

Network Working Group
Internet-Draft
Intended status: Informational
Expires: April 9, 2014

H. Alvestrand
A. Grange
Google
October 6, 2013

VP8 as RTCWEB Mandatory to Implement
draft-alvestrand-rtcweb-vp8-02

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 April 9, 2014.

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. Specification status	3
3.1. VP8 standardization status	3
4. Deployment status	4
5. Image quality evaluations	4
5.1. Objective evaluations	4
5.2. Subjective evaluations	5
6. Performance evaluation	5
6.1. Software	5
6.2. Hardware support	6
6.3. Hardware performance	6
7. IPR status	7
8. IANA Considerations	8
9. Security Considerations	9
10. Acknowledgements	9
11. References	9
11.1. Normative References	9
11.2. Informative References	9
Authors' Addresses	10

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. Specification status

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.

In the time since the original RFC publication, and indeed since the first publication of the VP8 bitstream format, there have been no changes to the decoder that broke bitstream compatibility.

3.1. VP8 standardization status

The VP8 codec has been proposed as a basis for standardization in MPEG, in response to its Call for Proposals for a royalty-free video codec. At its meeting in Vienna, Austria in July 2013, following the

presentation of subjective and objective quality evaluation results, and a focused discussion of possible IPR issues, MPEG passed a resolution calling for the creation of a new project (Video Coding for Browsers, or VCB), with the aim of producing a final DIS document (FDIS) by July 2014. (MPEG output document w13648).

At the meeting of the US National Body of MPEG in October 013, the USNB passed a resolution supporting this work, and expressing a preference for "options that maintain a native VP8 mode" - that is, no incompatible changes.

4. Deployment status

The VP8 codec has been extensively deployed in production services:

- o Skype (now part of Microsoft) used the codec extensively in its video conferencing software.
- o Google Hangouts is now fully converted to using VP8 on the various PC platforms. This platform now offers free videoconferencing in HD quality to everyone.
- o Google Remote Desktop uses VP8.
- o Google Chromecast uses VP8, showing what can be achieved with hardware decoding support.
- o Both the Firefox and Chrome WebRTC implementations use VP8 exclusively.

5. Image quality evaluations

5.1. Objective 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.

The software and the clips are available via the WEBM project's GIT repository:

http://git.chromium.org/gitweb/?p=webm/vpx_codec_comparison.git

Note: Tests run by Ericsson have demonstrated that it is possible to reduce the VP8 performance to be very close to that of baseline by running in "fixed QP" mode - selecting a single QP value in order to achieve a given bitrate. We believe this VP8 mode is an unrealistic mode for production use, and not what we should be evaluating.

5.2. Subjective evaluations

As part of the process of submitting VP8 for evaluation in ISO/IEC JTC1 SC29 WG11 (MPEG), the VP8 codec has been subjected to subjective and objective quality evaluations; the input reports are in WG11 documents N13775 (Vienna, MPEG 105 meeting, subjective numbers for VP8 performed by Vittorio Baroncini), M29364 (Incheon, subjective comparison between VP8 and IVC) and M28182 (Geneva, MPEG 103 meeting), respectively.

These tests were performed at the laboratories of Vittorio Baroncini, who is also a chair of the Testing subgroup of MPEG, and has performed many of the subjective tests done as part of the HEVC effort.

Together with the tests presented in document M29364, we also asked Vittorio Baroncini to do a subjective evaluation of VP8 compared to the AVC Baseline; the results of this evaluation are given in a separate presentation.

In all these cases, VP8 performed adequately in subjective evaluations; the numbers can be interpreted as showing that VP8 in "realtime" mode performed better than the "anchors" on both tests, but due to the amount of discussion occurring in the meetings about whether the precise parameters chosen for the tests made it a "fair" comparison, we will not state flatly that VP8 performed better than the anchors (AVC Baseline and AVC High Profile, respectively), but we will state flatly that there is no evidence that the anchors performed significantly better than VP8.

6. Performance evaluation

6.1. 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).

6.2. Hardware support

NOTE: This section contains mostly information that was valid as of October 2012. It will be updated.

As of October 2012, 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.

As of October 2012, more than a dozen chip manufacturers had announced chips with 1080p VP8 support, including Samsung (Exynos 5), NVIDIA (Tegra 3, Tegra 4), Marvell (Armada 1500), Broadcom (BCM28150), Texas Instruments (OMAP54xx), Freescale (i.MX 6), ST-Ericsson (NovaThor L9540), LG Electronics, Hisilicon (K3v2), Rockchip (RK2918, RK3066), Nufront (NS115), Ziilabs (ZMS40) and Allwinner (A10). Google estimates that a clear majority of leading mobile chipsets in 2013 will contain VP8 hardware support. (Nvidia Tegra4 info added after October 2012).

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.

The ChromeCast device, which is selling in significant numbers in the US, has VP8 hardware decode.

6.3. 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.

7. IPR status

The IETF has a long tradition of preferring non-encumbered IPR whenever possible, and especially to avoid IPR where using the technology requires making agreements with and payments to third parties as part of the cost of doing business. Among the reasons for this tradition is that the requirement for IPR agreements severely distorts the competitive landscape, and especially that it seriously hampers people attempting to implement standards in open source, or other business models where counting the number of installations or users is difficult, expensive or simply impossible.

As of this moment (October 4, 2013), the following IPR disclosures are filed in the IETF IPR database:

- o <https://datatracker.ietf.org/ipr/1571/> - by Google, declaring that the technology is royalty-free.
- o <https://datatracker.ietf.org/ipr/2035/> - by Nokia, which does not declare a royalty-free license.

The licensing terms for Google's IPR are available at <http://www.webmproject.org/license/additional/>.

The Nokia IPR mentioned above includes IPR that has been asserted in

ongoing litigation in Germany (Nokia v. HTC, District Court in Mannheim, Germany. 7 O 201/12); on one of the patents, the court has ruled that the phones in question (which support VP8) are not infringing. As mentioned in <http://blog.webmproject.org/2013/08/good-news-from-germany.html?m=0>; the case is still ongoing.

The following companies have asserted that any IPR relevant to VP8 they might have is available for licensing by Google under a royalty free license; the licensing terms are available at <http://www.webm-ccl.org/vp8/agreement/>, as well as details on the licensors:

- o CIF Licensing LLC
- o France Telecom
- o Fraunhofer-Gesellschaft zur Foerderung der angewandten Forschung e.V.
- o Fujitsu Limited
- o Koninklijke Philips Electronics N.V.
- o LG Electronics Inc.
- o Mitsubishi Electric Corporation
- o MPEG LA, LLC
- o NTT DOCOMO, INC
- o Panasonic Corporation
- o Samsung Electronics Co., Ltd.
- o Siemens Corporation

The license can be executed on-line from the link given above.

8. IANA Considerations

This document makes no request of IANA.

Note to RFC Editor: this section may be removed if this document is ever published as an RFC.

9. 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.

10. Acknowledgements

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

In addition, we wish to thank the people from the X264 mailing list who came forward with suggested improvements in the codec settings for the objective performance evaluations, Bo Burmann who re-ran the tests entirely independently of Google, Mohammed Raad and Lazar Bivolarski who prepared the materials for the subjective evaluation tests and Vittorio Baronici who performed them, and all the countless members of the RTCWEB working group who have debated extensively the matter of mandatory to implement video codecs.

11. References

11.1. 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-vp8-09 (work in progress), July 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.

11.2. Informative References

- [I-D.ietf-rtcweb-overview]
Alvestrand, H., "Overview: Real Time Protocols for Brower-

based Applications", draft-ietf-rtcweb-overview-08 (work in progress), September 2013.

Authors' Addresses

Harald Alvestrand
Google
Kungsbron 2
Stockholm, 11122
Sweden

Email: harald@alvestrand.no

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

Phone:
Fax:
Email: agrange@google.com
URI:

RTCWEB Working Group
Internet-Draft
Intended status: Standards Track
Expires: April 25, 2014

B. Burman
Ericsson
M. Isomaki
Nokia
B. Aboba
Microsoft Corporation
G. Martin-Cocher
BlackBerry Ltd
G. Mandyam
Qualcomm Innovation Center
X. Marjou
Orange
C. Jennings
J. Rosenberg
Cisco
D. Singer
Apple
October 22, 2013

H.264 as Mandatory to Implement Video Codec for WebRTC
draft-burman-rtcweb-h264-proposal-03

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 April 25, 2014.

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	2
2. Terminology	3
3. H.264 Overview	3
4. Implementations	3
5. Deployment	4
6. Licensing	6
6.1. Royalty Free for Innovation, Low-volume Shipments	6
6.2. Higher H.264/AVC Profile Tools Bundled	7
6.3. Licensing Stability	7
7. Performance	8
8. Profile/level	10
9. Negotiation	12
10. Summary	13
11. IANA Considerations	14
12. Security Considerations	14
13. Acknowledgements	14
14. References	14
14.1. Normative References	14
14.2. Informative References	14
Authors' Addresses	17

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:

H.264 Constrained High Profile Level 1.3, logically 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.

H.264 is mandatory in mobile wireless standards for multimedia telephony and packet switched streaming. It is also the leading de facto standard for web video content delivered in HTML5 or other technologies, and is supported in all major web browsers, mobile device platforms, and desktop operating systems.

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 Qualcomm, TI, Nvidia, Renesas, Mediatek, Huawei Hisilicon, Intel, Broadcom, Samsung. Those platforms all support H.264/AVC codec with dedicated hardware or DSP. The majority of the implementations also support low-delay real-time applications.

There are also other standards and specifications that support H.264. One notable area is wireless display standards, where H.264 support is pervasive among all the following leading standards:

- o AirPlay (Apple) [AirPlay].
- o WiDi (Intel) [WiDi].
- o Miracast (Wi-Fi Alliance) [Miracast].
- o Google Cast (Google) [GoogleCast].
- o DLNA (Sony) [DLNA].

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. a royalty-free open source implementation from Polycom including H.264/SVC support [Woon]. Microsoft has produced an H.264 prototype for use in browsers [CURtcWeb]. 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.

The WebM wiki [WEBM] shows only 3 (out of ~37) ARM SoCs which support VP8 encode and decode. All (~37) support H.264. This only represents a fraction of deployed SoCs. Almost all deployed SoCs, as well as future designs, support H.264 encode and decode, including desktop (Intel x86) chipsets.

The benefits of hardware encoder and decoder implementations typically have an order of magnitude or more performance advantage (e.g., 1080p versus 360p becomes achievable) and power savings (e.g., tens of milliwatts versus many hundreds of milliwatts or even watts are consumed just by the encoder and decoder). While VP8 proponents have argued codec power is not a major concern relative to displays, this neglects the advances in display technology that put the central processor back near the top power consumers.

5. Deployment

Today, the Internet runs on H.264 for real-time video communications. Though not yet on the web, video communications is in widespread usage on the Internet. It is supported in consumer applications both on the desktop and in mobile apps, provided by many players like Skype and Tango. It is in widespread usage for business communications, in many applications like Webex, Citrix Go-To-Meeting, Tandberg and Polycom telepresence systems, and many more. All of these are in widespread deployment and widespread usage, and are based on H.264.

If we want WebRTC to be successful, we must make sure it is something that can be adopted by the application providers who deploy real-time communications on the Internet. WebRTC needs to be for the developers - the people who are building applications. And a critical target customer base are the ones who are already doing voice and video communications - the ones with the network effect and user bases which need to be tapped to make this technology successful. If WebRTC does not embrace H.264, it will be at the risk of ignoring the needs of one of its most important set of potential adopters - the ones most eager to use it - the ones already in the market for real-time communications.

It may be argued that clients can be upgraded to support any new codec. Opus is mandatory despite no deployment. However, G.711 is also mandatory to ensure broad adoption. Likewise, H.264 should be mandatory to ensure broad video adoption, since it is as widely adopted in video as G.711 in voice. Also, video is more processing intensive than voice, and therefore often implemented in hardware that is not easily upgradeable. Other video systems use desktop software which can also be difficult to broadly upgrade. Still others provide SDKs and toolkits to third parties which cannot easily be upgraded. Others have mobile apps which users cannot be forcefully made to upgrade.

It may be argued that clients must be upgraded anyway to support ICE, DTLS-SRTP and other WebRTC requirements. Some will, some won't. For the latter, application providers will need to build server side gateways. While that adds cost and complexity, the need to transcode video would greatly escalate costs, perhaps making them prohibitive. The CPU cost for transcoding, and the corresponding impact on quality due to recoding and increased delays, are substantially larger compared to just transport-level gateway functions. Perhaps enough to make it impractical at scale.

It may be argued that deployed video systems and applications are insignificant compared to the larger number of web browsers that will support WebRTC. This misses a key point. Real-time communications exists amongst a set of users that can talk to each other, typically

because they are customers of the same service. Skype users can talk to each other. Tango users can talk to each other. There is, to date, relatively little federation for video between these providers, a problem which WebRTC is unlikely to remedy, as its causes have little to do with media stacks, and everything to do with business. Enabling real-time communications in the browser does not immediately create a connected user base that is the size of the web. WebRTC is just a media stack; the namespace is provided by the application provider, as is the size of the communications network to which that user can connect. Existing communications providers greatly value their user bases, and those user bases define the reachable communications network. When viewed in that lens, the most important thing for allowing a WebRTC user to reach a massive network, is enabling WebRTC to be usable by those which have existing networks of users. Of those, many are asking for H.264.

It may be argued that WebRTC should build for the future, and not be constrained by the past. This is reminiscent of the arguments made by those who advocated against IETF doing work on NAT or making NAT friendly protocols. The hope was the same - that IETF could, through standards, dictate the future as we wished it - that by designing protocols which didn't work through NAT, we would force the industry to move away from NAT and embrace IPv6. That strategy failed. The Internet is a living, breathing thing, constantly evolving. Those technologies which are successful are actually those which work for the Internet as it is today, not the Internet as we wish it could be. Those then allow the Internet to take a baby step forward, and from there, another step forward. Successful technologies require consideration for transition, as it is more important than the target. Just like NAT was, and still is, a reality on the Internet today, so too is H.264 a reality of the Internet today. Just like we could not upgrade the routers and switches to eliminate NAT, so too are we unable to upgrade many of the Internet endpoints today to instantly move away from H.264. We should learn from the past and define a WebRTC which can work with the applications in existence today, otherwise we significantly hinder the success and growth of WebRTC.

6. Licensing

6.1. Royalty Free for Innovation, Low-volume Shipments

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 [MPEGLA-License]. License fees for products that decode and encode

H.264 video remain though. Those fees [MPEGLA-Terms] are, and will very likely continue to be for the lifetime of MPEG-LA pool, \$0.20 per codec or less.

To paraphrase, the MPEG LA license does allow up to 100K units per year, per legal entity/company (type "a" sublicensees in MPEG LA's definition), to be shipped for zero (\$0) royalty cost. This should be adequate for many WebRTC innovators or start-ups to try out new implementations on a large set of users before incurring any patent royalty costs, a benefit to selecting a H.264/AVC profile as the mandatory codec.

6.2. Higher H.264/AVC Profile Tools Bundled

It should be noted that when one licenses the MPEG LA H.264/AVC pool, patents for higher profile tools - such as CABAC, 8x8 - are bundled in with those required for the Constrained Baseline Profile. Thus, these could optionally be used by WebRTC implementers to achieve even greater performance or efficiencies than using H.264 Constrained Baseline Profile alone.

It can also 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.

6.3. Licensing Stability

H.264 is a mature codec with a mature and well-known licensing model.

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, including patents owned by 11 patent holders [MpegLaVp8], under a open source friendly license with very restrictive reciprocity conditions.

Recently, VP8 was adopted as Working Draft for Video Coding for Browsers in MPEG, which is the first step in becoming an MPEG standard. As such, it will have to follow the ISO/IEC/ITU common patent policy [IsoIecItuPolicy], but IPR statements cannot be expected there for still some time. There is no guarantee that IPR

statements in MPEG will be royalty free (option 1), but may just as well be "Fair, Reasonable And Non-Discriminatory" (FRAND, option 2), and potential IPR owners that do not participate in this MPEG work are under no obligation to offer any license at all. This indicates that the licensing situation for VP8 has still not settled.

7. 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. While constituting an independent test material providing some indications, those tests however do not use exactly the proposed profiles and levels, which calls for performing a set of more targeted tests.

Google made a comparison test between VP8 and H.264 [GooglePSNR], providing a set of test scripts [GoogleScripts]. That test includes the use of rate control for both codecs. We believe this to be a comparison problem since rate control is part of the encoder, which as said above is typically not specified in video codec standards but left up to individual implementations. The quantization parameter (qp) level affects the rate/distortion tradeoff in video coding. Comparing using fixed qp-levels is what has typically been used when benchmarking new codecs, for example when benchmarking HEVC [H265] against H.264 in the JCT-VC [JCT-VC] standardization. We are going to select a codec (essentially bit stream format), not a rate control mechanism; once the codec is selected you can choose whatever rate control mechanism you wish that best suits your specific application. Therefore, we propose to compare the codecs with rate control off, using fixed quantization parameter (qp) levels.

Ericsson made a comparison using Google's published test scripts as baseline and changed the parameter settings in order to make it possible to measure using fixed qp. 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. We used the same eleven sequences as in the previous Google test, but limited them to the first 10 seconds since they varied from 10 seconds to minutes; this also eased computation time. The used video resolutions are 640x360 @ 30 fps, 640x480 @ 30 fps, 1280x720 @ 30 fps and 1280x720 @ 50 fps.

