

Network Working Group  
Internet-Draft  
Intended status: Informational  
Expires: August 29, 2013

H. Alvestrand  
A. Grange  
Google  
February 25, 2013

VP8 as RTCWEB Mandatory to Implement  
draft-alvestrand-rtcweb-vp8-01

Abstract

This document recommends that the RTCWEB working group choose the VP8 specification as a mandatory to implement video codec for RTCWEB implementations.

This document is not intended for publication as an RFC.

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 August 29, 2013.

Copyright Notice

Copyright (c) 2013 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 . . . . .	3
2. Requirements for an MTI codec . . . . .	3
3. Definition of VP8 . . . . .	3
4. Image quality evaluations . . . . .	3
5. Performance evaluation . . . . .	4
5.1. Software . . . . .	4
5.2. Hardware support . . . . .	4
5.3. Hardware performance . . . . .	5
6. IPR status . . . . .	5
7. IANA Considerations . . . . .	6
8. Security Considerations . . . . .	6
9. Acknowledgements . . . . .	6
10. References . . . . .	6
10.1. Normative References . . . . .	6
10.2. Informative References . . . . .	7
Authors' Addresses . . . . .	7

## 1. Introduction

As described in [I-D.ietf-rtcweb-overview], successful interoperable deployment of RTCWEB requires that implementations share a video codec. Not requiring a video codec will mean that this decision is left to processes outside the standards process, and risks the spectre of fragmented deployment.

This memo argues that VP8 should be that codec.

## 2. Requirements for an MTI codec

As outlined by the presentation given at the IETF meeting at IETF 84 in Vancouver, it is unclear what the hard requirements for a video codec are, but the items that it was suggested that proposals give information on were:

- o Image quality - comparative data was sought, but without defining a baseline
- o Performance - what resolutions / frame rates can be achieved in software on some common systems
- o Power consumption of hardware and/or software implementations
- o Hardware support
- o IPR status

This document lays out the available information in each category.

## 3. Definition of VP8

VP8 is defined in [RFC6386], and its RTP payload is defined in [I-D.ietf-payload-vp8]. There are no profiles; all decoders are able to decode all valid media streams.

## 4. Image quality evaluations

In tests carried out by Google on a set of ten sample video clips containing typical video-conference content, VP8 outperformed the x264 H.264 codec running the constrained baseline profile by on average 37.2%. That is, at the same quality, measured by PSNR, VP8 produced 37.2% fewer bits on average than H.264. VP8 outperformed H.264 on all ten of the test clips by between 19% and 64%. Both

codecs were configured in one-pass mode using settings conducive to real-time operation, and the ten clips varied in size between 640x360 pixels and 1280x720 pixels.

An independent evaluation by Christian Feller and Mohammed Raad, presented to ISO/IEC SC29 WG11 in July 2012, showed that VP8 performed better than the (H.264 baseline) anchors for the IVC project on a majority of the cases.

As part of the process of submitting VP8 for evaluation in ISO/IEC JTC1 SC29 WG11 (MPEG), Google is also commissioning an independent subjective quality evaluation effort.

## 5. Performance evaluation

### 5.1. Software

The current reference implementation is libvpx, developed in the WebM project.

The encoding speed in software depends on the quality setting. On a stock PC platform using an Intel Xeon CPU at 2.67 GHz, in a test using extremely difficult 720p material and encoding at a target data rate of 2 Mbit/sec, VP8's encoding speed varied from 48.4 fps (at the setting used in WebRTC today) to 96.2 fps (at the fastest setting), using a single thread. This variation in encode speed is achieved by changing the configuration of VP8 encoding tools in a deterministic way to trade-off encoding speed with output quality.

On a stock PC platform using an Intel Xeon CPU with 8 cores at 2.27GHz, tests using difficult 720p material encoded at 2 Mbit/sec show that using a single thread VP8 can decode at 200.50 fps (in comparison H.264, baseline profile, achieves 107.95 fps), using four threads VP8 decodes at 519.96 fps (H.264 achieves 363.73 fps), and using sixteen threads VP8 decodes at 1,076.49 fps (H.264 achieves 807.11 fps). .

### 5.2. Hardware support

To date, Google has licensed VP8 hardware accelerators to over 50 chip manufacturers, and VP8 hardware IP cores have also been made available by Imagination Technologies, Verisilicon and Chips & Media. Furthermore, Google is aware of several 3rd party implementations of VP8 decoders and encoders from the world's leading semiconductor companies.

At the time of this writing, more than a dozen of chip manufacturers

have announced chips with 1080p VP8 support, including Samsung (Exynos 5), NVIDIA (Tegra 3), Marvell (Armada 1500), Broadcom (BCM28150), Texas Instruments (OMAP54xx), Freescale (i.MX 6), ST-Ericsson (NovaThor L9540), LG Electronics, Hisilicon (K3v2), Rockchip (RK2918, RK3066), Nufont (NS115), Ziilabs (ZMS40) and Allwinner (A10). Google estimates that a clear majority of leading mobile chipsets in 2013 will contain VP8 hardware support.

The encoder chip produced by Quanta has allowed OEMs to integrate hardware HD VP8 encoding into their video camera hardware; this chip is available now. More suppliers have such a chip coming.

### 5.3. Hardware performance

Several of the aforementioned hardware implementations are based on the WebM video hardware designs described at <http://www.webmproject.org/hardware/>. Performance figures include:

- o Decode of 1080p video at 30 fps at less than 100 MHz clock frequency
- o Decoding more than ten simultaneous SD video streams on a single chip
- o Less than 25 milliwatts of power for 1080p decoding
- o Encoding 1080p video at 30 fps at less than 220 MHz clock frequency
- o Less than 80 milliwatts of power for HD video encoding

Based on the Hantro G1 multiformat decoder implementation, the VP8 hardware decoder is 45% smaller in silicon area than the H.264 High Profile decoder. VP8 also requires 18% less DRAM bandwidth than H.264 as it does not use bidirectional inter prediction, allowing significant reductions in the overall decoding system power consumption.

### 6. IPR status

Google has made its position clear with respect to Google-owned IPR in its licensing terms, <http://www.webmproject.org/license/additional/>.

As of this moment (October 5, 2012), Google's royalty-free license commitment is the only IPR statement filed against RFC 6386 in the IETF disclosures database.

Google has also submitted VP8 for consideration in ISO/IEC JTC1 SC29 WG11 (MPEG), in the IVC project (which aims for a royalty-free codec), and expects ISO to execute its ordinary process for resolution of IPR issues.

## 7. IANA Considerations

This document makes no request of IANA.

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

## 8. Security Considerations

Codec definitions do not in themselves comprise security risks, as long as there is no means of embedding active content in their datastream. VP8 does not contain such active content.

Codec implementations have frequently been the cause of security concerns. The reference implementation of VP8 has been extensively tested by Google security experts, and is believed to be free from exploitable vulnerabilities. There is a continuous program in place to ensure that any vulnerabilities identified are repaired as quickly as possible.

## 9. Acknowledgements

Several members of the Google VP8 team contributed to this memo.

## 10. References

### 10.1. Normative References

[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-vp8-08 (work in progress), January 2013.

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

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

Guide", RFC 6386, November 2011.

## 10.2. Informative References

[I-D.ietf-rtcweb-overview]

Alvestrand, H., "Overview: Real Time Protocols for Brower-based Applications", draft-ietf-rtcweb-overview-05 (work in progress), December 2012.

## Authors' Addresses

Harald Alvestrand  
Google  
Kungsbron 2  
Stockholm, 11122  
Sweden

Email: [harald@alvestrand.no](mailto:harald@alvestrand.no)

Adrian Grange  
Google  
1950 Charleston Road  
Mountain View, CA 94043  
USA

Phone:  
Fax:  
Email: [agrange@google.com](mailto:agrange@google.com)  
URI:





RTCWEB Working Group  
Internet-Draft  
Intended status: Standards Track  
Expires: August 29, 2013

B. Burman  
Ericsson  
M. Isomaki  
Nokia  
B. Aboba  
Microsoft Corporation  
G. Martin-Cocher  
RIM  
G. Mandyam  
Qualcomm Innovation Center  
X. Marjou  
France Telecom Orange  
February 25, 2013

H.264 as Mandatory to Implement Video Codec for WebRTC  
draft-burman-rtcweb-h264-proposal-01

Abstract

This document proposes that, and motivates why, H.264 should be a Mandatory To Implement video codec for WebRTC.

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 August 29, 2013.

Copyright Notice

Copyright (c) 2013 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 . . . . .	3
2. Terminology . . . . .	3
3. H.264 Overview . . . . .	3
4. Implementations . . . . .	3
5. Licensing . . . . .	4
6. Performance . . . . .	5
7. Profile/level . . . . .	6
8. Negotiation . . . . .	8
9. Summary . . . . .	8
10. IANA Considerations . . . . .	9
11. Security Considerations . . . . .	9
12. Acknowledgements . . . . .	9
13. References . . . . .	9
13.1. Normative References . . . . .	9
13.2. Informative References . . . . .	10
Authors' Addresses . . . . .	11

## 1. Introduction

The selection of a Mandatory To Implement (MTI) video codec for WebRTC has been discussed for quite some time in the RTCWEB WG. This document proposes that the H.264 video codec should be mandatory to implement for WebRTC implementations and gives motivation to this proposal.

The core of the proposal is that H.264 Constrained Baseline Profile Level 1.2 MUST be supported as Mandatory To Implement video codec. To enable higher quality for devices capable of it, support for H.264 Constrained High Profile Level 1.3, extended to support 720p resolution at 30 Hz framerate is RECOMMENDED.

This draft discusses the advantages of H.264 as the authors of this draft see them; a richness of implementations and hardware support, well known licensing conditions, good performance, and well defined handling of varying device capabilities.

## 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 BCP 14, RFC 2119 [RFC2119].

## 3. H.264 Overview

The video coding standard Advanced Video Coding (ITU-T H.264 | ISO/IEC 14496-10 [H264]) has been around for almost ten years by now. Developed jointly by MPEG and ITU-T in the Joint Video Team, it was published in its first version in 2003 and amended with support for higher-fidelity video in 2004. Other significant updates include support for scalability (2007) and multiview (2009). The codec goes under the names H.264, AVC and MPEG-4 Part10. In this memo the term "H.264" will be used.

H.264 was from the start very successful and has become widely adopted for (video) content as well as (video) communication services worldwide.

## 4. Implementations

Arguably, hardware or DSP acceleration for video encoding/decoding would be mostly beneficial for devices that has relatively lower

capacity in terms of CPU and power (smaller batteries), and the most common devices in this category are phones and tablets. There is a long list of vendors offering hardware or DSP implementations of H.264. In particular all vendors of platforms for mobile high-range phones, smartphones, and tablets support H.264/AVC High Profile encoding and decoding at least 1080p30, but those platforms are currently in general not used for low- to mid-range devices. These vendors are ST-Ericsson, Qualcomm, TI, Nvidia, Renesas, Mediatek, Huawei Hisilicon, Intel, Broadcom, Samsung. Those platforms all support H.264/AVC codec with dedicated HW or DSP. For at least the ST-Ericsson and Qualcomm hardware it is verified that the implementation has low-delay real-time support, but it seems likely that this is the case for at least the majority of the others as well.

There are also other specifications that implement support for H.264, such as HDMI(TM).

Regarding software implementations there is a long list of available implementations. Wikipedia provides an illustration of this with their list [Implementations], and more implementations appear, e.g. [Woon]. Not only are there standalone implementations available, including open source, but in addition recent Windows and Mac OS X versions support H.264 encoding and decoding.

## 5. Licensing

H.264 is a mature codec with a mature and well-known licensing model. MPEG-LA released their AVC Patent Portfolio License already in 2004 and in 2010 they announced that H.264 encoded Internet video is free to end users will never be charged royalties [MPEGLA]. Real-time generated content, the content most applicable to WebRTC, was free already from the establishment of the MPEG-LA license. License fees for products that decode and encode H.264 video remain though. Those fees are, and will very likely continue to be for the lifetime of MPEG-LA pool, \$0.20 per codec or less. It can be noted that for MPEG-LA, since one license covers both an encoder and decoder, there is no additional cost of using an encoder to an implementation that supports decoding of H.264.

It is a well-established fact that not all H.264 right holders are MPEG-LA pool members. H.264 is however an ITU/ISO/IEC international standard, developed under their respective patent policies, and all contributors must license their patents under Reasonable And Non-Discriminatory (RAND) terms. In the field of video coding, most major research groups interested in patents do contribute to the ITU/ISO/IEC standards process and are therefore bound by those terms.

VP8 is a much younger codec than H.264 and it is fair to say that the licensing situation is less clear than for H.264. Google has provided their patent rights on VP8 under a open source friendly license with very restrictive reciprocity conditions. According to MPEG-LA's web page [MpegLaVp8], MPEG-LA is in the process of forming a royalty-bearing patent pool for VP8. Also, according to press reports [DoJ], at least the US Department of Justice investigate MPEG-LA for anticompetitive activity in conjunction with the VP8 pool formation. This indicates that the licensing situation for VP8 has not settled.

## 6. Performance

Comparing video quality is difficult. Practically no modern video encoding method includes any bit-exact encoding where a given (video) input produces a specified encoded output bitstream. Instead, the encoded bitstream syntax and semantics are specified such that a decoder can correctly interpret it and produce a known output. This is true both for H.264 and VP8. Significant freedom is left to the encoder implementation to choose how to represent the encoded video, for example given a specific targeted bitrate. Thus it cannot in general be expected that any encoded video bitstream represents the best possible or most efficient representation, given the defined bitstream syntax elements available to that codec. The actually achieved quality for a certain bitstream, how close it is to the optimally possible with available syntax, at any given bitrate rather depends on the performance of the individual encoder implementation.

Also, not only is the resulting experienced video quality subjective, but also depends on the source material, on the point of operation and a number of other considerations. In addition, performance can be measured vs. bitrate, but also vs. e.g. complexity - and here another can of worms can be opened because complexity depends on hardware used (some platforms have video codec accelerations), SW platform (and how efficient it can use the hardware) and so on. On top of this comes that different implementations can have different performance, and can be operated in different ways (e.g. tradeoffs between complexity and quality can be made). Regardless of how a performance evaluation is carried out it can always be said that it is not "fair". This section nevertheless attempts to shed some light on this subject, and specifically the performance (measured against bitrate) of H.264 compared to VP8.

A number of studies [H264perf1][H264perf2][H264perf3] have been made to compare the compression efficiency performance between H.264 and VP8. These studies show that H.264 is in general performing better than VP8 but the studies are not specifically targeting video

conferencing. Therefore, Ericsson made a comparison where a number of video conferencing type sequences were encoded using both H.264 and VP8. Eight video conferencing type test sequences were used; three were taken from the MPEG/ITU test set (vidyo2-4) and five were recorded by Ericsson. The sequences were all 720p 25/30Hz.

The focus of that test was to evaluate the best compression efficiency that could be achieved with both codecs since it was believed to be harder to make a fair comparison trying to use complexity constraints. The results showed that H.264 High Profile provides an average bitrate compared to VP8 of -23% (minus here means that H.264 is better) using PSNR-based Bjontegaard Delta bitrate (BD-rate) [PSNRdiff]. H.264 Constrained High Profile provided -16% and Constrained Baseline Profile resulted in +16% (plus here means that VP8 is better).

For H.264, JM 18.3 in low-delay mode without reordering of B or P pictures was used. For VP8 encoding, v1.1.0 with the "best" preset was used.

Again, video quality is difficult to compare. The authors however believe that the data provided in this section shows that H.264 is at least on par with VP8. As a final note, the new H.265/HEVC standard clearly outperforms both of them, but the authors think it is premature to mandate HEVC for WebRTC.

## 7. Profile/level

H.264/AVC [H264] has a large number of encoding tools, grouped in functionally reasonable toolsets by codec profiles, and a wide range of possible implementation capability and complexity, specified by codec levels. It is typically not reasonable for H.264 encoders and decoders to implement maximum complexity capability for all of the available tools. Thus, any H.264 decoder implementation is typically not able to receive all possible H.264 streams. Which streams can be received is described by what profile and level the decoder conforms to. Any video stream produced by an H.264 encoder must keep within the limits defined by the intended receiving decoder's profile and level to ensure that the video stream can be correctly decoded.

Profiles can be "ranked" in terms of the amount of tools included, such that some profiles with few tools are "lower" than profiles with more tools. However, profiles are typically not strictly supersets or subsets of each other in terms of which tools are used, so a strict ranking cannot be defined. It is also in some cases possible to express compliance to the common subset of tools between two different profiles. This is fairly well described in [RFC6184].

When choosing a Mandatory To Implement codec, it is desirable to use a profile and level that is as widely supported as possible. Therefore, H.264 Constrained Baseline Profile Level 1.2 MUST be supported as Mandatory To Implement video codec. This is possible to support with significant margin in hardware devices (Section 4) and should likely also not cause performance problems for software-only implementations. All Level definitions (Annex A of [H264]) include a maximum framesize in macroblocks (16\*16 pixels) as well as a maximum processing requirement in macroblocks per second. That number of macroblocks per second can be almost freely distributed between framesize and framerate. The maximum framesize for Level 1.2 corresponds to 352\*288 pixels (CIF). Examples of allowed framesize and framerate combinations for Level 1.2 are CIF (352\*288 pixels) at 15 Hz, QVGA (320\*240 pixels) at 20 Hz, and QCIF (176\*144 pixels) at 60 Hz.

Recognizing that while the above profile and level will likely be possible to implement in any device, it is also likely not sufficient for applications that require higher quality. Therefore, it is RECOMMENDED that devices and implementations that can meet the additional requirements also implement at least H.264 Constrained High Profile Level 1.3, extended to support 720p resolution at 30 Hz framerate, but the extension MAY alternatively be made from any Level higher than 1.3.

Note that the lowest non-extended Level that support 720p30 is Level 3.1, but fully supporting Level 3.1 also requires fairly high bitrate, large buffers, and other encoding parameters included in that Level definition that are likely not reasonable for the targeted communication scenario. This method of extending a lower level in SDP (Section 8) with a smaller set of applicable parameters is fully in line with [RFC6184], and is already used by some video conferencing vendors.

When considering the main WebRTC use case, real-time communication, the lack of need to support interlaced image format in that context, the limited use of and added delay from bi-directionally predicted (B) pictures, and the added implementation and computation complexity that comes with interlace and B-picture handling suggests that Constrained High Profile should be preferred over High Profile as optional codec. Note also that while Constrained High Profile is currently less supported in devices than High Profile, any High Profile decoder will be capable of decoding a Constrained High Profile bitstream since it is a subset of High Profile. To make a High Profile encoder support Constrained High Profile encoding, it will have to turn off interlace encoding and turn off the use of bi-directional prediction.

## 8. Negotiation

Given that there exist a fairly large set of defined profiles and levels (Section 7), the probability is rather low that randomly chosen H.264 encoder and decoder implementations have exactly matching capabilities. In any communication scenario, there is therefore a need for a decoder to be able to convey its maximum supported profile and level that the encoder must not exceed.

In addition and depending on the wanted use case and the conditions that apply at a certain communication instance, there may also be a need to describe the currently wanted profile and level at the start of the communication session, which may be lower than the maximum supported by the implementation. In this scenario it may also be of interest to communicate from the encoder to the decoder both which profile and level that will actually be used and what is the maximum supported profile and level. The reason to communicate not only the starting point but also the maximum assumes that communication conditions may change during the conditions, maybe multiple times, possibly making another profile and level be a more appropriate choice.

Communication of maximum supported profile and level is the only mandatory SDP [RFC4566] parameter in the H.264 payload format [RFC6184], which also includes a large set of optional parameters, describing available use (decoder) and intended use (encoder) of those parameters for a specific offered [RFC3264] stream.

If the above mentioned (Section 7) capability for 720p30 is supported as an extension to Constrained High Profile Level 1.3 (or higher), the level extension SHOULD be signaled in SDP using the following parameters as defined in section 8.1 of [RFC6184]:

- o profile-level-id=640c0d (or corresponding to a higher Level of Constrained High profile)
- o max-fs=3600 (or greater)
- o max-mbps=108000 (or greater)
- o max-br=768 (or greater, whatever the device implementation can support)

## 9. Summary

H.264 is widely adopted and used for a large set of video services. This in turn is because H.264 offers great performance, reasonable



licensing terms (and manageable risks). As a consequence of its adoption for many services, a multitude implementations in software and hardware are available. Another result of the widespread adoption is that all associated technologies, such as payload formats, negotiation mechanisms and so on are well defined and standardized. In addition, using H.264 enables interoperability with many other services without video transcoding.

We therefore propose to the WG that H.264 shall be mandatory to implement for all WebRTC endpoints that support video, according to the details described in Section 7 and Section 8.

#### 10. IANA Considerations

This document makes no request of IANA.

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

#### 11. Security Considerations

No specific considerations apply to the information in this document.

#### 12. Acknowledgements

All that provided valuable descriptions, comments and insights about the H.264 codec on the IETF mailing lists.

#### 13. References

##### 13.1. Normative References

- [H264] ITU-T Recommendation H.264, "Advanced video coding for generic audiovisual services", March 2010.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.
- [RFC3264] Rosenberg, J. and H. Schulzrinne, "An Offer/Answer Model with Session Description Protocol (SDP)", RFC 3264, June 2002.
- [RFC4566] Handley, M., Jacobson, V., and C. Perkins, "SDP: Session Description Protocol", RFC 4566, July 2006.

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

### 13.2. Informative References

- [DoJ] "Web Video Rivalry Sparks U.S. Probe", <<http://online.wsj.com/article/SB10001424052748703752404576178833590548792.html>>.
- [H264perf1] Vatolin, D., "MPEG-4 AVC/H.264 Video Codecs Comparison 2010 - Appendixes", , May 2010, <[http://compression.graphicon.ru/video/codec\\_comparison/h264\\_2010/appendixes.html#Appendix\\_8](http://compression.graphicon.ru/video/codec_comparison/h264_2010/appendixes.html#Appendix_8)>.
- [H264perf2] Shah, K., "Implementation, performance analysis and comparison of VP8 and H.264.", University of Texas at Arlington Department of Electrical Engineering, 2011, <[http://www-ee.uta.edu/Dip/Courses/EE5359/2011SpringFinalReportPPT/Shah\\_EE5359Spring2011FinalPPT.pdf](http://www-ee.uta.edu/Dip/Courses/EE5359/2011SpringFinalReportPPT/Shah_EE5359Spring2011FinalPPT.pdf)>.
- [H264perf3] De Simone, F., Goldmann, L., Lee, J., and T. Ebrahimi, "Performance analysis of VP8 image and video compression based on subjective evaluations", Ecole Polytechnique F'd'rale de Lausanne (EPFL) , Aug 2011, <<http://infoscience.epfl.ch/record/168259/files/article.pdf>>.
- [Implementations] Wikipedia, "H.264/MPEG-4 AVC products and implementations", October 2012, <[http://en.wikipedia.org/wiki/H.264/MPEG-4\\_AVC\\_products\\_and\\_implementations](http://en.wikipedia.org/wiki/H.264/MPEG-4_AVC_products_and_implementations)>.
- [MPEGLA] "MPEG LA's AVC License Will Not Charge Royalties for Internet Video that is Free to End Users through Life of License", MPEGLA News Release, August 2010, <[www.mpeg-la.com/Lists/MPEG%20LA%20News%20List/Attachments/231/n-10-08-26.pdf](http://www.mpeg-la.com/Lists/MPEG%20LA%20News%20List/Attachments/231/n-10-08-26.pdf)>.
- [MpegLaVp8] "MPEG LA Announces Call for Patents Essential to VP8 Video Codec", <<http://www.mpeg-la.com/main/pid/vp8/default.aspx>>.
- [PSNRdiff] Bjontegaard, G., "Calculation of Average PSNR Differences

between RD-Curves", ITU-T SG16 Q.6 Document VCEG-M33,  
April 2001.

[Woon] "Polycom Delivers Open Standards-Based Scalable Video  
Coding (SVC) Technology, Royalty-Free to Industry",  
October 2012, <[http://www.polycom.com/content/www/en/  
company/news/press-releases/2012/20121004.html](http://www.polycom.com/content/www/en/company/news/press-releases/2012/20121004.html)>.

#### Authors' Addresses

Bo Burman  
Ericsson  
Farogatan 6  
Stockholm 16480  
Sweden

Email: [bo.burman@ericsson.com](mailto:bo.burman@ericsson.com)

Markus Isomaki  
Nokia  
Keilalahdentie 2-4  
Espoo FI-02150  
Finland

Email: [markus.isomaki@nokia.com](mailto:markus.isomaki@nokia.com)

Bernard Aboba  
Microsoft Corporation  
One Microsoft Way  
Redmond, WA 98052  
US

Email: [bernard\\_aboba@hotmail.com](mailto:bernard_aboba@hotmail.com)

Gaelle Martin-Cocher  
RIM  
1875 Buckhorn Gate  
Mississauga, ON L4W 5P1  
Canada

Email: [gmartincocher@rim.com](mailto:gmartincocher@rim.com)

Giri Mandyam  
Qualcomm Innovation Center

Email: mandyam@quicinc.com

Xavier Marjou  
France Telecom Orange  
2, avenue Pierre Marzin  
Lannion, 22307  
France

Email: xavier.marjou@orange.com



Network Working Group  
Internet-Draft  
Intended status: Standards Track  
Expires: May 16, 2018

H. Alvestrand  
Google  
November 12, 2017

Overview: Real Time Protocols for Browser-based Applications  
draft-ietf-rtcweb-overview-19

Abstract

This document gives an overview and context of a protocol suite intended for use with real-time applications that can be deployed in browsers - "real time communication on the Web".

It intends to serve as a starting and coordination point to make sure all the parts that are needed to achieve this goal are findable, and that the parts that belong in the Internet protocol suite are fully specified and on the right publication track.

This document is an Applicability Statement - it does not itself specify any protocol, but specifies which other specifications WebRTC compliant implementations are supposed to follow.

This document is a work item of the RTCWEB working group.

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 May 16, 2018.

Copyright Notice

Copyright (c) 2017 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 . . . . .	3
2. Principles and Terminology . . . . .	4
2.1. Goals of this document . . . . .	4
2.2. Relationship between API and protocol . . . . .	5
2.3. On interoperability and innovation . . . . .	7
2.4. Terminology . . . . .	8
3. Architecture and Functionality groups . . . . .	8
4. Data transport . . . . .	12
5. Data framing and securing . . . . .	13
6. Data formats . . . . .	13
7. Connection management . . . . .	13
8. Presentation and control . . . . .	14
9. Local system support functions . . . . .	14
10. IANA Considerations . . . . .	15
11. Security Considerations . . . . .	15
12. Acknowledgements . . . . .	16
13. References . . . . .	16
13.1. Normative References . . . . .	16
13.2. Informative References . . . . .	18
Appendix A. Change log . . . . .	20
A.1. Changes from draft-alvestrand-dispatch-rtcweb-datagram-00 to -01 . . . . .	20
A.2. Changes from draft-alvestrand-dispatch-01 to draft-alvestrand-rtcweb-overview-00 . . . . .	20
A.3. Changes from draft-alvestrand-rtcweb-00 to -01 . . . . .	20
A.4. Changes from draft-alvestrand-rtcweb-overview-01 to draft-ietf-rtcweb-overview-00 . . . . .	21
A.5. Changes from -00 to -01 of draft-ietf-rtcweb-overview . . . . .	21
A.6. Changes from -01 to -02 of draft-ietf-rtcweb-overview . . . . .	21
A.7. Changes from -02 to -03 of draft-ietf-rtcweb-overview . . . . .	21
A.8. Changes from -03 to -04 of draft-ietf-rtcweb-overview . . . . .	22
A.9. Changes from -04 to -05 of draft-ietf-rtcweb-overview . . . . .	22
A.10. Changes from -05 to -06 . . . . .	22
A.11. Changes from -06 to -07 . . . . .	22
A.12. Changes from -07 to -08 . . . . .	22
A.13. Changes from -08 to -09 . . . . .	22

A.14. Changes from -09 to -10 . . . . .	22
A.15. Changes from -10 to -11 . . . . .	23
A.16. Changes from -11 to -12 . . . . .	23
A.17. Changes from -12 to -13 . . . . .	23
A.18. Changes from -13 to -14 . . . . .	23
A.19. Changes from -14 to -15 . . . . .	23
A.20. Changes from -15 to -16 . . . . .	23
A.21. Changes from -16 to -17 . . . . .	24
A.22. Changes from -17 to -18 . . . . .	24
A.23. Changes from -18 to -19 . . . . .	24
Author's Address . . . . .	24

## 1. Introduction

The Internet was, from very early in its lifetime, considered a possible vehicle for the deployment of real-time, interactive applications - with the most easily imaginable being audio conversations (aka "Internet telephony") and video conferencing.

The first attempts to build this were dependent on special networks, special hardware and custom-built software, often at very high prices or at low quality, placing great demands on the infrastructure.

As the available bandwidth has increased, and as processors and other hardware has become ever faster, the barriers to participation have decreased, and it has become possible to deliver a satisfactory experience on commonly available computing hardware.

Still, there are a number of barriers to the ability to communicate universally - one of these is that there is, as of yet, no single set of communication protocols that all agree should be made available for communication; another is the sheer lack of universal identification systems (such as is served by telephone numbers or email addresses in other communications systems).

Development of The Universal Solution has, however, proved hard.

The last few years have also seen a new platform rise for deployment of services: The browser-embedded application, or "Web application". It turns out that as long as the browser platform has the necessary interfaces, it is possible to deliver almost any kind of service on it.

Traditionally, these interfaces have been delivered by plugins, which had to be downloaded and installed separately from the browser; in the development of HTML5, application developers see much promise in the possibility of making those interfaces available in a standardized way within the browser.



This memo describes a set of building blocks that can be made accessible and controllable through a Javascript API in a browser, and which together form a sufficient set of functions to allow the use of interactive audio and video in applications that communicate directly between browsers across the Internet. The resulting protocol suite is intended to enable all the applications that are described as required scenarios in the use cases document [RFC7478].

Other efforts, for instance the W3C Web Real-Time Communications, Web Applications Security, and Device and Sensor working groups, focus on making standardized APIs and interfaces available, within or alongside the HTML5 effort, for those functions. This memo concentrates on specifying the protocols and subprotocols that are needed to specify the interactions over the network.

Operators should note that deployment of WebRTC will result in a change in the nature of signaling for real time media on the network, and may result in a shift in the kinds of devices used to create and consume such media. In the case of signaling, WebRTC session setup will typically occur over TLS-secured web technologies using application-specific protocols. Operational techniques that involve inserting network elements to interpret SDP -- either through endpoint cooperation [RFC3361] or through the transparent insertion of SIP Application Level Gateways (ALGs) -- will not work with such signaling. In the case of networks using cooperative endpoints, the approaches defined in [RFC8155] may serve as a suitable replacement for [RFC3361]. The increase in browser-based communications may also lead to a shift away from dedicated real-time-communications hardware, such as SIP desk phones. This will diminish the efficacy of operational techniques that place dedicated real-time devices on their own network segment, address range, or VLAN for purposes such as applying traffic filtering and QoS. Applying the markings described in [I-D.ietf-tsvwg-rtcweb-qos] may be appropriate replacements for such techniques.

This memo uses the term "WebRTC" (note the case used) to refer to the overall effort consisting of both IETF and W3C efforts.

## 2. Principles and Terminology

### 2.1. Goals of this document

The goal of the WebRTC protocol specification is to specify a set of protocols that, if all are implemented, will allow an implementation to communicate with another implementation using audio, video and data sent along the most direct possible path between the participants.

This document is intended to serve as the roadmap to the WebRTC specifications. It defines terms used by other parts of the WebRTC protocol specifications, lists references to other specifications that don't need further elaboration in the WebRTC context, and gives pointers to other documents that form part of the WebRTC suite.

By reading this document and the documents it refers to, it should be possible to have all information needed to implement a WebRTC compatible implementation.

## 2.2. Relationship between API and protocol

The total WebRTC effort consists of two major parts, each consisting of multiple documents:

- o A protocol specification, done in the IETF
- o A Javascript API specification, defined in a series of W3C documents  
[W3C.WD-webrtc-20120209][W3C.WD-mediacapture-streams-20120628]

Together, these two specifications aim to provide an environment where Javascript embedded in any page, when suitably authorized by its user, is able to set up communication using audio, video and auxiliary data, as long as the browser supports this specification. The browser environment does not constrain the types of application in which this functionality can be used.

The protocol specification does not assume that all implementations implement this API; it is not intended to be necessary for interoperation to know whether the entity one is communicating with is a browser or another device implementing this specification.

The goal of cooperation between the protocol specification and the API specification is that for all options and features of the protocol specification, it should be clear which API calls to make to exercise that option or feature; similarly, for any sequence of API calls, it should be clear which protocol options and features will be invoked. Both subject to constraints of the implementation, of course.

The following terms are used across the documents specifying the WebRTC suite, in the specific meanings given here. Not all terms are used in this document. Other terms are used in their commonly used meaning.

Agent: Undefined term. See "SDP Agent" and "ICE Agent".

**Application Programming Interface (API):** A specification of a set of calls and events, usually tied to a programming language or an abstract formal specification such as WebIDL, with its defined semantics.

**Browser:** Used synonymously with "Interactive User Agent" as defined in the HTML specification [W3C.WD-html5-20110525]. See also "WebRTC User Agent".

**Data Channel:** An abstraction that allows data to be sent between WebRTC endpoints in the form of messages. Two endpoints can have multiple data channels between them.

**ICE Agent:** An implementation of the Interactive Connectivity Establishment (ICE) [RFC5245] protocol. An ICE Agent may also be an SDP Agent, but there exist ICE Agents that do not use SDP (for instance those that use Jingle [XEP-0166]).

**Interactive:** Communication between multiple parties, where the expectation is that an action from one party can cause a reaction by another party, and the reaction can be observed by the first party, with the total time required for the action/reaction/observation is on the order of no more than hundreds of milliseconds.

**Media:** Audio and video content. Not to be confused with "transmission media" such as wires.

**Media Path:** The path that media data follows from one WebRTC endpoint to another.

**Protocol:** A specification of a set of data units, their representation, and rules for their transmission, with their defined semantics. A protocol is usually thought of as going between systems.

**Real-time Media:** Media where generation of content and display of content are intended to occur closely together in time (on the order of no more than hundreds of milliseconds). Real-time media can be used to support interactive communication.

**SDP Agent:** The protocol implementation involved in the Session Description Protocol (SDP) offer/answer exchange, as defined in [RFC3264] section 3.

**Signaling:** Communication that happens in order to establish, manage and control media paths and data paths.

**Signaling Path:** The communication channels used between entities participating in signaling to transfer signaling. There may be more entities in the signaling path than in the media path.

**WebRTC Browser:** (also called a WebRTC User Agent or WebRTC UA) Something that conforms to both the protocol specification and the Javascript API cited above.

**WebRTC non-Browser:** Something that conforms to the protocol specification, but does not claim to implement the Javascript API. This can also be called a "WebRTC device" or "WebRTC native application".

**WebRTC Endpoint:** Either a WebRTC browser or a WebRTC non-browser. It conforms to the protocol specification.

**WebRTC-compatible Endpoint:** An endpoint that is able to successfully communicate with a WebRTC endpoint, but may fail to meet some requirements of a WebRTC endpoint. This may limit where in the network such an endpoint can be attached, or may limit the security guarantees that it offers to others. It is not constrained by this specification; when it is mentioned at all, it is to note the implications on WebRTC-compatible endpoints of the requirements placed on WebRTC endpoints.

**WebRTC Gateway:** A WebRTC-compatible endpoint that mediates media traffic to non-WebRTC entities.

All WebRTC browsers are WebRTC endpoints, so any requirement on a WebRTC endpoint also applies to a WebRTC browser.

A WebRTC non-browser may be capable of hosting applications in a similar way to the way in which a browser can host Javascript applications, typically by offering APIs in other languages. For instance it may be implemented as a library that offers a C++ API intended to be loaded into applications. In this case, similar security considerations as for Javascript may be needed; however, since such APIs are not defined or referenced here, this document cannot give any specific rules for those interfaces.

WebRTC gateways are described in a separate document, [I-D.ietf-rtcweb-gateways].

### 2.3. On interoperability and innovation

The "Mission statement of the IETF" [RFC3935] states that "The benefit of a standard to the Internet is in interoperability - that

multiple products implementing a standard are able to work together in order to deliver valuable functions to the Internet's users."

Communication on the Internet frequently occurs in two phases:

- o Two parties communicate, through some mechanism, what functionality they both are able to support
- o They use that shared communicative functionality to communicate, or, failing to find anything in common, give up on communication.

There are often many choices that can be made for communicative functionality; the history of the Internet is rife with the proposal, standardization, implementation, and success or failure of many types of options, in all sorts of protocols.

The goal of having a mandatory to implement function set is to prevent negotiation failure, not to preempt or prevent negotiation.

The presence of a mandatory to implement function set serves as a strong changer of the marketplace of deployment - in that it gives a guarantee that, as long as you conform to a specification, and the other party is willing to accept communication at the base level of that specification, you can communicate successfully.

The alternative, that is having no mandatory to implement, does not mean that you cannot communicate, it merely means that in order to be part of the communications partnership, you have to implement the standard "and then some". The "and then some" is usually called a profile of some sort; in the version most antithetical to the Internet ethos, that "and then some" consists of having to use a specific vendor's product only.

#### 2.4. 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. Architecture and Functionality groups

For browser-based applications, the model for real-time support does not assume that the browser will contain all the functions needed for an application such as a telephone or a video conference. The vision is that the browser will have the functions needed for a Web application, working in conjunction with its backend servers, to implement these functions.

This means that two vital interfaces need specification: The protocols that browsers use to talk to each other, without any intervening servers, and the APIs that are offered for a Javascript application to take advantage of the browser's functionality.

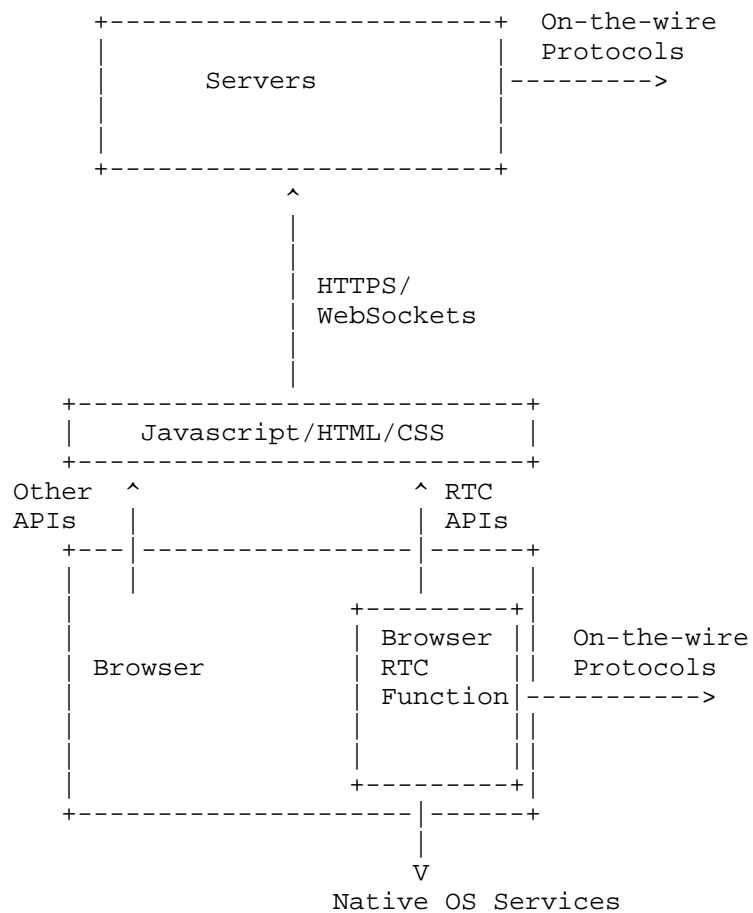


Figure 1: Browser Model

Note that HTTPS and WebSockets are also offered to the Javascript application through browser APIs.

As for all protocol and API specifications, there is no restriction that the protocols can only be used to talk to another browser; since they are fully specified, any endpoint that implements the protocols faithfully should be able to interoperate with the application running in the browser.

A commonly imagined model of deployment is the one depicted below. In the figure below JS is Javascript.

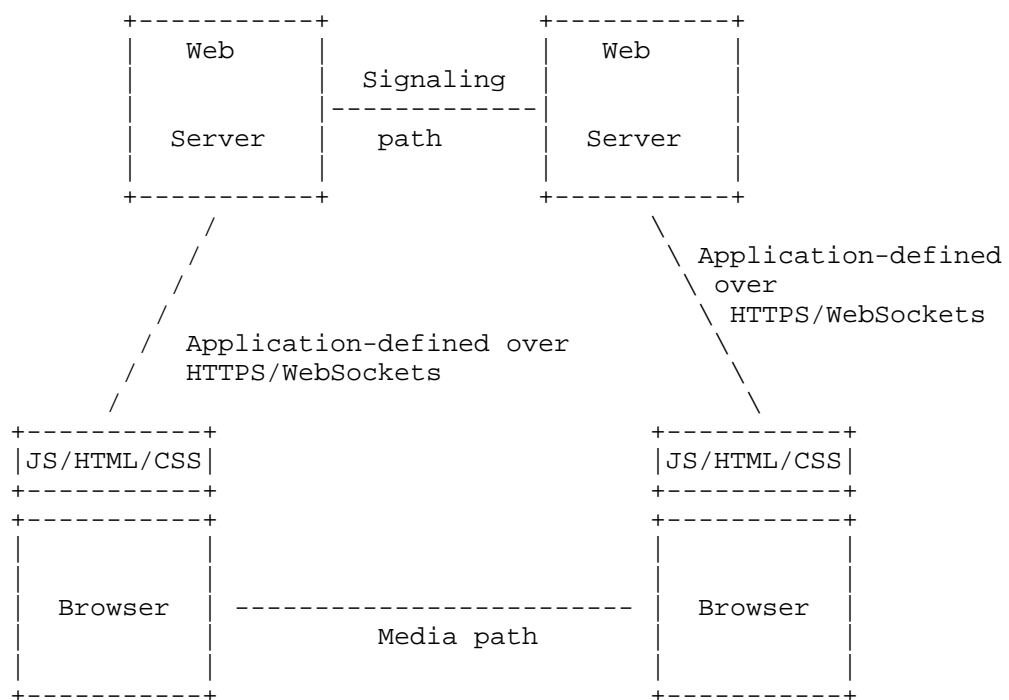


Figure 2: Browser RTC Trapezoid

On this drawing, the critical part to note is that the media path ("low path") goes directly between the browsers, so it has to be conformant to the specifications of the WebRTC protocol suite; the signaling path ("high path") goes via servers that can modify, translate or manipulate the signals as needed.

If the two Web servers are operated by different entities, the inter-server signaling mechanism needs to be agreed upon, either by

standardization or by other means of agreement. Existing protocols (e.g. SIP [RFC3261] or XMPP [RFC6120]) could be used between servers, while either a standards-based or proprietary protocol could be used between the browser and the web server.

For example, if both operators' servers implement SIP, SIP could be used for communication between servers, along with either a standardized signaling mechanism (e.g. SIP over WebSockets) or a proprietary signaling mechanism used between the application running in the browser and the web server. Similarly, if both operators' servers implement Extensible Messaging and Presence Protocol (XMPP), XMPP could be used for communication between XMPP servers, with either a standardized signaling mechanism (e.g. XMPP over WebSockets or BOSH [XEP-0124] or a proprietary signaling mechanism used between the application running in the browser and the web server.

The choice of protocols for client-server and inter-server signalling, and definition of the translation between them, is outside the scope of the WebRTC protocol suite described in the document.

The functionality groups that are needed in the browser can be specified, more or less from the bottom up, as:

- o Data transport: such as TCP, UDP and the means to securely set up connections between entities, as well as the functions for deciding when to send data: congestion management, bandwidth estimation and so on.
- o Data framing: RTP, SCTP, DTLS, and other data formats that serve as containers, and their functions for data confidentiality and integrity.
- o Data formats: Codec specifications, format specifications and functionality specifications for the data passed between systems. Audio and video codecs, as well as formats for data and document sharing, belong in this category. In order to make use of data formats, a way to describe them, a session description, is needed.
- o Connection management: Setting up connections, agreeing on data formats, changing data formats during the duration of a call; SDP, SIP, and Jingle/XMPP belong in this category.
- o Presentation and control: What needs to happen in order to ensure that interactions behave in a non-surprising manner. This can include floor control, screen layout, voice activated image switching and other such functions - where part of the system require the cooperation between parties. XCON and Cisco/



Tandberg's TIP were some attempts at specifying this kind of functionality; many applications have been built without standardized interfaces to these functions.

- o Local system support functions: These are things that need not be specified uniformly, because each participant may choose to do these in a way of the participant's choosing, without affecting the bits on the wire in a way that others have to be cognizant of. Examples in this category include echo cancellation (some forms of it), local authentication and authorization mechanisms, OS access control and the ability to do local recording of conversations.

Within each functionality group, it is important to preserve both freedom to innovate and the ability for global communication. Freedom to innovate is helped by doing the specification in terms of interfaces, not implementation; any implementation able to communicate according to the interfaces is a valid implementation. Ability to communicate globally is helped both by having core specifications be unencumbered by IPR issues and by having the formats and protocols be fully enough specified to allow for independent implementation.

One can think of the three first groups as forming a "media transport infrastructure", and of the three last groups as forming a "media service". In many contexts, it makes sense to use a common specification for the media transport infrastructure, which can be embedded in browsers and accessed using standard interfaces, and "let a thousand flowers bloom" in the "media service" layer; to achieve interoperable services, however, at least the first five of the six groups need to be specified.

#### 4. Data transport

Data transport refers to the sending and receiving of data over the network interfaces, the choice of network-layer addresses at each end of the communication, and the interaction with any intermediate entities that handle the data, but do not modify it (such as TURN relays).

It includes necessary functions for congestion control, retransmission, and in-order delivery.

WebRTC endpoints MUST implement the transport protocols described in [I-D.ietf-rtcweb-transports].

## 5. Data framing and securing

The format for media transport is RTP [RFC3550]. Implementation of SRTP [RFC3711] is REQUIRED for all implementations.

The detailed considerations for usage of functions from RTP and SRTP are given in [I-D.ietf-rtcweb-rtp-usage]. The security considerations for the WebRTC use case are in [I-D.ietf-rtcweb-security], and the resulting security functions are described in [I-D.ietf-rtcweb-security-arch].

Considerations for the transfer of data that is not in RTP format is described in [I-D.ietf-rtcweb-data-channel], and a supporting protocol for establishing individual data channels is described in [I-D.ietf-rtcweb-data-protocol]. WebRTC endpoints MUST implement these two specifications.

WebRTC endpoints MUST implement [I-D.ietf-rtcweb-rtp-usage], [I-D.ietf-rtcweb-security], [I-D.ietf-rtcweb-security-arch], and the requirements they include.

## 6. Data formats

The intent of this specification is to allow each communications event to use the data formats that are best suited for that particular instance, where a format is supported by both sides of the connection. However, a minimum standard is greatly helpful in order to ensure that communication can be achieved. This document specifies a minimum baseline that will be supported by all implementations of this specification, and leaves further codecs to be included at the will of the implementor.

WebRTC endpoints that support audio and/or video MUST implement the codecs and profiles required in [RFC7874] and [RFC7742].

## 7. Connection management

The methods, mechanisms and requirements for setting up, negotiating and tearing down connections is a large subject, and one where it is desirable to have both interoperability and freedom to innovate.

The following principles apply:

1. The WebRTC media negotiations will be capable of representing the same SDP offer/answer semantics [RFC3264] that are used in SIP, in such a way that it is possible to build a signaling gateway between SIP and the WebRTC media negotiation.

2. It will be possible to gateway between legacy SIP devices that support ICE and appropriate RTP / SDP mechanisms, codecs and security mechanisms without using a media gateway. A signaling gateway to convert between the signaling on the web side to the SIP signaling may be needed.
3. When an SDP for a new codec is specified, no other standardization should be required for it to be possible to use that in the web browsers. Adding new codecs which might have new SDP parameters should not change the APIs between the browser and Javascript application. As soon as the browsers support the new codecs, old applications written before the codecs were specified should automatically be able to use the new codecs where appropriate with no changes to the JS applications.

The particular choices made for WebRTC, and their implications for the API offered by a browser implementing WebRTC, are described in [I-D.ietf-rtcweb-jsep].

WebRTC browsers MUST implement [I-D.ietf-rtcweb-jsep].

WebRTC endpoints MUST implement the functions described in that document that relate to the network layer (e.g. Bundle [I-D.ietf-mmusic-sdp-bundle-negotiation], RTCP-mux [RFC5761] and Trickle ICE [I-D.ietf-ice-trickle]), but do not need to support the API functionality described there.

## 8. Presentation and control

The most important part of control is the user's control over the browser's interaction with input/output devices and communications channels. It is important that the user have some way of figuring out where his audio, video or texting is being sent, for what purported reason, and what guarantees are made by the parties that form part of this control channel. This is largely a local function between the browser, the underlying operating system and the user interface; this is specified in the peer connection API [W3C.WD-webrtc-20120209], and the media capture API [W3C.WD-mediacapture-streams-20120628].

WebRTC browsers MUST implement these two specifications.

## 9. Local system support functions

These are characterized by the fact that the quality of these functions strongly influence the user experience, but the exact algorithm does not need coordination. In some cases (for instance echo cancellation, as described below), the overall system definition

may need to specify that the overall system needs to have some characteristics for which these facilities are useful, without requiring them to be implemented a certain way.

Local functions include echo cancellation, volume control, camera management including focus, zoom, pan/tilt controls (if available), and more.

One would want to see certain parts of the system conform to certain properties, for instance:

- o Echo cancellation should be good enough to achieve the suppression of acoustical feedback loops below a perceptually noticeable level.
- o Privacy concerns MUST be satisfied; for instance, if remote control of camera is offered, the APIs should be available to let the local participant figure out who's controlling the camera, and possibly decide to revoke the permission for camera usage.
- o Automatic gain control, if present, should normalize a speaking voice into a reasonable dB range.

The requirements on WebRTC systems with regard to audio processing are found in [RFC7874] and includes more guidance about echo cancellation and AGC; the proposed API for control of local devices are found in [W3C.WD-mediacapture-streams-20120628].

WebRTC endpoints MUST implement the processing functions in [RFC7874]. (Together with the requirement in Section 6, this means that WebRTC endpoints MUST implement the whole document.)

## 10. IANA Considerations

This document makes no request of IANA.

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

## 11. Security Considerations

Security of the web-enabled real time communications comes in several pieces:

- o Security of the components: The browsers, and other servers involved. The most target-rich environment here is probably the browser; the aim here should be that the introduction of these components introduces no additional vulnerability.

- o Security of the communication channels: It should be easy for a participant to reassure himself of the security of his communication - by verifying the crypto parameters of the links he himself participates in, and to get reassurances from the other parties to the communication that they promise that appropriate measures are taken.
- o Security of the partners' identity: verifying that the participants are who they say they are (when positive identification is appropriate), or that their identity cannot be uncovered (when anonymity is a goal of the application).

The security analysis, and the requirements derived from that analysis, is contained in [I-D.ietf-rtcweb-security].

It is also important to read the security sections of [W3C.WD-mediacapture-streams-20120628] and [W3C.WD-webrtc-20120209].

## 12. Acknowledgements

The number of people who have taken part in the discussions surrounding this draft are too numerous to list, or even to identify. The ones below have made special, identifiable contributions; this does not mean that others' contributions are less important.

Thanks to Cary Bran, Cullen Jennings, Colin Perkins, Magnus Westerlund and Joerg Ott, who offered technical contributions on various versions of the draft.

Thanks to Jonathan Rosenberg, Matthew Kaufman and others at Skype for the ASCII drawings in section 1.

Thanks to Alissa Cooper, Bjoern Hoehrmann, Colin Perkins, Colton Shields, Eric Rescorla, Heath Matlock, Henry Sinnreich, Justin Uberti, Keith Drage, Magnus Westerlund, Olle E. Johansson, Sean Turner and Simon Leinen for document review.

