eilon Yardeni, Geir Sandbakken and Paal-Erik Martinsen for their review comments, suggestions and questions that helped to improve this document.

Special thanks to Dan Wing, who supported this document since its first publication back in 2008.

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.

[RFC4821] Mathis, M. and J. Heffner, "Packetization Layer Path MTU Discovery", RFC 4821, March 2007.

[RFC5389] Rosenberg, J., Mahy, R., Matthews, P., and D. Wing, "Session Traversal Utilities for NAT (STUN)", RFC 5389, October 2008.

[RFC5245] Rosenberg, J., "Interactive Connectivity Establishment (ICE): A Protocol for Network Address Translator (NAT) Traversal for Offer/Answer Protocols", RFC 5245, April 2010.

12.2. Informative References

[I-D.martinsen-tram-stuntrace] Martinsen, P. and D. Wing, "STUN Traceroute", draft-martinsen-tram-stuntrace-01 (work in progress), June 2015.

[I-D.martinsen-tram-turnbandwidthprobe] Martinsen, P., Andersen, T., Salgueiro, G., and M. Petit-Huguenin, "Traversal Using Relays around NAT (TURN) Bandwidth Probe", draft-martinsen-tram-turnbandwidthprobe-00 (work in progress), May 2015.

[I-D.ietf-payload-flexible-fec-scheme] Singh, V., Begen, A., and M. Zanaty, "RTP Payload Format for Non-Interleaved and Interleaved Parity Forward Error Correction (FEC)", draft-ietf-payload-flexible-fec-scheme-00 (work in progress), February 2015.

[RFC5780] MacDonald, D. and B. Lowekamp, "NAT Behavior Discovery Using Session Traversal Utilities for NAT (STUN)", RFC 5780, May 2010.

[RFC6982] Sheffer, Y. and A. Farrel, "Improving Awareness of Running Code: The Implementation Status Section", RFC 6982, July 2013.

Appendix A. Release Notes

This section must be removed before publication as an RFC.

A.1. Modifications between draft-petithuguenin-tram-stun-pmtud-01 and draft-petithuguenin-tram-stun-pmtud-00

- Moved some Introduction text to the Probing Mechanism section.
- Added cross-reference to the other two STUN troubleshooting mechanism drafts.
- Updated references.
- Added Gonzalo Salgueiro as co-author.

A.2. Modifications between draft-petithuguenin-tram-stun-pmtud-00 and draft-petithuguenin-behave-stun-pmtud-03

- General refresh for republication.

A.3. Modifications between draft-petithuguenin-behave-stun-pmtud-03 and draft-petithuguenin-behave-stun-pmtud-02

- Changed author address.
- Changed the IPR to trust200902.

A.4. Modifications between draft-petithuguenin-behave-stun-pmtud-02 and draft-petithuguenin-behave-stun-pmtud-01

- Replaced the transactions identifiers by packet identifiers
- Defined checksum and sequential numbers as possible packet identifiers.
- Updated the reference to RFC 5389
- The FINGERPRINT attribute is now mandatory.
- Changed the delay between Probe indication and Report request to be RTO/2 or 50 milliseconds.
- Added ICMP packet processing.
- Added Full-Stop Timeout detection.

- Stated that Binding request with PMTUD-SUPPORTED does not start the PMTUD process if already started.

A.5. Modifications between draft-petithuguenin-behave-stun-pmtud-01 and draft-petithuguenin-behave-stun-pmtud-00

- Removed the use of modified STUN transaction but shorten the retransmission for the simple probing mechanism.
- Added a complete probing mechanism.
- Removed the PADDING-RECEIVED attribute.
- Added release notes.

