

Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: January 17, 2020

J. Uberti
Google
Jul 16, 2019

WebRTC Forward Error Correction Requirements
draft-ietf-rtcweb-fec-10

Abstract

This document provides information and requirements for how Forward Error Correction (FEC) should be used by WebRTC implementations.

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 <https://datatracker.ietf.org/drafts/current/>.

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

This Internet-Draft will expire on January 17, 2020.

Copyright Notice

Copyright (c) 2019 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<https://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction	2
2. Terminology	2
3. Types of FEC	2
3.1. Separate FEC Stream	3
3.2. Redundant Encoding	3
3.3. Codec-Specific In-band FEC	3
4. FEC for Audio Content	4
4.1. Recommended Mechanism	4
4.2. Negotiating Support	5
5. FEC for Video Content	5
5.1. Recommended Mechanism	5
5.2. Negotiating Support	6
6. FEC for Application Content	6
7. Implementation Requirements	7
8. Adaptive Use of FEC	7
9. Security Considerations	8
10. IANA Considerations	8
11. Acknowledgements	8
12. References	8
12.1. Normative References	8
12.2. Informative References	9
Appendix A. Change log	11
Author's Address	13

1. Introduction

In situations where packet loss is high, or perfect media quality is essential, Forward Error Correction (FEC) can be used to proactively recover from packet losses. This specification provides guidance on which FEC mechanisms to use, and how to use them, for WebRTC implementations.

2. Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

3. Types of FEC

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 this can be

accomplished for RTP media streams [RFC3550]; this section enumerates the various mechanisms available and describes their tradeoffs.

3.1. Separate FEC Stream

This approach, as described in [RFC5956], Section 4.3, sends FEC packets as an independent RTP stream with its own synchronization source (SSRC, [RFC3550]) and payload type, multiplexed with the primary encoding. While this approach can protect multiple packets of the primary encoding with a single FEC packet, each FEC packet will have its own IP+UDP+RTP+FEC header, and this overhead can be excessive in some cases, e.g., when protecting each primary packet with a FEC packet.

This approach allows for recovery of entire RTP packets, including the full RTP header.

3.2. Redundant Encoding

This approach, as described in [RFC2198], allows for redundant data to be piggybacked on an existing primary encoding, all in a single packet. This redundant data may be an exact copy of a previous payload, or for codecs that support variable-bitrate encodings, possibly a smaller, lower-quality representation. In certain cases, the redundant data could include encodings of multiple prior audio frames.

Since there is only a single set of packet headers, this approach allows for a very efficient representation of primary + redundant data. However, this savings is only realized when the data all fits into a single packet (i.e. the size is less than a MTU). As a result, this approach is generally not useful for video content.

As described in [RFC2198], Section 4, this approach cannot recover certain parts of the RTP header, including the marker bit, CSRC information, and header extensions.

3.3. Codec-Specific In-band FEC

Some audio codecs, notably Opus [RFC6716] and AMR [RFC4867], support their own in-band FEC mechanism, where redundant data is included in the codec payload. This is similar to the redundant encoding mechanism described above, but as it adds no additional framing, it can be slightly more efficient.

For Opus, audio frames deemed important are re-encoded at a lower bitrate and appended to the next payload, allowing partial recovery of a lost packet. This scheme is fairly efficient; experiments

performed indicate that when Opus FEC is used, the overhead imposed is only about 20-30%, depending on the amount of protection needed. Note that this mechanism can only carry redundancy information for the immediately preceding audio frame; as such the decoder cannot fully recover multiple consecutive lost packets, which can be a problem on wireless networks. See [RFC6716], Section 2.1.7, and this Opus mailing list post [OpusFEC] for more details.

For AMR/AMR-WB, packets can contain copies or lower-quality encodings of multiple prior audio frames. See [RFC4867], Section 3.7.1 for details on this mechanism.

In-band FEC mechanisms cannot recover any of the RTP header.

4. FEC for Audio Content

The following section provides guidance on how to best use FEC for transmitting audio data. As indicated in Section 8 below, FEC should only be activated if network conditions warrant it, or upon explicit application request.

4.1. Recommended Mechanism

When using variable-bitrate codecs without an internal FEC, redundant encoding (as described in Section 3.2) with lower-fidelity version(s) of the previous packet(s) is RECOMMENDED. This provides reasonable protection of the payload with only moderate bitrate increase, as the redundant encodings can be significantly smaller than the primary encoding.

When using the Opus codec, use of the built-in Opus FEC mechanism is RECOMMENDED. This provides reasonable protection of the audio stream against individual losses, with minimal overhead. Note that, as indicated above, the built-in Opus FEC only provides single-frame redundancy; if multi-packet protection is needed, the aforementioned redundant encoding with reduced-bitrate Opus encodings SHOULD be used instead.

When using the AMR/AMR-WB codecs, use of their built-in FEC mechanism is RECOMMENDED. This provides slightly more efficient protection of the audio stream than redundant encoding.

When using constant-bitrate codecs, e.g., PCMU [RFC5391], redundant encoding MAY be used, but this will result in a potentially significant bitrate increase, and suddenly increasing bitrate to deal with losses from congestion 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.

As mentioned above, the recommended mechanisms do not allow recovery of parts of the RTP header that may be important in certain audio applications, e.g., CSRCs and RTP header extensions like those specified in [RFC6464] and [RFC6465]. Implementations SHOULD account for this and attempt to approximate this information, using an approach similar to those described in [RFC2198], Section 4, and [RFC6464], Section 5.

4.2. Negotiating Support

Support for redundant encoding of a given RTP stream SHOULD be indicated by including audio/red [RFC2198] as an additional supported media type for the associated m= section in the SDP offer [RFC3264]. Answerers can reject the use of redundant encoding by not including the audio/red media type in the corresponding m= section in the SDP answer.

Support for codec-specific FEC mechanisms are typically indicated via "a=fmtp" parameters.

For Opus, a receiver MUST indicate that it is prepared to use incoming FEC data with the "useinbandfec=1" parameter, as specified in [RFC7587]. This parameter is declarative and can be negotiated separately for either media direction.

For AMR/AMR-WB, support for redundant encoding, and the maximum supported depth, are controlled by the 'max-red' parameter, as specified in [RFC4867], Section 8.1. Receivers MUST include this parameter, and set it to an appropriate value, as specified in [TS.26114], Table 6.3.

5. FEC for Video Content

The following section provides guidance on how to best use FEC for transmitting video data. As indicated in Section 8 below, FEC should only be activated if network conditions warrant it, or upon explicit application request.

5.1. Recommended Mechanism

Video frames, due to their size, often require multiple RTP packets. As discussed above, a separate FEC stream can protect multiple packets with a single FEC packet. In addition, the "flexfec" FEC mechanism described in [I-D.ietf-payload-flexible-fec-scheme] is also

capable of protecting multiple RTP streams via a single FEC stream, including all the streams that are part of a BUNDLE [I-D.ietf-mmusic-sdp-bundle-negotiation] group. As a result, for video content, use of a separate FEC stream with the flexfec RTP payload format is RECOMMENDED.

To process the incoming FEC stream, the receiver can demultiplex it by SSRC, and then correlate it with the appropriate primary stream(s) via the CSRC(s) present in the RTP header of flexfec repair packets, or the SSRC field present in the FEC header of flexfec retransmission packets.

5.2. Negotiating Support

Support for a SSRC-multiplexed flexfec stream to protect a given RTP stream SHOULD be indicated by including one of the formats described in [I-D.ietf-payload-flexible-fec-scheme], Section 5.1.2, as an additional supported media type for the associated m= section in the SDP offer [RFC3264]. As mentioned above, when BUNDLE is used, only a single flexfec repair stream will be created for each BUNDLE group, even if flexfec is negotiated for each primary stream.

Answerers can reject the use of SSRC-multiplexed FEC, by not including the offered FEC formats in the corresponding m= section in the SDP answer.

Use of FEC-only m-lines, and grouping using the SDP group mechanism as described in [RFC5956], Section 4.1 is not currently defined for WebRTC, and SHOULD NOT be offered.