## 13. References

### 13.1. Normative References

[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.

- [I-D.ietf-rtcweb-data-protocol]  
Jesup, R., Loreto, S., and M. Tuexen, "WebRTC Data Channel Establishment Protocol", draft-ietf-rtcweb-data-protocol-09 (work in progress), January 2015.
- [I-D.ietf-rtcweb-jsep]  
Uberti, J., Jennings, C., and E. Rescorla, "JavaScript Session Establishment Protocol", draft-ietf-rtcweb-jsep-24 (work in progress), October 2017.
- [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-26 (work in progress), March 2016.
- [I-D.ietf-rtcweb-security]  
Rescorla, E., "Security Considerations for WebRTC", draft-ietf-rtcweb-security-09 (work in progress), October 2017.
- [I-D.ietf-rtcweb-security-arch]  
Rescorla, E., "WebRTC Security Architecture", draft-ietf-rtcweb-security-arch-13 (work in progress), October 2017.
- [I-D.ietf-rtcweb-transports]  
Alvestrand, H., "Transports for WebRTC", draft-ietf-rtcweb-transports-17 (work in progress), October 2016.
- [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>>.
- [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>>.
- [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>>.

- [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, <<https://www.rfc-editor.org/info/rfc5245>>.
- [RFC7742] Roach, A., "WebRTC Video Processing and Codec Requirements", RFC 7742, DOI 10.17487/RFC7742, March 2016, <<https://www.rfc-editor.org/info/rfc7742>>.
- [RFC7874] Valin, JM. and C. Bran, "WebRTC Audio Codec and Processing Requirements", RFC 7874, DOI 10.17487/RFC7874, May 2016, <<https://www.rfc-editor.org/info/rfc7874>>.
- [W3C.WD-mediacapture-streams-20120628]  
Burnett, D. and A. Narayanan, "Media Capture and Streams", World Wide Web Consortium WD WD-mediacapture-streams-20120628, June 2012, <<http://www.w3.org/TR/2012/WD-mediacapture-streams-20120628>>.
- [W3C.WD-webrtc-20120209]  
Bergkvist, A., Burnett, D., Jennings, C., and A. Narayanan, "WebRTC 1.0: Real-time Communication Between Browsers", World Wide Web Consortium WD WD-webrtc-20120209, February 2012, <<http://www.w3.org/TR/2012/WD-webrtc-20120209>>.

### 13.2. Informative References

- [I-D.ietf-ice-trickle]  
Ivov, E., Rescorla, E., Uberti, J., and P. Saint-Andre, "Trickle ICE: Incremental Provisioning of Candidates for the Interactive Connectivity Establishment (ICE) Protocol", draft-ietf-ice-trickle-14 (work in progress), September 2017.
- [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-39 (work in progress), August 2017.
- [I-D.ietf-rtcweb-gateways]  
Alvestrand, H. and U. Rauschenbach, "WebRTC Gateways", draft-ietf-rtcweb-gateways-02 (work in progress), January 2016.

- [I-D.ietf-tsvwg-rtcweb-qos]  
Jones, P., Dhesikan, S., Jennings, C., and D. Druta, "DSCP Packet Markings for WebRTC QoS", draft-ietf-tsvwg-rtcweb-qos-18 (work in progress), August 2016.
- [RFC3261] Rosenberg, J., Schulzrinne, H., Camarillo, G., Johnston, A., Peterson, J., Sparks, R., Handley, M., and E. Schooler, "SIP: Session Initiation Protocol", RFC 3261, DOI 10.17487/RFC3261, June 2002, <<https://www.rfc-editor.org/info/rfc3261>>.
- [RFC3361] Schulzrinne, H., "Dynamic Host Configuration Protocol (DHCP-for-IPv4) Option for Session Initiation Protocol (SIP) Servers", RFC 3361, DOI 10.17487/RFC3361, August 2002, <<https://www.rfc-editor.org/info/rfc3361>>.
- [RFC3935] Alvestrand, H., "A Mission Statement for the IETF", BCP 95, RFC 3935, DOI 10.17487/RFC3935, October 2004, <<https://www.rfc-editor.org/info/rfc3935>>.
- [RFC5761] Perkins, C. and M. Westerlund, "Multiplexing RTP Data and Control Packets on a Single Port", RFC 5761, DOI 10.17487/RFC5761, April 2010, <<https://www.rfc-editor.org/info/rfc5761>>.
- [RFC6120] Saint-Andre, P., "Extensible Messaging and Presence Protocol (XMPP): Core", RFC 6120, DOI 10.17487/RFC6120, March 2011, <<https://www.rfc-editor.org/info/rfc6120>>.
- [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>>.
- [RFC8155] Patil, P., Reddy, T., and D. Wing, "Traversal Using Relays around NAT (TURN) Server Auto Discovery", RFC 8155, DOI 10.17487/RFC8155, April 2017, <<https://www.rfc-editor.org/info/rfc8155>>.
- [W3C.WD-html5-20110525]  
Hickson, I., "HTML5", World Wide Web Consortium LastCall WD-html5-20110525, May 2011, <<http://www.w3.org/TR/2011/WD-html5-20110525>>.
- [XEP-0124]  
Paterson, I., Smith, D., Saint-Andre, P., Moffitt, J., Stout, L., and W. Tilanus, "BOSH", XSF XEP 0124, November 2016.



[XEP-0166]

Ludwig, S., Beda, J., Saint-Andre, P., McQueen, R., Egan, S., and J. Hildebrand, "Jingle", XSF XEP 0166, June 2007.

## Appendix A. Change log

This section may be deleted by the RFC Editor when preparing for publication.

### A.1. Changes from draft-alvestrand-dispatch-rtcweb-datagram-00 to -01

Added section "On interoperability and innovation"

Added data confidentiality and integrity to the "data framing" layer

Added congestion management requirements in the "data transport" layer section

Changed need for non-media data from "question: do we need this?" to "Open issue: How do we do this?"

Strengthened disclaimer that listed codecs are placeholders, not decisions.

More details on why the "local system support functions" section is there.

### A.2. Changes from draft-alvestrand-dispatch-01 to draft-alvestrand-rtcweb-overview-00

Added section on "Relationship between API and protocol"

Added terminology section

Mentioned congestion management as part of the "data transport" layer in the layer list

### A.3. Changes from draft-alvestrand-rtcweb-00 to -01

Removed most technical content, and replaced with pointers to drafts as requested and identified by the RTCWEB WG chairs.

Added content to acknowledgments section.

Added change log.

Spell-checked document.

A.4. Changes from draft-alvestrand-rtcweb-overview-01 to draft-ietf-rtcweb-overview-00

Changed draft name and document date.

Removed unused references

A.5. Changes from -00 to -01 of draft-ietf-rtcweb-overview

Added architecture figures to section 2.

Changed the description of "echo cancellation" under "local system support functions".

Added a few more definitions.

A.6. Changes from -01 to -02 of draft-ietf-rtcweb-overview

Added pointers to use cases, security and rtp-usage drafts (now WG drafts).

Changed description of SRTP from mandatory-to-use to mandatory-to-implement.

Added the "3 principles of negotiation" to the connection management section.

Added an explicit statement that ICE is required for both NAT and consent-to-receive.

A.7. Changes from -02 to -03 of draft-ietf-rtcweb-overview

Added references to a number of new drafts.

Expanded the description text under the "trapezoid" drawing with some more text discussed on the list.

Changed the "Connection management" sentence from "will be done using SDP offer/answer" to "will be capable of representing SDP offer/answer" - this seems more consistent with JSEP.

Added "security mechanisms" to the things a non-gateways SIP devices must support in order to not need a media gateway.

Added a definition for "browser".

## A.8. Changes from -03 to -04 of draft-ietf-rtcweb-overview

Made introduction more normative.

Several wording changes in response to review comments from EKR

Added an appendix to hold references and notes that are not yet in a separate document.

## A.9. Changes from -04 to -05 of draft-ietf-rtcweb-overview

Minor grammatical fixes. This is mainly a "keepalive" refresh.

## A.10. Changes from -05 to -06

Clarifications in response to Last Call review comments. Inserted reference to draft-ietf-rtcweb-audio.

## A.11. Changes from -06 to -07

Added a reference to the "unified plan" draft, and updated some references.

Otherwise, it's a "keepalive" draft.

## A.12. Changes from -07 to -08

Removed the appendix that detailed transports, and replaced it with a reference to draft-ietf-rtcweb-transports. Removed now-unused references.

## A.13. Changes from -08 to -09

Added text to the Abstract indicating that the intended status is an Applicability Statement.

## A.14. Changes from -09 to -10

Defined "WebRTC Browser" and "WebRTC device" as things that do, or don't, conform to the API.

Updated reference to data-protocol draft

Updated data formats to reference -rtcweb-audio- and not the expired -cbran draft.

Deleted references to -unified-plan

Deleted reference to -generic-idp (draft expired)

Added notes on which referenced documents WebRTC browsers or devices MUST conform to.

Added pointer to the security section of the API drafts.

A.15. Changes from -10 to -11

Added "WebRTC Gateway" as a third class of device, and referenced the doc describing them.

Made a number of text clarifications in response to document reviews.

A.16. Changes from -11 to -12

Refined entity definitions to define "WebRTC endpoint" and "WebRTC-compatible endpoint".

Changed remaining usage of the term "RTCWEB" to "WebRTC", including in the page header.

A.17. Changes from -12 to -13

Changed "WebRTC device" to be "WebRTC non-browser", per decision at IETF 91. This led to the need for "WebRTC endpoint" as the common label for both, and the usage of that term in the rest of the document.

Added words about WebRTC APIs in languages other than Javascript.

Referenced draft-ietf-rtcweb-video for video codecs to support.

A.18. Changes from -13 to -14

None. This is a "keepalive" update.

A.19. Changes from -14 to -15

Changed "gateways" reference to point to the WG document.

A.20. Changes from -15 to -16

None. This is a "keepalive" publication.

A.21. Changes from -16 to -17

Addressed review comments by Olle E. Johansson and Magnus Westerlund

A.22. Changes from -17 to -18

Addressed review comments from Sean Turner and Alissa Cooper

A.23. Changes from -18 to -19

A number of grammatical issues were fixed.

Added note on operational impact of WebRTC.

Unified all definitions into the definitions list.

Added a reference for BOSH.

Changed ICE reference from 5245bis to RFC 5245.

Author's Address

Harald T. Alvestrand  
Google  
Kungsbron 2  
Stockholm 11122  
Sweden

Email: harald@alvestrand.no

Network Working Group  
Internet-Draft  
Intended status: Standards Track  
Expires: August 29, 2013

C. Perkins  
University of Glasgow  
M. Westerlund  
Ericsson  
J. Ott  
Aalto University  
February 25, 2013

Web Real-Time Communication (WebRTC): Media Transport and Use of RTP  
draft-ietf-rtcweb-rtp-usage-06

Abstract

The Web Real-Time Communication (WebRTC) framework provides support for direct interactive rich communication using audio, video, text, collaboration, games, etc. between two peers' web-browsers. This memo describes the media transport aspects of the WebRTC framework. It specifies how the Real-time Transport Protocol (RTP) is used in the WebRTC context, and gives requirements for which RTP features, profiles, and extensions need to be supported.

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 August 29, 2013.

Copyright Notice

Copyright (c) 2013 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 . . . . .	4
2. Rationale . . . . .	4
3. Terminology . . . . .	5
4. WebRTC Use of RTP: Core Protocols . . . . .	6
4.1. RTP and RTCP . . . . .	6
4.2. Choice of the RTP Profile . . . . .	7
4.3. Choice of RTP Payload Formats . . . . .	8
4.4. RTP Session Multiplexing . . . . .	8
4.5. RTP and RTCP Multiplexing . . . . .	9
4.6. Reduced Size RTCP . . . . .	10
4.7. Symmetric RTP/RTCP . . . . .	10
4.8. Choice of RTP Synchronisation Source (SSRC) . . . . .	10
4.9. Generation of the RTCP Canonical Name (CNAME) . . . . .	11
5. WebRTC Use of RTP: Extensions . . . . .	11
5.1. Conferencing Extensions . . . . .	11
5.1.1. Full Intra Request (FIR) . . . . .	12
5.1.2. Picture Loss Indication (PLI) . . . . .	13
5.1.3. Slice Loss Indication (SLI) . . . . .	13
5.1.4. Reference Picture Selection Indication (RPSI) . . . . .	13
5.1.5. Temporal-Spatial Trade-off Request (TSTR) . . . . .	13
5.1.6. Temporary Maximum Media Stream Bit Rate Request (TMMBR) . . . . .	13
5.2. Header Extensions . . . . .	14
5.2.1. Rapid Synchronisation . . . . .	14
5.2.2. Client-to-Mixer Audio Level . . . . .	14
5.2.3. Mixer-to-Client Audio Level . . . . .	15
6. WebRTC Use of RTP: Improving Transport Robustness . . . . .	15
6.1. Negative Acknowledgements and RTP Retransmission . . . . .	15
6.2. Forward Error Correction (FEC) . . . . .	16
7. WebRTC Use of RTP: Rate Control and Media Adaptation . . . . .	17
7.1. Boundary Conditions and Circuit Breakers . . . . .	17
7.2. RTCP Extensions for Congestion Control . . . . .	18
7.3. RTCP Limitations for Congestion Control . . . . .	18
7.4. Congestion Control Interoperability With Legacy Systems . . . . .	19
8. WebRTC Use of RTP: Performance Monitoring . . . . .	20
9. WebRTC Use of RTP: Future Extensions . . . . .	20
10. Signalling Considerations . . . . .	20
11. WebRTC API Considerations . . . . .	22
12. RTP Implementation Considerations . . . . .	23

12.1. RTP Sessions and PeerConnections . . . . .	23
12.2. Multiple Sources . . . . .	24
12.3. Multiparty . . . . .	25
12.4. SSRC Collision Detection . . . . .	26
12.5. Contributing Sources and the CSRC List . . . . .	27
12.6. Media Synchronization . . . . .	27
12.7. Multiple RTP End-points . . . . .	28
12.8. Simulcast . . . . .	29
12.9. Differentiated Treatment of Flows . . . . .	29
13. Open Issues . . . . .	31
14. IANA Considerations . . . . .	32
15. Security Considerations . . . . .	32
16. Acknowledgements . . . . .	33
17. References . . . . .	33
17.1. Normative References . . . . .	33
17.2. Informative References . . . . .	36
Appendix A. Supported RTP Topologies . . . . .	38
A.1. Point to Point . . . . .	38
A.2. Multi-Unicast (Mesh) . . . . .	41
A.3. Mixer Based . . . . .	44
A.3.1. Media Mixing . . . . .	44
A.3.2. Media Switching . . . . .	47
A.3.3. Media Projecting . . . . .	50
A.4. Translator Based . . . . .	53
A.4.1. Transcoder . . . . .	53
A.4.2. Gateway / Protocol Translator . . . . .	54
A.4.3. Relay . . . . .	56
A.5. End-point Forwarding . . . . .	60
A.6. Simulcast . . . . .	61
Authors' Addresses . . . . .	62



## 1. Introduction

The Real-time Transport Protocol (RTP) [RFC3550] provides a framework for delivery of audio and video teleconferencing data and other real-time media applications. Previous work has defined the RTP protocol, along with numerous profiles, payload formats, and other extensions. When combined with appropriate signalling, these form the basis for many teleconferencing systems.

The Web Real-Time communication (WebRTC) framework provides the protocol building blocks to support direct, interactive, real-time communication using audio, video, collaboration, games, etc., between two peers' web-browsers. This memo describes how the RTP framework is to be used in the WebRTC context. It proposes a baseline set of RTP features that are to be implemented by all WebRTC-aware end-points, along with suggested extensions for enhanced functionality.

The WebRTC overview [I-D.ietf-rtcweb-overview] outlines the complete WebRTC framework, of which this memo is a part.

The structure of this memo is as follows. Section 2 outlines our rationale in preparing this memo and choosing these RTP features. Section 3 defines requirement terminology. Requirements for core RTP protocols are described in Section 4 and suggested RTP extensions are described in Section 5. Section 6 outlines mechanisms that can increase robustness to network problems, while Section 7 describes congestion control and rate adaptation mechanisms. The discussion of mandated RTP mechanisms concludes in Section 8 with a review of performance monitoring and network management tools that can be used in the WebRTC context. Section 9 gives some guidelines for future incorporation of other RTP and RTP Control Protocol (RTCP) extensions into this framework. Section 10 describes requirements placed on the signalling channel. Section 11 discusses the relationship between features of the RTP framework and the WebRTC application programming interface (API), and Section 12 discusses RTP implementation considerations. This memo concludes with an appendix discussing several different RTP Topologies, and how they affect the RTP session(s) and various implementation details of possible realization of central nodes.

## 2. Rationale

The RTP framework comprises the RTP data transfer protocol, the RTP control protocol, and numerous RTP payload formats, profiles, and extensions. This range of add-ons has allowed RTP to meet various needs that were not envisaged by the original protocol designers, and to support many new media encodings, but raises the question of what

extensions are to be supported by new implementations. The development of the WebRTC framework provides an opportunity for us to review the available RTP features and extensions, and to define a common baseline feature set for all WebRTC implementations of RTP. This builds on the past 15 years development of RTP to mandate the use of extensions that have shown widespread utility, while still remaining compatible with the wide installed base of RTP implementations where possible.

Other RTP and RTCP extensions not discussed in this document can be implemented by WebRTC end-points if they are beneficial for new use cases. However, they are not necessary to address the WebRTC use cases and requirements identified to date [I-D.ietf-rtcweb-use-cases-and-requirements].

While the baseline set of RTP features and extensions defined in this memo is targeted at the requirements of the WebRTC framework, it is expected to be broadly useful for other conferencing-related uses of RTP. In particular, it is likely that this set of RTP features and extensions will be appropriate for other desktop or mobile video conferencing systems, or for room-based high-quality telepresence applications.

### 3. 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]. The RFC 2119 interpretation of these key words applies only when written in ALL CAPS. Lower- or mixed-case uses of these key words are not to be interpreted as carrying special significance in this memo.

We define the following terms:

**RTP Media Stream:** A sequence of RTP packets, and associated RTCP packets, using a single synchronisation source (SSRC) that together carries part or all of the content of a specific Media Type from a specific sender source within a given RTP session.

**RTP Session:** As defined by [RFC3550], the endpoints belonging to the same RTP Session are those that share a single SSRC space. That is, those endpoints can see an SSRC identifier transmitted by any one of the other endpoints. An endpoint can see an SSRC either directly in RTP and RTCP packets, or as a contributing source (CSRC) in RTP packets from a mixer. The RTP Session scope is hence decided by the endpoints' network interconnection topology, in combination with RTP and RTCP forwarding strategies deployed by

endpoints and any interconnecting middle nodes.

WebRTC MediaStream: The MediaStream concept defined by the W3C in the API.

Other terms are used according to their definitions from the RTP Specification [RFC3550] and WebRTC overview [I-D.ietf-rtcweb-overview] documents.

#### 4. WebRTC Use of RTP: Core Protocols

The following sections describe the core features of RTP and RTCP that need to be implemented, along with the mandated RTP profiles and payload formats. Also described are the core extensions providing essential features that all WebRTC implementations need to implement to function effectively on today's networks.

##### 4.1. RTP and RTCP

The Real-time Transport Protocol (RTP) [RFC3550] is REQUIRED to be implemented as the media transport protocol for WebRTC. RTP itself comprises two parts: the RTP data transfer protocol, and the RTP control protocol (RTCP). RTCP is a fundamental and integral part of RTP, and MUST be implemented in all WebRTC applications.

The following RTP and RTCP features are sometimes omitted in limited functionality implementations of RTP, but are REQUIRED in all WebRTC implementations:

- o Support for use of multiple simultaneous SSRC values in a single RTP session, including support for RTP end-points that send many SSRC values simultaneously.
- o Random choice of SSRC on joining a session; collision detection and resolution for SSRC values (but see also Section 4.8).
- o Support for reception of RTP data packets containing CSRC lists, as generated by RTP mixers, and RTCP packets relating to CSRCs.
- o Support for sending correct synchronization information in the RTCP Sender Reports, to allow a receiver to implement lip-sync, with RECOMMENDED support for the rapid RTP synchronisation extensions (see Section 5.2.1).
- o Support for sending and receiving RTCP SR, RR, SDES, and BYE packet types, with OPTIONAL support for other RTCP packet types; implementations MUST ignore unknown RTCP packet types.

- o Support for multiple end-points in a single RTP session, and for scaling the RTCP transmission interval according to the number of participants in the session; support for randomised RTCP transmission intervals to avoid synchronisation of RTCP reports; support for RTCP timer reconsideration.
- o Support for configuring the RTCP bandwidth as a fraction of the media bandwidth, and for configuring the fraction of the RTCP bandwidth allocated to senders, e.g., using the SDP "b=" line.

It is known that a significant number of legacy RTP implementations, especially those targeted at VoIP-only systems, do not support all of the above features, and in some cases do not support RTCP at all. Implementers are advised to consider the requirements for graceful degradation when interoperating with legacy implementations.

Other implementation considerations are discussed in Section 12.

#### 4.2. Choice of the RTP Profile

The complete specification of RTP for a particular application domain requires the choice of an RTP Profile. For WebRTC use, the "Extended Secure RTP Profile for Real-time Transport Control Protocol (RTCP)-Based Feedback (RTP/SAVPF)" [RFC5124] as extended by [I-D.ietf-avtcore-avp-codecs] MUST be implemented. This builds on the basic RTP/AVP profile [RFC3551], the RTP profile for RTCP-based feedback (RTP/AVPF) [RFC4585], and the secure RTP profile (RTP/SAVP) [RFC3711].

The RTCP-based feedback extensions [RFC4585] are needed for the improved RTCP timer model, that allows more flexible transmission of RTCP packets in response to events, rather than strictly according to bandwidth. This is vital for being able to report congestion events. These extensions also save RTCP bandwidth, and will commonly only use the full RTCP bandwidth allocation if there are many events that require feedback. They are also needed to make use of the RTP conferencing extensions discussed in Section 5.1.

Note: The enhanced RTCP timer model defined in the RTP/AVPF profile is backwards compatible with legacy systems that implement only the base RTP/AVP profile, given some constraints on parameter configuration such as the RTCP bandwidth value and "trr-int" (the most important factor for interworking with RTP/AVP end-points via a gateway is to set the trr-int parameter to a value representing 4 seconds).

The secure RTP profile [RFC3711] is needed to provide media encryption, integrity protection, replay protection and a limited

form of source authentication. WebRTC implementations MUST NOT send packets using the basic RTP/AVP profile or the RTP/AVPF profile; they MUST employ the full RTP/SAVPF profile to protect all RTP and RTCP packets that are generated. The default and mandatory to implement transforms listed in Section 5 of [RFC3711] SHALL apply.

Implementations MUST support DTLS-SRTP [RFC5764] for key-management. Other key management schemes MAY be supported.

#### 4.3. Choice of RTP Payload Formats

Implementations MUST follow the WebRTC Audio Codec and Processing Requirements [I-D.ietf-rtcweb-audio] and SHOULD follow the updated recommendations for audio codecs in the RTP/AVP Profile [I-D.ietf-avtcore-avp-codecs]. Support for other audio codecs is OPTIONAL.

(tbd: the mandatory to implement video codec is not yet decided)

Endpoints MAY signal support for multiple RTP payload formats, or multiple configurations of a single RTP payload format, provided each payload format uses a different RTP payload type number. An endpoint that has signalled support for multiple RTP payload formats SHOULD accept data in any of those payload formats at any time, unless it has previously signalled limitations on its decoding capability. This requirement is constrained if several media types are sent in the same RTP session. In such a case, a source (SSRC) is restricted to switching only between the RTP payload formats signalled for the media type that is being sent by that source; see Section 4.4. To support rapid rate adaptation by changing codec, RTP does not require advance signalling for changes between RTP payload formats that were signalled during session set-up.

An RTP sender that changes between two RTP payload types that use different RTP clock rates MUST follow the recommendations in Section 4.1 of [I-D.ietf-avtext-multiple-clock-rates]. RTP receivers MUST follow the recommendations in Section 4.3 of [I-D.ietf-avtext-multiple-clock-rates], in order to support sources that switch between clock rates in an RTP session (these recommendations for receivers are backwards compatible with the case where senders use only a single clock rate).

#### 4.4. RTP Session Multiplexing

An association amongst a set of participants communicating with RTP is known as an RTP session. A participant can be involved in multiple RTP sessions at the same time. In a multimedia session, each medium has typically been carried in a separate RTP session with

its own RTCP packets (i.e., one RTP session for the audio, with a separate RTP session using a different transport address for the video; if SDP is used, this corresponds to one RTP session for each "m=" line in the SDP). WebRTC implementations of RTP are REQUIRED to implement support for multimedia sessions in this way, for compatibility with legacy systems.

In today's networks, however, with the widespread use of Network Address/Port Translators (NAT/NAPT) and Firewalls (FW), it is desirable to reduce the number of transport addresses used by real-time media applications using RTP by combining all RTP media streams in a single RTP session. Using a single RTP session also effects the possibility for differentiated treatment of media flows. This is further discussed in Section 12.9. WebRTC implementations of RTP are REQUIRED to support transport of all RTP media streams, independent of media type, in a single RTP session according to [I-D.ietf-avtcore-multi-media-rtp-session]. If such RTP session set-up is to be used, this MUST be negotiated during the signalling phase [I-D.ietf-mmusic-sdp-bundle-negotiation].

Support for multiple RTP sessions over a single UDP flow as defined by [I-D.westerlund-avtcore-transport-multiplexing] is RECOMMENDED/OPTIONAL. If multiple RTP sessions are to be multiplexed onto a single UDP flow, this MUST be negotiated during the signalling phase.

(tbd: No consensus on the level of support of Multiple RTP sessions over a single UDP flow.)

Further discussion about when different RTP session structures and multiplexing methods are suitable can be found in the memo on Guidelines for using the Multiplexing Features of RTP [I-D.westerlund-avtcore-multiplex-architecture].

#### 4.5. RTP and RTCP Multiplexing

Historically, RTP and RTCP have been run on separate transport layer addresses (e.g., two UDP ports for each RTP session, one port for RTP and one port for RTCP). With the increased use of Network Address/Port Translation (NAPT) this has become problematic, since maintaining multiple NAT bindings can be costly. It also complicates firewall administration, since multiple ports need to be opened to allow RTP traffic. To reduce these costs and session set-up times, support for multiplexing RTP data packets and RTCP control packets on a single port for each RTP session is REQUIRED, as specified in [RFC5761]. For backwards compatibility, implementations are also REQUIRED to support sending of RTP and RTCP to separate destination ports.

Note that the use of RTP and RTCP multiplexed onto a single transport port ensures that there is occasional traffic sent on that port, even if there is no active media traffic. This can be useful to keep NAT bindings alive, and is the recommend method for application level keep-alives of RTP sessions [RFC6263].

#### 4.6. Reduced Size RTCP

RTCP packets are usually sent as compound RTCP packets, and [RFC3550] requires that those compound packets start with an Sender Report (SR) or Receiver Report (RR) packet. When using frequent RTCP feedback messages under the RTP/AVPF Profile [RFC4585] these statistics are not needed in every packet, and unnecessarily increase the mean RTCP packet size. This can limit the frequency at which RTCP packets can be sent within the RTCP bandwidth share.

To avoid this problem, [RFC5506] specifies how to reduce the mean RTCP message size and allow for more frequent feedback. Frequent feedback, in turn, is essential to make real-time applications quickly aware of changing network conditions, and to allow them to adapt their transmission and encoding behaviour. Support for sending RTCP feedback packets as [RFC5506] non-compound packets is REQUIRED, but MUST be negotiated using the signalling channel before use. For backwards compatibility, implementations are also REQUIRED to support the use of compound RTCP feedback packets if the remote endpoint does not agree to the use of non-compound RTCP in the signalling exchange.

#### 4.7. Symmetric RTP/RTCP

To ease traversal of NAT and firewall devices, implementations are REQUIRED to implement and use Symmetric RTP [RFC4961]. This requires that the IP address and port used for sending and receiving RTP and RTCP packets are identical. The reasons for using symmetric RTP is primarily to avoid issues with NAT and Firewalls by ensuring that the flow is actually bi-directional and thus kept alive and registered as flow the intended recipient actually wants. In addition, it saves resources, specifically ports at the end-points, but also in the network as NAT mappings or firewall state is not unnecessary bloated. Also the amount of QoS state is reduced.

#### 4.8. Choice of RTP Synchronisation Source (SSRC)

Implementations are REQUIRED to support signalled RTP SSRC values, using the "a=ssrc:" SDP attribute defined in Sections 4.1 and 5 of [RFC5576], and MUST also support the "previous-ssrc" source attribute defined in Section 6.2 of [RFC5576]. Other attributes defined in [RFC5576] MAY be supported.

Use of the "a=ssrc:" attribute is OPTIONAL. Implementations MUST support random SSRC assignment, and MUST support SSRC collision detection and resolution, both according to [RFC3550].

#### 4.9. Generation of the RTCP Canonical Name (CNAME)

The RTCP Canonical Name (CNAME) provides a persistent transport-level identifier for an RTP endpoint. While the Synchronisation Source (SSRC) identifier for an RTP endpoint can change if a collision is detected, or when the RTP application is restarted, its RTCP CNAME is meant to stay unchanged, so that RTP endpoints can be uniquely identified and associated with their RTP media streams within a set of related RTP sessions. For proper functionality, each RTP endpoint needs to have a unique RTCP CNAME value.

The RTP specification [RFC3550] includes guidelines for choosing a unique RTP CNAME, but these are not sufficient in the presence of NAT devices. In addition, long-term persistent identifiers can be problematic from a privacy viewpoint. Accordingly, support for generating a short-term persistent RTCP CNAMEs following [I-D.ietf-avtcore-6222bis] is RECOMMENDED.

An WebRTC end-point MUST support reception of any CNAME that matches the syntax limitations specified by the RTP specification [RFC3550] and cannot assume that any CNAME will be chosen according to the form suggested above.

### 5. WebRTC Use of RTP: Extensions

There are a number of RTP extensions that are either needed to obtain full functionality, or extremely useful to improve on the baseline performance, in the WebRTC application context. One set of these extensions is related to conferencing, while others are more generic in nature. The following subsections describe the various RTP extensions mandated or suggested for use within the WebRTC context.

#### 5.1. Conferencing Extensions

RTP is inherently a group communication protocol. Groups can be implemented using a centralised server, multi-unicast, or using IP multicast. While IP multicast was popular in early deployments, in today's practice, overlay-based conferencing dominates, typically using one or more central servers to connect endpoints in a star or flat tree topology. These central servers can be implemented in a number of ways as discussed in Appendix A, and in the memo on RTP Topologies [I-D.westerlund-avtcore-rtp-topologies-update].



As discussed in Section 3.7 of [I-D.westerlund-avtcore-rtp-topologies-update], the use of a video switching MCU makes the use of RTCP for congestion control, or any type of quality reports, very problematic. Also, as discussed in section 3.8 of [I-D.westerlund-avtcore-rtp-topologies-update], the use of a content modifying MCU with RTCP termination breaks RTP loop detection and removes the ability for receivers to identify active senders. RTP Transport Translators (Topo-Translator) are not of immediate interest to WebRTC, although the main difference compared to point to point is the possibility of seeing multiple different transport paths in any RTCP feedback. Accordingly, only Point to Point (Topo-Point-to-Point), Multiple concurrent Point to Point (Mesh) and RTP Mixers (Topo-Mixer) topologies are needed to achieve the use-cases to be supported in WebRTC initially. These RECOMMENDED topologies are expected to be supported by all WebRTC end-points (these topologies require no special RTP-layer support in the end-point if the RTP features mandated in this memo are implemented).

The RTP extensions described below to be used with centralised conferencing -- where one RTP Mixer (e.g., a conference bridge) receives a participant's RTP media streams and distributes them to the other participants -- are not necessary for interoperability; an RTP endpoint that does not implement these extensions will work correctly, but might offer poor performance. Support for the listed extensions will greatly improve the quality of experience and, to provide a reasonable baseline quality, some these extensions are mandatory to be supported by WebRTC end-points.

The RTCP conferencing extensions are defined in Extended RTP Profile for Real-time Transport Control Protocol (RTCP)-Based Feedback (RTP/AVPF) [RFC4585] and the "Codec Control Messages in the RTP Audio-Visual Profile with Feedback (AVPF)" (CCM) [RFC5104] and are fully usable by the Secure variant of this profile (RTP/SAVPF) [RFC5124].

#### 5.1.1. Full Intra Request (FIR)

The Full Intra Request is defined in Sections 3.5.1 and 4.3.1 of the Codec Control Messages [RFC5104]. This message is used to make the mixer request a new Intra picture from a participant in the session. This is used when switching between sources to ensure that the receivers can decode the video or other predictive media encoding with long prediction chains. It is REQUIRED that WebRTC senders understand the react to this feedback message since it greatly improves the user experience when using centralised mixer-based conferencing; support for sending the FIR message is OPTIONAL.

#### 5.1.2. Picture Loss Indication (PLI)

The Picture Loss Indication is defined in Section 6.3.1 of the RTP/AVPF profile [RFC4585]. It is used by a receiver to tell the sending encoder that it lost the decoder context and would like to have it repaired somehow. This is semantically different from the Full Intra Request above as there could be multiple ways to fulfil the request. It is REQUIRED that WebRTC senders understand and react to this feedback message as a loss tolerance mechanism; receivers MAY send PLI messages.

#### 5.1.3. Slice Loss Indication (SLI)

The Slice Loss Indicator is defined in Section 6.3.2 of the RTP/AVPF profile [RFC4585]. It is used by a receiver to tell the encoder that it has detected the loss or corruption of one or more consecutive macro blocks, and would like to have these repaired somehow. The use of this feedback message is OPTIONAL as a loss tolerance mechanism.

#### 5.1.4. Reference Picture Selection Indication (RPSI)

Reference Picture Selection Indication (RPSI) is defined in Section 6.3.3 of the RTP/AVPF profile [RFC4585]. Some video coding standards allow the use of older reference pictures than the most recent one for predictive coding. If such a codec is in used, and if the encoder has learned about a loss of encoder-decoder synchronisation, a known-as-correct reference picture can be used for future coding. The RPSI message allows this to be signalled. Support for RPSI messages is OPTIONAL.

#### 5.1.5. Temporal-Spatial Trade-off Request (TSTR)

The temporal-spatial trade-off request and notification are defined in Sections 3.5.2 and 4.3.2 of [RFC5104]. This request can be used to ask the video encoder to change the trade-off it makes between temporal and spatial resolution, for example to prefer high spatial image quality but low frame rate. Support for TSTR requests and notifications is OPTIONAL.

#### 5.1.6. Temporary Maximum Media Stream Bit Rate Request (TMMBR)

This feedback message is defined in Sections 3.5.4 and 4.2.1 of the Codec Control Messages [RFC5104]. This message and its notification message are used by a media receiver to inform the sending party that there is a current limitation on the amount of bandwidth available to this receiver. This can be various reasons for this: for example, an RTP mixer can use this message to limit the media rate of the sender being forwarded by the mixer (without doing media transcoding) to fit

the bottlenecks existing towards the other session participants. It is REQUIRED that this feedback message is supported. WebRTC senders are REQUIRED to implement support for TMMBR messages, and MUST follow bandwidth limitations set by a TMMBR message received for their SSRC. The sending of TMMBR requests is OPTIONAL.

## 5.2. Header Extensions

The RTP specification [RFC3550] provides the capability to include RTP header extensions containing in-band data, but the format and semantics of the extensions are poorly specified. The use of header extensions is OPTIONAL in the WebRTC context, but if they are used, they MUST be formatted and signalled following the general mechanism for RTP header extensions defined in [RFC5285], since this gives well-defined semantics to RTP header extensions.

As noted in [RFC5285], the requirement from the RTP specification that header extensions are "designed so that the header extension may be ignored" [RFC3550] stands. To be specific, header extensions MUST only be used for data that can safely be ignored by the recipient without affecting interoperability, and MUST NOT be used when the presence of the extension has changed the form or nature of the rest of the packet in a way that is not compatible with the way the stream is signalled (e.g., as defined by the payload type). Valid examples might include metadata that is additional to the usual RTP information.

### 5.2.1. Rapid Synchronisation

Many RTP sessions require synchronisation between audio, video, and other content. This synchronisation is performed by receivers, using information contained in RTCP SR packets, as described in the RTP specification [RFC3550]. This basic mechanism can be slow, however, so it is RECOMMENDED that the rapid RTP synchronisation extensions described in [RFC6051] be implemented. The rapid synchronisation extensions use the general RTP header extension mechanism [RFC5285], which requires signalling, but are otherwise backwards compatible.

### 5.2.2. Client-to-Mixer Audio Level

The Client to Mixer Audio Level extension [RFC6464] is an RTP header extension used by a client to inform a mixer about the level of audio activity in the packet to which the header is attached. This enables a central node to make mixing or selection decisions without decoding or detailed inspection of the payload, reducing the complexity in some types of central RTP nodes. It can also save decoding resources in receivers, which can choose to decode only the most relevant RTP media streams based on audio activity levels.

The Client-to-Mixer Audio Level [RFC6464] extension is RECOMMENDED to be implemented. If it is implemented, it is REQUIRED that the header extensions are encrypted according to [I-D.ietf-avtcore-srtp-encrypted-header-ext] since the information contained in these header extensions can be considered sensitive.

#### 5.2.3. Mixer-to-Client Audio Level

The Mixer to Client Audio Level header extension [RFC6465] provides the client with the audio level of the different sources mixed into a common mix by a RTP mixer. This enables a user interface to indicate the relative activity level of each session participant, rather than just being included or not based on the CSRC field. This is a pure optimisations of non critical functions, and is hence OPTIONAL to implement. If it is implemented, it is REQUIRED that the header extensions are encrypted according to [I-D.ietf-avtcore-srtp-encrypted-header-ext] since the information contained in these header extensions can be considered sensitive.

### 6. WebRTC Use of RTP: Improving Transport Robustness

There are some tools that can make RTP flows robust against Packet loss and reduce the impact on media quality. However, they all add extra bits compared to a non-robust stream. These extra bits need to be considered, and the aggregate bit-rate MUST be rate-controlled. Thus, improving robustness might require a lower base encoding quality, but has the potential to deliver that quality with fewer errors. The mechanisms described in the following sub-sections can be used to improve tolerance to packet loss.

#### 6.1. Negative Acknowledgements and RTP Retransmission

As a consequence of supporting the RTP/SAVPF profile, implementations will support negative acknowledgements (NACKs) for RTP data packets [RFC4585]. This feedback can be used to inform a sender of the loss of particular RTP packets, subject to the capacity limitations of the RTCP feedback channel. A sender can use this information to optimise the user experience by adapting the media encoding to compensate for known lost packets, for example.

Senders are REQUIRED to understand the Generic NACK message defined in Section 6.2.1 of [RFC4585], but MAY choose to ignore this feedback (following Section 4.2 of [RFC4585]). Receivers MAY send NACKs for missing RTP packets; [RFC4585] provides some guidelines on when to send NACKs. It is not expected that a receiver will send a NACK for every lost RTP packet, rather it needs to consider the cost of sending NACK feedback, and the importance of the lost packet, to make

an informed decision on whether it is worth telling the sender about a packet loss event.

The RTP Retransmission Payload Format [RFC4588] offers the ability to retransmit lost packets based on NACK feedback. Retransmission needs to be used with care in interactive real-time applications to ensure that the retransmitted packet arrives in time to be useful, but can be effective in environments with relatively low network RTT (an RTP sender can estimate the RTT to the receivers using the information in RTCP SR and RR packets). The use of retransmissions can also increase the forward RTP bandwidth, and can potentially worsen the problem if the packet loss was caused by network congestion. We note, however, that retransmission of an important lost packet to repair decoder state can have lower cost than sending a full intra frame. It is not appropriate to blindly retransmit RTP packets in response to a NACK. The importance of lost packets and the likelihood of them arriving in time to be useful needs to be considered before RTP retransmission is used.

Receivers are REQUIRED to implement support for RTP retransmission packets [RFC4588]. Senders MAY send RTP retransmission packets in response to NACKs if the RTP retransmission payload format has been negotiated for the session, and if the sender believes it is useful to send a retransmission of the packet(s) referenced in the NACK. An RTP sender is not expected to retransmit every NACKed packet.

## 6.2. Forward Error Correction (FEC)

The use of Forward Error Correction (FEC) can provide an effective protection against some degree of packet loss, at the cost of steady bandwidth overhead. There are several FEC schemes that are defined for use with RTP. Some of these schemes are specific to a particular RTP payload format, others operate across RTP packets and can be used with any payload format. It needs to be noted that using redundant encoding or FEC will lead to increased play out delay, which needs to be considered when choosing the redundancy or FEC formats and their respective parameters.

If an RTP payload format negotiated for use in a WebRTC session supports redundant transmission or FEC as a standard feature of that payload format, then that support MAY be used in the WebRTC session, subject to any appropriate signalling.

There are several block-based FEC schemes that are designed for use with RTP independent of the chosen RTP payload format. At the time of this writing there is no consensus on which, if any, of these FEC schemes is appropriate for use in the WebRTC context. Accordingly, this memo makes no recommendation on the choice of block-based FEC

for WebRTC use.

## 7. WebRTC Use of RTP: Rate Control and Media Adaptation

WebRTC will be used in heterogeneous network environments using a variety set of link technologies, including both wired and wireless links, to interconnect potentially large groups of users around the world. As a result, the network paths between users can have widely varying one-way delays, available bit-rates, load levels, and traffic mixtures. Individual end-points can open one or more RTP sessions to each participant in a WebRTC conference, and there can be several participants. Each of these RTP sessions can contain different types of media, and the type of media, bit rate, and number of flows can be highly asymmetric. Non-RTP traffic can share the network paths RTP flows. Since the network environment is not predictable or stable, WebRTC endpoints MUST ensure that the RTP traffic they generate can adapt to match changes in the available network capacity.

The quality of experience for users of WebRTC implementation is very dependent on effective adaptation of the media to the limitations of the network. End-points have to be designed so they do not transmit significantly more data than the network path can support, except for very short time periods, otherwise high levels of network packet loss or delay spikes will occur, causing media quality degradation. The limiting factor on the capacity of the network path might be the link bandwidth, or it might be competition with other traffic on the link (this can be non-WebRTC traffic, traffic due to other WebRTC flows, or even competition with other WebRTC flows in the same session).

An effective media congestion control algorithm is therefore an essential part of the WebRTC framework. However, at the time of this writing, there is no standard congestion control algorithm that can be used for interactive media applications such as WebRTC flows. Some requirements for congestion control algorithms for WebRTC sessions are discussed in [I-D.jesup-rtp-congestion-reqs], and it is expected that a future version of this memo will mandate the use of a congestion control algorithm that satisfies these requirements.

### 7.1. Boundary Conditions and Circuit Breakers

In the absence of a concrete congestion control algorithm, all WebRTC implementations MUST implement the RTP circuit breaker algorithm that is in described [I-D.ietf-avtcore-rtp-circuit-breakers]. The circuit breaker defines a conservative boundary condition for safe operation, chosen such that applications that trigger the circuit breaker will almost certainly be causing severe network congestion. Any future RTP congestion control algorithms are expected to operate within the

envelope allowed by the circuit breaker.

The session establishment signalling will also necessarily establish boundaries to which the media bit-rate will conform. The choice of media codecs provides upper- and lower-bounds on the supported bit-rates that the application can utilise to provide useful quality, and the packetization choices that exist. In addition, the signalling channel can establish maximum media bit-rate boundaries using the SDP "b=AS:" or "b=CT:" lines, and the RTP/AVPF Temporary Maximum Media Stream Bit Rate (TMMBR) Requests (see Section 5.1.6 of this memo). The combination of media codec choice and signalled bandwidth limits SHOULD be used to limit traffic based on known bandwidth limitations, for example the capacity of the edge links, to the extent possible.

## 7.2. RTCP Extensions for Congestion Control

As described in Section 5.1.6, the Temporary Maximum Media Stream Bit Rate (TMMBR) request is supported by WebRTC senders. This request can be used by a media receiver to impose limitations on the media sender based on the receiver's determined bit-rate limitations, to provide a limited means of congestion control.

(tbd: What other RTP/RTCP extensions are needed?)

With proprietary congestion control algorithms issues can arise when different algorithms and implementations interact in a communication session. If the different implementations have made different choices in regards to the type of adaptation, for example one sender based, and one receiver based, then one could end up in situation where one direction is dual controlled, when the other direction is not controlled.

(tbd: How to ensure that both paths and sender and receiver based solutions can interact)

## 7.3. RTCP Limitations for Congestion Control

Experience with the congestion control algorithms of TCP [RFC5681], TFRC [RFC5348], and DCCP [RFC4341], [RFC4342], [RFC4828], has shown that feedback on packet arrivals needs to be sent roughly once per round trip time. We note that the real-time media traffic might not have to adapt to changing path conditions as rapidly as needed for the elastic applications TCP was designed for, but frequent feedback is still needed to allow the congestion control algorithm to track the path dynamics.

The total RTCP bandwidth is limited in its transmission rate to a fraction of the RTP traffic (by default 5%). RTCP packets are larger

than, e.g., TCP ACKs (even when non-compound RTCP packets are used). The RTP media stream bit rate thus limits the maximum feedback rate as a function of the mean RTCP packet size.

Interactive communication might not be able to afford waiting for packet losses to occur to indicate congestion, because an increase in play out delay due to queuing (most prominent in wireless networks) can easily lead to packets being dropped due to late arrival at the receiver. Therefore, more sophisticated cues might need to be reported -- to be defined in a suitable congestion control framework as noted above -- which, in turn, increase the report size again. For example, different RTCP XR report blocks (jointly) provide the necessary details to implement a variety of congestion control algorithms, but the (compound) report size grows quickly.

In group communication, the share of RTCP bandwidth needs to be shared by all group members, reducing the capacity and thus the reporting frequency per node.

Example: assuming 512 kbit/s video yields 3200 bytes/s RTCP bandwidth, split across two entities in a point-to-point session. An endpoint could thus send a report of 100 bytes about every 70ms or for every other frame in a 30 fps video.

#### 7.4. Congestion Control Interoperability With Legacy Systems

There are legacy implementations that do not implement RTCP, and hence do not provide any congestion feedback. Congestion control cannot be performed with these end-points. WebRTC implementations that need to interwork with such end-points MUST limit their transmission to a low rate, equivalent to a VoIP call using a low bandwidth codec, that is unlikely to cause any significant congestion.

When interworking with legacy implementations that support RTCP using the RTP/AVP profile [RFC3551], congestion feedback is provided in RTCP RR packets every few seconds. Implementations that have to interwork with such end-points MUST ensure that they keep within the RTP circuit breaker [I-D.ietf-avtcore-rtp-circuit-breakers] constraints to limit the congestion they can cause.

If a legacy end-point supports RTP/AVPF, this enables negotiation of important parameters for frequent reporting, such as the "trr-int" parameter, and the possibility that the end-point supports some useful feedback format for congestion control purpose such as TMMBR [RFC5104]. Implementations that have to interwork with such end-points MUST ensure that they stay within the RTP circuit breaker [I-D.ietf-avtcore-rtp-circuit-breakers] constraints to limit the



congestion they can cause, but might find that they can achieve better congestion response depending on the amount of feedback that is available.

#### 8. WebRTC Use of RTP: Performance Monitoring

RTCP does contains a basic set of RTP flow monitoring metrics like packet loss and jitter. There are a number of extensions that could be included in the set to be supported. However, in most cases which RTP monitoring that is needed depends on the application, which makes it difficult to select which to include when the set of applications is very large.

Exposing some metrics in the WebRTC API needs to be considered allowing the application to gather the measurements of interest. However, security implications for the different data sets exposed will need to be considered in this.

(tbd: If any RTCP XR metrics need to be added is still an open question, but possible to extend at a later stage)

#### 9. WebRTC Use of RTP: Future Extensions

It is possible that the core set of RTP protocols and RTP extensions specified in this memo will prove insufficient for the future needs of WebRTC applications. In this case, future updates to this memo MUST be made following the Guidelines for Writers of RTP Payload Format Specifications [RFC2736] and Guidelines for Extending the RTP Control Protocol [RFC5968], and SHOULD take into account any future guidelines for extending RTP and related protocols that have been developed.

Authors of future extensions are urged to consider the wide range of environments in which RTP is used when recommending extensions, since extensions that are applicable in some scenarios can be problematic in others. Where possible, the WebRTC framework will adopt RTP extensions that are of general utility, to enable easy implementation of a gateway to other applications using RTP, rather than adopt mechanisms that are narrowly targeted at specific WebRTC use cases.

#### 10. Signalling Considerations

RTP is built with the assumption of an external signalling channel that can be used to configure the RTP sessions and their features. The basic configuration of an RTP session consists of the following

parameters:

RTP Profile: The name of the RTP profile to be used in session. The RTP/AVP [RFC3551] and RTP/AVPF [RFC4585] profiles can interoperate on basic level, as can their secure variants RTP/SAVP [RFC3711] and RTP/SAVPF [RFC5124]. The secure variants of the profiles do not directly interoperate with the non-secure variants, due to the presence of additional header fields in addition to any cryptographic transformation of the packet content. As WebRTC requires the usage of the RTP/SAVPF profile this can be inferred as there is only a single profile, but in SDP this is still information that has to be signalled. Interworking functions might transform this into RTP/SAVP for a legacy use case by indicating to the WebRTC end-point a RTP/SAVPF end-point and limiting the usage of the a=rtcp attribute to indicate a trr-int value of 4 seconds.

Transport Information: Source and destination IP address(s) and ports for RTP and RTCP MUST be signalled for each RTP session. In WebRTC these transport addresses will be provided by ICE that signals candidates and arrives at nominated candidate address pairs. If RTP and RTCP multiplexing [RFC5761] is to be used, such that a single port is used for RTP and RTCP flows, this MUST be signalled (see Section 4.5). If several RTP sessions are to be multiplexed onto a single transport layer flow, this MUST also be signalled (see Section 4.4).

RTP Payload Types, media formats, and media format parameters: The mapping between media type names (and hence the RTP payload formats to be used) and the RTP payload type numbers MUST be signalled. Each media type MAY also have a number of media type parameters that MUST also be signalled to configure the codec and RTP payload format (the "a=fmtp:" line from SDP).

RTP Extensions: The RTP extensions to be used SHOULD be agreed upon, including any parameters for each respective extension. At the very least, this will help avoiding using bandwidth for features that the other end-point will ignore. But for certain mechanisms there is requirement for this to happen as interoperability failure otherwise happens.

RTCP Bandwidth: Support for exchanging RTCP Bandwidth values to the end-points will be necessary. This SHALL be done as described in "Session Description Protocol (SDP) Bandwidth Modifiers for RTP Control Protocol (RTCP) Bandwidth" [RFC3556], or something semantically equivalent. This also ensures that the end-points have a common view of the RTCP bandwidth, this is important as too different view of the bandwidths can lead to failure to

interoperate.

These parameters are often expressed in SDP messages conveyed within an offer/answer exchange. RTP does not depend on SDP or on the offer/answer model, but does require all the necessary parameters to be agreed upon, and provided to the RTP implementation. We note that in the WebRTC context it will depend on the signalling model and API how these parameters need to be configured but they will be need to either set in the API or explicitly signalled between the peers.

## 11. WebRTC API Considerations

The WebRTC API and its media function have the concept of a WebRTC `MediaStream` that consists of zero or more tracks. A track is an individual stream of media from any type of media source like a microphone or a camera, but also conceptual sources, like a audio mix or a video composition, are possible. The tracks within a WebRTC `MediaStream` are expected to be synchronized.

A track correspond to the media received with one particular SSRC. There might be additional SSRCS associated with that SSRC, like for RTP retransmission or Forward Error Correction. However, one SSRC will identify an RTP media stream and its timing.

As a result, a WebRTC `MediaStream` is a collection of SSRCS carrying the different media included in the synchronised aggregate. Therefore, also the synchronization state associated with the included SSRCS are part of concept. It is important to consider that there can be multiple different WebRTC `MediaStreams` containing a given Track (SSRC). To avoid unnecessary duplication of media at the transport level in such cases, a need arises for a binding defining which WebRTC `MediaStreams` a given SSRC is associated with at the signalling level.