We used two H.264 encoder implementations:

- o X264, which is an open-source codec that can operate in everything from real-time to slow
- o JM, which is the (Joint Model) reference implementation that was used to develop H.264, and is very slow but attempts to be very efficient in terms of bits per quality

This is a summary of the results (complete scripts and results available here [H264VP8Tests]):

Test	Resulting bitrate at equivalent quality
X264 Constrained Baseline vs VP8	H.264 wins with 1%
JM Constrained Baseline vs VP8	H.264 wins with 4%
X264 Constrained High vs VP8	H.264 wins with 25%

JM Constrained High vs VP8	H.264 wins with 24%	
+-----+-----+-----+		

Table 1: Performance Comparison Results

It is interesting to note that the measurements are more stable in this test; the variance of the percentages for the different sequences is now around 70, down from around 700 in Google's test. We believe this is due to the removal of the rate controller, which acts as noise on the measurements.

It can also be noted that the Google method of calculating the rate differences does not give exactly the same numbers as the JCT-VC way of calculating Bjontegaard Delta bitrate (BD-rate) [PSNRdiff]. The main difference is that the JM score for Constrained High in the table above (Table 1) is around 29% better than VP8 if the JCT-VC way of calculating BD-rate is used.

A rough complexity estimate can be obtained from the total running times for the tests:

- o X264: 1 hour 3 minutes
- o VP8: 2 hours 0 minutes
- o JM: An order of magnitude slower

Again, video quality is difficult to compare. The authors however believe that the data provided in this section shows that H.264 Constrained Baseline is at least on par with VP8, while H.264 Constrained High seems to have a clear quality advantage. As a final note, the new H.265/HEVC standard [H265] clearly outperforms all three, but the authors think it is premature to mandate HEVC for WebRTC.

8. 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, logically extended to support 720p resolution at 30 Hz framerate, but in formal specification text it would have to be expressed as a restriction on a higher level.

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 9) 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 bi-predictive (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-prediction.

The below table summarizes the H.264 video encoding features used by Constrained Baseline Profile (CBP) and Constrained High Profile (CHP). For more information on the listed features, see [WikipediaAVC].

Feature	CBP	CHP
Bit depth per sample	8	8
Chroma formats	4:2:0	4:2:0
Flexible Macroblock Ordering (FMO)	No	No
Arbitrary Slice Ordering (ASO)	No	No
Redundant Slices	No	No
Data Partitioning	No	No
SI and SP slices	No	No
Interlaced coding	No	No
B slices	No	No
CABAC entropy coding	No	Yes
Monochrome 4:0:0	No	Yes
8x8 vs. 4x4 transform adaptivity	No	Yes
Quantization scaling matrices	No	Yes
Separate color QP control	No	Yes
Separate color plane coding	No	No
Predictive lossless coding	No	No
Weighted prediction	No	Yes

9. Negotiation

Given that there exist a fairly large set of defined profiles and levels (Section 8) in the H.264 specification, 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 8) capability for 720p30 is supported as an extension to Constrained High Profile Level 1.3 (or higher), the logical 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)

10. 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 8 and Section 9.

11. IANA Considerations

This document makes no request of IANA.

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

12. Security Considerations

No specific considerations apply to the information in this document.

13. Acknowledgements

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

14. References

14.1. Normative References

- [H264] ITU-T Recommendation H.264, "Advanced video coding for generic audiovisual services", April 2013, <<http://www.itu.int/rec/T-REC-H.264-201304-I>>.
- [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.

14.2. Informative References

- [AirPlay] Apple Inc, "AirPlay Overview: About AirPlay", September 2012, <<https://developer.apple.com/library/ios/documentation/AudioVideo/Conceptual/AirPlayGuide/Introduction/Introduction.html>>.
- [CURtcWeb] Microsoft Open Technologies, Inc., "CU-RTC-Web-Video", July 2013, <<http://html5labs.interoperabilitybridges.com/prototypes/cu-rtc-web-video/cu-rtc-web-video/info>>.

- [DLNA] DLNA(R), "Technical Overview", 2013, <<http://www.dlna.org/dlna-for-industry/digital-living/how-it-works/technical-overview>>.
- [GoogleCast] Google, "Supported Media Types - Google Cast", October 2013, <https://developers.google.com/cast/supported_media_types>.
- [GooglePSNR] The WebM Project, "VP8 Results", April 2013, <http://downloads.webmproject.org/ietf_tests/vp8_vs_h264_quality.html>.
- [GoogleScripts] The WebM Project, "VP8 vs H.264 Test Scripts", April 2013, <http://downloads.webmproject.org/ietf_tests/vp8_vs_h264.tar.xz>.
- [H264VP8Tests] Ericsson, "More H.264 vs VP8 tests", June 2013, <<http://www.ietf.org/mail-archive/web/rtcweb/current/zipDGJUU9JZ8n.zip>>.
- [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>.
- [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>.
- [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>>.
- [H265] ITU-T Recommendation H.265, "High Efficiency Video Coding", April 2013, <<http://www.itu.int/rec/T-REC-H.265-201304-I>>.

[Implementations]

Wikipedia, "H.264/MPEG-4 AVC products and implementations", April 2013, <http://en.wikipedia.org/wiki/H.264/MPEG-4_AVC_products_and_implementations>.

[IsoIecItuPolicy]

ISO, "ISO/IEC/ITU common patent policy", April 2007, <http://isotc.iso.org/livelink/livelink/fetch/2000/2122/3770791/Common_Policy.htm>.

[JCT-VC]

ITU-T, "JCT-VC - Joint Collaborative Team on Video Coding", , <<http://www.itu.int/en/ITU-T/studygroups/2013-2016/16/Pages/video/jctvc.aspx>>.

[MPEGLA-License]

MPEG LA, "AVC Patent Portfolio License Briefing", May 2009, <<http://www.mpegla.com/main/programs/avc/Documents/avcweb.pdf>>.

[MPEGLA-Terms]

MPEG LA, "SUMMARY OF AVC/H.264 LICENSE TERMS", , <http://www.mpegla.com/main/programs/avc/Documents/AVC_TermsSummary.pdf>.

[MPEGLA]

MPEG LA, "MPEG LAs 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.mpegla.com/Lists/MPEG%20LA%20News%20List/Attachments/231/n-10-08-26.pdf>.

[Miracast]

Wi-Fi Alliance(R), "What formats does Miracast support?", 2013, <<http://www.wi-fi.org/knowledge-center/faq/what-formats-does-miracast-support>>.

[MpegLaVp8]

O'Reilly, T., "Google and MPEG LA Announce Agreement Covering VP8 Video Format", March 2013, <<http://www.mpegla.com/Lists/MPEG%20LA%20News%20List/Attachments/88/n-13-03-07.pdf>>.

[PSNRdiff]

Bjontegaard, G., "Calculation of Average PSNR Differences between RD-Curves", ITU-T SG16 Q.6 Document VCEG-M33, April 2001.

[WEBM]

The WebM Project, "ARM SoCs", , <<http://wiki.webmproject.org/hardware/arm-socs>>.

- [WiDi] Intel Corporation, "Intel(R) Wireless Display and Intel(R) Pro Wireless Display", October 2013, <<http://www.intel.com/content/www/us/en/architecture-and-technology/intel-wireless-display.html>>.
- [WikipediaAVC] Wikipedia, "H.264/MPEG-4 AVC", October 2013, <http://en.wikipedia.org/wiki/H.264/MPEG-4_AVC>.
- [Woon] Polycom, "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>>.

Authors' Addresses

Bo Burman
Ericsson
Farogatan 6
Stockholm 16480
Sweden

Email: bo.burman@ericsson.com

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

Email: markus.isomaki@nokia.com

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

Email: bernard_aboba@hotmail.com

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

Email: gmartincocher@blackberry.com

Giri Mandyam
Qualcomm Innovation Center

Email: mandyam@quicinc.com

Xavier Marjou
Orange
2, avenue Pierre Marzin
Lannion 22307
France

Email: xavier.marjou@orange.com

Cullen Jennings
Cisco
170 West Tasman Drive
San Jose, CA 95134
United States

Email: fluffy@cisco.com

Jonathan Rosenberg
Cisco
170 West Tasman Drive
San Jose, CA 95134
USA

Email: jdrosen@cisco.com

David Singer
Apple

Email: singer@apple.com

RTCWeb Working Group
Internet-Draft
Intended status: Standards Track
Expires: April 24, 2014

R. Jesup
Mozilla
S. Loreto
Ericsson
M. Tuexen
Muenster Univ. of Appl. Sciences
October 21, 2013

RTCWeb Data Channels
draft-ietf-rtcweb-data-channel-06.txt

Abstract

The Real-Time Communication in WEB-browsers (RTCWeb) working group is charged to provide protocol support for direct interactive rich communication using audio, video, and data between two peers' web-browsers. This document specifies the non-media data transport aspects of the RTCWeb framework. It provides an architectural overview of how the Stream Control Transmission Protocol (SCTP) is used in the RTCWeb context as a generic transport service allowing WEB-browsers to exchange generic data from peer to peer.

Status of This Memo

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

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

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

This Internet-Draft will expire on April 24, 2014.

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	2
2. Conventions	3
3. Use Cases	3
3.1. Use Cases for Unreliable Data Channels	3
3.2. Use Cases for Reliable Data Channels	3
4. Requirements	4
5. SCTP over DTLS over UDP Considerations	5
6. The Usage of SCTP in the RTCWeb Context	8
6.1. SCTP Protocol Considerations	8
6.2. Association Setup	9
6.3. SCTP Streams	9
6.4. Channel Definition	10
6.5. Opening a Channel	10
6.6. Transferring User Data on a Channel	10
6.7. Closing a Channel	11
7. Security Considerations	11
8. IANA Considerations	12
9. Acknowledgments	12
10. References	12
10.1. Normative References	12
10.2. Informative References	14
Authors' Addresses	14

1. Introduction

Non-media data types in the context of RTCWeb are handled by using SCTP [RFC4960] encapsulated in DTLS [RFC6347].

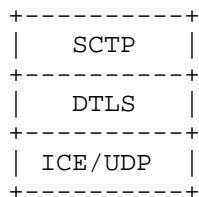


Figure 1: Basic stack diagram

The encapsulation of SCTP over DTLS (see [I-D.ietf-tsvwg-sctp-dtls-encaps]) over ICE/UDP (see [RFC5245]) provides a NAT traversal solution together with confidentiality, source authentication, and integrity protected transfers. This data transport service operates in parallel to the media transports, and all of them can eventually share a single transport-layer port number.

SCTP as specified in [RFC4960] with the partial reliability extension defined in [RFC3758] provides multiple streams natively with reliable, and partially-reliable delivery modes for user messages. Using the reconfiguration extension defined in [RFC6525] allows to increase the number of streams during the lifetime of an SCTP association and to reset individual SCTP streams.

The remainder of this document is organized as follows: Section 3 and Section 4 provide use cases and requirements for both unreliable and reliable peer to peer data channels; Section 5 arguments SCTP over DTLS over UDP; Section 6 provides the specification of how SCTP should be used by the RTCWeb protocol framework for transporting non-media data between WEB-browsers.

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. Use Cases

This section defined use cases specific to data channels. For general use cases see [I-D.ietf-rtcweb-use-cases-and-requirements].

3.1. Use Cases for Unreliable Data Channels

U-C 1: A real-time game where position and object state information is sent via one or more unreliable data channels. Note that at any time there may be no media channels, or all media channels may be inactive, and that there may also be reliable data channels in use.

U-C 2: Providing non-critical information to a user about the reason for a state update in a video chat or conference, such as mute state.

3.2. Use Cases for Reliable Data Channels

U-C 3: A real-time game where critical state information needs to be transferred, such as control information. Such a game may have no media channels, or they may be inactive at any given time, or may only be added due to in-game actions.

U-C 4: Non-realtime file transfers between people chatting. Note that this may involve a large number of files to transfer sequentially or in parallel, such as when sharing a folder of images or a directory of files.

U-C 5: Realtime text chat during an audio and/or video call with an individual or with multiple people in a conference.

U-C 6: Renegotiation of the set of media streams in the PeerConnection.

U-C 7: Proxy browsing, where a browser uses data channels of a PeerConnection to send and receive HTTP/HTTPS requests and data, for example to avoid local internet filtering or monitoring.

4. Requirements

This section lists the requirements for P2P data channels between two browsers.

- Req. 1: Multiple simultaneous data channels **MUST** be supported. Note that there may 0 or more media streams in parallel with the data channels, and the number and state (active/inactive) of the media streams may change at any time.
- Req. 2: Both reliable and unreliable data channels **MUST** be supported.
- Req. 3: Data channels **MUST** be congestion controlled; either individually, as a class, or in conjunction with the media streams, to ensure that data channels don't cause congestion problems for the media streams, and that the RTCWeb PeerConnection as a whole is fair with competing traffic such as TCP.
- Req. 4: The application **SHOULD** be able to provide guidance as to the relative priority of each data channel relative to each other, and relative to the media streams. [TBD: how this is encoded and what the impact of this is.] This will interact with the congestion control algorithms.

- Req. 5: Data channels MUST be secured; allowing for confidentiality, integrity and source authentication. See [I-D.ietf-rtcweb-security] and [I-D.ietf-rtcweb-security-arch] for detailed info.
- Req. 6: Data channels MUST provide message fragmentation support such that IP-layer fragmentation can be avoided no matter how large a message the JavaScript application passes to be sent. It also MUST ensure that large data channel transfers don't unduly delay traffic on other data channels.
- Req. 7: The data channel transport protocol MUST NOT encode local IP addresses inside its protocol fields; doing so reveals potentially private information, and leads to failure if the address is depended upon.
- Req. 8: The data channel transport protocol SHOULD support unbounded-length "messages" (i.e., a virtual socket stream) at the application layer, for such things as image-file-transfer; Implementations might enforce a reasonable message size limit.
- Req. 9: The data channel transport protocol SHOULD avoid IP fragmentation. It MUST support PMTU (Path MTU) discovery and MUST NOT rely on ICMP or ICMPv6 being generated or being passed back, especially for PMTU discovery.
- Req. 10: It MUST be possible to implement the protocol stack in the user application space.

5. SCTP over DTLS over UDP Considerations

The important features of SCTP in the RTCWeb context are:

- o Usage of a TCP-friendly congestion control.
- o The congestion control is modifiable for integration with media stream congestion control.
- o Support of multiple unidirectional streams, each providing its own notion of ordered message delivery.
- o Support of ordered and out-of-order message delivery.
- o Supporting arbitrary large user message by providing fragmentation and reassembly.
- o Support of PMTU-discovery.

- o Support of reliable or partially reliable message transport.

SCTP multihoming will not be used in RTCWeb. The SCTP layer will simply act as if it were running on a single-homed host, since that is the abstraction that the lower layer (a connection oriented, unreliable datagram service) exposes.

The encapsulation of SCTP over DTLS defined in [I-D.ietf-tsvwg-sctp-dtls-encaps] provides confidentiality, source authenticated, and integrity protected transfers. Using DTLS over UDP in combination with ICE enables NAT traversal in IPv4 based networks. SCTP as specified in [RFC4960] MUST be used in combination with the extension defined in [RFC3758] and provides the following interesting features for transporting non-media data between browsers:

- o Support of multiple unidirectional streams.
- o Ordered and unordered delivery of user messages.
- o Reliable and partial-reliable transport of user messages.

Each SCTP user message contains a so called Payload Protocol Identifier (PPID) that is passed to SCTP by its upper layer and sent to its peer. This value can be used to multiplex multiple protocols over a single SCTP association. The sender provides for each protocol a specific PPID and the receiver can demultiplex the messages based on the received PPID.

The encapsulation of SCTP over DTLS, together with the SCTP features listed above satisfies all the requirements listed in Section 4.

The layering of protocols for WebRTC is shown in the following Figure 2.

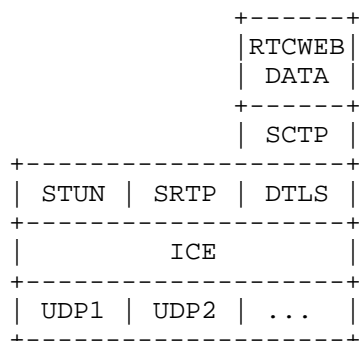


Figure 2: WebRTC protocol layers

This stack (especially in contrast to DTLS over SCTP [RFC6083] in combination with SCTP over UDP [RFC6951]) has been chosen because it

- o supports the transmission of arbitrary large user messages.
- o shares the DTLS connection with the media channels.
- o provides privacy for the SCTP control information.

Considering the protocol stack of Figure 2 the usage of DTLS over UDP is specified in [RFC6347], while the usage of SCTP on top of DTLS is specified in [I-D.ietf-tsvwg-sctp-dtls-encaps].

Since DTLS is typically implemented in user-land, the SCTP stack also needs to be a user-land stack.

When using DTLS as the lower layer, only single homed SCTP associations MUST be used, since DTLS does not expose any address management to its upper layer. The ICE/UDP layer can handle IP address changes during a session without needing to notify the DTLS and SCTP layers, though it would be advantageous to retest Path MTU on an IP address change.

DTLS implementations used for this stack SHOULD support controlling fields of the IP layer like the Don't Fragment (DF)-bit in case of IPv4 and the Differentiated Services Code Point (DSCP) field required for supporting [I-D.ietf-rtcweb-qos]. Being able to set the (DF)-bit in case of IPv4 is required for performing path MTU discovery. The DTLS implementation SHOULD also support sending user messages exceeding the Path MTU.

Incoming ICMP or ICMPv6 messages can't be processed by the SCTP layer, since there is no way to identify the corresponding

association. Therefore SCTP MUST support performing Path MTU discovery without relying on ICMP or ICMPv6 as specified in [RFC4821] using probing messages specified in [RFC4820]. The initial Path MTU at the IP layer MUST NOT exceed 1200 bytes for IPv4 and 1280 for IPv6. Taking an overhead of 20 bytes for IPv4, 40 bytes for IPv6, 8 bytes for UDP, 13 + X for DTLS and 28 bytes for SCTP into account, this results in an SCTP payload of 1131 - X when IPv4 is used and 1192 - X bytes when IPv6 is used.