Answerers SHOULD reject any FEC-only m-lines, unless they specifically know how to handle such a thing in a WebRTC context (perhaps defined by a future version of the WebRTC specifications).

6. FEC for Application Content

WebRTC also supports the ability to send generic application data, and provides transport-level retransmission mechanisms to support full and partial (e.g. timed) reliability. See [I-D.ietf-rtcweb-data-channel] for details.

Because the application can control exactly what data to send, it has the ability to monitor packet statistics and perform its own application-level FEC, if necessary.

As a result, this document makes no recommendations regarding FEC for the underlying data transport.

7. Implementation Requirements

To support the functionality recommended above, implementations **MUST** be able to receive and make use of the relevant FEC formats for their supported audio codecs, and **MUST** indicate this support, as described in Section 4. Use of these formats when sending, as mentioned above, is **RECOMMENDED**.

The general FEC mechanism described in [I-D.ietf-payload-flexible-fec-scheme] **SHOULD** also be supported, as mentioned in Section 5.

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

8. Adaptive Use of FEC

Because use of FEC always causes redundant data to be transmitted, and the total amount of data must remain within any bandwidth limits indicated by congestion control and the receiver, this will lead to less bandwidth available for the primary encoding, even when the redundant data is not being used. This is in contrast to methods like RTX [RFC4588] or flexfec's retransmission mode ([I-D.ietf-payload-flexible-fec-scheme], Section 1.1.7), which only transmit redundant data when necessary, at the cost of an extra roundtrip and thereby increased media latency.

Given this, WebRTC implementations **SHOULD** prefer using RTX or flexfec retransmissions instead of FEC when the connection RTT is within the application's latency budget, and otherwise **SHOULD** only transmit the amount of FEC needed to protect against the observed packet loss (which can be determined, e.g., by monitoring transmit packet loss data from RTCP Receiver Reports [RFC3550]), unless the application indicates it is willing to pay a quality penalty to proactively avoid losses.

Note that when probing bandwidth, i.e., speculatively sending extra data to determine if additional link capacity exists, FEC data **SHOULD** be used as the additional data. Given that extra data is going to be sent regardless, it makes sense to have that data protect the primary payload; in addition, FEC can typically be applied in a way that increases bandwidth only modestly, which is necessary when probing.

When using FEC with layered codecs, e.g., [RFC6386], where only base layer frames are critical to the decoding of future frames, implementations **SHOULD** only apply FEC to these base layer frames.

Finally, it should be noted that although applying redundancy is often useful in protecting a stream against packet loss, if the loss is caused by network congestion, the additional bandwidth used by the redundant data may actually make the situation worse, and can lead to significant degradation of the network.

9. Security Considerations

In the WebRTC context, FEC is specifically concerned with recovering data from lost packets; any corrupted packets will be discarded by the SRTP [RFC3711] decryption process. Therefore, as described in [RFC3711], Section 10, the default processing when using FEC with SRTP is to perform FEC followed by SRTP at the sender, and SRTP followed by FEC at the receiver. This ordering is used for all the SRTP Protection Profiles used in DTLS-SRTP [RFC5763], which are enumerated in [RFC5764], Section 4.1.2.

Additional security considerations for each individual FEC mechanism are enumerated in their respective documents.

10. IANA Considerations

This document requires no actions from IANA.

11. Acknowledgements

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

12. References

12.1. Normative References

- [I-D.ietf-payload-flexible-fec-scheme]
Zanaty, M., Singh, V., Begen, A., and G. Mandyam, "RTP Payload Format for Flexible Forward Error Correction (FEC)", draft-ietf-payload-flexible-fec-scheme-20 (work in progress), May 2019.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<https://www.rfc-editor.org/info/rfc2119>>.

- [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, DOI 10.17487/RFC2198, September 1997, <<https://www.rfc-editor.org/info/rfc2198>>.
- [RFC3264] Rosenberg, J. and H. Schulzrinne, "An Offer/Answer Model with Session Description Protocol (SDP)", RFC 3264, DOI 10.17487/RFC3264, June 2002, <<https://www.rfc-editor.org/info/rfc3264>>.
- [RFC4867] Sjöberg, J., Westerlund, M., Lankaniemi, A., and Q. Xie, "RTP Payload Format and File Storage Format for the Adaptive Multi-Rate (AMR) and Adaptive Multi-Rate Wideband (AMR-WB) Audio Codecs", RFC 4867, DOI 10.17487/RFC4867, April 2007, <<https://www.rfc-editor.org/info/rfc4867>>.
- [RFC5956] Begen, A., "Forward Error Correction Grouping Semantics in the Session Description Protocol", RFC 5956, DOI 10.17487/RFC5956, September 2010, <<https://www.rfc-editor.org/info/rfc5956>>.
- [RFC7587] Spittka, J., Vos, K., and JM. Valin, "RTP Payload Format for the Opus Speech and Audio Codec", RFC 7587, DOI 10.17487/RFC7587, June 2015, <<https://www.rfc-editor.org/info/rfc7587>>.
- [RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <<https://www.rfc-editor.org/info/rfc8174>>.
- [TS.26114] 3GPP, "IP Multimedia Subsystem (IMS); Multimedia telephony; Media handling and interaction", 3GPP TS 26.114 15.0.0, September 2017.

12.2. Informative References

- [I-D.ietf-mmusic-sdp-bundle-negotiation] Holmberg, C., Alvestrand, H., and C. Jennings, "Negotiating Media Multiplexing Using the Session Description Protocol (SDP)", draft-ietf-mmusic-sdp-bundle-negotiation-54 (work in progress), December 2018.
- [I-D.ietf-rtcweb-data-channel] Jesup, R., Loreto, S., and M. Tuexen, "WebRTC Data Channels", draft-ietf-rtcweb-data-channel-13 (work in progress), January 2015.

- [OpusFEC] Terriberry, T., "Opus FEC", January 2013.
- [RFC3550] Schulzrinne, H., Casner, S., Frederick, R., and V. Jacobson, "RTP: A Transport Protocol for Real-Time Applications", STD 64, RFC 3550, DOI 10.17487/RFC3550, July 2003, <<https://www.rfc-editor.org/info/rfc3550>>.
- [RFC3711] Baugher, M., McGrew, D., Naslund, M., Carrara, E., and K. Norrman, "The Secure Real-time Transport Protocol (SRTP)", RFC 3711, DOI 10.17487/RFC3711, March 2004, <<https://www.rfc-editor.org/info/rfc3711>>.
- [RFC4588] Rey, J., Leon, D., Miyazaki, A., Varsa, V., and R. Hakenberg, "RTP Retransmission Payload Format", RFC 4588, DOI 10.17487/RFC4588, July 2006, <<https://www.rfc-editor.org/info/rfc4588>>.
- [RFC5109] Li, A., Ed., "RTP Payload Format for Generic Forward Error Correction", RFC 5109, DOI 10.17487/RFC5109, December 2007, <<https://www.rfc-editor.org/info/rfc5109>>.
- [RFC5391] Sollaud, A., "RTP Payload Format for ITU-T Recommendation G.711.1", RFC 5391, DOI 10.17487/RFC5391, November 2008, <<https://www.rfc-editor.org/info/rfc5391>>.
- [RFC5763] Fischl, J., Tschofenig, H., and E. Rescorla, "Framework for Establishing a Secure Real-time Transport Protocol (SRTP) Security Context Using Datagram Transport Layer Security (DTLS)", RFC 5763, DOI 10.17487/RFC5763, May 2010, <<https://www.rfc-editor.org/info/rfc5763>>.
- [RFC5764] McGrew, D. and E. Rescorla, "Datagram Transport Layer Security (DTLS) Extension to Establish Keys for the Secure Real-time Transport Protocol (SRTP)", RFC 5764, DOI 10.17487/RFC5764, May 2010, <<https://www.rfc-editor.org/info/rfc5764>>.
- [RFC6386] Bankoski, J., Koleszar, J., Quillio, L., Salonen, J., Wilkins, P., and Y. Xu, "VP8 Data Format and Decoding Guide", RFC 6386, DOI 10.17487/RFC6386, November 2011, <<https://www.rfc-editor.org/info/rfc6386>>.
- [RFC6464] Lennox, J., Ed., Iovov, E., and E. Marocco, "A Real-time Transport Protocol (RTP) Header Extension for Client-to-Mixer Audio Level Indication", RFC 6464, DOI 10.17487/RFC6464, December 2011, <<https://www.rfc-editor.org/info/rfc6464>>.