A proposal for how the binding between WebRTC `MediaStreams` and SSRC can be done is specified in "Cross Session Stream Identification in the Session Description Protocol" [I-D.alvestrand-rtcweb-msid].

(tbd: This text needs to be improved and achieved consensus on. Interim meeting in June 2012 shows large differences in opinions.)

(tbd: It is an open question whether these considerations are best discussed in this draft, in the W3C WebRTC API spec, or elsewhere.

## 12. RTP Implementation Considerations

The following discussion provides some guidance on the implementation of the RTP features described in this memo. The focus is on a WebRTC end-point implementation perspective, and while some mention is made of the behaviour of middleboxes, that is not the focus of this memo.

### 12.1. RTP Sessions and PeerConnections

An RTP session is an association among RTP nodes, which have a single shared SSRC space. An RTP session can include a large number of end-points and nodes, each sourcing, sinking, manipulating, or reporting on the RTP media streams being sent within the RTP session.

A PeerConnection is a point-to-point association between an end-point and some other peer node. That peer node can be either an end-point or a centralized processing node of some type. Hence, an RTP session can terminate immediately at the far end of a PeerConnection, or it might continue as further discussed below for multiparty sessions (Section 12.3) and sessions with multiple end points (Section 12.7).

A PeerConnection can contain one or more RTP sessions, depending on how it is set up, and how many UDP flows it uses. A common usage has been to have one RTP session per media type, e.g. one for audio and one for video, each sent over a different UDP flow. However, the default usage in WebRTC will be to use one RTP session for all media types, with RTP and RTCP multiplexing (Section 4.5) also mandated. This RTP session then uses only one UDP flow. However, for legacy interworking and flow-based network prioritization (Section 12.9), a WebRTC end-point needs to support a mode of operation where one RTP session per media type is used. Currently, each RTP session has to use its own UDP flow in this case, however it might be possible to multiplex several RTP sessions over a single UDP flow, see Section 4.4.

The multi-unicast- or mesh-based multi-party topology (Figure 1) is a good example for this section as it concerns the relation between RTP sessions and PeerConnections. In this topology, each participant sends individual unicast RTP/UDP/IP flows to each of the other participants using independent PeerConnections in a full mesh. This topology has the benefit of not requiring central nodes. The downside is that it increases the used bandwidth at each sender by requiring one copy of the RTP media streams for each participant that are part of the same session beyond the sender itself. Hence, this topology is limited to scenarios with few participants unless the media is very low bandwidth.

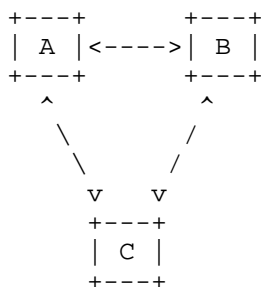


Figure 1: Multi-unicast

The multi-unicast topology could be implemented as a single RTP session, spanning multiple peer-to-peer transport layer connections, or as several pairwise RTP sessions, one between each pair of peers. To maintain a coherent mapping between the relation between RTP sessions and PeerConnections we recommend that one implements this as individual RTP sessions. The only downside is that end-point A will not learn of the quality of any transmission happening between B and C based on RTCP. This has not been seen as a significant downside as no one has yet seen a clear need for why A would need to know about the B's and C's communication. An advantage of using separate RTP sessions is that it enables using different media bit-rates to the different peers, thus not forcing B to endure the same quality reductions if there are limitations in the transport from A to C as C will.

## 12.2. Multiple Sources

A WebRTC end-point might have multiple cameras, microphones or audio inputs and thus a single end-point can source multiple RTP media streams of the same media type concurrently. Even if an end-point does not have multiple media sources of the same media type it has to support transmission using multiple SSRCs concurrently in the same RTP session. This is due to the requirement on an WebRTC end-point to support multiple media types in one RTP session. For example, one audio and one video source can result in the end-point sending with two different SSRCs in the same RTP session. As multi-party conferences are supported, as discussed below in Section 12.3, a WebRTC end-point will need to be capable of receiving, decoding and play out multiple RTP media streams of the same type concurrently.

tbd: Are any mechanism needed to signal limitations in the number of active SSRC that an end-point can handle?

### 12.3. Multiparty

There are numerous situations and clear use cases for WebRTC supporting RTP sessions supporting multi-party. This can be realized in a number of ways using a number of different implementation strategies. In the following, the focus is on the different set of WebRTC end-point requirements that arise from different sets of multi-party topologies.

The multi-unicast mesh (Figure 1)-based multi-party topology discussed above provides a non-centralized solution but can incur a heavy tax on the end-points' outgoing paths. It can also consume large amount of encoding resources if each outgoing stream is specifically encoded. If an encoding is transmitted to multiple parties, as in some implementations of the mesh case, a requirement on the end-point becomes to be able to create RTP media streams suitable for multiple destinations requirements. These requirements can both be dependent on transport path and the different end-points preferences related to play out of the media.

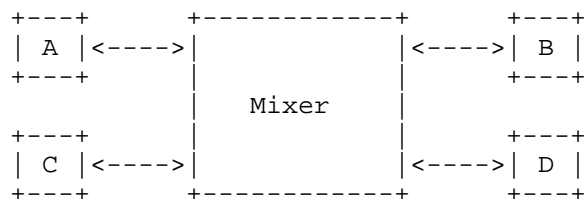


Figure 2: RTP Mixer with Only Unicast Paths

A Mixer (Figure 2) is an RTP end-point that optimizes the transmission of RTP media streams from certain perspectives, either by only sending some of the received RTP media stream to any given receiver or by providing a combined RTP media stream out of a set of contributing streams. There are various methods of implementation as discussed in Appendix A.3. A common aspect is that these central nodes can use a number of tools to control the media encoding provided by a WebRTC end-point. This includes functions like requesting breaking the encoding chain and have the encoder produce a so called Intra frame. Another is limiting the bit-rate of a given stream to better suit the mixer view of the multiple down-streams. Others are controlling the most suitable frame-rate, picture resolution, the trade-off between frame-rate and spatial quality.

A mixer gets a significant responsibility to correctly perform congestion control, source identification, manage synchronization while providing the application with suitable media optimizations.

Mixers also need to be trusted nodes when it comes to security as it manipulates either RTP or the media itself before sending it on towards the end-point(s), thus they need to be able to decrypt and then encrypt it before sending it out.

#### 12.4. SSRC Collision Detection

The RTP standard [RFC3550] requires any RTP implementation to have support for detecting and handling SSRC collisions, i.e., resolve the conflict when two different end-points use the same SSRC value. This requirement also applies to WebRTC end-points. There are several scenarios where SSRC collisions can occur.

In a point-to-point session where each SSRC is associated with either of the two end-points and where the main media carrying SSRC identifier will be announced in the signalling channel, a collision is less likely to occur due to the information about used SSRCS provided by Source-Specific SDP Attributes [RFC5576]. Still if both end-points start uses an new SSRC identifier prior to having signalled it to the peer and received acknowledgement on the signalling message, there can be collisions. The Source-Specific SDP Attributes [RFC5576] contains no mechanism to resolve SSRC collisions or reject a end-points usage of an SSRC.

There could also appear SSRC values that are not signalled. This is more likely than it appears as certain RTP functions need extra SSRCS to provide functionality related to another (the "main") SSRC, for example, SSRC multiplexed RTP retransmission [RFC4588]. In those cases, an end-point can create a new SSRC that strictly doesn't need to be announced over the signalling channel to function correctly on both RTP and PeerConnection level.

The more likely case for SSRC collision is that multiple end-points in a multiparty conference create new sources and signals those towards the central server. In cases where the SSRC/CSRC are propagated between the different end-points from the central node collisions can occur.

Another scenario is when the central node manages to connect an end-point's PeerConnection to another PeerConnection the end-point already has, thus forming a loop where the end-point will receive its own traffic. While is is clearly considered a bug, it is important that the end-point is able to recognise and handle the case when it occurs. This case becomes even more problematic when media mixers, and so on, are involved, where the stream received is a different stream but still contains this client's input.

These SSRC/CSRC collisions can only be handled on RTP level as long

as the same RTP session is extended across multiple PeerConnections by a RTP middlebox. To resolve the more generic case where multiple PeerConnections are interconnected, then identification of the media source(s) part of a MediaStreamTrack being propagated across multiple interconnected PeerConnection needs to be preserved across these interconnections.

#### 12.5. Contributing Sources and the CSRC List

RTP allows a mixer, or other RTP-layer middlebox, to combine media flows from multiple sources to form a new media flow. The RTP data packets in that new flow will include a Contributing Source (CSRC) list, indicating which original SSRCs contributed to the combined packet. As described in Section 4.1, implementations need to support reception of RTP data packets containing a CSRC list and RTCP packets that relate to sources present in the CSRC list.

The CSRC list can change on a packet-by-packet basis, depending on the mixing operation being performed. Knowledge of what sources contributed to a particular RTP packet can be important if the user interface indicates which participants are active in the session. Changes in the CSRC list included in packets needs to be exposed to the WebRTC application using some API, if the application is to be able to track changes in session participation. It is desirable to map CSRC values back into WebRTC MediaStream identities as they cross this API, to avoid exposing the SSRC/CSRC name space to JavaScript applications.

If the mixer-to-client audio level extension [RFC6465] is being used in the session (see Section 5.2.3), the information in the CSRC list is augmented by audio level information for each contributing source. This information can usefully be exposed in the user interface.

This memo does not require implementations to be able to add a CSRC list to outgoing RTP packets. It is expected that the any CSRC list will be added by a mixer or other middlebox that performs in-network processing of RTP streams. If there is a desire to allow end-system mixing, the requirement in Section 4.1 will need to be updated to support setting the CSRC list in outgoing RTP data packets.

#### 12.6. Media Synchronization

When an end-point sends media from more than one media source, it needs to consider if (and which of) these media sources are to be synchronized. In RTP/RTCP, synchronisation is provided by having a set of RTP media streams be indicated as coming from the same synchronisation context and logical end-point by using the same CNAME identifier.



The next provision is that the internal clocks of all media sources, i.e., what drives the RTP timestamp, can be correlated to a system clock that is provided in RTCP Sender Reports encoded in an NTP format. By correlating all RTP timestamps to a common system clock for all sources, the timing relation of the different RTP media streams, also across multiple RTP sessions can be derived at the receiver and, if desired, the streams can be synchronized. The requirement is for the media sender to provide the correlation information; it is up to the receiver to use it or not.

#### 12.7. Multiple RTP End-points

Some usages of RTP beyond the recommend topologies result in that an WebRTC end-point sending media in an RTP session out over a single PeerConnection will receive receiver reports from multiple RTP receivers. Note that receiving multiple receiver reports is expected because any RTP node that has multiple SSRCs has to report to the media sender. The difference here is that they are multiple nodes, and thus will likely have different path characteristics.

RTP Mixers can create a situation where an end-point experiences a situation in-between a session with only two end-points and multiple end-points. Mixers are expected to not forward RTCP reports regarding RTP media streams across themselves. This is due to the difference in the RTP media streams provided to the different end-points. The original media source lacks information about a mixer's manipulations prior to sending it the different receivers. This scenario also results in that an end-point's feedback or requests goes to the mixer. When the mixer can't act on this by itself, it is forced to go to the original media source to fulfil the receivers request. This will not necessarily be explicitly visible any RTP and RTCP traffic, but the interactions and the time to complete them will indicate such dependencies.

The topologies in which an end-point receives receiver reports from multiple other end-points are the centralized relay, multicast and an end-point forwarding an RTP media stream. Having multiple RTP nodes receive an RTP flow and send reports and feedback about it has several impacts. As previously discussed (Section 12.3) any codec control and rate control needs to be capable of merging the requirements and preferences to provide a single best encoding according to the situation RTP media stream. Specifically, when it comes to congestion control it needs to be capable of identifying the different end-points to form independent congestion state information for each different path.

Providing source authentication in multi-party scenarios is a challenge. In the mixer-based topologies, end-points source

authentication is based on, firstly, verifying that media comes from the mixer by cryptographic verification and, secondly, trust in the mixer to correctly identify any source towards the end-point. In RTP sessions where multiple end-points are directly visible to an end-point, all end-points will have knowledge about each others' master keys, and can thus inject packets claimed to come from another end-point in the session. Any node performing relay can perform non-cryptographic mitigation by preventing forwarding of packets that have SSRC fields that came from other end-points before. For cryptographic verification of the source SRTP would require additional security mechanisms, like TESLA for SRTP [RFC4383].

#### 12.8. Simulcast

This section discusses simulcast in the meaning of providing a node, for example a Mixer, with multiple different encoded versions of the same media source. In the WebRTC context, this could be accomplished in two ways. One is to establish multiple PeerConnection all being feed the same set of WebRTC MediaStreams. Another method is to use multiple WebRTC MediaStreams that are differently configured when it comes to the media parameters. This would result in that multiple different RTP Media Streams (SSRCs) being in used with different encoding based on the same media source (camera, microphone).

When intending to use simulcast it is important that this is made explicit so that the end-points don't automatically try to optimize away the different encodings and provide a single common version. Thus, some explicit indications that the intent really is to have different media encodings is likely needed. It is to be noted that it might be a central node, rather than an WebRTC end-point that would benefit from receiving simulcast media sources.

tbd: How to perform simulcast needs to be determined and the appropriate API or signalling for its usage needs to be defined.

#### 12.9. Differentiated Treatment of Flows

There are use cases for differentiated treatment of RTP media streams. Such differentiation can happen at several places in the system. First of all is the prioritization within the end-point sending the media, which controls, both which RTP media streams that will be sent, and their allocation of bit-rate out of the current available aggregate as determined by the congestion control.

It is expected that the WebRTC API will allow the application to indicate relative priorities for different MediaStreamTracks. These priorities can then be used to influence the local RTP processing, especially when it comes to congestion control response in how to

divide the available bandwidth between the RTP flows. Any changes in relative priority will also need to be considered for RTP flows that are associated with the main RTP flows, such as RTP retransmission streams and FEC. The importance of such associated RTP traffic flows is dependent on the media type and codec used, in regards to how robust that codec is to packet loss. However, a default policy might be to use the same priority for associated RTP flows as for the primary RTP flow.

Secondly, the network can prioritize packet flows, including RTP media streams. Typically, differential treatment includes two steps, the first being identifying whether an IP packet belongs to a class that has to be treated differently, the second the actual mechanism to prioritize packets. This is done according to three methods;

**DiffServ:** The end-point marks a packet with a DiffServ code point to indicate to the network that the packet belongs to a particular class.

**Flow based:** Packets that need to be given a particular treatment are identified using a combination of IP and port address.

**Deep Packet Inspection:** A network classifier (DPI) inspects the packet and tries to determine if the packet represents a particular application and type that is to be prioritized.

Flow-based differentiation will provide the same treatment to all packets within a flow, i.e., relative prioritization is not possible. Moreover, if the resources are limited it might not be possible to provide differential treatment compared to best-effort for all the flows in a WebRTC application. When flow-based differentiation is available the WebRTC application needs to know about it so that it can provide the separation of the RTP media streams onto different UDP flows to enable a more granular usage of flow based differentiation. That way at least providing different prioritization of audio and video if desired by application.

DiffServ assumes that either the end-point or a classifier can mark the packets with an appropriate DSCP so that the packets are treated according to that marking. If the end-point is to mark the traffic two requirements arise in the WebRTC context: 1) The WebRTC application or browser has to know which DSCP to use and that it can use them on some set of RTP media streams. 2) The information needs to be propagated to the operating system when transmitting the packet. These issues are discussed in DSCP and other packet markings for RTCWeb QoS [I-D.ietf-rtcweb-qos].

For packet based marking schemes it would be possible in the context

to mark individual RTP packets differently based on the relative priority of the RTP payload. For example video codecs that has I,P and B pictures could prioritise any payloads carrying only B frames less, as these are less damaging to loose. But as default policy all RTP packets related to a media stream ought to be provided with the same prioritization.

It is also important to consider how RTCP packets associated with a particular RTP media flow need to be marked. RTCP compound packets with Sender Reports (SR), ought to be marked with the same priority as the RTP media flow itself, so the RTCP-based round-trip time (RTT) measurements are done using the same flow priority as the media flow experiences. RTCP compound packets containing RR packet ought to be sent with the priority used by the majority of the RTP media flows reported on. RTCP packets containing time-critical feedback packets can use higher priority to improve the timeliness and likelihood of delivery of such feedback.

### 13. Open Issues

This section contains a summary of the open issues or to be done things noted in the document:

1. Need to add references to the RTP payload format for the Video Codec chosen in Section 4.3.
2. The methods and solutions for RTP multiplexing over a single transport is not yet finalized in Section 4.4.
3. RTP congestion control algorithms will probably require some feedback information to be conveyed in RTCP. Are the tools that are mandated by this memo sufficient, or do we need additional information Section 7.2?
4. RTP congestion control could be implementing using either a sender-based algorithm or a receiver-based algorithm. To ensure interoperability, does this memo need to mandate which end is in charge of congestion control for a path Section 7.2?
5. Still open if any RTCP XR performance metrics are needed, as discussed in Section 8.
6. The API mapping to RTP level concepts has to be agreed and documented in Section 11.
7. An open question if any requirements are needed to agree and limit the number of simultaneously used media sources (SSRCs)

within an RTP session. See Section 12.2.

8. The method for achieving simulcast of a media source has to be decided as discussed in Section 12.8.
9. Possible documentation of what support for differentiated treatment that are needed on RTP level as the API and the network level specification matures as discussed in Section 12.9.
10. Editing of Appendix A to remove redundancy between this and the update of RTP Topologies [I-D.westerlund-avtcore-rtp-topologies-update].

#### 14. IANA Considerations

This memo makes no request of IANA.

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

#### 15. Security Considerations

The overall security architecture for WebRTC is described in [I-D.ietf-rtcweb-security-arch], and security considerations for the WebRTC framework are described in [I-D.ietf-rtcweb-security]. These considerations apply to this memo also.

The security considerations of the RTP specification, the RTP/SAVPF profile, and the various RTP/RTCP extensions and RTP payload formats that form the complete protocol suite described in this memo apply. We do not believe there are any new security considerations resulting from the combination of these various protocol extensions.

The Extended Secure RTP Profile for Real-time Transport Control Protocol (RTCP)-Based Feedback [RFC5124] (RTP/SAVPF) provides handling of fundamental issues by offering confidentiality, integrity and partial source authentication. A mandatory to implement media security solution is (tbd).

RTCP packets convey a Canonical Name (CNAME) identifier that is used to associate media flows that need to be synchronised across related RTP sessions. Inappropriate choice of CNAME values can be a privacy concern, since long-term persistent CNAME identifiers can be used to track users across multiple WebRTC calls. Section 4.9 of this memo provides guidelines for generation of untraceable CNAME values that

alleviate this risk.

The guidelines in [RFC6562] apply when using variable bit rate (VBR) audio codecs such as Opus (see Section 4.3 for discussion of mandated audio codecs). These guidelines in [RFC6562] also apply, but are of lesser importance, when using the client-to-mixer audio level header extensions (Section 5.2.2) or the mixer-to-client audio level header extensions (Section 5.2.3).

## 16. Acknowledgements

The authors would like to thank Harald Alvestrand, Cary Bran, Charles Eckel and Cullen Jennings for valuable feedback.

## 17. References

### 17.1. Normative References

[I-D.ietf-avtcore-6222bis]

Rescorla, E. and A. Begen, "Guidelines for Choosing RTP Control Protocol (RTCP) Canonical Names (CNAMEs)", draft-ietf-avtcore-6222bis-00 (work in progress), December 2012.

[I-D.ietf-avtcore-avp-codecs]

Terriberry, T., "Update to Recommended Codecs for the AVP RTP Profile", draft-ietf-avtcore-avp-codecs-00 (work in progress), January 2013.

[I-D.ietf-avtcore-multi-media-rtp-session]

Westerlund, M., Perkins, C., and J. Lennox, "Multiple Media Types in an RTP Session", draft-ietf-avtcore-multi-media-rtp-session-01 (work in progress), October 2012.

[I-D.ietf-avtcore-rtp-circuit-breakers]

Perkins, C. and V. Singh, "Multimedia Congestion Control: Circuit Breakers for Unicast RTP Sessions", draft-ietf-avtcore-rtp-circuit-breakers-02 (work in progress), February 2013.

[I-D.ietf-avtcore-srtp-encrypted-header-ext]

Lennox, J., "Encryption of Header Extensions in the Secure Real-Time Transport Protocol (SRTP)", draft-ietf-avtcore-srtp-encrypted-header-ext-05 (work in progress), February 2013.

- [I-D.ietf-avtext-multiple-clock-rates]  
Petit-Huguenin, M. and G. Zorn, "Support for Multiple Clock Rates in an RTP Session",  
draft-ietf-avtext-multiple-clock-rates-08 (work in progress), November 2012.
- [I-D.ietf-mmusic-sdp-bundle-negotiation]  
Holmberg, C., Alvestrand, H., and C. Jennings,  
"Multiplexing Negotiation Using Session Description Protocol (SDP) Port Numbers",  
draft-ietf-mmusic-sdp-bundle-negotiation-03 (work in progress), February 2013.
- [I-D.ietf-rtcweb-audio]  
Valin, J. and C. Bran, "WebRTC Audio Codec and Processing Requirements", draft-ietf-rtcweb-audio-01 (work in progress), November 2012.
- [I-D.ietf-rtcweb-overview]  
Alvestrand, H., "Overview: Real Time Protocols for Browser-based Applications", draft-ietf-rtcweb-overview-06 (work in progress), February 2013.
- [I-D.ietf-rtcweb-security]  
Rescorla, E., "Security Considerations for RTC-Web",  
draft-ietf-rtcweb-security-04 (work in progress),  
January 2013.
- [I-D.ietf-rtcweb-security-arch]  
Rescorla, E., "RTCWEB Security Architecture",  
draft-ietf-rtcweb-security-arch-06 (work in progress),  
January 2013.
- [I-D.westerlund-avtcore-transport-multiplexing]  
Westerlund, M. and C. Perkins, "Multiple RTP Sessions on a Single Lower-Layer Transport",  
draft-westerlund-avtcore-transport-multiplexing-04 (work in progress), October 2012.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.
- [RFC2736] Handley, M. and C. Perkins, "Guidelines for Writers of RTP Payload Format Specifications", BCP 36, RFC 2736, December 1999.
- [RFC3550] Schulzrinne, H., Casner, S., Frederick, R., and V. Jacobson, "RTP: A Transport Protocol for Real-Time

Applications", STD 64, RFC 3550, July 2003.

- [RFC3551] Schulzrinne, H. and S. Casner, "RTP Profile for Audio and Video Conferences with Minimal Control", STD 65, RFC 3551, July 2003.
- [RFC3556] Casner, S., "Session Description Protocol (SDP) Bandwidth Modifiers for RTP Control Protocol (RTCP) Bandwidth", RFC 3556, July 2003.
- [RFC3711] Baugher, M., McGrew, D., Naslund, M., Carrara, E., and K. Norrman, "The Secure Real-time Transport Protocol (SRTP)", RFC 3711, March 2004.
- [RFC4585] Ott, J., Wenger, S., Sato, N., Burmeister, C., and J. Rey, "Extended RTP Profile for Real-time Transport Control Protocol (RTCP)-Based Feedback (RTP/AVPF)", RFC 4585, July 2006.
- [RFC4588] Rey, J., Leon, D., Miyazaki, A., Varsa, V., and R. Hakenberg, "RTP Retransmission Payload Format", RFC 4588, July 2006.
- [RFC4961] Wing, D., "Symmetric RTP / RTP Control Protocol (RTCP)", BCP 131, RFC 4961, July 2007.
- [RFC5104] Wenger, S., Chandra, U., Westerlund, M., and B. Burman, "Codec Control Messages in the RTP Audio-Visual Profile with Feedback (AVPF)", RFC 5104, February 2008.
- [RFC5124] Ott, J. and E. Carrara, "Extended Secure RTP Profile for Real-time Transport Control Protocol (RTCP)-Based Feedback (RTP/SAVPF)", RFC 5124, February 2008.
- [RFC5285] Singer, D. and H. Desineni, "A General Mechanism for RTP Header Extensions", RFC 5285, July 2008.
- [RFC5506] Johansson, I. and M. Westerlund, "Support for Reduced-Size Real-Time Transport Control Protocol (RTCP): Opportunities and Consequences", RFC 5506, April 2009.
- [RFC5761] Perkins, C. and M. Westerlund, "Multiplexing RTP Data and Control Packets on a Single Port", RFC 5761, April 2010.
- [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, May 2010.



- [RFC6051] Perkins, C. and T. Schierl, "Rapid Synchronisation of RTP Flows", RFC 6051, November 2010.
- [RFC6464] Lennox, J., Ivov, E., and E. Marocco, "A Real-time Transport Protocol (RTP) Header Extension for Client-to-Mixer Audio Level Indication", RFC 6464, December 2011.
- [RFC6465] Ivov, E., Marocco, E., and J. Lennox, "A Real-time Transport Protocol (RTP) Header Extension for Mixer-to-Client Audio Level Indication", RFC 6465, December 2011.
- [RFC6562] Perkins, C. and JM. Valin, "Guidelines for the Use of Variable Bit Rate Audio with Secure RTP", RFC 6562, March 2012.

## 17.2. Informative References

- [I-D.alvestrand-rtcweb-msid]  
Alvestrand, H., "Cross Session Stream Identification in the Session Description Protocol",  
draft-alvestrand-rtcweb-msid-02 (work in progress),  
May 2012.
- [I-D.ietf-avt-srtp-ekt]  
Wing, D., McGrew, D., and K. Fischer, "Encrypted Key Transport for Secure RTP", draft-ietf-avt-srtp-ekt-03  
(work in progress), October 2011.
- [I-D.ietf-rtcweb-qos]  
Dhesikan, S., Druta, D., Jones, P., and J. Polk, "DSCP and other packet markings for RTCWeb QoS",  
draft-ietf-rtcweb-qos-00 (work in progress), October 2012.
- [I-D.ietf-rtcweb-use-cases-and-requirements]  
Holmberg, C., Hakansson, S., and G. Eriksson, "Web Real-Time Communication Use-cases and Requirements",  
draft-ietf-rtcweb-use-cases-and-requirements-10 (work in progress), December 2012.
- [I-D.jesup-rtp-congestion-reqs]  
Jesup, R. and H. Alvestrand, "Congestion Control Requirements For Real Time Media",  
draft-jesup-rtp-congestion-reqs-00 (work in progress),  
March 2012.
- [I-D.westerlund-avtcore-multiplex-architecture]  
Westerlund, M., Burman, B., Perkins, C., and H. Alvestrand, "Guidelines for using the Multiplexing

Features of RTP",  
draft-westerlund-avtcore-multiplex-architecture-02 (work  
in progress), July 2012.

- [I-D.westerlund-avtcore-rtp-topologies-update]  
Westerlund, M. and S. Wenger, "RTP Topologies",  
draft-westerlund-avtcore-rtp-topologies-update-02 (work in  
progress), February 2013.
- [RFC4341] Floyd, S. and E. Kohler, "Profile for Datagram Congestion  
Control Protocol (DCCP) Congestion Control ID 2: TCP-like  
Congestion Control", RFC 4341, March 2006.
- [RFC4342] Floyd, S., Kohler, E., and J. Padhye, "Profile for  
Datagram Congestion Control Protocol (DCCP) Congestion  
Control ID 3: TCP-Friendly Rate Control (TFRC)", RFC 4342,  
March 2006.
- [RFC4383] Baugher, M. and E. Carrara, "The Use of Timed Efficient  
Stream Loss-Tolerant Authentication (TESLA) in the Secure  
Real-time Transport Protocol (SRTP)", RFC 4383,  
February 2006.
- [RFC4828] Floyd, S. and E. Kohler, "TCP Friendly Rate Control  
(TFRC): The Small-Packet (SP) Variant", RFC 4828,  
April 2007.
- [RFC5348] Floyd, S., Handley, M., Padhye, J., and J. Widmer, "TCP  
Friendly Rate Control (TFRC): Protocol Specification",  
RFC 5348, September 2008.
- [RFC5576] Lennox, J., Ott, J., and T. Schierl, "Source-Specific  
Media Attributes in the Session Description Protocol  
(SDP)", RFC 5576, June 2009.
- [RFC5681] Allman, M., Paxson, V., and E. Blanton, "TCP Congestion  
Control", RFC 5681, September 2009.
- [RFC5968] Ott, J. and C. Perkins, "Guidelines for Extending the RTP  
Control Protocol (RTCP)", RFC 5968, September 2010.
- [RFC6263] Marjou, X. and A. Sollaud, "Application Mechanism for  
Keeping Alive the NAT Mappings Associated with RTP / RTP  
Control Protocol (RTCP) Flows", RFC 6263, June 2011.

## Appendix A. Supported RTP Topologies

RTP supports both unicast and group communication, with participants being connected using wide range of transport-layer topologies. Some of these topologies involve only the end-points, while others use RTP translators and mixers to provide in-network processing. Properties of some RTP topologies are discussed in [I-D.westerlund-avtcore-rtp-topologies-update], and we further describe those expected to be useful for WebRTC in the following. We also goes into important RTP session aspects that the topology or implementation variant can place on a WebRTC end-point.

This section includes RTP topologies beyond the RECOMMENDED ones. This in an attempt to highlight the differences and the in many case small differences in implementation to support a larger set of possible topologies.

(tbd: This section needs reworking and clearer relation to [I-D.westerlund-avtcore-rtp-topologies-update].)

### A.1. Point to Point

The point-to-point RTP topology (Figure 3) is the simplest scenario for WebRTC applications. This is going to be very common for user to user calls.



Figure 3: Point to Point

This being the basic one lets use the topology to high-light a couple of details that are common for all RTP usage in the WebRTC context. First is the intention to multiplex RTP and RTCP over the same UDP-flow. Secondly is the question of using only a single RTP session or one per media type for legacy interoperability. Thirdly is the question of using multiple sender sources (SSRCs) per end-point.

Historically, RTP and RTCP have been run on separate UDP ports. With the increased use of Network Address/Port Translation (NAPT) this has become problematic, since maintaining multiple NAT bindings can be costly. It also complicates firewall administration, since multiple ports need to be opened to allow RTP traffic. To reduce these costs and session set-up times, support for multiplexing RTP data packets and RTCP control packets on a single port [RFC5761] will be supported.

In cases where there is only one type of media (e.g., a voice-only call) this topology will be implemented as a single RTP session, with bidirectional flows of RTP and RTCP packets, all then multiplexed onto a single 5-tuple. If multiple types of media are to be used (e.g., audio and video), then each type media can be sent as a separate RTP session using a different 5-tuple, allowing for separate transport level treatment of each type of media. Alternatively, all types of media can be multiplexed onto a single 5-tuple as a single RTP session, or as several RTP sessions if using a demultiplexing shim. Multiplexing different types of media onto a single 5-tuple places some limitations on how RTP is used, as described in "RTP Multiplexing Architecture"

[I-D.westerlund-avtcore-multiplex-architecture]. It is not expected that these limitations will significantly affect the scenarios targeted by WebRTC, but they can impact interoperability with legacy systems.

An RTP session have good support for simultaneously transport multiple media sources. Each media source uses an unique SSRC identifier and each SSRC has independent RTP sequence number and timestamp spaces. This is being utilized in WebRTC for several cases. One is to enable multiple media sources of the same type, an end-point that has two video cameras can potentially transmit video from both to its peer(s). Another usage is when a single RTP session is being used for both multiple media types, thus an end-point can transmit both audio and video to the peer(s). Thirdly to support multi-party cases as will be discussed below support for multiple SSRC of the same media type is needed.

Thus we can introduce a couple of different notations in the below two alternate figures of a single peer connection in a point to point set-up. The first depicting a setup where the peer connection established has two different RTP sessions, one for audio and one for video. The second one using a single RTP session. In both cases A has two video streams to send and one audio stream. B has only one audio and video stream. These are used to illustrate the relation between a peerConnection, the UDP flow(s), the RTP session(s) and the SSRCs that will be used in the later cases also. In the below figures RTCP flows are not included. They will flow bi-directionally between any RTP session instances in the different nodes.

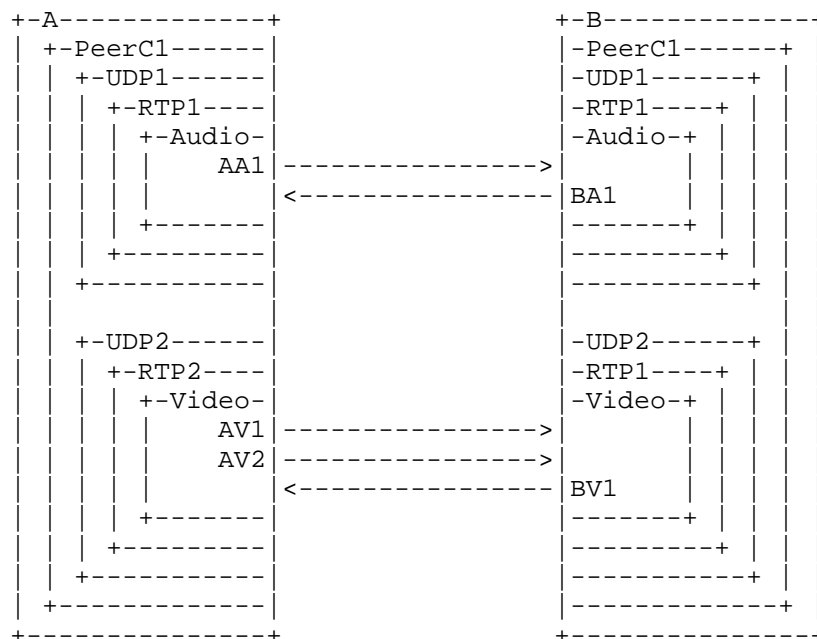


Figure 4: Point to Point: Multiple RTP sessions

As can be seen above in the Point to Point: Multiple RTP sessions (Figure 4) the single Peer Connection contains two RTP sessions over different UDP flows UDP 1 and UDP 2, i.e. their 5-tuples will be different, normally on source and destination ports. The first RTP session (RTP1) carries audio, one stream in each direction AA1 and BA1. The second RTP session contains two video streams from A (AV1 and AV2) and one from B to A (BV1).

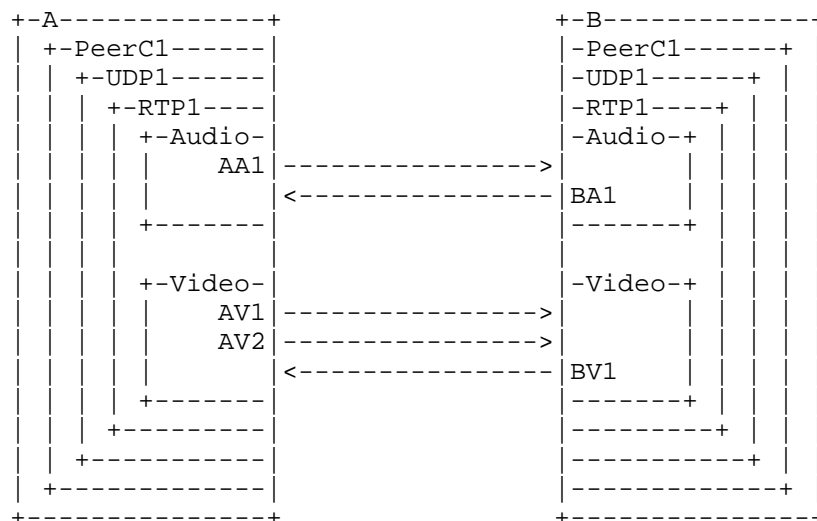


Figure 5: Point to Point: Single RTP session.

In (Figure 5) there is only a single UDP flow and RTP session (RTP1). This RTP session carries a total of five (5) RTP media streams (SSRCs). From A to B there is Audio (AA1) and two video (AV1 and AV2). From B to A there is Audio (BA1) and Video (BV1).

#### A.2. Multi-Unicast (Mesh)

For small multiparty calls, it is practical to set up a multi-unicast topology (Figure 6). In this topology, each participant sends individual unicast RTP/UDP/IP flows to each of the other participants using independent PeerConnections in a full mesh.

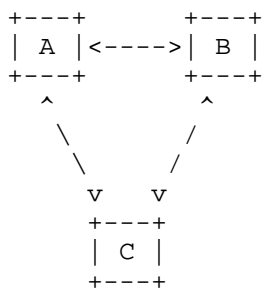


Figure 6: Multi-unicast

This topology has the benefit of not requiring central nodes. The downside is that it increases the used bandwidth at each sender by

requiring one copy of the RTP media streams for each participant that are part of the same session beyond the sender itself. Hence, this topology is limited to scenarios with few participants unless the media is very low bandwidth. The multi-unicast topology could be implemented as a single RTP session, spanning multiple peer-to-peer transport layer connections, or as several pairwise RTP sessions, one between each pair of peers. To maintain a coherent mapping between the relation between RTP sessions and PeerConnections we recommend that one implements this as individual RTP sessions. The only downside is that end-point A will not learn of the quality of any transmission happening between B and C based on RTCP. This has not been seen as a significant downside as now one has yet seen a need for why A would need to know about the B's and C's communication. An advantage of using separate RTP sessions is that it enables using different media bit-rates to the different peers, thus not forcing B to endure the same quality reductions if there are limitations in the transport from A to C as C will.

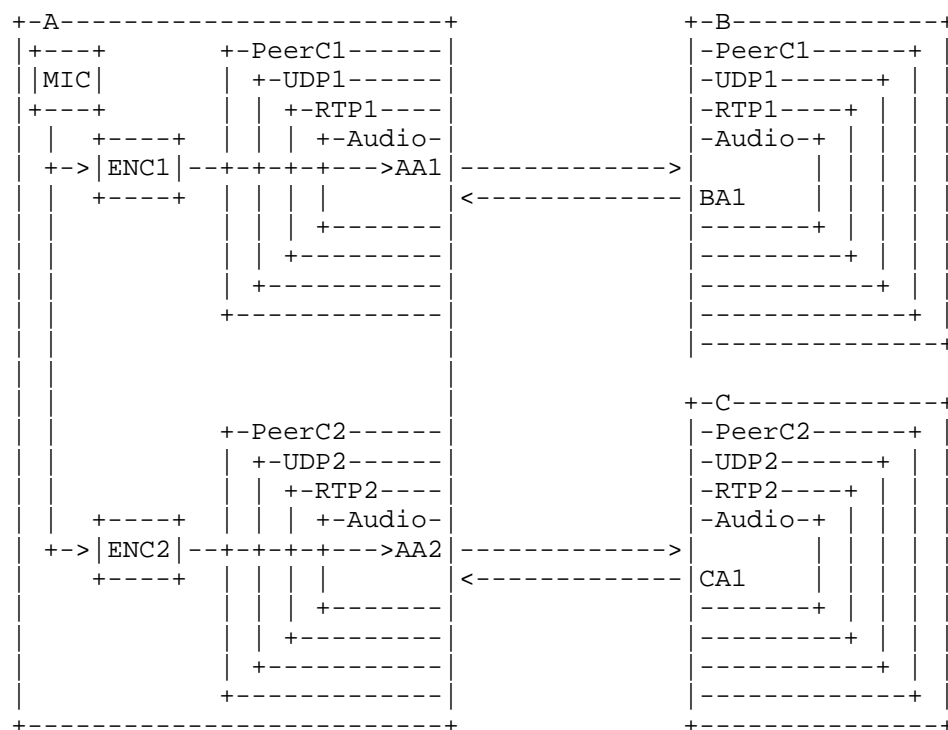


Figure 7: Session structure for Multi-Unicast Setup

Lets review how the RTP sessions looks from A's perspective by considering both how the media is a handled and what PeerConnections

and RTP sessions that are set-up in Figure 7. A's microphone is captured and the digital audio can then be feed into two different encoder instances each beeing associated with two different PeerConnections (PeerC1 and PeerC2) each containing independent RTP sessions (RTP1 and RTP2). The SSRCs in each RTP session will be completely independent and the media bit-rate produced by the encoder can also be tuned to address any congestion control requirements between A and B differently then for the path A to C.

For media encodings which are more resource consuming, like video, one could expect that it will be common that end-points that are resource constrained will use a different implementation strategy where the encoder is shared between the different PeerConnections as shown below Figure 8.

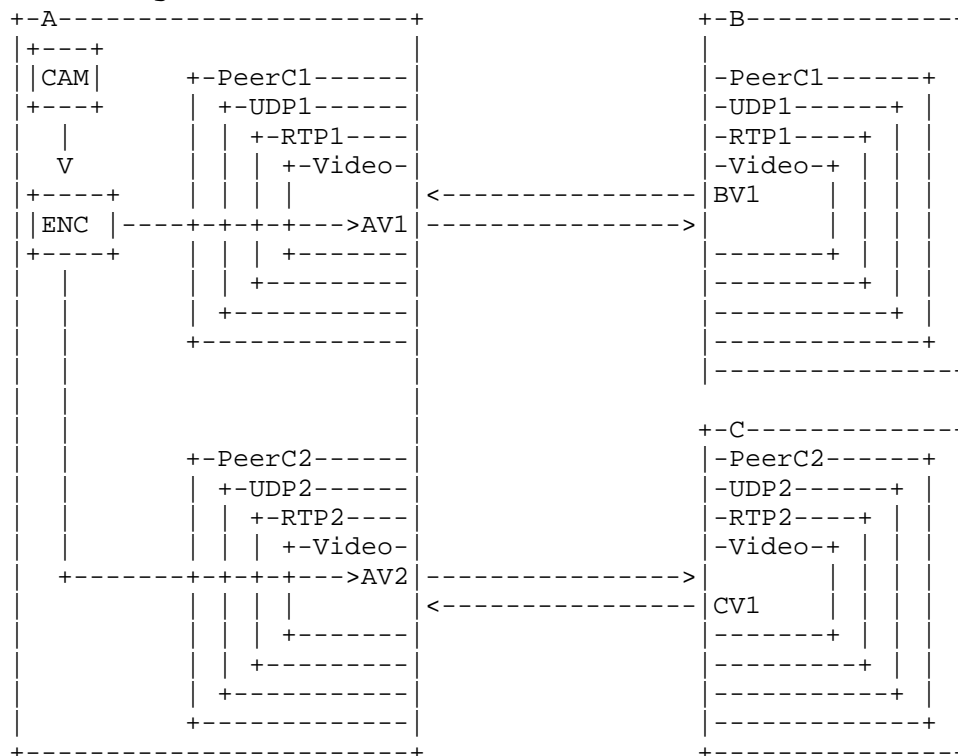


Figure 8: Single Encoder Multi-Unicast Setup

This will clearly save resources consumed by encoding but does introduce the need for the end-point A to make decisions on how it encodes the media so it suites delivery to both B and C. This is not limited to congestion control, also preferred resolution to receive based on dispalpy area available is another aspect requiring



consideration. The need for this type of decision logic does arise in several different topologies and implementation.

### A.3. Mixer Based

An mixer (Figure 9) is a centralised point that selects or mixes content in a conference to optimise the RTP session so that each end-point only needs connect to one entity, the mixer. The mixer can also reduce the bit-rate needed from the mixer down to a conference participants as the media sent from the mixer to the end-point can be optimised in different ways. These optimisations include methods like only choosing media from the currently most active speaker or mixing together audio so that only one audio stream is needed instead of 3 in the depicted scenario (Figure 9).

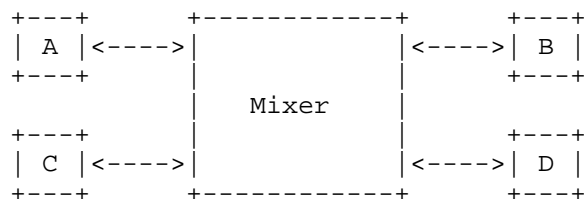


Figure 9: RTP Mixer with Only Unicast Paths

Mixers have two downsides, the first is that the mixer has to be a trusted node as they either performs media operations or at least re-packetize the media. Both type of operations requires when using SRTP that the mixer verifies integrity, decrypts the content, perform its operation and form new RTP packets, encrypts and integrity protect them. This applies to all types of mixers described below.

The second downside is that all these operations and optimization of the session requires processing. How much depends on the implementation as will become evident below.

The implementation of an mixer can take several different forms and we will discuss the main themes available that doesn't break RTP.

Please note that a Mixer could also contain translator functionalities, like a media transcoder to adjust the media bit-rate or codec used on a particular RTP media stream.

#### A.3.1. Media Mixing

This type of mixer is one which clearly can be called RTP mixer is likely the one that most thinks of when they hear the term mixer. Its basic patter of operation is that it will receive the different

participants RTP media stream. Select which that are to be included in a media domain mix of the incoming RTP media streams. Then create a single outgoing stream from this mix.

Audio mixing is straight forward and commonly possible to do for a number of participants. Lets assume that you want to mix N number of streams from different participants. Then the mixer need to perform decoding N times. Then it needs to produce N or N+1 mixes, the reasons that different mixes are needed are so that each contributing source get a mix which don't contain themselves, as this would result in an echo. When N is lower than the number of all participants one can produce a Mix of all N streams for the group that are curently not included in the mix, thus N+1 mixes. These audio streams are then encoded again, RTP packetized and sent out.

Video can't really be "mixed" and produce something particular useful for the users, however creating an composition out of the contributed video streams can be done. In fact it can be done in a number of ways, tiling the different streams creating a chessboard, selecting someone as more important and showing them large and a number of other sources as smaller is another. Also here one commonly need to produce a number of different compositions so that the contributing part doesn't need to see themselves. Then the mixer re-encodes the created video stream, RTP packetize it and send it out

The problem with media mixing is that it both consume large amount of media processing and encoding resources. The second is the quality degradation created by decoding and re-encoding the RTP media stream. Its advantage is that it is quite simplistic for the clients to handle as they don't need to handle local mixing and composition.

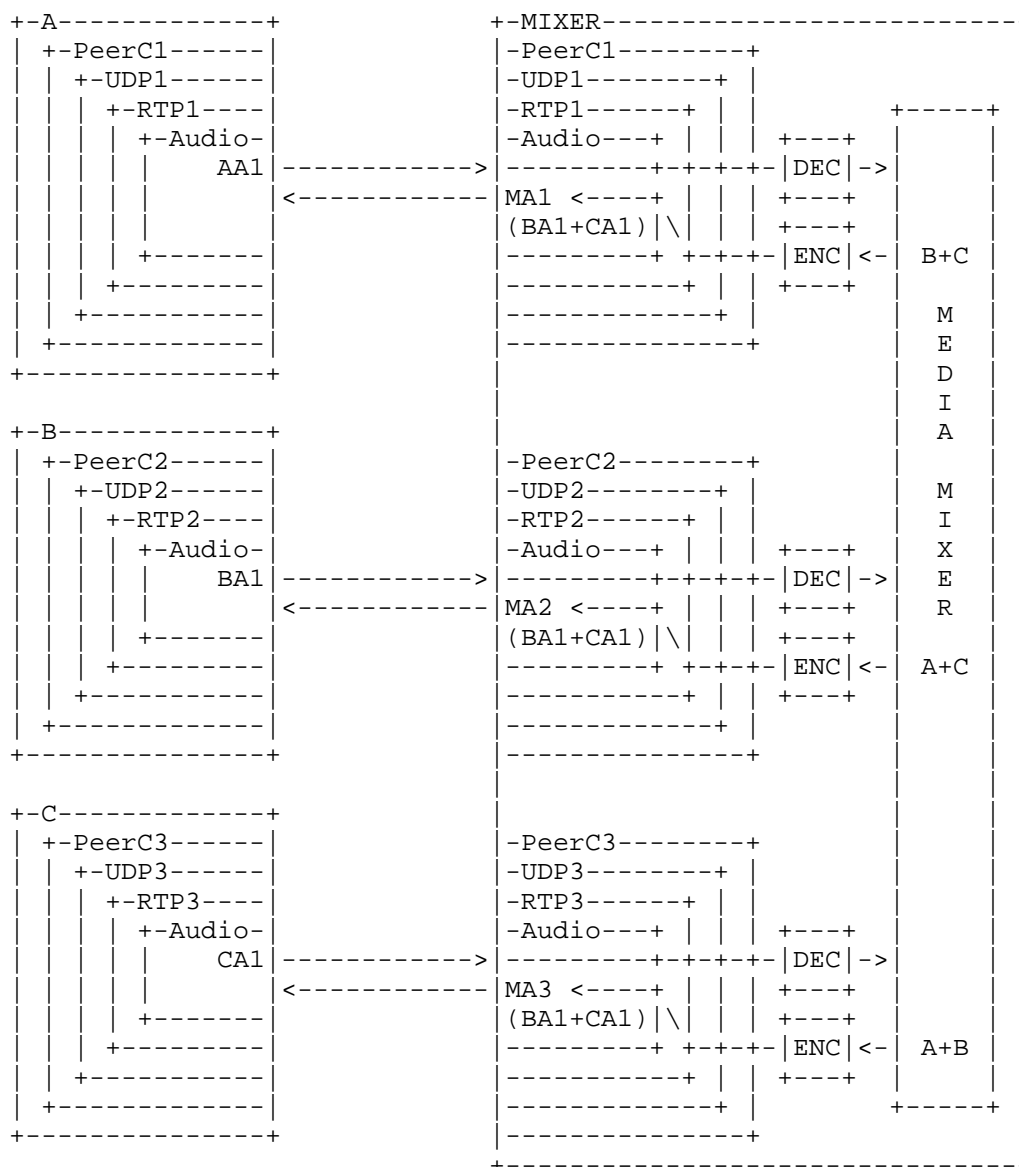


Figure 10: Session and SSRC details for Media Mixer

From an RTP perspective media mixing can be very straight forward as can be seen in Figure 10. The mixer present one SSRC towards the peer client, e.g. M1 to Peer A, which is the media mix of the other participants. As each peer receives a different version produced by the mixer there are no actual relation between the different RTP

sessions in the actual media or the transport level information. There is however one connection between RTP1-RTP3 in this figure. It has to do with the SSRC space and the identity information. When A receives the MA1 stream which is a combination of BA1 and CA1 streams in the other PeerConnections RTP could enable the mixer to include CSRC information in the MA1 stream to identify the contributing source BA1 and CA1.

The CSRC has in its turn utility in RTP extensions, like the in Section 5.2.3 discussed Mixer to Client audio levels RTP header extension [RFC6465]. If the SSRC from one PeerConnection are used as CSRC in another PeerConnection then RTP1, RTP2 and RTP3 becomes one joint session as they have a common SSRC space. At this stage one also need to consider which RTCP information one need to expose in the different legs. For the above situation commonly nothing more than the Source Description (SDES) information and RTCP BYE for CSRC need to be exposed. The main goal would be to enable the correct binding against the application logic and other information sources. This also enables loop detection in the RTP session.

#### A.3.1.1. RTP Session Termination

There exist an possible implementation choice to have the RTP sessions being separated between the different legs in the multi-party communication session and only generate RTP media streams in each without carrying on RTP/RTCP level any identity information about the contributing sources. This removes both the functionality that CSRC can provide and the possibility to use any extensions that build on CSRC and the loop detection. It might appear a simplification if SSRC collision would occur between two different end-points as they can be avoided to be resolved and instead remapped between the independent sessions if at all exposed. However, SSRC/CSRC remapping requires that SSRC/CSRC are never exposed to the WebRTC JavaScript client to use as reference. This as they only have local importance if they are used on a multi-party session scope the result would be mis-referencing. Also SSRC collision handling will still be needed as it can occur between the mixer and the end-point.

Session termination might appear to resolve some issues, it however creates other issues that needs resolving, like loop detection, identification of contributing sources and the need to handle mapped identities and ensure that the right one is used towards the right identities and never used directly between multiple end-points.

#### A.3.2. Media Switching

An RTP Mixer based on media switching avoids the media decoding and encoding cycle in the mixer, but not the decryption and re-encryption

cycle as one rewrites RTP headers. This both reduces the amount of computational resources needed in the mixer and increases the media quality per transmitted bit. This is achieved by letting the mixer have a number of SSRCs that represents conceptual or functional streams the mixer produces. These streams are created by selecting media from one of the by the mixer received RTP media streams and forward the media using the mixers own SSRCs. The mixer can then switch between available sources if that is needed by the concept for the source, like currently active speaker.

