RTCWeb Working Group
Internet-Draft
Intended status: Standards Track
Expires: September 10, 2015

H. Alvestrand
Google
U. Rauschenbach
Nokia Networks

March 09, 2015

WebRTC Gateways
draft-alvestrand-rtcweb-gateways-02

Abstract

This document specifies conformance requirements for a class of WebRTC-compatible endpoints called "WebRTC gateways", which interconnect between WebRTC endpoints and devices that are not WebRTC endpoints.

Requirements Language

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 RFC 2119 [RFC2119].

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

The WebRTC model described in [I-D.ietf-rtcweb-overview] is focused on direct browser to browser communication as its primary use case. Nevertheless, it is clearly interesting to have WebRTC endpoints connect to other types of devices, including but not limited to SIP phones, legacy phones, CLUE-based teleconferencing systems, XMPP-based conferencing systems, and entirely proprietary devices or systems.

WebRTC gateways are a specific type of WebRTC-compatible endpoints which enable the exchange of media streams between WebRTC endpoints on one side, and the other types of devices mentioned above on the other side.

This document describes the requirements that need to be placed on such gateways, both the requirements on WebRTC endpoints that can be relaxed and the additional requirements that need to be applied.

A WebRTC gateway is a WebRTC-compatible endpoint, and will thus not be conformant with all requirements for a WebRTC endpoint (it does not do everything a WebRTC endpoint does), but is able to interoperate with WebRTC endpoints.

1.1. Implications of the gateway environment

A gateway will be limited in the functionality it can offer by the system or class of devices it is gatewaying to. For instance, a gateway into the telephone system will not be able to relay data or video, no matter how much it is required. Therefore, a number of functions that are mandatory to support in WebRTC endpoints are not mandatory on gateways; the requirement on the gateway is that it is able to negotiate those features away correctly.

1.2. Signalling model

The WebRTC model is that signalling is outside the scope of the specification. This document does not change that.

Nevertheless, any practical gateway needs to deal with signalling. For that, this document assumes that the overall system consists of an application running in the WebRTC browser, possibly one or more signalling relays that mediate signalling and thereby enable communication between the application and the gateway, and the actual gateway that is responsible for handling the media flows.

The application, the signalling relays (if any) and the gateway together need to be able to:

- adhere to the offer/answer semantics
- deal with the description of configuration coming from the browser; this is specified in SDP format in the WebRTC browser API
- generate the information that is needed by the browser to set up the session, and express that information in the form of SDP.

The shorthand notation "The gateway MUST/SHOULD/MAY support <SDP function xxx>" used below means that an application running in the Web browser, the signalling relays, and the gateway together MUST/SHOULD/MAY support this functionality; it is not a requirement

that this happens at the media gateway itself.

2. WebRTC non-browser requirements that can be relaxed

WebRTC gateways are intended to communicate with WebRTC endpoints. WebRTC gateways are no User Agents. They are therefore expected to conform to the requirements for WebRTC non-browsers in [I-D.ietf-rtcweb-overview], with the exceptions defined in this section.

Since a gateway is expected to be deployed where it can be reached with a static IP address (as seen from the client), a WebRTC gateway does not need to support full ICE; it therefore MAY implement ICE-Lite only.

ICE-Lite implementations do not send consent checks, so a gateway MAY choose not to send consent checks too, but MUST respond to consent checks it receives.

A gateway is expected to not need to hide its location, so it does not need to support functionality for operating only via a TURN server; instead it MAY choose to produce Host ICE candidates only.

If a gateway serves as a media relay into another RTP domain, it MAY choose to support only features available in that network. This means that it MAY not (need to) support Bundle and any of the RTP/RTCP extensions related to it, RTCP-Mux, or Trickle Ice. However, the gateway MUST support DTLS-SRTP, since this is required for interworking with WebRTC endpoints.

If a gateway serves as a media relay into a network or to devices not implementing the WebRTC Datachannel, it MAY choose to not support the Datachannel.

3. Additional WebRTC gateway requirements

(nothing yet)

4. IANA Considerations

This document makes no request of IANA.

Note to RFC Editor: this section may be removed on publication as an RFC.

5. Security Considerations

A WebRTC gateway may operate in two security modes: Security-context termination and security-context relaying.

Relaying is only possible where signed and encrypted content can be passed through unchanged, and where keys can be exchanged directly between the endpoints.

When the gateway terminates the security context, it means that the WebRTC user has to place trust in the gateway to perform all verification of identity and protection of content in the realm on the other side of the gateway; there is no way the end-user can detect a man-in-the-middle attack, an identity spoofing attack or a recording done at the gateway. For many scenarios, this is not going to be seen as a problem, but needs to be considered when one decides to use a gatewayed service.

6. Acknowledgements

Several comments from Christer Holmberg were included.