In general, the lower layer interface of an SCTP implementation SHOULD be adapted to address the differences between IPv4 and IPv6 (being connection-less) or DTLS (being connection-oriented).

When protocol stack of Figure 2 is used, DTLS protects the complete SCTP packet, so it provides confidentiality, integrity and source authentication of the complete SCTP packet.

This protocol stack MUST support the usage of multiple SCTP streams. A user message can be sent ordered or unordered and with partial or full reliability. The partial reliability extension MUST support policies to limit

- o the transmission and retransmission by time.
- o the number of retransmissions.

Limiting the number of retransmissions to zero combined with unordered delivery provides a UDP-like service where each user message is sent exactly once and delivered in the order received.

SCTP provides congestion control on a per-association base. This means that all SCTP streams within a single SCTP association share the same congestion window. Traffic not being sent over SCTP is not covered by the SCTP congestion control. Using a congestion control different from the standard one might improve the impact on the parallel SRTP media streams. Since SCTP does not support the negotiation of a congestion control algorithm, alternate congestion controls SHOULD only require a different sender side behavior using existing information carried in the association.

6. The Usage of SCTP in the RTCWeb Context

6.1. SCTP Protocol Considerations

The DTLS encapsulation of SCTP packets as described in [I-D.ietf-tsvwg-sctp-dtls-encaps] MUST be used.

The following SCTP protocol extensions are required:

- o The stream reset extension defined in [RFC6525] MUST be supported. It is used for closing channels.
- o The dynamic address reconfiguration extension defined in [RFC5061] MUST be used to signal the support of the stream reset extension defined in [RFC6525], other features of [RFC5061] MUST NOT be used.
- o The partial reliability extension defined in [RFC3758] MUST be supported. In addition to the timed reliability PR-SCTP policy defined in [RFC3758], the limited retransmission policy defined in [I-D.tuexen-tsvwg-sctp-prpolicies] MUST be supported.

Once support for message interleaving as currently being discussed in [I-D.stewart-tsvwg-sctp-ndata] is available, it SHOULD be supported.

6.2. Association Setup

The SCTP association will be set up when the two endpoints of the WebRTC PeerConnection agree on opening it, as negotiated by JSEP (typically an exchange of SDP) [I-D.ietf-rtcweb-jsep]. Additionally, the negotiation SHOULD include some type of congestion control selection. It will use the DTLS connection selected via SDP; typically this will be shared via BUNDLE or equivalent with DTLS connections used to key the DTLS-SRTP media streams.

The application SHOULD indicate the initial number of streams required when opening the association, and if no value is supplied, the implementation SHOULD provide an appropriate default. If more simultaneous streams are needed, [RFC6525] allows adding additional (but not removing) streams to an existing association. Note there can be up to 65536 SCTP streams per SCTP association in each direction.

6.3. SCTP Streams

SCTP defines a stream as a unidirectional logical channel existing within an SCTP association one to another SCTP endpoint. The streams are used to provide the notion of in-sequence delivery and for multiplexing. Each user message is sent on a particular stream, either order or unordered. Ordering is preserved only for ordered messages sent on the same stream.

6.4. Channel Definition

The W3C has consensus on defining the application API for WebRTC DataChannels to be bidirectional. They also consider the notions of in-sequence, out-of-sequence, reliable and unreliable as properties of Channels. One strong wish is for the application-level API to be close to the API for WebSockets, which implies bidirectional streams of data and waiting for onopen to fire before sending, a textual label used to identify the meaning of the stream, among other things. Each data channel also has a priority. These priorities MUST NOT be strict priorities.

The realization of a bidirectional Data Channel is a pair of one incoming stream and one outgoing SCTP stream.

Note that there's no requirement for the SCTP streams used to create a bidirectional channel have the same number in each direction. How stream values are selected is protocol and implementation dependent.

6.5. Opening a Channel

Data channels can be opened by using internal or external negotiation. The details are out of scope of this document.

A simple protocol for internal negotiation is specified in [I-D.ietf-rtcweb-data-protocol] and MUST be supported.

When one side wants to open a channel using external negotiation, it picks a Stream. This can be based on the DTLS role (the client picks even stream identifiers, the server odd stream identifiers) or done in a different way. However, the application is responsible for avoiding collisions with existing Streams. If it attempts to re-use a Stream which is part of an existing Channel, the addition SHOULD fail. In addition to choosing a Stream, the application SHOULD also inform the protocol of the options to use for sending messages. The application MUST ensure in an application-specific manner that the other side will also inform the protocol that the selected Stream is to be used, and the parameters for sending data from that side.

6.6. Transferring User Data on a Channel

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.

No more than one message should be put into an SCTP user message.

The SCTP Payload Protocol Identifiers (PPIDs) are used to signal the interpretation of the "Payload data". For identifying a JavaScript string the PPID "DOMString Last" MUST be used, for JavaScript binary data (ArrayBuffer or Blob) the PPID "Binary Data Last" MUST be used (see Section 8).

The SCTP base protocol specified in [RFC4960] does not support the interleaving of user messages. Therefore sending a large user message can monopolize the SCTP association. To overcome this limitation, [I-D.stewart-tsvwg-sctp-ndata] defines an extension to support message interleaving. Once such an extension is available, it SHOULD be used.

As long as message interleaving is not supported, the sending application SHOULD fragment large user messages for reliable and ordered data channels. For sending large JavaScript strings, it uses the PPID "DOMString Partial" for all but the last fragments and the PPID "DOMString Last" for the last one. For JavaScript binary data the PPIDs "Binary Data Partial" and "Binary Data Last" are used. The reassembly based on the PPID MUST be supported. For data channel which are not reliable and ordered, the sender MAY limit the maximum message size to avoid monopolization.

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.

The sender MAY disable the Nagle algorithm to minimize the latency.

6.7. Closing a Channel

Closing of a Data Channel MUST be signaled by resetting the corresponding outgoing streams [RFC6525]. Resetting a stream set the Stream Sequence Numbers (SSNs) of the stream back to 'zero' with a corresponding notification to the application layer that the reset has been performed. Streams are available to reuse after a reset has been performed.

[RFC6525] also guarantees that all the messages are delivered (or abandoned) before resetting the stream.

7. Security Considerations

This document does not add any additional considerations to the ones given in [I-D.ietf-rtcweb-security] and [I-D.ietf-rtcweb-security-arch].

8. IANA Considerations

[NOTE to RFC-Editor:

"RFCXXXX" is to be replaced by the RFC number you assign this document.

]

This document uses four already registered SCTP Payload Protocol Identifiers (PPIDs). [RFC4960] creates the registry "SCTP Payload Protocol Identifiers" from which these identifiers were assigned. IANA is requested to update the reference of these four assignments to point to this document. Therefore these four assignments should be updated to read:

Value	SCTP PPID	Reference
DOMString Last	51	[RFCXXXX]
Binary Data Partial	52	[RFCXXXX]
Binary Data Last	53	[RFCXXXX]
DOMString Partial	54	[RFCXXXX]

9. Acknowledgments

Many thanks for comments, ideas, and text from Harald Alvestrand, Adam Bergkvist, Cullen Jennings, Eric Rescorla, Randall Stewart, Justin Uberti, and Magnus Westerlund.

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.
- [RFC3758] Stewart, R., Ramalho, M., Xie, Q., Tuexen, M., and P. Conrad, "Stream Control Transmission Protocol (SCTP) Partial Reliability Extension", RFC 3758, May 2004.
- [RFC4820] Tuexen, M., Stewart, R., and P. Lei, "Padding Chunk and Parameter for the Stream Control Transmission Protocol (SCTP)", RFC 4820, March 2007.

- [RFC4821] Mathis, M. and J. Heffner, "Packetization Layer Path MTU Discovery", RFC 4821, March 2007.
- [RFC4960] Stewart, R., "Stream Control Transmission Protocol", RFC 4960, September 2007.
- [RFC5061] Stewart, R., Xie, Q., Tuexen, M., Maruyama, S., and M. Kozuka, "Stream Control Transmission Protocol (SCTP) Dynamic Address Reconfiguration", RFC 5061, September 2007.
- [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.
- [RFC6525] Stewart, R., Tuexen, M., and P. Lei, "Stream Control Transmission Protocol (SCTP) Stream Reconfiguration", RFC 6525, February 2012.
- [I-D.stewart-tsvwg-sctp-ndata]
Stewart, R., Tuexen, M., Loreto, S., and R. Seggelmann, "A New Data Chunk for Stream Control Transmission Protocol", draft-stewart-tsvwg-sctp-ndata-03 (work in progress), October 2013.
- [I-D.ietf-rtcweb-data-protocol]
Jesup, R., Loreto, S., and M. Tuexen, "WebRTC Data Channel Protocol", draft-ietf-rtcweb-data-protocol-00 (work in progress), July 2013.
- [I-D.ietf-tsvwg-sctp-dtls-encaps]
Tuexen, M., Stewart, R., Jesup, R., and S. Loreto, "DTLS Encapsulation of SCTP Packets", draft-ietf-tsvwg-sctp-dtls-encaps-02 (work in progress), October 2013.
- [I-D.ietf-rtcweb-security]
Rescorla, E., "Security Considerations for WebRTC", draft-ietf-rtcweb-security-05 (work in progress), July 2013.
- [I-D.ietf-rtcweb-security-arch]
Rescorla, E., "WebRTC Security Architecture", draft-ietf-rtcweb-security-arch-07 (work in progress), July 2013.
- [I-D.ietf-rtcweb-jsep]

Uberti, J. and C. Jennings, "Javascript Session Establishment Protocol", draft-ietf-rtcweb-jsep-04 (work in progress), September 2013.

[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.

10.2. Informative References

[RFC6083] Tuexen, M., Seggelmann, R., and E. Rescorla, "Datagram Transport Layer Security (DTLS) for Stream Control Transmission Protocol (SCTP)", RFC 6083, January 2011.

[RFC6951] Tuexen, M. and R. Stewart, "UDP Encapsulation of Stream Control Transmission Protocol (SCTP) Packets for End-Host to End-Host Communication", RFC 6951, May 2013.

[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-12 (work in progress), October 2013.

[I-D.tuexen-tsvwg-sctp-prpolicies]

Tuexen, M., Seggelmann, R., Stewart, R., and S. Loreto, "Additional Policies for the Partial Delivery Extension of the Stream Control Transmission Protocol", draft-tuexen-tsvwg-sctp-prpolicies-03 (work in progress), October 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 Working Group
Internet-Draft
Intended status: Standards Track
Expires: April 24, 2014

R. Jesup
Mozilla
S. Loreto
Ericsson
M. Tuexen
Muenster Univ. of Appl. Sciences
October 21, 2013

RTCWeb Data Channel Protocol
draft-ietf-rtcweb-data-protocol-01.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 a simple protocol for establishing symmetric data channels between the peers. It uses a two way handshake and allows sending of user data without waiting for the handshake to complete.

Status of This Memo

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

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

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

This Internet-Draft will expire on April 24, 2014.

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	2
2. Conventions	2
3. Terminology	2
4. Protocol Overview	3
5. Message Formats	4
5.1. DATA_CHANNEL_OPEN Message	4
5.2. DATA_CHANNEL_ACK Message	6
6. Procedures	6
7. Security Considerations	7
8. IANA Considerations	7
8.1. SCTP Payload Protocol Identifier	7
8.2. New Message Type Registry	8
8.3. New Channel Type Registry	8
8.4. New Protocol Registry	9
9. Acknowledgments	9
10. References	10
10.1. Normative References	10
10.2. Informational References	10
Authors' Addresses	10

1. Introduction

The data channel protocol is designed to provide, in the WebRTC data channel context [I-D.ietf-rtcweb-data-channel], a simple in-band method to open symmetric data channels. As discussed in [I-D.ietf-rtcweb-data-channel], the protocol uses the Stream Control Transmission Protocol (SCTP) [RFC4960] encapsulated in the Datagram Transport Layer Security (DTLS) [RFC6347] as described in [I-D.ietf-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 (0-65534). Note: stream identifier 65535 is reserved due to SCTP Init messages allowing a maximum of 65535 streams to be negotiated (0-65534).

Channel: Two Streams with the same identifier, one in each direction, which 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.

The set of consistent properties includes

- o whether the messages are transmitted reliable or unreliable. In case of unreliable transmissions, the same level of unreliability is used.
- o whether the messages are delivered in-order or out-of order.
- o an optional label for the Channel.
- o an optional protocol for the Channel.
- o the outgoing SCTP stream.

The data channel protocol uses a two way handshake to open a data channel. The side wanting to open a data channel selects an unused Stream and sends a DATA_CHANNEL_OPEN message. The peer responds with a DATA_CHANNEL_ACK message. Then the data channel is open. Please note that the opening side can send user messages before the DATA_CHANNEL_ACK is received. These data channel messages are sent on the same Stream as the user messages belonging to the data channel. The demultiplexing is based on the SCTP payload protocol identifier.

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 based on the underlying DTLS connection role when used in RTCWeb, with the side acting as the DTLS client using even stream identifiers.

Note: There is no attempt to resolve label glare; if both sides open a Channel labeled "x" at the same time, there will be two Channels labeled "x" - one on an even Stream pair, one on an odd pair.

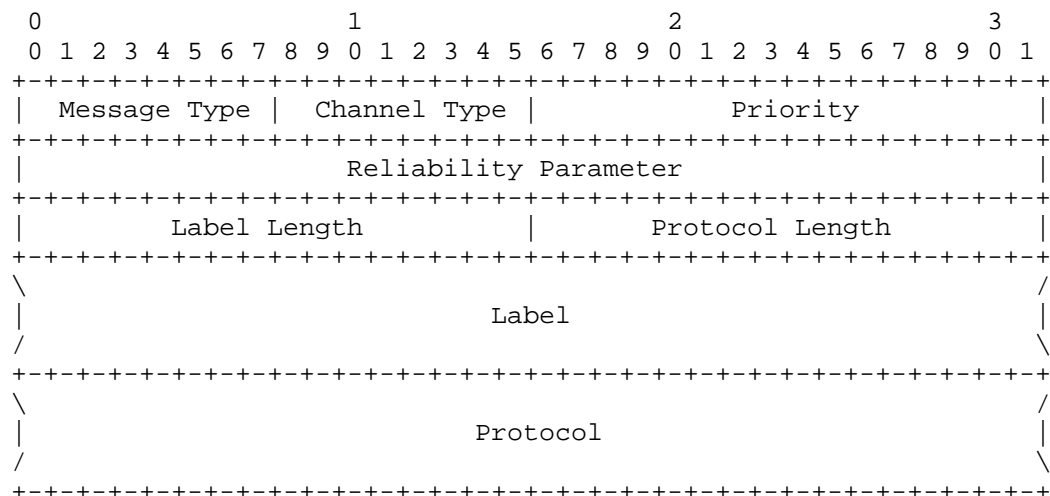
The protocol field is to ease cross-application interoperation ("federation") by identifying the data being passed with an IANA-registered string, and may be useful for homogenous applications which may create more than one type of Channel.

5. Message Formats

Every data channel protocol message starts with a one byte field called "Message Type" which indicates the type of the message. The corresponding values are managed by IANA (see Section 8.2).

5.1. DATA_CHANNEL_OPEN Message

This message is sent initially on the stream used for user messages using the channel.



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.

Channel Type: 1 byte (unsigned integer)

This field specifies the type of the channel to be opened and the values are managed by IANA (see Section 8.3):

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 milli-seconds in the Reliability Parameter. This life-time starts when providing the user message to the Javascript engine.

DATA_CHANNEL_PARTIAL_RELIABLE_TIMED_UNORDERED (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 milli-seconds in the Reliability Parameter. This life-time starts when providing the user message to the Javascript engine.

Priority: 2 bytes (integer)
The priority of the channel.

Reliability Parameter: 4 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.

Label Length: 2 bytes (unsigned integer)
The length of the label field in bytes.

Protocol Length: 2 bytes (unsigned 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 is an IANA-registered protocol (see Section 8.4).

5.2. DATA_CHANNEL_ACK Message

This message is sent in response to an `DATA_CHANNEL_OPEN_RESPONSE` message on the stream used for user messages using the channel. Reception of this message tells the opener that the channel setup handshake is complete.

```

      0                   1                   2                   3
      0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
| Message Type |
+---+---+---+---+---+

```

Message Type: 1 byte (unsigned integer)

This field holds the IANA defined message type for the the `DATA_CHANNEL_ACK` message. The suggested value of this field for IANA is 0x02.

6. Procedures

All data channel protocol messages **MUST** be sent requesting ordered delivery and using reliable transmission. They **MUST** be sent on the same outgoing SCTP stream as the user messages belonging to the corresponding data channel. Multiplexing and demultiplexing is done by using the SCTP payload protocol identifier (PPID). Therefore data channel protocol message **MUST** be sent with the assigned PPID for the data channel protocol (see Section 8.1). Other message **MUST NOT** be sent using this PPID.

If one sides wants to open a data channel, it chooses a free outgoing SCTP stream. If the side is the DTLS client, it **MUST** choose an even stream identifier, if the side is the DTLS server, it **MUST** choose an odd one. It fills in the parameters of the `DATA_CHANNEL_OPEN` message and sends it on the chosen SCTP stream.

After the `DATA_CHANNEL_OPEN` message has been sent, the sender of it can start sending messages containing user data without waiting for the reception of the corresponding `DATA_CHANNEL_ACK` message. However, before the `DATA_CHANNEL_ACK` message or any other message has been received on the data channel, all other messages containing user

data and belonging to the data channel MUST be sent ordered, not matter whether the data channel is ordered or not. After the DATA_CHANNEL_ACK or any other message has been received on the data channel, messages containing user data MUST be send ordered on ordered data channels and MUST be sent unordered on unordered data channels. Therefore receiving a message containing user data on an unused SCTP stream indicates an error. The corresponding outgoing SCTP stream MUST be reset using [RFC6525].