- [RFC6465] Iovov, E., Ed., Marocco, E., Ed., and J. Lennox, "A Real-time Transport Protocol (RTP) Header Extension for Mixer-to-Client Audio Level Indication", RFC 6465, DOI 10.17487/RFC6465, December 2011, <<https://www.rfc-editor.org/info/rfc6465>>.
- [RFC6716] Valin, JM., Vos, K., and T. Terriberry, "Definition of the Opus Audio Codec", RFC 6716, DOI 10.17487/RFC6716, September 2012, <<https://www.rfc-editor.org/info/rfc6716>>.

Appendix A. Change log

Changes in draft -10:

- o Additional editorial changes from IETF LC.

Changes in draft -09:

- o Editorial changes from IETF LC.
- o Added new reference for Opus FEC.

Changes in draft -08:

- o Switch to RFC 8174 boilerplate.

Changes in draft -07:

- o Clarify how bandwidth management interacts with FEC.
- o Make 3GPP reference normative.

Changes in draft -06:

- o Discuss how multiple streams can be protected by a single FlexFEC stream.
- o Discuss FEC for bandwidth probing.
- o Add note about recovery of RTP headers and header extensions.
- o Add note about FEC/SRTP ordering.
- o Clarify flexfec demux text, and mention retransmits.
- o Clarify text regarding offers/answers.
- o Make RFC2198 support SHOULD strength.

- o Clean up references.

Changes in draft -05:

- o No changes.

Changes in draft -04:

- o Discussion of layered codecs.
- o Discussion of RTX.
- o Clarified implementation requirements.
- o FlexFEC MUST -> SHOULD.
- o Clarified AMR max-red handling.
- o Updated references.

Changes in draft -03:

- o Added overhead stats for Opus.
- o Expanded discussion of multi-packet FEC for Opus.
- o Added discussion of AMR/AMR-WB.
- o Removed discussion of ssrc-group.
- o Referenced the data channel doc.
- o Referenced the RTP/RTCP RFC.
- o Several small edits based on feedback from Magnus.

Changes in draft -02:

- o Expanded discussion of FEC-only m-lines, and how they should be handled in offers and answers.

Changes in draft -01:

- o Tweaked abstract/intro text that was ambiguously normative.
- o Removed text on FEC for Opus in CELT mode.

- o Changed RFC 2198 recommendation for PCMU to be MAY instead of NOT RECOMMENDED, based on list feedback.
- o Explicitly called out application data as something not addressed in this document.
- o Updated flexible-fec reference.

Changes in draft -00:

- o Initial version, from sidebar conversation at IETF 90.

Author's Address

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

Email: justin@uberti.name

Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: January 3, 2020

J. Uberti
Google
July 2, 2019

WebRTC IP Address Handling Requirements
draft-ietf-rtcweb-ip-handling-12

Abstract

This document provides information and requirements for how IP addresses should be handled by WebRTC implementations.

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 <https://datatracker.ietf.org/drafts/current/>.

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

This Internet-Draft will expire on January 3, 2020.

Copyright Notice

Copyright (c) 2019 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<https://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction	2
2. Terminology	2
3. Problem Statement	2
4. Goals	4
5. Detailed Design	5
5.1. Principles	5
5.2. Modes and Recommendations	5
6. Implementation Guidance	7
6.1. Ensuring Normal Routing	7
6.2. Determining Associated Local Addresses	7
7. Application Guidance	8
8. Security Considerations	8
9. IANA Considerations	8
10. Acknowledgements	8
11. References	8
11.1. Normative References	8
11.2. Informative References	9
Appendix A. Change log	10
Author's Address	12

1. Introduction

One of WebRTC's key features is its support of peer-to-peer connections. However, when establishing such a connection, which involves connection attempts from various IP addresses, WebRTC may allow a web application to learn additional information about the user compared to an application that only uses the Hypertext Transfer Protocol (HTTP) [RFC7230]. This may be problematic in certain cases. This document summarizes the concerns, and makes recommendations on how WebRTC implementations should best handle the tradeoff between privacy and media performance.

2. Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119][RFC8174] when, and only when, they appear in all capitals, as shown here.

3. Problem Statement

In order to establish a peer-to-peer connection, WebRTC implementations use Interactive Connectivity Establishment (ICE) [RFC8445], which attempts to discover multiple IP addresses using techniques such as Session Traversal Utilities for NAT (STUN)

[RFC5389] and Traversal Using Relays around NAT (TURN) [RFC5766], and then checks the connectivity of each local-address-remote-address pair in order to select the best one. The addresses that are collected usually consist of an endpoint's private physical or virtual addresses and its public Internet addresses.

These addresses are provided to the web application so that they can be communicated to the remote endpoint for its checks. This allows the application to learn more about the local network configuration than it would from a typical HTTP scenario, in which the web server would only see a single public Internet address, i.e., the address from which the HTTP request was sent.

The information revealed falls into three categories:

1. If the client is multihomed, additional public IP addresses for the client can be learned. In particular, if the client tries to hide its physical location through a Virtual Private Network (VPN), and the VPN and local OS support routing over multiple interfaces (a "split-tunnel" VPN), WebRTC can discover not only the public address for the VPN, but also the ISP public address over which the VPN is running.
2. If the client is behind a Network Address Translator (NAT), the client's private IP addresses, often [RFC1918] addresses, can be learned.
3. If the client is behind a proxy (a client-configured "classical application proxy", as defined in [RFC1919], Section 3), but direct access to the Internet is permitted, WebRTC's STUN checks will bypass the proxy and reveal the public IP address of the client. This concern also applies to the "enterprise TURN server" scenario described in [RFC7478], Section 2.3.5.1, if, as above, direct Internet access is permitted. However, when the term "proxy" is used in this document, it is always in reference to an [RFC1919] proxy server.

Of these three concerns, the first is the most significant, because for some users, the purpose of using a VPN is for anonymity. However, different VPN users will have different needs, and some VPN users (e.g., corporate VPN users) may in fact prefer WebRTC to send media traffic directly, i.e., not through the VPN.

The second concern is less significant but valid nonetheless. The core issue is that web applications can learn about addresses that are not exposed to the internet; typically these addresses are IPv4, but they can also be IPv6, as in the case of NAT64 [RFC6146]. While disclosure of the [RFC4941] IPv6 addresses recommended by

[I-D.ietf-rtcweb-transports] is fairly benign due to their intentionally short lifetimes, IPv4 addresses present some challenges. Although private IPv4 addresses often contain minimal entropy (e.g., 192.168.0.2, a fairly common address), in the worst case, they can contain 24 bits of entropy with an indefinite lifetime. As such, they can be a fairly significant fingerprinting surface. In addition, intranet web sites can be attacked more easily when their IPv4 address range is externally known.

Private IP addresses can also act as an identifier that allows web applications running in isolated browsing contexts (e.g., normal and private browsing) to learn that they are running on the same device. This could allow the application sessions to be correlated, defeating some of the privacy protections provided by isolation. It should be noted that private addresses are just one potential mechanism for this correlation and this is an area for further study.

The third concern is the least common, as proxy administrators can already control this behavior through organizational firewall policy, and generally, forcing WebRTC traffic through a proxy server will have negative effects on both the proxy and on media quality.

Note also that these concerns predate WebRTC; Adobe Flash Player has provided similar functionality since the introduction of Real-Time Media Flow Protocol (RTMFP) support [RFC7016] in 2008.