7. Change history

Changes from draft-alvestrand-rtcweb-gateways-00

- Aligned terminology with draft-rtcweb-overview-12
- Rewrote text on signaling to improve clarity
- Editorial nits

Changes from draft-alvestrand-rtcweb-gateways-01

- Aligned terminology with draft-rtcweb-overview-13 ("non-browser")
- Nits

8. Normative 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.

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

Authors' Addresses

Harald Alvestrand
Google

Email: harald@alvestrand.no

Uwe Rauschenbach
Nokia Networks

Email: uwe.rauschenbach@nokia.com

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

A.B. Roach
Mozilla

June 12, 2015

WebRTC Video Processing and Codec Requirements
draft-ietf-rtcweb-video-06

Abstract

This specification provides the requirements and considerations for WebRTC applications to send and receive video across a network. It specifies the video processing that is required, as well as video codecs and their parameters.

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 14, 2015.

Table of Contents

1. Introduction

One of the major functions of WebRTC endpoints is the ability to send and receive interactive video. The video might come from a camera, a screen recording, a stored file, or some other source. This specification provides the requirements and considerations for WebRTC applications to send and receive video across a network. It specifies the video processing that is required, as well as video codecs and their parameters.

Note that this document only discusses those issues dealing with video codec handling. Issues that are related to transport of media streams across the network are specified in [I-D.ietf-rtcweb-rtp-usage].

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].

3. Pre and Post Processing

This section provides guidance on pre- and post-processing of video streams.

Unless specified otherwise by the SDP or codec, the color space SHOULD be sRGB [SRGB]. For clarity, this is the color space

indicated by codepoint 1 from "ColourPrimaries" as defined in [IEC23001-8].

Unless specified otherwise by the SDP or codec, the video scan pattern for video codecs is Y'CbCr 4:2:0.

3.1. Camera Source Video

This document imposes no normative requirements on camera capture; however, implementors are encouraged to take advantage of the following features, if feasible for their platform:

- Automatic focus, if applicable for the camera in use
- Automatic white balance
- Automatic light level control
- Dynamic frame rate for video capture based on actual encoding in use (e.g., if encoding at 15 fps due to bandwidth constraints, low light conditions, or application settings, the camera will ideally capture at 15 fps rather than a higher rate).

3.2. Screen Source Video

If the video source is some portion of a computer screen (e.g., desktop or application sharing), then the considerations in this section also apply.

Because screen-sourced video can change resolution (due to, e.g., window resizing and similar operations), WebRTC video recipients MUST be prepared to handle mid-stream resolution changes in a way that preserves their utility. Precise handling (e.g., resizing the element a video is rendered in versus scaling down the received stream; decisions around letter/pillarboxing) is left to the discretion of the application.

Note that the default video scan format (Y'CbCr 4:2:0) is known to be less than optimal for the representation of screen content produced by most systems in use at the time of this document's publication, which generally use RGB with at least 24 bits per sample. In the future, it may be advisable to use video codecs optimized for screen content for the representation of this type of content.

Additionally, attention is drawn to the requirements in [I-D.ietf-rtcweb-security-arch] section 5.2 and the considerations in [I-D.ietf-rtcweb-security] section 4.1.1.

4. Stream Orientation

In some circumstances - and notably those involving mobile devices - the orientation of the camera may not match the orientation used by the encoder. Of more importance, the orientation may change over the course of a call, requiring the receiver to change the orientation in which it renders the stream.

While the sender may elect to simply change the pre-encoding orientation of frames, this may not be practical or efficient (in particular, in cases where the interface to the camera returns pre-compressed video frames). Note that the potential for this behavior adds another set of circumstances under which the resolution of a screen might change in the middle of a video stream, in addition to those mentioned under "Screen Sourced Video," above.

To accommodate these circumstances, RTCWEB implementations that can generate media in orientations other than the default MUST support generating the R0 and R1 bits of the Coordination of Video Orientation (CVO) mechanism described in section 7.4.5 of [TS26.114], and MUST send them for all orientations when the peer indicates support for the mechanism. They MAY support sending the other bits in the CVO extension, including the higher-resolution rotation bits. All implementations SHOULD support interpretation of the R0 and R1 bits, and MAY support the other CVO bits.

Further, some codecs support in-band signaling of orientation (for example, the SEI "Display Orientation" messages in H.264 and H.265). If CVO has been negotiated, then the sender MUST NOT make use of such codec-specific mechanisms. However, when support for CVO is not signaled in the SDP, then such implementations MAY make use of the codec-specific mechanisms instead.

5. Mandatory to Implement Video Codec

For the definitions of "WebRTC Browser," "WebRTC Non-Browser", and "WebRTC-Compatible Endpoint" as they are used in this section, please refer to [I-D.ietf-rtcweb-overview].

WebRTC Browsers MUST implement the VP8 video codec as described in [RFC6386] and H.264 Constrained Baseline as described in [H264].