Authors' Addresses

Marc Petit-Huguenin
Impedance Mismatch

Email: marc@petit-huguenin.org

Gonzalo Salgueiro
Cisco Systems, Inc.
7200-12 Kit Creek Road
Research Triangle Park, NC 27709
United States

Email: gsalguei@cisco.com

tram
Internet-Draft
Intended status: Standards Track
Expires: January 1, 2016

A. Wang
China Telecom
B. Liu
Huawei Technologies

July 2, 2015

A Lightweight TURN Architecture and Specification (TURN-Lite) draft-wang-tram-turnlite-03

Abstract

This document proposes a new relay-based NAT traversal architecture called TURN-Lite which could simplify the data communication process between two hosts that locates behind some non-BEHAVE compliant [RFC4787] [RFC5382] NAT devices. The key mechanism in TURN-Lite is the newly defined "Couple" operation (using STUN [RFC5389] message format) which allows the TURN-Lite servers to be easily incorporated into existing CGN devices/CDN nodes which are already deployed within the network in a distributed manner.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at http://datatracker.ietf.org/drafts/current/.

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."

This Internet-Draft will expire on September 10, 2015.

Copyright Notice

Table of Contents

1. Introduction

1.1. Motivations

This document proposes a new relay-based NAT traversal architecture called TURN-Lite based on the following motivations.

1) Leverage ISPs' infrastructures

Currently, the deployment of TURN [RFC5766] is very limited and most of the application providers use their own platform to transfer the data between two hosts that behind NATs and to translate the

communication packets between two hosts in different address families.

The relay devices deployed centrally by various application providers often lead to inefficient data transmit between two hosts. The relay devices must deal with complex network layer problems which the application providers are not familiar with.

On the other hand, service providers have deployed many CGN devices/ CDN nodes in a distributed manner within their networks. If the service providers can use these CGN devices/CDN nodes as the relay devices for communication between two hosts behind NATs or that from different address family, and open their data translation/forwarding capability to the application providers, the host to host communication will be more efficient. Given most of the CGNs are capable of translating packets between IPv4 and IPv6, the adoption of IPv6 technology will also be accelerated.

2) Simplify the communication procedures

TURN-Lite needs less communication procedures than TURN of which the procedures are considered very complex to be integrated into the ISPs' infrastructure, for example:

o TURN solution has to closely interact with ICE

Within current TURN solution, there are scenarios where the ICE or other NAT-hole punching procedures must be included for the success of communication via TURN servers. The key point is that TURN allocates different relay transport address-port pairs for different hosts.

Each client must first use TURN allocation request to get their transport relay address-port pairs, and then must use ICE procedure (connectivity check) or other similar signaling to punch holes for these transport relay addresses on the alongside NAT devices. Or else the relayed UDP/TCP packet will be blocked.

Even with the above procedures, there still exist some risks that the packets be rejected by TURN server due to the

permission list that created by client via "CreatePermission Request" before it sending data to the peer. In order to mitigate such issues, current TURN solution requires the TURN servers only check the IP address part of the relay transport address, and ignore the port portion. But this will again introduce some attack risks because different host may share

one public IP address when the CGN device is deployed within network.

- o IPv4/IPv6 Relay Address/Port Reservation and Allocation

  Another drawback of different relay transport addresses for different host is that the TURN server must reserve some IPv4/IPv6 address block for the allocation and plan the TCP/UDP port usage for each host. When TURN servers are deployed in a distribute manner (For example when they are incorporated into the CGN devices), there will be much coordination work to do for the address/port reservation and allocation on the TURN servers.

- o Simultaneous TCP/UDP connections burden on TURN server

  Current TURN solution requires the TURN servers to open and listen on many TCP/UDP ports at the same time, Under TURN solution for TCP [RFC6062], each host requires two connections to the TURN server. This will increase the burden on TURN server and the complexity to incorporate them into the CGN/CDN devices.

- o Different procedures for TCP/UDP communication

  Current TURN solution adopts different procedures for the TCP and UDP communication channel. So for one TURN server to provide the TCP/UDP relay function, it must implement two different procedures. This again increases the complexity of the TURN server implementation, especially in CGN devices.