4. Goals

WebRTC's support of secure peer-to-peer connections facilitates deployment of decentralized systems, which can have privacy benefits. As a result, blunt solutions that disable WebRTC or make it significantly harder to use are undesirable. This document takes a more nuanced approach, with the following goals:

- o Provide a framework for understanding the problem so that controls might be provided to make different tradeoffs regarding performance and privacy concerns with WebRTC.
- o Using that framework, define settings that enable peer-to-peer communications, each with a different balance between performance and privacy.
- o Finally, provide recommendations for default settings that provide reasonable performance without also exposing addressing information in a way that might violate user expectations.

5. Detailed Design

5.1. Principles

The key principles for our framework are stated below:

1. By default, WebRTC traffic should follow typical IP routing, i.e., WebRTC should use the same interface used for HTTP traffic, and only the system's 'typical' public addresses (or those of an enterprise TURN server, if present) should be visible to the application. However, in the interest of optimal media quality, it should be possible to enable WebRTC to make use of all network interfaces to determine the ideal route.
2. By default, WebRTC should be able to negotiate direct peer-to-peer connections between endpoints (i.e., without traversing a NAT or relay server) when such connections are possible. This ensures that applications that need true peer-to-peer routing for bandwidth or latency reasons can operate successfully.
3. It should be possible to configure WebRTC to not disclose private local IP addresses, to avoid the issues associated with web applications learning such addresses. This document does not require this to be the default state, as there is no currently defined mechanism that can satisfy this requirement as well as the aforementioned requirement to allow direct peer-to-peer connections.
4. By default, WebRTC traffic should not be sent through proxy servers, due to the media quality problems associated with sending WebRTC traffic over TCP, which is almost always used when communicating with such proxies, as well as proxy performance issues that may result from proxying WebRTC's long-lived, high-bandwidth connections. However, it should be possible to force WebRTC to send its traffic through a configured proxy if desired.

5.2. Modes and Recommendations

Based on these ideas, we define four specific modes of WebRTC behavior, reflecting different media quality/privacy tradeoffs:

- Mode 1: Enumerate all addresses: WebRTC MUST use all network interfaces to attempt communication with STUN servers, TURN servers, or peers. This will converge on the best media path, and is ideal when media performance is the highest priority, but it discloses the most information.

Mode 2: Default route + associated local addresses: WebRTC MUST follow the kernel routing table rules, which will typically cause media packets to take the same route as the application's HTTP traffic. If an enterprise TURN server is present, the preferred route MUST be through this TURN server. Once an interface has been chosen, the private IPv4 and IPv6 addresses associated with this interface MUST be discovered and provided to the application as host candidates. This ensures that direct connections can still be established in this mode.

Mode 3: Default route only: This is the the same as Mode 2, except that the associated private addresses MUST NOT be provided; the only IP addresses gathered are those discovered via mechanisms like STUN and TURN (on the default route). This may cause traffic to hairpin through a NAT, fall back to an application TURN server, or fail altogether, with resulting quality implications.

Mode 4: Force proxy: This is the same as Mode 3, but when the application's HTTP traffic is sent through a proxy, WebRTC media traffic MUST also be proxied. If the proxy does not support UDP (as is the case for all HTTP and most SOCKS [RFC1928] proxies), or the WebRTC implementation does not support UDP proxying, the use of UDP will be disabled, and TCP will be used to send and receive media through the proxy. Use of TCP will result in reduced media quality, in addition to any performance considerations associated with sending all WebRTC media through the proxy server.

Mode 1 MUST NOT be used unless user consent has been provided. The details of this consent are left to the implementation; one potential mechanism is to tie this consent to `getUserMedia` (device permissions) consent, described in [I-D.ietf-rtcweb-security-arch], Section 6.2. Alternatively, implementations can provide a specific mechanism to obtain user consent.

In cases where user consent has not been obtained, Mode 2 SHOULD be used.

These defaults provide a reasonable tradeoff that permits trusted WebRTC applications to achieve optimal network performance, but gives applications without consent (e.g., 1-way streaming or data channel applications) only the minimum information needed to achieve direct connections, as defined in Mode 2. However, implementations MAY choose stricter modes if desired, e.g., if a user indicates they want all WebRTC traffic to follow the default route.

Future documents may define additional modes and/or update the recommended default modes.

Note that the suggested defaults can still be used even for organizations that want all external WebRTC traffic to traverse a proxy or enterprise TURN server, simply by setting an organizational firewall policy that allows WebRTC traffic to only leave through the proxy or TURN server. This provides a way to ensure the proxy or TURN server is used for any external traffic, but still allows direct connections (and, in the proxy case, avoids the performance issues associated with forcing media through said proxy) for intra-organization traffic.

6. Implementation Guidance

This section provides guidance to WebRTC implementations on how to implement the policies described above.

6.1. Ensuring Normal Routing

When trying to follow typical IP routing, as required by Modes 2 and 3, the simplest approach is to bind() the sockets used for peer-to-peer connections to the wildcard addresses (0.0.0.0 for IPv4, :: for IPv6), which allows the OS to route WebRTC traffic the same way as it would HTTP traffic. STUN and TURN will work as usual, and host candidates can still be determined as mentioned below.

6.2. Determining Associated Local Addresses

When binding to a wildcard address, some extra work is needed to determine the associated local address required by Mode 2, which we define as the source address that would be used for any packets sent to the web application host (assuming that UDP and TCP get the same routing treatment). Use of the web application host as a destination ensures the right source address is selected, regardless of where the application resides (e.g., on an intranet).

First, the appropriate remote IPv4/IPv6 address is obtained by resolving the host component of the web application URI [RFC3986]. If the client is behind a proxy and cannot resolve these IPs via DNS, the address of the proxy can be used instead. Or, if the web application was loaded from a file:// URI [RFC8089], rather than over the network, the implementation can fall back to a well-known DNS name or IP address.

Once a suitable remote IP has been determined, the implementation can create a UDP socket, bind() it to the appropriate wildcard address, and then connect() to the remote IP. Generally, this results in the

socket being assigned a local address based on the kernel routing table, without sending any packets over the network.

Finally, the socket can be queried using `getsockname()` or the equivalent to determine the appropriate local address.

7. Application Guidance

The recommendations mentioned in this document may cause certain WebRTC applications to malfunction. In order to be robust in all scenarios, the following guidelines are provided for applications:

- o Applications SHOULD deploy a TURN server with support for both UDP and TCP connections to the server. This ensures that connectivity can still be established, even when Mode 3 or 4 are in use, assuming the TURN server can be reached.
- o Applications SHOULD detect when they don't have access to the full set of ICE candidates by checking for the presence of host candidates. If no host candidates are present, Mode 3 or 4 above is in use; this knowledge can be useful for diagnostic purposes.

8. Security Considerations

This document describes several potential privacy and security concerns associated with WebRTC peer-to-peer connections, and provides mechanisms and recommendations for WebRTC implementations to address these concerns.

9. IANA Considerations

This document requires no actions from IANA.

10. Acknowledgements

Several people provided input into this document, including Bernard Aboba, Harald Alvestrand, Youenn Fablet, Ted Hardie, Matthew Kaufmann, Eric Rescorla, Adam Roach, and Martin Thomson.

11. References

11.1. Normative References

- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<https://www.rfc-editor.org/info/rfc2119>>.

- [RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, DOI 10.17487/RFC3986, January 2005, <<https://www.rfc-editor.org/info/rfc3986>>.
- [RFC5389] Rosenberg, J., Mahy, R., Matthews, P., and D. Wing, "Session Traversal Utilities for NAT (STUN)", RFC 5389, DOI 10.17487/RFC5389, October 2008, <<https://www.rfc-editor.org/info/rfc5389>>.
- [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, DOI 10.17487/RFC5766, April 2010, <<https://www.rfc-editor.org/info/rfc5766>>.
- [RFC8089] Kerwin, M., "The "file" URI Scheme", RFC 8089, DOI 10.17487/RFC8089, February 2017, <<https://www.rfc-editor.org/info/rfc8089>>.
- [RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <<https://www.rfc-editor.org/info/rfc8174>>.
- [RFC8445] Keranen, A., Holmberg, C., and J. Rosenberg, "Interactive Connectivity Establishment (ICE): A Protocol for Network Address Translator (NAT) Traversal", RFC 8445, DOI 10.17487/RFC8445, July 2018, <<https://www.rfc-editor.org/info/rfc8445>>.