WebRTC Non-Browsers that support transmitting and/or receiving video MUST implement the VP8 video codec as described in [RFC6386] and H.264 Constrained Baseline as described in [H264].

> NOTE: To promote the use of non-royalty bearing video codecs, participants in the RTCWEB working group, and any successor

working groups in the IETF, intend to monitor the evolving licensing landscape as it pertains to the two mandatory-to-implement codecs. If compelling evidence arises that one of the codecs is available for use on a royalty-free basis, the working group plans to revisit the question of which codecs are required for Non-Browsers, with the intention being that the royalty-free codec will remain mandatory to implement, and the other will become optional.

These provisions apply to WebRTC Non-Browsers only. There is no plan to revisit the codecs required for WebRTC Browsers.

"WebRTC-compatible endpoints" are free to implement any video codecs they see fit. This follows logically from the definition of "WebRTC-compatible endpoint." It is, of course, advisable to implement at least one of the video codecs that is mandated for WebRTC Browsers, and implementors are encouraged to do so.

6. Codec-Specific Considerations

SDP allows for codec-independent indication of preferred video resolutions using the mechanism described in [RFC6236]. WebRTC endpoints MAY send an "a=imageattr" attribute to indicate the maximum resolution they wish to receive. Senders SHOULD interpret and honor this attribute by limiting the encoded resolution to the indicated maximum size, as the receiver may not be capable of handling higher resolutions.

Additionally, codecs may include codec-specific means of signaling maximum receiver abilities with regards to resolution, frame rate, and bitrate.

Unless otherwise signaled in SDP, recipients of video streams MUST be able to decode video at a rate of at least 20 fps at a resolution of at least 320 pixels by 240 pixels. These values are selected based on the recommendations in [HSUP1].

Encoders are encouraged to support encoding media with at least the same resolution and frame rates cited above.

6.1. VP8

For the VP8 codec, defined in [RFC6386], endpoints MUST support the payload formats defined in [I-D.ietf-payload-vp8].

In addition to the [RFC6236] mechanism, VP8 encoders MUST limit the streams they send to conform to the values indicated by receivers in the corresponding max-fr and max-fs SDP attributes.

Unless otherwise signaled, implementations that use VP8 MUST encode and decode pixels with a implied 1:1 (square) aspect ratio.

6.2. H.264

For the [H264] codec, endpoints MUST support the payload formats defined in [RFC6184]. In addition, they MUST support Constrained Baseline Profile Level 1.2, and they SHOULD support H.264 Constrained High Profile Level 1.3.

Implementations of the H.264 codec have utilized a wide variety of optional parameters. To improve interoperability the following parameter settings are specified:

packetization-mode: Packetization-mode 1 MUST be supported. Other modes MAY be negotiated and used.

profile-level-id: Implementations MUST include this parameter within SDP and MUST interpret it when receiving it.

max-mbps, max-smbps, max-fs, max-cpb, max-dpb, and max-br: These parameters allow the implementation to specify that they can support certain features of H.264 at higher rates and values than those signalled by their level (set with profile-level-id). Implementations MAY include these parameters in their SDP, but SHOULD interpret them when receiving them, allowing them to send the highest quality of video possible.

sprop-parameter-sets: H.264 allows sequence and picture information to be sent both in-band, and out-of-band. WebRTC implementations MUST signal this information in-band. This means that WebRTC implementations MUST NOT include this parameter in the SDP they generate.

H.264 codecs MAY send and MUST support proper interpretation of SEI "filler payload" and "full frame freeze" messages. "Full frame freeze" messages are used in video switching MCUs, to ensure a stable decoded displayed picture while switching among various input streams.

When the use of the video orientation (CVO) RTP header extension is not signaled as part of the SDP, H.264 implementations MAY send and SHOULD support proper interpretation of Display Orientation SEI messages.

Implementations MAY send and act upon "User data registered by Rec. ITU-T T.35" and "User data unregistered" messages. Even if they do

not act on them, implementations MUST be prepared to receive such messages without any ill effects.

Unless otherwise signaled, implementations that use H.264 MUST encode and decode pixels with a implied 1:1 (square) aspect ratio.

7. Security Considerations

This specification does not introduce any new mechanisms or security concerns beyond what is in the other documents it references. In WebRTC, video is protected using DTLS/SRTP. A complete discussion of the security considerations can be found in [I-D.ietf-rtcweb-security] and [I-D.ietf-rtcweb-security-arch]. Implementors should consider whether the use of variable bit rate video codecs are appropriate for their application, keeping in mind that the degree of inter-frame change (and, by inference, the amount of motion in the frame) may be deduced by an eavesdropper based on the video stream’s bit rate.

Implementors making use of H.264 are also advised to take careful note of the "Security Considerations" section of [RFC6184], paying special regard to the normative requirement pertaining to SEI messages.