If a DATA_CHANNEL_OPEN message is received on an unused stream, the stream identifier corresponds to the role of the peer and all parameters in the DATA_CHANNEL_OPEN message are valid, then a corresponding DATA_CHANNEL_ACK message is sent on the stream with the same stream identifier as the one the DATA_CHANNEL_OPEN message was received on.

If a DATA_CHANNEL_OPEN message is received on an already used SCTP stream or there are any problems with parameters within the DATA_CHANNEL_OPEN message or the DATA_CHANNEL_OPEN message itself is not well-formed, the receiver MUST reset the corresponding outgoing SCTP stream using [RFC6525] and MUST NOT send a DATA_CHANNEL_ACK message in response to the received message. Therefore, receiving an SCTP stream reset request for a stream on which no DATA_CHANNEL_ACK message has been received indicates to the sender of the corresponding DATA_CHANNEL_OPEN message the failure of the data channel setup procedure.

7. Security Considerations

This document does not add any additional considerations to the ones given in [I-D.ietf-rtcweb-security] and [I-D.ietf-rtcweb-security-arch].

8. IANA Considerations

[NOTE to RFC-Editor:

"RFCXXXX" is to be replaced by the RFC number you assign this document.

]

IANA is asked to update the reference of an already existing SCTP PPID assignment and to create three new registries for the data channel protocol.

8.1. SCTP Payload Protocol Identifier

This document uses one already registered SCTP Payload Protocol Identifier (PPID). [RFC4960] creates the registry "SCTP Payload Protocol Identifiers" from which this identifier was assigned. IANA is requested to update the reference of this assignment to point to this document. Therefore this assignment should be updated to read:

Value	SCTP PPID	Reference
WebRTC Control	50	[RFCXXXX]

8.2. New Message Type Registry

IANA is requested to create a new registration table "Message Type Registry" for the data channel protocol to manage the one byte "Message Type" field in data channel messages (see Section 5).

The assignment of new message types is done through an RFC required action, as defined in [RFC5226]. Documentation of the new message type MUST contain the following information:

1. A name for the new message type;
2. A detailed procedural description of the use of messages with the new type within the operation of the data channel protocol.

Initially the following values need to be registered:

Name	Type	Reference
Reserved	0x00	[RFCXXXX]
Reserved	0x01	[RFCXXXX]
DATA_CHANNEL_ACK	0x02	[RFCXXXX]
DATA_CHANNEL_OPEN	0x03	[RFCXXXX]
Unassigned	0x04-0xfe	
Reserved	0xff	[RFCXXXX]

8.3. New Channel Type Registry

IANA is requested to create a new registration table "Channel Type Registry" for the data channel protocol to manage the one byte "Channel Type" field in DATA_CHANNEL_OPEN messages (see Section 5.1).

The assignment of new message types is done through an RFC required action, as defined in [RFC5226]. Documentation of the new channel type MUST contain the following information:

1. A name for the new channel type;
2. A detailed procedural description of the user message handling for data channels using this new channel type.

Please note that if new channel types support ordered and unordered message delivery, the high order bit SHOULD be used to indicate whether the message delivery is unordered or not.

Initially the following values need to be registered:

Name	Type	Reference
DATA_CHANNEL_RELIABLE	0x00	[RFCXXXX]
DATA_CHANNEL_RELIABLE_UNORDERED	0x80	[RFCXXXX]
DATA_CHANNEL_PARTIAL_RELIABLE_REXMIT	0x01	[RFCXXXX]
DATA_CHANNEL_PARTIAL_RELIABLE_REXMIT_UNORDERED	0x81	[RFCXXXX]
DATA_CHANNEL_PARTIAL_RELIABLE_TIMED	0x02	[RFCXXXX]
DATA_CHANNEL_PARTIAL_RELIABLE_TIMED_UNORDERED	0x82	[RFCXXXX]
Reserved	0x7f	[RFCXXXX]
Reserved	0xff	[RFCXXXX]
Unassigned	rest	

8.4. New Protocol Registry

IANA is requested to create a new registration table "Protocol Registry" for the data channel protocol to manage the "Protocol" field of type string in DATA_CHANNEL_OPEN messages (see Section 5.1).

The assignment of new message types is done through an First Come First Served action, as defined in [RFC5226]. Documentation of the new protocol MUST contain the following information:

1. A name for the protocol;
2. A reference for the protocol indicated by the registered string.

Initially this registry is empty.

9. 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.

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.
- [RFC4960] Stewart, R., "Stream Control Transmission Protocol", RFC 4960, September 2007.
- [RFC5226] Narten, T. and H. Alvestrand, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 5226, May 2008.
- [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-tsvwg-sctp-dtls-encaps]
Tuexen, M., Stewart, R., Jesup, R., and S. Loreto, "DTLS Encapsulation of SCTP Packets", draft-ietf-tsvwg-sctp-dtls-encaps-02 (work in progress), October 2013.

10.2. Informational References

- [I-D.ietf-rtcweb-data-channel]
Jesup, R., Loreto, S., and M. Tuexen, "RTCWeb Data Channels", draft-ietf-rtcweb-data-channel-05 (work in progress), July 2013.
- [I-D.ietf-rtcweb-security]
Rescorla, E., "Security Considerations for WebRTC", draft-ietf-rtcweb-security-05 (work in progress), July 2013.
- [I-D.ietf-rtcweb-security-arch]
Rescorla, E., "WebRTC Security Architecture", draft-ietf-rtcweb-security-arch-07 (work in progress), July 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

Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: April 25, 2014

J. Uberti
Google
C. Jennings
Cisco
October 22, 2013

Javascript Session Establishment Protocol
draft-ietf-rtcweb-jsep-05

Abstract

This document describes the mechanisms for allowing a Javascript application to control the signaling plane of a multimedia session via the interface specified in the W3C RTCPeerConnection API, and discusses how this relates to existing signaling protocols.

Status of This Memo

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

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

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

This Internet-Draft will expire on April 25, 2014.

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.	General Design of JSEP	3
1.2.	Other Approaches Considered	5
2.	Terminology	6
3.	Semantics and Syntax	6
3.1.	Signaling Model	6
3.2.	Session Descriptions and State Machine	7
3.3.	Session Description Format	9
3.4.	ICE	10
3.4.1.	ICE Candidate Trickling	10
3.4.1.1.	ICE Candidate Format	10
3.5.	Interactions With Forking	11
3.5.1.	Sequential Forking	11
3.5.2.	Parallel Forking	12
3.6.	Session Rehydration	13
4.	Interface	13
4.1.	Methods	14
4.1.1.	createOffer	14
4.1.2.	createAnswer	15
4.1.3.	SessionDescriptionType	16
4.1.3.1.	Use of Provisional Answers	16
4.1.3.2.	Rollback	17
4.1.4.	setLocalDescription	18
4.1.5.	setRemoteDescription	18
4.1.6.	localDescription	19
4.1.7.	remoteDescription	19
4.1.8.	updateIce	19
4.1.9.	addIceCandidate	19
5.	SDP Interaction Procedures	20
5.1.	Requirements Overview	20
5.1.1.	Implementation Requirements	20
5.1.2.	Usage Requirements	21
5.2.	Constructing an Offer	22
5.2.1.	Initial Offers	22
5.2.2.	Subsequent Offers	26
5.2.3.	Constraints Handling	28
5.2.3.1.	OfferToReceiveAudio	28
5.2.3.2.	OfferToReceiveVideo	28
5.2.3.3.	VoiceActivityDetection	28
5.2.3.4.	IceRestart	29
5.3.	Generating an Answer	29
5.3.1.	Initial Answers	29
5.3.2.	Subsequent Answers	33
5.3.3.	Constraints Handling	33
5.4.	Parsing an Offer	33
5.5.	Parsing an Answer	33

5.6. Applying a Local Description	33
5.7. Applying a Remote Description	33
6. Configurable SDP Parameters	33
7. Security Considerations	34
8. IANA Considerations	34
9. Acknowledgements	34
10. References	35
10.1. Normative References	35
10.2. Informative References	37
Appendix A. JSEP Implementation Examples	38
A.1. Example API Flows	38
A.1.1. Call using ROAP	38
A.1.2. Call using XMPP	39
A.1.3. Adding video to a call, using XMPP	40
A.1.4. Simultaneous add of video streams, using XMPP	41
A.1.5. Call using SIP	41
A.1.6. Handling early media (e.g. 1-800-GO FEDEX), using SIP	42
A.2. Example Session Descriptions	43
A.2.1. createOffer	43
A.2.2. createAnswer	45
A.2.3. Call Flows	46
Appendix B. Change log	46
Authors' Addresses	48

1. Introduction

This document describes how the W3C WEBRTC `RTCPeerConnection` interface[W3C.WD-webrtc-20111027] is used to control the setup, management and teardown of a multimedia session.

1.1. General Design of JSEP

The thinking behind WebRTC call setup has been to fully specify and control the media plane, but to leave the signaling plane up to the application as much as possible. The rationale is that different applications may prefer to use different protocols, such as the existing SIP or Jingle call signaling protocols, or something custom to the particular application, perhaps for a novel use case. In this approach, the key information that needs to be exchanged is the multimedia session description, which specifies the necessary transport and media configuration information necessary to establish the media plane.

The browser environment also has its own challenges that pose problems for an embedded signaling state machine. One of these is that the user may reload the web page at any time. If the browser is fully in charge of the signaling state, this will result in the loss of the call when this state is wiped by the reload. However, if the

state can be stored at the server, and pushed back down to the new page, the call can be resumed with minimal interruption.

With these considerations in mind, this document describes the Javascript Session Establishment Protocol (JSEP) that allows for full control of the signaling state machine from Javascript. This mechanism effectively removes the browser almost completely from the core signaling flow; the only interface needed is a way for the application to pass in the local and remote session descriptions negotiated by whatever signaling mechanism is used, and a way to interact with the ICE state machine.

In this document, the use of JSEP is described as if it always occurs between two browsers. Note though in many cases it will actually be between a browser and some kind of server, such as a gateway or MCU. This distinction is invisible to the browser; it just follows the instructions it is given via the API.

JSEP's handling of session descriptions is simple and straightforward. Whenever an offer/answer exchange is needed, the initiating side creates an offer by calling a `createOffer()` API. The application optionally modifies that offer, and then uses it to set up its local config via the `setLocalDescription()` API. The offer is then sent off to the remote side over its preferred signaling mechanism (e.g., WebSockets); upon receipt of that offer, the remote party installs it using the `setRemoteDescription()` API.

When the call is accepted, the callee uses the `createAnswer()` API to generate an appropriate answer, applies it using `setLocalDescription()`, and sends the answer back to the initiator over the signaling channel. When the offerer gets that answer, it installs it using `setRemoteDescription()`, and initial setup is complete. This process can be repeated for additional offer/answer exchanges.

Regarding ICE [RFC5245], JSEP decouples the ICE state machine from the overall signaling state machine, as the ICE state machine must remain in the browser, because only the browser has the necessary knowledge of candidates and other transport info. Performing this separation also provides additional flexibility; in protocols that decouple session descriptions from transport, such as Jingle, the transport information can be sent separately; in protocols that don't, such as SIP, the information can be used in the aggregated form. Sending transport information separately can allow for faster ICE and DTLS startup, since the necessary roundtrips can occur while waiting for the remote side to accept the session.

Through its abstraction of signaling, the JSEP approach does require the application to be aware of the signaling process. While the application does not need to understand the contents of session descriptions to set up a call, the application must call the right APIs at the right times, convert the session descriptions and ICE information into the defined messages of its chosen signaling protocol, and perform the reverse conversion on the messages it receives from the other side.

One way to mitigate this is to provide a Javascript library that hides this complexity from the developer; said library would implement a given signaling protocol along with its state machine and serialization code, presenting a higher level call-oriented interface to the application developer. For example, this library could easily adapt the JSEP API into the API that was proposed for the ROAP signaling protocol [I-D.jennings-rtcweb-signaling], which would perform a ROAP call setup under the covers, interacting with the application only when it needs a signaling message to be sent. In the same fashion, one could also implement other popular signaling protocols, including SIP or Jingle. This allow JSEP to provide greater control for the experienced developer without forcing any additional complexity on the novice developer.

1.2. Other Approaches Considered

One approach that was considered instead of JSEP was to include a lightweight signaling protocol. Instead of providing session descriptions to the API, the API would produce and consume messages from this protocol. While providing a more high-level API, this put more control of signaling within the browser, forcing the browser to have to understand and handle concepts like signaling glare. In addition, it prevented the application from driving the state machine to a desired state, as is needed in the page reload case.

A second approach that was considered but not chosen was to decouple the management of the media control objects from session descriptions, instead offering APIs that would control each component directly. This was rejected based on a feeling that requiring exposure of this level of complexity to the application programmer would not be beneficial; it would result in an API where even a simple example would require a significant amount of code to orchestrate all the needed interactions, as well as creating a large API surface that needed to be agreed upon and documented. In addition, these API points could be called in any order, resulting in a more complex set of interactions with the media subsystem than the JSEP approach, which specifies how session descriptions are to be evaluated and applied.

One variation on JSEP that was considered was to keep the basic session description-oriented API, but to move the mechanism for generating offers and answers out of the browser. Instead of providing createOffer/createAnswer methods within the browser, this approach would instead expose a getCapabilities API which would provide the application with the information it needed in order to generate its own session descriptions. This increases the amount of work that the application needs to do; it needs to know how to generate session descriptions from capabilities, and especially how to generate the correct answer from an arbitrary offer and the supported capabilities. While this could certainly be addressed by using a library like the one mentioned above, it basically forces the use of said library even for a simple example. Providing createOffer/createAnswer avoids this problem, but still allows applications to generate their own offers/answers (to a large extent) if they choose, using the description generated by createOffer as an indication of the browser's capabilities.

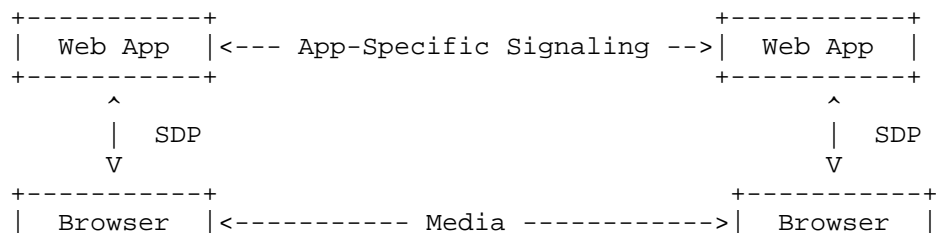
2. Terminology

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 [RFC2119].

3. Semantics and Syntax

3.1. Signaling Model

JSEP does not specify a particular signaling model or state machine, other than the generic need to exchange SDP media descriptions in the fashion described by [RFC3264] (offer/answer) in order for both sides of the session to know how to conduct the session. JSEP provides mechanisms to create offers and answers, as well as to apply them to a session. However, the browser is totally decoupled from the actual mechanism by which these offers and answers are communicated to the remote side, including addressing, retransmission, forking, and glare handling. These issues are left entirely up to the application; the application has complete control over which offers and answers get handed to the browser, and when.



+-----+

+-----+

Figure 1: JSEP Signaling Model

3.2. Session Descriptions and State Machine

In order to establish the media plane, the user agent needs specific parameters to indicate what to transmit to the remote side, as well as how to handle the media that is received. These parameters are determined by the exchange of session descriptions in offers and answers, and there are certain details to this process that must be handled in the JSEP APIs.

Whether a session description applies to the local side or the remote side affects the meaning of that description. For example, the list of codecs sent to a remote party indicates what the local side is willing to receive, which, when intersected with the set of codecs the remote side supports, specifies what the remote side should send. However, not all parameters follow this rule; for example, the SRTP parameters [RFC4568] sent to a remote party indicate what the local side will use to encrypt, and thereby what the remote party should expect to receive; the remote party will have to accept these parameters, with no option to choose a different value.

In addition, various RFCs put different conditions on the format of offers versus answers. For example, a offer may propose multiple SRTP configurations, but an answer may only contain a single SRTP configuration.

Lastly, while the exact media parameters are only known only after a offer and an answer have been exchanged, it is possible for the offerer to receive media after they have sent an offer and before they have received an answer. To properly process incoming media in this case, the offerer's media handler must be aware of the details of the offer before the answer arrives.

Therefore, in order to handle session descriptions properly, the user agent needs:

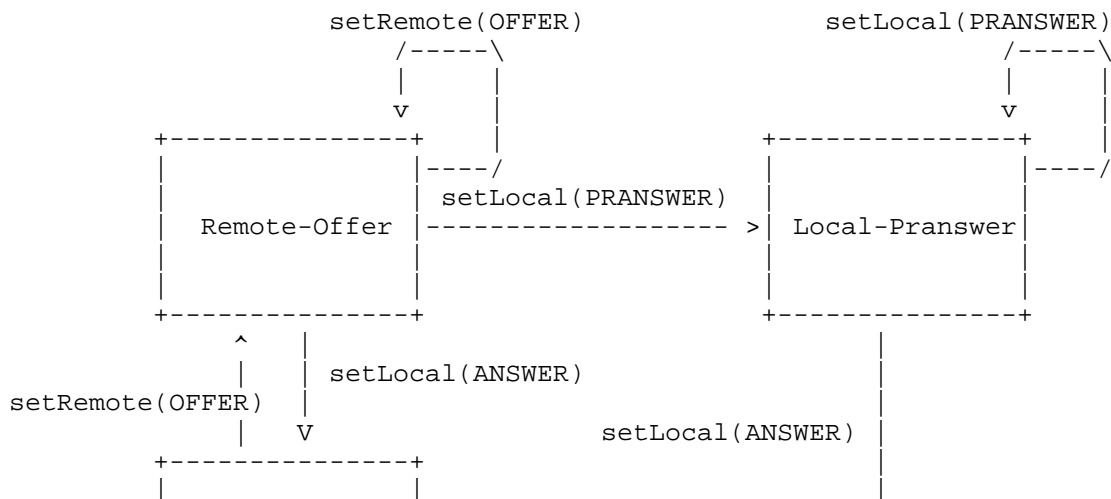
1. To know if a session description pertains to the local or remote side.
2. To know if a session description is an offer or an answer.
3. To allow the offer to be specified independently of the answer.