- o Communication complexity between two different TURN servers

  Current TURN solution cannot assure two hosts select the same TURN server, and then it must deal with the communication situation between two different TURN servers. This scenario has not been covered by the current TURN related drafts. The client must reuse the XOR-PEER-ADDRESS attribute to include the relay address of the peer to reach the second TURN server.

  On the other hand, because the hosts select their own TURN server, there is no mechanism to assure the relayed path is most optimal for them. The application latency will be increased when this occurs.

TURN-Lite solution will simplify the above mentioned complexity and make the TCP/UDP data relay function be easily incorporated into the existing distributed CGN devices or other kinds distributed devices i.e. the CDN nodes etc.

1.2. Relationship with TURN

This document doesn't intent to replace TURN with TURN-Lite, but consider TURN-Lite as a complementary solution along with TURN for some specific scenarios.

If one SP wants to open its infrastructure to accelerate their customers' (mainly regarding to application providers) client-to-client communications within the SP's domain, TURN-Lite could be a good candidate.

2. Requirements Language and Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119] when they appear in ALL CAPS. When these words are not in ALL CAPS (such as "should" or "Should"), they have their usual English meanings, and are not to be interpreted as [RFC2119] key words.

- Application Provider: the service providers who provide client to client communications through the Internet. E.g. VoIP service providers, instant message service providers etc.

- Relay Selector: which is in charge of selecting a proper relay device (CGN or CDN nodes) for the communicating hosts behind NATs. Normally, the RS is a function located in the network's management plane and possibly a part of the NMS server

- TURN-Lite: lightweight TURN architecture. The word "lightweight" is in the perspective of an application provider.

- TURN-Lite Client: the TURN-Lite entity that deployed in the application providers' networks; be responsible for TURN-Lite signaling/control interactions with the TURN-Lite servers.

- TURN-Lite Server: the TURN-Lite entity that deployed in the ISP's networks; be mainly responsible for the data relay between an application providers' clients. Normally, the TURN-Lite servers collated with the CGNs (Carrier Grade NATs) within the service provider.