11.2. Informative References

- [I-D.ietf-rtcweb-security-arch] Rescorla, E., "WebRTC Security Architecture", draft-ietf-rtcweb-security-arch-18 (work in progress), February 2019.
- [I-D.ietf-rtcweb-transports] Alvestrand, H., "Transports for WebRTC", draft-ietf-rtcweb-transports-17 (work in progress), October 2016.
- [RFC1918] Rekhter, Y., Moskowitz, B., Karrenberg, D., de Groot, G., and E. Lear, "Address Allocation for Private Internets", BCP 5, RFC 1918, DOI 10.17487/RFC1918, February 1996, <<https://www.rfc-editor.org/info/rfc1918>>.
- [RFC1919] Chatel, M., "Classical versus Transparent IP Proxies", RFC 1919, DOI 10.17487/RFC1919, March 1996, <<https://www.rfc-editor.org/info/rfc1919>>.

- [RFC1928] Leech, M., Ganis, M., Lee, Y., Kuris, R., Koblas, D., and L. Jones, "SOCKS Protocol Version 5", RFC 1928, DOI 10.17487/RFC1928, March 1996, <<https://www.rfc-editor.org/info/rfc1928>>.
- [RFC4941] Narten, T., Draves, R., and S. Krishnan, "Privacy Extensions for Stateless Address Autoconfiguration in IPv6", RFC 4941, DOI 10.17487/RFC4941, September 2007, <<https://www.rfc-editor.org/info/rfc4941>>.
- [RFC6146] Bagnulo, M., Matthews, P., and I. van Beijnum, "Stateful NAT64: Network Address and Protocol Translation from IPv6 Clients to IPv4 Servers", RFC 6146, DOI 10.17487/RFC6146, April 2011, <<https://www.rfc-editor.org/info/rfc6146>>.
- [RFC7016] Thornburgh, M., "Adobe's Secure Real-Time Media Flow Protocol", RFC 7016, DOI 10.17487/RFC7016, November 2013, <<https://www.rfc-editor.org/info/rfc7016>>.
- [RFC7230] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing", RFC 7230, DOI 10.17487/RFC7230, June 2014, <<https://www.rfc-editor.org/info/rfc7230>>.
- [RFC7478] Holmberg, C., Hakansson, S., and G. Eriksson, "Web Real-Time Communication Use Cases and Requirements", RFC 7478, DOI 10.17487/RFC7478, March 2015, <<https://www.rfc-editor.org/info/rfc7478>>.

Appendix A. Change log

Changes in draft -12:

- o Editorial updates from IETF LC review.

Changes in draft -11:

- o Editorial updates from AD review.

Changes in draft -10:

- o Incorporate feedback from IETF 102 on the problem space.
- o Note that future versions of the document may define new modes.

Changes in draft -09:

- o Fixed confusing text regarding enterprise TURN servers.

Changes in draft -08:

- o Discuss how enterprise TURN servers should be handled.

Changes in draft -07:

- o Clarify consent guidance.

Changes in draft -06:

- o Clarify recommendations.
- o Split implementation guidance into two sections.

Changes in draft -05:

- o Separated framework definition from implementation techniques.
- o Removed RETURN references.
- o Use origin when determining local IPs, rather than a well-known IP.

Changes in draft -04:

- o Rewording and cleanup in abstract, intro, and problem statement.
- o Added 2119 boilerplate.
- o Fixed weird reference spacing.
- o Expanded acronyms on first use.
- o Removed 8.8.8.8 mention.
- o Removed mention of future browser considerations.

Changes in draft -03:

- o Clarified when to use which modes.
- o Added 2119 qualifiers to make normative statements.
- o Defined 'proxy'.
- o Mentioned split tunnels in problem statement.

Changes in draft -02:

- o Recommendations -> Requirements
- o Updated text regarding consent.

Changes in draft -01:

- o Incorporated feedback from Adam Roach; changes to discussion of cam/mic permission, as well as use of proxies, and various editorial changes.
- o Added several more references.

Changes in draft -00:

- o Published as WG draft.

Author's Address

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

Email: justin@uberti.name

RTCWEB WG
Internet-Draft
Intended status: Informational
Expires: May 4, 2017

E. Rescorla
RTFM, Inc.
October 31, 2016

Piggybacked DTLS Handshakes in SDP
draft-rescorla-dtls-in-sdp-01

Abstract

This document describes a mechanism for embedding DTLS handshake messages in SDP descriptions. This technique allows implementations to shave a full round-trip off of DTLS-SRTP session establishment, while retaining compatibility with ordinary DTLS-SRTP endpoints.

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 May 4, 2017.

Copyright Notice

Copyright (c) 2016 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction	2
2. Protocol Overview	4
2.1. DTLS 1.2	4
2.2. DTLS 1.3	5
3. Attribute Definition	7
4. Interactions	7
4.1. ICE	7
4.2. Forking	8
4.3. RTCWEB Identity	8
5. Examples	8
6. Security Considerations	8
7. IANA Considerations	9
8. References	9
8.1. Normative References	9
8.2. Informative References	10
8.3. URIs	11
Appendix A. Speculative: Server False-Start	11
Appendix B. Acknowledgements	13
Author's Address	13

1. Introduction

DTLS-SRTP [RFC5763][RFC5763] uses a DTLS [RFC6347] handshake to establish keys which are then used to key SRTP [RFC3711]. The DTLS negotiation is tied to the offer/answer [RFC3264] transaction via an "a=fingerprint" attribute [RFC4572] in the SDP [RFC4566]. The common message flow is shown below for DTLS 1.2.

This figure and the rest of this document adopt the following assumptions about network behavior:

- o ICE [RFC5245] is in use but that both endpoints implement endpoint-independent filtering [RFC5389] so that STUN checks succeed immediately.
- o Signaling messages take the same time to be delivered as direct messages [this is generally false.]

Links to detailed diagrams with a more accurate vertical scale can be found below each diagram.