JSEP addresses this by adding both a `setLocalDescription` and a `setRemoteDescription` method and having session description objects

contain a type field indicating the type of session description being supplied. This satisfies the requirements listed above for both the offerer, who first calls `setLocalDescription(sdp [offer])` and then later `setRemoteDescription(sdp [answer])`, as well as for the answerer, who first calls `setRemoteDescription(sdp [offer])` and then later `setLocalDescription(sdp [answer])`.

JSEP also allows for an answer to be treated as provisional by the application. Provisional answers provide a way for an answerer to communicate initial session parameters back to the offerer, in order to allow the session to begin, while allowing a final answer to be specified later. This concept of a final answer is important to the offer/answer model; when such an answer is received, any extra resources allocated by the caller can be released, now that the exact session configuration is known. These "resources" can include things like extra ICE components, TURN candidates, or video decoders. Provisional answers, on the other hand, do no such deallocation results; as a result, multiple dissimilar provisional answers can be received and applied during call setup.

In [RFC3264], the constraint at the signaling level is that only one offer can be outstanding for a given session, but from the media stack level, a new offer can be generated at any point. For example, when using SIP for signaling, if one offer is sent, then cancelled using a SIP CANCEL, another offer can be generated even though no answer was received for the first offer. To support this, the JSEP media layer can provide an offer whenever the Javascript application needs one for the signaling. The answerer can send back zero or more provisional answers, and finally end the offer-answer exchange by sending a final answer. The state machine for this is as follows:



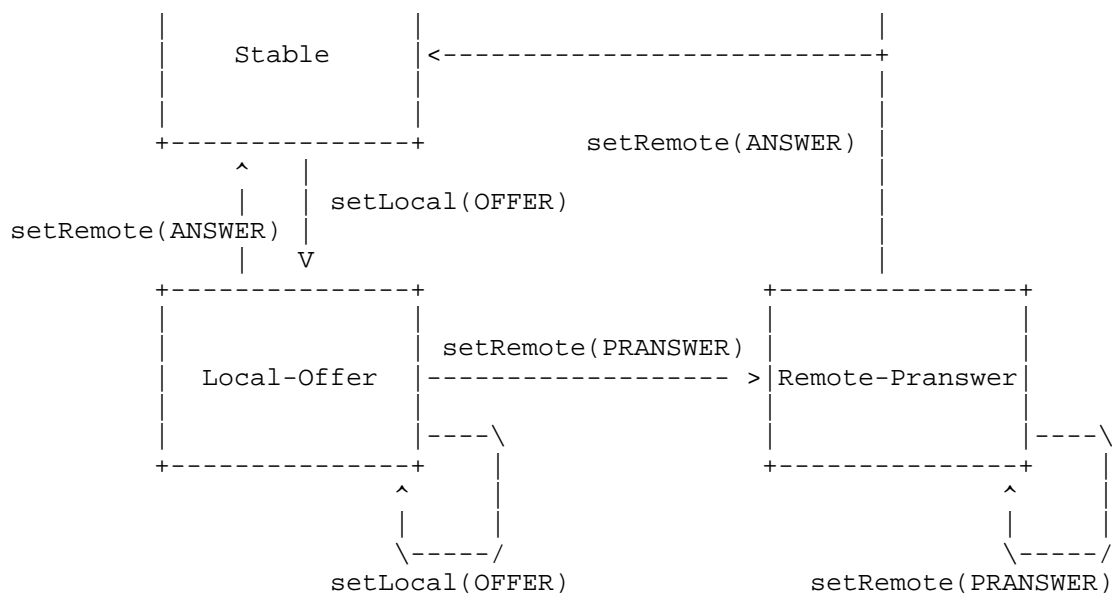


Figure 2: JSEP State Machine

Aside from these state transitions, there is no other difference between the handling of provisional ("pranswer") and final ("answer") answers.

3.3. Session Description Format

In the WebRTC specification, session descriptions are formatted as SDP messages. While this format is not optimal for manipulation from Javascript, it is widely accepted, and frequently updated with new features. Any alternate encoding of session descriptions would have to keep pace with the changes to SDP, at least until the time that this new encoding eclipsed SDP in popularity. As a result, JSEP currently uses SDP as the internal representation for its session descriptions.

However, to simplify Javascript processing, and provide for future flexibility, the SDP syntax is encapsulated within a `SessionDescription` object, which can be constructed from SDP, and be serialized out to SDP. If future specifications agree on a JSON format for session descriptions, we could easily enable this object to generate and consume that JSON.

Other methods may be added to SessionDescription in the future to simplify handling of SessionDescriptions from Javascript. In the meantime, Javascript libraries can be used to perform these manipulations.

Note that most applications should be able to treat the SessionDescriptions produced and consumed by these various API calls as opaque blobs; that is, the application will not need to read or change them. The W3C API will provide appropriate APIs to allow the application to control various session parameters, which will provide the necessary information to the browser about what sort of SessionDescription to produce.

3.4. ICE

When a new ICE candidate is available, the ICE Agent will notify the application via a callback; these candidates will automatically be added to the local session description. When all candidates have been gathered, the callback will also be invoked to signal that the gathering process is complete.

3.4.1. ICE Candidate Trickling

Candidate trickling is a technique through which a caller may incrementally provide candidates to the callee after the initial offer has been dispatched; the semantics of "Trickle ICE" are defined in [I-D.ietf-mmusic-trickle-ice]. This process allows the callee to begin acting upon the call and setting up the ICE (and perhaps DTLS) connections immediately, without having to wait for the caller to gather all possible candidates. This results in faster call startup in cases where gathering is not performed prior to initiating the call.

JSEP supports optional candidate trickling by providing APIs that provide control and feedback on the ICE candidate gathering process. Applications that support candidate trickling can send the initial offer immediately and send individual candidates when they get the notified of a new candidate; applications that do not support this feature can simply wait for the indication that gathering is complete, and then create and send their offer, with all the candidates, at this time.

Upon receipt of trickled candidates, the receiving application will supply them to its ICE Agent. This triggers the ICE Agent to start using the new remote candidates for connectivity checks.

3.4.1.1. ICE Candidate Format

As with session descriptions, the syntax of the IceCandidate object provides some abstraction, but can be easily converted to and from the SDP candidate lines.

The candidate lines are the only SDP information that is contained within IceCandidate, as they represent the only information needed that is not present in the initial offer (i.e. for trickle candidates). This information is carried with the same syntax as the "candidate-attribute" field defined for ICE. For example:

```
candidate:1 1 UDP 1694498815 192.0.2.33 10000 typ host
```

The IceCandidate object also contains fields to indicate which m-line it should be associated with. The m line can be identified in one of two ways; either by a m-line index, or a MID. The m-line index is a zero-based index, referring to the Nth m-line in the SDP. The MID uses the "media stream identification", as defined in [RFC5888], to identify the m-line. WebRTC implementations creating an ICE Candidate object MUST populate both of these fields. Implementations receiving an ICE Candidate object SHOULD use the MID if they implement that functionality, or the m-line index, if not.

3.5. Interactions With Forking

Some call signaling systems allow various types of forking where an SDP Offer may be provided to more than one device. For example, SIP [RFC3261] defines both a "Parallel Search" and "Sequential Search". Although these are primarily signaling level issues that are outside the scope of JSEP, they do have some impact on the configuration of the media plane which is relevant. When forking happens at the signaling layer, the Javascript application responsible for the signaling needs to make the decisions about what media should be sent or received at any point of time, as well as which remote endpoint it should communicate with; JSEP is used to make sure the media engine can make the RTP and media perform as required by the application. The basic operations that the applications can have the media engine do are:

Start exchanging media to a given remote peer, but keep all the resources reserved in the offer.

Start exchanging media with a given remote peer, and free any resources in the offer that are not being used.

3.5.1. Sequential Forking

Sequential forking involves a call being dispatched to multiple remote callees, where each callee can accept the call, but only one active session ever exists at a time; no mixing of received media is performed.

JSEP handles sequential forking well, allowing the application to easily control the policy for selecting the desired remote endpoint. When an answer arrives from one of the callees, the application can choose to apply it either as a provisional answer, leaving open the possibility of using a different answer in the future, or apply it as a final answer, ending the setup flow.

In a "first-one-wins" situation, the first answer will be applied as a final answer, and the application will reject any subsequent answers. In SIP parlance, this would be ACK + BYE.

In a "last-one-wins" situation, all answers would be applied as provisional answers, and any previous call leg will be terminated. At some point, the application will end the setup process, perhaps with a timer; at this point, the application could reapply the existing remote description as a final answer.

3.5.2. Parallel Forking

Parallel forking involves a call being dispatched to multiple remote callees, where each callee can accept the call, and multiple simultaneous active signaling sessions can be established as a result. If multiple callees send media at the same time, the possibilities for handling this are described in Section 3.1 of [RFC3960]. Most SIP devices today only support exchanging media with a single device at a time, and do not try to mix multiple early media audio sources, as that could result in a confusing situation. For example, consider having a European ringback tone mixed together with the North American ringback tone - the resulting sound would not be like either tone, and would confuse the user. If the signaling application wishes to only exchange media with one of the remote endpoints at a time, then from a media engine point of view, this is exactly like the sequential forking case.

In the parallel forking case where the Javascript application wishes to simultaneously exchange media with multiple peers, the flow is slightly more complex, but the Javascript application can follow the strategy that [RFC3960] describes using UPDATE. (It is worth noting that use cases where this is the desired behavior are very unusual.) The UPDATE approach allows the signaling to set up a separate media flow for each peer that it wishes to exchange media with. In JSEP, this offer used in the UPDATE would be formed by simply creating a new PeerConnection and making sure that the same local media streams

have been added into this new PeerConnection. Then the new PeerConnection object would produce a SDP offer that could be used by the signaling to perform the UPDATE strategy discussed in [RFC3960].

As a result of sharing the media streams, the application will end up with N parallel PeerConnection sessions, each with a local and remote description and their own local and remote addresses. The media flow from these sessions can be managed by specifying SDP direction attributes in the descriptions, or the application can choose to play out the media from all sessions mixed together. Of course, if the application wants to only keep a single session, it can simply terminate the sessions that it no longer needs.

3.6. Session Rehydration

In the event that the local application state is reinitialized, either due to a user reload of the page, or a decision within the application to reload itself (perhaps to update to a new version), it is possible to keep an existing session alive, via a process called "rehydration". The explicit goal of rehydration is to carry out this session resumption with no interaction with the remote side other than normal call signaling messages.

With rehydration, the current signaling state is persisted somewhere outside of the page, perhaps on the application server, or in browser local storage. The page is then reloaded, the saved signaling state is retrieved, and a new PeerConnection object is created for the session. The previously obtained MediaStreams are re-acquired, and are given the same IDs as the original session; this ensures the IDs in use by the remote side continue to work. Next, a new offer is generated by the new PeerConnection; this offer will have new ICE and possibly new DTLS-SRTP certificate fingerprints (since the old ICE and SRTP state has been lost). Finally, this offer is used to re-initiate the session with the existing remote endpoint, who simply sees the new offer as an in-call renegotiation, and replies with an answer that can be supplied to setRemoteDescription. ICE processing proceeds as usual, and as soon as connectivity is established, the session will be back up and running again.

[OPEN ISSUE: EKR proposed an alternative rehydration approach where the actual internal PeerConnection object in the browser was kept alive for some time after the web page was killed and provided some way for a new page to acquire the old PeerConnection object.]

4. Interface

This section details the basic operations that must be present to implement JSEP functionality. The actual API exposed in the W3C API

may have somewhat different syntax, but should map easily to these concepts.

4.1. Methods

4.1.1. createOffer

The createOffer method generates a blob of SDP that contains a [RFC3264] offer with the supported configurations for the session, including descriptions of the local MediaStreams attached to this PeerConnection, the codec/RTP/RTCP options supported by this implementation, and any candidates that have been gathered by the ICE Agent. A constraints parameters may be supplied to provide additional control over the generated offer. This constraints parameter should allow for the following manipulations to be performed:

- o To indicate support for a media type even if no MediaStreamTracks of that type have been added to the session (e.g., an audio call that wants to receive video.)
- o To trigger an ICE restart, for the purpose of reestablishing connectivity.
- o For re-offer cases, to request an offer that contains the full set of supported capabilities, as opposed to just the currently negotiated parameters.

In the initial offer, the generated SDP will contain all desired functionality for the session (certain parts that are supported but not desired by default may be omitted); for each SDP line, the generation of the SDP will follow the process defined for generating an initial offer from the document that specifies the given SDP line. The exact handling of initial offer generation is detailed in Section 5.2.1. below.

In the event `createOffer` is called after the session is established, `createOffer` will generate an offer to modify the current session based on any changes that have been made to the session, e.g. adding or removing `MediaStreams`, or requesting an ICE restart. For each existing stream, the generation of each SDP line must follow the process defined for generating an updated offer from the document that specifies the given SDP line. For each new stream, the generation of the SDP must follow the process of generating an initial offer, as mentioned above. If no changes have been made, or for SDP lines that are unaffected by the requested changes, the offer will only contain the parameters negotiated by the last offer-answer exchange. The exact handling of subsequent offer generation is detailed in Section 5.2.2. below.

Session descriptions generated by `createOffer` must be immediately usable by `setLocalDescription`; if a system has limited resources (e.g. a finite number of decoders), `createOffer` should return an offer that reflects the current state of the system, so that `setLocalDescription` will succeed when it attempts to acquire those resources. Because this method may need to inspect the system state to determine the currently available resources, it may be implemented as an async operation.

Calling this method may do things such as generate new ICE credentials, but does not result in candidate gathering, or cause media to start or stop flowing.

4.1.1.2. `createAnswer`

The `createAnswer` method generates a blob of SDP that contains a [RFC3264] SDP answer with the supported configuration for the session that is compatible with the parameters supplied in the offer. Like `createOffer`, the returned blob contains descriptions of the local `MediaStreams` attached to this `PeerConnection`, the codec/RTP/RTCP options negotiated for this session, and any candidates that have been gathered by the ICE Agent. A `constraints` parameter may be supplied to provide additional control over the generated answer.

As an answer, the generated SDP will contain a specific configuration that specifies how the media plane should be established; for each SDP line, the generation of the SDP must follow the process defined for generating an answer from the document that specifies the given SDP line. The exact handling of answer generation is detailed in Section 5.3. below.

Session descriptions generated by `createAnswer` must be immediately usable by `setLocalDescription`; like `createOffer`, the returned description should reflect the current state of the system. Because

this method may need to inspect the system state to determine the currently available resources, it may need to be implemented as an async operation.

Calling this method may do things such as generate new ICE credentials, but does not trigger candidate gathering or change media state.

4.1.3. SessionDescriptionType

Session description objects (RTCSessionDescription) may be of type "offer", "pranswer", and "answer". These types provide information as to how the description parameter should be parsed, and how the media state should be changed.

"offer" indicates that a description should be parsed as an offer; said description may include many possible media configurations. A description used as an "offer" may be applied anytime the PeerConnection is in a stable state, or as an update to a previously supplied but unanswered "offer".

"pranswer" indicates that a description should be parsed as an answer, but not a final answer, and so should not result in the freeing of allocated resources. It may result in the start of media transmission, if the answer does not specify an inactive media direction. A description used as a "pranswer" may be applied as a response to an "offer", or an update to a previously sent "answer".

"answer" indicates that a description should be parsed as an answer, the offer-answer exchange should be considered complete, and any resources (decoders, candidates) that are no longer needed can be released. A description used as an "answer" may be applied as a response to a "offer", or an update to a previously sent "pranswer".

The only difference between a provisional and final answer is that the final answer results in the freeing of any unused resources that were allocated as a result of the offer. As such, the application can use some discretion on whether an answer should be applied as provisional or final, and can change the type of the session description as needed. For example, in a serial forking scenario, an application may receive multiple "final" answers, one from each remote endpoint. The application could choose to accept the initial answers as provisional answers, and only apply an answer as final when it receives one that meets its criteria (e.g. a live user instead of voicemail).

4.1.3.1. Use of Provisional Answers

Most web applications will not need to create answers using the "pranswer" type. The preferred handling for a web application would be to create and send an "inactive" answer more or less immediately after receiving the offer, instead of waiting for a human user to physically answer the call. Later, when the human input is received, the application can create a new "sendrecv" offer to update the previous offer/answer pair and start the media flow. This approach is preferred because it minimizes the amount of time that the offer-answer exchange is left open, in addition to avoiding media clipping by ensuring the transport is ready to go by the time the call is physically answered. However, some applications may not be able to do this, particularly ones that are attempting to gateway to other signaling protocols. In these cases, "pranswer" can still allow the application to warm up the transport.

Consider a typical web application that will set up a data channel, an audio channel, and a video channel. When an endpoint receives an offer with these channels, it could send an answer accepting the data channel for two-way data, and accepting the audio and video tracks as inactive or receive-only. It could then ask the user to accept the call, acquire the local media streams, and send a new offer to the remote side moving the audio and video to be two-way media. By the time the human has accepted the call and sent the new offer, it is likely that the ICE and DTLS handshaking for all the channels will already be set up.

4.1.3.2. Rollback

In certain situations it may be desirable to "undo" a change made to `setLocalDescription` or `setRemoteDescription`. Consider a case where a call is ongoing, and one side wants to change some of the session parameters; that side generates an updated offer and then calls `setLocalDescription`. However, the remote side, either before or after `setRemoteDescription`, decides it does not want to accept the new parameters, and sends a reject message back to the offerer. Now, the offerer, and possibly the answerer as well, need to return to a stable state and the previous local/remote description. To support this, we introduce the concept of "rollback".