To achieve a coherent RTP media stream from the mixer's SSRC the mixer is forced to rewrite the incoming RTP packet's header. First the SSRC field has to be set to the value of the Mixer's SSRC. Secondly, the sequence number is set to the next in the sequence of outgoing packets it sent. Thirdly the RTP timestamp value needs to be adjusted using an offset that changes each time one switch media source. Finally depending on the negotiation the RTP payload type value representing this particular RTP payload configuration might have to be changed if the different PeerConnections have not arrived on the same numbering for a given configuration. This also requires that the different end-points do support a common set of codecs, otherwise media transcoding for codec compatibility is still needed.

Lets consider the operation of media switching mixer that supports a video conference with six participants (A-F) where the two latest speakers in the conference are shown to each participants. Thus the mixer has two SSRCs sending video to each peer.

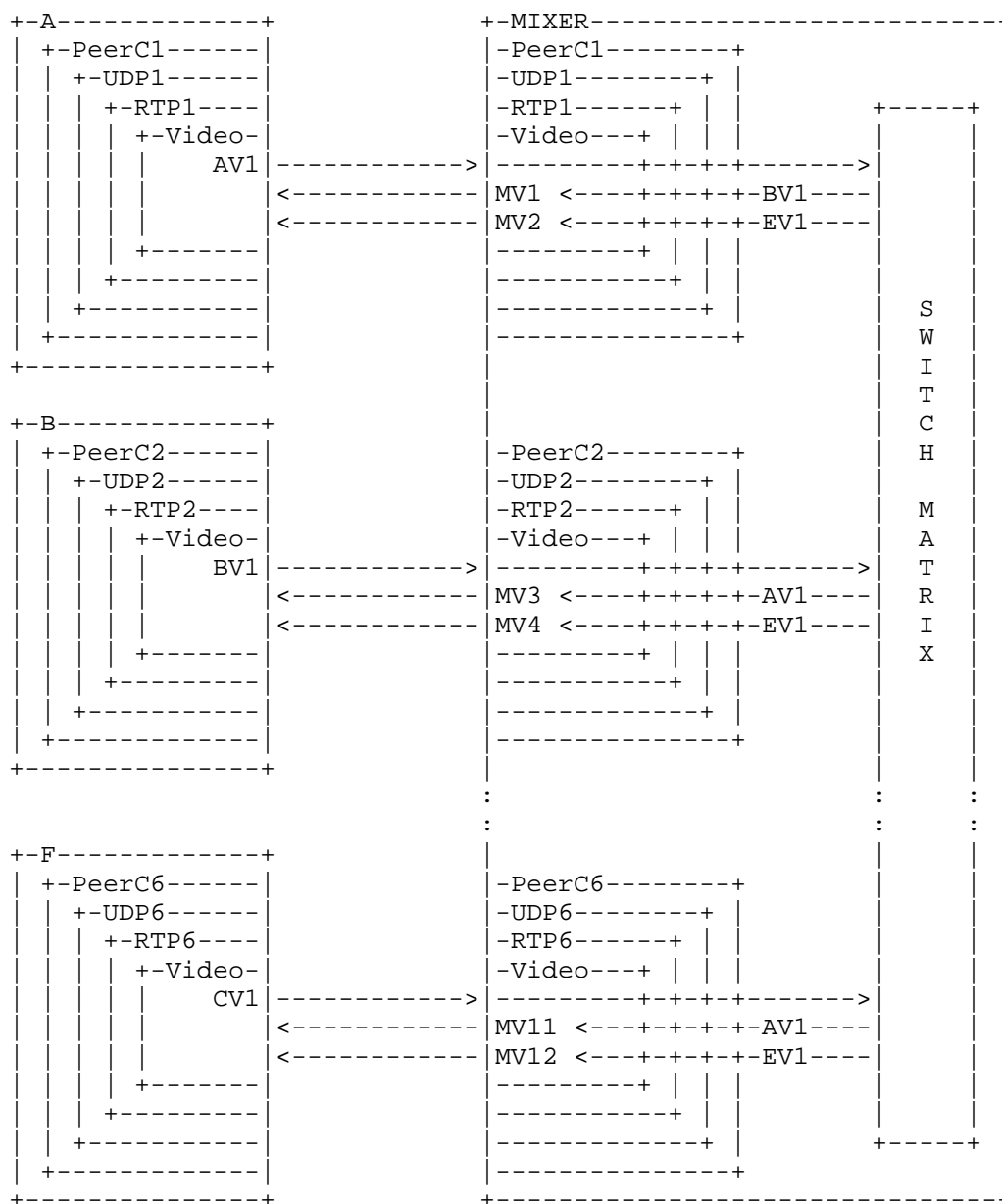


Figure 11: Media Switching RTP Mixer

The Media Switching RTP mixer can similar to the Media Mixing one reduce the bit-rate needed towards the different peers by selecting and switching in a sub-set of RTP media streams out of the ones it

receives from the conference participations.

To ensure that a media receiver can correctly decode the RTP media stream after a switch, it becomes necessary to ensure for state saving codecs that they start from default state at the point of switching. Thus one common tool for video is to request that the encoding creates an intra picture, something that isn't dependent on earlier state. This can be done using Full Intra Request RTCP codec control message as discussed in Section 5.1.1.

Also in this type of mixer one could consider to terminate the RTP sessions fully between the different PeerConnection. The same arguments and considerations as discussed in Appendix A.3.1.1 applies here.

#### A.3.3. Media Projecting

Another method for handling media in the RTP mixer is to project all potential sources (SSRCs) into a per end-point independent RTP session. The mixer can then select which of the potential sources that are currently actively transmitting media, despite that the mixer in another RTP session receives media from that end-point. This is similar to the media switching Mixer but have some important differences in RTP details.

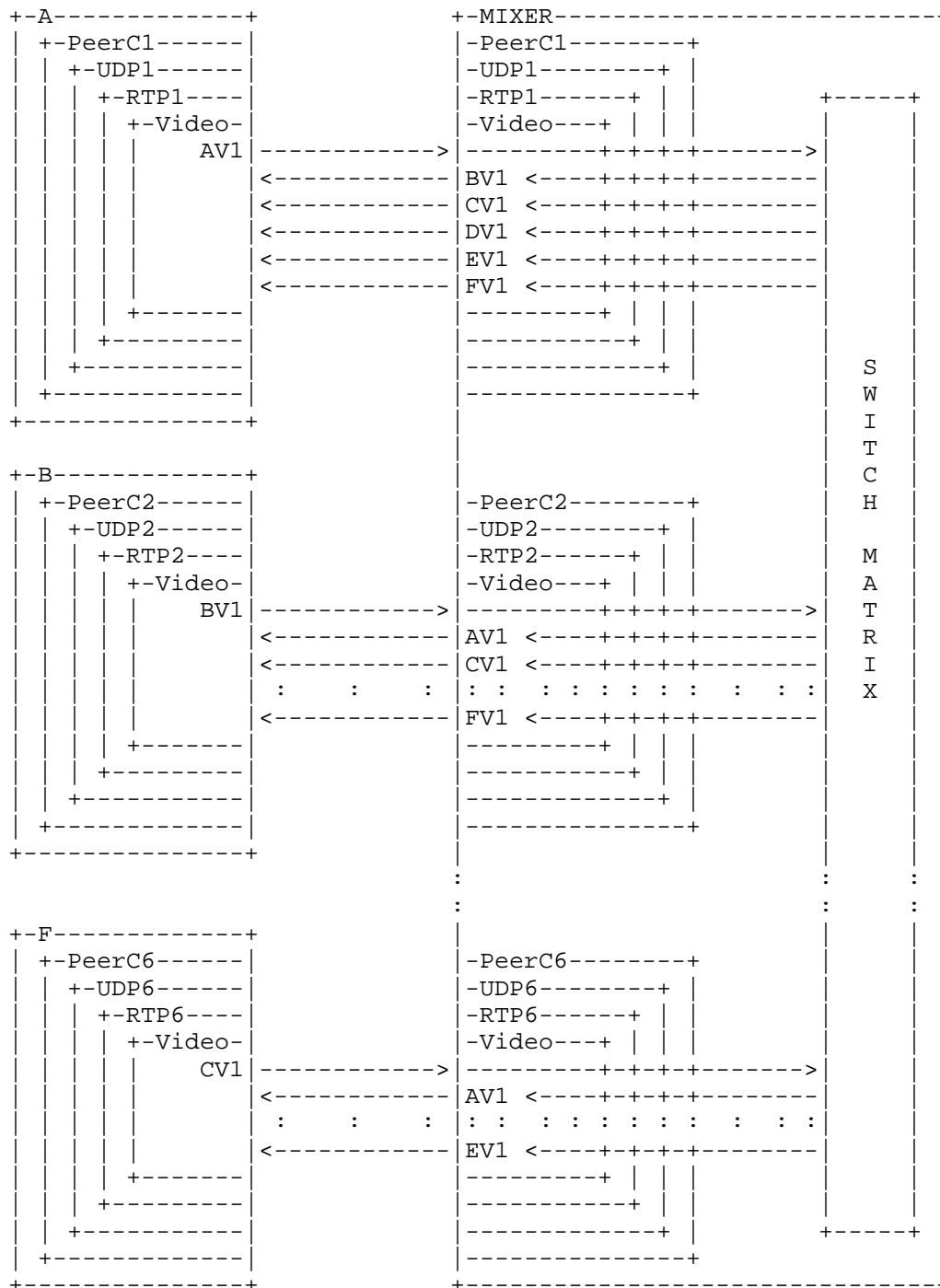




Figure 12: Media Projecting Mixer

So in this six participant conference depicted above in (Figure 12) one can see that end-point A will in this case be aware of 5 incoming SSRCs, BV1-FV1. If this mixer intend to have the same behavior as in Appendix A.3.2 where the mixer provides the end-points with the two latest speaking end-points, then only two out of these five SSRCs will concurrently transmit media to A. As the mixer selects which source in the different RTP sessions that transmit media to the end-points each RTP media stream will require some rewriting when being projected from one session into another. The main thing is that the sequence number will need to be consecutively incremented based on the packet actually being transmitted in each RTP session. Thus the RTP sequence number offset will change each time a source is turned on in RTP session.

As the RTP sessions are independent the SSRC numbers used can be handled independently also thus working around any SSRC collisions by having remapping tables between the RTP sessions. However the related WebRTC MediaStream signalling need to be correspondingly changed to ensure consistent WebRTC MediaStream to SSRC mappings between the different PeerConnections and the same comment that higher functions MUST NOT use SSRC as references to RTP media streams applies also here.

The mixer will also be responsible to act on any RTCP codec control requests coming from an end-point and decide if it can act on it locally or needs to translate the request into the RTP session that contains the media source. Both end-points and the mixer will need to implement conference related codec control functionalities to provide a good experience. Full Intra Request to request from the media source to provide switching points between the sources, Temporary Maximum Media Bit-rate Request (TMMBR) to enable the mixer to aggregate congestion control response towards the media source and have it adjust its bit-rate in case the limitation is not in the source to mixer link.

This version of the mixer also puts different requirements on the end-point when it comes to decoder instances and handling of the RTP media streams providing media. As each projected SSRC can at any time provide media the end-point either needs to handle having thus many allocated decoder instances or have efficient switching of decoder contexts in a more limited set of actual decoder instances to cope with the switches. The WebRTC application also gets more responsibility to update how the media provides is to be presented to the user.

#### A.4. Translator Based

There is also a variety of translators. The core commonality is that they do not need to make themselves visible in the RTP level by having an SSRC themselves. Instead they sit between one or more end-point and perform translation at some level. It can be media transcoding, protocol translation or covering missing functionality for a legacy end-point or simply relay packets between transport domains or to realize multi-party. We will go in details below.

##### A.4.1. Transcoder

A transcoder operates on media level and really used for two purposes, the first is to allow two end-points that doesn't have a common set of media codecs to communicate by translating from one codec to another. The second is to change the bit-rate to a lower one. For WebRTC end-points communicating with each other only the first one is relevant. In certain legacy deployment media transcoder will be necessary to ensure both codecs and bit-rate falls within the envelope the legacy end-point supports.

As transcoding requires access to the media, the transcoder has to be within the security context and access any media encryption and integrity keys. On the RTP plane a media transcoder will in practice fork the RTP session into two different domains that are highly decoupled when it comes to media parameters and reporting, but not identities. To maintain signalling bindings to SSRCs a transcoder is likely needing to use the SSRC of one end-point to represent the transcoded RTP media stream to the other end-point(s). The congestion control loop can be terminated in the transcoder as the media bit-rate being sent by the transcoder can be adjusted independently of the incoming bit-rate. However, for optimizing performance and resource consumption the translator needs to consider what signals or bit-rate reductions it needs to send towards the source end-point. For example receiving a 2.5 Mbps video stream and then send out a 250 kbps video stream after transcoding is a waste of resources. In most cases a 500 kbps video stream from the source in the right resolution is likely to provide equal quality after transcoding as the 2.5 Mbps source stream. At the same time increasing media bit-rate further than what is needed to represent the incoming quality accurate is also wasted resources.

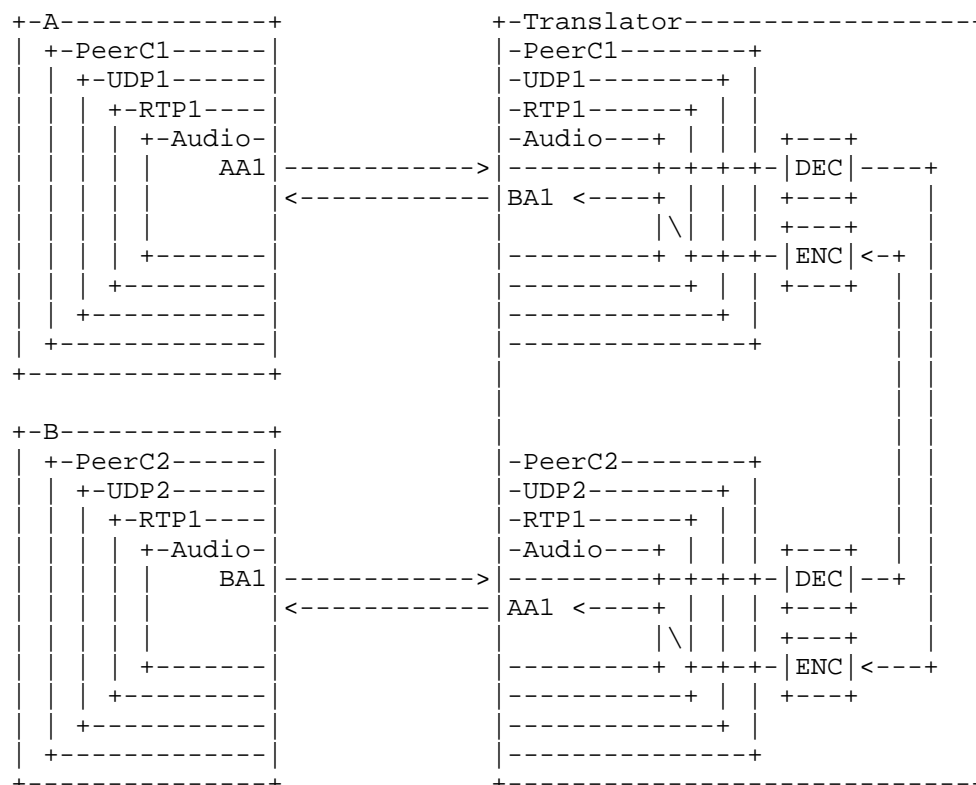


Figure 13: Media Transcoder

Figure 13 exposes some important details. First of all you can see the SSRC identifiers used by the translator are the corresponding end-points. Secondly, there is a relation between the RTP sessions in the two different PeerConnections that are represented by having both parts be identified by the same level and they need to share certain contexts. Also certain type of RTCP messages will need to be bridged between the two parts. Certain RTCP feedback messages are likely needed to be sourced by the translator in response to actions by the translator and its media encoder.

#### A.4.2. Gateway / Protocol Translator

Gateways are used when some protocol feature that are needed are not supported by an end-point wants to participate in session. This RTP translator in Figure 14 takes on the role of ensuring that from the perspective of participant A, participant B appears as a fully compliant WebRTC end-point (that is, it is the combination of the Translator and participant B that looks like a WebRTC end point).

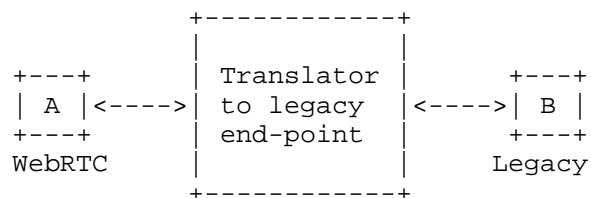


Figure 14: Gateway (RTP translator) towards legacy end-point

For WebRTC there are a number of requirements that could force the need for a gateway if a WebRTC end-point is to communicate with a legacy end-point, such as support of ICE and DTLS-SRTP for key management. On RTP level the main functions that might be missing in a legacy implementation that otherwise support RTP are RTCP in general, SRTP implementation, congestion control and feedback messages needed to make it work.

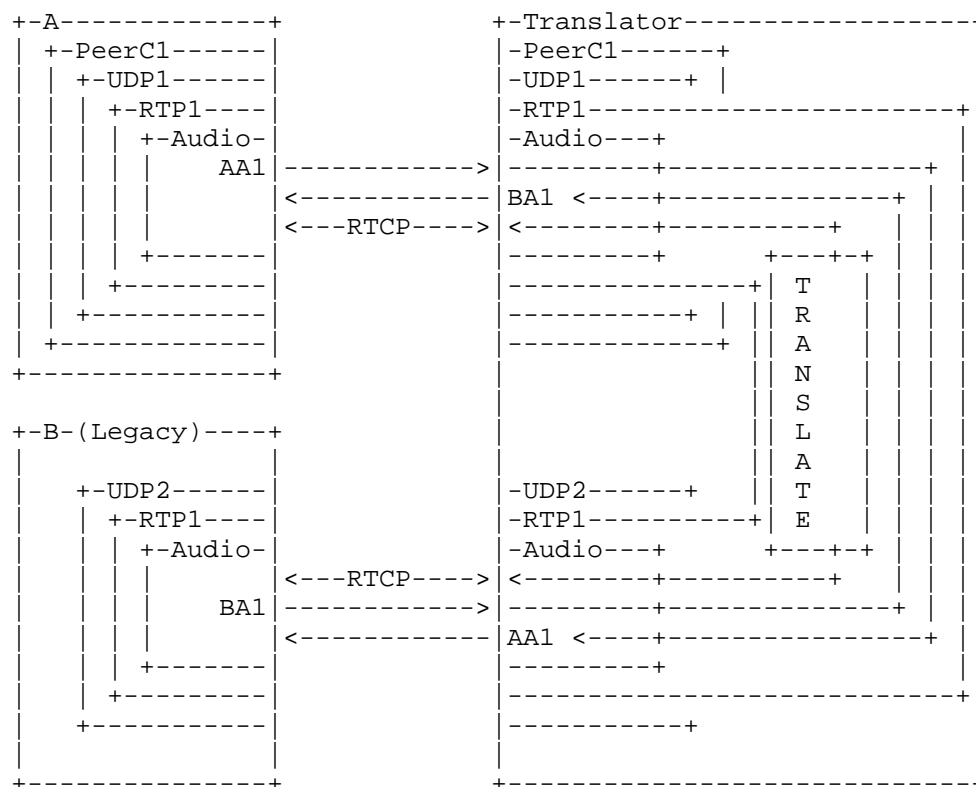


Figure 15: RTP/RTCP Protocol Translator

The legacy gateway can be implemented in several ways and what it need to change is highly dependent on what functions it need to proxy for the legacy end-point. One possibility is depicted in Figure 15 where the RTP media streams are compatible and forward without changes. However, their RTP header values are captured to enable the RTCP translator to create RTCP reception information related to the leg between the end-point and the translator. This can then be combined with the more basic RTCP reports that the legacy endpoint (B) provides to give compatible and expected RTCP reporting to A. Thus enabling at least full congestion control on the path between A and the translator. If B has limited possibilities for congestion response for the media then the translator might need the capability to perform media transcoding to address cases where it otherwise would need to terminate media transmission.

As the translator are generating RTP/RTCP traffic on behalf of B to A it will need to be able to correctly protect these packets that it translates or generates. Thus security context information are needed in this type of translator if it operates on the RTP/RTCP packet content or media. In fact one of the more likely scenario is that the translator (gateway) will need to have two different security contexts one towards A and one towards B and for each RTP/RTCP packet do a authenticity verification, decryption followed by a encryption and integrity protection operation to resolve mismatch in security systems.

#### A.4.3. Relay

There exist a class of translators that operates on transport level below RTP and thus do not effect RTP/RTCP packets directly. They come in two distinct flavours, the one used to bridge between two different transport or address domains to more function as a gateway and the second one which is to provide a group communication feature as depicted below in Figure 16.

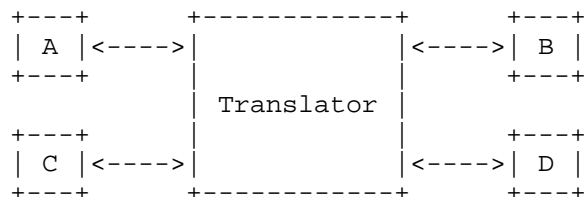


Figure 16: RTP Translator (Relay) with Only Unicast Paths

The first kind is straight forward and is likely to exist in WebRTC context when an legacy end-point is compatible with the exception for ICE, and thus needs a gateway that terminates the ICE and then

forwards all the RTP/RTCP traffic and key management to the end-point only rewriting the IP/UDP to forward the packet to the legacy node.

The second type is useful if one wants a less complex central node or a central node that is outside of the security context and thus do not have access to the media. This relay takes on the role of forwarding the media (RTP and RTCP) packets to the other end-points but doesn't perform any RTP or media processing. Such a device simply forwards the media from each sender to all of the other participants, and is sometimes called a transport-layer translator. In Figure 16, participant A will only need to send a media once to the relay, which will redistribute it by sending a copy of the stream to participants B, C, and D. Participant A will still receive three RTP streams with the media from B, C and D if they transmit simultaneously. This is from an RTP perspective resulting in an RTP session that behaves equivalent to one transporter over an IP Any Source Multicast (ASM).

This results in one common RTP session between all participants despite that there will be independent PeerConnections created to the translator as depicted below Figure 17.

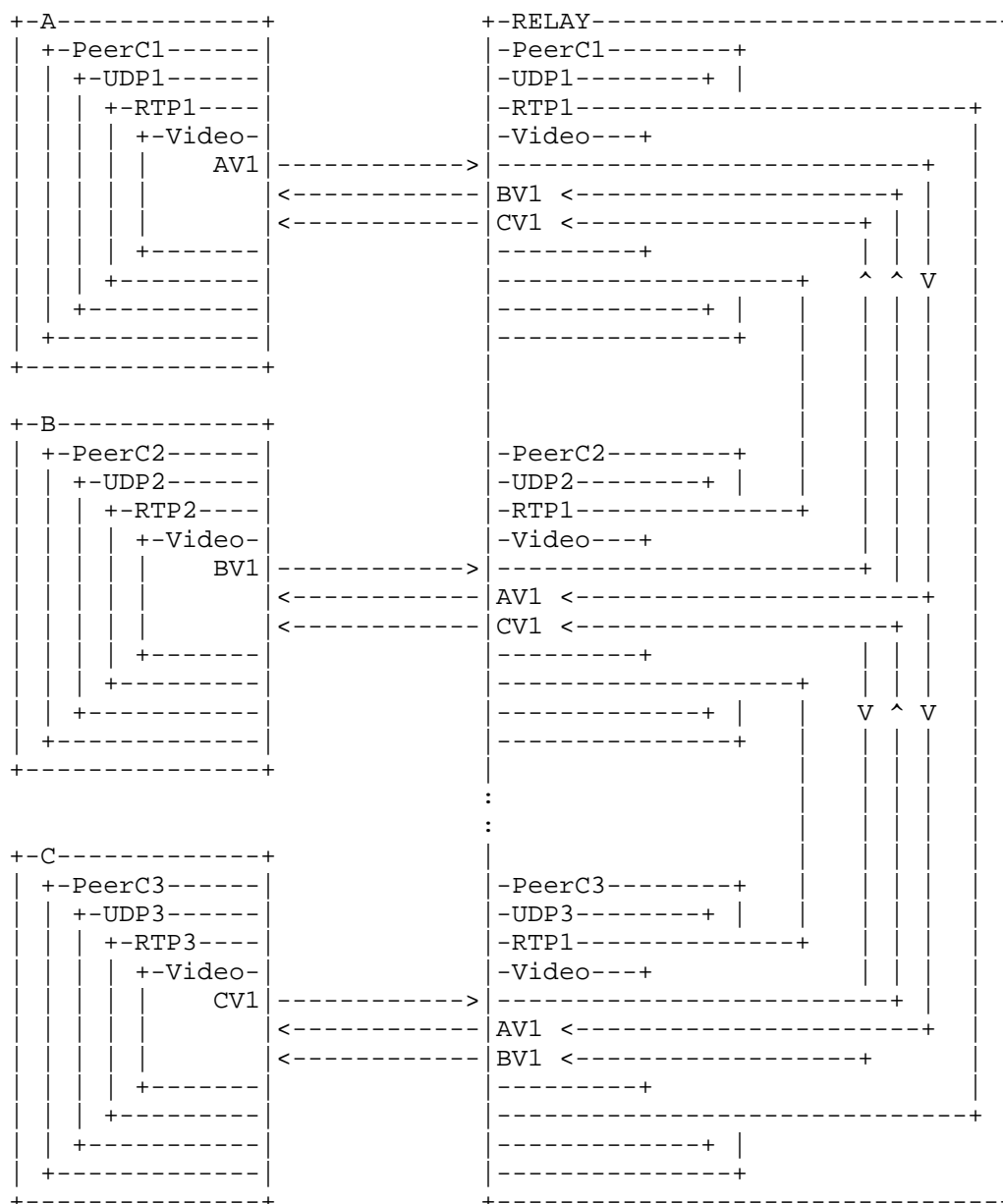


Figure 17: Transport Multi-party Relay

As the Relay RTP and RTCP packets between the UDP flows as indicated by the arrows for the media flow a given WebRTC end-point, like A will see the remote sources BV1 and CV1. There will be also two

different network paths between A, and B or C. This results in that the client A has to be capable of handling that when determining congestion state that there might exist multiple destinations on the far side of a PeerConnection and that these paths have to be treated differently. It also results in a requirement to combine the different congestion states into a decision to transmit a particular RTP media stream suitable to all participants.

It is also important to note that the relay can not perform selective relaying of some sources and not others. The reason is that the RTCP reporting in that case becomes inconsistent and without explicit information about it being blocked has to be interpreted as severe congestion.

In this usage it is also necessary that the session management has configured a common set of RTP configuration including RTP payload formats as when A sends a packet with pt=97 it will arrive at both B and C carrying pt=97 and having the same packetization and encoding, no entity will have manipulated the packet.

When it comes to security there exist some additional requirements to ensure that the property that the relay can't read the media traffic is enforced. First of all the key to be used has to be agreed such so that the relay doesn't get it, e.g. no DTLS-SRTP handshake with the relay, instead some other method needs to be used. Secondly, the keying structure has to be capable of handling multiple end-points in the same RTP session.

The second problem can basically be solved in two ways. Either a common master key from which all derive their per source key for SRTP. The second alternative which might be more practical is that each end-point has its own key used to protect all RTP/RTCP packets it sends. Each participants key are then distributed to the other participants. This second method could be implemented using DTLS-SRTP to a special key server and then use Encrypted Key Transport [I-D.ietf-avt-srtp-ekt] to distribute the actual used key to the other participants in the RTP session Figure 18. The first one could be achieved using MIKEY messages in SDP.



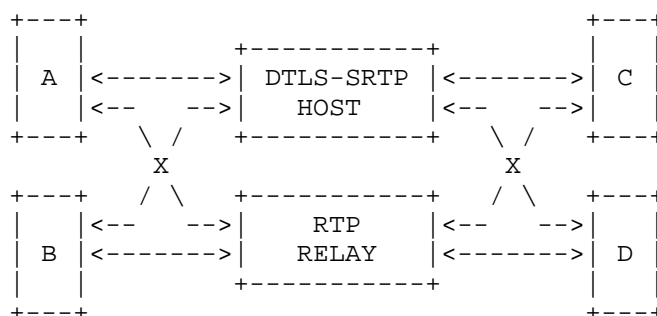


Figure 18: DTLS-SRTP host and RTP Relay Separated

The relay can still verify that a given SSRC isn't used or spoofed by another participant within the multi-party session by binding SSRCs on their first usage to a given source address and port pair. Packets carrying that source SSRC from other addresses can be suppressed to prevent spoofing. This is possible as long as SRTP is used which leaves the SSRC of the packet originator in RTP and RTCP packets in the clear. If such packet level method for enforcing source authentication within the group, then there exist cryptographic methods such as TESLA [RFC4383] that could be used for true source authentication.

#### A.5. End-point Forwarding

An WebRTC end-point (B in Figure 19) will receive a WebRTC MediaStream (set of SSRCs) over a PeerConnection (from A). For the moment is not decided if the end-point is allowed or not to in its turn send that WebRTC MediaStream over another PeerConnection to C. This section discusses the RTP and end-point implications of allowing such functionality, which on the API level is extremely simplistic to perform.

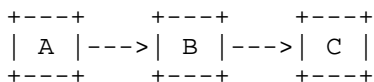


Figure 19: MediaStream Forwarding

There exist two main approaches to how B forwards the media from A to C. The first one is to simply relay the RTP media stream. The second one is for B to act as a transcoder. Lets consider both approaches.

A relay approach will result in that the WebRTC end-points will have to have the same capabilities as being discussed in Relay (Appendix A.4.3). Thus A will see an RTP session that is extended

beyond the PeerConnection and see two different receiving end-points with different path characteristics (B and C). Thus A's congestion control needs to be capable of handling this. The security solution can either support mechanism that allows A to inform C about the key A is using despite B and C having agreed on another set of keys. Alternatively B will decrypt and then re-encrypt using a new key. The relay based approach has the advantage that B does not need to transcode the media thus both maintaining the quality of the encoding and reducing B's complexity requirements. If the right security solutions are supported then also C will be able to verify the authenticity of the media coming from A. As downside A are forced to take both B and C into consideration when delivering content.

The media transcoder approach is similar to having B act as Mixer terminating the RTP session combined with the transcoder as discussed in Appendix A.4.1. A will only see B as receiver of its media. B will responsible to produce a RTP media stream suitable for the B to C PeerConnection. This might require media transcoding for congestion control purpose to produce a suitable bit-rate. Thus loosing media quality in the transcoding and forcing B to spend the resource on the transcoding. The media transcoding does result in a separation of the two different legs removing almost all dependencies. B could choice to implement logic to optimize its media transcoding operation, by for example requesting media properties that are suitable for C also, thus trying to avoid it having to transcode the content and only forward the media payloads between the two sides. For that optimization to be practical WebRTC end-points have to support sufficiently good tools for codec control.

#### A.6. Simulcast

This section discusses simulcast in the meaning of providing a node, for example a stream switching Mixer, with multiple different encoded version of the same media source. In the WebRTC context that appears to be most easily accomplished by establishing multiple PeerConnection all being feed the same set of WebRTC MediaStreams. Each PeerConnection is then configured to deliver a particular media quality and thus media bit-rate. This will work well as long as the end-point implements media encoding according to Figure 7. Then each PeerConnection will receive an independently encoded version and the codec parameters can be agreed specifically in the context of this PeerConnection.

For simulcast to work one needs to prevent that the end-point deliver content encoded as depicted in Figure 8. If a single encoder instance is feed to multiple PeerConnections the intention of performing simulcast will fail.

Thus it needs to be considered to explicitly signal which of the two implementation strategies that are desired and which will be done. At least making the application and possible the central node interested in receiving simulcast of an end-points RTP media streams to be aware if it will function or not.

#### Authors' Addresses

Colin Perkins  
University of Glasgow  
School of Computing Science  
Glasgow G12 8QQ  
United Kingdom

Email: [csp@csp Perkins.org](mailto:csp@csp Perkins.org)

Magnus Westerlund  
Ericsson  
Farogatan 6  
SE-164 80 Kista  
Sweden

Phone: +46 10 714 82 87  
Email: [magnus.westerlund@ericsson.com](mailto:magnus.westerlund@ericsson.com)

Joerg Ott  
Aalto University  
School of Electrical Engineering  
Espoo 02150  
Finland

Email: [jorg.ott@aalto.fi](mailto:jorg.ott@aalto.fi)



RTC-Web  
Internet-Draft  
Intended status: Standards Track  
Expires: January 6, 2020

E. Rescorla  
RTFM, Inc.  
July 5, 2019

Security Considerations for WebRTC  
draft-ietf-rtcweb-security-12

Abstract

WebRTC is a protocol suite for use with real-time applications that can be deployed in browsers - "real time communication on the Web". This document defines the WebRTC threat model and analyzes the security threats of WebRTC in that model.

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 6, 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.

This document may contain material from IETF Documents or IETF Contributions published or made publicly available before November 10, 2008. The person(s) controlling the copyright in some of this material may not have granted the IETF Trust the right to allow modifications of such material outside the IETF Standards Process. Without obtaining an adequate license from the person(s) controlling the copyright in such materials, this document may not be modified outside the IETF Standards Process, and derivative works of it may not be created outside the IETF Standards Process, except to format it for publication as an RFC or to translate it into languages other than English.

## Table of Contents

1. Introduction . . . . .	3
2. Terminology . . . . .	4
3. The Browser Threat Model . . . . .	4
3.1. Access to Local Resources . . . . .	5
3.2. Same-Origin Policy . . . . .	5
3.3. Bypassing SOP: CORS, WebSockets, and consent to communicate . . . . .	6
4. Security for WebRTC Applications . . . . .	7
4.1. Access to Local Devices . . . . .	7
4.1.1. Threats from Screen Sharing . . . . .	8
4.1.2. Calling Scenarios and User Expectations . . . . .	8
4.1.2.1. Dedicated Calling Services . . . . .	9
4.1.2.2. Calling the Site You're On . . . . .	9
4.1.3. Origin-Based Security . . . . .	10
4.1.4. Security Properties of the Calling Page . . . . .	11
4.2. Communications Consent Verification . . . . .	12
4.2.1. ICE . . . . .	13
4.2.2. Masking . . . . .	13
4.2.3. Backward Compatibility . . . . .	14
4.2.4. IP Location Privacy . . . . .	15
4.3. Communications Security . . . . .	15
4.3.1. Protecting Against Retrospective Compromise . . . . .	16
4.3.2. Protecting Against During-Call Attack . . . . .	17
4.3.2.1. Key Continuity . . . . .	17
4.3.2.2. Short Authentication Strings . . . . .	18
4.3.2.3. Third Party Identity . . . . .	19
4.3.2.4. Page Access to Media . . . . .	19
4.3.3. Malicious Peers . . . . .	20
4.4. Privacy Considerations . . . . .	20
4.4.1. Correlation of Anonymous Calls . . . . .	21
4.4.2. Browser Fingerprinting . . . . .	21
5. Security Considerations . . . . .	21
6. Acknowledgements . . . . .	21
7. IANA Considerations . . . . .	21

8. Changes Since -04 . . . . .	21
9. References . . . . .	22
9.1. Normative References . . . . .	22
9.2. Informative References . . . . .	22
Author's Address . . . . .	25

## 1. Introduction

The Real-Time Communications on the Web (RTCWEB) working group has standardized protocols for real-time communications between Web browsers, generally called "WebRTC" [I-D.ietf-rtcweb-overview]. The major use cases for WebRTC technology are real-time audio and/or video calls, Web conferencing, and direct data transfer. Unlike most conventional real-time systems, (e.g., SIP-based [RFC3261] soft phones) WebRTC communications are directly controlled by some Web server. A simple case is shown below.

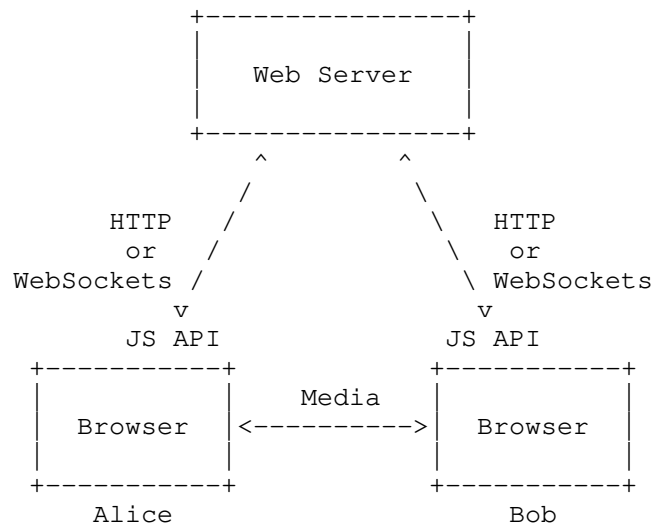


Figure 1: A simple WebRTC system

In the system shown in Figure 1, Alice and Bob both have WebRTC-enabled browsers and they visit some Web server which operates a calling service. Each of their browsers exposes standardized JavaScript calling APIs (implemented as browser built-ins) which are used by the Web server to set up a call between Alice and Bob. The Web server also serves as the signaling channel to transport control messages between the browsers. While this system is topologically similar to a conventional SIP-based system (with the Web server acting as the signaling service and browsers acting as softphones),

control has moved to the central Web server; the browser simply provides API points that are used by the calling service. As with any Web application, the Web server can move logic between the server and JavaScript in the browser, but regardless of where the code is executing, it is ultimately under control of the server.

It should be immediately apparent that this type of system poses new security challenges beyond those of a conventional VoIP system. In particular, it needs to contend with malicious calling services. For example, if the calling service can cause the browser to make a call at any time to any callee of its choice, then this facility can be used to bug a user's computer without their knowledge, simply by placing a call to some recording service. More subtly, if the exposed APIs allow the server to instruct the browser to send arbitrary content, then they can be used to bypass firewalls or mount denial of service attacks. Any successful system will need to be resistant to this and other attacks.

A companion document [I-D.ietf-rtcweb-security-arch] describes a security architecture intended to address the issues raised in this document.

## 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. The Browser Threat Model

The security requirements for WebRTC follow directly from the requirement that the browser's job is to protect the user. Huang et al. [huang-w2sp] summarize the core browser security guarantee as:

Users can safely visit arbitrary web sites and execute scripts provided by those sites.

It is important to realize that this includes sites hosting arbitrary malicious scripts. The motivation for this requirement is simple: it is trivial for attackers to divert users to sites of their choice. For instance, an attacker can purchase display advertisements which direct the user (either automatically or via user clicking) to their site, at which point the browser will execute the attacker's scripts. Thus, it is important that it be safe to view arbitrarily malicious pages. Of course, browsers inevitably have bugs which cause them to fall short of this goal, but any new WebRTC functionality must be



designed with the intent to meet this standard. The remainder of this section provides more background on the existing Web security model.

In this model, then, the browser acts as a Trusted Computing Base (TCB) both from the user's perspective and to some extent from the server's. While HTML and JavaScript (JS) provided by the server can cause the browser to execute a variety of actions, those scripts operate in a sandbox that isolates them both from the user's computer and from each other, as detailed below.

Conventionally, we refer to either web attackers, who are able to induce you to visit their sites but do not control the network, and network attackers, who are able to control your network. Network attackers correspond to the [RFC3552] "Internet Threat Model". Note that in some cases, a network attacker is also a web attacker, since transport protocols that do not provide integrity protection allow the network to inject traffic as if they were any communications peer. TLS, and HTTPS in particular, prevent against these attacks, but when analyzing HTTP connections, we must assume that traffic is going to the attacker.

### 3.1. Access to Local Resources

While the browser has access to local resources such as keying material, files, the camera, and the microphone, it strictly limits or forbids web servers from accessing those same resources. For instance, while it is possible to produce an HTML form which will allow file upload, a script cannot do so without user consent and in fact cannot even suggest a specific file (e.g., /etc/passwd); the user must explicitly select the file and consent to its upload. [Note: in many cases browsers are explicitly designed to avoid dialogs with the semantics of "click here to bypass security checks", as extensive research [crantor-wolf] shows that users are prone to consent under such circumstances.]

Similarly, while Flash programs (SWFs) [SWF] can access the camera and microphone, they explicitly require that the user consent to that access. In addition, some resources simply cannot be accessed from the browser at all. For instance, there is no real way to run specific executables directly from a script (though the user can of course be induced to download executable files and run them).

### 3.2. Same-Origin Policy

Many other resources are accessible but isolated. For instance, while scripts are allowed to make HTTP requests via the XMLHttpRequest() API (see [XmlHttpRequest]) those requests are not

allowed to be made to any server, but rather solely to the same ORIGIN from whence the script came [RFC6454] (although CORS [CORS] and WebSockets [RFC6455] provide an escape hatch from this restriction, as described below.) This SAME ORIGIN POLICY (SOP) prevents server A from mounting attacks on server B via the user's browser, which protects both the user (e.g., from misuse of his credentials) and the server B (e.g., from DoS attack).

More generally, SOP forces scripts from each site to run in their own, isolated, sandboxes. While there are techniques to allow them to interact, those interactions generally must be mutually consensual (by each site) and are limited to certain channels. For instance, multiple pages/browser panes from the same origin can read each other's JS variables, but pages from the different origins--or even iframes from different origins on the same page--cannot.

### 3.3. Bypassing SOP: CORS, WebSockets, and consent to communicate

While SOP serves an important security function, it also makes it inconvenient to write certain classes of applications. In particular, mash-ups, in which a script from origin A uses resources from origin B, can only be achieved via a certain amount of hackery. The W3C Cross-Origin Resource Sharing (CORS) spec [CORS] is a response to this demand. In CORS, when a script from origin A executes what would otherwise be a forbidden cross-origin request, the browser instead contacts the target server to determine whether it is willing to allow cross-origin requests from A. If it is so willing, the browser then allows the request. This consent verification process is designed to safely allow cross-origin requests.

While CORS is designed to allow cross-origin HTTP requests, WebSockets [RFC6455] allows cross-origin establishment of transparent channels. Once a WebSockets connection has been established from a script to a site, the script can exchange any traffic it likes without being required to frame it as a series of HTTP request/response transactions. As with CORS, a WebSockets transaction starts with a consent verification stage to avoid allowing scripts to simply send arbitrary data to another origin.

While consent verification is conceptually simple--just do a handshake before you start exchanging the real data--experience has shown that designing a correct consent verification system is difficult. In particular, Huang et al. [huang-w2sp] have shown vulnerabilities in the existing Java and Flash consent verification techniques and in a simplified version of the WebSockets handshake. In particular, it is important to be wary of CROSS-PROTOCOL attacks in which the attacking script generates traffic which is acceptable

to some non-Web protocol state machine. In order to resist this form of attack, WebSockets incorporates a masking technique intended to randomize the bits on the wire, thus making it more difficult to generate traffic which resembles a given protocol.

#### 4. Security for WebRTC Applications

##### 4.1. Access to Local Devices

As discussed in Section 1, allowing arbitrary sites to initiate calls violates the core Web security guarantee; without some access restrictions on local devices, any malicious site could simply bug a user. At minimum, then, it MUST NOT be possible for arbitrary sites to initiate calls to arbitrary locations without user consent. This immediately raises the question, however, of what should be the scope of user consent.

In order for the user to make an intelligent decision about whether to allow a call (and hence his camera and microphone input to be routed somewhere), he must understand either who is requesting access, where the media is going, or both. As detailed below, there are two basic conceptual models:

1. You are sending your media to entity A because you want to talk to Entity A (e.g., your mother).
2. Entity A (e.g., a calling service) asks to access the user's devices with the assurance that it will transfer the media to entity B (e.g., your mother)

In either case, identity is at the heart of any consent decision. Moreover, the identity of the party the browser is connecting to is all that the browser can meaningfully enforce; if you are calling A, A can simply forward the media to C. Similarly, if you authorize A to place a call to B, A can call C instead. In either cases, all the browser is able to do is verify and check authorization for whoever is controlling where the media goes. The target of the media can of course advertise a security/privacy policy, but this is not something that the browser can enforce. Even so, there are a variety of different consent scenarios that motivate different technical consent mechanisms. We discuss these mechanisms in the sections below.

It's important to understand that consent to access local devices is largely orthogonal to consent to transmit various kinds of data over the network (see Section 4.2). Consent for device access is largely a matter of protecting the user's privacy from malicious sites. By contrast, consent to send network traffic is about preventing the user's browser from being used to attack its local network. Thus, we

need to ensure communications consent even if the site is not able to access the camera and microphone at all (hence WebSockets's consent mechanism) and similarly we need to be concerned with the site accessing the user's camera and microphone even if the data is to be sent back to the site via conventional HTTP-based network mechanisms such as HTTP POST.

#### 4.1.1. Threats from Screen Sharing

In addition to camera and microphone access, there has been demand for screen and/or application sharing functionality. Unfortunately, the security implications of this functionality are much harder for users to intuitively analyze than for camera and microphone access. (See <http://lists.w3.org/Archives/Public/public-webrtc/2013Mar/0024.html> for a full analysis.)

The most obvious threats are simply those of "oversharing". I.e., the user may believe they are sharing a window when in fact they are sharing an application, or may forget they are sharing their whole screen, icons, notifications, and all. This is already an issue with existing screen sharing technologies and is made somewhat worse if a partially trusted site is responsible for asking for the resource to be shared rather than having the user propose it.

A less obvious threat involves the impact of screen sharing on the Web security model. A key part of the Same-Origin Policy is that HTML or JS from site A can reference content from site B and cause the browser to load it, but (unless explicitly permitted) cannot see the result. However, if a web application from a site is screen sharing the browser, then this violates that invariant, with serious security consequences. For example, an attacker site might request screen sharing and then briefly open up a new Window to the user's bank or webmail account, using screen sharing to read the resulting displayed content. A more sophisticated attack would be open up a source view window to a site and use the screen sharing result to view anti cross-site request forgery tokens.

These threats suggest that screen/application sharing might need a higher level of user consent than access to the camera or microphone.

#### 4.1.2. Calling Scenarios and User Expectations

While a large number of possible calling scenarios are possible, the scenarios discussed in this section illustrate many of the difficulties of identifying the relevant scope of consent.

#### 4.1.2.1. Dedicated Calling Services

The first scenario we consider is a dedicated calling service. In this case, the user has a relationship with a calling site and repeatedly makes calls on it. It is likely that rather than having to give permission for each call that the user will want to give the calling service long-term access to the camera and microphone. This is a natural fit for a long-term consent mechanism (e.g., installing an app store "application" to indicate permission for the calling service.) A variant of the dedicated calling service is a gaming site (e.g., a poker site) which hosts a dedicated calling service to allow players to call each other.

With any kind of service where the user may use the same service to talk to many different people, there is a question about whether the user can know who they are talking to. If I grant permission to calling service A to make calls on my behalf, then I am implicitly granting it permission to bug my computer whenever it wants. This suggests another consent model in which a site is authorized to make calls but only to certain target entities (identified via media-plane cryptographic mechanisms as described in Section 4.3.2 and especially Section 4.3.2.3.) Note that the question of consent here is related to but distinct from the question of peer identity: I might be willing to allow a calling site to in general initiate calls on my behalf but still have some calls via that site where I can be sure that the site is not listening in.

#### 4.1.2.2. Calling the Site You're On

Another simple scenario is calling the site you're actually visiting. The paradigmatic case here is the "click here to talk to a representative" windows that appear on many shopping sites. In this case, the user's expectation is that they are calling the site they're actually visiting. However, it is unlikely that they want to provide a general consent to such a site; just because I want some information on a car doesn't mean that I want the car manufacturer to be able to activate my microphone whenever they please. Thus, this suggests the need for a second consent mechanism where I only grant consent for the duration of a given call. As described in Section 3.1, great care must be taken in the design of this interface to avoid the users just clicking through. Note also that the user interface chrome, which is the representation through which the user interacts with the user agent itself, must clearly display elements showing that the call is continuing in order to avoid attacks where the calling site just leaves it up indefinitely but shows a Web UI that implies otherwise.

#### 4.1.3. Origin-Based Security

Now that we have described the calling scenarios, we can start to reason about the security requirements.

As discussed in Section 3.2, the basic unit of Web sandboxing is the origin, and so it is natural to scope consent to origin. Specifically, a script from origin A **MUST** only be allowed to initiate communications (and hence to access camera and microphone) if the user has specifically authorized access for that origin. It is of course technically possible to have coarser-scoped permissions, but because the Web model is scoped to origin, this creates a difficult mismatch.

Arguably, origin is not fine-grained enough. Consider the situation where Alice visits a site and authorizes it to make a single call. If consent is expressed solely in terms of origin, then at any future visit to that site (including one induced via mash-up or ad network), the site can bug Alice's computer, use the computer to place bogus calls, etc. While in principle Alice could grant and then revoke the privilege, in practice privileges accumulate; if we are concerned about this attack, something else is needed. There are a number of potential countermeasures to this sort of issue.

##### Individual Consent

Ask the user for permission for each call.

##### Callee-oriented Consent

Only allow calls to a given user.

##### Cryptographic Consent

Only allow calls to a given set of peer keying material or to a cryptographically established identity.

Unfortunately, none of these approaches is satisfactory for all cases. As discussed above, individual consent puts the user's approval in the UI flow for every call. Not only does this quickly become annoying but it can train the user to simply click "OK", at which point the consent becomes useless. Thus, while it may be necessary to have individual consent in some case, this is not a suitable solution for (for instance) the calling service case. Where

necessary, in-flow user interfaces must be carefully designed to avoid the risk of the user blindly clicking through.

The other two options are designed to restrict calls to a given target. Callee-oriented consent provided by the calling site would not work well because a malicious site can claim that the user is calling any user of his choice. One fix for this is to tie calls to a cryptographically-established identity. While not suitable for all cases, this approach may be useful for some. If we consider the case of advertising, it's not particularly convenient to require the advertiser to instantiate an iframe on the hosting site just to get permission; a more convenient approach is to cryptographically tie the advertiser's certificate to the communication directly. We're still tying permissions to origin here, but to the media origin (and-or destination) rather than to the Web origin.

[I-D.ietf-rtcweb-security-arch] describes mechanisms which facilitate this sort of consent.

Another case where media-level cryptographic identity makes sense is when a user really does not trust the calling site. For instance, I might be worried that the calling service will attempt to bug my computer, but I also want to be able to conveniently call my friends. If consent is tied to particular communications endpoints, then my risk is limited. Naturally, it is somewhat challenging to design UI primitives which express this sort of policy. The problem becomes even more challenging in multi-user calling cases.

#### 4.1.4. Security Properties of the Calling Page

Origin-based security is intended to secure against web attackers. However, we must also consider the case of network attackers. Consider the case where I have granted permission to a calling service by an origin that has the HTTP scheme, e.g., `http://calling-service.example.com`. If I ever use my computer on an unsecured network (e.g., a hotspot or if my own home wireless network is insecure), and browse any HTTP site, then an attacker can bug my computer. The attack proceeds like this:

1. I connect to `http://anything.example.org/`. Note that this site is unaffiliated with the calling service.
2. The attacker modifies my HTTP connection to inject an IFRAME (or a redirect) to `http://calling-service.example.com`
3. The attacker forges the response from `http://calling-service.example.com/` to inject JS to initiate a call to himself.

Note that this attack does not depend on the media being insecure. Because the call is to the attacker, it is also encrypted to him. Moreover, it need not be executed immediately; the attacker can "infect" the origin semi-permanently (e.g., with a web worker or a popped-up window that is hidden under the main window.) and thus be able to bug me long after I have left the infected network. This risk is created by allowing calls at all from a page fetched over HTTP.

Even if calls are only possible from HTTPS [RFC2818] sites, if those sites include active content (e.g., JavaScript) from an untrusted site, that JavaScript is executed in the security context of the page [finer-grained]. This could lead to compromise of a call even if the parent page is safe. Note: this issue is not restricted to PAGES which contain untrusted content. If any page from a given origin ever loads JavaScript from an attacker, then it is possible for that attacker to infect the browser's notion of that origin semi-permanently.