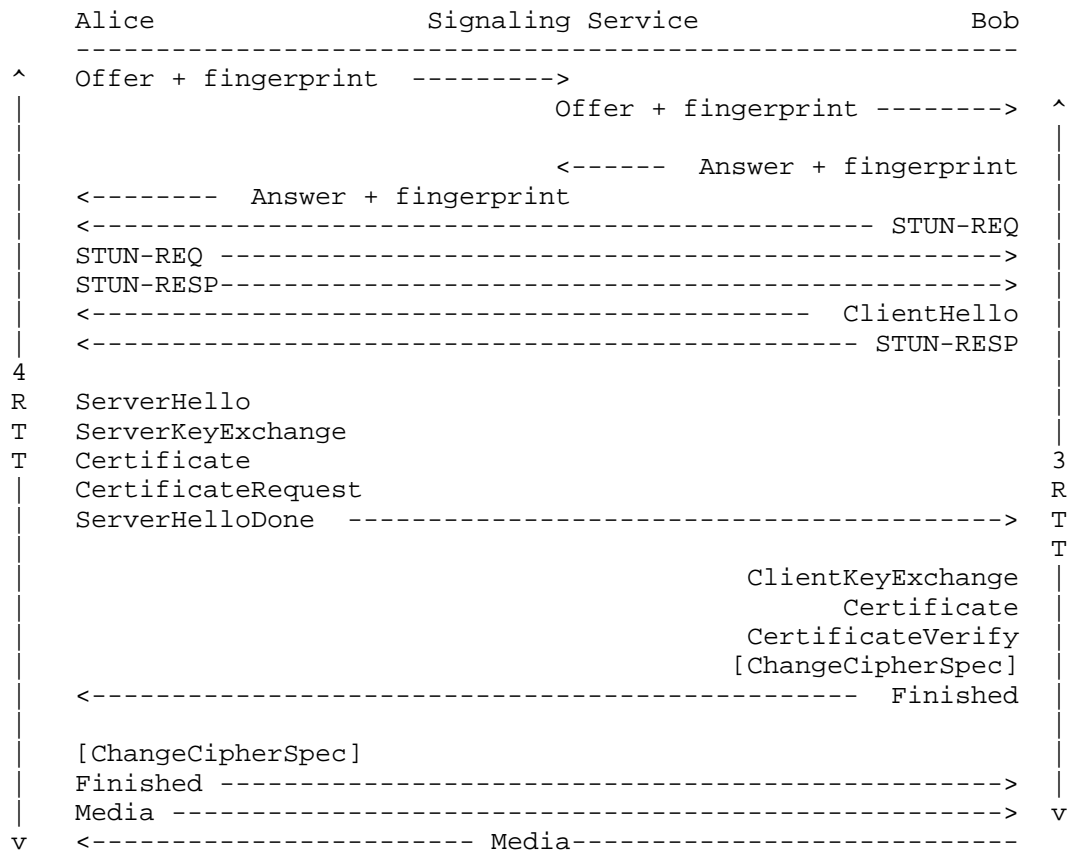


Figure 1: Standard DTLS-SRTP Negotiation

Better picture [1]

In this flow, the earliest that Alice can start sending media is after receiving Bob's Finished and the earliest Bob can start sending media is upon receiving Alice's Finished, and neither side can send any DTLS messages until they have had a successful STUN check. The result is that in the best case, Alice receives media four round trips after sending the offer and Bob receives media three round trips after receiving Alice's offer.

This document describes a technique for improving call setup time by piggybacking the first round of DTLS messages on the signaling messages. This reduces latency by a full round trip for both DTLS 1.2 and DTLS 1.3 handshakes, and for DTLS 1.3 [I-D.ietf-tls-tls13] allows the answerer to start sending media immediately upon receiving the offer, or, if ICE is used, upon ICE completion.

2. Protocol Overview

The basic concept, as shown in Figure 2, is for Alice to send her ClientHello in her Offer and Bob to send the server's first flight (ServerHello...ServerHelloDone for DTLS 1.2) in his Answer.

2.1. DTLS 1.2

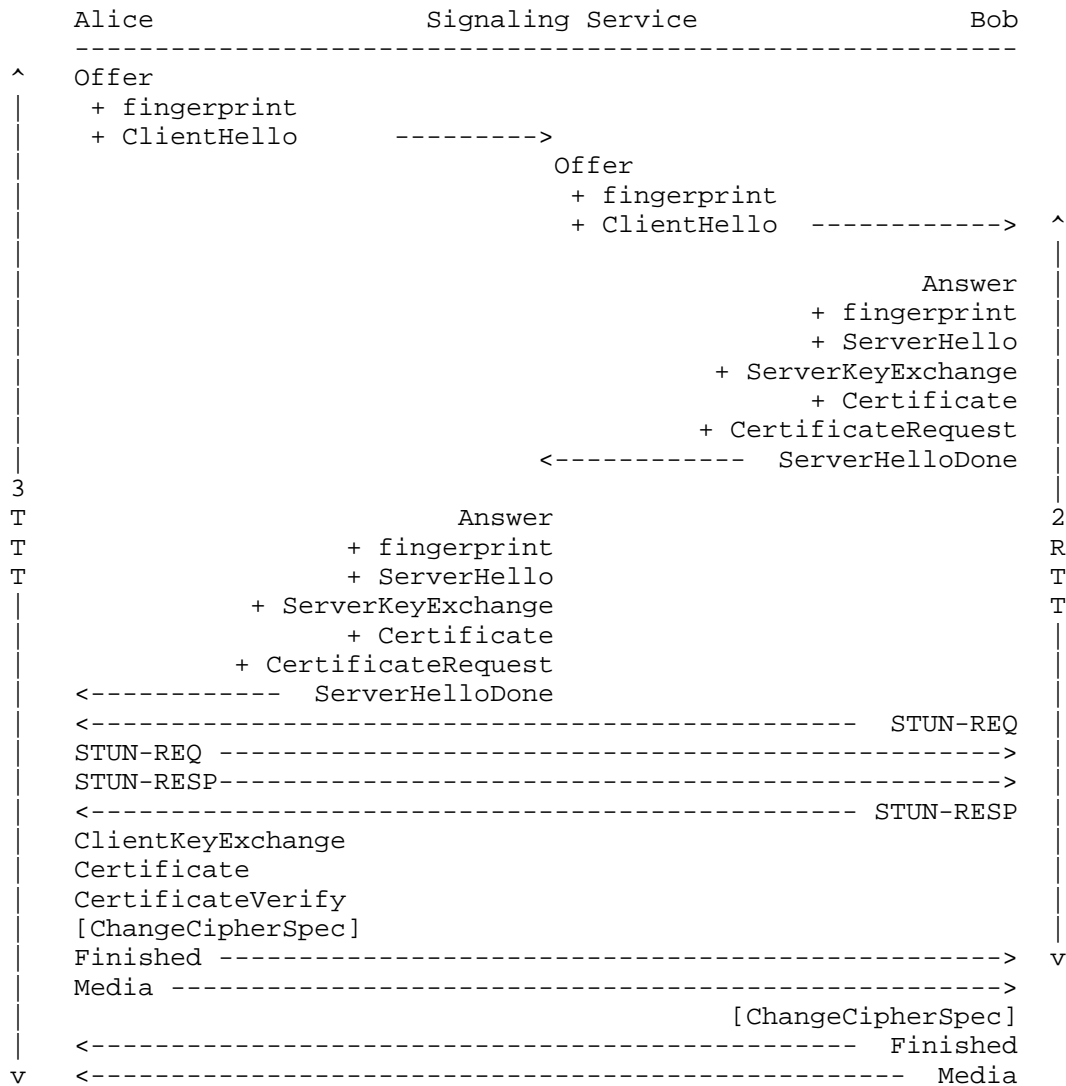


Figure 2: Piggybacked DTLS-SRTP Negotiation (TLS 1.2)

Better picture [2]

Note that in this flow, the active/passive (DTLS client/server) roles are reversed and Alice becomes the client. Because this is a basically symmetrical transaction, this is not an issue.

It should be immediately apparent that this exchange shaves off a full round trip from Bob's perspective (despite actually only shaving a half a round trip from the number of messages). The reason is that Bob does not need to wait for Alice's Finished to send but can piggyback his data on his Finished.

This change also shaves off a round trip from Alice's perspective because Alice can now safely perform TLS False Start [I-D.ietf-tls-falsestart] and send traffic prior to receiving Bob's Finished message. When only fingerprints are carried in the handshake, then extensions such as [RFC7301] indicators and DTLS-SRTP negotiation are not protected. However, in this case because those indicators are carried in the hello messages which are now tied to the signaling channel, they are authenticated via the same mechanisms that authenticate the fingerprint.

Note: One could argue that under some conditions Bob could do False Start in the ordinary handshake, but it's much harder to analyze and even then it leaves Alice one round trip slower than she would be with this optimization.

2.2. DTLS 1.3

Figure Figure 3 shows the impact of this optimization on DTLS 1.3.