3. Solution Overview

3.1. Reference Architecture of TURN-Lite

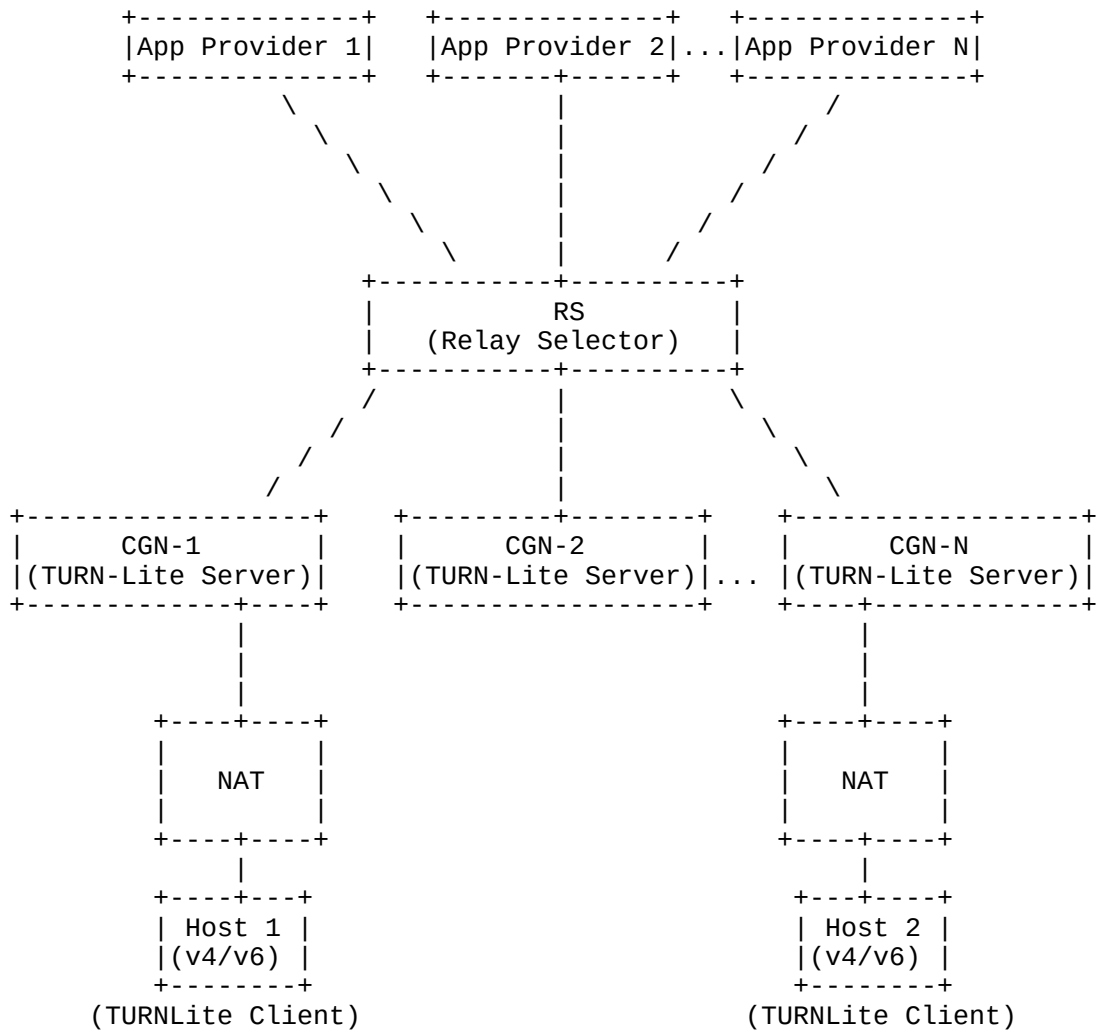


Fig. 3-1: TURN-Lite Arhitecture

As depicted in above figure, the application clients that located on hosts act as the TURN-Lite clients while the CGNs act as TURN-Lite Servers. There is a Relay Selector (RS) for choosing a proper CGN to relay traffic between the two hosts. In practice, the RS could be a dedicated server or a function located in the management plane servers such as NMS server.

RS has the intelligent route selection capability to choose a proper CGN for a given host pair. RS sends the data relay indication to the selected CGN devices/CDN node via the newly defined "Couple" method.

BEHAVE compliant and non-BEHAVE compliant NAT traverse [RFC4787] [RFC5382] is supported in TURN-Lite. IPv6 and IPv4 host communication is also supported.

3.2. Solution Rationale

The solution could be briefly described in the following sections.

3.2.1. Relay Selector Reflection and Selection

Each host that wants to communicate with the other host should send STUN message to the RS (Relay Selector), and get their reflex addresses to the RS (here we refer to REFLX-RS).

The application provider needs to select a suitable RS and informs it to the hosts (e.g. via application specific client-server protocol). The detailed RS selection mechanism and criteria are out of the scope of this document, but some general considerations are as the following.

- If the hosts locate in the same ISP/administrative domain, then the RS selection is fairly easy since normally there is only one RS in one ISP; even there are multiple RSes in one ISP, the application provider should also be able to select a suitable RS based on the addresses of the two hosts.

- If the hosts locate in two ISPs/administrative domains (assuming both of the ISPs providing TURN-Lite service to the application provider), the application provider can select one RS based on pre-defined policies (the simplest way is just arbitrarily choosing one RS in one of the ISPs).

- The application provider can also select two RS to deal with the communication between two hosts that located in different service provider. Under such situation, the application provider will send one extend "Couple"command to each RS, let the RS tunnel the customer’s data to another RS. The detail process of this situation will be provided further. Currently, we focus only the one ISP scenario.

3.2.2. Relay Selection

Each host will report its REFLX-RS address to its application server. If two hosts want to communicate, the application server will send the two hosts’ REFLX-RS addresses to the selected RS, to let the RS select one appropriate relay device to relay the traffic.