#### 4.2. Communications Consent Verification

As discussed in Section 3.3, allowing web applications unrestricted network access via the browser introduces the risk of using the browser as an attack platform against machines which would not otherwise be accessible to the malicious site, for instance because they are topologically restricted (e.g., behind a firewall or NAT). In order to prevent this form of attack as well as cross-protocol attacks it is important to require that the target of traffic explicitly consent to receiving the traffic in question. Until that consent has been verified for a given endpoint, traffic other than the consent handshake MUST NOT be sent to that endpoint.

Note that consent verification is not sufficient to prevent overuse of network resources. Because WebRTC allows for a Web site to create data flows between two browser instances without user consent, it is possible for a malicious site to chew up a significant amount of a user's bandwidth without incurring significant costs to himself by setting up such a channel to another user. However, as a practical matter there are a large number of Web sites which can act as data sources, so an attacker can at least use downlink bandwidth with existing Web APIs. However, this potential DoS vector reinforces the need for adequate congestion control for WebRTC protocols to ensure that they play fair with other demands on the user's bandwidth.



#### 4.2.1. ICE

Verifying receiver consent requires some sort of explicit handshake, but conveniently we already need one in order to do NAT hole-punching. Interactive Connectivity Establishment (ICE) [RFC8445] includes a handshake designed to verify that the receiving element wishes to receive traffic from the sender. It is important to remember here that the site initiating ICE is presumed malicious; in order for the handshake to be secure the receiving element **MUST** demonstrate receipt/knowledge of some value not available to the site (thus preventing the site from forging responses). In order to achieve this objective with ICE, the STUN transaction IDs must be generated by the browser and **MUST NOT** be made available to the initiating script, even via a diagnostic interface. Verifying receiver consent also requires verifying the receiver wants to receive traffic from a particular sender, and at this time; for example a malicious site may simply attempt ICE to known servers that are using ICE for other sessions. ICE provides this verification as well, by using the STUN credentials as a form of per-session shared secret. Those credentials are known to the Web application, but would need to also be known and used by the STUN-receiving element to be useful.

There also needs to be some mechanism for the browser to verify that the target of the traffic continues to wish to receive it. Because ICE keepalives are indications, they will not work here. [RFC7675] describes the mechanism for providing consent freshness.

#### 4.2.2. Masking

Once consent is verified, there still is some concern about misinterpretation attacks as described by Huang et al.[huang-w2sp]. Where TCP is used the risk is substantial due to the potential presence of transparent proxies and therefore if TCP is to be used, then WebSockets style masking **MUST** be employed.

Since DTLS (with the anti-chosen plaintext mechanisms required by TLS 1.1) does not allow the attacker to generate predictable ciphertext, there is no need for masking of protocols running over DTLS (e.g. SCTP over DTLS, UDP over DTLS, etc.).

Note that in principle an attacker could exert some control over SRTP packets by using a combination of the WebAudio API and extremely tight timing control. The primary risk here seems to be carriage of SRTP over TURN TCP. However, as SRTP packets have an extremely characteristic packet header it seems unlikely that any but the most aggressive intermediaries would be confused into thinking that another application layer protocol was in use.

#### 4.2.3. Backward Compatibility

A requirement to use ICE limits compatibility with legacy non-ICE clients. It seems unsafe to completely remove the requirement for some check. All proposed checks have the common feature that the browser sends some message to the candidate traffic recipient and refuses to send other traffic until that message has been replied to. The message/reply pair must be generated in such a way that an attacker who controls the Web application cannot forge them, generally by having the message contain some secret value that must be incorporated (e.g., echoed, hashed into, etc.). Non-ICE candidates for this role (in cases where the legacy endpoint has a public address) include:

- o STUN checks without using ICE (i.e., the non-RTC-web endpoint sets up a STUN responder.)
- o Use of RTCP as an implicit reachability check.

In the RTCP approach, the WebRTC endpoint is allowed to send a limited number of RTP packets prior to receiving consent. This allows a short window of attack. In addition, some legacy endpoints do not support RTCP, so this is a much more expensive solution for such endpoints, for which it would likely be easier to implement ICE. For these two reasons, an RTCP-based approach does not seem to address the security issue satisfactorily.

In the STUN approach, the WebRTC endpoint is able to verify that the recipient is running some kind of STUN endpoint but unless the STUN responder is integrated with the ICE username/password establishment system, the WebRTC endpoint cannot verify that the recipient consents to this particular call. This may be an issue if existing STUN servers are operated at addresses that are not able to handle bandwidth-based attacks. Thus, this approach does not seem satisfactory either.

If the systems are tightly integrated (i.e., the STUN endpoint responds with responses authenticated with ICE credentials) then this issue does not exist. However, such a design is very close to an ICE-Lite implementation (indeed, arguably is one). An intermediate approach would be to have a STUN extension that indicated that one was responding to WebRTC checks but not computing integrity checks based on the ICE credentials. This would allow the use of standalone STUN servers without the risk of confusing them with legacy STUN servers. If a non-ICE legacy solution is needed, then this is probably the best choice.

Once initial consent is verified, we also need to verify continuing consent, in order to avoid attacks where two people briefly share an IP (e.g., behind a NAT in an Internet cafe) and the attacker arranges for a large, unstoppable, traffic flow to the network and then leaves. The appropriate technologies here are fairly similar to those for initial consent, though are perhaps weaker since the threats are less severe.

#### 4.2.4. IP Location Privacy

Note that as soon as the callee sends their ICE candidates, the caller learns the callee's IP addresses. The callee's server reflexive address reveals a lot of information about the callee's location. In order to avoid tracking, implementations may wish to suppress the start of ICE negotiation until the callee has answered. In addition, either side may wish to hide their location from the other side entirely by forcing all traffic through a TURN server.

In ordinary operation, the site learns the browser's IP address, though it may be hidden via mechanisms like Tor [<http://www.torproject.org>] or a VPN. However, because sites can cause the browser to provide IP addresses, this provides a mechanism for sites to learn about the user's network environment even if the user is behind a VPN that masks their IP address. Implementations may wish to provide settings which suppress all non-VPN candidates if the user is on certain kinds of VPN, especially privacy-oriented systems such as Tor. See [I-D.ietf-rtcweb-ip-handling] for additional information.

#### 4.3. Communications Security

Finally, we consider a problem familiar from the SIP world: communications security. For obvious reasons, it **MUST** be possible for the communicating parties to establish a channel which is secure against both message recovery and message modification. (See [RFC5479] for more details.) This service must be provided for both data and voice/video. Ideally the same security mechanisms would be used for both types of content. Technology for providing this service (for instance, SRTP [RFC3711], DTLS [RFC6347] and DTLS-SRTP [RFC5763]) is well understood. However, we must examine this technology in the WebRTC context, where the threat model is somewhat different.

In general, it is important to understand that unlike a conventional SIP proxy, the calling service (i.e., the Web server) controls not only the channel between the communicating endpoints but also the application running on the user's browser. While in principle it is possible for the browser to cut the calling service out of the loop

and directly present trusted information (and perhaps get consent), practice in modern browsers is to avoid this whenever possible. "In-flow" modal dialogs which require the user to consent to specific actions are particularly disfavored as human factors research indicates that unless they are made extremely invasive, users simply agree to them without actually consciously giving consent. [abarth-rtcweb]. Thus, nearly all the UI will necessarily be rendered by the browser but under control of the calling service. This likely includes the peer's identity information, which, after all, is only meaningful in the context of some calling service.

This limitation does not mean that preventing attack by the calling service is completely hopeless. However, we need to distinguish between two classes of attack:

Retrospective compromise of calling service.

The calling service is non-malicious during a call but subsequently is compromised and wishes to attack an older call (often called a "passive attack")

During-call attack by calling service.

The calling service is compromised during the call it wishes to attack (often called an "active attack").

Providing security against the former type of attack is practical using the techniques discussed in Section 4.3.1. However, it is extremely difficult to prevent a trusted but malicious calling service from actively attacking a user's calls, either by mounting a Man-in-the-Middle (MITM) attack or by diverting them entirely. (Note that this attack applies equally to a network attacker if communications to the calling service are not secured.) We discuss some potential approaches and why they are likely to be impractical in Section 4.3.2.

#### 4.3.1. Protecting Against Retrospective Compromise

In a retrospective attack, the calling service was uncompromised during the call, but that an attacker subsequently wants to recover the content of the call. We assume that the attacker has access to the protected media stream as well as having full control of the calling service.

If the calling service has access to the traffic keying material (as in SDES [RFC4568]), then retrospective attack is trivial. This form

of attack is particularly serious in the Web context because it is standard practice in Web services to run extensive logging and monitoring. Thus, it is highly likely that if the traffic key is part of any HTTP request it will be logged somewhere and thus subject to subsequent compromise. It is this consideration that makes an automatic, public key-based key exchange mechanism imperative for WebRTC (this is a good idea for any communications security system) and this mechanism SHOULD provide perfect forward secrecy (PFS). The signaling channel/calling service can be used to authenticate this mechanism.

In addition, if end-to-end keying is in used, the system MUST NOT provide any APIs to extract either long-term keying material or to directly access any stored traffic keys. Otherwise, an attacker who subsequently compromised the calling service might be able to use those APIs to recover the traffic keys and thus compromise the traffic.

#### 4.3.2. Protecting Against During-Call Attack

Protecting against attacks during a call is a more difficult proposition. Even if the calling service cannot directly access keying material (as recommended in the previous section), it can simply mount a man-in-the-middle attack on the connection, telling Alice that she is calling Bob and Bob that he is calling Alice, while in fact the calling service is acting as a calling bridge and capturing all the traffic. Protecting against this form of attack requires positive authentication of the remote endpoint such as explicit out-of-band key verification (e.g., by a fingerprint) or a third-party identity service as described in [I-D.ietf-rtcweb-security-arch].

##### 4.3.2.1. Key Continuity

One natural approach is to use "key continuity". While a malicious calling service can present any identity it chooses to the user, it cannot produce a private key that maps to a given public key. Thus, it is possible for the browser to note a given user's public key and generate an alarm whenever that user's key changes. SSH [RFC4251] uses a similar technique. (Note that the need to avoid explicit user consent on every call precludes the browser requiring an immediate manual check of the peer's key).

Unfortunately, this sort of key continuity mechanism is far less useful in the WebRTC context. First, much of the virtue of WebRTC (and any Web application) is that it is not bound to particular piece of client software. Thus, it will be not only possible but routine for a user to use multiple browsers on different computers which will

of course have different keying material (SACRED [RFC3760] notwithstanding.) Thus, users will frequently be alerted to key mismatches which are in fact completely legitimate, with the result that they are trained to simply click through them. As it is known that users routinely will click through far more dire warnings [cranor-wolf], it seems extremely unlikely that any key continuity mechanism will be effective rather than simply annoying.

Moreover, it is trivial to bypass even this kind of mechanism. Recall that unlike the case of SSH, the browser never directly gets the peer's identity from the user. Rather, it is provided by the calling service. Even enabling a mechanism of this type would require an API to allow the calling service to tell the browser "this is a call to user X". All the calling service needs to do to avoid triggering a key continuity warning is to tell the browser that "this is a call to user Y" where Y is confusable with X. Even if the user actually checks the other side's name (which all available evidence indicates is unlikely), this would require (a) the browser to use the trusted UI to provide the name and (b) the user to not be fooled by similar appearing names.

#### 4.3.2.2. Short Authentication Strings

ZRTP [RFC6189] uses a "short authentication string" (SAS) which is derived from the key agreement protocol. This SAS is designed to be compared by the users (e.g., read aloud over the voice channel or transmitted via an out of band channel) and if confirmed by both sides precludes MITM attack. The intention is that the SAS is used once and then key continuity (though a different mechanism from that discussed above) is used thereafter.

Unfortunately, the SAS does not offer a practical solution to the problem of a compromised calling service. "Voice conversion" systems, which modify voice from one speaker to make it sound like another, are an active area of research. These systems are already good enough to fool both automatic recognition systems [farus-conversion] and humans [kain-conversion] in many cases, and are of course likely to improve in future, especially in an environment where the user just wants to get on with the phone call. Thus, even if SAS is effective today, it is likely not to be so for much longer.

Additionally, it is unclear that users will actually use an SAS. As discussed above, the browser UI constraints preclude requiring the SAS exchange prior to completing the call and so it must be voluntary; at most the browser will provide some UI indicator that the SAS has not yet been checked. However, it is well-known that

when faced with optional security mechanisms, many users simply ignore them [whitten-johnny].

Once users have checked the SAS once, key continuity is required to avoid them needing to check it on every call. However, this is problematic for reasons indicated in Section 4.3.2.1. In principle it is of course possible to render a different UI element to indicate that calls are using an unauthenticated set of keying material (recall that the attacker can just present a slightly different name so that the attack shows the same UI as a call to a new device or to someone you haven't called before) but as a practical matter, users simply ignore such indicators even in the rather more dire case of mixed content warnings.

#### 4.3.2.3. Third Party Identity

The conventional approach to providing communications identity has of course been to have some third party identity system (e.g., PKI) to authenticate the endpoints. Such mechanisms have proven to be too cumbersome for use by typical users (and nearly too cumbersome for administrators). However, a new generation of Web-based identity providers (BrowserID, Federated Google Login, Facebook Connect, OAuth [RFC6749], OpenID [OpenID], WebFinger [RFC7033]), has recently been developed and use Web technologies to provide lightweight (from the user's perspective) third-party authenticated transactions. It is possible to use systems of this type to authenticate WebRTC calls, linking them to existing user notions of identity (e.g., Facebook adjacencies). Specifically, the third-party identity system is used to bind the user's identity to cryptographic keying material which is then used to authenticate the calling endpoints. Calls which are authenticated in this fashion are naturally resistant even to active MITM attack by the calling site.

Note that there is one special case in which PKI-style certificates do provide a practical solution: calls from end-users to large sites. For instance, if you are making a call to Amazon.com, then Amazon can easily get a certificate to authenticate their media traffic, just as they get one to authenticate their Web traffic. This does not provide additional security value in cases in which the calling site and the media peer are one in the same, but might be useful in cases in which third parties (e.g., ad networks or retailers) arrange for calls but do not participate in them.

#### 4.3.2.4. Page Access to Media

Identifying the identity of the far media endpoint is a necessary but not sufficient condition for providing media security. In WebRTC, media flows are rendered into HTML5 MediaStreams which can be

manipulated by the calling site. Obviously, if the site can modify or view the media, then the user is not getting the level of assurance they would expect from being able to authenticate their peer. In many cases, this is acceptable because the user values site-based special effects over complete security from the site. However, there are also cases where users wish to know that the site cannot interfere. In order to facilitate that, it will be necessary to provide features whereby the site can verifiably give up access to the media streams. This verification must be possible both from the local side and the remote side. I.e., users must be able to verify that the person called has engaged a secure media mode (see Section 4.3.3). In order to achieve this it will be necessary to cryptographically bind an indication of the local media access policy into the cryptographic authentication procedures detailed in the previous sections.

It should be noted that the use of this secure media mode is left to the discretion of the site. When such a mode is engaged, the browser will need to provide indicia to the user that the associated media has been authenticated as coming from the identified user. This allows WebRTC services that wish to claim end-to-end security to do so in a way that can be easily verified by the user. This model requires that the remote party's browser be included in the TCB, as described in Section 3.

#### 4.3.3. Malicious Peers

One class of attack that we do not generally try to prevent is malicious peers. For instance, no matter what confidentiality measures you employ the person you are talking to might record the call and publish it on the Internet. Similarly, we do not attempt to prevent them from using voice or video processing technology from hiding or changing their appearance. While technologies (DRM, etc.) do exist to attempt to address these issues, they are generally not compatible with open systems and WebRTC does not address them.

Similarly, we make no attempt to prevent prank calling or other unwanted calls. In general, this is in the scope of the calling site, though because WebRTC does offer some forms of strong authentication, that may be useful as part of a defense against such attacks.

#### 4.4. Privacy Considerations



#### 4.4.1. Correlation of Anonymous Calls

While persistent endpoint identifiers can be a useful security feature (see Section 4.3.2.1) they can also represent a privacy threat in settings where the user wishes to be anonymous. WebRTC provides a number of possible persistent identifiers such as DTLS certificates (if they are reused between connections) and RTCP CNAMEs (if generated according to [RFC6222] rather than the privacy preserving mode of [RFC7022]). In order to prevent this type of correlation, browsers need to provide mechanisms to reset these identifiers (e.g., with the same lifetime as cookies). Moreover, the API should provide mechanisms to allow sites intended for anonymous calling to force the minting of fresh identifiers. In addition, IP addresses can be a source of call linkage [I-D.ietf-rtcweb-ip-handling].

#### 4.4.2. Browser Fingerprinting

Any new set of API features adds a risk of browser fingerprinting, and WebRTC is no exception. Specifically, sites can use the presence or absence of specific devices as a browser fingerprint. In general, the API needs to be balanced between functionality and the incremental fingerprint risk. See [Fingerprinting].

### 5. Security Considerations

This entire document is about security.

### 6. Acknowledgements

Bernard Aboba, Harald Alvestrand, Dan Druta, Cullen Jennings, Alan Johnston, Hadriel Kaplan (S 4.2.1), Matthew Kaufman, Martin Thomson, Magnus Westerlund.

### 7. IANA Considerations

There are no IANA considerations.

### 8. Changes Since -04

- o Replaced RTCWEB and RTC-Web with WebRTC, except when referring to the IETF WG
- o Removed discussion of the IFRAMED advertisement case, since we decided not to treat it specially.
- o Added a privacy section considerations section.

- o Significant edits to the SAS section to reflect Alan Johnston's comments.
- o Added some discussion if IP location privacy and Tor.
- o Updated the "communications consent" section to reflrect draft-ietf.
- o Added a section about "malicious peers".
- o Added a section describing screen sharing threats.
- o Assorted editorial changes.

## 9. References

### 9.1. 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>>.
- [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>>.

### 9.2. Informative References

- [abarth-rtcweb] Barth, A., "Prompting the user is security failure", RTC-Web Workshop, September 2010.
- [CORS] van Kesteren, A., "Cross-Origin Resource Sharing", January 2014.
- [cranor-wolf] Sunshine, J., Egelman, S., Almuhiemedi, H., Atri, N., and L. cranor, "Crying Wolf: An Empirical Study of SSL Warning Effectiveness", Proceedings of the 18th USENIX Security Symposium, 2009, August 2009.
- [farus-conversion] Farrus, M., Erro, D., and J. Hernando, "Speaker Recognition Robustness to Voice Conversion", January 2008.

- [finer-grained]  
Barth, A. and C. Jackson, "Beware of Finer-Grained Origins", W2SP, 2008, July 2008.
- [Fingerprinting]  
"Fingerprinting Guidance for Web Specification Authors (Draft)", November 2013.
- [huang-w2sp]  
Huang, L-S., Chen, E., Barth, A., Rescorla, E., and C. Jackson, "Talking to Yourself for Fun and Profit", W2SP, 2011, May 2011.
- [I-D.ietf-rtcweb-ip-handling]  
Uberti, J., "WebRTC IP Address Handling Requirements", draft-ietf-rtcweb-ip-handling-12 (work in progress), July 2019.
- [I-D.ietf-rtcweb-overview]  
Alvestrand, H., "Overview: Real Time Protocols for Browser-based Applications", draft-ietf-rtcweb-overview-19 (work in progress), November 2017.
- [I-D.ietf-rtcweb-security-arch]  
Rescorla, E., "WebRTC Security Architecture", draft-ietf-rtcweb-security-arch-18 (work in progress), February 2019.
- [kain-conversion]  
Kain, A. and M. Macon, "Design and Evaluation of a Voice Conversion Algorithm based on Spectral Envelope Mapping and Residual Prediction", Proceedings of ICASSP, May 2001, May 2001.
- [OpenID]  
Sakimura, N., Bradley, J., Jones, M., de Medeiros, B., and C. Mortimore, "OpenID Connect Core 1.0", November 2014.
- [RFC2818]  
Rescorla, E., "HTTP Over TLS", RFC 2818, DOI 10.17487/RFC2818, May 2000, <<https://www.rfc-editor.org/info/rfc2818>>.
- [RFC3261]  
Rosenberg, J., Schulzrinne, H., Camarillo, G., Johnston, A., Peterson, J., Sparks, R., Handley, M., and E. Schooler, "SIP: Session Initiation Protocol", RFC 3261, DOI 10.17487/RFC3261, June 2002, <<https://www.rfc-editor.org/info/rfc3261>>.

- [RFC3552] Rescorla, E. and B. Korver, "Guidelines for Writing RFC Text on Security Considerations", BCP 72, RFC 3552, DOI 10.17487/RFC3552, July 2003, <<https://www.rfc-editor.org/info/rfc3552>>.
- [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>>.
- [RFC3760] Gustafson, D., Just, M., and M. Nystrom, "Securely Available Credentials (SACRED) - Credential Server Framework", RFC 3760, DOI 10.17487/RFC3760, April 2004, <<https://www.rfc-editor.org/info/rfc3760>>.
- [RFC4251] Ylonen, T. and C. Lonvick, Ed., "The Secure Shell (SSH) Protocol Architecture", RFC 4251, DOI 10.17487/RFC4251, January 2006, <<https://www.rfc-editor.org/info/rfc4251>>.
- [RFC4568] Andreasen, F., Baugher, M., and D. Wing, "Session Description Protocol (SDP) Security Descriptions for Media Streams", RFC 4568, DOI 10.17487/RFC4568, July 2006, <<https://www.rfc-editor.org/info/rfc4568>>.
- [RFC5479] Wing, D., Ed., Fries, S., Tschofenig, H., and F. Audet, "Requirements and Analysis of Media Security Management Protocols", RFC 5479, DOI 10.17487/RFC5479, April 2009, <<https://www.rfc-editor.org/info/rfc5479>>.
- [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>>.
- [RFC6189] Zimmermann, P., Johnston, A., Ed., and J. Callas, "ZRTP: Media Path Key Agreement for Unicast Secure RTP", RFC 6189, DOI 10.17487/RFC6189, April 2011, <<https://www.rfc-editor.org/info/rfc6189>>.
- [RFC6222] Begen, A., Perkins, C., and D. Wing, "Guidelines for Choosing RTP Control Protocol (RTCP) Canonical Names (CNAMEs)", RFC 6222, DOI 10.17487/RFC6222, April 2011, <<https://www.rfc-editor.org/info/rfc6222>>.
- [RFC6347] Rescorla, E. and N. Modadugu, "Datagram Transport Layer Security Version 1.2", RFC 6347, DOI 10.17487/RFC6347, January 2012, <<https://www.rfc-editor.org/info/rfc6347>>.

- [RFC6454] Barth, A., "The Web Origin Concept", RFC 6454, DOI 10.17487/RFC6454, December 2011, <<https://www.rfc-editor.org/info/rfc6454>>.
- [RFC6455] Fette, I. and A. Melnikov, "The WebSocket Protocol", RFC 6455, DOI 10.17487/RFC6455, December 2011, <<https://www.rfc-editor.org/info/rfc6455>>.
- [RFC6749] Hardt, D., Ed., "The OAuth 2.0 Authorization Framework", RFC 6749, DOI 10.17487/RFC6749, October 2012, <<https://www.rfc-editor.org/info/rfc6749>>.
- [RFC7022] Begen, A., Perkins, C., Wing, D., and E. Rescorla, "Guidelines for Choosing RTP Control Protocol (RTCP) Canonical Names (CNAMEs)", RFC 7022, DOI 10.17487/RFC7022, September 2013, <<https://www.rfc-editor.org/info/rfc7022>>.
- [RFC7033] Jones, P., Salgueiro, G., Jones, M., and J. Smarr, "WebFinger", RFC 7033, DOI 10.17487/RFC7033, September 2013, <<https://www.rfc-editor.org/info/rfc7033>>.
- [RFC7675] Perumal, M., Wing, D., Ravindranath, R., Reddy, T., and M. Thomson, "Session Traversal Utilities for NAT (STUN) Usage for Consent Freshness", RFC 7675, DOI 10.17487/RFC7675, October 2015, <<https://www.rfc-editor.org/info/rfc7675>>.
- [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>>.
- [SWF] "SWF File Format Specification Version 19", April 2013.
- [whitten-johnny] Whitten, A. and J. Tygar, "Why Johnny Can't Encrypt: A Usability Evaluation of PGP 5.0", Proceedings of the 8th USENIX Security Symposium, 1999, August 1999.
- [XmlHttpRequest] van Kesteren, A., "XMLHttpRequest Level 2", January 2012.

Author's Address

Eric Rescorla  
RTFM, Inc.  
2064 Edgewood Drive  
Palo Alto, CA 94303  
USA

Phone: +1 650 678 2350  
Email: [ekr@rtfm.com](mailto:ekr@rtfm.com)

RTCWEB  
Internet-Draft  
Intended status: Standards Track  
Expires: January 22, 2020

E. Rescorla  
RTFM, Inc.  
July 21, 2019

WebRTC Security Architecture  
draft-ietf-rtcweb-security-arch-20

Abstract

This document defines the security architecture for WebRTC, a protocol suite intended for use with real-time applications that can be deployed in browsers - "real time communication on the Web".

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 22, 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.

This document may contain material from IETF Documents or IETF Contributions published or made publicly available before November 10, 2008. The person(s) controlling the copyright in some of this material may not have granted the IETF Trust the right to allow modifications of such material outside the IETF Standards Process. Without obtaining an adequate license from the person(s) controlling the copyright in such materials, this document may not be modified outside the IETF Standards Process, and derivative works of it may not be created outside the IETF Standards Process, except to format it for publication as an RFC or to translate it into languages other than English.

## Table of Contents

1. Introduction . . . . .	3
2. Terminology . . . . .	5
3. Trust Model . . . . .	5
3.1. Authenticated Entities . . . . .	5
3.2. Unauthenticated Entities . . . . .	6
4. Overview . . . . .	6
4.1. Initial Signaling . . . . .	8
4.2. Media Consent Verification . . . . .	10
4.3. DTLS Handshake . . . . .	11
4.4. Communications and Consent Freshness . . . . .	11
5. SDP Identity Attribute . . . . .	12
5.1. Offer/Answer Considerations . . . . .	13
5.1.1. Generating the Initial SDP Offer . . . . .	13
5.1.2. Generating of SDP Answer . . . . .	14
5.1.3. Processing an SDP Offer or Answer . . . . .	14
5.1.4. Modifying the Session . . . . .	14
6. Detailed Technical Description . . . . .	14
6.1. Origin and Web Security Issues . . . . .	14
6.2. Device Permissions Model . . . . .	15
6.3. Communications Consent . . . . .	17
6.4. IP Location Privacy . . . . .	17
6.5. Communications Security . . . . .	18
7. Web-Based Peer Authentication . . . . .	20
7.1. Trust Relationships: IdPs, APs, and RPs . . . . .	21
7.2. Overview of Operation . . . . .	23
7.3. Items for Standardization . . . . .	24
7.4. Binding Identity Assertions to JSEP Offer/Answer Transactions . . . . .	24
7.4.1. Carrying Identity Assertions . . . . .	25
7.5. Determining the IdP URI . . . . .	26
7.5.1. Authenticating Party . . . . .	27
7.5.2. Relying Party . . . . .	28
7.6. Requesting Assertions . . . . .	28
7.7. Managing User Login . . . . .	29



8. Verifying Assertions . . . . .	29
8.1. Identity Formats . . . . .	30
9. Security Considerations . . . . .	31
9.1. Communications Security . . . . .	31
9.2. Privacy . . . . .	32
9.3. Denial of Service . . . . .	33
9.4. IdP Authentication Mechanism . . . . .	34
9.4.1. PeerConnection Origin Check . . . . .	34
9.4.2. IdP Well-known URI . . . . .	34
9.4.3. Privacy of IdP-generated identities and the hosting site . . . . .	35
9.4.4. Security of Third-Party IdPs . . . . .	35
9.4.4.1. Confusable Characters . . . . .	35
9.4.5. Web Security Feature Interactions . . . . .	35
9.4.5.1. Popup Blocking . . . . .	36
9.4.5.2. Third Party Cookies . . . . .	36
10. IANA Considerations . . . . .	36
11. Acknowledgements . . . . .	37
12. Changes . . . . .	37
12.1. Changes since -15 . . . . .	37
12.2. Changes since -11 . . . . .	37
12.3. Changes since -10 . . . . .	37
12.4. Changes since -06 . . . . .	37
12.5. Changes since -05 . . . . .	38
12.6. Changes since -03 . . . . .	38
12.7. Changes since -03 . . . . .	38
12.8. Changes since -02 . . . . .	38
13. References . . . . .	39
13.1. Normative References . . . . .	39
13.2. Informative References . . . . .	42
Author's Address . . . . .	43

## 1. Introduction

The Real-Time Communications on the Web (RTCWEB) working group standardized protocols for real-time communications between Web browsers, generally called "WebRTC" [I-D.ietf-rtcweb-overview]. The major use cases for WebRTC technology are real-time audio and/or video calls, Web conferencing, and direct data transfer. Unlike most conventional real-time systems, (e.g., SIP-based [RFC3261] soft phones) WebRTC communications are directly controlled by some Web server, via a JavaScript (JS) API as shown in Figure 1.

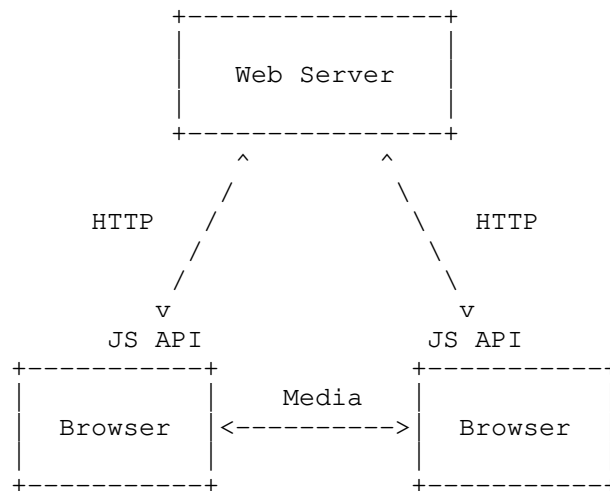


Figure 1: A simple WebRTC system

A more complicated system might allow for interdomain calling, as shown in Figure 2. The protocol to be used between the domains is not standardized by WebRTC, but given the installed base and the form of the WebRTC API is likely to be something SDP-based like SIP or something like Extensible Messaging and Presence Protocol (XMPP) [RFC6120].

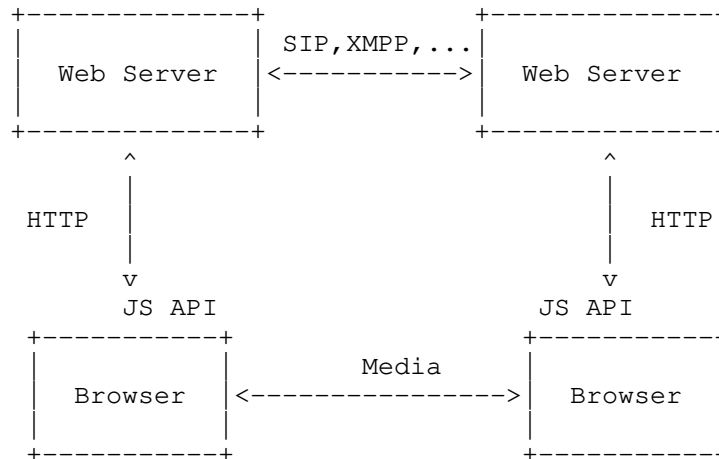


Figure 2: A multidomain WebRTC system

This system presents a number of new security challenges, which are analyzed in [I-D.ietf-rtcweb-security]. This document describes a

security architecture for WebRTC which addresses the threats and requirements described in that document.

## 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. Trust Model

The basic assumption of this architecture is that network resources exist in a hierarchy of trust, rooted in the browser, which serves as the user's Trusted Computing Base (TCB). Any security property which the user wishes to have enforced must be ultimately guaranteed by the browser (or transitively by some property the browser verifies). Conversely, if the browser is compromised, then no security guarantees are possible. Note that there are cases (e.g., Internet kiosks) where the user can't really trust the browser that much. In these cases, the level of security provided is limited by how much they trust the browser.

Optimally, we would not rely on trust in any entities other than the browser. However, this is unfortunately not possible if we wish to have a functional system. Other network elements fall into two categories: those which can be authenticated by the browser and thus can be granted permissions to access sensitive resources, and those which cannot be authenticated and thus are untrusted.

### 3.1. Authenticated Entities

There are two major classes of authenticated entities in the system:

- o Calling services: Web sites whose origin we can verify (optimally via HTTPS, but in some cases because we are on a topologically restricted network, such as behind a firewall, and can infer authentication from firewall behavior).
- o Other users: WebRTC peers whose origin we can verify cryptographically (optimally via DTLS-SRTP).

Note that merely being authenticated does not make these entities trusted. For instance, just because we can verify that <https://www.example.org/> is owned by Dr. Evil does not mean that we can trust Dr. Evil to access our camera and microphone. However, it gives the user an opportunity to determine whether he wishes to trust

Dr. Evil or not; after all, if he desires to contact Dr. Evil (perhaps to arrange for ransom payment), it's safe to temporarily give him access to the camera and microphone for the purpose of the call, but he doesn't want Dr. Evil to be able to access his camera and microphone other than during the call. The point here is that we must first identify other elements before we can determine whether and how much to trust them. Additionally, sometimes we need to identify the communicating peer before we know what policies to apply.

### 3.2. Unauthenticated Entities

Other than the above entities, we are not generally able to identify other network elements, thus we cannot trust them. This does not mean that it is not possible to have any interaction with them, but it means that we must assume that they will behave maliciously and design a system which is secure even if they do so.

## 4. Overview

This section describes a typical WebRTC session and shows how the various security elements interact and what guarantees are provided to the user. The example in this section is a "best case" scenario in which we provide the maximal amount of user authentication and media privacy with the minimal level of trust in the calling service. Simpler versions with lower levels of security are also possible and are noted in the text where applicable. It's also important to recognize the tension between security (or performance) and privacy. The example shown here is aimed towards settings where we are more concerned about secure calling than about privacy, but as we shall see, there are settings where one might wish to make different tradeoffs--this architecture is still compatible with those settings.

For the purposes of this example, we assume the topology shown in the figures below. This topology is derived from the topology shown in Figure 1, but separates Alice and Bob's identities from the process of signaling. Specifically, Alice and Bob have relationships with some Identity Provider (IdP) that supports a protocol (such as OpenID Connect) that can be used to demonstrate their identity to other parties. For instance, Alice might have an account with a social network which she can then use to authenticate to other web sites without explicitly having an account with those sites; this is a fairly conventional pattern on the Web. Section 7.1 provides an overview of Identity Providers and the relevant terminology. Alice and Bob might have relationships with different IdPs as well.

This separation of identity provision and signaling isn't particularly important in "closed world" cases where Alice and Bob

are users on the same social network and have identities based on that domain (Figure 3). However, there are important settings where that is not the case, such as federation (calls from one domain to another; Figure 4) and calling on untrusted sites, such as where two users who have a relationship via a given social network want to call each other on another, untrusted, site, such as a poker site.

Note that the servers themselves are also authenticated by an external identity service, the SSL/TLS certificate infrastructure (not shown). As is conventional in the Web, all identities are ultimately rooted in that system. For instance, when an IdP makes an identity assertion, the Relying Party consuming that assertion is able to verify because it is able to connect to the IdP via HTTPS.

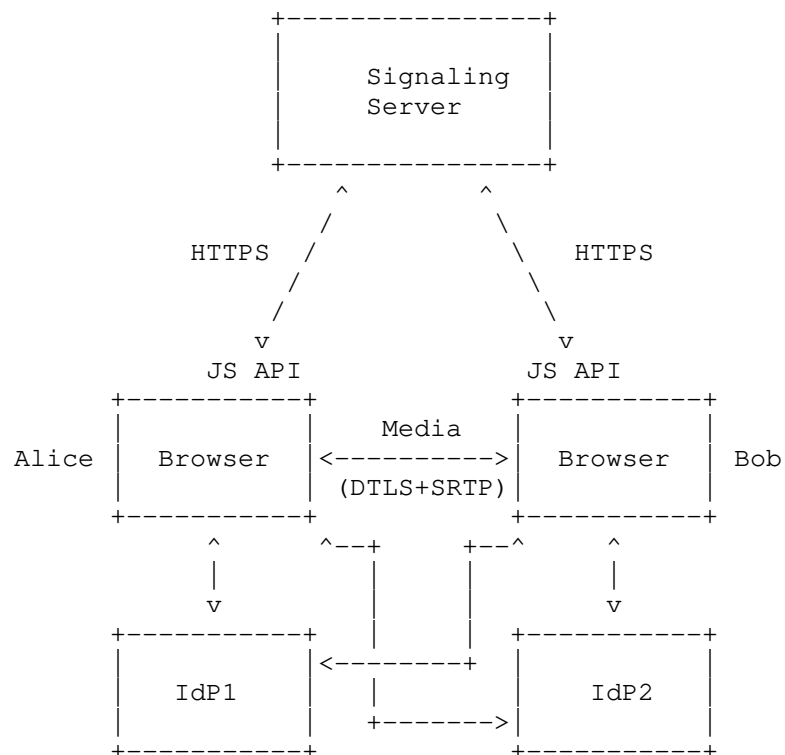


Figure 3: A call with IdP-based identity

Figure 4 shows essentially the same calling scenario but with a call between two separate domains (i.e., a federated case), as in Figure 2. As mentioned above, the domains communicate by some unspecified protocol and providing separate signaling and identity

allows for calls to be authenticated regardless of the details of the inter-domain protocol.

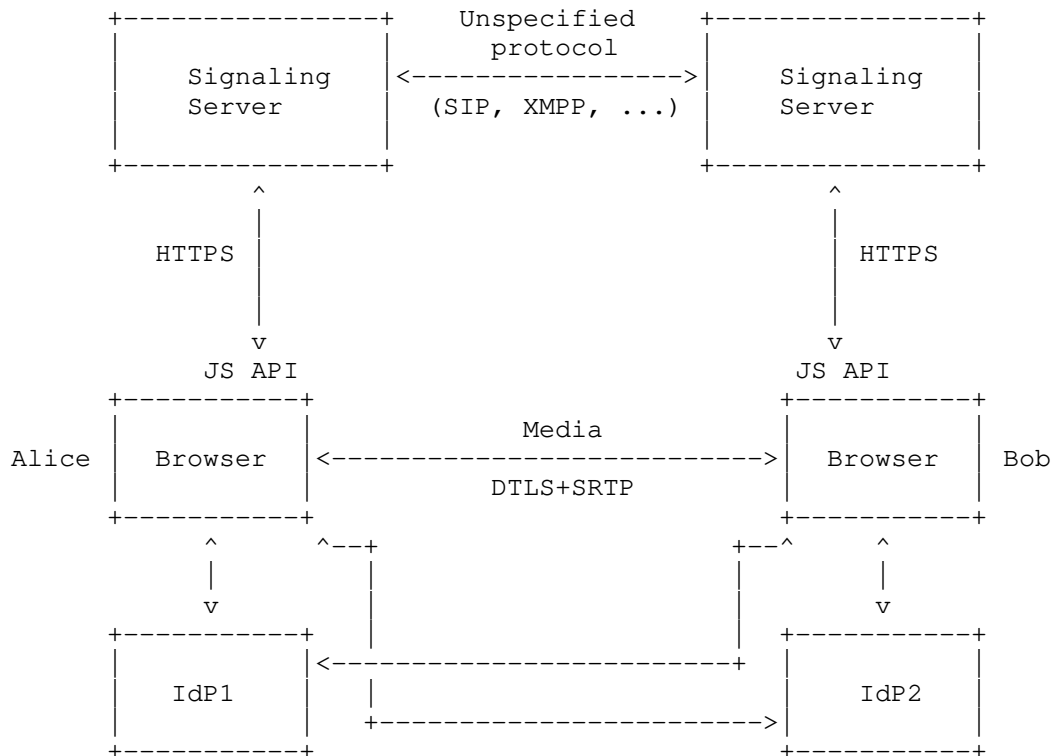


Figure 4: A federated call with IdP-based identity

#### 4.1. Initial Signaling

For simplicity, assume the topology in Figure 3. Alice and Bob are both users of a common calling service; they both have approved the calling service to make calls (we defer the discussion of device access permissions until later). They are both connected to the calling service via HTTPS and so know the origin with some level of confidence. They also have accounts with some identity provider. This sort of identity service is becoming increasingly common in the Web environment (with technologies such as Federated Google Login, Facebook Connect, OAuth, OpenID, WebFinger), and is often provided as a side effect service of a user's ordinary accounts with some service. In this example, we show Alice and Bob using a separate identity service, though the identity service may be the same entity as the calling service or there may be no identity service at all.

Alice is logged onto the calling service and decides to call Bob. She can see from the calling service that he is online and the calling service presents a JS UI in the form of a button next to Bob's name which says "Call". Alice clicks the button, which initiates a JS callback that instantiates a `PeerConnection` object. This does not require a security check: JS from any origin is allowed to get this far.

Once the `PeerConnection` is created, the calling service JS needs to set up some media. Because this is an audio/video call, it creates a `MediaStream` with two `MediaStreamTracks`, one connected to an audio input and one connected to a video input. At this point the first security check is required: untrusted origins are not allowed to access the camera and microphone, so the browser prompts Alice for permission.

In the current W3C API, once some streams have been added, Alice's browser + JS generates a signaling message [I-D.ietf-rtcweb-jsep] containing:

- o Media channel information
- o Interactive Connectivity Establishment (ICE) [RFC8445] candidates
- o A fingerprint attribute binding the communication to a key pair [RFC5763]. Note that this key may simply be ephemerally generated for this call or specific to this domain, and Alice may have a large number of such keys.

Prior to sending out the signaling message, the `PeerConnection` code contacts the identity service and obtains an assertion binding Alice's identity to her fingerprint. The exact details depend on the identity service (though as discussed in Section 7 `PeerConnection` can be agnostic to them), but for now it's easiest to think of as an OAuth token. The assertion may bind other information to the identity besides the fingerprint, but at minimum it needs to bind the fingerprint.

This message is sent to the signaling server, e.g., by `XMLHttpRequest` [`XmlHttpRequest`] or by `WebSockets` [RFC6455], over TLS [RFC5246]. The signaling server processes the message from Alice's browser, determines that this is a call to Bob and sends a signaling message to Bob's browser (again, the format is currently undefined). The JS on Bob's browser processes it, and alerts Bob to the incoming call and to Alice's identity. In this case, Alice has provided an identity assertion and so Bob's browser contacts Alice's identity provider (again, this is done in a generic way so the browser has no specific knowledge of the IdP) to verify the assertion. It is also

possible to have IdPs with which the browser has a specific trust relationship, as described in Section 7.1. This allows the browser to display a trusted element in the browser chrome indicating that a call is coming in from Alice. If Alice is in Bob's address book, then this interface might also include her real name, a picture, etc. The calling site will also provide some user interface element (e.g., a button) to allow Bob to answer the call, though this is most likely not part of the trusted UI.

If Bob agrees a `PeerConnection` is instantiated with the message from Alice's side. Then, a similar process occurs as on Alice's browser: Bob's browser prompts him for device permission, the media streams are created, and a return signaling message containing media information, ICE candidates, and a fingerprint is sent back to Alice via the signaling service. If Bob has a relationship with an IdP, the message will also come with an identity assertion.

At this point, Alice and Bob each know that the other party wants to have a secure call with them. Based purely on the interface provided by the signaling server, they know that the signaling server claims that the call is from Alice to Bob. This level of security is provided merely by having the fingerprint in the message and having that message received securely from the signaling server. Because the far end sent an identity assertion along with their message, they know that this is verifiable from the IdP as well. Note that if the call is federated, as shown in Figure 4 then Alice is able to verify Bob's identity in a way that is not mediated by either her signaling server or Bob's. Rather, she verifies it directly with Bob's IdP.

Of course, the call works perfectly well if either Alice or Bob doesn't have a relationship with an IdP; they just get a lower level of assurance. I.e., they simply have whatever information their calling site claims about the caller/callee's identity. Moreover, Alice might wish to make an anonymous call through an anonymous calling site, in which case she would of course just not provide any identity assertion and the calling site would mask her identity from Bob.

#### 4.2. Media Consent Verification

As described in ([I-D.ietf-rtcweb-security]; Section 4.2) media consent verification is provided via ICE. Thus, Alice and Bob perform ICE checks with each other. At the completion of these checks, they are ready to send non-ICE data.

At this point, Alice knows that (a) Bob (assuming he is verified via his IdP) or someone else who the signaling service is claiming is Bob is willing to exchange traffic with her and (b) that either Bob is at



the IP address which she has verified via ICE or there is an attacker who is on-path to that IP address detouring the traffic. Note that it is not possible for an attacker who is on-path between Alice and Bob but not attached to the signaling service to spoof these checks because they do not have the ICE credentials. Bob has the same security guarantees with respect to Alice.

#### 4.3. DTLS Handshake

Once the requisite ICE checks have completed, Alice and Bob can set up a secure channel or channels. This is performed via DTLS [RFC6347] and DTLS-SRTP [RFC5763] keying for SRTP [RFC3711] for the media channel and SCTP over DTLS [RFC8261] for data channels. Specifically, Alice and Bob perform a DTLS handshake on every component which has been established by ICE. The total number of channels depends on the amount of muxing; in the most likely case we are using both RTP/RTCP mux and muxing multiple media streams on the same channel, in which case there is only one DTLS handshake. Once the DTLS handshake has completed, the keys are exported [RFC5705] and used to key SRTP for the media channels.

At this point, Alice and Bob know that they share a set of secure data and/or media channels with keys which are not known to any third-party attacker. If Alice and Bob authenticated via their IdPs, then they also know that the signaling service is not mounting a man-in-the-middle attack on their traffic. Even if they do not use an IdP, as long as they have minimal trust in the signaling service not to perform a man-in-the-middle attack, they know that their communications are secure against the signaling service as well (i.e., that the signaling service cannot mount a passive attack on the communications).

#### 4.4. Communications and Consent Freshness

From a security perspective, everything from here on in is a little anticlimactic: Alice and Bob exchange data protected by the keys negotiated by DTLS. Because of the security guarantees discussed in the previous sections, they know that the communications are encrypted and authenticated.

The one remaining security property we need to establish is "consent freshness", i.e., allowing Alice to verify that Bob is still prepared to receive her communications so that Alice does not continue to send large traffic volumes to entities which went abruptly offline. ICE specifies periodic STUN keepalives but only if media is not flowing. Because the consent issue is more difficult here, we require WebRTC implementations to periodically send keepalives. As described in Section 5.3, these keepalives MUST be based on the consent freshness

mechanism specified in [RFC7675]. If a keepalive fails and no new ICE channels can be established, then the session is terminated.

## 5. SDP Identity Attribute

The SDP 'identity' attribute is a session-level attribute that is used by an endpoint to convey its identity assertion to its peer. The identity assertion value is encoded as Base-64, as described in Section 4 of [RFC4648].

The procedures in this section are based on the assumption that the identity assertion of an endpoint is bound to the fingerprints of the endpoint. This does not preclude the definition of alternative means of binding an assertion to the endpoint, but such means are outside the scope of this specification.

The semantics of multiple 'identity' attributes within an offer or answer are undefined. Implementations SHOULD only include a single 'identity' attribute in an offer or answer and relying parties MAY elect to ignore all but the first 'identity' attribute.

Name: identity

Value: identity-assertion

Usage Level: session

Charset Dependent: no

Default Value: N/A

Name: identity

## Syntax:

```

identity-assertion      = identity-assertion-value
                          *(SP identity-extension)
identity-assertion-value = base64
identity-extension      = extension-name [ "=" extension-value ]
extension-name          = token
extension-value         = 1*(%x01-09 / %x0b-0c / %x0e-3a / %x3c-ff)
                          ; byte-string from [RFC4566]

```

<ALPHA and DIGIT as defined in [RFC4566]>

<base64 as defined in [RFC4566]>

## Example:

```

a=identity:\
  eyJpZHAiOnsiZG9tYWluIjoizXhhbXBsZS5vcmdiLCJwcm90b2NvbCI6ImJvZ3Vz\
  In0sImFzc2VydGlvbii6IntcImlkZW50aXR5XCI6XCJib2JAZXhhbXBsZS5vcmdc\
  IixcImNvbnRlbnRzXCI6XCJhYmNkZWZnaGlqa2xtbm9wcXJzdHV2d3l6XCIsXCJz\
  aWduYXRlcmVcIjpcIjAxMDIwMzA0MDUwNlwiSj9

```

Note that long lines in the example are folded to meet the column width constraints of this document; the backslash ("\") at the end of a line, the carriage return that follows, and whitespace shall be ignored.

This specification does not define any extensions for the attribute.

The identity-assertion value is a JSON [RFC8259] encoded string. The JSON object contains two keys: "assertion" and "idp". The "assertion" key value contains an opaque string that is consumed by the IdP. The "idp" key value contains a dictionary with one or two further values that identify the IdP. See Section 7.6 for more details.

## 5.1. Offer/Answer Considerations

This section defines the SDP Offer/Answer [RFC3264] considerations for the SDP 'identity' attribute.

Within this section, 'initial offer' refers to the first offer in the SDP session that contains an SDP "identity" attribute.

## 5.1.1. Generating the Initial SDP Offer

When an offerer sends an offer, in order to provide its identity assertion to the peer, it includes an 'identity' attribute in the offer. In addition, the offerer includes one or more SDP

'fingerprint' attributes. The 'identity' attribute MUST be bound to all the 'fingerprint' attributes in the session description.

#### 5.1.2. Generating of SDP Answer

If the answerer elects to include an 'identity' attribute, it follows the same steps as those in Section 5.1.1. The answerer can choose to include or omit an 'identity' attribute independently, regardless of whether the offerer did so.

#### 5.1.3. Processing an SDP Offer or Answer

When an endpoint receives an offer or answer that contains an 'identity' attribute, the answerer can use the the attribute information to contact the IdP and verify the identity of the peer. If the identity requires a third-party IdP as described in Section 7.1 then that IdP will need to have been specifically configured. If the identity verification fails, the answerer MUST discard the offer or answer as malformed.

#### 5.1.4. Modifying the Session

When modifying a session, if the set of fingerprints is unchanged, then the sender MAY send the same 'identity' attribute. In this case, the established identity MUST be applied to existing DTLS connections as well as new connections established using one of those fingerprints. Note that [I-D.ietf-rtcweb-jsep], Section 5.2.1 requires that each media section use the same set of fingerprints for every media section. If a new identity attribute is received, then the receiver MUST apply that identity to all existing connections.

If the set of fingerprints changes, then the sender MUST either send a new 'identity' attribute or none at all. Because a change in fingerprints also causes a new DTLS connection to be established, the receiver MUST discard all previously established identities.

### 6. Detailed Technical Description

#### 6.1. Origin and Web Security Issues

The basic unit of permissions for WebRTC is the origin [RFC6454]. Because the security of the origin depends on being able to authenticate content from that origin, the origin can only be securely established if data is transferred over HTTPS [RFC2818]. Thus, clients MUST treat HTTP and HTTPS origins as different permissions domains. Note: this follows directly from the origin security model and is stated here merely for clarity.

Many web browsers currently forbid by default any active mixed content on HTTPS pages. That is, when JavaScript is loaded from an HTTP origin onto an HTTPS page, an error is displayed and the HTTP content is not executed unless the user overrides the error. Any browser which enforces such a policy will also not permit access to WebRTC functionality from mixed content pages (because they never display mixed content). Browsers which allow active mixed content MUST nevertheless disable WebRTC functionality in mixed content settings.

Note that it is possible for a page which was not mixed content to become mixed content during the duration of the call. The major risk here is that the newly arrived insecure JS might redirect media to a location controlled by the attacker. Implementations MUST either choose to terminate the call or display a warning at that point.

Also note that the security architecture depends on the keying material not being available to move between origins. But, it is assumed that the identity assertion can be passed to anyone that the page cares to.

## 6.2. Device Permissions Model

Implementations MUST obtain explicit user consent prior to providing access to the camera and/or microphone. Implementations MUST at minimum support the following two permissions models for HTTPS origins.

- o Requests for one-time camera/microphone access.
- o Requests for permanent access.