8. IANA Considerations

This document requires no actions from IANA.

9. Acknowledgements

The author would like to thank Gaelle Martin-Cocher, Stephan Wenger, and Bernard Aboba for their detailed feedback and assistance with this document. Thanks to Cullen Jennings for providing text and review, and to Russ Housley for a careful final review. This draft includes text from draft-cbran-rtcweb-codec.

10. References

10.1. Normative References

[H264] ITU-T Recommendation H.264, "Advanced video coding for generic audiovisual services (V9)", February 2014, <http://www.itu.int/rec/T-REC-H.264>.

[HSUP1] ITU-T Recommendation H.Sup1, "Application profile - Sign language and lip-reading real-time conversation using low bit rate video communication", May 1999, <http://www.itu.int/rec/T-REC-H.Sup1>.

[I-D.ietf-payload-vp8]
Westin, P., Lundin, H., Glover, M., Uberti, J., and F. Galligan, "RTP Payload Format for VP8 Video", draft-ietf-payload-vp8-16 (work in progress), June 2015.

[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.

[IEC23001-8]
ISO/IEC 23001-8:2013/DCOR1, "Coding independent media description code points", 2013, <http://standards.iso.org/ittf/PubliclyAvailableStandards/c062088_ISO_IEC_23001-8_2013.zip>.

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

[RFC6184] Wang, Y.-K., Even, R., Kristensen, T., and R. Jesup, "RTP Payload Format for H.264 Video", RFC 6184, May 2011.

[RFC6236] Johansson, I. and K. Jung, "Negotiation of Generic Image Attributes in the Session Description Protocol (SDP)", RFC 6236, May 2011.

[RFC6386] Bankoski, J., Koleszar, J., Quillio, L., Salonen, J., Wilkins, P., and Y. Xu, "VP8 Data Format and Decoding Guide", RFC 6386, November 2011.

[SRGB] IEC 61966-2-1, "Multimedia systems and equipment - Colour measurement and management - Part 2-1: Colour management - Default RGB colour space - sRGB.", October 1999, <http://www.colour.org/tc8-05/Docs/colorspace/61966-2-1.pdf>.

[TS26.114]
3GPP TS 26.114 V12.8.0, "3rd Generation Partnership Project; Technical Specification Group Services and System Aspects; IP Multimedia Subsystem (IMS); Multimedia Telephony; Media handling and interaction (Release 12)", December 2014, <http://www.3gpp.org/DynaReport/26114.htm>.

10.2. Informative References

[I-D.ietf-rtcweb-rtp-usage]

Perkins, C., Westerlund, M., and J. Ott, "Web Real-Time Communication (WebRTC): Media Transport and Use of RTP", draft-ietf-rtcweb-rtp-usage-24 (work in progress), May 2015.

[I-D.ietf-rtcweb-security-arch]
Rescorla, E., "WebRTC Security Architecture", draft-ietf-rtcweb-security-arch-11 (work in progress), March 2015.

[I-D.ietf-rtcweb-security]
Rescorla, E., "Security Considerations for WebRTC", draft-ietf-rtcweb-security-08 (work in progress), February 2015.

Author's Address

Adam Roach
Mozilla
\
Dallas
US

Phone: +1 650 903 0800 x863
Email: adam@nostrum.com

Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: October 9, 2015

B. Schwartz
J. Uberti
Google

April 7, 2015

Recursively Encapsulated TURN (RETURN) for Connectivity and Privacy in WebRTC
draft-schwartz-rtcweb-return-06

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 October 9, 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 the 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