Generally, the RS can select the appropriate relay device based solely on the REFLX-RS addresses of these two hosts, for example, select one relay device that locate in the middle of the communication path. This approach is possible since the relay

behavior is within one ISP's domain that the RS could be possible to learn the knowledge of all CGNs (relays) within that domain.

The selection criteria can also be expanded to include other factors, such as the privacy concern of the communication peers, the linkage usage information between the host and the relay device etc. For example, RS can select one relay device that locates far from the communication peer to hide the location of the peer. This might sacrifice the communication efficiency but increase the protection of the host privacy. Anyway, RS has more flexible control over the relay selection, upon the requirement of communication hosts, or the consideration of relay service provider.

After the relay device selection, the RS will respond the IP address of the selected relay device to the communication peer, together with the well known port that used by every relay device. The combination of this relay IP address and the well-known port form the relay transport address of the communication peers, each peer will use this relay transport address to communicate.

When two hosts located within one administration domain, the centralized relay point selection and control architecture can easily achieve one low latency communication path because it knows the whole network condition of its own. When two hosts located within different administration domains, the TURN-Lite solution will also work except that some end-to-end communication efficiency might be sacrificed unless there is some coordination between these two administration domains.

3.2.3. Forming "Couple" Command

Each host will send again one STUN message to the selected relay transport address, get the new reflex address(here we refer to REFLX-Relay) to the selected relay device, and reports it to the RS, together with the previous reflex address to the RS (which is REFLX-RS).

The RS will use the REFLX-RS addresses to find out which two peers will communicate (such communication pair information is gotten from Section 3.2.2). RS will retrieve the corresponding REFLX-Relay address of the communication peer, forms the "Couple" command based on such information, and sends the "Couple" command to the selected relay transport address.

Upon receiving the "Couple" command, the relay device will add one item to its forwarding table. The forwarding table will bind the reflex addresses of the two peers, the required transport protocol and other additional information.

3.2.4. Data Relay

Each host will then send the data traffic directly to the unique relay transport address. The source address of this packet will be changed by the alongside NAT devices that located between the host and the relay device.

When this packet arrives to the relay address, its source address will be one of the RFLEX-Relay addresses. The relay device will search the forwarding table that formed in Section 3.2.3. If the REFLX-Relay address in one item match the source address of the received packet, then the other REFLX-Relay address will be retrieved and be used as the destination address of the application packet, the packet's source address will be changed to the relay transport address.

After the conversion, the packet will be sent by the relay device. This packet will be routed to the corresponding peer, according to its REFLX-Relay address.

The application return packet will be sent again back to the same relay device via the relay transport address. The similar search process and convert work will be done by the relay device. The converted return packet will then be routed to the packet originator.

4. New STUN Method Definition

In order to let the CGN device to build one Couple item upon the request of RS, it is needed to define one general Couple message to transfer the related information.

4.1. Couple Operation

The Couple request defines the relationship between two TCP or UDP half-connections, the translation rule that converts both the source address and destination address of pass through packet in both directions.

Couple Opcode: It defines how to bind two half-connections that ends at the CGN's well-known relay transport address together. When CGN device receives the Couple request, it will create one map table item

that includes the reflex IP address/port [REFLX-Relay] of both hosts that lies behind the NAT device and the protocol that the host will use to communicate.

When the CGN device receives the packet from one host, it will use the reflex source address/port to lookup the map table item; converts the source address/port of this packet to the relay transport address