A rollback returns the state machine to its previous state, and the local or remote description to its previous value. Any resources or candidates that were allocated by the new local description are discarded; any media that is received will be processed according to the previous session description.

A rollback is performed by supplying a session description of type "rollback" to either `setLocalDescription` or `setRemoteDescription`, depending on which needs to be rolled back (i.e. if the new offer was

supplied to `setLocalDescription`, the rollback should be done on `setLocalDescription` as well.)

4.1.4. `setLocalDescription`

The `setLocalDescription` method instructs the `PeerConnection` to apply the supplied SDP blob as its local configuration. The type field indicates whether the blob should be processed as an offer, provisional answer, or final answer; offers and answers are checked differently, using the various rules that exist for each SDP line.

This API changes the local media state; among other things, it sets up local resources for receiving and decoding media. In order to successfully handle scenarios where the application wants to offer to change from one media format to a different, incompatible format, the `PeerConnection` must be able to simultaneously support use of both the old and new local descriptions (e.g. support codecs that exist in both descriptions) until a final answer is received, at which point the `PeerConnection` can fully adopt the new local description, or roll back to the old description if the remote side denied the change.

This API indirectly controls the candidate gathering process. When a local description is supplied, and the number of transports currently in use does not match the number of transports needed by the local description, the `PeerConnection` will create transports as needed and begin gathering candidates for them.

If `setRemoteDescription` was previously called with an offer, and `setLocalDescription` is called with an answer (provisional or final), and the media directions are compatible, and media are available to send, this will result in the starting of media transmission.

4.1.5. `setRemoteDescription`

The `setRemoteDescription` method instructs the `PeerConnection` to apply the supplied SDP blob as the desired remote configuration. As in `setLocalDescription`, the type field of the indicates how the blob should be processed.

This API changes the local media state; among other things, it sets up local resources for sending and encoding media.

If `setRemoteDescription` was previously called with an offer, and `setLocalDescription` is called with an answer (provisional or final), and the media directions are compatible, and media are available to send, this will result in the starting of media transmission.

4.1.6. localDescription

The localDescription method returns a copy of the current local configuration, i.e. what was most recently passed to setLocalDescription, plus any local candidates that have been generated by the ICE Agent.

TODO: Do we need to expose accessors for both the current and proposed local description?

A null object will be returned if the local description has not yet been established, or if the PeerConnection has been closed.

4.1.7. remoteDescription

The remoteDescription method returns a copy of the current remote configuration, i.e. what was most recently passed to setRemoteDescription, plus any remote candidates that have been supplied via processIceMessage.

TODO: Do we need to expose accessors for both the current and proposed remote description?

A null object will be returned if the remote description has not yet been established, or if the PeerConnection has been closed.

4.1.8. updateIce

The updateIce method allows the configuration of the ICE Agent to be changed during the session, primarily for changing which types of local candidates are provided to the application and used for connectivity checks. A callee may initially configure the ICE Agent to use only relay candidates, to avoid leaking location information, but update this configuration to use all candidates once the call is accepted.

Regardless of the configuration, the gathering process collects all available candidates, but excluded candidates will not be surfaced in onIceCandidate callback or used for connectivity checks.

This call may result in a change to the state of the ICE Agent, and may result in a change to media state if it results in connectivity being established.

4.1.9. addIceCandidate

The addIceCandidate method provides a remote candidate to the ICE Agent, which, if parsed successfully, will be added to the remote

description according to the rules defined for Trickle ICE. Connectivity checks will be sent to the new candidate.

This call will result in a change to the state of the ICE Agent, and may result in a change to media state if it results in connectivity being established.

5. SDP Interaction Procedures

This section describes the specific procedures to be followed when creating and parsing SDP objects.

5.1. Requirements Overview

JSEP implementations must comply with the specifications listed below that govern the creation and processing of offers and answers.

The first set of specifications is the "mandatory-to-implement" set. All implementations must support these behaviors, but may not use all of them if the remote side, which may not be a JSEP endpoint, does not support them.

The second set of specifications is the "mandatory-to-use" set. The local JSEP endpoint and any remote endpoint must indicate support for these specifications in their session descriptions.

5.1.1. Implementation Requirements

This list of mandatory-to-implement specifications is derived from the requirements outlined in [I-D.ietf-rtcweb-rtp-usage].

R-1 [RFC4566] is the base SDP specification and MUST be implemented.

R-2 [RFC5764] MUST be supported for signaling the UDP/TLS/RTP/SAVPF RTP profile.

R-3 [RFC5245] MUST be implemented for signaling the ICE credentials and candidate lines corresponding to each media stream. The ICE implementation MUST be a Full implementation, not a Lite implementation.

R-4 [RFC5763] MUST be implemented to signal DTLS certificate fingerprints.

R-5 [RFC4568] MUST NOT be implemented to signal SDES SRTP keying information.

- R-6 The [RFC5888] grouping framework MUST be implemented for signaling grouping information, and MUST be used to identify m= lines via the a=mid attribute.
- R-7 [I-D.ietf-mmusic-msid] MUST be supported, in order to signal associations between RTP objects and W3C MediaStreams and MediaStreamTracks in a standard way.
- R-8 The bundle mechanism in [I-D.ietf-mmusic-sdp-bundle-negotiation] MUST be supported to signal the ability to multiplex RTP streams on a single UDP port, in order to avoid excessive use of port number resources.
- R-9 The SDP attributes of "sendonly", "recvonly", "inactive", and "sendrecv" from [RFC4566] MUST be implemented to signal information about media direction.
- R-10 [RFC5576] MUST be implemented to signal RTP SSRC values.
- R-11 [RFC4585] MUST be implemented to signal RTCP based feedback.
- R-12 [RFC5761] MUST be implemented to signal multiplexing of RTP and RTCP.
- R-13 [RFC5506] MUST be implemented to signal reduced-size RTCP messages.
- R-14 [RFC3556] with bandwidth modifiers MAY be supported for specifying RTCP bandwidth as a fraction of the media bandwidth, RTCP fraction allocated to the senders and setting maximum media bit-rate boundaries.

As required by [RFC4566], Section 5.13, JSEP implementations MUST ignore unknown attribute (a=) lines.

5.1.2. Usage Requirements

All session descriptions handled by JSEP endpoints, both local and remote, MUST indicate support for the following specifications. If any of these are absent, this omission MUST be treated as an error.

- R-1 Either the UDP/TLS/RTP/SAVP or the UDP/TLS/RTP/SAVPF RTP profile, as specified in [RFC5764], MUST be used.
- R-2 ICE, as specified in [RFC5245], MUST be used. Note that the remote endpoint MAY use a Lite implementation.
- R-3 DTLS-SRTP, as specified in [RFC5763], MUST be used.

5.2. Constructing an Offer

When `createOffer` is called, a new SDP description must be created that includes the functionality specified in [I-D.ietf-rtcweb-rtp-usage]. The exact details of this process are explained below.

5.2.1. Initial Offers

When `createOffer` is called for the first time, the result is known as the initial offer.

The first step in generating an initial offer is to generate session-level attributes, as specified in [RFC4566], Section 5. Specifically:

- o The first SDP line MUST be "v=0", as specified in [RFC4566], Section 5.1
- o The second SDP line MUST be an "o=" line, as specified in [RFC4566], Section 5.2. The value of the <username> field SHOULD be "-". The value of the <sess-id> field SHOULD be a cryptographically random number. To ensure uniqueness, this number SHOULD be at least 64 bits long. The value of the <sess-version> field SHOULD be zero. The value of the <nettype> <addrtype> <unicast-address> tuple SHOULD be set to a non-meaningful address, such as IN IP4 0.0.0.0, to prevent leaking the local address in this field. As mentioned in [RFC4566], the entire o= line needs to be unique, but selecting a random number for <sess-id> is sufficient to accomplish this.
- o The third SDP line MUST be a "s=" line, as specified in [RFC4566], Section 5.3; to match the "o=" line, a single dash SHOULD be used as the session name, e.g. "s=-".
- o Session Information ("i="), URI ("u="), Email Address ("e="), Phone Number ("p="), Bandwidth ("b="), Repeat Times ("r="), and Time Zones ("z=") lines are not useful in this context and SHOULD NOT be included.
- o Encryption Keys ("k=") lines do not provide sufficient security and MUST NOT be included.
- o A "t=" line MUST be added, as specified in [RFC4566], Section 5.9; both <start-time> and <stop-time> SHOULD be set to zero, e.g. "t=0 0".

- o An "a=msid-semantic:WMS" line MUST be added, as specified in [I-D.ietf-mmusic-msid], Section 4.

The next step is to generate m= sections for each MediaStreamTrack that has been added to the PeerConnection via the addStream method. (Note that this method takes a MediaStream, which can contain multiple MediaStreamTracks, and therefore multiple m= sections can be generated even if addStream is only called once.) When generating m= sections, the ordering is based on (1) the order in which the MediaStreams were added to the PeerConnection, and (2) the alphabetical ordering of the media type for the MediaStreamTrack. For example, if a MediaStream containing both an audio and a video MediaStreamTrack is added to a PeerConnection, the resultant m=audio section will precede the m=video section.

Each m= section, provided it is not being bundled into another m= section, MUST generate a unique set of ICE credentials and gather its own unique set of ICE candidates. Otherwise, it MUST use the same ICE credentials and candidates that were used in the m= section that it is being bundled into.

For DTLS, all m= sections MUST use the certificate for the identity that has been specified for the PeerConnection; as a result, they MUST all have the same [RFC4572]fingerprint value.

Each m= section should be generated as specified in [RFC4566], Section 5.14. For the m= line itself, the following rules MUST be followed:

- o The port value is set to the port of the default ICE candidate for this m= section; if this m= section is not being bundled into another m= section, the port value MUST be unique. If no candidates have yet been gathered, and a 'null' port value is being used, as indicated in [I-D.ietf-mmusic-trickle-ice], Section 5.1., this port MUST still be unique.
- o To properly indicate use of DTLS, the <proto> field MUST be set to "UDP/TLS/RTP/SAVPF", as specified in [RFC5764], Section 8.

Each m= section MUST include the following attribute lines:

- o An "a=mid" line, as specified in [RFC5888], Section 4.
- o An "a=msid" line, as specified in [I-D.ietf-mmusic-msid], Section 2.
- o [OPEN ISSUE: Use of App Token versus stream-correlator]

- o An "a=sendrecv" line, as specified in [RFC3264], Section 5.1.
- o For each supported codec, "a=rtpmap" and "a=fmtp" lines, as specified in [RFC4566], Section 6. For audio, the codecs specified in [I-D.ietf-rtcweb-audio], Section 3, MUST be supported.
- o For each primary codec where RTP retransmission should be used, a corresponding "a=rtpmap" line indicating "rtx" with the clock rate of the primary codec and an "a=fmtp" line that references the payload type for the primary codec, as specified in [RFC4588], Section 8.1.
- o For each supported FEC mechanism, a corresponding "a=rtpmap" line indicating the desired FEC codec.
- o "a=ice-ufrag" and "a=ice-passwd" lines, as specified in [RFC5245], Section 15.4.
- o An "a=ice-options" line, with the "trickle" option, as specified in [I-D.ietf-mmusic-trickle-ice], Section 4.
- o For each candidate that has been gathered during the most recent gathering phase, an "a=candidate" line, as specified in [RFC5245], Section 4.3., paragraph 3.
- o For the current default candidate, a "c=" line, as specific in [RFC5245], Section 4.3., paragraph 6. If no candidates have been gathered yet, the default candidate should be set to the 'null' value defined in [I-D.ietf-mmusic-trickle-ice], Section 5.1.
- o An "a=fingerprint" line, as specified in [RFC4572], Section 5; the algorithm used for the fingerprint MUST match that used in the certificate signature.
- o An "a=setup" line, as specified in [RFC4145], Section 4, and clarified for use in DTLS-SRTP scenarios in [RFC5763], Section 5. The role value in the offer MUST be "actpass".
- o An "a=rtcp-mux" line, as specified in [RFC5761], Section 5.1.1.
- o An "a=rtcp-rsize" line, as specified in [RFC5506], Section 5.

- o For each supported RTP header extension, an "a=extmap" line, as specified in [RFC5285], Section 5. The list of header extensions that SHOULD/MUST be supported is specified in [I-D.ietf-rtcweb-rtp-usage], Section 5.2. Any header extensions that require encryption MUST be specified as indicated in [RFC6904], Section 4.
- o For each supported RTCP feedback mechanism, an "a=rtcp-fb" mechanism, as specified in [RFC4585], Section 4.2. The list of RTCP feedback mechanisms that SHOULD/MUST be supported is specified in [I-D.ietf-rtcweb-rtp-usage], Section 5.1.
- o An "a=ssrc" line, as specified in [RFC5576], Section 4.1, indicating the SSRC to be used for sending media, along with the mandatory "cname" source attribute, as specified in Section 6.1, indicating the CNAME for the source. The CNAME must be generated in accordance with draft-rescorla-random-cname-00. [OPEN ISSUE: How are CNAMEs specified for MSTs? Are they randomly generated for each MediaStream? If so, can two MediaStreams be synced?]
- o If RTX is supported for this media type, another "a=ssrc" line with the RTX SSRC, and an "a=ssrc-group" line, as specified in [RFC5576], section 4.2, with semantics set to "FID" and including the primary and RTX SSRCs.
- o If FEC is supported for this media type, another "a=ssrc" line with the FEC SSRC, and an "a=ssrc-group" line, as specified in [RFC5576], section 4.2, with semantics set to "FEC" and including the primary and FEC SSRCs.
- o [OPEN ISSUE: Handling of a=imageattr]
- o [TODO: bundle-only]

Lastly, if a data channel has been created, a m= section MUST be generated for data. The <media> field MUST be set to "application" and the <proto> field MUST be set to "DTLS/SCTP", as specified in [I-D.ietf-mmusic-sctp-sdp], Section 3; the "fmt" value MUST be set to the SCTP port number, as specified in Section 4.1.

Within the data m= section, the "a=mid", "a=ice-ufrag", "a=ice-passwd", "a=ice-options", "a=candidate", "a=fingerprint", and "a=setup" lines MUST be included as mentioned above, along with an "a=sctpmap" line referencing the SCTP port number and specifying the application protocol indicated in [I-D.ietf-rtcweb-data-protocol]. [OPEN ISSUE: the -00 of this document is missing this information.]

Once all m= sections have been generated, a session-level "a=group" attribute MUST be added as specified in [RFC5888]. This attribute MUST have semantics "BUNDLE", and identify the m= sections to be bundled. [OPEN ISSUE: Need to determine exactly how this decision is made.]

Attributes that are common between all m= sections MAY be moved to session-level, if desired.

Attributes other than the ones specified above MAY be included, except for the following attributes which are specifically incompatible with the requirements of [I-D.ietf-rtcweb-rtp-usage], and MUST NOT be included:

- o "a=crypto"
- o "a=key-mgmt"
- o "a=ice-lite"

Note that when BUNDLE is used, any additional attributes that are added MUST follow the advice in [I-D.nandakumar-mmusic-sdp-mux-attributes] on how those attributes interact with BUNDLE.

5.2.2. Subsequent Offers

When createOffer is called a second (or later) time, or is called after a local description has already been installed, the processing is somewhat different than for an initial offer.

If the initial offer was not applied using setLocalDescription, meaning the PeerConnection is still in the "stable" state, the steps for generating an initial offer should be followed, subject to the following restriction:

- o The fields of the "o=" line MUST stay the same except for the <session-version> field, which MUST increment if the session description changes in any way, including the addition of ICE candidates.

If the initial offer was applied using setLocalDescription, but an answer from the remote side has not yet been applied, meaning the PeerConnection is still in the "local-offer" state, an offer is generated by following the steps in the "stable" state above, along with these exceptions:

- o The "s=" and "t=" lines MUST stay the same.

- o Each "a=mid" line MUST stay the same.
- o Each "a=ice-ufrag" and "a=ice-pwd" line MUST stay the same.
- o For MediaStreamTracks that are still present, the "a=msid", "a=ssrc", and "a=ssrc-group" lines MUST stay the same.
- o If any MediaStreamTracks have been removed, either through the removeStream method or by removing them from an added MediaStream, their m= sections MUST be marked as recvonly by changing the value of the [RFC3264] directional attribute to "a=recvonly". The "a=msid", "a=ssrc", and "a=ssrc-group" lines MUST be removed from the associated m= sections.

If the initial offer was applied using setLocalDescription, and an answer from the remote side has been applied using setRemoteDescription, meaning the PeerConnection is in the "remote-pranswer" or "stable" states, an offer is generated based on the negotiated session descriptions by following the steps mentioned for the "local-offer" state above, along with these exceptions: [OPEN ISSUE: should this be permitted in the remote-pranswer state?]

- o If a m= section was rejected, i.e. has had its port set to zero in either the local or remote description, it MUST remain rejected and have a zero port in the new offer, as indicated in RFC3264, Section 5.1.
- o If a m= section exists in the current local description, but has its state set to inactive or recvonly, and a new MediaStreamTrack is added, the previously existing m= section MUST be recycled instead of creating a new m= section. [OPEN ISSUE: Nail down exactly what this means. Should the codecs remain the same? (No.) Should ICE restart? (No.) Can the "a=mid" attribute be changed? (Yes?)]
- o If a m= section exists in the current local description, but does not have an associated MediaStreamTrack (i.e. it is inactive or recvonly), a corresponding m= section MUST be generated in the new offer, but without "a=msid", "a=ssrc", or "a=ssrc-group" attributes, and the appropriate directional attribute must be specified.