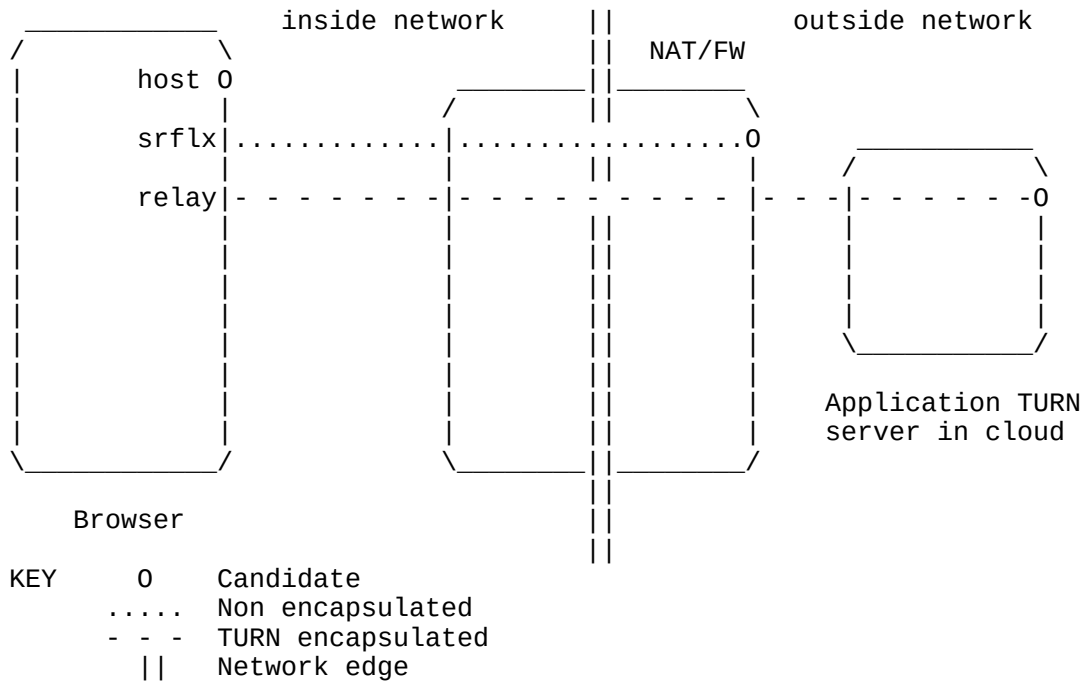

Figure 1: Basic WebRTC ICE Candidates with TURN Server

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.

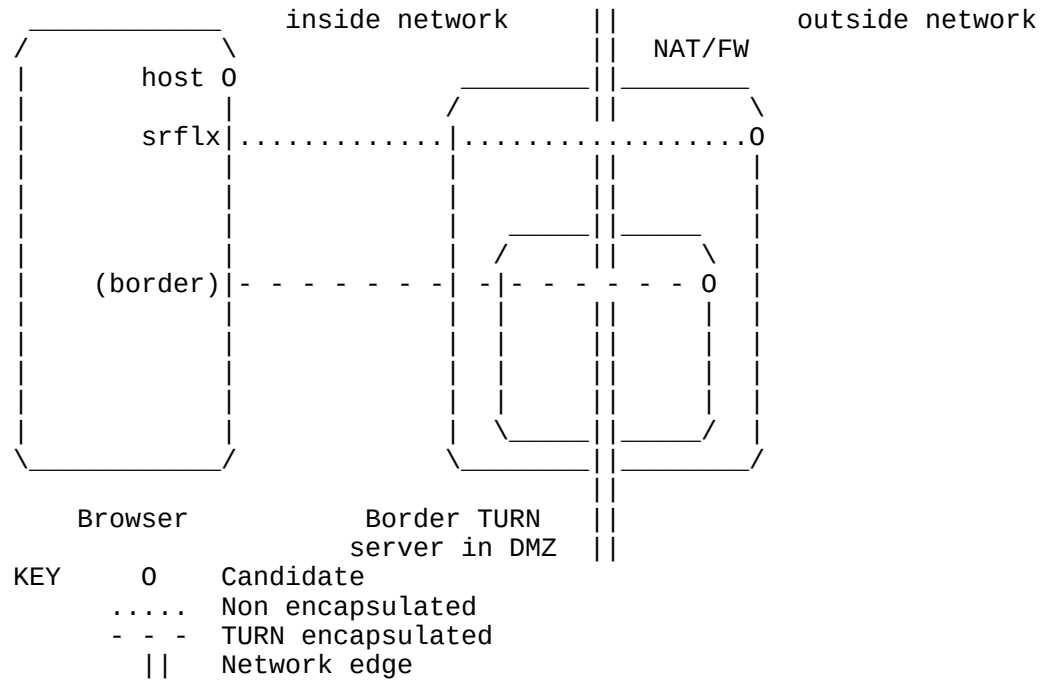

Figure 2: WebRTC ICE Candidates with DMZ TURN Server

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.