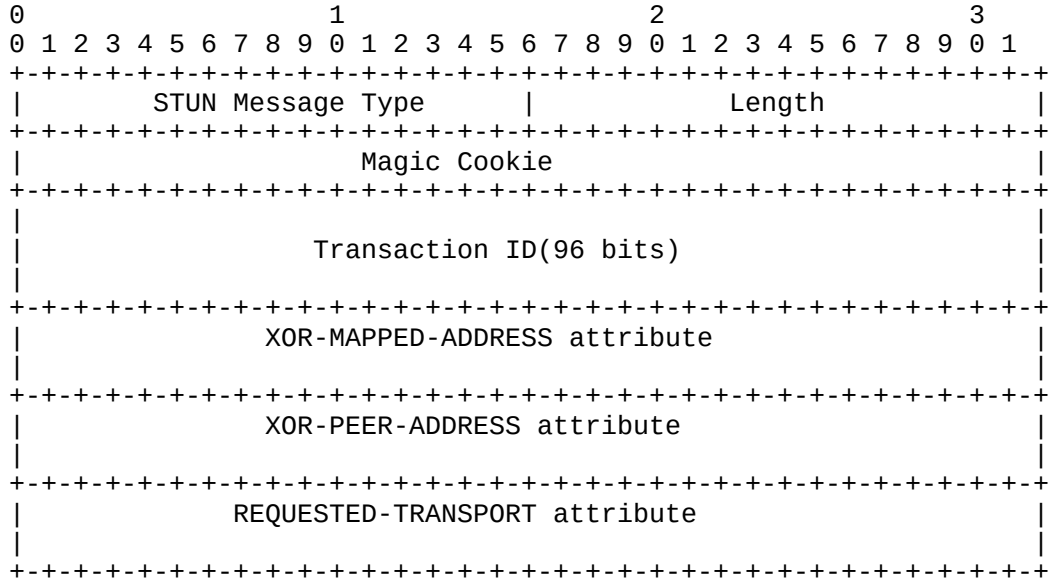
of the CGN device and converts the destination address/port of this packet to the reflex address [REFLX-Relay] that results from the map table lookup action.

The converted packet will be routed to NAT side of the other host, converted by the NAT device and then to the other host. The return packet will be delivered to the relay transport address of CGN/CDN device and be converted in reverse accordingly.

4.2. Couple Operation Packet Format

The Couple Opcode allows RS to create a new explicit couple table on the CGN device(TURN-Lite Server), instructs the CGN device to accomplish the related translation work.

The following diagram shows the Opcode layout for the Couple Opcode request/response format.



STUN Message Type — Couple method: value TBD. only request/response semantics

Decouple method: value TBD. only request/response semantics

Length — The same definition as STUN prococol [RFC5389]

Magic Cookie	The same definition as STUN prococol [RFC5389]
Transaction ID	The same definition as STUN prococol [RFC5389]
XOR-MAPPED-ADDRESS	The same definition as STUN prococol [RFC5389]. The value should be the RFLX-Relay address of the host.
XOR-PEER-ADDRESS	The same definition as TURN prococol [RFC5766]. The value should be the RFLX-Relay address of the peer.
REQUESTED-TRANSPORT	The same definition as TURN prococol [RFC5766]. the value of the "protocol" fiel should be TCP or UDP.

Fig.4-1: Couple Opcode Request/Response Format

5. Detailed Example

5.1. Procedures of Communication Traversing Symmetric NATs

When one of the communication hosts located behind the symmetric NAT device, the host-to-host communication must via one relay device. Below are the key procedures of TURN-Lite solution, we use the Fig 3-1 to describe the example.

Please note the communication procedures between the hosts and the application server are out of the scope of this document, we only focus on the key procedure proposed by this document.

1. If Host 1 and Host 2 want to communicate with each other, they will send STUN binding message to the RS IPv4 address/port, get their reflex address to RS[REFLX-RS].

2. RS will select one CGN device to relay the packet, based on the RS addresses information of the two peers. Here we assume it select CGN-1 as the relay device. RS will notify Host 1 and Host 2 of their relay transport address, both will use the same relay IP address/port on CGN-1.

3. Host 1 and Host 2 will send STUN binding message to CGN-1, get their relay address to CGN-1[REFLX-Relay] and report them to RS, together with RS addresses gotten in step 1). Here we assume the [REFLX-Relay] address of Host 1 is 192.0.2.1:7000, and [REFLX-Relay] address of Host 2 is 192.0.2.150:32102.