In addition, for each previously existing, non-rejected m= section in the new offer, the following adjustments are made based on the contents of the corresponding m= section in the current remote description:

- o The m= line and corresponding "a=rtpmap" and "a=fmtp" lines MUST only include codecs present in the remote description.
- o The RTP header extensions MUST only include those that are present in the remote description.
- o The RTCP feedback extensions MUST only include those that are present in the remote description.
- o The "a=rtcp-mux" line MUST only be added if present in the remote description.
- o The "a=rtcp-rsize" line MUST only be added if present in the remote description.

5.2.3. Constraints Handling

The `createOffer` method takes as a parameter a `MediaConstraints` object. Special processing is performed when generating a SDP description if the following constraints are present.

5.2.3.1. OfferToReceiveAudio

If the "OfferToReceiveAudio" constraint is specified, with a value of "true", the offer MUST include a non-rejected m= section with media type "audio", even if no audio `MediaStreamTrack` has been added to the `PeerConnection`. This allows the offerer to receive audio even when not sending it; accordingly, the directional attribute on the audio m= section MUST be set to `recvonly`. If this constraint is specified when an audio `MediaStreamTrack` has already been added to the `PeerConnection`, or a non-rejected m= section with media type "audio" previously existed, it has no effect.

5.2.3.2. OfferToReceiveVideo

If the "OfferToReceiveAudio" constraint is specified, with a value of "true", the offer MUST include a m= section with media type "video", even if no video `MediaStreamTrack` has been added to the `PeerConnection`. This allows the offerer to receive video even when not sending it; accordingly, the directional attribute on the video m= section MUST be set to `recvonly`. If this constraint is specified when an video `MediaStreamTrack` has already been added to the `PeerConnection`, or a non-rejected m= section with media type "video" previously existed, it has no effect.

5.2.3.3. VoiceActivityDetection

If the "VoiceActivityDetection" constraint is specified, with a value of "true", the offer MUST indicate support for silence suppression by including comfort noise ("CN") codecs for each supported clock rate, as specified in [RFC3389], Section 5.1.

5.2.3.4. IceRestart

If the "IceRestart" constraint is specified, with a value of "true", the offer MUST indicate an ICE restart by generating new ICE ufrag and pwd attributes, as specified in RFC5245, Section 9.1.1.1. If this constraint is specified on an initial offer, it has no effect (since a new ICE ufrag and pwd are already generated).

5.3. Generating an Answer

When createAnswer is called, a new SDP description must be created that is compatible with the supplied remote description as well as the requirements specified in [I-D.ietf-rtcweb-rtp-usage]. The exact details of this process are explained below.

5.3.1. Initial Answers

When createAnswer is called for the first time after a remote description has been provided, the result is known as the initial answer. If no remote description has been installed, an answer cannot be generated, and an error MUST be returned.

Note that the remote description SDP may not have been created by a JSEP endpoint and may not conform to all the requirements listed in Section 5.2. For many cases, this is not a problem. However, if any mandatory SDP attributes are missing, or functionality listed as mandatory-to-use above is not present, this MUST be treated as an error, and MUST cause the affected m= sections to be marked as rejected.

The first step in generating an initial answer is to generate session-level attributes. The process here is identical to that indicated in the Initial Offers section above.

The next step is to generate m= sections for each m= section that is present in the remote offer, as specified in [RFC3264], Section 6. For the purposes of this discussion, any session-level attributes in the offer that are also valid as media-level attributes SHALL be considered to be present in each m= section.

If any of the offered m= sections have been rejected, by stopping the associated remote MediaStreamTrack, the corresponding m= section in the answer MUST be marked as rejected by setting the port in the m=

line to zero, as indicated in [RFC3264], Section 6., and processing continues with the next m= section.

For each non-rejected m= section of a given media type, if there is a local `MediaStreamTrack` of the specified type which has been added to the `PeerConnection` via `addStream` and not yet associated with a m= section, the `MediaStreamTrack` is associated with the m= section at this time. If there are more m= sections of a certain type than `MediaStreamTracks`, some m= sections will not have an associated `MediaStreamTrack`. If there are more `MediaStreamTracks` of a certain type than m= sections, only the first N `MediaStreamTracks` will be able to be associated in the constructed answer. The remainder will need to be associated in a subsequent offer.

Each m= section should then generated as specified in [RFC3264], Section 6.1. Because use of DTLS is mandatory, the <proto> field MUST be set to "UDP/TLS/RTP/SAVPF". If the offer supports BUNDLE, all m= sections to be BUNDLED must use the same ICE credentials and candidates; all m= sections not being BUNDLED must use unique ICE credentials and candidates. Each m= section MUST include the following:

- o If present in the offer, an "a=mid" line, as specified in [RFC5888], Section 9.1. The "mid" value MUST match that specified in the offer.
- o If a local `MediaStreamTrack` has been associated, an "a=msid" line, as specified in [I-D.ietf-mmusic-msid], Section 2.
- o [OPEN ISSUE: Use of App Token versus stream-correlator]
- o If a local `MediaStreamTrack` has been associated, an "a=sendrecv" line, as specified in [RFC3264], Section 6.1. If no local `MediaStreamTrack` has been associated, an "a=recvonly" line. [TODO: handle non-sendrecv offered m= sections]
- o For each supported codec that is present in the offer, "a=rtpmap" and "a=fmtp" lines, as specified in [RFC4566], Section 6, and [RFC3264], Section 6.1. For audio, the codecs specified in [I-D.ietf-rtcweb-audio], Section 3, MUST be supported. Note that for simplicity, the answerer MAY use different payload types for codecs than the offerer, as it is not prohibited by Section 6.1.

- o If "rtx" is present in the offer, for each primary codec where RTP retransmission should be used, a corresponding "a=rtpmap" line indicating "rtx" with the clock rate of the primary codec and an "a=fmtp" line that references the payload type for the primary codec, as specified in [RFC4588], Section 8.1.
- o For each supported FEC mechanism that is present in the offer, a corresponding "a=rtpmap" line indicating the desired FEC codec.
- o "a=ice-ufrag" and "a=ice-passwd" lines, as specified in [RFC5245], Section 15.4.
- o If the "trickle" ICE option is present in the offer, an "a=ice-options" line, with the "trickle" option, as specified in [I-D.ietf-mmusic-trickle-ice], Section 4.
- o For each candidate that has been gathered during the most recent gathering phase, an "a=candidate" line, as specified in [RFC5245], Section 4.3., paragraph 3.
- o For the current default candidate, a "c=" line, as specific in [RFC5245], Section 4.3., paragraph 6. If no candidates have been gathered yet, the default candidate should be set to the 'null' value defined in [I-D.ietf-mmusic-trickle-ice], Section 5.1.
- o An "a=fingerprint" line, as specified in [RFC4572], Section 5; the algorithm used for the fingerprint MUST match that used in the certificate signature.
- o An "a=setup" line, as specified in [RFC4145], Section 4, and clarified for use in DTLS-SRTP scenarios in [RFC5763], Section 5. The role value in the answer MUST be "active" or "passive"; the "active" role is RECOMMENDED.
- o If present in the offer, an "a=rtcp-mux" line, as specified in [RFC5761], Section 5.1.1.
- o If present in the offer, an "a=rtcp-rsize" line, as specified in [RFC5506], Section 5.
- o For each supported RTP header extension that is present in the offer, an "a=extmap" line, as specified in [RFC5285], Section 5. The list of header extensions that SHOULD/MUST be supported is specified in [I-D.ietf-rtcweb-rtp-usage], Section 5.2. Any header extensions that require encryption MUST be specified as indicated in [RFC6904], Section 4.

- o For each supported RTCP feedback mechanism that is present in the offer, an "a=rtcp-fb" mechanism, as specified in [RFC4585], Section 4.2. The list of RTCP feedback mechanisms that SHOULD/MUST be supported is specified in [I-D.ietf-rtcweb-rtp-usage], Section 5.1.
- o If a local MediaStreamTrack has been associated, an "a=ssrc" line, as specified in [RFC5576], Section 4.1, indicating the SSRC to be used for sending media.
- o If a local MediaStreamTrack has been associated, and RTX has been negotiated for this m= section, another "a=ssrc" line with the RTX SSRC, and an "a=ssrc-group" line, as specified in [RFC5576], section 4.2, with semantics set to "FID" and including the primary and RTX SSRCs.
- o If a local MediaStreamTrack has been associated, and FEC has been negotiated for this m= section, another "a=ssrc" line with the FEC SSRC, and an "a=ssrc-group" line, as specified in [RFC5576], section 4.2, with semantics set to "FEC" and including the primary and FEC SSRCs.
- o [OPEN ISSUE: Handling of a=imageattr]
- o [TODO: bundle-only]

If a data channel m= section has been offered, a m= section MUST also be generated for data. The <media> field MUST be set to "application" and the <proto> field MUST be set to "DTLS/SCTP", as specified in [I-D.ietf-mmusic-sctp-sdp], Section 3; the "fmt" value MUST be set to the SCTP port number, as specified in Section 4.1.

Within the data m= section, the "a=mid", "a=ice-ufrag", "a=ice-passwd", "a=ice-options", "a=candidate", "a=fingerprint", and "a=setup" lines MUST be included as mentioned above, along with an "a=sctpmap" line referencing the SCTP port number and specifying the application protocol indicated in [I-D.ietf-rtcweb-data-protocol]. [OPEN ISSUE: the -00 of this document is missing this information.]

[TODO: processing of BUNDLE group]

Attributes that are common between all m= sections MAY be moved to session-level, if desired.

The attributes prohibited in the creation of offers are also prohibited in the creation of answers.

- 5.3.2. Subsequent Answers
- 5.3.3. Constraints Handling
- 5.4. Parsing an Offer
- 5.5. Parsing an Answer
- 5.6. Applying a Local Description
- 5.7. Applying a Remote Description
- 6. Configurable SDP Parameters

Note: This section is still very early and is likely to significantly change as we get a better understanding of a) the use cases for this b) the implications at the protocol level c) feedback from implementors on what they can do.

The following elements of the SDP media description MUST NOT be changed between the createOffer and the setLocalDescription, since they reflect transport attributes that are solely under browser control, and the browser MUST NOT honor an attempt to change them:

- o The number, type and port number of m-lines.
- o The generated ICE credentials (a=ice-ufrag and a=ice-pwd).
- o The set of ICE candidates and their parameters (a=candidate).

The following modifications, if done by the browser to a description between createOffer/createAnswer and the setLocalDescription, MUST be honored by the browser:

- o Remove or reorder codecs (m=)

The following parameters may be controlled by constraints passed into createOffer/createAnswer. As an open issue, these changes may also be performed by manipulating the SDP returned from createOffer/createAnswer, as indicated above, as long as the capabilities of the endpoint are not exceeded (e.g. asking for a resolution greater than what the endpoint can encode):

- o disable BUNDLE (a=group)
- o disable RTCP mux (a=rtcp-mux)
- o change send resolution or frame rate

- o change desired recv resolution or frame rate
- o change maximum total bandwidth (b=) [OPEN ISSUE: need to clarify if this is CT or AS - see section 5.8 of [RFC4566]]
- o remove desired AVPF mechanisms (a=rtcp-fb)
- o remove RTP header extensions (a=extmap)
- o change media send/recv state (a=sendonly/recvonly/inactive)

For example, an application could implement call hold by adding an a=inactive attribute to its local description, and then applying and signaling that description.

The application can also modify the SDP to reduce the capabilities in the offer it sends to the far side in any way the application sees fit, as long as it is a valid SDP offer and specifies a subset of what the browser is expecting to do.

As always, the application is solely responsible for what it sends to the other party, and all incoming SDP will be processed by the browser to the extent of its capabilities. It is an error to assume that all SDP is well-formed; however, one should be able to assume that any implementation of this specification will be able to process, as a remote offer or answer, unmodified SDP coming from any other implementation of this specification.

7. Security Considerations

The intent of the WebRTC protocol suite is to provide an environment that is securable by default: all media is encrypted, keys are exchanged in a secure fashion, and the Javascript API includes functions that can be used to verify the identity of communication partners.

8. IANA Considerations

This document requires no actions from IANA.

9. Acknowledgements

Significant text incorporated in the draft as well and review was provided by Harald Alvestrand and Suhas Nandakumar. Dan Burnett, Neil Stratford, Eric Rescorla, Anant Narayanan, Andrew Hutton, Richard Ejzak, and Adam Bergkvist all provided valuable feedback on this proposal. Matthew Kaufman provided the observation that keeping state out of the browser allows a call to continue even if the page is reloaded.

10. References

10.1. Normative References

- [I-D.ietf-mmusic-msid]
Alvestrand, H., "Cross Session Stream Identification in the Session Description Protocol", draft-ietf-mmusic-msid-01 (work in progress), August 2013.
- [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-04 (work in progress), June 2013.
- [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-04 (work in progress), June 2013.
- [I-D.ietf-rtcweb-audio]
Valin, J. and C. Bran, "WebRTC Audio Codec and Processing Requirements", draft-ietf-rtcweb-audio-02 (work in progress), August 2013.
- [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-09 (work in progress), September 2013.
- [I-D.nandakumar-mmusic-sdp-mux-attributes]
Nandakumar, S., "A Framework for SDP Attributes when Multiplexing", draft-nandakumar-mmusic-sdp-mux-attributes-03 (work in progress), July 2013.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

- [RFC3261] Rosenberg, J., Schulzrinne, H., Camarillo, G., Johnston, A., Peterson, J., Sparks, R., Handley, M., and E. Schooler, "SIP: Session Initiation Protocol", RFC 3261, June 2002.
- [RFC3264] Rosenberg, J. and H. Schulzrinne, "An Offer/Answer Model with Session Description Protocol (SDP)", RFC 3264, June 2002.
- [RFC4145] Yon, D. and G. Camarillo, "TCP-Based Media Transport in the Session Description Protocol (SDP)", RFC 4145, September 2005.
- [RFC4566] Handley, M., Jacobson, V., and C. Perkins, "SDP: Session Description Protocol", RFC 4566, July 2006.
- [RFC4572] Lennox, J., "Connection-Oriented Media Transport over the Transport Layer Security (TLS) Protocol in the Session Description Protocol (SDP)", RFC 4572, July 2006.
- [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.
- [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.
- [RFC5245] Rosenberg, J., "Interactive Connectivity Establishment (ICE): A Protocol for Network Address Translator (NAT) Traversal for Offer/Answer Protocols", RFC 5245, April 2010.
- [RFC5285] Singer, D. and H. Desineni, "A General Mechanism for RTP Header Extensions", RFC 5285, July 2008.
- [RFC5761] Perkins, C. and M. Westerlund, "Multiplexing RTP Data and Control Packets on a Single Port", RFC 5761, April 2010.
- [RFC5888] Camarillo, G. and H. Schulzrinne, "The Session Description Protocol (SDP) Grouping Framework", RFC 5888, June 2010.
- [RFC6904] Lennox, J., "Encryption of Header Extensions in the Secure Real-time Transport Protocol (SRTP)", RFC 6904, April 2013.

10.2. Informative References

- [I-D.ietf-mmusic-trickle-ice]
Ivov, E., Rescorla, E., and J. Uberti, "Trickle ICE: Incremental Provisioning of Candidates for the Interactive Connectivity Establishment (ICE) Protocol", draft-ietf-mmusic-trickle-ice-00 (work in progress), March 2013.
- [I-D.ietf-rtcweb-data-protocol]
Jesup, R., Loreto, S., and M. Tuexen, "WebRTC Data Channel Protocol", draft-ietf-rtcweb-data-protocol-04 (work in progress), February 2013.
- [I-D.jennings-rtcweb-signaling]
Jennings, C., Rosenberg, J., and R. Jesup, "RTCWeb Offer/Answer Protocol (ROAP)", draft-jennings-rtcweb-signaling-01 (work in progress), October 2011.
- [I-D.nandakumar-rtcweb-sdp]
Nandakumar, S. and C. Jennings, "SDP for the WebRTC", draft-nandakumar-rtcweb-sdp-02 (work in progress), July 2013.
- [RFC3389] Zopf, R., "Real-time Transport Protocol (RTP) Payload for Comfort Noise (CN)", RFC 3389, September 2002.
- [RFC3556] Casner, S., "Session Description Protocol (SDP) Bandwidth Modifiers for RTP Control Protocol (RTCP) Bandwidth", RFC 3556, July 2003.
- [RFC3960] Camarillo, G. and H. Schulzrinne, "Early Media and Ringing Tone Generation in the Session Initiation Protocol (SIP)", RFC 3960, December 2004.
- [RFC4568] Andreasen, F., Baugher, M., and D. Wing, "Session Description Protocol (SDP) Security Descriptions for Media Streams", RFC 4568, July 2006.
- [RFC4588] Rey, J., Leon, D., Miyazaki, A., Varsa, V., and R. Hakenberg, "RTP Retransmission Payload Format", RFC 4588, July 2006.
- [RFC5506] Johansson, I. and M. Westerlund, "Support for Reduced-Size Real-Time Transport Control Protocol (RTCP): Opportunities and Consequences", RFC 5506, April 2009.

- [RFC5576] Lennox, J., Ott, J., and T. Schierl, "Source-Specific Media Attributes in the Session Description Protocol (SDP)", RFC 5576, June 2009.
- [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, May 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.
- [W3C.WD-webrtc-20111027]
Bergkvist, A., Burnett, D., Narayanan, A., and C. Jennings, "WebRTC 1.0: Real-time Communication Between Browsers", World Wide Web Consortium WD WD-webrtc-20111027, October 2011,
<<http://www.w3.org/TR/2011/WD-webrtc-20111027>>.

Appendix A. JSEP Implementation Examples

A.1. Example API Flows

Below are several sample flows for the new PeerConnection and library APIs, demonstrating when the various APIs are called in different situations and with various transport protocols. For clarity and simplicity, the createOffer/createAnswer calls are assumed to be synchronous in these examples, whereas the actual APIs are async.