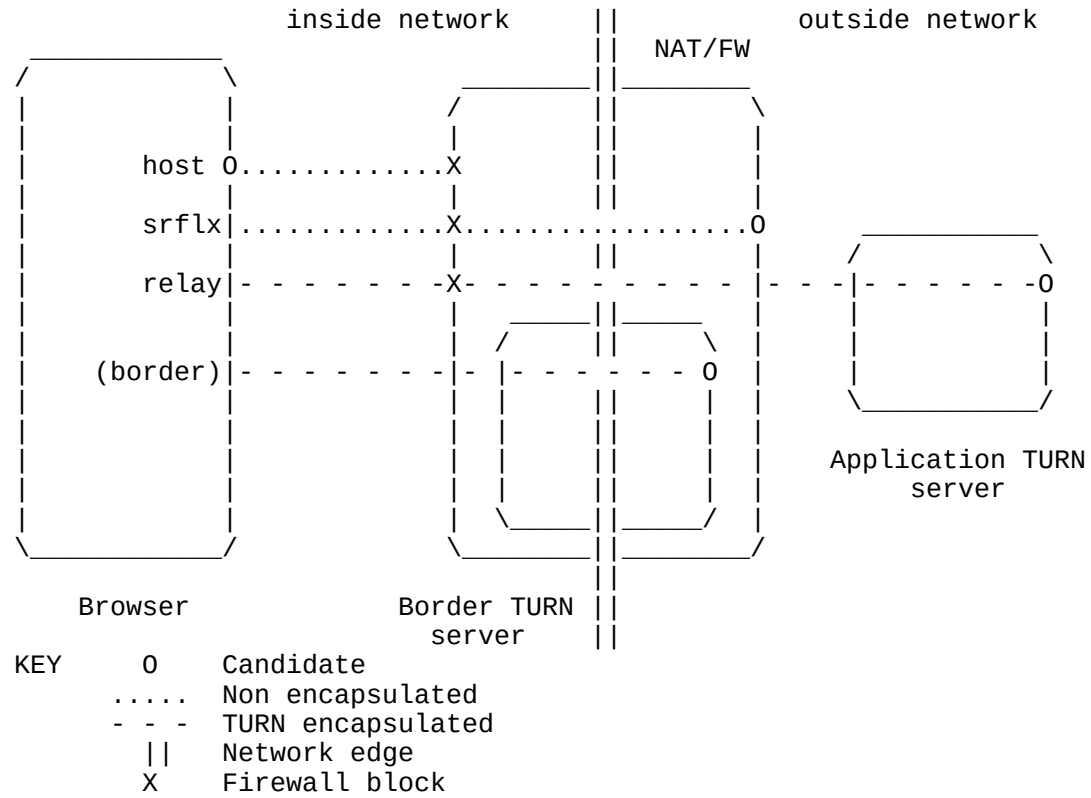

Figure 3: WebRTC ICE Candidates with Application and Border TURN Servers

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.

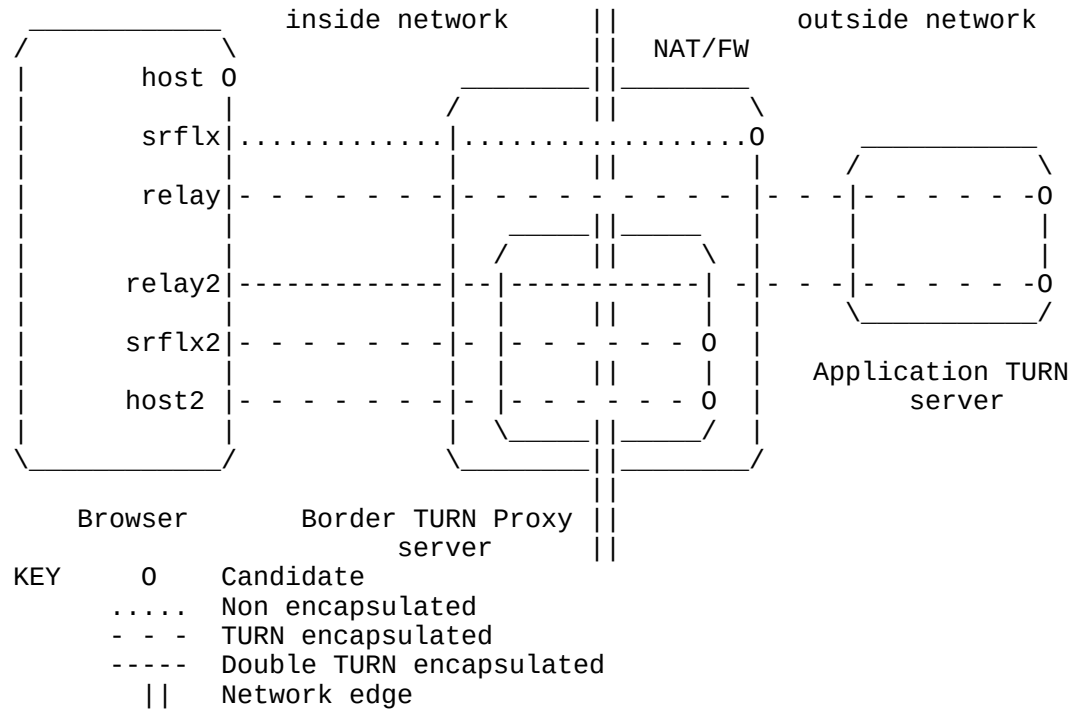

Figure 4: WebRTC ICE Candidates with Application TURN and Border TURN Proxy Servers

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.