Because HTTP origins cannot be securely established against network attackers, implementations MUST refuse all permissions grants for HTTP origins.

In addition, they SHOULD support requests for access that promise that media from this grant will be sent to a single communicating peer (obviously there could be other requests for other peers), eE.g., "Call customerservice@example.org". The semantics of this request are that the media stream from the camera and microphone will only be routed through a connection which has been cryptographically verified (through the IdP mechanism or an X.509 certificate in the DTLS-SRTP handshake) as being associated with the stated identity. Note that it is unlikely that browsers would have X.509 certificates, but servers might. Browsers servicing such requests SHOULD clearly indicate that identity to the user when asking for permission. The idea behind this type of permissions is that a user might have a

fairly narrow list of peers he is willing to communicate with, e.g., "my mother" rather than "anyone on Facebook". Narrow permissions grants allow the browser to do that enforcement.

**API Requirement:** The API **MUST** provide a mechanism for the requesting JS to relinquish the ability to see or modify the media (e.g., via `MediaStream.record()`). Combined with secure authentication of the communicating peer, this allows a user to be sure that the calling site is not accessing or modifying their conversion.

**UI Requirement:** The UI **MUST** clearly indicate when the user's camera and microphone are in use. This indication **MUST NOT** be suppressable by the JS and **MUST** clearly indicate how to terminate device access, and provide a UI means to immediately stop camera/microphone input without the JS being able to prevent it.

**UI Requirement:** If the UI indication of camera/microphone use are displayed in the browser such that minimizing the browser window would hide the indication, or the JS creating an overlapping window would hide the indication, then the browser **SHOULD** stop camera and microphone input when the indication is hidden. [Note: this may not be necessary in systems that are non-windows-based but that have good notifications support, such as phones.]

- o Browsers **MUST NOT** permit permanent screen or application sharing permissions to be installed as a response to a JS request for permissions. Instead, they must require some other user action such as a permissions setting or an application install experience to grant permission to a site.
- o Browsers **MUST** provide a separate dialog request for screen/application sharing permissions even if the media request is made at the same time as camera and microphone.
- o The browser **MUST** indicate any windows which are currently being shared in some unambiguous way. Windows which are not visible **MUST NOT** be shared even if the application is being shared. If the screen is being shared, then that **MUST** be indicated.

Browsers **MAY** permit the formation of data channels without any direct user approval. Because sites can always tunnel data through the server, further restrictions on the data channel do not provide any additional security. (See Section 6.3 for a related issue).

Implementations which support some form of direct user authentication **SHOULD** also provide a policy by which a user can authorize calls only to specific communicating peers. Specifically, the implementation **SHOULD** provide the following interfaces/controls:

- o Allow future calls to this verified user.
- o Allow future calls to any verified user who is in my system address book (this only works with address book integration, of course).

Implementations SHOULD also provide a different user interface indication when calls are in progress to users whose identities are directly verifiable. Section 6.5 provides more on this.

### 6.3. Communications Consent

Browser client implementations of WebRTC MUST implement ICE. Server gateway implementations which operate only at public IP addresses MUST implement either full ICE or ICE-Lite [RFC8445].

Browser implementations MUST verify reachability via ICE prior to sending any non-ICE packets to a given destination. Implementations MUST NOT provide the ICE transaction ID to JavaScript during the lifetime of the transaction (i.e., during the period when the ICE stack would accept a new response for that transaction). The JS MUST NOT be permitted to control the local ufrag and password, though it of course knows it.

While continuing consent is required, the ICE [RFC8445]; Section 10 keepalives use STUN Binding Indications which are one-way and therefore not sufficient. The current WG consensus is to use ICE Binding Requests for continuing consent freshness. ICE already requires that implementations respond to such requests, so this approach is maximally compatible. A separate document will profile the ICE timers to be used; see [RFC7675].

### 6.4. IP Location Privacy

A side effect of the default ICE behavior is that the peer learns one's IP address, which leaks large amounts of location information. This has negative privacy consequences in some circumstances. The API requirements in this section are intended to mitigate this issue. Note that these requirements are not intended to protect the user's IP address from a malicious site. In general, the site will learn at least a user's server reflexive address from any HTTP transaction. Rather, these requirements are intended to allow a site to cooperate with the user to hide the user's IP address from the other side of the call. Hiding the user's IP address from the server requires some sort of explicit privacy preserving mechanism on the client (e.g., Tor Browser [<https://www.torproject.org/projects/torbrowser.html.en>]) and is out of scope for this specification.

API Requirement: The API MUST provide a mechanism to allow the JS to suppress ICE negotiation (though perhaps to allow candidate gathering) until the user has decided to answer the call [note: determining when the call has been answered is a question for the JS.] This enables a user to prevent a peer from learning their IP address if they elect not to answer a call and also from learning whether the user is online.

API Requirement: The API MUST provide a mechanism for the calling application JS to indicate that only TURN candidates are to be used. This prevents the peer from learning one's IP address at all. This mechanism MUST also permit suppression of the related address field, since that leaks local addresses.

API Requirement: The API MUST provide a mechanism for the calling application to reconfigure an existing call to add non-TURN candidates. Taken together, this and the previous requirement allow ICE negotiation to start immediately on incoming call notification, thus reducing post-dial delay, but also to avoid disclosing the user's IP address until they have decided to answer. They also allow users to completely hide their IP address for the duration of the call. Finally, they allow a mechanism for the user to optimize performance by reconfiguring to allow non-TURN candidates during an active call if the user decides they no longer need to hide their IP address

Note that some enterprises may operate proxies and/or NATs designed to hide internal IP addresses from the outside world. WebRTC provides no explicit mechanism to allow this function. Either such enterprises need to proxy the HTTP/HTTPS and modify the SDP and/or the JS, or there needs to be browser support to set the "TURN-only" policy regardless of the site's preferences.

## 6.5. Communications Security

Implementations MUST support SRTP [RFC3711]. Implementations MUST support DTLS [RFC6347] and DTLS-SRTP [RFC5763][RFC5764] for SRTP keying. Implementations MUST support SCTP over DTLS [RFC8261].

All media channels MUST be secured via SRTP and SRTCP. Media traffic MUST NOT be sent over plain (unencrypted) RTP or RTCP; that is, implementations MUST NOT negotiate cipher suites with NULL encryption modes. DTLS-SRTP MUST be offered for every media channel. WebRTC implementations MUST NOT offer SDP Security Descriptions [RFC4568] or select it if offered. A SRTP MKI MUST NOT be used.

All data channels MUST be secured via DTLS.



All Implementations MUST support DTLS 1.2 with the TLS\_ECDHE\_ECDSA\_WITH\_AES\_128\_GCM\_SHA256 cipher suite and the P-256 curve [FIPS186]. Earlier drafts of this specification required DTLS 1.0 with the cipher suite TLS\_ECDHE\_ECDSA\_WITH\_AES\_128\_CBC\_SHA, and at the time of this writing some implementations do not support DTLS 1.2; endpoints which support only DTLS 1.2 might encounter interoperability issues. The DTLS-SRTP protection profile SRTP\_AES128\_CM\_HMAC\_SHA1\_80 MUST be supported for SRTP. Implementations MUST favor cipher suites which support (Perfect Forward Secrecy) PFS over non-PFS cipher suites and SHOULD favor AEAD over non-AEAD cipher suites.

Implementations MUST NOT implement DTLS renegotiation and MUST reject it with a "no\_renegotiation" alert if offered.

Endpoints MUST NOT implement TLS False Start [RFC7918].

API Requirement: The API MUST generate a new authentication key pair for every new call by default. This is intended to allow for unlinkability.

API Requirement: The API MUST provide a means to reuse a key pair for calls. This can be used to enable key continuity-based authentication, and could be used to amortize key generation costs.

API Requirement: Unless the user specifically configures an external key pair, different key pairs MUST be used for each origin. (This avoids creating a super-cookie.)

API Requirement: When DTLS-SRTP is used, the API MUST NOT permit the JS to obtain the negotiated keying material. This requirement preserves the end-to-end security of the media.

UI Requirements: A user-oriented client MUST provide an "inspector" interface which allows the user to determine the security characteristics of the media.

The following properties SHOULD be displayed "up-front" in the browser chrome, i.e., without requiring the user to ask for them:

- \* A client MUST provide a user interface through which a user may determine the security characteristics for currently-displayed audio and video stream(s)

- \* A client MUST provide a user interface through which a user may determine the security characteristics for transmissions of their microphone audio and camera video.
- \* If the far endpoint was directly verified, either via a third-party verifiable X.509 certificate or via a Web IdP mechanism (see Section 7) the "security characteristics" MUST include the verified information. X.509 identities and Web IdP identities have similar semantics and should be displayed in a similar way.

The following properties are more likely to require some "drill-down" from the user:

- \* The "security characteristics" MUST indicate the cryptographic algorithms in use (For example: "AES-CBC".)
- \* The "security characteristics" MUST indicate whether PFS is provided.
- \* The "security characteristics" MUST include some mechanism to allow an out-of-band verification of the peer, such as a certificate fingerprint or a Short Authentication String (SAS). These are compared by the peers to authenticate one another.

## 7. Web-Based Peer Authentication

In a number of cases, it is desirable for the endpoint (i.e., the browser) to be able to directly identify the endpoint on the other side without trusting the signaling service to which they are connected. For instance, users may be making a call via a federated system where they wish to get direct authentication of the other side. Alternately, they may be making a call on a site which they minimally trust (such as a poker site) but to someone who has an identity on a site they do trust (such as a social network.)

Recently, a number of Web-based identity technologies (OAuth, Facebook Connect etc.) have been developed. While the details vary, what these technologies share is that they have a Web-based (i.e., HTTP/HTTPS) identity provider which attests to Alice's identity. For instance, if Alice has an account at example.org, Alice could use the example.org identity provider to prove to others that Alice is alice@example.org. The development of these technologies allows us

to separate calling from identity provision: Alice could call you on a poker site but identify herself as `alice@example.org`.

Whatever the underlying technology, the general principle is that the party which is being authenticated is NOT the signaling site but rather the user (and their browser). Similarly, the relying party is the browser and not the signaling site. Thus, the browser **MUST** generate the input to the IdP assertion process and display the results of the verification process to the user in a way which cannot be imitated by the calling site.

The mechanisms defined in this document do not require the browser to implement any particular identity protocol or to support any particular IdP. Instead, this document provides a generic interface which any IdP can implement. Thus, new IdPs and protocols can be introduced without change to either the browser or the calling service. This avoids the need to make a commitment to any particular identity protocol, although browsers may opt to directly implement some identity protocols in order to provide superior performance or UI properties.

#### 7.1. Trust Relationships: IdPs, APs, and RPs

Any federated identity protocol has three major participants:

**Authenticating Party (AP):** The entity which is trying to establish its identity.

**Identity Provider (IdP):** The entity which is vouching for the AP's identity.

**Relying Party (RP):** The entity which is trying to verify the AP's identity.

The AP and the IdP have an account relationship of some kind: the AP registers with the IdP and is able to subsequently authenticate directly to the IdP (e.g., with a password). This means that the browser must somehow know which IdP(s) the user has an account relationship with. This can either be something that the user configures into the browser or that is configured at the calling site and then provided to the PeerConnection by the Web application at the calling site. The use case for having this information configured into the browser is that the user may "log into" the browser to bind it to some identity. This is becoming common in new browsers.

However, it should also be possible for the IdP information to simply be provided by the calling application.

At a high level there are two kinds of IdPs:

**Authoritative:** IdPs which have verifiable control of some section of the identity space. For instance, in the realm of e-mail, the operator of "example.com" has complete control of the namespace ending in "@example.com". Thus, "alice@example.com" is whoever the operator says it is. Examples of systems with authoritative identity providers include DNSSEC, RFC 4474, and Facebook Connect (Facebook identities only make sense within the context of the Facebook system).

**Third-Party:** IdPs which don't have control of their section of the identity space but instead verify user's identities via some unspecified mechanism and then attest to it. Because the IdP doesn't actually control the namespace, RPs need to trust that the IdP is correctly verifying AP identities, and there can potentially be multiple IdPs attesting to the same section of the identity space. Probably the best-known example of a third-party identity provider is SSL/TLS certificates, where there are a large number of CAs all of whom can attest to any domain name.

If an AP is authenticating via an authoritative IdP, then the RP does not need to explicitly configure trust in the IdP at all. The identity mechanism can directly verify that the IdP indeed made the relevant identity assertion (a function provided by the mechanisms in this document), and any assertion it makes about an identity for which it is authoritative is directly verifiable. Note that this does not mean that the IdP might not lie, but that is a trustworthiness judgement that the user can make at the time he looks at the identity.

By contrast, if an AP is authenticating via a third-party IdP, the RP needs to explicitly trust that IdP (hence the need for an explicit trust anchor list in PKI-based SSL/TLS clients). The list of trustable IdPs needs to be configured directly into the browser, either by the user or potentially by the browser manufacturer. This is a significant advantage of authoritative IdPs and implies that if third-party IdPs are to be supported, the potential number needs to be fairly small.

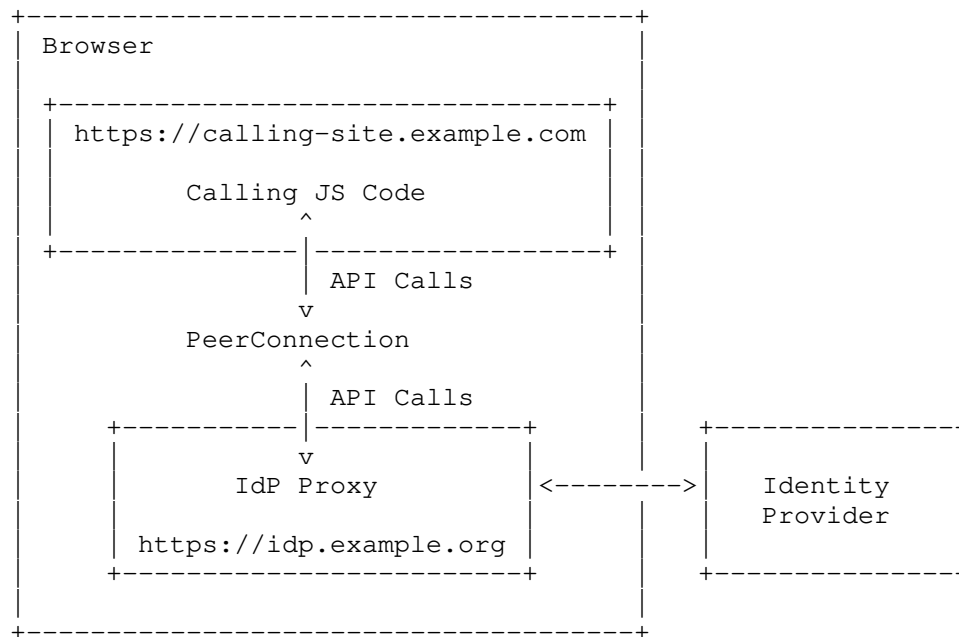
## 7.2. Overview of Operation

In order to provide security without trusting the calling site, the PeerConnection component of the browser must interact directly with the IdP. The details of the mechanism are described in the W3C API specification, but the general idea is that the PeerConnection component downloads JS from a specific location on the IdP dictated by the IdP domain name. That JS (the "IdP proxy") runs in an isolated security context within the browser and the PeerConnection talks to it via a secure message passing channel.

Note that there are two logically separate functions here:

- o Identity assertion generation.
- o Identity assertion verification.

The same IdP JS "endpoint" is used for both functions but of course a given IdP might behave differently and load new JS to perform one function or the other.



When the PeerConnection object wants to interact with the IdP, the sequence of events is as follows:

1. The browser (the PeerConnection component) instantiates an IdP proxy. This allows the IdP to load whatever JS is necessary into the proxy. The resulting code runs in the IdP's security context.
2. The IdP registers an object with the browser that conforms to the API defined in [webrtc-api].
3. The browser invokes methods on the object registered by the IdP proxy to create or verify identity assertions.

This approach allows us to decouple the browser from any particular identity provider; the browser need only know how to load the IdP's JavaScript--the location of which is determined based on the IdP's identity--and to call the generic API for requesting and verifying identity assertions. The IdP provides whatever logic is necessary to bridge the generic protocol to the IdP's specific requirements. Thus, a single browser can support any number of identity protocols, including being forward compatible with IdPs which did not exist at the time the browser was written.

### 7.3. Items for Standardization

There are two parts to this work:

- o The precise information from the signaling message that must be cryptographically bound to the user's identity and a mechanism for carrying assertions in JSEP messages. This is specified in Section 7.4.
- o The interface to the IdP, which is defined in the companion W3C WebRTC API specification [webrtc-api].

The WebRTC API specification also defines JavaScript interfaces that the calling application can use to specify which IdP to use. That API also provides access to the assertion-generation capability and the status of the validation process.

### 7.4. Binding Identity Assertions to JSEP Offer/Answer Transactions

An identity assertion binds the user's identity (as asserted by the IdP) to the SDP offer/answer exchange and specifically to the media. In order to achieve this, the PeerConnection must provide the DTLS-SRTP fingerprint to be bound to the identity. This is provided as a JavaScript object (also known as a dictionary or hash) with a single "fingerprint" key, as shown below:

```
{
  "fingerprint":
  [
    { "algorithm": "sha-256",
      "digest": "4A:AD:B9:B1:3F:....:E5:7C:AB" },
    { "algorithm": "sha-1",
      "digest": "74:E9:76:C8:19:....:F4:45:6B" }
  ]
}
```

The "fingerprint" value is an array of objects. Each object in the array contains "algorithm" and "digest" values, which correspond directly to the algorithm and digest values in the "fingerprint" attribute of the SDP [RFC8122].

This object is encoded in a JSON [RFC8259] string for passing to the IdP. The identity assertion returned by the IdP, which is encoded in the "identity" attribute, is a JSON object that is encoded as described in Section 7.4.1.

This structure does not need to be interpreted by the IdP or the IdP proxy. It is consumed solely by the RP's browser. The IdP merely treats it as an opaque value to be attested to. Thus, new parameters can be added to the assertion without modifying the IdP.

#### 7.4.1. Carrying Identity Assertions

Once an IdP has generated an assertion (see Section 7.6), it is attached to the SDP offer/answer message. This is done by adding a new 'identity' attribute to the SDP. The sole contents of this value is the identity assertion. The identity assertion produced by the IdP is encoded into a UTF-8 JSON text, then Base64-encoded [RFC4648] to produce this string. For example:

```

v=0
o=- 1181923068 1181923196 IN IP4 ual.example.com
s=example1
c=IN IP4 ual.example.com
a=fingerprint:sha-1 \
  4A:AD:B9:B1:3F:82:18:3B:54:02:12:DF:3E:5D:49:6B:19:E5:7C:AB
a=identity:\
  eyJpZHAiOnsizG9tYWluIjoizXhhbXBsZS5vcmdiLCJwcm90b2NvbCI6ImJvZ3Vz\
  In0sImFzc2VydgIvbiI6IntcImlkZW50aXR5XCI6XCJib2JAZXhhbXBsZS5vcmdc\
  IixcImNvbmlbnRzXCI6XCJhYmNkZWZnaGlqa2xtbm9wcXJzdHV2d3l6XCIsXCJz\
  aWduYXR1cmVcIjpcIjAxMDIwMzA0MDUwNlwiJ9
a=...
t=0 0
m=audio 6056 RTP/SAVP 0
a=sendrecv
...

```

Note that long lines in the example are folded to meet the column width constraints of this document; the backslash ("`\`") at the end of a line, the carriage return that follows, and whitespace shall be ignored.

The 'identity' attribute attests to all "fingerprint" attributes in the session description. It is therefore a session-level attribute.

Multiple "fingerprint" values can be used to offer alternative certificates for a peer. The "identity" attribute MUST include all fingerprint values that are included in "fingerprint" attributes of the session description.

The RP browser MUST verify that the in-use certificate for a DTLS connection is in the set of fingerprints returned from the IdP when verifying an assertion.

## 7.5. Determining the IdP URI

In order to ensure that the IdP is under control of the domain owner rather than someone who merely has an account on the domain owner's server (e.g., in shared hosting scenarios), the IdP JavaScript is hosted at a deterministic location based on the IdP's domain name. Each IdP proxy instance is associated with two values:

**Authority:** The authority [RFC3986] at which the IdP's service is hosted.

**protocol:** The specific IdP protocol which the IdP is using. This is a completely opaque IdP-specific string, but allows an IdP to implement two protocols in parallel. This value may be the empty



string. If no value for protocol is provided, a value of "default" is used.

Each IdP MUST serve its initial entry page (i.e., the one loaded by the IdP proxy) from a well-known URI [RFC5785]. The well-known URI for an IdP proxy is formed from the following URI components:

1. The scheme, "https:". An IdP MUST be loaded using HTTPS [RFC2818].
2. The authority [RFC3986]. As noted above, the authority MAY contain a non-default port number or userinfo sub-component. Both are removed when determining if an asserted identity matches the name of the IdP.
3. The path, starting with "/.well-known/idp-proxy/" and appended with the IdP protocol. Note that the separator characters '/' (%2F) and '\' (%5C) MUST NOT be permitted in the protocol field, lest an attacker be able to direct requests outside of the controlled "/.well-known/" prefix. Query and fragment values MAY be used by including '?' or '#' characters.

For example, for the IdP "identity.example.com" and the protocol "example", the URL would be:

```
https://identity.example.com/.well-known/idp-proxy/example
```

The IdP MAY redirect requests to this URL, but they MUST retain the "https" scheme. This changes the effective origin of the IdP, but not the domain of the identities that the IdP is permitted to assert and validate. I.e., the IdP is still regarded as authoritative for the original domain.

#### 7.5.1. Authenticating Party

How an AP determines the appropriate IdP domain is out of scope of this specification. In general, however, the AP has some actual account relationship with the IdP, as this identity is what the IdP is attesting to. Thus, the AP somehow supplies the IdP information to the browser. Some potential mechanisms include:

- o Provided by the user directly.
- o Selected from some set of IdPs known to the calling site. E.g., a button that shows "Authenticate via Facebook Connect"

### 7.5.2. Relying Party

Unlike the AP, the RP need not have any particular relationship with the IdP. Rather, it needs to be able to process whatever assertion is provided by the AP. As the assertion contains the IdP's identity in the "idp" field of the JSON-encoded object (see Section 7.6), the URI can be constructed directly from the assertion, and thus the RP can directly verify the technical validity of the assertion with no user interaction. Authoritative assertions need only be verifiable. Third-party assertions also MUST be verified against local policy, as described in Section 8.1.

### 7.6. Requesting Assertions

The input to identity assertion is the JSON-encoded object described in Section 7.4 that contains the set of certificate fingerprints the browser intends to use. This string is treated as opaque from the perspective of the IdP.

The browser also identifies the origin that the PeerConnection is run in, which allows the IdP to make decisions based on who is requesting the assertion.

An application can optionally provide a user identifier hint when specifying an IdP. This value is a hint that the IdP can use to select amongst multiple identities, or to avoid providing assertions for unwanted identities. The "username" is a string that has no meaning to any entity other than the IdP, it can contain any data the IdP needs in order to correctly generate an assertion.

An identity assertion that is successfully provided by the IdP consists of the following information:

idp: The domain name of an IdP and the protocol string. This MAY identify a different IdP or protocol from the one that generated the assertion.

assertion: An opaque value containing the assertion itself. This is only interpretable by the identified IdP or the IdP code running in the client.

Figure 5 shows an example assertion formatted as JSON. In this case, the message has presumably been digitally signed/MACed in some way that the IdP can later verify it, but this is an implementation detail and out of scope of this document.

```
{
  "idp":{
    "domain": "example.org",
    "protocol": "bogus"
  },
  "assertion": "{\\"identity\\":\\"bob@example.org\\",
                \\"contents\\":\\"abcdefghijklmnopqrstuvwyz\\",
                \\"signature\\":\\"010203040506\\"}"
}
```

Figure 5: Example assertion

For use in signaling, the assertion is serialized into JSON, Base64-encoded [RFC4648], and used as the value of the "identity" attribute. IdPs SHOULD ensure that any assertions they generate cannot be interpreted in a different context. E.g., they should use a distinct format or have separate cryptographic keys for assertion generation and other purposes. Line breaks are inserted solely for readability.

#### 7.7. Managing User Login

In order to generate an identity assertion, the IdP needs proof of the user's identity. It is common practice to authenticate users (using passwords or multi-factor authentication), then use Cookies [RFC6265] or HTTP authentication [RFC7617] for subsequent exchanges.

The IdP proxy is able to access cookies, HTTP authentication or other persistent session data because it operates in the security context of the IdP origin. Therefore, if a user is logged in, the IdP could have all the information needed to generate an assertion.

An IdP proxy is unable to generate an assertion if the user is not logged in, or the IdP wants to interact with the user to acquire more information before generating the assertion. If the IdP wants to interact with the user before generating an assertion, the IdP proxy can fail to generate an assertion and instead indicate a URL where login should proceed.

The application can then load the provided URL to enable the user to enter credentials. The communication between the application and the IdP is described in [webrtc-api].

#### 8. Verifying Assertions

The input to identity validation is the assertion string taken from a decoded 'identity' attribute.

The IdP proxy verifies the assertion. Depending on the identity protocol, the proxy might contact the IdP server or other servers. For instance, an OAuth-based protocol will likely require using the IdP as an oracle, whereas with a signature-based scheme might be able to verify the assertion without contacting the IdP, provided that it has cached the relevant public key.

Regardless of the mechanism, if verification succeeds, a successful response from the IdP proxy consists of the following information:

identity: The identity of the AP from the IdP's perspective.

Details of this are provided in Section 8.1.

contents: The original unmodified string provided by the AP as input to the assertion generation process.

Figure 6 shows an example response, which is JSON-formatted.

```
{
  "identity": "bob@example.org",
  "contents": "{\"fingerprint\":[ ... ]}"
}
```

Figure 6: Example verification result

### 8.1. Identity Formats

The identity provided from the IdP to the RP browser MUST consist of a string representing the user's identity. This string is in the form "<user>@<domain>", where "user" consists of any character, and domain is an internationalized domain name [RFC5890] encoded as a sequence of U-labels.

The PeerConnection API MUST check this string as follows:

1. If the "domain" portion of the string is equal to the domain name of the IdP proxy, then the assertion is valid, as the IdP is authoritative for this domain. Comparison of domain names is done using the label equivalence rule defined in Section 2.3.2.4 of [RFC5890].
2. If the "domain" portion of the string is not equal to the domain name of the IdP proxy, then the PeerConnection object MUST reject the assertion unless both:
  1. the IdP domain is trusted as an acceptable third-party IdP;  
and

2. local policy is configured to trust this IdP domain for the domain portion of the identity string.

Any "@" or "%" characters in the "user" portion of the identity MUST be escaped according to the "Percent-Encoding" rules defined in Section 2.1 of [RFC3986]. Characters other than "@" and "%" MUST NOT be percent-encoded. For example, with a "user" of "user@133" and a "domain" of "identity.example.com", the resulting string will be encoded as "user%40133@identity.example.com".

Implementations are cautioned to take care when displaying user identities containing escaped "@" characters. If such characters are unescaped prior to display, implementations MUST distinguish between the domain of the IdP proxy and any domain that might be implied by the portion of the "<user>" portion that appears after the escaped "@" sign.

## 9. Security Considerations

Much of the security analysis of this problem is contained in [I-D.ietf-rtcweb-security] or in the discussion of the particular issues above. In order to avoid repetition, this section focuses on (a) residual threats that are not addressed by this document and (b) threats produced by failure/misbehavior of one of the components in the system.

### 9.1. Communications Security

IF HTTPS is not used to secure communications to the signaling server, and the identity mechanism used in Section 7 is not used, then any on-path attacker can replace the DTLS-SRTP fingerprints in the handshake and thus substitute its own identity for that of either endpoint.

Even if HTTPS is used, the signaling server can potentially mount a man-in-the-middle attack unless implementations have some mechanism for independently verifying keys. The UI requirements in Section 6.5 are designed to provide such a mechanism for motivated/security conscious users, but are not suitable for general use. The identity service mechanisms in Section 7 are more suitable for general use. Note, however, that a malicious signaling service can strip off any such identity assertions, though it cannot forge new ones. Note that all of the third-party security mechanisms available (whether X.509 certificates or a third-party IdP) rely on the security of the third party--this is of course also true of the user's connection to the Web site itself. Users who wish to assure themselves of security against a malicious identity provider can only do so by verifying

peer credentials directly, e.g., by checking the peer's fingerprint against a value delivered out of band.

In order to protect against malicious content JavaScript, that JavaScript MUST NOT be allowed to have direct access to---or perform computations with---DTLS keys. For instance, if content JS were able to compute digital signatures, then it would be possible for content JS to get an identity assertion for a browser's generated key and then use that assertion plus a signature by the key to authenticate a call protected under an ephemeral Diffie-Hellman (DH) key controlled by the content JS, thus violating the security guarantees otherwise provided by the IdP mechanism. Note that it is not sufficient merely to deny the content JS direct access to the keys, as some have suggested doing with the WebCrypto API [webcrypto]. The JS must also not be allowed to perform operations that would be valid for a DTLS endpoint. By far the safest approach is simply to deny the ability to perform any operations that depend on secret information associated with the key. Operations that depend on public information, such as exporting the public key are of course safe.

## 9.2. Privacy

The requirements in this document are intended to allow:

- o Users to participate in calls without revealing their location.
- o Potential callees to avoid revealing their location and even presence status prior to agreeing to answer a call.

However, these privacy protections come at a performance cost in terms of using TURN relays and, in the latter case, delaying ICE. Sites SHOULD make users aware of these tradeoffs.

Note that the protections provided here assume a non-malicious calling service. As the calling service always knows the users status and (absent the use of a technology like Tor) their IP address, they can violate the users privacy at will. Users who wish privacy against the calling sites they are using must use separate privacy enhancing technologies such as Tor. Combined WebRTC/Tor implementations SHOULD arrange to route the media as well as the signaling through Tor. Currently this will produce very suboptimal performance.

Additionally, any identifier which persists across multiple calls is potentially a problem for privacy, especially for anonymous calling services. Such services SHOULD instruct the browser to use separate DTLS keys for each call and also to use TURN throughout the call. Otherwise, the other side will learn linkable information that would

allow them to correlate the browser across multiple calls. Additionally, browsers SHOULD implement the privacy-preserving CNAME generation mode of [RFC7022].

### 9.3. Denial of Service

The consent mechanisms described in this document are intended to mitigate denial of service attacks in which an attacker uses clients to send large amounts of traffic to a victim without the consent of the victim. While these mechanisms are sufficient to protect victims who have not implemented WebRTC at all, WebRTC implementations need to be more careful.

Consider the case of a call center which accepts calls via WebRTC. An attacker proxies the call center's front-end and arranges for multiple clients to initiate calls to the call center. Note that this requires user consent in many cases but because the data channel does not need consent, he can use that directly. Since ICE will complete, browsers can then be induced to send large amounts of data to the victim call center if it supports the data channel at all. Preventing this attack requires that automated WebRTC implementations implement sensible flow control and have the ability to triage out (i.e., stop responding to ICE probes on) calls which are behaving badly, and especially to be prepared to remotely throttle the data channel in the absence of plausible audio and video (which the attacker cannot control).

Another related attack is for the signaling service to swap the ICE candidates for the audio and video streams, thus forcing a browser to send video to the sink that the other victim expects will contain audio (perhaps it is only expecting audio!) potentially causing overload. Muxing multiple media flows over a single transport makes it harder to individually suppress a single flow by denying ICE keepalives. Either media-level (RTCP) mechanisms must be used or the implementation must deny responses entirely, thus terminating the call.

Yet another attack, suggested by Magnus Westerlund, is for the attacker to cross-connect offers and answers as follows. It induces the victim to make a call and then uses its control of other users' browsers to get them to attempt a call to someone. It then translates their offers into apparent answers to the victim, which looks like large-scale parallel forking. The victim still responds to ICE responses and now the browsers all try to send media to the victim. Implementations can defend themselves from this attack by only responding to ICE Binding Requests for a limited number of remote ufrags (this is the reason for the requirement that the JS not be able to control the ufrag and password).

[I-D.ietf-rtcweb-rtp-usage] Section 13 documents a number of potential RTCP-based DoS attacks and countermeasures.

Note that attacks based on confusing one end or the other about consent are possible even in the face of the third-party identity mechanism as long as major parts of the signaling messages are not signed. On the other hand, signing the entire message severely restricts the capabilities of the calling application, so there are difficult tradeoffs here.

#### 9.4. IdP Authentication Mechanism

This mechanism relies for its security on the IdP and on the PeerConnection correctly enforcing the security invariants described above. At a high level, the IdP is attesting that the user identified in the assertion wishes to be associated with the assertion. Thus, it must not be possible for arbitrary third parties to get assertions tied to a user or to produce assertions that RPs will accept.

##### 9.4.1. PeerConnection Origin Check

Fundamentally, the IdP proxy is just a piece of HTML and JS loaded by the browser, so nothing stops a Web attacker from creating their own IFRAME, loading the IdP proxy HTML/JS, and requesting a signature over his own keys rather than those generated in the browser. However, that proxy would be in the attacker's origin, not the IdP's origin. Only the browser itself can instantiate a context that (a) is in the IdP's origin and (b) exposes the correct API surface. Thus, the IdP proxy on the sender's side MUST ensure that it is running in the IdP's origin prior to issuing assertions.

Note that this check only asserts that the browser (or some other entity with access to the user's authentication data) attests to the request and hence to the fingerprint. It does not demonstrate that the browser has access to the associated private key, and therefore an attacker can attach their own identity to another party's keying material, thus making a call which comes from Alice appear to come from the attacker. See [I-D.ietf-mmusic-sdp-uks] for defenses against this form of attack.

##### 9.4.2. IdP Well-known URI

As described in Section 7.5 the IdP proxy HTML/JS landing page is located at a well-known URI based on the IdP's domain name. This requirement prevents an attacker who can write some resources at the IdP (e.g., on one's Facebook wall) from being able to impersonate the IdP.



#### 9.4.3. Privacy of IdP-generated identities and the hosting site

Depending on the structure of the IdP's assertions, the calling site may learn the user's identity from the perspective of the IdP. In many cases this is not an issue because the user is authenticating to the site via the IdP in any case, for instance when the user has logged in with Facebook Connect and is then authenticating their call with a Facebook identity. However, in other case, the user may not have already revealed their identity to the site. In general, IdPs SHOULD either verify that the user is willing to have their identity revealed to the site (e.g., through the usual IdP permissions dialog) or arrange that the identity information is only available to known RPs (e.g., social graph adjacencies) but not to the calling site. The "domain" field of the assertion request can be used to check that the user has agreed to disclose their identity to the calling site; because it is supplied by the PeerConnection it can be trusted to be correct.

#### 9.4.4. Security of Third-Party IdPs

As discussed above, each third-party IdP represents a new universal trust point and therefore the number of these IdPs needs to be quite limited. Most IdPs, even those which issue unqualified identities such as Facebook, can be recast as authoritative IdPs (e.g., 123456@facebook.com). However, in such cases, the user interface implications are not entirely desirable. One intermediate approach is to have special (potentially user configurable) UI for large authoritative IdPs, thus allowing the user to instantly grasp that the call is being authenticated by Facebook, Google, etc.

##### 9.4.4.1. Confusable Characters

Because a broad range of characters are permitted in identity strings, it may be possible for attackers to craft identities which are confusable with other identities (see [RFC6943] for more on this topic). This is a problem with any identifier space of this type (e.g., e-mail addresses). Those minting identifiers should avoid mixed scripts and similar confusable characters. Those presenting these identifiers to a user should consider highlighting cases of mixed script usage (see [RFC5890], section 4.4). Other best practices are still in development.

#### 9.4.5. Web Security Feature Interactions

A number of optional Web security features have the potential to cause issues for this mechanism, as discussed below.

#### 9.4.5.1. Popup Blocking

When popup blocking is in use, the IdP proxy is unable to generate popup windows, dialogs or any other form of user interactions. This prevents the IdP proxy from being used to circumvent user interaction. The "LOGINNEEDED" message allows the IdP proxy to inform the calling site of a need for user login, providing the information necessary to satisfy this requirement without resorting to direct user interaction from the IdP proxy itself.

#### 9.4.5.2. Third Party Cookies

Some browsers allow users to block third party cookies (cookies associated with origins other than the top level page) for privacy reasons. Any IdP which uses cookies to persist logins will be broken by third-party cookie blocking. One option is to accept this as a limitation; another is to have the PeerConnection object disable third-party cookie blocking for the IdP proxy.

### 10. IANA Considerations

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

Contact Name: IESG (iesg@ietf.org)

Attribute Name: identity

Long Form: identity

Type of Attribute: session-level

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

Purpose: This attribute carries an identity assertion, binding an identity to the transport-level security session.

Appropriate Values: See Section 5 of RFCXXXX [[Editor Note: This document.]]

Mux Category: NORMAL.

This section registers the "idp-proxy" well-known URI from [RFC5785].

URI suffix: idp-proxy

Change controller: IETF

## 11. Acknowledgements

Bernard Aboba, Harald Alvestrand, Richard Barnes, Dan Druta, Cullen Jennings, Hadriel Kaplan, Matthew Kaufman, Jim McEachern, Martin Thomson, Magnus Westerland. Matthew Kaufman provided the UI material in Section 6.5. Christer Holmberg provided the initial version of Section 5.1.

## 12. Changes

[RFC Editor: Please remove this section prior to publication.]

### 12.1. Changes since -15

Rewrite the Identity section in more conventional offer/answer format.

Clarify rules on changing identities.

### 12.2. Changes since -11

Update discussion of IdP security model

Replace "domain name" with RFC 3986 Authority

Clean up discussion of how to generate IdP URI.

Remove obsolete text about null cipher suites.

Remove obsolete appendixes about older IdP systems

Require support for ECDSA, PFS, and AEAD

### 12.3. Changes since -10

Update cipher suite profiles.

Rework IdP interaction based on implementation experience in Firefox.

### 12.4. Changes since -06

Replaced RTCWEB and RTC-Web with WebRTC, except when referring to the IETF WG

Forbade use in mixed content as discussed in Orlando.

Added a requirement to surface NULL ciphers to the top-level.

Tried to clarify SRTP versus DTLS-SRTP.

Added a section on screen sharing permissions.

Assorted editorial work.

#### 12.5. Changes since -05

The following changes have been made since the -05 draft.

- o Response to comments from Richard Barnes
- o More explanation of the IdP security properties and the federation use case.
- o Editorial cleanup.

#### 12.6. Changes since -03

Version -04 was a version control mistake. Please ignore.

The following changes have been made since the -04 draft.

- o Move origin check from IdP to RP per discussion in YVR.
- o Clarified treatment of X.509-level identities.
- o Editorial cleanup.

#### 12.7. Changes since -03

#### 12.8. Changes since -02

The following changes have been made since the -02 draft.

- o Forbid persistent HTTP permissions.
- o Clarified the text in S 5.4 to clearly refer to requirements on the API to provide functionality to the site.
- o Fold in the IETF portion of draft-rescorla-rtcweb-generic-idp
- o Retarget the continuing consent section to assume Binding Requests
- o Added some more privacy and linkage text in various places.

- o Editorial improvements

## 13. References

### 13.1. Normative References

- [FIPS186] National Institute of Standards and Technology (NIST), "Digital Signature Standard (DSS)", NIST PUB 186-4 , July 2013.
- [I-D.ietf-mmusic-sdp-uks]  
Thomson, M. and E. Rescorla, "Unknown Key Share Attacks on uses of TLS with the Session Description Protocol (SDP)", draft-ietf-mmusic-sdp-uks-06 (work in progress), July 2019.
- [I-D.ietf-rtcweb-jsep]  
Uberti, J., Jennings, C., and E. Rescorla, "JavaScript Session Establishment Protocol", draft-ietf-rtcweb-jsep-26 (work in progress), February 2019.
- [I-D.ietf-rtcweb-overview]  
Alvestrand, H., "Overview: Real Time Protocols for Browser-based Applications", draft-ietf-rtcweb-overview-19 (work in progress), November 2017.
- [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-26 (work in progress), March 2016.
- [I-D.ietf-rtcweb-security]  
Rescorla, E., "Security Considerations for WebRTC", draft-ietf-rtcweb-security-12 (work in progress), July 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>>.
- [RFC2818] Rescorla, E., "HTTP Over TLS", RFC 2818, DOI 10.17487/RFC2818, May 2000, <<https://www.rfc-editor.org/info/rfc2818>>.

- [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>>.
- [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>>.
- [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>>.
- [RFC4566] Handley, M., Jacobson, V., and C. Perkins, "SDP: Session Description Protocol", RFC 4566, DOI 10.17487/RFC4566, July 2006, <<https://www.rfc-editor.org/info/rfc4566>>.
- [RFC4568] Andreassen, F., Baugher, M., and D. Wing, "Session Description Protocol (SDP) Security Descriptions for Media Streams", RFC 4568, DOI 10.17487/RFC4568, July 2006, <<https://www.rfc-editor.org/info/rfc4568>>.
- [RFC4648] Josefsson, S., "The Base16, Base32, and Base64 Data Encodings", RFC 4648, DOI 10.17487/RFC4648, October 2006, <<https://www.rfc-editor.org/info/rfc4648>>.
- [RFC5246] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", RFC 5246, DOI 10.17487/RFC5246, August 2008, <<https://www.rfc-editor.org/info/rfc5246>>.
- [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>>.

- [RFC5785] Nottingham, M. and E. Hammer-Lahav, "Defining Well-Known Uniform Resource Identifiers (URIs)", RFC 5785, DOI 10.17487/RFC5785, April 2010, <<https://www.rfc-editor.org/info/rfc5785>>.
- [RFC5890] Klensin, J., "Internationalized Domain Names for Applications (IDNA): Definitions and Document Framework", RFC 5890, DOI 10.17487/RFC5890, August 2010, <<https://www.rfc-editor.org/info/rfc5890>>.
- [RFC6347] Rescorla, E. and N. Modadugu, "Datagram Transport Layer Security Version 1.2", RFC 6347, DOI 10.17487/RFC6347, January 2012, <<https://www.rfc-editor.org/info/rfc6347>>.
- [RFC6454] Barth, A., "The Web Origin Concept", RFC 6454, DOI 10.17487/RFC6454, December 2011, <<https://www.rfc-editor.org/info/rfc6454>>.
- [RFC7022] Begen, A., Perkins, C., Wing, D., and E. Rescorla, "Guidelines for Choosing RTP Control Protocol (RTCP) Canonical Names (CNAMEs)", RFC 7022, DOI 10.17487/RFC7022, September 2013, <<https://www.rfc-editor.org/info/rfc7022>>.
- [RFC7675] Perumal, M., Wing, D., Ravindranath, R., Reddy, T., and M. Thomson, "Session Traversal Utilities for NAT (STUN) Usage for Consent Freshness", RFC 7675, DOI 10.17487/RFC7675, October 2015, <<https://www.rfc-editor.org/info/rfc7675>>.
- [RFC7918] Langley, A., Modadugu, N., and B. Moeller, "Transport Layer Security (TLS) False Start", RFC 7918, DOI 10.17487/RFC7918, August 2016, <<https://www.rfc-editor.org/info/rfc7918>>.
- [RFC8122] Lennox, J. and C. Holmberg, "Connection-Oriented Media Transport over the Transport Layer Security (TLS) Protocol in the Session Description Protocol (SDP)", RFC 8122, DOI 10.17487/RFC8122, March 2017, <<https://www.rfc-editor.org/info/rfc8122>>.
- [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>>.
- [RFC8259] Bray, T., Ed., "The JavaScript Object Notation (JSON) Data Interchange Format", STD 90, RFC 8259, DOI 10.17487/RFC8259, December 2017, <<https://www.rfc-editor.org/info/rfc8259>>.

- [RFC8261] Tuexen, M., Stewart, R., Jesup, R., and S. Loreto, "Datagram Transport Layer Security (DTLS) Encapsulation of SCTP Packets", RFC 8261, DOI 10.17487/RFC8261, November 2017, <<https://www.rfc-editor.org/info/rfc8261>>.
- [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>>.
- [webcrypto] editors, W., "Web Cryptography API", June 2013.  
Available at <http://www.w3.org/TR/WebCryptoAPI/>
- [webrtc-api] editors, W., "WebRTC 1.0: Real-time Communication Between Browsers", October 2011.  
Available at <http://dev.w3.org/2011/webrtc/editor/webrtc.html>

### 13.2. Informative References

- [RFC3261] Rosenberg, J., Schulzrinne, H., Camarillo, G., Johnston, A., Peterson, J., Sparks, R., Handley, M., and E. Schooler, "SIP: Session Initiation Protocol", RFC 3261, DOI 10.17487/RFC3261, June 2002, <<https://www.rfc-editor.org/info/rfc3261>>.
- [RFC5705] Rescorla, E., "Keying Material Exporters for Transport Layer Security (TLS)", RFC 5705, DOI 10.17487/RFC5705, March 2010, <<https://www.rfc-editor.org/info/rfc5705>>.
- [RFC6120] Saint-Andre, P., "Extensible Messaging and Presence Protocol (XMPP): Core", RFC 6120, DOI 10.17487/RFC6120, March 2011, <<https://www.rfc-editor.org/info/rfc6120>>.
- [RFC6265] Barth, A., "HTTP State Management Mechanism", RFC 6265, DOI 10.17487/RFC6265, April 2011, <<https://www.rfc-editor.org/info/rfc6265>>.
- [RFC6455] Fette, I. and A. Melnikov, "The WebSocket Protocol", RFC 6455, DOI 10.17487/RFC6455, December 2011, <<https://www.rfc-editor.org/info/rfc6455>>.



[RFC6943] Thaler, D., Ed., "Issues in Identifier Comparison for Security Purposes", RFC 6943, DOI 10.17487/RFC6943, May 2013, <<https://www.rfc-editor.org/info/rfc6943>>.

[RFC7617] Reschke, J., "The 'Basic' HTTP Authentication Scheme", RFC 7617, DOI 10.17487/RFC7617, September 2015, <<https://www.rfc-editor.org/info/rfc7617>>.

[XmlHttpRequest]  
van Kesteren, A., "XMLHttpRequest Level 2", January 2012.

Author's Address

Eric Rescorla  
RTFM, Inc.  
2064 Edgewood Drive  
Palo Alto, CA 94303  
USA

Phone: +1 650 678 2350  
Email: [ekr@rtfm.com](mailto:ekr@rtfm.com)

RTCWEB Working Group  
Internet-Draft  
Intended status: Informational  
Expires: July 27, 2015

C. Holmberg  
S. Hakansson  
G. Eriksson  
Ericsson  
January 23, 2015

Web Real-Time Communication Use-cases and Requirements  
draft-ietf-rtcweb-use-cases-and-requirements-16.txt

Abstract

This document describes web based real-time communication use-cases. Requirements on the browser functionality are derived from the use-cases.

This document was developed in an initial phase of the work with rather minor updates at later stages. It has not really served as a tool in deciding features or scope for the WGs efforts so far. It is being published to record the early conclusions of the working group. It will not be used as a set of rigid guidelines that specifications and implementations will be held to in the future.

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 July 27, 2015.

Copyright Notice

Copyright (c) 2015 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 . . . . .	3
2. Conventions . . . . .	3
3. Use-cases . . . . .	3
3.1. Introduction . . . . .	4
3.2. Common requirements . . . . .	4
3.3. Browser-to-browser use-cases . . . . .	4
3.3.1. Simple Video Communication Service . . . . .	4
3.3.2. Simple Video Communication Service, NAT/Firewall that blocks UDP . . . . .	7
3.3.3. Simple Video Communication Service, Firewall that only allows traffic via a HTTP Proxy . . . . .	7
3.3.4. Simple Video Communication Service, global service provider . . . . .	7
3.3.5. Simple Video Communication Service, enterprise aspects . . . . .	8
3.3.6. Simple Video Communication Service, access change . .	9
3.3.7. Simple Video Communication Service, QoS . . . . .	10
3.3.8. Simple Video Communication Service with screen sharing . . . . .	10
3.3.9. Simple Video Communication Service with file exchange	11
3.3.10. Hockey Game Viewer . . . . .	11
3.3.11. Multiparty video communication . . . . .	12
3.3.12. Multiparty on-line game with voice communication . .	14
3.4. Browser - GW/Server use cases . . . . .	15
3.4.1. Telephony terminal . . . . .	15
3.4.2. Fedex Call . . . . .	16
3.4.3. Video conferencing system with central server . . . .	17
4. Requirements summary . . . . .	18
4.1. General . . . . .	18
4.2. Browser requirements . . . . .	18
5. IANA Considerations . . . . .	22
6. Security Considerations . . . . .	22
6.1. Introduction . . . . .	22
6.2. Browser Considerations . . . . .	22
6.3. Web Application Considerations . . . . .	23
7. Acknowledgements . . . . .	23
8. Change Log . . . . .	23
9. Normative References . . . . .	30
Appendix A. API requirements . . . . .	30

Authors' Addresses . . . . .	33
------------------------------	----

## 1. Introduction

This document presents a few use-cases of web applications that are executed in a browser and use real-time communication capabilities. In most of the use-cases all end-user clients are web applications, but there are some use-cases where at least one of the end-user clients is of another type (e.g. a mobile phone or a SIP User Agent (UA)).

Based on the use-cases, the document derives requirements related to browser functionality. These requirements are named "Fn", where n is an integer, and are listed in conjunction with the use-cases. A summary is provided in Section 4.2.

This document was developed in an initial phase of the work with rather minor updates at later stages. It has not really served as a tool in deciding features or scope for the WGs efforts so far. It is proposed to be used in a later phase to evaluate the protocols and solutions developed by the WG.

This document also lists requirements related to the API to be used by web applications as an appendix. The reason is that the W3C WebRTC WG has decided to not develop its own use-case/requirement document, but instead use this document. These requirements are named "An", where n is an integer, and are described in Appendix A.

This document was developed in an initial phase of the work with rather minor updates at later stages. It has not really served as a tool in deciding features or scope for the WGs efforts so far. It is being published to record the early conclusions of the working group. It will not be used as a set of rigid guidelines that specifications and implementations will be held to in the future.

## 2. Conventions

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14, RFC 2119 [RFC2119].

## 3. Use-cases

### 3.1. Introduction

This section describes web based real-time communication use-cases, from which requirements are derived.

The following considerations are applicable to all use cases:

- o Clients can be on IPv4-only
- o Clients can be on IPv6-only
- o Clients can be on dual-stack
- o Clients can be connected to networks with different throughput capabilities
- o Clients can be on variable-media-quality networks (wireless)
- o Clients can be on congested networks
- o Clients can be on firewalled networks with no UDP allowed
- o Clients can be on networks with a NAT or IPv4-IPv6 translation devices using any type of Mapping and Filtering behaviors (as described in RFC4787).

### 3.2. Common requirements

The requirements retrieved from the Simple Video Communication Service use-case (Section 3.3.1) by default apply to all other use-cases, and are considered common. For each individual use-case, only the additional requirements are listed.

### 3.3. Browser-to-browser use-cases

#### 3.3.1. Simple Video Communication Service

##### 3.3.1.1. Description

Two or more users have loaded a video communication web application into their browsers, provided by the same service provider, and logged into the service it provides. The web service publishes information about user login status by pushing updates to the web application in the browsers. When one online user selects a peer online user, a 1-1 audiovisual communication session between the browsers of the two peers is initiated. The invited user might accept or reject the session.

During session establishment a self-view is displayed, and once the session has been established the video sent from the remote peer is displayed in addition to the self-view. During the session, each user can select to remove and re-insert the self-view as often as desired. Each user can also change the sizes of his/her two video displays during the session. Each user can also pause sending of media (audio, video, or both) and mute incoming media.

It is essential that media and data be encrypted, authenticated and integrity protected on a per IP packet basis and that media and data packets failing the integrity check not be delivered to the application.

The application gives the users the opportunity to stop it from exposing the host IP address to the application of the other user.

Any session participant can end the session at any time.

The two users may be using communication devices with different operating systems and browsers from different vendors.

The web service monitors the quality of the service (focus on quality of audio and video) the end-users experience.