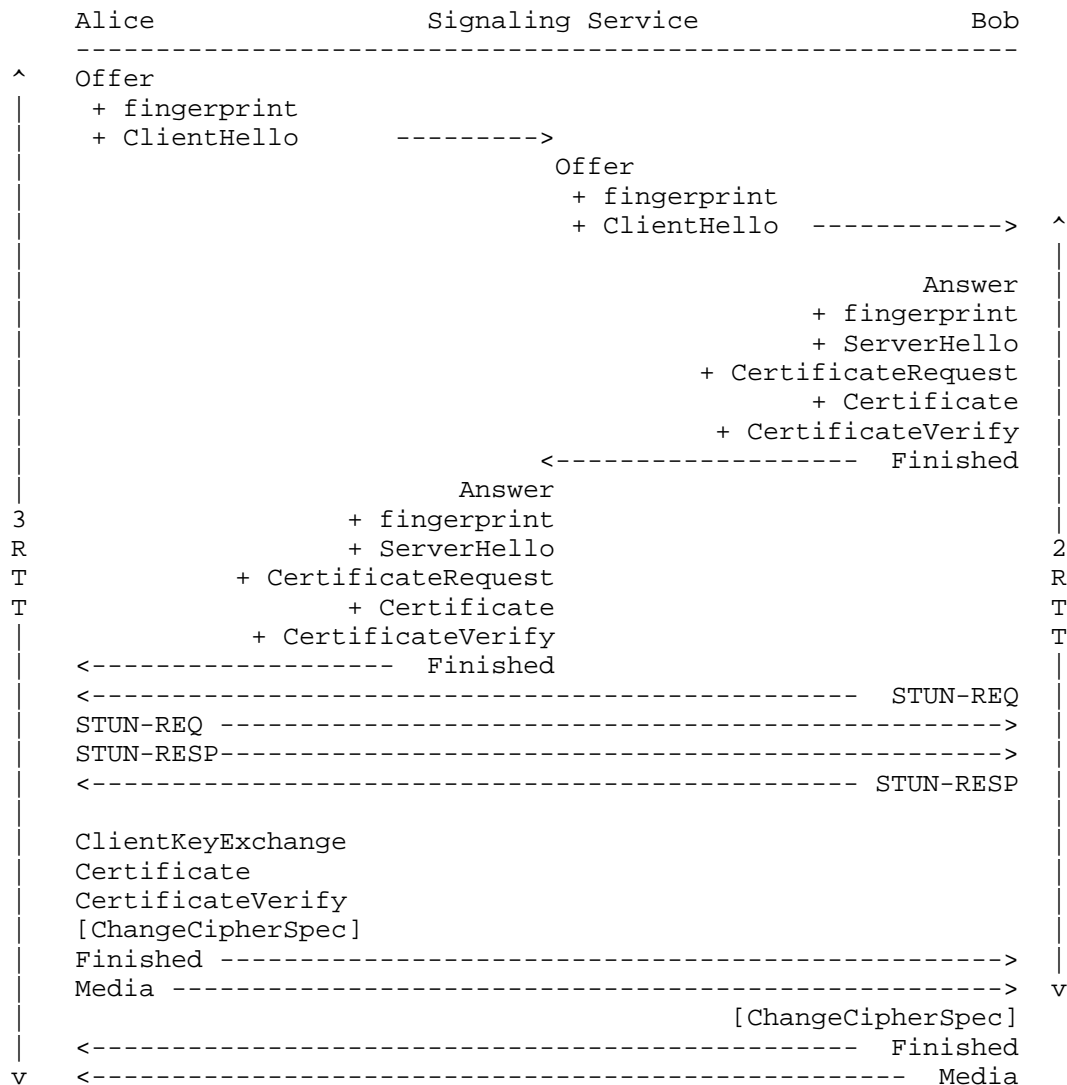


Figure 3: Piggybacked DTLS-SRTP Negotiation (TLS 1.3)

Better picture [3]

Alice cannot send any sooner than with DTLS 1.2 because sending at the point when she receives Bob's first message is already optimal. It may be possible for Bob to shave off yet another round trip, however. As described in Appendix A.

3. Attribute Definition

This document defines a new media-level SDP attribute, "a=dtls-message". This message is used to contain DTLS messages. The syntax of this attribute is:

```
attribute          =/  dtls-message-attribute

dtls-message-attribute =  "dtls-message" ":" role SP value

role               =    "client" / "server"

value              =    1*(ALPHA / DIGIT / "+" / "/" / "=" )
                      ; base64 encoded message
```

An offeror which wishes to use the optimization defined in this document shall send his ClientHello in the "a=dtls-message" attribute of its initial offer with the role "client" and MUST use "a=setup:actpass". This allows the peer to either:

- o Reject the optimization, in which case it ignores the attribute.
- o Accept the optimization, in which case it MUST use "a=setup:passive" and send its first flight (starting with ServerHello) and using the role "server" in its response. These messages are simply serialized end-to-end as they would be on the wire. It MAY also choose to send its first flight separately in the media channel; DTLS implementations already handle retransmits properly.

The offerer MUST be able to detect whether an incoming DTLS message is a ClientHello or a ServerHello and adapt accordingly.

In subsequent negotiations, implementations MUST maintain these roles.

4. Interactions

This optimization has a number of interactions with existing pieces of protocol machinery.

4.1. ICE

When ICE is in use, there is a race condition between the answerer's ICE checks (at which point it will be able to send the first flight on the media channel) and the answerer's Answer, which contains the first flight. For this reason, we allow implementations to send the first flight on both channels. However, as a practical matter it is

reasonably likely that when ICE is in use the Answer will arrive first, for two reasons:

- o The answerer consumes a full RTT doing a STUN check to verify the path to the offerer (even in the best case where the first STUN check succeeds). Thus, even if the path through the signaling server is twice as expensive as the direct path, there is a reasonable chance that the answer will arrive first.
- o If the offerer is behind a NAT without endpoint-independent filtering, the answerer's ICE checks will be discarded until the offerer sends its own ICE checks, which it can only do upon receiving the answer.

In this case, although a comparison of Figure 1 and Figure 2 would show the ClientHello (in ordinary DTLS) and the ServerHello (when piggybacked) as arriving at the same time, in fact the ServerHello may arrive up to a full RTT first, but the offerer can SEND its second flight immediately upon its STUN check succeeding, which happens first, thus increasing the advantage of this technique.

4.2. Forking

This technique does not interact very well with forking. Because each ClientHello is only usable for one server, the system must somehow ensure that only one of the forks takes up the piggybacked offers. The easiest approach is for any intermediary which does a fork to strip out the "a=dtls-message" attribute. An alternative would be to add another attribute which could be stripped out (this might interact better with RTCWEB Identity). Note that [RFC4474] protects against any SDP modifications, but I think at this point it's clear that that's not practical.

4.3. RTCWEB Identity

RTCWEB Identity assertions need to cover these DTLS messages.

5. Examples

[we need examples.]

6. Security Considerations

The security implications of this technique are described throughout this document.

7. IANA Considerations

This specification defines the "dtls-message" SDP attribute per the procedures of Section 8.2.4 of [RFC4566]. The required information for the registration is included here:

Contact Name: Eric Rescorla (ekr@rftm.com)

Attribute Name: dtls-message

Long Form: dtls-message

Type of Attribute: session-level

Charset Considerations: This attribute is not subject to the charset attribute.

Purpose: This attribute carries piggybacked DTLS message.

Appropriate Values: This document

8. References

8.1. Normative References

[I-D.ietf-tls-falsestart]

Langley, A., Modadugu, N., and B. Moeller, "Transport Layer Security (TLS) False Start", draft-ietf-tls-falsestart-02 (work in progress), May 2016.

[I-D.ietf-tls-tls13]

Rescorla, E., "The Transport Layer Security (TLS) Protocol Version 1.3", draft-ietf-tls-tls13-14 (work in progress), July 2016.

[RFC3264] Rosenberg, J. and H. Schulzrinne, "An Offer/Answer Model with Session Description Protocol (SDP)", RFC 3264, DOI 10.17487/RFC3264, June 2002, <<http://www.rfc-editor.org/info/rfc3264>>.

[RFC3711] Baugher, M., McGrew, D., Naslund, M., Carrara, E., and K. Norrman, "The Secure Real-time Transport Protocol (SRTP)", RFC 3711, DOI 10.17487/RFC3711, March 2004, <<http://www.rfc-editor.org/info/rfc3711>>.

[RFC4566] Handley, M., Jacobson, V., and C. Perkins, "SDP: Session Description Protocol", RFC 4566, DOI 10.17487/RFC4566, July 2006, <<http://www.rfc-editor.org/info/rfc4566>>.