4. RS will form the "Couple" message, which include mainly the [REFLX-Relay] addresses of Host 1 and Host 2 and their communication protocol, here we assume they use TCP to communicate.

5. Upon receiving the "Couple" message, the CGN-1 device will form one forwarding table that look like below:

Reflextive transport address of Host1	Reflextive transport address of Host2	Transport Protocol
192.0.2.1:7000	192.0.2.150:32102	TCP

Table 5-1: Couple Table Example (symmetric case)

6. Host1 will send the application data to the relay transport address in CGN-1.

7. CGN device will look up the Couple table, use the source address of received packet(192.0.2.1:7000 in this example) to get the reflex IPv4 address of Host 2.

8. It then will change the source address of the packet to the relay transport address in CGN device, the destination address of this packet to the IPv4 reflex address of Host 2. The translated packet will be forwarded to Host 2.

9. The return traffic will also be sent to the same relay transport address in CGN-1, converted by the CGN device, and sent back to Host 1.

5.2. Procedures of IPv4 and IPv6 Host Communication

If Host 1 is one IPv4 node and Host 2 is one IPv6 node. The communication process are similar, except the relay address that is sent to the Host 1 is the IPv4 address of the CGN-1 and the relay address that is sent to the Host 2 is the IPv6 address of the CGN-1. Host 1 and Host 2 will not notice that they are talking to one node that in different address family.

The relay device selection process is same as the above example. Here we describe the procedure from step 3.

3. Host 1 and Host 2 will send STUN binding message to CGN-1, get their relay address to CGN-1[REFLX-Relay] and report them to RS, together with RS addresses gotten in step 1). Here we assume the

[REFLX-Relay] address of Host 1 is 192.0.2.1:7000, and [REFLX-Relay] address of Host 2 is 2001:c68:300:105::1[49191].

4. RS will form the "Couple" message, which include mainly the [REFLX-Relay] addresses of Host 1 and Host 2 and their communication protocol, here we assume they use TCP to communicate.

5. Upon receiving the "Couple" message, the CGN-1 device will form one forwarding table that look like below:

Reflextive transport address of Host1	Reflextive transport address of Host2	Transport Protocol
192.0.2.1:7000	2001:c68:300:105::1[49191]	UDP

Table 5-2: Couple Table Example (different address families case)

6. Host1 will send the application data to the relay transport address in CGN-1.

7. CGN device will look up the Couple table, use the source address of received packet(192.0.2.1:7000 in this example) to get the reflex IPv6 address of Host 2.

8. It then will change the source address of the packet to the relay transport IPv6 address in CGN device, the destination address of this packet to the IPv6 reflex address of Host 2. The translated packet will be forwarded to Host 2.

9. The return traffic will also be sent to the same relay transport address in CGN-1, converted by the CGN device, and sent back to Host 1.

6. TURN-Lite Benefits

Comparing to TURN, TURN-Lite could provide following benefits:

o Decoupled from ICE

TURN is tightly coupled with ICE. Operations like NAT punching, create permission .etc all require ICE connectivity check packets.

Benefited from the couple operation, TURN-Lite doesn't necessarily need ICE interaction.

- Avoid the Create Permission issues in TURN

  In the TURN-Lite solution, each communication pair will use the same relay server and the same relay address. The relay permission action required by TURN solution is replaced with the "Couple" command. There is no ambiguity for the relay permission because "Couple" command use the ip address and port information of the communication pair simultaneously. There are also no possible attacks due to the loose control of the current TURN permission treaments.

- Less Relay Address/Port Consumption and Management

  TURN-LiteTURN-Lite doesn't need to allocate different address-port pair for each session initiated from the hosts. Thus, it could obviously reduce the resource consumption and the human burden for planning the resource allocation.

- Simplified Procedures

  Theoretically, it requires only two commands to accomplish the relay function, compared with over eight commands that required by TURN solution. Due to every host communicate with the well-known relay address, there is no additional requirement for punching holes in the NAT devices, which is indispensable for the current TURN solution.