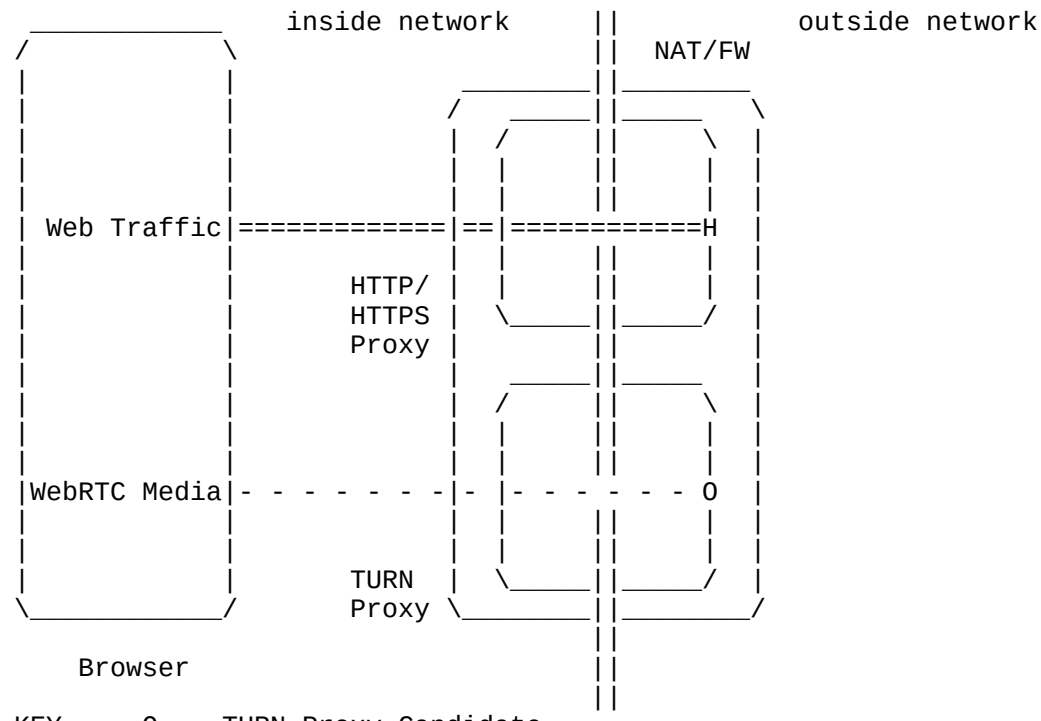

Figure 5: Similarity between HTTP/HTTPS Proxy and TURN Proxy

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

Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: April 30, 2015

J. Uberti
Google

October 27, 2014

WebRTC Forward Error Correction Requirements
draft-uberti-rtcweb-fec-00

Abstract

This document makes recommendations for how Forward Error Correction (FEC) should be used by WebRTC applications.

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 April 30, 2015.

Copyright Notice

Table of Contents

1. Introduction

In situations where packet loss is high, or media quality must be perfect, Forward Error Correction (FEC) can be used to proactively recover from packet losses. This document describes what FEC mechanisms should be used by WebRTC client implementations.

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].

3. Types of FEC

By its name, FEC describes the sending of redundant information in an outgoing packet stream so that information can still be recovered even in the face of packet loss. There are multiple ways in which this can be accomplished; this section enumerates the various mechanisms and describes their tradeoffs.

3.1. Separate FEC Stream

This approach, as described in [RFC5956], Section 4.3, sends FEC packets as an independent SSRC-multiplexed stream, with its own SSRC and payload type. While by far the most flexible, each FEC packet will have its own IP+UDP+RTP+FEC header, leading to additional overhead of the FEC stream.

3.2. Redundant Encoding

This approach, as descibed in [RFC2198], allows for redundant data to be piggybacked on an existing primary encoding in a single packet. This redundant data may be an exact copy of a previous packet, or for codecs that support variable-bitrate encodings, possibly a smaller, lower-quality representation. Since there is only a single set of packet headers, this allows for a very efficient representation of primary + redundant data. However, this savings is only realized when the two encodings both fit into a single packet (i.e. less than a MTU). This approach is also only applicable to audio content.

3.3. Codec-Specific In-band FEC

Some audio codecs, notably Opus [RFC6716], support their own in-band FEC mechanism, where FEC data is included in the codec payload. In the case of Opus specifically, packets deemed as important are re-encoded at a lower bitrate and added to the subsequent packet, allowing partial recovery of a lost packet. See [RFC6716], Section 2.1.7 for details.

4. FEC for Audio Content

The following section provides guidance on how to best use FEC for transmitting audio data. As indicated in Section 7 below, FEC should only be activated if network conditions warrant it, or upon explicit application request.

4.1. Recommended Mechanism

When using the Opus codec in its default (hybrid) mode, use of the built-in Opus FEC mechanism is RECOMMENDED. This provides reasonable protection of the audio stream against typical losses, with moderate overhead. [TODO: add stats] Note though that this mechanism only protects the SILK layer of the Opus codec; the CELT portion is not protected. This is not an issue when Opus is running in hybrid mode, as the lower frequencies will still be able to be recovered, with minimal quality impact.