#### 3.3.1.2. Common Requirements

REQ-ID	DESCRIPTION
F1	The browser must be able to use microphones and cameras as input devices to generate streams.
F2	The browser must be able to send streams and data to a peer in the presence of NATs.
F3	Transmitted streams and data must be rate controlled (meaning that the browser must, regardless of application behavior, reduce send rate when there is congestion).
F4	The browser must be able to receive, process and render streams and data ("render" does not apply for data) from peers.
F5	The browser should be able to render good quality audio and video even in the presence of reasonable levels of jitter and packet losses.

- F6        The browser must detect when a stream from a peer is not received anymore.
- 
- F7        When there are both incoming and outgoing audio streams, echo cancellation must be made available to avoid disturbing echo during conversation.
- 
- F8        The browser must support synchronization of audio and video.
- 
- F9        The browser should use encoding of streams suitable for the current rendering (e.g. video display size) and should change parameters if the rendering changes during the session.
- 
- F10       The browser must support a baseline audio and video codec.
- 
- F11       It must be possible to protect streams and data from wiretapping [RFC2804][RFC7258].
- 
- F12       The browser must enable verification, given the right circumstances and by use of other trusted communication, that streams and data received have not been manipulated by any party.
- 
- F13       The browser must encrypt, authenticate and integrity protect media and data on a per IP packet basis, and must drop incoming media and data packets that fail the per IP packet integrity check. In addition, the browser must support a mechanism for cryptographically binding media and data security keys to the user identity (see R-ID-BINDING in [RFC5479]).
- 
- F14       The browser must make it possible to set up a call between two parties without one party learning the other party's host IP address.
- 
- F15       The browser must be able to collect statistics, related to the transport of audio and video between peers, needed to estimate quality of experience.
- 

A1, A2, A3, A4, A5, A6, A7, A8, A9, A10, A11, A12, A25, A26

### 3.3.2. Simple Video Communication Service, NAT/Firewall that blocks UDP

#### 3.3.2.1. Description

This use-case is almost identical to the Simple Video Communication Service use-case (Section 3.3.1). The difference is that one of the users is behind a NAT/Firewall that blocks UDP traffic.

#### 3.3.2.2. Additional Requirements

REQ-ID	DESCRIPTION
F18	The browser must be able to send streams and data to a peer in the presence of NATs and Firewalls that block UDP traffic.

### 3.3.3. Simple Video Communication Service, Firewall that only allows traffic via a HTTP Proxy

#### 3.3.3.1. Description

This use-case is almost identical to the Simple Video Communication Service use-case (Section 3.3.1). The difference is that one of the users is behind a Firewall that only allows traffic via a HTTP Proxy.

#### 3.3.3.2. Additional Requirements

REQ-ID	DESCRIPTION
F21	The browser must be able to send streams and data to a peer in the presence of Firewalls that only allows traffic via a HTTP Proxy, when Firewall policy allows WebRTC traffic.

### 3.3.4. Simple Video Communication Service, global service provider

#### 3.3.4.1. Description

This use-case is almost identical to the Simple Video Communication Service use-case (Section 3.3.1).



What is added is that the service provider is operating over large geographical areas (or even globally).

Assuming that the Interactive Connectivity Establishment (ICE) mechanism [RFC5245] will be used, this means that the service provider would like to be able to provide several STUN and TURN servers (via the app) to the browser; selection of which one(s) to use is part of the ICE processing. Other reasons for wanting to provide several STUN and TURN servers include support for IPv4 and IPv6, load balancing and redundancy.

Note that ICE support being mandatory does not preclude a WebRTC endpoint from supporting more traversal mechanisms than ICE using STUN and TURN.

#### 3.3.4.2. Additional Requirements

REQ-ID	DESCRIPTION
F19	The browser must be able to use several STUN and TURN servers

A22

#### 3.3.5. Simple Video Communication Service, enterprise aspects

##### 3.3.5.1. Description

This use-case is similar to the Simple Video Communication Service use-case (Section 3.3.1).

What is added is aspects when using the service in enterprises. ICE is assumed in the further description of this use-case.

An enterprise that uses a RTCWEB based web application for communication desires to audit all RTCWEB based application sessions used from inside the company towards any external peer. To be able to do this they deploy a TURN server that straddles the boundary between the internal and the external network.

The firewall will block all attempts to use STUN with an external destination unless they go to the enterprise auditing TURN server. In cases where employees are using RTCWEB applications provided by an external service provider they still want the traffic to stay inside their internal network and in addition not load the straddling TURN server, thus they deploy a STUN server allowing the RTCWEB client to

determine its server reflexive address on the internal side. Thus enabling cases where peers are both on the internal side to connect without the traffic leaving the internal network. It must be possible to configure the browsers used in the enterprise with network specific STUN and TURN servers. This should be possible to achieve by auto-configuration methods. The RTCWEB functionality will need to utilize both network specific STUN and TURN resources and STUN and TURN servers provisioned by the web application.

#### 3.3.5.2. Additional Requirements

REQ-ID	DESCRIPTION
--------	-------------

F20	The browser must support the use of STUN and TURN servers that are supplied by entities other than the web application (i.e. the network provider).
-----	---

#### 3.3.6. Simple Video Communication Service, access change

##### 3.3.6.1. Description

This use-case is almost identical to the Simple Video Communication Service use-case (Section 3.3.1). The difference is that the user changes network access during the session.

The communication device used by one of the users has several network adapters (Ethernet, WiFi, Cellular). The communication device is accessing the Internet using Ethernet, but the user has to start a trip during the session. The communication device automatically changes to use WiFi when the Ethernet cable is removed and then moves to cellular access to the Internet when moving out of WiFi coverage. The session continues even though the access method changes.

##### 3.3.6.2. Additional Requirements

REQ-ID	DESCRIPTION
--------	-------------

F17	The communication session must survive across a change of the network interface used by the session
-----	---

### 3.3.7. Simple Video Communication Service, QoS

#### 3.3.7.1. Description

This use-case is almost identical to the Simple Video Communication Service, access change use-case (Section 3.3.6). The use of Quality of Service (QoS) capabilities is added:

The user in the previous use case that starts a trip is behind a common residential router that supports differentiation of traffic. In addition, the user's provider of cellular access has QoS support enabled. The user is able to take advantage of the QoS support both when accessing via the residential router and when using cellular.

#### 3.3.7.2. Additional Requirements

REQ-ID	DESCRIPTION
F17	The communication session must survive across a change of the network interface used by the session
F22	The browser should be able to take advantage of available capabilities (supplied by network nodes) to differentiate voice, video and data appropriately.

### 3.3.8. Simple Video Communication Service with screen sharing

#### 3.3.8.1. Description

This use-case has the audio and video communication of the Simple Video Communication Service use-case (Section 3.3.1).

But in addition to this, one of the users can share what is being displayed on her/his screen with a peer. The user can choose to share the entire screen, part of the screen (part selected by the user) or what a selected application displays with the peer.

#### 3.3.8.2. Additional Requirements

REQ-ID	DESCRIPTION
F36	The browser must be able to generate streams using the entire user display, a specific area of the user's display or the information being displayed by a specific application.

A21

### 3.3.9. Simple Video Communication Service with file exchange

#### 3.3.9.1. Description

This use-case has the audio and video communication of the Simple Video Communication Service use-case (Section 3.3.1).

But in addition to this, the users can send and receive files stored in the file system of the device used.

#### 3.3.9.2. Additional Requirements

REQ-ID	DESCRIPTION
F35	The browser must be able to send reliable data traffic to a peer browser.

A21, A24

### 3.3.10. Hockey Game Viewer

#### 3.3.10.1. Description

An ice-hockey club uses an application that enables talent scouts to, in real-time, show and discuss games and players with the club manager. The talent scouts use a mobile phone with two cameras, one front facing and one rear facing.

The club manager uses a desktop, equipped with one camera, for viewing the game and discussing with the talent scout.

Before the game starts, and during game breaks, the talent scout and the manager have a 1-1 audiovisual communication session. On the mobile phone, only the camera facing the talent scout is used. On the user display of the mobile phone, the video of the club manager

is shown with a picture-in-picture thumbnail of the rear facing camera (self-view). On the display of the desktop, the video of the talent scout is shown with a picture-in-picture thumbnail of the desktop camera (self-view).

When the game is on-going, the talent scout activates the use of the front facing camera, and that stream is sent to the desktop (the stream from the rear facing camera continues to be sent all the time). The video stream captured by the front facing camera (that is capturing the game) of the mobile phone is shown in a big window on the desktop screen, with picture-in-picture thumbnails of the rear facing camera and the desktop camera (self-view). On the display of the mobile phone the game is shown (front facing camera) with picture-in-picture thumbnails of the rear facing camera (self-view) and the desktop camera. As the most important stream in this phase is the video showing the game, the application used in the talent scout's mobile sets higher priority for that stream.

### 3.3.10.2. Additional Requirements

REQ-ID	DESCRIPTION
F22	The browser should be able to take advantage of available capabilities (supplied by network nodes) to differentiate voice, video and data appropriately.
F25	The browser must be able to render several concurrent audio and video streams.

A17, A23

### 3.3.11. Multiparty video communication

#### 3.3.11.1. Description

In this use-case, the Simple Video Communication Service use-case (Section 3.3.1) is extended by allowing multiparty sessions. No central server is involved - the browser of each participant sends and receives streams to and from all other session participants. The web application in the browser of each user is responsible for setting up streams to all receivers.

In order to enhance the user experience, the web application renders the audio coming from different participants so that it is

experienced to come from different spatial locations. This is done automatically, but users can change how the different participants are placed in the (virtual) room. In addition the levels in the audio signals are adjusted before mixing.

Another feature intended to enhance the use experience is that the video window that displays the video of the currently speaking peer is highlighted.

Each video stream received is by default displayed in a thumbnail frame within the browser, but users can change the display size.

Note: What this use-case adds in terms of requirements is capabilities to send streams to and receive streams from several peers concurrently, as well as the capabilities to render the video from all received streams and be able to spatialize, level adjust and mix the audio from all received streams locally in the browser. It also adds the capability to measure the audio level/activity.

#### 3.3.11.2. Additional Requirements

REQ-ID	DESCRIPTION
F23	The browser must be able to transmit streams and data to several peers concurrently.
F24	The browser must be able to receive streams and data from multiple peers concurrently.
F25	The browser must be able to render several concurrent audio and video streams.
F26	The browser must be able to mix several audio streams.
F27	The browser must be able to apply spatialization effects to audio streams.
F28	The browser must be able to measure the voice activity level in audio streams.
F29	The browser must be able to change the voice activity level in audio streams.
A13, A14, A15, A16	

### 3.3.12. Multiparty on-line game with voice communication

#### 3.3.12.1. Description

This use case is based on the previous one. In this use-case, the voice part of the multiparty video communication use case is used in the context of an on-line game. The received voice audio media is rendered together with game sound objects. For example, the sound of a tank moving from left to right over the screen must be rendered and played to the user together with the voice media.

Quick updates of the game state is required, and have higher priority than the voice.

Note: the difference regarding local audio processing compared to the "Multiparty video communication" use-case is that other sound objects than the streams must be possible to be included in the spatialization and mixing. "Other sound objects" could for example be a file with the sound of the tank; that file could be stored locally or remotely.

#### 3.3.12.2. Additional Requirements

REQ-ID	DESCRIPTION
F22	The browser should be able to take advantage of available capabilities (supplied by network nodes) to differentiate voice, video and data appropriately.
F23	The browser must be able to transmit streams and data to several peers concurrently.
F24	The browser must be able to receive streams and data from multiple peers concurrently.
F25	The browser must be able to render several concurrent audio and video streams.
F26	The browser must be able to mix several audio streams.
F27	The browser must be able to apply spatialization effects when playing audio streams.
F28	The browser must be able to measure the voice activity level in audio streams.
F29	The browser must be able to change the voice activity level in audio streams.
F30	The browser must be able to process and mix sound objects (media that is retrieved from another source than the established media stream(s) with the peer(s) with audio streams.
F34	The browser must be able to send short latency unreliable datagram traffic to a peer browser [RFC5405].

A13, A14, A15, A16, A17, A18, A23

#### 3.4. Browser - GW/Server use cases

##### 3.4.1. Telephony terminal



#### 3.4.1.1. Description

A mobile telephony operator allows its customers to use a web browser to access their services. After a simple log in the user can place and receive calls in the same way as when using a normal mobile phone. When a call is received or placed, the identity is shown in the same manner as when a mobile phone is used.

Note: With "place and receive calls in the same way as when using a normal mobile phone" it is meant that you can dial a number, and that your mobile telephony operator has made available your phone contacts on line, so they are available and can be clicked to call, and be used to present the identity of an incoming call. If the callee is not in your phone contacts the number is displayed. Furthermore, your call logs are available, and updated with the calls made/received from the browser. And for people receiving calls made from the web browser the usual identity (i.e. the phone number of the mobile phone) will be presented.

#### 3.4.1.2. Additional Requirements

REQ-ID	DESCRIPTION
F31	The browser must support an audio media format (codec) that is commonly supported by existing telephony services.
F33	The browser must be able to initiate and accept a media session where the data needed for establishment can be carried in SIP.

#### 3.4.2. Fedex Call

##### 3.4.2.1. Description

Alice uses her web browser with a service that allows her to call PSTN numbers. Alice calls 1-800-gofedex. Alice should be able to hear the initial prompts from the fedex Interactive Voice Responder (IVR) and when the IVR says press 1, there should be a way for Alice to navigate the IVR.

##### 3.4.2.2. Additional Requirements

REQ-ID	DESCRIPTION
F31	The browser must support an audio media format (codec) that is commonly supported by existing telephony services.
F32	There should be a way to navigate a Dual-tone multi-frequency signaling (DTMF) based Interactive voice response (IVR) System

### 3.4.3. Video conferencing system with central server

#### 3.4.3.1. Description

An organization uses a video communication system that supports the establishment of multiparty video sessions using a central conference server.

The browser of each participant sends an audio stream (type in terms of mono, stereo, 5.1, ... depending on the equipment of the participant) to the central server. The central server mixes the audio streams (and can in the mixing process naturally add effects such as spatialization) and sends towards each participant a mixed audio stream which is played to the user.

The browser of each participant sends video towards the server. For each participant one high resolution video is displayed in a large window, while a number of low resolution videos are displayed in smaller windows. The server selects what video streams to be forwarded as main- and thumbnail videos respectively, based on speech activity. As the video streams to display can change quite frequently (as the conversation flows) it is important that the delay from when a video stream is selected for display until the video can be displayed is short.

All participants are authenticated by the central server, and authorized to connect to the central server. The participants are identified to each other by the central server, and the participants do not have access to each others' credentials such as e-mail addresses or login IDs.

Note: This use-case adds requirements on support for fast stream switches F16. There exist several solutions that enable the server to forward one high resolution and several low resolution video streams: a) each browser could send a high resolution, but scalable

stream, and the server could send just the base layer for the low resolution streams, b) each browser could in a simulcast fashion send one high resolution and one low resolution stream, and the server just selects or c) each browser sends just a high resolution stream, the server transcodes into low resolution streams as required.

#### 3.4.3.2. Additional Requirements

REQ-ID	DESCRIPTION
F16	The browser must support insertion of reference frames in outgoing media streams when requested by a peer.
F25	The browser must be able to render several concurrent audio and video streams.

### 4. Requirements summary

#### 4.1. General

This section contains the requirements on the browser derived from the use-cases in Section 3.

NOTE: It is assumed that the user applications are executed on a browser. Whether the capabilities to implement specific browser requirements are implemented by the browser application, or are provided to the browser application by the underlying operating system, is outside the scope of this document.

#### 4.2. Browser requirements

Common, basic requirements	
REQ-ID	DESCRIPTION
F1	The browser must be able to use microphones and cameras as input devices to generate streams.
F2	The browser must be able to send streams and data to a peer in the presence of NATs.
F3	Transmitted streams and data must be rate controlled (meaning that the browser must, regardless of application behavior, reduce send rate when there is congestion).

- 
- F4        The browser must be able to receive, process and render streams and data ("render" does not apply for data) from peers.
- 
- F5        The browser should be able to render good quality audio and video even in the presence of reasonable levels of jitter and packet losses.
- 
- F6        The browser must detect when a stream from a peer is not received anymore
- 
- F7        When there are both incoming and outgoing audio streams, echo cancellation must be made available to avoid disturbing echo during conversation.
- 
- F8        The browser must support synchronization of audio and video.
- 
- F9        The browser should use encoding of streams suitable for the current rendering (e.g. video display size) and should change parameters if the rendering changes during the session
- 
- F10       The browser must support a baseline audio and video codec
- 
- F11       It must be possible to protect streams and data from wiretapping [RFC2804][RFC7258].
- 
- F12       The browser must enable verification, given the right circumstances and by use of other trusted communication, that streams and data received have not been manipulated by any party.
- 
- F13       The browser must encrypt, authenticate and integrity protect media and data on a per-packet basis, and must drop incoming media and data packets that fail the per-packet integrity check. In addition, the browser must support a mechanism for cryptographically binding media and data security keys to the user identity (see R-ID-BINDING in [RFC5479]).
- 
- F14       The browser must make it possible to set up a call between two parties without one party

learning the other party's host IP address.

-----  
F15      The browser must be able to collect statistics,  
          related to the transport of audio and video  
          between peers, needed to estimate quality of  
          experience.  
-----

Requirements related to network and topology

-----  
REQ-ID      DESCRIPTION  
-----

F16      The browser must support insertion of reference frames  
          in outgoing media streams when requested by a peer.  
-----

F17      The communication session must survive across a  
          change of the network interface used by the  
          session  
-----

F18      The browser must be able to send streams and  
          data to a peer in the presence of NATs and  
          Firewalls that block UDP traffic.  
-----

F19      The browser must be able to use several STUN  
          and TURN servers  
-----

F20      The browser must support the use of STUN and TURN  
          servers that are supplied by entities other than  
          the web application (i.e. the network provider).  
-----

F21      The browser must be able to send streams and  
          data to a peer in the presence of Firewalls that only  
          allows traffic via a HTTP Proxy, when Firewall policy  
          allows WebRTC traffic.  
-----

F22      The browser should be able to take advantage  
          of available capabilities (supplied by network  
          nodes) to differentiate voice, video and data  
          appropriately.  
-----

Requirements related to multiple peers and streams

-----  
REQ-ID      DESCRIPTION  
-----

F23      The browser must be able to transmit streams and  
          data to several peers concurrently.  
-----

F24      The browser must be able to receive streams and  
          data from multiple peers concurrently.  
-----

---

F25      The browser must be able to render several concurrent audio and video streams.

---

F26      The browser must be able to mix several audio streams.

---

Requirements related to audio processing

---

REQ-ID	DESCRIPTION
--------	-------------

---

F27	The browser must be able to apply spatialization effects when playing audio streams.
-----	--

---

F28	The browser must be able to measure the voice activity level in audio streams.
-----	--

---

F29	The browser must be able to change the voice activity level in audio streams.
-----	---

---

F30	The browser must be able to process and mix sound objects (media that is retrieved from another source than the established media stream(s) with the peer(s) with audio streams.
-----	--

---

Requirements related to legacy interop

---

REQ-ID	DESCRIPTION
--------	-------------

---

F31	The browser must support an audio media format (codec) that is commonly supported by existing telephony services.
-----	---

---

F32	There should be a way to navigate a Dual-tone multi-frequency signaling (DTMF) based Interactive voice response (IVR) System
-----	--

---

F33	The browser must be able to initiate and accept a media session where the data needed for establishment can be carried in SIP.
-----	--

---

Other requirements

---

REQ-ID	DESCRIPTION
--------	-------------

---

F34	The browser must be able to send short latency unreliable datagram traffic to a peer browser [RFC5405].
-----	---

---

- 
- F35      The browser must be able to send reliable data traffic to a peer browser.
- 
- F36      The browser must be able to generate streams using the entire user display, a specific area of the user's display or the information being displayed by a specific application.
- 

## 5. IANA Considerations

There are no IANA actions in this document.

## 6. Security Considerations

### 6.1. Introduction

A malicious web application might use the browser to perform Denial Of Service (DOS) attacks on NAT infrastructure, or on peer devices. Also, a malicious web application might silently establish outgoing, and accept incoming, streams on an already established connection.

Based on the identified security risks, this section will describe security considerations for the browser and web application.

### 6.2. Browser Considerations

The browser is expected to provide mechanisms for getting user consent to use device resources such as camera and microphone.

The browser is expected to provide mechanisms for informing the user that device resources such as camera and microphone are in use ("hot").

The browser must provide mechanisms for users to revise and even completely revoke consent to use device resources such as camera and microphone.

The browser is expected to provide mechanisms for getting user consent to use the screen (or a certain part of it) or what a certain application displays on the screen as source for streams.

The browser is expected to provide mechanisms for informing the user that the screen, part thereof or an application is serving as a stream source ("hot").

The browser must provide mechanisms for users to revise and even completely revoke consent to use the screen, part thereof or an application is serving as a stream source.

The browser is expected to provide mechanisms in order to assure that streams are the ones the recipient intended to receive.

The browser is expected to provide mechanisms that allows the users to verify that the streams received have not be manipulated (F12).

The browser needs to ensure that media is not sent, and that received media is not rendered, until the associated stream establishment and handshake procedures with the remote peer have been successfully finished.

The browser needs to ensure that the stream negotiation procedures are not seen as Denial Of Service (DOS) by other entities.

### 6.3. Web Application Considerations

The web application is expected to ensure user consent in sending and receiving media streams.

## 7. Acknowledgements

The authors wish to thank Bernard Aboba, Gunnar Hellstrom, Martin Thomson, Lars Eggert, Matthew Kaufman, Emil Ivov, Eric Rescorla, Eric Burger, John Leslie, Dan Wing, Richard Barnes, Barry Dingle, Dale Worley, Ted hardie, Mary Barnes, Dan Burnett, Stephan Wenger, Harald Alvestrand, Cullen Jennings, Andrew Hutton and everyone else in the RTCWEB community that have provided comments, feedback, text and improvement proposals on the document. A big thank you to everyone that provided comments as part of the IESG evaluation, and to everyone else that provided comments and input in order to improve the document.

## 8. Change Log

[RFC EDITOR NOTE: Please remove this section when publishing]

Changes from draft-ietf-rtcweb-use-cases-and-requirements-15

- o Changes based on comment from Stephen Farrell:



- o - A1 modified, to also cover access to the local file system.
- o Changes based on comments from Benoit Claise:
- o - RFC 5245 added to references.
- o - Note added to Annex A, indicating that the API requirements are not normative.
- o Changes based on comments from Brian Carpenter:
- o - RFC 7258 added to references.
- o - Terminology fixes:
- o -- 'prioritize' -> 'differentiate'.
- o -- 'prioritization' -> 'differentiation'.

#### Changes from draft-ietf-rtcweb-use-cases-and-requirements-14

- o Changes based on comments from the ops-dir:
- o - Editorial fixes.
- o - F13: 'per-packet basis' -> 'per IP packet basis'.
- o - F22: Text corrected in one occurrence.
- o - F25: 'audio' added.
- o Changes based on comments from IESG
- o - Editorial fixes.
- o - Disclaimer text suggested by Alissa Cooper added.
- o - F11: Reference to RFC 7258 added.
- o - F27: 'when playing' removed.

#### Changes from draft-ietf-rtcweb-use-cases-and-requirements-10

- o Described that the API requirements are really from a W3C perspective and are supplied as an appendix in the introduction. Moved API requirements to an Appendix.

- o Removed the "Conventions" section with the key-words and reference to RFC2119. Also changed uppercase MUST's/SHOULD's to lowercase.
- o Added a note on the proposed use of the document to the introduction.
- o Removed the note talking about WS from the "Firewall that only allows http" use-case.
- o Removed the word "Skype" that was used as example in one of the use-cases.
- o Clarified F3 (the req saying the everything the browser sends must be rate controlled).
- o Removed the TBD saying we need to define reasonable levels from the requirement saying that quality must be good even in presence of packet losses (F5), and changed "must" to "should" (Based on a list discussion involving Bernard).
- o Removed F6 ("The browser must be able to handle high loss and jitter levels in a graceful way."), also after a list discussion.
- o Clarified F7 (used to say that the browser must support fast stream switches, now says that reference frames must be inserted when requested).
- o Removed the questions from F9 (echo cancellation), F10 (synchronization), F21 (telephony codec).
- o Exchanged "restrictive firewalls" for "limited middleboxes" in F19 (as proposed by Martin).
- o Expanded DTMF and IVR in F22 (proposed by Martin)
- o Added ref to RFC5405 in F23 (proposed by Lars Eggert).
- o Exchanged "service provided" for "web application" in F32.
- o Changed the text in 3.2.1 that motivates F36 (new text "It is essential that media and data be encrypted, authenticated ... bound to the user identity."); and rewrote F36, included a ref to RFC5479.
- o Changed "quality of service" to "quality of experience" in F38.
- o Added F39.

- o Used new formulation of A17 (proposed by Martin).
- o Updated A20.
- o Updated A25.

Changes from draft-ietf-rtcweb-use-cases-and-requirements-09

- o Changed "video communication session" to "audiovisual communication session."

Changes from draft-ietf-rtcweb-use-cases-and-requirements-08

- o Changed "eavesdropping" to "wiretapping" and referenced RFC2804.
- o Removed informal ref webrtc\_req; that document has been abandoned by the W3C webrtc WG.
- o Added use-case where one user is behind a Firewall that only allows http; derived req. F37.
- o Changed F24 slightly; MUST-> SHOULD, inserted "available".
- o Added a clause to "Simple video communication service" saying that the service provider monitors the quality of service, and derived reqs F38 and A26.

Changes from draft-ietf-rtcweb-use-cases-and-requirements-07

- o Added "and data exchange" to 1. Introduction.
- o Removed cone and symmetric NAT from 4.1 Introduction, refers to RFC4787 instead.
- o Added text on enabling verification of that the media has not been manipulated by anyone to use-case "Simple Video Communication Service", derived req. F35
- o Added text on that the browser should reject media (data) that has been created/injected/modified by non-trusted party, derived req. F36
- o Added text on enabling the app to refrain from revealing IP address to use-case "Simple Video Communication Service", derived req. A25
- o Added use-case "Simple Video Communication Service with file exchange", derived reqs F33 and A24

- o Added priority of video streams to "Hockey game viewer" use case, added priority of data to "on-line game use-case", derived reqs F34 and A23
- o In F22, "the IVR" -> "a DTMF based IVR".
- o Updated req F23 to clarify that requirements such as NAT traversal, protection from eavesdropping, rate control applies also to datagram.

Changes from draft-ietf-rtcweb-use-cases-and-requirements-06

- o Renaming of requirements (FaI1 -> F31), (FaI2 -> F32) and (AaI1 -> A22)

Changes from draft-ietf-rtcweb-use-cases-and-requirements-05

- o Added use-case "global service provider", derived reqs associated with several STUN/TURN servers
- o Added use-case "enterprise aspects", derived req associated with enabling the network provider to supply STUN and TURN servers
- o The requirements from the above are ICE specific and labeled accordingly
- o Separated the requirements phrased like "processing such as pan, mix and render" for audio to be specific reqs on spatialization, level measurement, level adjustment and mixing (discussed on the lists in <http://www.ietf.org/mail-archive/web/rtcweb/current/msg01648.html> and <http://lists.w3.org/Archives/Public/public-webrtc/2011Sep/0102.html>)
- o Added use-case on sharing as decided in <http://www.ietf.org/mail-archive/web/rtcweb/current/msg01700.html>, derived reqs F30 and A21
- o Added the list of common considerations proposed in mail <http://www.ietf.org/mail-archive/web/rtcweb/current/msg01562.html> to the Introduction of the use-case section

Changes from draft-ietf-rtcweb-use-cases-and-requirements-04

- o Most changes based on the input from Dan Burnett <http://www.ietf.org/mail-archive/web/rtcweb/current/msg00948.html>
- o Many editorial changes
- o 4.2.1.1 Clarified

- o Some clarification added to 4.3.1.1 as a note
- o F-requirements updated (see reply to Dan's mail).
- o Almost all A-requirements updated to start "The Web API MUST provide ..."
- o A8 removed, A9 rephrased to cover A8 and old A9
- o A15 rephrased
- o For more details, and discussion, look at the response to Dan's mail <http://www.ietf.org/mail-archive/web/rtcweb/current/msg01177.html>

#### Changes from draft-ietf-rtcweb-use-cases-and-requirements-03

- o Editorials
- o Changed when the self-view is displayed in 4.2.1.1, and added words about allowing users to remove and re-insert it.
- o Clarified 4.2.6.1
- o Removed the "mono" stuff from 4.2.7.1
- o Added that communication should not be possible to eavesdrop to most use cases - and req. F17
- o Re-phrased 4.3.3.1 to not describe the technical solution so much, and removed "stereo" stuff. Solution possibilities are now in a note.
- o Re-inserted API requirements after discussion in the W3C webrtc WG. (Re-phrased A15 and added A18 compared to version -02).

#### Changes from draft-ietf-rtcweb-use-cases-and-requirements-02

- o Removed description/list of API requirements, instead
- o Reference to W3C webrtc\_reqs document for API requirements

#### Changes from draft-ietf-rtcweb-ucreqs-01

- o Changed Intended status to Information
- o Changed "Ipr" to "trust200902"

- o Added use case "Simple video communication service, NAT/Firewall that blocks UDP", and derived new req F26
- o Added use case "Distributed Music Band" and derived new req A17
- o Added F24 as requirement derived from use case "Simple video communication service with inter-operator calling"
- o Added section "Additional use cases"
- o Added text about ID handling to multiparty with central server use case
- o Re-phrased A1 slightly

Changes from draft-ietf-rtcweb-ucreqs-00

- o - Reshuffled: Just two main groups of use cases (b2b and b2GW/Server); removed some specific use cases and added them instead as flavors to the base use case (Simple video communication)
- o - Changed the formulation of F19
- o - Removed the requirement on an API for DTMF
- o - Removed "FX3: There SHOULD be a mapping of the minimum needed data for setting up connections into SIP, so that the restriction to SIP-carriable data can be verified. Not a rew on the browser but rather on a document"
- o - (see <http://www.ietf.org/mail-archive/web/rtcweb/current/msg00227.html> for more details)
- o -Added text on informing user of that mic/cam is being used and that it must be possible to revoke permission to use them in section 7.

Changes from draft-holmberg-rtcweb-ucreqs-01

- o - Draft name changed to draft-ietf-rtcweb-ucreqs
- o - Use-case grouping introduced
- o - Additional use-cases added
- o - Additional reqs added (derived from use cases): F19-F25, A16-A17

Changes from draft-holmberg-rtcweb-ucreqs-00

- o - Mapping between use-cases and requirements added (Harald Alvestrand, 090311)
- o - Additional security considerations text (Harald Alvestrand, 090311)
- o - Clarification that user applications are assumed to be executed by a browser (Ted Hardie, 080311)
- o - Editorial corrections and clarifications

## 9. Normative References

- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.
- [RFC2804] IAB and IESG, "IETF Policy on Wiretapping", RFC 2804, May 2000.
- [RFC5245] Rosenberg, J., "Interactive Connectivity Establishment (ICE): A Protocol for Network Address Translator (NAT) Traversal for Offer/Answer Protocols", RFC 5245, April 2010.
- [RFC5405] Eggert, L. and G. Fairhurst, "Unicast UDP Usage Guidelines for Application Designers", BCP 145, RFC 5405, November 2008.
- [RFC5479] Wing, D., Fries, S., Tschofenig, H., and F. Audet, "Requirements and Analysis of Media Security Management Protocols", RFC 5479, April 2009.
- [RFC7258] Farrell, S. and H. Tschofenig, "Pervasive Monitoring Is an Attack", BCP 188, RFC 7258, May 2014.

## Appendix A. API requirements

This section contains the requirements on the API derived from the use-cases in Section 3.

NOTE: As W3C is responsible for the API, the API requirements in this specification are not normative.

REQ-ID	DESCRIPTION
-----	
A1	The Web API must provide means for the application to ask the browser for permission

to use cameras and microphones as input devices,  
and to have access to the local file system.

- 
- A2      The Web API must provide means for the web application to control how streams generated by input devices are used.
- 
- A3      The Web API must provide means for the web application to control the local rendering of streams (locally generated streams and streams received from a peer).
- 
- A4      The Web API must provide means for the web application to initiate sending of stream/stream components to a peer.
- 
- A5      The Web API must provide means for the web application to control the media format (codec) to be used for the streams sent to a peer.
- NOTE: The level of control depends on whether the codec negotiation is handled by the browser or the web application.
- 
- A6      The Web API must provide means for the web application to modify the media format for streams sent to a peer after a media stream has been established.
- 
- A7      The Web API must provide means for informing the web application of whether the establishment of a stream with a peer was successful or not.
- 
- A8      The Web API must provide means for the web application to mute/unmute a stream or stream component(s). When a stream is sent to a peer mute status must be preserved in the stream received by the peer.
- 
- A9      The Web API must provide means for the web application to cease the sending of a stream to a peer.
- 
- A10     The Web API must provide means for the web application to cease processing and rendering of a stream received from a peer.
-



- A11        The Web API must provide means for informing the web application when a stream from a peer is no longer received.
- 
- A12        The Web API must provide means for informing the web application when high loss rates occur.
- 
- A13        The Web API must provide means for the web application to apply spatialization effects to audio streams.
- 
- A14        The Web API must provide means for the web application to detect the level in audio streams.
- 
- A15        The Web API must provide means for the web application to adjust the level in audio streams.
- 
- A16        The Web API must provide means for the web application to mix audio streams.
- 
- A17        The Web API must provide a way to identify streams such that an application is able to match streams on a sending peer with the same stream on all receiving peers.
- 
- A18        The Web API must provide a mechanism for sending and receiving isolated discrete chunks of data.
- 
- A19        The Web API must provide means for the web application to indicate the type of audio signal (speech, audio) for audio stream(s)/stream component(s).
- 
- A20        It must be possible for an initiator or a responder web application to indicate the types of media it is willing to accept incoming streams for when setting up a connection (audio, video, other). The types of media to be accepted can be a subset of the types of media the browser is able to accept.
- 
- A21        The Web API must provide means for the application to ask the browser for permission to the screen, a certain area on the screen or what a certain application displays on the

screen as input to streams.

- 
- A22      The Web API must provide means for the application to specify several STUN and/or TURN servers to use.
- 
- A23      The Web API must provide means for the application to specify the priority to apply for outgoing streams and data.
- 
- A24      The Web API must provide a mechanism for sending and receiving files.
- 
- A25      It must be possible for the application to instruct the browser to refrain from exposing the host IP address to the application
- 
- A26      The Web API must provide means for the application to obtain the statistics (related to transport, and collected by the browser) needed to estimate quality of service.
- 

#### Authors' Addresses

Christer Holmberg  
Ericsson  
Hirsalantie 11  
Jorvas 02420  
Finland

Email: [christer.holmberg@ericsson.com](mailto:christer.holmberg@ericsson.com)

Stefan Hakansson  
Ericsson  
Laboratoriegatan 11  
Lulea 97128  
Sweden

Email: [stefan.lk.hakansson@ericsson.com](mailto:stefan.lk.hakansson@ericsson.com)

Goran AP Eriksson  
Ericsson  
Farogatan 6  
Stockholm 16480  
Sweden

Email: [goran.ap.eriksson@ericsson.com](mailto:goran.ap.eriksson@ericsson.com)

RTCWeb Working Group  
Internet-Draft  
Intended status: Standards Track  
Expires: August 30, 2013

R. Jesup  
Mozilla  
S. Loreto  
Ericsson  
M. Tuexen  
Muenster Univ. of Appl. Sciences  
February 26, 2013

WebRTC Data Channel Protocol  
draft-jesup-rtcweb-data-protocol-04.txt

Abstract

The Web Real-Time Communication (WebRTC) working group is charged to provide protocols to support for direct interactive rich communication using audio, video, and data between two peers' web-browsers. This document specifies an actual (minor) protocol for how the JS-layer DataChannel objects provide the data channels between the peers.

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 August 30, 2013.

Copyright Notice

Copyright (c) 2013 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 . . . . .	3
2. Conventions . . . . .	3
3. Terminology . . . . .	3
4. Protocol Overview . . . . .	3
5. Opening Handshake . . . . .	4
6. Control Messages . . . . .	4
6.1. DATA_CHANNEL_OPEN Message . . . . .	4
7. Procedures . . . . .	6
7.1. Adding a Channel . . . . .	6
7.2. Closing a Channel . . . . .	7
7.3. Sending and Receiving Data . . . . .	7
8. Security Considerations . . . . .	7
9. IANA Considerations . . . . .	7
10. Acknowledgments . . . . .	8
11. References . . . . .	8
11.1. Normative References . . . . .	8
11.2. Informational References . . . . .	9
Authors' Addresses . . . . .	9

## 1. Introduction

The DataChannel Protocol is designed to provide, in the WebRTC context [I-D.ietf-rtcweb-overview], a generic transport service allowing Web Browser to exchange generic data in a bidirectional peer to peer fashion. As discussed in [I-D.ietf-rtcweb-data-channel] the protocol uses Stream Control Transmission Protocol (SCTP) [RFC4960] encapsulated on Datagram Transport Layer Security (DTLS) [RFC6347] as described in [I-D.tuexen-tsvwg-sctp-dtls-encaps] to benefit from their already standardized transport and security features.

## 2. Conventions

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

## 3. Terminology

This document uses the following terms:

Association: An SCTP association.

Stream: A unidirectional stream of an SCTP association. It is uniquely identified by a stream identifier.

Channel: Two Streams with the same identifier, one in each direction, that are managed together.

## 4. Protocol Overview

This protocol is a simple, low-overhead way to establish bidirectional Channels over an SCTP association with a consistent set of properties.

Channels are created by sending an DATA\_CHANNEL\_OPEN message on an unused Stream. There is no handshake, and the channel is available to send on as soon as the DATA\_CHANNEL\_OPEN has been sent.

To avoid glare in opening Channels, each side must use either even or odd Streams when sending a DATA\_CHANNEL\_OPEN message. The method used to determine which side uses odd or even is TBD and may be based on DTLS connection roles when used in rtcweb.

There is no attempt to resolve label glare; if both sides open a Channel labelled "x" at the same time, there will be two Channels labelled "x" - one on an even Stream pair, one on an odd pair.

The protocol field is to ease cross-application interoperability ("federation") by identifying the data being passed with an IANA-registered string.

Data that arrives which on an unused Stream MUST be held until a DATA\_CHANNEL\_OPEN arrives for that Channel, or if the protocol stack had been told to expect data on that Stream and deliver it immediately, or until [TBD - report error]. This allows for external negotiation of streams (or assumption of negotiation by cooperating applications). If a later DATA\_CHANNEL\_OPEN arrives that conflicts with the pre-set properties of the Channel, an error should be signaled to higher levels.

Channels are closed by resetting the Stream.

## 5. Opening Handshake

The opening handshake is based on the multimedia session description exchange that happens between the browsers, typically through a Web Server acting as the signaling service.

[I-D.ietf-mmusic-sctp-sdp] defines the protocol identifier, 'SCTP/DTLS', and defines how to establish an SCTP association over DTLS using the Session Description Protocol (SDP).

The SCTP association is created with the number of streams specified by the application, and if not specified, then it SHOULD default to 16 streams.

It is recommended that additional streams be available dynamically based on [RFC6525].

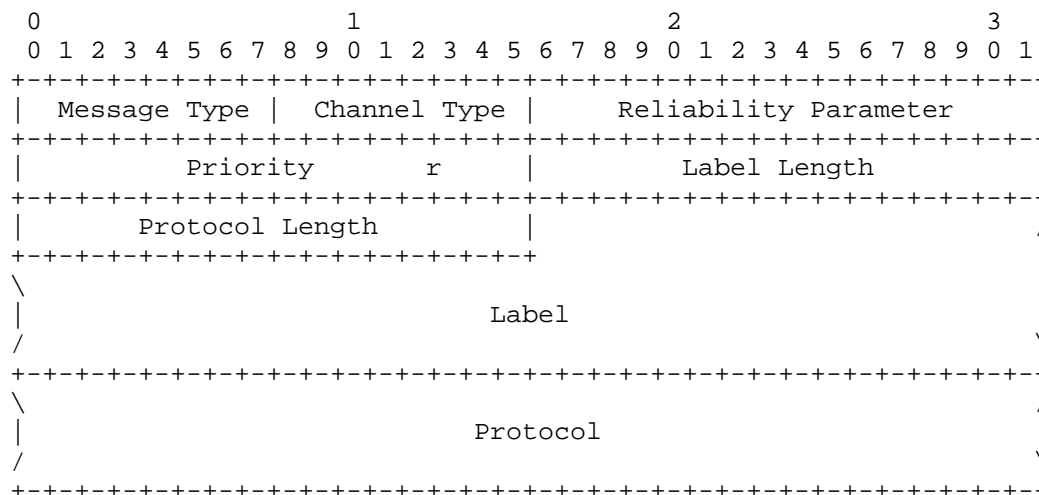
## 6. Control Messages

Control Messages are sent to manage opening bidirectional channels.

A DATA\_CHANNEL\_OPEN message is sent on the Stream that is intended to be used to send in that direction, and this creates a bidirectional Channel that may be used by both sides to send data.

### 6.1. DATA\_CHANNEL\_OPEN Message

This message is sent initially on the stream used for user messages using the channel. All DATA\_CHANNEL\_OPEN messages MUST be sent reliably and in-order.



Message Type: 1 byte (unsigned integer)

This field holds the IANA defined message type for the the DATA\_CHANNEL\_OPEN message. The suggested value of this field for IANA is 0x03. NOTE: values 0x00-0x02 were used in an older draft with incompatible structures. Any future incompatible message changes should define new message types.

Channel Type: 1 byte (unsigned integer)

This field specifies the type of the channel to be opened:

DATA\_CHANNEL\_RELIABLE (0x00): The channel provides a reliable in-order bi-directional communication channel.

DATA\_CHANNEL\_RELIABLE\_UNORDERED (0x80): The channel provides a reliable unordered bi-directional communication channel.

DATA\_CHANNEL\_PARTIAL\_RELIABLE\_REXMIT (0x01): The channel provides a partially-reliable in-order bi-directional Communication channel. User messages will not be retransmitted more times than specified in the Reliability Parameter.

DATA\_CHANNEL\_PARTIAL\_RELIABLE\_REXMIT\_UNORDERED (0x81): The channel provides a partial reliable unordered bi-directional Communication channel. User messages will not be retransmitted more times than specified in the Reliability Parameter.

DATA\_CHANNEL\_PARTIAL\_RELIABLE\_TIMED (0x02): The channel provides a partial reliable in-order bi-directional Communication channel. User messages might not be transmitted or retransmitted after a specified life-time given in milliseconds in the Reliability Parameter. This life-time starts when providing the user message to the Javascript engine.



DATA\_CHANNEL\_PARTIAL\_RELIABLE\_TIMED (0x82): The channel provides a partial reliable unordered bi-directional Communication channel. User messages might not be transmitted or retransmitted after a specified life-time given in milliseconds in the Reliability Parameter. This life-time starts when providing the user message to the Javascript engine.

Reliability Parameter: 2 bytes (unsigned integer)  
This field is ignored if a reliable channel is used.  
If a partial reliable channel with limited number of retransmissions is used, this field specifies the number of retransmissions. If a partial reliable channel with limited lifetime is used, this field specifies the maximum lifetime in milliseconds.

Priority: 2 bytes (integer)  
The priority of the channel.

Label Length: 2 bytes (integer)  
The length of the label field in bytes.

Protocol Length: 2 bytes (integer)  
The length of the protocol field in bytes.

Label: Variable Length (sequence of characters)  
The name of the channel. This may be an empty string.

Protocol: Variable Length (sequence of characters)  
The protocol for the channel. This may be an empty string. If used, it SHOULD be an IANA-registered protocol.

## 7. Procedures

### 7.1. Adding a Channel

When one side wants to add a channel, it picks an unused outgoing stream (either even or odd, depending on TBD); if no unused streams are available a negotiation to increase the number is done. It should also check that the other side has the same channel available, and if not then initiate an increase in the number of streams. It then sends a DATA\_CHANNEL\_OPEN control message on the outgoing stream.

When an DATA\_CHANNEL\_OPEN is received on an incoming stream, the Stream is associated with the newly-created Channel. If any data had arrived on the Stream before the Open arrives and had been buffered, it is now released on the new Channel.

The channel\_type and reliability\_parameters fields of the DATA\_CHANNEL\_OPEN message MUST be used to set up the reverse side of the Channel so that both directions use the same options by default.

## 7.2. Closing a Channel

Channels **MUST** be closed by resetting the outgoing stream. If an incoming stream is reset by the peer, an corresponding outgoing stream reset **SHOULD** be issued. If both streams of a channel are reset, the channel is closed and the streams are available for reuse for new channel opens.

## 7.3. Sending and Receiving Data

Data shall be sent using PPID's other than the Data Channel Control PPID. These PPID's should be registered with IANA via (TBD). The meaning of these data PPIDs and the format of the data shall be specific to the usage of this protocol, and typically shall be provided to the higher layers to allow proper decoding of the data.

It is **RECOMMENDED** that higher layers wishing to transfer large messages fragment them using PPIDs or other mechanisms to avoid monopolization of the SCTP association by the transfer of a single large message, unless a future SCTP draft relaxes this concern. If fragmented solely with PPID values, then transmission must occur on a reliable in-order channel. If in-band application framing is used, then other options may be possible.

For WebRTC, data PPID's for DOMStrings and binary data (and fragmentation thereof) shall be created.

All data sent on a Channel in both directions **MUST** be sent over the underlying Stream using the reliability defined when the Channel was opened unless the options are changed, or per-message options are specified by a higher level.

Data may be sent immediately after sending or receiving a DATA\_CHANNEL\_OPEN message.

It is recommended that message size be kept within certain size bounds (TBD) as applications will not be able to support arbitrarily-large single messages.

## 8. Security Considerations

To be done.

## 9. IANA Considerations

This document also defines three new SCTP Payload Protocol

Identifiers (PPIDs). RFC 4960 [RFC4960] creates the registry from which these identifiers have been assigned. The following values have been reserved:

WebRTC Control - #To Be Assigned

NOTE: not needed if Stream 0 is dedicated to control

DOMString - #To Be Assigned

Binary Data Partial - #To Be Assigned

Binary Data Last - #To Be Assigned

## 10. Acknowledgments

The authors wish to thank Martin Thompson, Cullen Jennings, Harald Alvestrand, Peter Thatcher, Adam Bergkvist, Justin Uberti, Randall Stewart, Stefan Haekansson and many others for their invaluable comments.

## 11. References

### 11.1. Normative References

- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.
- [RFC3758] Stewart, R., Ramalho, M., Xie, Q., Tuexen, M., and P. Conrad, "Stream Control Transmission Protocol (SCTP) Partial Reliability Extension", RFC 3758, May 2004.
- [RFC4960] Stewart, R., "Stream Control Transmission Protocol", RFC 4960, September 2007.
- [RFC6347] Rescorla, E. and N. Modadugu, "Datagram Transport Layer Security Version 1.2", RFC 6347, January 2012.
- [RFC6525] Stewart, R., Tuexen, M., and P. Lei, "Stream Control Transmission Protocol (SCTP) Stream Reconfiguration", RFC 6525, February 2012.
- [I-D.ietf-mmusic-sctp-sdp] Loreto, S. and G. Camarillo, "Stream Control Transmission Protocol (SCTP)-Based Media Transport in the Session Description Protocol (SDP)", draft-ietf-mmusic-sctp-sdp-03 (work in progress), January 2013.
- [I-D.tuexen-tsvwg-sctp-dtls-encaps] Jesup, R., Loreto, S., Stewart, R., and M. Tuexen, "DTLS

Encapsulation of SCTP Packets for RTCWEB",  
draft-tuexen-tsvwg-sctp-dtls-encaps-01 (work in progress),  
July 2012.

## 11.2. Informational References

[I-D.ietf-rtcweb-overview]

Alvestrand, H., "Overview: Real Time Protocols for Browser-based Applications", draft-ietf-rtcweb-overview-06 (work in progress), February 2013.

[I-D.ietf-rtcweb-data-channel]

Jesup, R., Loreto, S., and M. Tuexen, "RTCWeb Data Channels", draft-ietf-rtcweb-data-channel-03 (work in progress), February 2013.

## Authors' Addresses

Randell Jesup  
Mozilla  
US

Email: randell-ietf@jesup.org

Salvatore Loreto  
Ericsson  
Hirsalantie 11  
Jorvas 02420  
FI

Email: salvatore.loreto@ericsson.com

Michael Tuexen  
Muenster University of Applied Sciences  
Stegerwaldstrasse 39  
Steinfurt 48565  
DE

Email: tuexen@fh-muenster.de



RTCWEB  
Internet-Draft  
Intended status: Informational  
Expires: August 2, 2013

G. Mandyam  
Qualcomm Innovation Center  
M. Luby  
Qualcomm Technologies Inc.  
T. Stockhammer  
Nomor Research  
C. Foisy  
Qualcomm Technologies Inc.  
January 29, 2013

Forward Error Correction for WebRTC using FEC FRAME  
draft-mandyam-rtcweb-fecframe-00

Abstract

WebRTC provides a solution for peer-to-peer streaming between web applications by leveraging a Real-Time Protocol (RTP) stream between two clients. This RTP stream is expected to be sent over an UDP (Universal Datagram Protocol) connection, which by definition has no built-in reliability. Recently the FEC FRAME Working Group of the IETF has come up with a framework and technical recommendations for applying forward error correction (FEC) to unreliable streams. This framework can be applied to WebRTC with minimal changes to the specification.

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 August 2, 2013.

Copyright Notice

Copyright (c) 2013 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 . . . . .	3
1.1. Block Code Introduction . . . . .	3
2. FEC FRAME Overview . . . . .	4
3. Latency Mitigation . . . . .	6
4. Session Description Protocol Impacts . . . . .	7
5. Discussion . . . . .	10
6. IANA Considerations . . . . .	11
7. Security Considerations . . . . .	11
8. References . . . . .	11
8.1. Normative References . . . . .	11
8.2. Informative References . . . . .	11
Appendix A. Additional Stuff . . . . .	12
Authors' Addresses . . . . .	12

## 1. Introduction

Forward Error Correction (FEC) is a well-known technique for improving the reliability of packet transmission over networks that do not provide guaranteed packet delivery by adding repair packets to the original packets. The IETF has defined a "building block" approach to the specification of FEC to streaming protocols, based on RFC 5052 [RFC5052] and RFC 6363 [RFC6363]. Borrowing from the terminology of RFC 5052 [RFC5052], the FEC can be applied to any payload of a CDP (content delivery protocol). Since WebRTC defines RTP as its target CDP for media streaming, it stands to reason that the IETF building block framework is readily applicable to WebRTC.

### 1.1. Block Code Introduction

Detailed understanding of all the different types of FEC is beyond the scope of this document. Note that FEC in its most fundamental description involves the encoding of a message derived from an alphabet into a representation that is also derived from the same alphabet. Moreover, FEC encoding results in redundancy, i.e. the addition of information to the message that can help in retrieving the original message in the presence of loss of parts of the transmission. If the message is composed of  $k$  individual entries from the alphabet and the FEC encoding results in a new collection of entries (also referred to as a codeword) of length  $n$ , then the FEC code is said to be of rate  $k/n$  or is sometimes said to be a  $(k,n)$  code.

FEC's that operate on individual messages without dependency on other messages in a given sequence are sometimes referred to as block codes. As another way of visualizing this, assume that a message to be sent over a communications channel is derived from a grouping of symbols derived from an alphabet (e.g. a binary alphabet can be represented by the set  $\{0,1\}$ ), and can be represented as a collection of words  $\{m_0, m_1, \dots, m_{(k-1)}\}$ . Then the resultant codeword generated by applying the FEC can be represented as a collection of words derived from the same alphabet  $\{c_0, c_1, \dots, c_{(n-1)}\}$ . If all  $n$  of the words of this codeword are sent over a communications link and at most  $n-k$  of the words of the codeword are lost, then ideally an FEC code can completely recover the original message.