	TURN Solution	TURN-Lite Solution
Required Commands	1. Binding 2. Allocate 3. Send 4. Data 5. Channel Bind 6. Connect 7. ConnectionBind 8. ConnectionAttempt	1. Binding 2. Couple

Table 6-1: Procedures comparison between TURN and TURN-Lite

- Unified solution for TCP/UDP and IPv4(6)-IPv6(4) data relay

TURN-Lite supports the data relay for the communication between two hosts, uses same mechanism for TCP and UDP transport protocol, even for the communication between different address families.

- o Support for optimal relay selection

  Because of the central deployed RS could have the whole network's routing/path knowledge, TURN-Lite is more likely to achieve an optimal relay (TURN-Lite server) selection based on various information such as the reflective transport addresses of the two communicating peers, the link usage information between two peers and the load status of the candidate TURN-Lite servers etc.

  With the RS's knowledge, TURN-Lite is also more likely to achieve better relay selection for some specific requirements. For example, if one peer wants to hide its ip address to protect its privacy, the RS in TURN-Lite architecture could possibly select one TURN-Lite server that located far away from the host.

7. TURN-Lite Deployment Considerations

The TURN-Lite Server can be deployed in distributed manner. The most appropriate devices for incorporating this function are the CGN devices that have been deployed distributed by the service provider. Each distributed TURN-Lite Server has one unique public IPv4/IPv6 transport address.

The RS can select the appropriate TURN-Lite Server based on the proximity of the TURN-Lite server with the communication hosts and can also use other criteria to influence the selection procedure, as described in Section 3.

8. Security Considerations

The additional requirement of TURN-Lite is authenticating the couple operation from the RS. When the communication channel between the RS and the TURN-Lite server is secured, such security risks can be mitigated accordingly.

9. IANA Considerations

This draft requires IANA to allocate following STUN methods:

Couple: value TBD.

Decouple: value TBD.

10. Conclusions

Currently, the communication between two hosts that located behind NAT devices, especially that behind the symmetric NAT devices is emerging. With the development of IPv6 technology, the communication between two hosts that in different address families needs also be considered. Under the TURN-Lite architecture, the communication requests for IPv4/IPv4, IPv4/IPv6 scenario can be met in one general solution. Such solution can alleviate the burden of various CP/SP to deploy the TURN server by themselves, exploit and open the capabilities of CGN device that deployed by service provider to the third party(CP/SP), make the host-to-host communication more efficient.

11. Acknowledgements

Many valuable comments were received from Brandon Williams, Oleg Moskalenko, Jonathan Rosenberg, and Dan Wing etc.

This document was produced using the xml2rfc tool [RFC2629].

12. References

12.1. Normative References

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

[RFC2629] Rose, M., "Writing I-Ds and RFCs using XML", RFC 2629, June 1999.

[RFC5389] Rosenberg, J., Mahy, R., Matthews, P., and D. Wing, "Session Traversal Utilities for NAT (STUN)", RFC 5389, October 2008.

[RFC5766] Mahy, R., Matthews, P., and J. Rosenberg, "Traversal Using Relays around NAT (TURN): Relay Extensions to Session Traversal Utilities for NAT (STUN)", RFC 5766, April 2010.

12.2. Informative References

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

[RFC5382] Guha, S., Biswas, K., Ford, B., Sivakumar, S., and P. Srisuresh, "NAT Behavioral Requirements for TCP", BCP 142, RFC 5382, October 2008.

[RFC6062] Perreault, S. and J. Rosenberg, "Traversal Using Relays around NAT (TURN) Extensions for TCP Allocations", RFC 6062, November 2010.

Authors' Addresses

Aijun Wang
China Telecom
No.118,Xizhimenneidajie,Xicheng District
Beijing, 100035
P.R. China

Email: wangaj@ctbri.com.cn

Bing Liu
Huawei Technologies
Q14, Huawei Campus, No.156 Beiqing Road, Hai-Dian District
Beijing, 100095
P.R. China

Email: leo.liubing@huawei.com