When using Opus in CELT mode, or other variable-bitrate codecs, use of [RFC2198] redundant encoding with a lower-fidelity version of the previous packet is RECOMMENDED. When using Opus specifically, the lower-fidelity version can simply be a truncated version of the previous Opus packet. [TODO: decide exact truncated size] This provides reasonable protection of the payload with minimal overhead.

When using constant-bitrate codecs, e.g. PCMU, use of [RFC2198] redundant encoding is NOT RECOMMENDED, as this will result in a potentially significant bitrate increase. Furthermore, suddenly increasing the bitrate to deal with packet losses may actually make things worse.

Because of the lower packet rate of audio encodings, usually a single packet per frame, use of a separate FEC stream comes with a higher overhead than other mechanisms, and therefore is NOT RECOMMENDED.

4.2. Negotiating Support

Support for redundant encoding can be indicated by offering "red" as a supported payload type in the offer. Answerers can reject the use of redundant encoding by not including "red" as a supported payload type in the answer.

Support for codec-specific FEC mechanisms are typically indicated via "a=fmtp" parameters. For Opus specifically, this is controlled by the "useinbandfec=1" parameter, as specified in [I-D.ietf-payload-rtp-opus]. These parameters are declarative and can be negotiated separately for either media direction.

5. FEC for Video Content

The following section provides guidance on how to best use FEC for transmitting video data. As indicated in Section 7 below, FEC should only be activated if network conditions warrant it, or upon explicit application request.

5.1. Recommended Mechanism

For video content, use of a separate FEC stream with the RTP payload format described in [I-D.singh-payload-rtp-1d2d-parity-scheme] is RECOMMENDED. The receiver can demultiplex the incoming FEC stream by SSRC and correlate it with the primary stream via the ssrc-group mechanism.

Note that this only allows the FEC stream to protect a single primary stream. Support for protecting multiple primary streams with a

single FEC stream is complicated by WebRTC's 1-m-line-per-stream policy and requires further study.

5.2. Negotiating Support

To offer support for a separate FEC stream, the offerer MUST offer one of the formats described in [I-D.singh-payload-rtp-1d2d-parity-scheme], Section 5.1, as well as a ssrc-group with "FEC-FR" semantics as described in [RFC5956], Section 4.3.

Answerers can reject the use of FEC by not including FEC payloads in the answer.

6. Implementation Requirements

To support the functionality recommended above, implementations MUST support the redundant encoding mechanism described in [RFC2198] and the FEC mechanism described in [RFC5956] and [I-D.singh-payload-rtp-1d2d-parity-scheme].

Implementations MAY support additional FEC mechanisms if desired, e.g. [RFC5109].

7. Adaptive Use of FEC

Since use of FEC causes redundant data to be transmitted, this will lead to less bandwidth available for the primary encoding, when in a bandwidth-constrained environment. Given this, WebRTC implementations SHOULD only transmit FEC data when network conditions indicate that this is advisable (e.g. by monitoring transmit packet loss data from RTCP Receiver Reports), or the application indicates it is willing to pay a quality penalty to proactively avoid losses.

8. Security Considerations

TODO

9. IANA Considerations

This document requires no actions from IANA.

10. Acknowledgements

Several people provided significant input into this document, including Jonathan Lennox, Giri Mandyam, Varun Singh, Tim Terriberry, and Mo Zanaty.

11. References

11.1. Normative References

[I-D.singh-payload-rtp-1d2d-parity-scheme]
Singh, V., Begen, A., and M. Zanaty, "RTP Payload Format for Non-Interleaved and Interleaved Parity Forward Error Correction (FEC)", draft-singh-payload-rtp-1d2d-parity-scheme-00 (work in progress), October 2014.

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

[RFC2198] Perkins, C., Kouvelas, I., Hodson, O., Hardman, V., Handley, M., Bolot, J., Vega-Garcia, A., and S. Fosse-Parisis, "RTP Payload for Redundant Audio Data", RFC 2198, September 1997.

[RFC5956] Begen, A., "Forward Error Correction Grouping Semantics in the Session Description Protocol", RFC 5956, September 2010.

11.2. Informative References

[I-D.ietf-payload-rtp-opus]
Spittka, J., Vos, K., and J. Valin, "RTP Payload Format for Opus Speech and Audio Codec", draft-ietf-payload-rtp-opus-03 (work in progress), July 2014.

[RFC5109] Li, A., "RTP Payload Format for Generic Forward Error Correction", RFC 5109, December 2007.

[RFC6716] Valin, JM., Vos, K., and T. Terriberry, "Definition of the Opus Audio Codec", RFC 6716, September 2012.

Appendix A. Change log

Changes in draft -00:

- Initial version, from sidebar conversation at IETF 90.

Author’s Address

Justin Uberti
Google
747 6th Ave S
Kirkland, WA 98033
USA

Email: justin@uberti.name