- [RFC4572] Lennox, J., "Connection-Oriented Media Transport over the Transport Layer Security (TLS) Protocol in the Session Description Protocol (SDP)", RFC 4572, DOI 10.17487/RFC4572, July 2006, <<http://www.rfc-editor.org/info/rfc4572>>.
- [RFC5245] Rosenberg, J., "Interactive Connectivity Establishment (ICE): A Protocol for Network Address Translator (NAT) Traversal for Offer/Answer Protocols", RFC 5245, DOI 10.17487/RFC5245, April 2010, <<http://www.rfc-editor.org/info/rfc5245>>.
- [RFC5389] Rosenberg, J., Mahy, R., Matthews, P., and D. Wing, "Session Traversal Utilities for NAT (STUN)", RFC 5389, DOI 10.17487/RFC5389, October 2008, <<http://www.rfc-editor.org/info/rfc5389>>.
- [RFC5763] Fischl, J., Tschofenig, H., and E. Rescorla, "Framework for Establishing a Secure Real-time Transport Protocol (SRTP) Security Context Using Datagram Transport Layer Security (DTLS)", RFC 5763, DOI 10.17487/RFC5763, May 2010, <<http://www.rfc-editor.org/info/rfc5763>>.
- [RFC6347] Rescorla, E. and N. Modadugu, "Datagram Transport Layer Security Version 1.2", RFC 6347, DOI 10.17487/RFC6347, January 2012, <<http://www.rfc-editor.org/info/rfc6347>>.
- [RFC7301] Friedl, S., Popov, A., Langley, A., and E. Stephan, "Transport Layer Security (TLS) Application-Layer Protocol Negotiation Extension", RFC 7301, DOI 10.17487/RFC7301, July 2014, <<http://www.rfc-editor.org/info/rfc7301>>.

8.2. Informative References

- [I-D.thomson-avtcore-sdp-uks]
Thomson, M. and E. Rescorla, "Unknown Key Share Attacks on uses of Transport Layer Security with the Session Description Protocol (SDP)", draft-thomson-avtcore-sdp-uks-00 (work in progress), October 2016.
- [RFC4474] Peterson, J. and C. Jennings, "Enhancements for Authenticated Identity Management in the Session Initiation Protocol (SIP)", RFC 4474, DOI 10.17487/RFC4474, August 2006, <<http://www.rfc-editor.org/info/rfc4474>>.

8.3. URIs

- [1] <https://raw.githubusercontent.com/ekr/dtls-in-sdp/master/normal-12.png>
- [2] <https://raw.githubusercontent.com/ekr/dtls-in-sdp/master/piggybacked-12.png>
- [3] <https://raw.githubusercontent.com/ekr/dtls-in-sdp/master/piggybacked-13.png>
- [4] <https://raw.githubusercontent.com/ekr/dtls-in-sdp/master/piggybacked-13-falsestart.png>

Appendix A. Speculative: Server False-Start

WARNING: THE FOLLOWING SECTION HAS NOT RECEIVED ANY REAL SECURITY REVIEW AND MAY BE A REALLY BAD IDEA.

It has been observed that as if Alice uses a fresh DH ephemeral, then Bob knows (because he can trust the signaling service) that Alice's DH ephemeral corresponds to Alice and can therefore encrypt under the joint DH shared secret without waiting for Alice's CertificateVerify, as shown in Figure 4.

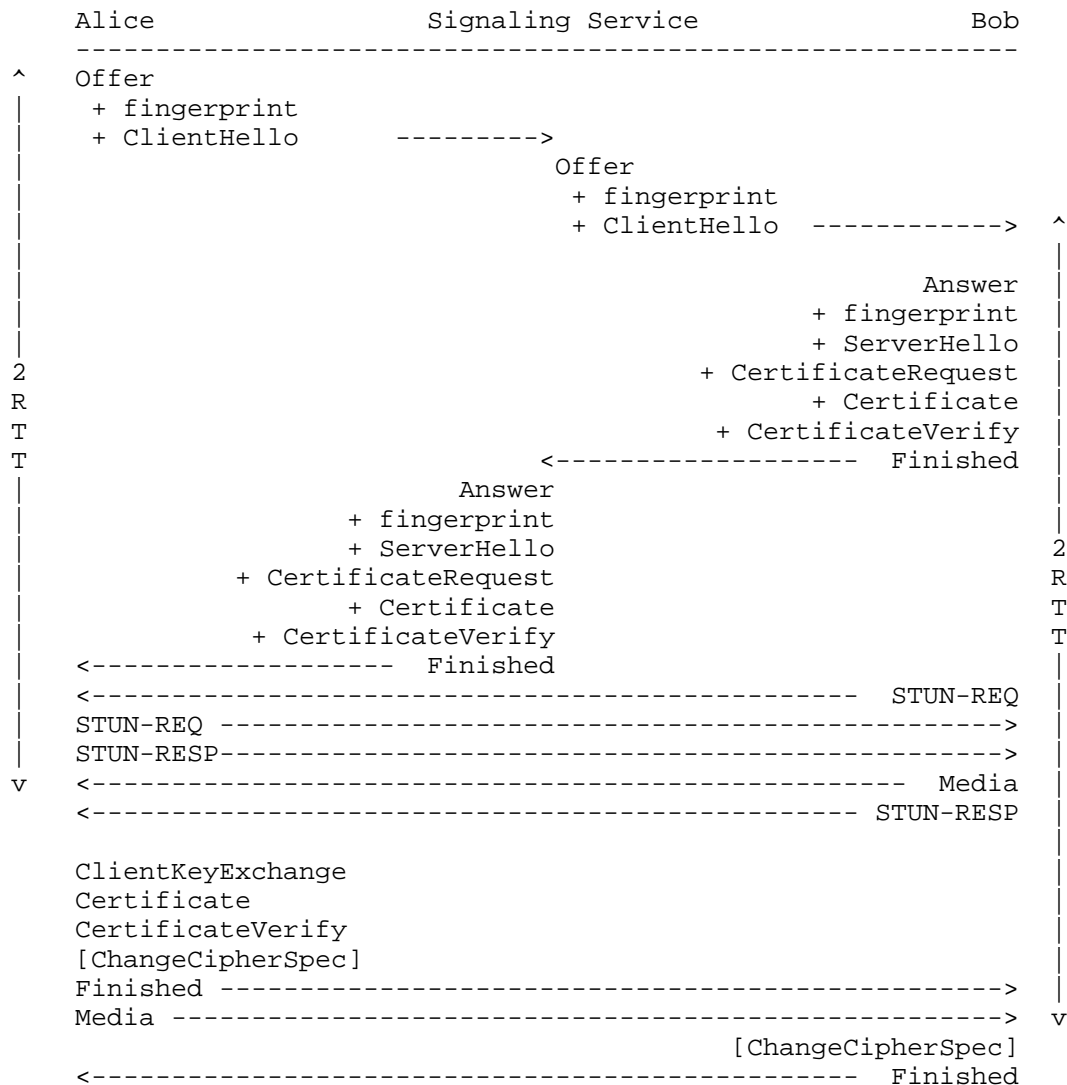


Figure 4: Piggybacked DTLS-SRTP Negotiation (TLS 1.3 with false start)

Better picture [4]

This has demonstrably inferior security properties if Alice is using a long-term key (for key continuity or fingerprint validation), because Bob has not yet verified that Alice controls that key and does not even know if Alice is using a fresh DH ephemeral, if implementations decide to adopt this optimization, they must do

something hacky like Send data immediately but generate an error if the handshake, including a signature, does not complete within some reasonable period (a small number of measured round trips) [Just one reason why this is a questionable technique.]. This technique may also complicate dealing with the issues raised in [I-D.thomson-avtcore-sdp-uks].

Appendix B. Acknowledgements

Thanks to Cullen Jennings, Martin Thomson, and Justin Uberti for helpful suggestions.

Author's Address

Eric Rescorla
RTFM, Inc.

Email: ekr@rtfm.com