The block code can be represented in terms of a generator matrix  $G$  (of binary entries) acting upon a message vector  $m$ .



$$G = \begin{bmatrix} g_{(0,0)} & g_{(0,1)} & \dots & g_{(0,n-1)} \\ g_{(1,0)} & g_{(1,1)} & \dots & g_{(1,n-1)} \\ \vdots & \vdots & \ddots & \vdots \\ g_{(k-1,0)} & g_{(k-1,1)} & \dots & g_{(k-1,n-1)} \end{bmatrix}, m = [m_0 \ m_1 \ \dots \ m_{(k-1)}]$$

Generator Matrix and Message Vector

Figure 1

A codeword  $c$  is generated by a matrix-vector multiplication of  $G$  with  $m$ . Some of the words of the codeword  $c$ , each word sent in a separate packet together with a word identifier, may be lost and not arrive at the receiver. The receiver can determine which words are received in packets from the word identifiers included in the packets. Based on the word identifiers, the FEC decoder can recover the original message.

If the block code in question is systematic, then a subset of the generated codeword is the original message itself. Such codes allow for a clean separation of the codeword into source symbols and redundancy symbols. In this case, then generator matrix can be represented as

$$G = \begin{bmatrix} g_{(0,0)} & g_{(0,1)} & \dots & g_{(0,n-k-1)} & 1 & 0 & 0 & \dots & 0 \\ g_{(1,0)} & g_{(1,1)} & \dots & g_{(1,n-k-1)} & 0 & 1 & 0 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ g_{(k-1,0)} & g_{(k-1,1)} & \dots & g_{(k-1,n-k-1)} & 0 & 0 & 0 & \dots & 1 \end{bmatrix}$$

Generator Matrix for Systematic Code

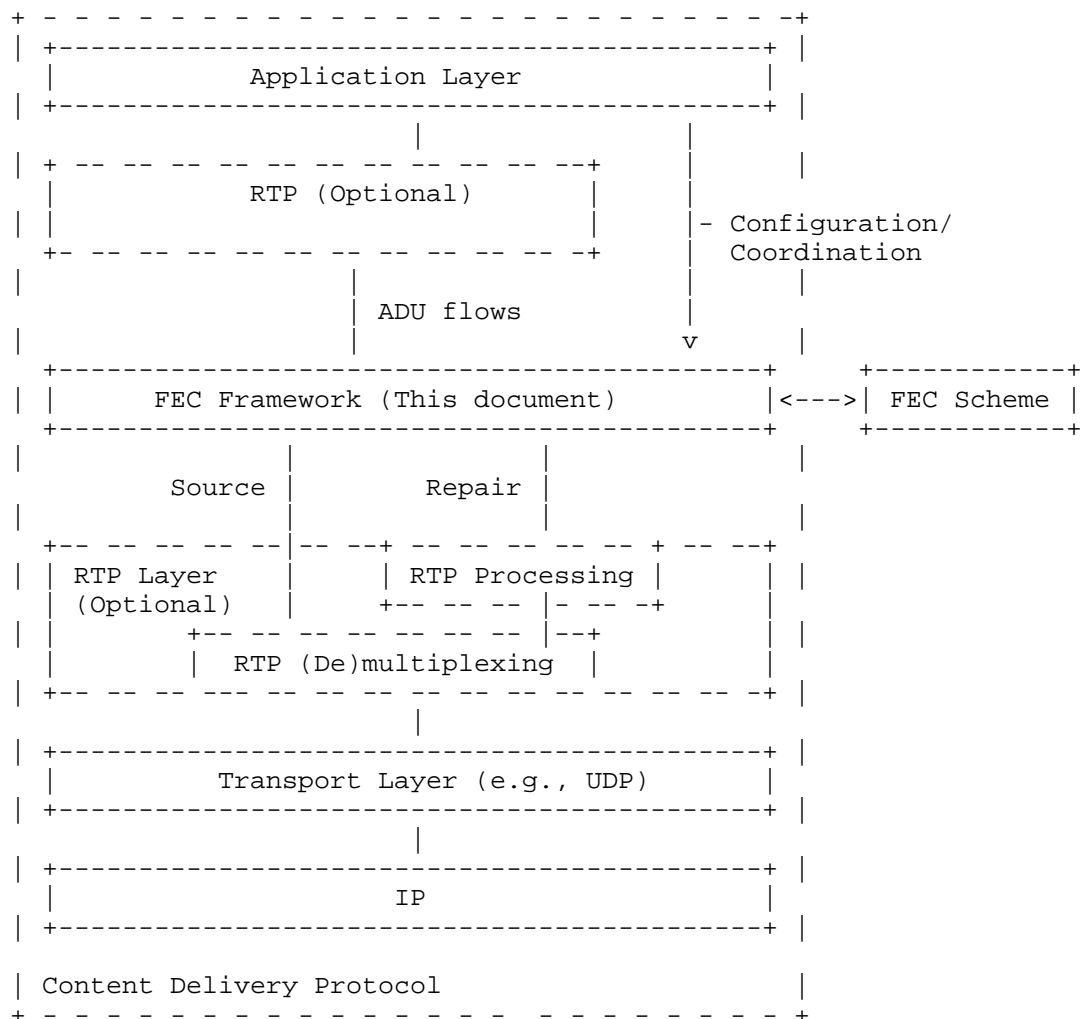
Figure 2

A more detailed overview of block codes, their derivation, and theoretical analysis can be found in textbooks such as [Wicker].

## 2. FEC FRAME Overview

The building block approach of RFC 6363 [RFC6363] allows for a straightforward application of a preferred block code to streaming protocols such as RTP. The chosen block code must satisfy the requirements of RFC 5052 [RFC5052]. A valid FEC encoding scheme will have an IANA-assigned FEC Encoding ID. Since the building block approach to applying block codes to streaming protocols has been standardized by the IETF, it is not necessary to discuss the

specified approach in this document. However, a simple mapping between the Framework Architecture of RFC 6363 [RFC6363] is reproduced here to provide context.



FEC Framework Architecture

Figure 3

With respect to the above figure, the application layer would essentially be a logical collection of the user agent along with the

WebRTC-enabled web application, and the application data unit (ADU) flows could be the RTP packets provided by the user agent implementation of WebRTC. Note that it is also allowed in the specification to multiplex additional RTP-encapsulated redundancy packets onto UDP, which is useful for providing additional resiliency for multicast traffic. After application of block encoding, the encoded blocks can be identified by an FEC Source ID which is appended to the block of data. It is not required if there are other means to indicate to the receiver a unique identifier for the encoded data block.

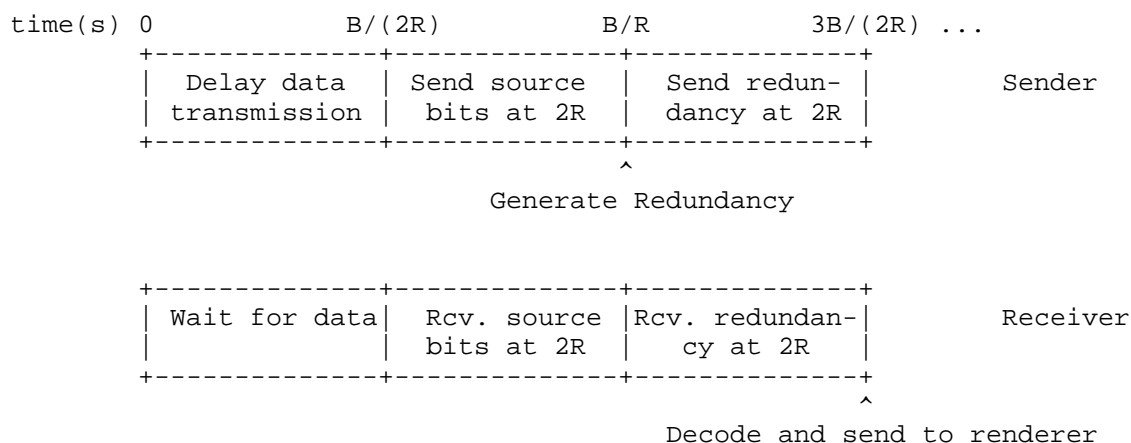
It should be noted that the use of FEC does not come without its own costs. For instance, the selection of the block size (ADU size) has a direct impact on latency as the transmission of the stream must be delayed while the payload is formed for FEC encoding. This is important due to the fact that many block coding schemes tend to be more effective with larger block sizes. Moreover, encoding and decoding latency are applicable (although advancement in processor speed has made this less of an issue). Finally, the amount of redundancy affects the overall throughput. The larger the amount of redundancy, the less likelihood that random losses will result in the receiver being unable to decode data. However, greater redundancy (i.e. a smaller  $k/n$  ratio) has a direct impact on the amount of application data that can be sent over a fixed time interval.

### 3. Latency Mitigation

While FEC encoding provides resiliency in the face of packet loss, it also introduces latency. Assume a system where the source rate for a stream is  $R$  bits/second, and the number of bits to be encoded using a  $k/n$  code is  $B$ . In order to encode  $B$  bits to produce the necessary redundancy, which is in the amount given by  $B(n-k)/k$ , it is necessary for the sender to buffer a block of  $B$  information bits. Therefore, one would assume that the sender would have to delay transmission by at least the time it takes to generate one block of information bits from the source, i.e.  $B/R$ . However, note that in most block codes, the first part of the codeword sent is the actual source information bits. Therefore it is conceivable that transmission from the sender to receiver could commence while the source bits are being buffered at the sender-side encoder.

Assume that the rate of the code is  $k/n = 1/2$ , i.e. the number of redundancy bits is equal to the number of source bits when encoding a block of data. The sender, as one latency mitigation strategy, could delay transmission of the encoded data (source and redundancy bits) by half the duration of a source block,  $B/(2R)$ , and then send at twice the source data rate ( $2R$ ). After the entire block of

information bits  $B$  has been buffered at the sender, then the sender can transmit the remaining code bits at  $2R$ . This results in an overall latency of  $B(n-k)/(2k)$ , or half of the time it takes for the source to generate  $B$  bits of data.



Latency Mitigation Strategy Timeline for Rate 1/2 Block Code

Figure 4

#### 4. Session Description Protocol Impacts

The implications for SDP at the time of the writing of this document are not fully known as there is still debate as to the semantics of SDP for WebRTC. A proposal for SDP usage in WebRTC was described in [I-D.nandakumar-rtcweb-sdp]. In addition, the SDP elements required for FEC are described in RFC 6364 [RFC6364]. Leveraging the example of Section 5.1 in [I-D.nandakumar-rtcweb-sdp], a 2-way video and audio session offer/answer exchange can be depicted for two sample endpoints (Alice and Bob) with the video stream being protected by FEC:

Alice->Bob: Offer(Audio:G.711,AMR-WB Video:H.264 FEC-encoding:Reed-Solomon,LDPC Staircase)

Bob->Alice: Answer(Audio:G.711,AMR-WB Video:H.264 FEC-encoding:Reed-Solomon)

Alice->Bob: Two-way AMR-WB Audio, H.264 Video, Reed-Solomon FEC-Encoding

SDP Contents	Notes
v=0 o=alice 20518 0 IN IP4 0.0.0.0 s=FEC for WebRTC t=0 0 a=ice-ufrag:074c6550 a=ice-pwd:a28a397a4c3f31747dlee3474af08a068 a=fingerprint:sha-1 99:41:49:83:4a:97:0e:1f:ef:6d:f7:c9:c7:7 0:9d:1f:66:79:a8:07 a=group:FEC-FR S1 R1  m=audio 54609 RTP/SAVPF 0 109 98 c= IN IP4 24.23.204.141 a=rtpmap:0 PCMU/8000 a=rtpmap:109 AMR-WB/16000/2 a=fmtp:99 interleaving=30 a=maxptime:100 a=sendrecv a=mid:S0 a=rtcp-mux b=AS:256 b=RS:0 b=RR:0  a=candidate:0 1 UDP 2113667327 192.168.1.4 54609 typ host a=candidate:1 1 UDP 694302207 24.23.204.141 54609 typ srflx raddr 192.168.1.4 rport 54609 a=rtcp-fb:109 nack  m=video 62537 RTP/SAVPF 99 120 c= IN IP4 24.23.204.141 a=rtpmap:99 H264/90000 a=fmtp:99 profile-level-id=4d0028;packetization-mode=1 a=fec-source-flow: id=0 a=mid:S1 a=sendrecv a=rtcp-mux m=application 30000 UDP/FEC c= IN IP4 24.23.204.141 a=fec-repair-flow: encoding-id=2;	Protocol Version Session Origin Session Name Time session is active Session Level ICE param Session Level ICE param  Session Level DTLS Fingerprint for SRTP  FEC group source/repair  Connection Data G.711 8kbps 2-chan AMR-WB 16 kbps  Can send/rcv audio  Can mux RTP/RTCP Bandwidth RTCP Bandwidth RTCP Bandwidth  Host ICE Candidate for audio Server Reflexive ICE Candidate for the above host candidate NACK RTCP feedback  Connection data-source  Source flow for FEC  Can send/rcv video Can mux RTP/RTCP  Connection data-repair Reed-Solomon code

fssi=E:1400,S:0,m:8	support
a=fec-repair-flow: encoding-id=3;	LDPC Staircase support
fssi=seed:1234,E:1400,S:0,nlm3:0	
a=repair-window:200ms	Time duration of source and repair blocks
a=mid:R1	
a=candidate:0 1 UDP 2113667327	Host ICE Candidate
192.168.1.4 62537 typ host	for video
a=candidate:1 1 UDP 1694302207	Server Reflexive ICE
24.23.204.141 62537 typ srflx raddr	Candidate for the
192.168.1.4 rport 62537	above host candidate

SDP Offer (Alice to Bob)

Figure 5

SDP Contents	Notes
v=0	Protocol Version
o=bob 16833 0 IN IP4 0.0.0.0	Session Origin
s=FEC for WebRTC	Session Name
t=0 0	Time session is active
a=ice-frag:c300d85b	Session Level ICE param
a=ice-pwd:de4e99bd291c325921d5d47efbabd9	Session Level ICE param
a2	
a=fingerprint:sha-1	Session Level DTLS
99:41:49:83:4a:97:0e:1f:ef:6d:f7:c9:c7:7	Fingerprint for SRTP
0:9d:1f:66:79:a8:07	
a=group:FEC-FR S1 R1	FEC group source/repair
m=audio 49203 RTP/SAVPF 109	
c= IN IP4 98.248.92.77	Connection Data
a=rtpmap:109 AMR-WB/16000/2	2-chan AMR-WB 16 kbps
a=fmtp:99 interleaving=30	
a=maxptime:100	
a=sendrecv	Can send/rcv audio
a=mid:S0	
a=rtcp-mux	Can mux RTP/RTCP
b=AS:256	Bandwidth
b=RS:0	RTCP Bandwidth
b=RR:0	RTCP Bandwidth

a=candidate:0 1 UDP 2113667327 192.168.1.7 49203 typ host a=candidate:1 1 UDP 694302207 98.248.92.77 49203 typ srflx raddr 192.168.1.7 rport 49203 a=rtcp-fb:109 nack	Host ICE Candidate for audio Server Reflexive ICE Candidate for the above host candidate NACK RTCP feedback
m=video 63130 RTP/SAVPF 99 c= IN IP4 98.248.92.771 a=rtpmap:99 H264/90000 a=fmtp:99 profile-level-id=4d0028;packetization-mode=1 a=fec-source-flow: id=0 a=mid:S1 a=sendrecv a=rtcp-mux m=application 30000 UDP/FEC c= IN IP4 98.248.92.771 a=fec-repair-flow: encoding-id=2; fssi=E:1400,S:0,m:8 a=repair-window:200ms  a=mid:R1	Connection data-source   Source flow for FEC  Can send/rcv video Can mux RTP/RTCP  Connection data-repair Reed-Solomon code support Time duration of source and repair blocks
a=candidate:0 1 UDP 2113667327 192.168.1.7 63130 typ host a=candidate:1 1 UDP 1694302207 98.248.92.77 63130 typ srflx raddr 192.168.1.7 rport 63130	Host ICE Candidate for video Server Reflexive ICE Candidate for the above host candidate

SDP Answer (Bob to Alice)

Figure 6

## 5. Discussion

The use of FEC in WebRTC should not require a significant standards change, as the FEC Framework approved by the IETF already specifies the use of FEC for streaming protocols. There certainly exists tradeoffs between the benefits of FEC at smaller block sizes, and the latency incurred due to larger block sizes. However, these tradeoffs should be considered by WebRTC implementers and not as part of the standardization effort for WebRTC. An item that can be considered is whether a specific FEC scheme should be designated as mandatory-to-

implement, so as to provide a level of interoperability among WebRTC clients.

## 6. IANA Considerations

This memo includes no request to IANA.

## 7. Security Considerations

Security considerations are TBD.

## 8. References

### 8.1. Normative References

- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.
- [RFC5052] Watson, M., Luby, M., and L. Vicisano, "Forward Error Correction (FEC) Building Block", RFC 5052, August 2007.
- [RFC6363] Watson, M., Begen, A., and V. Roca, "Forward Error Correction (FEC) Framework", RFC 6363, October 2011.
- [RFC6364] Begen, A., "Session Description Protocol Elements for the Forward Error Correction (FEC) Framework", RFC 6364, October 2011.

### 8.2. Informative References

- [I-D.nandakumar-rtcweb-sdp]  
Nandakumar, S. and C. Jennings, "SDP for the WebRTC", draft-nandakumar-rtcweb-sdp-00 (work in progress), October 2012.
- [I-D.narten-iana-considerations-rfc2434bis]  
Narten, T. and H. Alvestrand, "Guidelines for Writing an IANA Considerations Section in RFCs", draft-narten-iana-considerations-rfc2434bis-09 (work in progress), March 2008.
- [RFC2629] Rose, M., "Writing I-Ds and RFCs using XML", RFC 2629, June 1999.
- [RFC3552] Rescorla, E. and B. Korver, "Guidelines for Writing RFC



Text on Security Considerations", BCP 72, RFC 3552,  
July 2003.

[Wicker] Wicker, S., "Error Control Systems", Upper Saddle River,  
NJ: Prentice-Hall Inc., 1995.

#### Appendix A. Additional Stuff

This becomes an Appendix.

#### Authors' Addresses

Giridhar Mandyam  
Qualcomm Innovation Center  
5775 Morehouse Drive  
San Diego, California 92121  
USA

Phone: +1 858 651 7200  
Email: mandyam@quicinc.com

Mike Luby  
Qualcomm Technologies Inc.  
2030 Addison Street  
Berkeley, California 94704  
USA

Phone: +1 510 725 3502  
Email: luby@gti.qualcomm.com

Thomas Stockhammer  
Nomor Research  
Brecherspitzstrasse 8  
Munich 81541  
Germany

Phone: +49 8997898002  
Email: stockhammer@nomor.de

Christian Foisy  
Qualcomm Technologies Inc.

Phone: +1 450 510 3202  
Email: cfoisy@qti.qualcomm.com



RTCWeb  
Internet-Draft  
Intended status: Informational  
Expires: August 23, 2013

J. Marcon  
Alcatel-Lucent  
February 19, 2013

RTCWeb data channel management  
draft-marcon-rtcweb-data-channel-management-00

Abstract

The Real-Time Communication in WEB-browsers (RTCWeb) working group is charged to provide protocols to support direct interactive rich communication using audio, video, and data between two peers' web-browsers. For the support of data communication, the RTCWeb working group has in particular defined the concept of bi-directional data channels over SCTP. How to transport application messages on these data channels seems straightforward (i.e. they can be carried as SCTP user messages), however it is yet to be decided how to establish and manage these data channels. This document specifies a method for this, which relies first on a lightweight and scalable out-of-band negotiation of data channel configurations (within the SDP offer/answer exchange) and second on the signaling of the configuration in use in the SCTP user message itself. Once these configurations are negotiated, further creations of data channels can occur purely in-band by simply sending user messages, which avoids to define a new in-band data channel protocol.

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 August 23, 2013.

Copyright Notice

Copyright (c) 2013 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 . . . . .	3
2. Conventions . . . . .	4
3. Terminology . . . . .	4
4. Overview . . . . .	5
5. Data channel configuration and message properties . . . . .	6
6. Procedures . . . . .	7
6.1. Initialization . . . . .	7
6.2. Opening a data channel out-of-band . . . . .	8
6.3. Opening a data channel in-band . . . . .	9
6.4. Closing a data channel . . . . .	10
6.5. Sending and Receiving Data . . . . .	10
6.6. Out-of-band signaling . . . . .	12
7. Security Considerations . . . . .	13
8. IANA Considerations . . . . .	13
9. Acknowledgments . . . . .	13
10. References . . . . .	13
10.1. Normative References . . . . .	13
10.2. Informative References . . . . .	14
Author's Address . . . . .	15

## 1. Introduction

[I-D.ietf-rtcweb-data-channel] provides use cases and requirements for the definition of RTCWeb data channels, and outlines how the Stream Control Transmission Protocol (SCTP) [RFC4960] encapsulated within Datagram Transport Layer Security (DTLS) [RFC6347] can be used for this purpose. While some of these requirements easily map to existing capabilities of the SCTP protocol and extensions (e.g. application messages can be carried as SCTP user messages), SCTP and existing SCTP extensions do not natively support the following requirements:

- o data channel bidirectionality (this can be achieved by pairing one SCTP outbound stream and one SCTP inbound stream)
- o data channel priority
- o partial reliability of delivery, based on a maximum number of retransmissions
- o general data channel setup

For setting up the SCTP association, the in-band SCTP association initialization is assisted out-of-band by JSEP [I-D.ietf-rtcweb-jsep] and the SDP Offer/Answer model [RFC3264]. For setting up each data channel, several approaches can be considered:

1. a purely in-band data channel setup - such a protocol does not exist today.
2. a hybrid in-band / out-of-band data channel setup, where the in-band signaling relies on a new protocol defined on top of SCTP user messages. The proposal [I-D.jesup-rtcweb-data-protocol] follows this approach.
3. an out-of-band negotiation of data channel configurations, minimally assisted by some lightweight in-band signaling allowing further in-band creations of data channels.

This document describes the latter approach, preferred by the author for the following reasons:

- o Minimal need for SDP renegotiation: the initial offer/answer for establishing the SCTP association is often enough.
- o Scalability of the SDP signaling: typically it is as light as a couple of attribute lines regardless of the number of data channels created in the session.

- o Potential interoperability with other systems, due to the use of out-of-band signaling.
- o Ability to create data channels purely in-band, once the data channel configurations are negotiated
- o No need for a new in-band data channel control protocol.
- o Speed: No control message overhead for the in-band creation of data channels: sending user messages automatically creates new data channels.
- o Simplicity of implementation.

As a result, the proposal can easily cope with strenuous data transmission scenarios, like:

- o The transfer of a high number of files, eventually in parallel.
- o The fast opening and closing of one data channel.

## 2. Conventions

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

## 3. Terminology

This document uses the following terms:

Agent: As defined in [RFC3264], an agent is the protocol implementation involved in the offer/answer exchange. There are two agents involved in an offer/answer exchange.

Configuration: a fixed set of data channel parameters, constraining the configuration of some or all the data channels created on top of a given SCTP association.

Data channel: A bidirectional channel consisting of paired SCTP outbound and inbound streams.

In-band: transmission through the peer-to-peer SCTP association.

Out-of-band: transmission through the RTCWeb signaling path, using JSEP [I-D.ietf-rtcweb-jsep] and the SDP Offer/Answer model [RFC3264].

Peer: From the perspective of one of the agents in a session, its peer is the other agent. Specifically, from the perspective of the SDP offerer, the peer is the SDP answerer. From the perspective of the SDP answerer, the peer is the SDP offerer.

Stream: A unidirectional stream of an SCTP association. It is uniquely identified by a stream identifier.

#### 4. Overview

This section provides an overview of the approach detailed in this document.

A data channel configuration is an identified fixed set of data channel parameters potentially applicable to some or all of the data channels created during the session. These parameters include: order of delivery, reliability of delivery, subprotocol.

Configurations are uniquely identified throughout the session, and negotiated out-of-band between the endpoints. The configuration concept is transparent to the application, which sets up and handles each data channel individually. Whenever the application creates a new data channel, the browser internally checks if the passed set of parameters strictly matches an existing configuration, and if not generates a new configuration identifier for this set. In the latter case only does the browser trigger the application for an SDP renegotiation.

In the SDP offer, the offerer associates to the m=application SDP line that defines the SCTP association one attribute line per data channel configuration.

For each data channel configuration in the offer that is accepted by the answerer, the answerer echoes in the answer the configurations supported and accepted. Once the offerer receives the answer and (in case of an initial offer) the SCTP initialization is complete, each data channel locally created using one of the accepted configurations is signaled to the application as open for transmission.

Each created data channel is bound to to one negotiated configuration.

By convention, the inbound and outbound streams of a data channel



have the same SCTP stream number. This stream number is selected by the first endpoint sending a user message on this channel. Till this happens, an open data channel has no assigned stream number.

Data channel messages are sent as SCTP user messages, preceded in the DATA chunk User Data field by two bytes specifying data channel configuration identifier as well as the message data framing type (textual or binary).

A user message received on a stream number not assigned to any data channel automatically opens a data channel, set up according to the configuration signaled in the user message.

Closing a data channel is done in-band by resetting the Stream Sequence Number (SSN) of the related SCTP inbound and outbound streams, as defined in [RFC6525].

This proposal requires the registration of one SCTP Payload Protocol Identifier.

## 5. Data channel configuration and message properties

For the proposal in this document, a data channel configuration is characterized by the following properties:

- o configuration identifier, a 12-bit integer unique across all the data channel configurations managed during the lifetime of an SCTP association.
- o ownership: the configuration is owned by the endpoint which originated the very first offer that included this configuration, for a given SCTP association.
- o reliability of delivery: full reliability (as per [RFC4960]) or partial reliability with max transmission lifetime (as per [RFC3758]) or partial reliability with max number of retransmissions.
- o order of delivery: ordered or unordered
- o subprotocol identifier
- o subprotocol setup data, if applicable

For the proposal in this document, a data channel is characterized from an endpoint viewpoint by the following properties:

- o Configuration identifier. It can bind multiple data channels at a time.
- o Label: local human-readable description of the data channel.
- o Data channel priority
- o SCTP stream number: common to the SCTP outbound stream and inbound stream composing the data channel.
- o state, which can have the following values:
  - \* Connecting: data channel opened locally, and awaiting opening acknowledgment by the peer.
  - \* Open: data channel available for bidirectional data transmission.
  - \* Closing: data channel closed locally, and awaiting closing acknowledgment by the peer.
  - \* Closed: data channel closed by the agent, and acknowledged as closed by the peer.

A message sent over a data channel inherits from the transmission properties configured to this data channel: reliability and order of delivery. In addition, the message is characterized by the following message-specific property:

- o transport format encoding: text or binary.

Note that for API simplification purpose, reliability, order of delivery and payload protocol identifier are not configurable per user message, but per data channel only.

The payload protocol identifier (PPID) field present in SCTP DATA chunks is used to signal the data framing described in this document. This value is to be obtained from the SCTP Payload Protocol Identifiers registry managed by IANA.

## 6. Procedures

### 6.1. Initialization

The PeerConnection and underlying SCTP association are initialized with N data channels, all in Closed state, and with respective outbound and inbound stream numbers ranging from 0 to N-1. The

number N can be specified by the application or else defaults to 16.

## 6.2. Opening a data channel out-of-band

An application creating a data channel providing some data channel parameters to the agent's browser. If the subset of these parameters composing a data channel configuration does not strictly match an existing configuration, the browser assigns a new configuration identifier to this subset, and binds it to the data channel. The configuration identifier is generated incrementally, starting from zero for each SCTP association. Identifiers of configurations rejected by the answerer must never be used again.

In addition, if the application requests the creation:

- o at a time where the endpoint is in a stable state with an SCTP association already set up, and if the match of configuration is successful, the browser then sets the data channel state to Open.
- o Otherwise the browser sets the data channel state to Connecting. Moreover, unless the endpoint is in an init state and createOffer has not yet been called, the browser notifies to the application the need for an SDP renegotiation.

The created data channel has no assigned SCTP stream number yet. At this stage though the user agent can anticipate a shortage of available SCTP streams and send in-band the request to increase the number of SCTP streams.

The new offer (if any) contains one 'm-line' for the SCTP association, and one attribute line per data channel configuration. This list of configurations must include the new configurations as well as all the configurations successfully negotiated in previous offer/answer exchanges for this SCTP association.

The peer's browser receiving the offer does the following:

1. for the data channels that are in Open state but which are bound to a configuration no longer present in the offer, change their state to Closed
2. for each newly offered configuration, the peer's browser then informs the application of a new offered data channel along with the configuration specifics. The application can accept this data channel (intended: configuration) by creating a new data channel using the configuration parameters. Not doing so will mean to reject the configuration in the answer.

Note: for each new configuration, the offerer expectedly creates one data channel or more, whereas the answerer creates one data channel only. How the final data channel pairing (and SCTP stream number assignment) is resolved is further explained in this document.

Note: the answerer can only reject new configurations, configurations previously negotiated cannot be removed from the configuration list associated with the SCTP association.

The agent's browser receiving the answer does the following:

1. for the data channels not in Closed state and bound to a configuration no longer listed in the answer, change their state to Closed.
2. for each newly offered data channel configuration accepted by the answerer, change the state of any data channel bound to this configuration from Connecting to Open.

### 6.3. Opening a data channel in-band

Each user message sent in a data channel includes the identifier of the configuration which this data channel is bound to. This signaling allows to enable or speed up the opening of new data channels in-band:

- o Case A: In a stable state, the local creation of a data channel with parameters mapping to a negotiated configuration creates the data channel in Open state immediately, and does not signal this to the peer. Some time later, when the application sends its first message on this data channel:
  1. the agent's browser selects for the lifetime of the data channel an SCTP outbound stream number not used by any channel.
  2. it then sends the user message over this SCTP stream.
  3. the peer's browser SACKnowledges the user message, using an SCTP stream of same stream number (expectedly unused). It then notifies the peer's application of the data channel opening.
- o Case B: Once the answerer has accepted a new offerer's configuration, and has subsequently opened a data channel bound to this configuration, the answerer's application may choose to send user messages on this channel immediately. The offerer receiving

this message should:

1. route this message to one of the Connecting-state data channels bound to the same configuration.
2. change its state to Open, and sets its SCTP outbound stream number (expectedly unused) to the SCTP inbound stream number of the message.
3. delivers the message to the application.

#### 6.4. Closing a data channel

When the application requests to close a data channel, the agent's browser initiates an in-band Stream Sequence Number (SSN) reset of the related SCTP inbound and outbound streams. These streams are then available for further reuse.

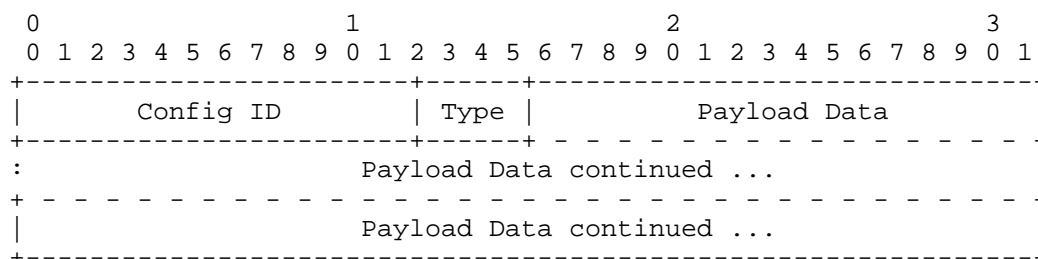
#### 6.5. Sending and Receiving Data

To expose to upper layers an API similar to the Web Socket API [WSAPI], the agent's browser needs to specify to the peer's browser the framing type of each data channel message, amongst binary or text (UTF-8).

In addition, each user message needs to carry the identifier of the configuration which the data channel is bound to.

For the sending of a user message over an opened data channel, the agent's browser:

1. converts the message data from UTF-16 to UTF-8 if provided by the application as a DOMString.
2. sets in the DATA chunk the Unordered bit, Payload Protocol Identifier and Stream Number as per data channel configuration.
3. constructs the User Data field as the framing type byte(s) followed by the (converted) message data



RTCWeb data framing

Config ID : 12 bits

Identifies a data channel configuration negotiated out of band between the two endpoints.

Note: to be considered if the Config ID should be included in all user messages.

Type : 4 bits

Defines the data framing type of "Payload data". If an unknown type is received, the receiving endpoint must reject the user message. The following values are defined:

- \* x0 denotes a binary frame
- \* x1 denotes a UTF-8 encoded text frame
- \* x2-xFFF are reserved for further use

An agent's browser receiving a user message on a data channel behaves as follows:

- o If the configuration identifier in the message does not map to any negotiated configuration, or if the data channel (identified by the message stream number) is in Closing or Connecting state, reject (or discard?) the message.
- o Otherwise (i.e. supported configuration identifier and Open state):

If the endpoint has just sent the very first user message on this data channel and has not yet received the SACK, it means that both endpoints attempt to dynamically create a data channel by the sending of a user message. In this case, if the endpoint has ownership of the signaled configuration, the

browser must discard (reject?) the message.

Note: another way of avoiding this conflict is to state by convention that the endpoint which initiated the offer for the SCTP association establishment owns all the even stream numbers, while the other endpoint owns all the odd stream numbers.

Otherwise deliver the message.

- o Otherwise (i.e. supported configuration identifier and Closed state):

If a Connecting-state data channel exists with no assigned stream number, open it with a stream number set to the message stream number.

If not, open a new data channel with a stream number set to the message stream number

Finally, deliver the message.

## 6.6. Out-of-band signaling

To signal the data channel configurations intended for use during the lifetime of an SCTP association, the agent completes the SDP <fmt> section of the m=application SDP line defined for the SCTP association. For each data channel configuration previously negotiated or newly added at the time of offer generation, the agent's browser:

- o must specify: configuration identifier.
- o may specify: order of delivery, reliability of delivery, subprotocol, subprotocol configuration data.

As an example in the offer (line split for readability):

```
m=application 54111 DTLS/SCTP 5000
a=sctpmap:5000 webrtc-DataChannel 16
a=sctp-protocol:5000 config=1;label="channel 1";subprotocol="chat";
a=sctp-protocol:5000 config=2;label="channel 2";
subprotocol="file transfer";max_retr=3
```

data channel configuration setup in SDP offer

An in the returned answer (line split for readability):

```
m=application 54111 DTLS/SCTP 5000
a=sctpmap:5000 webrtc-DataChannel 16
a=sctp-protocol:5000 config=2;label="channel 2";
subprotocol="file transfer";max_retr=3
```

data channel configuration setup in SDP answer

In reply to this offer, the peer constructs in the answer the data channel configuration list of the SCTP association as follows:

- o drop any unwanted or unsupported data channel configuration
- o echo the other configurations, and preserve the following properties at least: configuration identifier, subprotocol (if any). Specifies also the peer-specific properties (subprotocol setup data).

This may need to be specified via MMUSIC.

## 7. Security Considerations

To be completed.

## 8. IANA Considerations

To be completed.

## 9. Acknowledgments

The authors wish to thank Richard Ejzak, ... for their invaluable comments.

## 10. References

### 10.1. Normative References

- [I-D.ietf-rtcweb-jsep] Uberti, J. and C. Jennings, "Javascript Session Establishment Protocol", draft-ietf-rtcweb-jsep-02 (work in progress), October 2012.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.



- [RFC3758] Stewart, R., Ramalho, M., Xie, Q., Tuexen, M., and P. Conrad, "Stream Control Transmission Protocol (SCTP) Partial Reliability Extension", RFC 3758, May 2004.
- [RFC4960] Stewart, R., "Stream Control Transmission Protocol", RFC 4960, September 2007.
- [RFC6347] Rescorla, E. and N. Modadugu, "Datagram Transport Layer Security Version 1.2", RFC 6347, January 2012.
- [RFC6525] Stewart, R., Tuexen, M., and P. Lei, "Stream Control Transmission Protocol (SCTP) Stream Reconfiguration", RFC 6525, February 2012.

## 10.2. Informative References

- [I-D.ietf-rtcweb-data-channel]  
Jesup, R., Loreto, S., and M. Tuexen, "RTCWeb Datagram Connection", draft-ietf-rtcweb-data-channel-02 (work in progress), October 2012.
- [I-D.jesup-rtcweb-data-protocol]  
Jesup, R., Loreto, S., and M. Tuexen, "WebRTC Data Channel Protocol", draft-jesup-rtcweb-data-protocol-03 (work in progress), September 2012.
- [ITU.T140.1998]  
"Protocol for Multimedia Application Text Conversation", ITU-T Recommendation T.140, February 1998.
- [RFC3264] Rosenberg, J. and H. Schulzrinne, "An Offer/Answer Model with Session Description Protocol (SDP)", RFC 3264, June 2002.
- [WebRtcAPI]  
Bergkvist, A., Burnett, D., Narayanan, A., and C. Jennings, "WebRTC 1.0: Real-time Communication Between Browsers", World Wide Web Consortium WD WD-webrtc-20120821, August 2012,  
<<http://www.w3.org/TR/2012/WD-webrtc-20120821>>.
- [WebSocketAPI]  
Hickson, I., "The WebSocket API", World Wide Web Consortium LastCall WD-websockets-20120809, August 2012,  
<<http://www.w3.org/TR/2012/WD-websockets-20120809>>.

Author's Address

Jerome Marcon  
Alcatel-Lucent  
Route de Villejust  
Nozay 91620  
France

Email: [jerome.marcon@alcatel-lucent.com](mailto:jerome.marcon@alcatel-lucent.com)



Network Working Group  
Internet-Draft  
Intended status: Informational  
Expires: April 18, 2013

X. Marjou  
P. Philippe  
France Telecom Orange  
October 15, 2012

Video codec for WebRTC.  
draft-marjou-rtcweb-video-codec-00

Abstract

In the context of WebRTC, there is currently no consensus on the video codec(s) that need to be mandatory to implement. This draft gives some arguments in favor of H.264.

Status of this Memo

This Internet-Draft is submitted to IETF 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 18, 2013.

Copyright Notice

Copyright (c) 2012 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 . . . . .	3
2. Terminology . . . . .	3
3. Rationale and Position . . . . .	3
4. Security Considerations . . . . .	4
5. IANA Considerations . . . . .	4
6. Acknowledgements . . . . .	4
7. References . . . . .	4
7.1. Normative references . . . . .	4
7.2. Informative references . . . . .	4
Authors' Addresses . . . . .	4

## 1. Introduction

In the context of WebRTC, there is currently no consensus on the video codec(s) that need to be mandatory to implement.

In order to reach a consensus, the RTCWEB chairs have solicited internet-drafts naming proposed mandatory-to-implement video codecs (c.f. [rtcweb-mail]).

This draft gives some arguments in favor of H.264.

## 2. Terminology

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

## 3. Rationale and Position

Many videoconferencing systems exist today (e.g. fact sheets of services at [h264-ftob]), mainly for professional services but also for individual consumers.

We believe that WebRTC, when used as a mean to interconnect Web browsers to these existing services, can be a driver for enabling more users to access them.

As an example, all Orange video conferencing systems operate using the H.264/AVC technology. H.264/AVC benefits from many available implementations, tuned for different architectures, and has clear licensing conditions. VP8 has no footprint in this market, independent implementations are rare, licensing conditions are not yet clarified (free license offered from one patent owner while MPEG LA operates a Patent Pool with at least 12 members (c.f. [press-article])).

With this current status, it is believed that incorporating the mandatory to implement video codec having the bigger footprint will permit a better adoption and interconnection of WebRTC to existing services leading to a successful standard.

Hence we strongly support H.264/AVC to be part of the mandatory to implement codecs.

#### 4. Security Considerations

None.

#### 5. IANA Considerations

None.

#### 6. Acknowledgements

#### 7. References

##### 7.1. Normative references

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

##### 7.2. Informative references

[h264-ftob]  
Orange, "<http://www.orange-business.com/en/mnc2/collaboration/conferencing/index.jsp>".

[press-article]  
streamingmedia.com, "<http://www.streamingmedia.com/Articles/Editorial/Featured-Articles/WebM-Patent-Fight-Ahead-for-Google-76781.aspx>".

[rtcweb-mail]  
IETF, "<http://www.ietf.org/mail-archive/web/rtcweb/current/msg05070.html>".

#### Authors' Addresses

Xavier Marjou  
France Telecom Orange  
2, avenue Pierre Marzin  
Lannion 22307  
France

Email: [xavier.marjou@orange.com](mailto:xavier.marjou@orange.com)

Pierrick Philippe  
France Telecom Orange  
2, avenue Pierre Marzin  
Lannion 22307  
France

Email: [pierrick.philippe@orange.com](mailto:pierrick.philippe@orange.com)





RTCWEB  
Internet-Draft  
Intended status: Standards Track  
Expires: August 22, 2013

M. Thomson  
Microsoft  
February 18, 2013

Data Channels for RTCWEB  
draft-thomson-rtcweb-data-00

Abstract

RTCWEB have selected SCTP over DTLS over UDP with ICE for peer-to-peer data channels. There is some debate over the best way to negotiate channels. This proposal is a nose-to-tail description of an alternative to existing proposals.

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 August 22, 2013.

Copyright Notice

Copyright (c) 2013 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 . . . . .	3
1.1. Terminology . . . . .	3
2. Overview of Operation . . . . .	3
3. (In)consistent Properties . . . . .	5
3.1. Negotiation . . . . .	5
3.2. Dealing with Mismatched Properties . . . . .	5
4. Available Data Channel Properties . . . . .	6
5. SDP Format . . . . .	7
6. Payload Protocol Identifiers . . . . .	8
7. IANA Considerations . . . . .	8
8. Security Considerations . . . . .	8
9. Acknowledgments . . . . .	9
10. References . . . . .	9
10.1. Normative References . . . . .	9
10.2. Informative References . . . . .	9
Author's Address . . . . .	9

## 1. Introduction

RTCWEB [I-D.ietf-rtcweb-overview] has defined the use of the Stream Control Transmission Protocol (SCTP) [RFC4960] over Datagram Transport Layer Security (DTLS) [RFC6347] over UDP with Interactive Connectivity Establishment (ICE) [RFC5245] for peer-to-peer data channels.

This document describes a proposal for how this protocol stack is used. The proposal attempts to reconcile the following basic requirements:

- o the ability to have data channels used interchangeably with websockets, after establishment
- o the ability to use as many SCTP features as possible

Like other proposals, this proposal uses an API that is largely interchangeable with the WebSockets API [REF:TBD]. Of course, that alone is insufficient because the way that data channels are created is completely different to websockets [RFC6455]. Only the general usage of the API follows the WebSockets API, channel establishment requires a very different process.

Furthermore, not every application will care for compatibility with the WebSockets API. For those applications, additional properties are exposed to enable valuable SCTP features.

In these aspects, all data channel proposals are the same. The details differ. For example, this one doesn't need an in-band protocol. It even avoids the need for negotiation, except where it is needed. If not for the fact that the WebSockets API designers - in their infinite wisdom - decided to distinguish text from binary, it wouldn't even need to use a PPID to identify textual messages.

### 1.1. Terminology

In this document, the key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" are to be interpreted as described in BCP 14, RFC 2119 [RFC2119] and indicate requirement levels for compliant implementations.

## 2. Overview of Operation

A data channel is a bidirectional communication medium between WebRTC peers.

Every data channel is bound to a specific SCTP stream number. The same SCTP stream identifier is used for both directions of the data channel. Though SCTP streams are unidirectional, and this concept doesn't hold any particular meaning for SCTP, this simplification ensures that channels can be created with minimal overhead.

Each data channel has a set of properties that governs how it sends messages. Unlike other SCTP APIs where properties like reliability settings are set on a per-message basis, this API places these properties on the data channel. This allows the API to behave exactly like the WebSockets API when sending messages. Details of the available data channel properties are included in Section 4.

There are three ways that a data channel can be created. All three result in an object representing the data channel being provided to the application. Each differs in the manner of delivery and how properties are selected for the data channel.

1. The application can request the creation of a new data channel directly. The browser selects appropriate properties for the channel, using any values provided by the application and providing defaults for others.

This triggers a notification to the application that indicates that it needs to renegotiate the session.

2. Offer/answer negotiation can trigger the creation of a new data channel. In this case, the session description provided in an offer or answer describes the properties of the channel.
3. Messages can arrive on an SCTP stream that does not have a data channel allocated. If messages arrive on a stream, the browser provides default values for all stream properties.

Channel creation can fail if there are an insufficient number of available SCTP streams. This is based on either a local unwillingness to receive more streams, or based on knowledge of the unwillingness of the peer to receive more streams.

Creation can also fail if the application specifies a stream ID that is already in use. These should trigger the appropriate error mechanisms (exceptions or something).

Channels are closed by sending a RE-CONFIG chunk that includes Incoming and Outgoing SSN Reset Request parameters, as defined in [RFC6525]. Closing a channel doesn't permit the sending of a code and message as exposed in the WebSockets API, any values that are provided by the application are discarded.

### 3. (In)consistent Properties

The creation of the first data channel will require offer/answer negotiation. This is necessary to ensure that the SCTP association is created, including ICE, the DTLS handshake, plus any authentication that might be required.

Once an SCTP association is live, data channels can be used to exchange messages immediately after they are created. The drawback is that messages arrive at the peer without any information about what properties the sender attaches to the corresponding data channel.

How much property consistency matters to the application will depend on the application. If the application is performing SS7 signaling using M3UA [RFC4666], this is unlikely to matter, but some applications could rely on having consistent data channel properties.

#### 3.1. Negotiation

The safest (and slowest) way to establish new channels with consistent properties is to negotiate them. This is performed using an offer/answer exchange. The application is able to choose where and when this negotiation occurs. If there is an existing data channel, then this provides a low latency path for performing this negotiation.

The negotiation includes a description for every SCTP stream that a peer is sending that includes all of the data channel properties (see Section 4), so that the receiver can create a data channel with the same properties. The browser creates a data channel with the described properties and provides that to the application. If the data channel description appears in an offer, the answer describes the data channel that is used on the same stream number.

An offer or answer that includes a description for a data channel that already exists, then the properties of that data channel are not modified. An answer **MUST** include the properties of the existent data channel, not the channel that the offer describes.

#### 3.2. Dealing with Mismatched Properties

An application that chooses to send on data channels prior to negotiation will cause the receiving peer to create a data channel with a default configuration. Applications can handle this in a number of ways:

- o The application on the receiving peer can create data channels in the same order as the sender to ensure consistent properties. This is possible because stream identifiers are assigned in the same way by both peers.
- o The application on the receiving peer might apply application-specific default values for all non-negotiated channels.
- o The application on the receiving peer might not care about having consistent data channel properties. Note that data channel properties only apply to the sending of messages.

Note: It is possible to provide an application with information about the values that are in use by a peer. This would in most cases be possible after negotiation, though some properties are revealed when new messages arrive. Of course, this is a lot of effort after the application has already effectively declared that it doesn't care. Given that the application could exchange this information using the data channel(s) it has convenient, adding new APIs seem of very low value.

[[Irrelevant API Note: Adding a data channel triggers a notification to the application that it should renegotiate the session. Normally, the 'negotiation needed' state is cleared when the negotiation commences (or completes?). If the application decides to send packets, then the damage is done and there is no point negotiating. That being the case, a data channel could be removed from the set of unnegotiated things upon sending a packet. Negotiation from this point isn't going to change anything.]]

#### 4. Available Data Channel Properties

The following properties are exposed to the application. All of these properties can be set during the creation of a data channel. Once the channel object is created, these properties are mutable, with the exception of "streamId".

**streamId** The SCTP stream ID to use for the channel. If not provided by the application, the lowest valued stream ID that is not already in use by a data channel is selected. If the channel is created as a result of negotiation or packet arrival, the stream ID has already been chosen.

**binaryPPID** The SCTP payload protocol identifier (PPID) that is used for binary messages. Textual messages are always sent using a PPID that indicates textual content, so this value only determines the PPID that is attached to binary messages. This field is a 32-

bit number.

Unless otherwise specified, channels use the PPID for WebRTC binary data channels. Channels created in response to the receipt of a message use the PPID from the received message, unless the message uses the PPID for WebRTC textual data channels, which causes the binary PPID to be selected instead. Details on the newly defined PPIDs are included in Section 6.

**reliabilityTime** The amount of time (in milliseconds) that the browser will attempt to retransmit messages for reliable delivery. Together with **reliabilityRetransmissions**, this enables unreliable or partially reliable transmission of data. The default value for this property is the largest number available (e.g., `Number.POSITIVE_INFINITY`).

**reliabilityRetransmissions** The number of retransmissions that the browser will make for any packet reliable delivery. Together with **reliabilityTime**, this enables unreliable or partially reliable transmission of data. The default value for this property is the largest number available (e.g., `Number.POSITIVE_INFINITY`).

**label** The label to assign to the data channel. A default value is selected by the receiving browser.

**protocol** A protocol label that identifies the protocol used on the data channel. This property is undefined unless set by the application.

**binaryType** The **BinaryType** defined in The WebSockets API determines whether binary data is provided to the application as **Blob** or **ArrayBuffer** objects. The default value is "blob". Note that changing this might not have an immediate effect if messages have started to arrive prior to the change.

All these properties are specific to each message that is sent. Changes to mutable properties take effect for the next message that is sent on the channel.

This isn't a W3C WebRTC document, so specifics of the API aren't really relevant, but it is imagined that these properties could be passed in a dictionary to the method that creates a new data channel and exposed as attributes on the data channel object.

## 5. SDP Format

The properties described above appear in SDP. All properties are



declarative. Though the specifics of the syntax doesn't matter much, the following example could indicate something that would work:

```
m=application 12345 SCTP/DTLS 0 1 2 5
c=IN IP6 ::1
a=fmtp:0 binaryPPID=177;label=control
a=fmtp:1 label=chat
a=fmtp:2 label=characters;reliabilityTime=2000;protocol=lrudfb
a=fmtp:5 label=bullets;reliabilityTime=5000
```

[[Formal SDP grammar TBD]]

This assumes that the SCTP port number (inside the DTLS encapsulation) is fixed, so that this doesn't need to be indicated anywhere. I'm not sure whether asking IANA for a port allocation makes sense though.

## 6. Payload Protocol Identifiers

Two SCTP payload protocol identifiers are defined for WebRTC data channels.

The WebRTC textual data channel PPID (number TBD) is used for all messages that are identified as being textual. The payload of messages marked with this PPID MUST be UTF-8 encoded text.

The WebRTC binary data channel PPID (number TBD) is used as the default PPID for new data channels.

## 7. IANA Considerations

This document probably should register the PPIDs, but I don't really have time to do that right now.

## 8. Security Considerations

I thought about this, and I can't think of any specific security considerations. I could blather on about willingness to accept streams and large volumes of data, but that's pretty lame.

I'm sure something will turn up eventually.

## 9. Acknowledgments

This document is a rush job. If it survives for any longer than a couple of weeks, no doubt this section will come in handy.

## 10. References

### 10.1. Normative References

- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.
- [RFC4566] Handley, M., Jacobson, V., and C. Perkins, "SDP: Session Description Protocol", RFC 4566, July 2006.
- [RFC4960] Stewart, R., "Stream Control Transmission Protocol", RFC 4960, September 2007.
- [RFC6525] Stewart, R., Tuexen, M., and P. Lei, "Stream Control Transmission Protocol (SCTP) Stream Reconfiguration", RFC 6525, February 2012.

### 10.2. Informative References

- [I-D.ietf-rtcweb-overview] Alvestrand, H., "Overview: Real Time Protocols for Brower-based Applications", draft-ietf-rtcweb-overview-05 (work in progress), December 2012.
- [RFC4666] Morneault, K. and J. Pastor-Balbas, "Signaling System 7 (SS7) Message Transfer Part 3 (MTP3) - User Adaptation Layer (M3UA)", RFC 4666, September 2006.
- [RFC5245] Rosenberg, J., "Interactive Connectivity Establishment (ICE): A Protocol for Network Address Translator (NAT) Traversal for Offer/Answer Protocols", RFC 5245, April 2010.
- [RFC6347] Rescorla, E. and N. Modadugu, "Datagram Transport Layer Security Version 1.2", RFC 6347, January 2012.
- [RFC6455] Fette, I. and A. Melnikov, "The WebSocket Protocol", RFC 6455, December 2011.

Author's Address

Martin Thomson  
Microsoft  
3210 Porter Drive  
Palo Alto, CA 94304  
US

Phone: +1 650-353-1925  
Email: martin.thomson@skype.net

