Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: July 29, 2016

B. Schwartz
J. Uberti
Google

January 26, 2016

Recursively Encapsulated TURN (RETURN) for Connectivity and Privacy in WebRTC
draft-ietf-rtcweb-return-01

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 will expire on July 29, 2016.

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

```
      ____________    inside network     ||           outside network
     /            \                      ||  NAT/FW
     |       host O              ________||________
     |            |             /        ||        \
     |       srflx|.............|..................O      ___________
     |            |             |        ||        |     /           \
     |       relay|- - - - - - -|- - - - - - - - - |- - -|- - - - - -O
     |            |             |        ||        |     |           |
     |            |             |        ||        |     |           |
     |            |             |        ||        |     |           |
     |            |             |        ||        |     |           |
     |            |             |        ||        |     \___________/
     |            |             |        ||        |
     |            |             |        ||        |    Application TURN
     |            |             |        ||        |    server in cloud
     \____________/             \________||________/
                                         ||
         Browser                         ||
                                         ||
     KEY     O    Candidate
           .....  Non encapsulated
           - - -  TURN encapsulated
             ||   Network edge
```

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.

```
       ____________    inside network     ||           outside network
      /            \                      ||  NAT/FW
      |       host O              ________||________
      |            |             /        ||        \
      |       srflx|.............|..................O
      |            |             |        ||        |
      |            |             |        ||        |
      |            |             |   _____||_____   |
      |            |             |  /     ||     \  |
      |    (border)|- - - - - - -| -|- - - - - - O  |
      |            |             |  |     ||     |  |
      |            |             |  |     ||     |  |
      |            |             |  |     ||     |  |
      |            |             |  |     ||     |  |
      |            |             |  \_____||_____/  |
      \____________/             \________||________/
                                          ||
          Browser           Border TURN   ||
                           server in DMZ  ||
      KEY     O    Candidate
            .....  Non encapsulated
            - - -  TURN encapsulated
              ||   Network edge
```

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.

```
                      inside network     ||           outside network
      ____________                       ||  NAT/FW
     /            \              ________||________
     |            |             /        ||        \
     |            |             |        ||        |
     |       host O.............X        ||        |
     |            |             |        ||        |
     |       srflx|.............X..................O      ___________
     |            |             |        ||        |     /           \
     |       relay|- - - - - - -X- - - - - - - - - |- - -|- - - - - -O
     |            |             |   _____||_____   |     |           |
     |            |             |  /     ||     \  |     |           |
     |    (border)|- - - - - - -|- |- - - - - - O  |     |           |
     |            |             |  |     ||     |  |     \___________/
     |            |             |  |     ||     |  |
     |            |             |  |     ||     |  |    Application TURN
     |            |             |  |     ||     |  |         server
     |            |             |  \_____||_____/  |
     \____________/             \________||________/
                                         ||
         Browser             Border TURN ||
                               server    ||
     KEY     O    Candidate
           .....  Non encapsulated
           - - -  TURN encapsulated
             ||   Network edge
             X    Firewall block
```

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.

```
       ____________    inside network     ||           outside network
      /            \                      ||  NAT/FW
      |       host O              ________||________
      |            |             /        ||        \
      |       srflx|.............|..................O      ___________
      |            |             |        ||        |     /           \
      |       relay|- - - - - - -|- - - - - - - - - |- - -|- - - - - -O
      |            |             |   _____||_____   |     |           |
      |            |             |  /     ||     \  |     |           |
      |      relay2|-------------|--|------------| -|- - -|- - - - - -O
      |            |             |  |     ||     |  |     \___________/
      |      srflx2|- - - - - - -|- |- - - - - - O  |
      |            |             |  |     ||     |  |   Application TURN
      |      host2 |- - - - - - -|- |- - - - - - O  |         server
      |            |             |  \_____||_____/  |
      \____________/             \________||________/
                                          ||
          Browser       Border TURN Proxy ||
                              server      ||
      KEY     O    Candidate
            .....  Non encapsulated
            - - -  TURN encapsulated
            -----  Double TURN encapsulated
              ||   Network edge
```

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.

```
       ____________    inside network     ||           outside network
      /            \                      ||  NAT/FW
      |            |              ________||________
      |            |             /   _____||_____   \
      |            |             |  /     ||     \  |
      |            |             |  |     ||     |  |
      |            |             |  |     ||     |  |
      | Web Traffic|=============|==|============H  |
      |            |             |  |     ||     |  |
      |            |       HTTP/ |  |     ||     |  |
      |            |       HTTPS |  \_____||_____/  |
      |            |       Proxy |        ||        |
      |            |             |   _____||_____   |
      |            |             |  /     ||     \  |
      |            |             |  |     ||     |  |
      |            |             |  |     ||     |  |
      |WebRTC Media|- - - - - - -|- |- - - - - - O  |
      |            |             |  |     ||     |  |
      |            |             |  |     ||     |  |
      |            |       TURN  |  \_____||_____/  |
      \____________/       Proxy \________||________/
                                          ||
          Browser                         ||
                                          ||
      KEY     O    TURN Proxy Candidate
              H    HTTP/HTTPS Proxy Address
            - - -  TURN encapsulated
            =====  HTTP/HTTPS web traffic
              ||   Network edge
```

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