A.1.1. Call using ROAP

This example demonstrates a ROAP call, without the use of trickle candidates.

```
// Call is initiated toward Answerer
OffererJS->OffererUA: pc = new PeerConnection();
OffererJS->OffererUA: pc.addStream(localStream, null);
OffererUA->OffererJS: iceCallback(candidate);
OffererJS->OffererUA: offer = pc.createOffer(null);
OffererJS->OffererUA: pc.setLocalDescription("offer", offer);
OffererJS->AnswererJS: { "type": "OFFER", "sdp": offer }

// OFFER arrives at Answerer
AnswererJS->AnswererUA: pc = new PeerConnection();
AnswererJS->AnswererUA: pc.setRemoteDescription("offer", msg.sdp);
AnswererUA->AnswererJS: onaddstream(remoteStream);
AnswererUA->OffererUA: iceCallback(candidate);
```

```
// Answerer accepts call
AnswererJS->AnswererUA: pc.addStream(localStream, null);
AnswererJS->AnswererUA: answer = pc.createAnswer(msg.sdp, null);
AnswererJS->AnswererUA: pc.setLocalDescription("answer", answer);
AnswererJS->OffererJS:  {"type":"ANSWER","sdp":answer }

// ANSWER arrives at Offerer
OffererJS->OffererUA:  pc.setRemoteDescription("answer", answer);
OffererUA->OffererJS:  onaddstream(remoteStream);

// ICE Completes (at Answerer)
AnswererUA->OffererUA:  Media

// ICE Completes (at Offerer)
OffererJS->AnswererJS:  {"type":"OK" }
OffererUA->AnswererUA:  Media
```

A.1.2. Call using XMPP

This example demonstrates an XMPP call, making use of trickle candidates.

```
// Call is initiated toward Answerer
OffererJS->OffererUA:  pc = new PeerConnection();
OffererJS->OffererUA:  pc.addStream(localStream, null);
OffererJS->OffererUA:  offer = pc.createOffer(null);
OffererJS->OffererUA:  pc.setLocalDescription("offer", offer);
OffererJS:            xmpp = createSessionInitiate(offer);
OffererJS->AnswererJS: <jingle action="session-initiate"/>

OffererJS->OffererUA:  pc.startIce();
OffererUA->OffererJS:  onicecandidate(cand);
OffererJS:            createTransportInfo(cand);
OffererJS->AnswererJS: <jingle action="transport-info"/>

// session-initiate arrives at Answerer
AnswererJS->AnswererUA: pc = new PeerConnection();
AnswererJS:            offer = parseSessionInitiate(xmpp);
AnswererJS->AnswererUA: pc.setRemoteDescription("offer", offer);
AnswererUA->AnswererJS: onaddstream(remoteStream);

// transport-infos arrive at Answerer
AnswererJS->AnswererUA: candidate = parseTransportInfo(xmpp);
AnswererJS->AnswererUA: pc.addIceCandidate(candidate);
AnswererUA->AnswererJS: onicecandidate(cand)
AnswererJS:            createTransportInfo(cand);
AnswererJS->OffererJS: <jingle action="transport-info"/>
```

```
// transport-infos arrive at Offerer
OffererJS->OffererUA:  candidates = parseTransportInfo(xmpp);
OffererJS->OffererUA:  pc.addIceCandidate(candidates);

// Answerer accepts call
AnswererJS->AnswererUA: pc.addStream(localStream, null);
AnswererJS->AnswererUA: answer = pc.createAnswer(offer, null);
AnswererJS:            xmpp = createSessionAccept(answer);
AnswererJS->AnswererUA: pc.setLocalDescription("answer", answer);
AnswererJS->OffererJS:  <jingle action="session-accept"/>

// session-accept arrives at Offerer
OffererJS:            answer = parseSessionAccept(xmpp);
OffererJS->OffererUA:  pc.setRemoteDescription("answer", answer);
OffererUA->OffererJS:  onaddstream(remoteStream);

// ICE Completes (at Answerer)
AnswererUA->OffererUA: Media

// ICE Completes (at Offerer)
OffererUA->AnswererUA: Media
```

A.1.1.3. Adding video to a call, using XMPP

This example demonstrates an XMPP call, where the XMPP content-add mechanism is used to add video media to an existing session. For simplicity, candidate exchange is not shown.

Note that the offerer for the change to the session may be different than the original call offerer.

```
// Offerer adds video stream
OffererJS->OffererUA:  pc.addStream(videoStream)
OffererJS->OffererUA:  offer = pc.createOffer(null);
OffererJS:            xmpp = createContentAdd(offer);
OffererJS->OffererUA:  pc.setLocalDescription("offer", offer);
OffererJS->AnswererJS: <jingle action="content-add"/>

// content-add arrives at Answerer
AnswererJS:            offer = parseContentAdd(xmpp);
AnswererJS->AnswererUA: pc.setRemoteDescription("offer", offer);
AnswererJS->AnswererUA: answer = pc.createAnswer(offer, null);
AnswererJS->AnswererUA: pc.setLocalDescription("answer", answer);
AnswererJS:            xmpp = createContentAccept(answer);
AnswererJS->OffererJS:  <jingle action="content-accept"/>

// content-accept arrives at Offerer
```

```
OffererJS:          answer = parseContentAccept(xmpp);
OffererJS->OffererUA: pc.setRemoteDescription("answer", answer);
```

A.1.4. Simultaneous add of video streams, using XMPP

This example demonstrates an XMPP call, where new video sources are added at the same time to a call that already has video; since adding these sources only affects one side of the call, there is no conflict. The XMPP description-info mechanism is used to indicate the new sources to the remote side.

```
// Offerer and "Answerer" add video streams at the same time
OffererJS->OffererUA: pc.addStream(offererVideoStream2)
OffererJS->OffererUA: offer = pc.createOffer(null);
OffererJS:          xmpp = createDescriptionInfo(offer);
OffererJS->OffererUA: pc.setLocalDescription("offer", offer);
OffererJS->AnswererJS: <jingle action="description-info"/>

AnswererJS->AnswererUA: pc.addStream(answererVideoStream2)
AnswererJS->AnswererUA: offer = pc.createOffer(null);
AnswererJS:          xmpp = createDescriptionInfo(offer);
AnswererJS->AnswererUA: pc.setLocalDescription("offer", offer);
AnswererJS->OffererJS: <jingle action="description-info"/>

// description-info arrives at "Answerer", and is acked
AnswererJS:          offer = parseDescriptionInfo(xmpp);
AnswererJS->OffererJS: <iq type="result"/> // ack

// description-info arrives at Offerer, and is acked
OffererJS:          offer = parseDescriptionInfo(xmpp);
OffererJS->AnswererJS: <iq type="result"/> // ack

// ack arrives at Offerer; remote offer is used as an answer
OffererJS->OffererUA: pc.setRemoteDescription("answer", offer);

// ack arrives at "Answerer"; remote offer is used as an answer
AnswererJS->AnswererUA: pc.setRemoteDescription("answer", offer);
```

A.1.5. Call using SIP

This example demonstrates a simple SIP call (e.g. where the client talks to a SIP proxy over WebSockets).

```
// Call is initiated toward Answerer
OffererJS->OffererUA: pc = new PeerConnection();
OffererJS->OffererUA: pc.addStream(localStream, null);
```



```
OffererUA->OffererJS:  onicecandidate(candidate);
OffererJS->OffererUA:  offer = pc.createOffer(null);
OffererJS->OffererUA:  pc.setLocalDescription("offer", offer);
OffererJS:              sip = createInvite(offer);
OffererJS->AnswererJS: SIP INVITE w/ SDP

// INVITE arrives at Answerer
AnswererJS->AnswererUA: pc = new PeerConnection();
AnswererJS:              offer = parseInvite(sip);
AnswererJS->AnswererUA: pc.setRemoteDescription("offer", offer);
AnswererUA->AnswererJS: onaddstream(remoteStream);
AnswererUA->OffererUA:  onicecandidate(candidate);

// Answerer accepts call
AnswererJS->AnswererUA: pc.addStream(localStream, null);
AnswererJS->AnswererUA: answer = pc.createAnswer(offer, null);
AnswererJS:              sip = createResponse(200, answer);
AnswererJS->AnswererUA: pc.setLocalDescription("answer", answer);
AnswererJS->OffererJS:  200 OK w/ SDP

// 200 OK arrives at Offerer
OffererJS:              answer = parseResponse(sip);
OffererJS->OffererUA:  pc.setRemoteDescription("answer", answer);
OffererUA->OffererJS:  onaddstream(remoteStream);
OffererJS->AnswererJS: ACK

// ICE Completes (at Answerer)
AnswererUA->OffererUA:  Media

// ICE Completes (at Offerer)
OffererUA->AnswererUA:  Media
```

A.1.6. Handling early media (e.g. 1-800-GO FEDEX), using SIP

This example demonstrates how early media could be handled; for simplicity, only the offerer side of the call is shown.

```
// Call is initiated toward Answerer
OffererJS->OffererUA:  pc = new PeerConnection();
OffererJS->OffererUA:  pc.addStream(localStream, null);
OffererUA->OffererJS:  onicecandidate(candidate);
OffererJS->OffererUA:  offer = pc.createOffer(null);
OffererJS->OffererUA:  pc.setLocalDescription("offer", offer);
OffererJS:              sip = createInvite(offer);
OffererJS->AnswererJS: SIP INVITE w/ SDP

// 180 Ringing is received by offerer, w/ SDP
```

```

OffererJS:          answer = parseResponse(sip);
OffererJS->OffererUA: pc.setRemoteDescription("pranswer", answer);
OffererUA->OffererJS: onaddstream(remoteStream);

// ICE Completes (at Offerer)
OffererUA->AnswererUA: Media

// 200 OK arrives at Offerer
OffererJS:          answer = parseResponse(sip);
OffererJS->OffererUA: pc.setRemoteDescription("answer", answer);
OffererJS->AnswererJS: ACK

```

A.2. Example Session Descriptions

A.2.1. createOffer

This SDP shows a typical initial offer, created by `createOffer` for a `PeerConnection` with a single audio `MediaStreamTrack`, a single video `MediaStreamTrack`, and a single data channel. Host candidates have also already been gathered. Note some lines have been broken into two lines for formatting reasons.

```

v=0
o=- 4962303333179871722 1 IN IP4 0.0.0.0
s=-
t=0 0
a=msid-semantic:WMS
a=group:BUNDLE audio video data
m=audio 56500 UDP/TLS/RTP/SAVPF 111 0 8 126
c=IN IP4 192.0.2.1
a=rtcp:56501 IN IP4 192.0.2.1
a=candidate:3348148302 1 udp 2113937151 192.0.2.1 56500
      typ host generation 0
a=candidate:3348148302 2 udp 2113937151 192.0.2.1 56501
      typ host generation 0
a=ice-ufrag:ETEnlv9DoTMB9J4r
a=ice-pwd:OtSK0WpNtpUjkY4+86js7ZQl
a=ice-options:trickle
a=mid:audio
a=extmap:1 urn:ietf:params:rtp-hdext:ssrc-audio-level
a=sendrecv
a=rtcp-mux
a=rtcp-rsize
a=fingerprint:sha-256
      19:E2:1C:3B:4B:9F:81:E6:B8:5C:F4:A5:A8:D8:73:04
      BB:05:2F:70:9F:04:A9:0E:05:E9:26:33:E8:70:88:A2
a=setup:actpass

```

```
a=rtpmap:111 opus/48000/2
a=fmtp:111 minptime=10
a=rtpmap:0 PCMU/8000
a=rtpmap:8 PCMA/8000
a=rtpmap:126 telephone-event/8000
a=maxptime:60
a=ssrc:1732846380 cname:EocUGlf0fcg/yvY7
a=msid:47017fee-b6c1-4162-929c-a25110252400
      f83006c5-a0ff-4e0a-9ed9-d3e6747be7d9
m=video 56502 UDP/TLS/RTP/SAVPF 100 115 116 117
c=IN IP4 192.0.2.1
a=rtcp:56503 IN IP4 192.0.2.1
a=candidate:3348148302 1 udp 2113937151 192.0.2.1 56502
      typ host generation 0
a=candidate:3348148302 2 udp 2113937151 192.0.2.1 56503
      typ host generation 0
a=ice-ufrag:BGKkWnG5GmiUpdIV
a=ice-pwd:mqyWSAjvtKwTGnvHPztQ9mIf
a=ice-options:trickle
a=mid:video
a=extmap:2 urn:ietf:params:rtp-hdrext:toffset
a=extmap:3 http://www.webrtc.org/experiments/rtp-hdrext/abs-send-time
a=sendrecv
a=rtcp-mux
a=rtcp-rsize
a=fingerprint:sha-256
      19:E2:1C:3B:4B:9F:81:E6:B8:5C:F4:A5:A8:D8:73:04
      :BB:05:2F:70:9F:04:A9:0E:05:E9:26:33:E8:70:88:A2
a=setup:actpass
a=rtpmap:100 VP8/90000
a=rtcp-fb:100 ccm fir
a=rtcp-fb:100 nack
a=rtcp-fb:100 goog-remb
a=rtpmap:115 rtx/90000
a=fmtp:115 apt=100
a=rtpmap:116 red/90000
a=rtpmap:117 ulpfec/90000
a=ssrc:1366781083 cname:EocUGlf0fcg/yvY7
a=ssrc:1366781084 cname:EocUGlf0fcg/yvY7
a=ssrc:1366781085 cname:EocUGlf0fcg/yvY7
a=ssrc-group:FID 1366781083 1366781084
a=ssrc-group:FEC 1366781083 1366781085
a=msid:61317484-2ed4-49d7-9eb7-1414322a7aae
      f30bdb4a-5db8-49b5-bcdc-e0c9a23172e0
m=application 56504 DTLS/SCTP 5000
c=IN IP4 192.0.2.1
a=candidate:3348148302 1 udp 2113937151 192.0.2.1 56504
      typ host generation 0
```

```
a=ice-ufrag:VD5v2BnbZm3mgP3d
a=ice-pwd:+Jlkuox+VVIUDqxcfIDuTZMH
a=ice-options:trickle
a=mid:data
a=fingerprint:sha-256 19:E2:1C:3B:4B:9F:81:E6:B8:5C:F4:A5:A8:D8:73:04
                        :BB:05:2F:70:9F:04:A9:0E:05:E9:26:33:E8:70:88:A2
a=setup:actpass
a=sctpmap:5000 webrtc-datachannel 16
```

A.2.2. createAnswer

This SDP shows a typical initial answer to the above offer, created by createAnswer for a PeerConnection with a single audio MediaStreamTrack, a single video MediaStreamTrack, and a single data channel. Host candidates have also already been gathered. Note some lines have been broken into two lines for formatting reasons.

```
v=0
o=- 6729291447651054566 1 IN IP4 0.0.0.0
s=-
t=0 0
a=msid-semantic:WMS
a=group:BUNDLE audio video data
m=audio 20000 UDP/TLS/RTP/SAVPF 111 0 8 126
c=IN IP4 192.0.2.2
a=candidate:2299743422 1 udp 2113937151 192.0.2.2 20000
                typ host generation 0
a=ice-ufrag:6sFvz2gdLkEwjZEr
a=ice-pwd:cOTZKZNVlO9RSGsEGM63JXT2
a=fingerprint:sha-256 6B:8B:F0:65:5F:78:E2:51:3B:AC:6F:F3:3F:46:1B:35
                        :DC:B8:5F:64:1A:24:C2:43:F0:A1:58:D0:A1:2C:19:08
a=setup:active
a=mid:audio
a=extmap:1 urn:ietf:params:rtp-hdrext:ssrc-audio-level
a=sendrecv
a=rtcp-mux
a=rtpmap:111 opus/48000/2
a=fmtp:111 minptime=10
a=rtpmap:0 PCMU/8000
a=rtpmap:8 PCMA/8000
a=rtpmap:126 telephone-event/8000
a=maxptime:60
a=ssrc:3429951804 cname:Q/NWslao1HmN4Xa5
a=msid:PI39StLS8W7ZbQ1lsJsWUXkr3Zf12fJUvzQ1
                PI39StLS8W7ZbQ1lsJsWUXkr3Zf12fJUvzQ1a0
m=video 20000 UDP/TLS/RTP/SAVPF 100 115 116 117
c=IN IP4 192.0.2.2
```

```

a=candidate:2299743422 1 udp 2113937151 192.0.2.2 20000
        typ host generation 0
a=ice-ufrag:6sFvz2gdLkEwjZEr
a=ice-pwd:cOTZKZNVlO9RSGsEGM63JXT2
a=fingerprint:sha-256 6B:8B:F0:65:5F:78:E2:51:3B:AC:6F:F3:3F:46:1B:35
        :DC:B8:5F:64:1A:24:C2:43:F0:A1:58:D0:A1:2C:19:08

a=setup:active
a=mid:video
a=extmap:2 urn:ietf:params:rtp-hdrext:toffset
a=extmap:3 http://www.webrtc.org/experiments/rtp-hdrext/abs-send-time
a=sendrecv
a=rtcp-mux
a=rtpmap:100 VP8/90000
a=rtcp-fb:100 ccm fir
a=rtcp-fb:100 nack
a=rtcp-fb:100 goog-remb
a=rtpmap:115 rtx/90000
a=fmtp:115 apt=100
a=rtpmap:116 red/90000
a=rtpmap:117 ulpfec/90000
a=ssrc:3229706345 cname:Q/NWslao1HmN4Xa5
a=ssrc:3229706346 cname:Q/NWslao1HmN4Xa5
a=ssrc:3229706347 cname:Q/NWslao1HmN4Xa5
a=ssrc-group:FID 3229706345 3229706346
a=ssrc-group:FEC 3229706345 3229706347
a=msid:PI39StLS8W7ZbQ1lsJsWUXkr3Zf12fJUvzQ1
        PI39StLS8W7ZbQ1lsJsWUXkr3Zf12fJUvzQ1v0
m=application 20000 DTLS/SCTP 5000
c=IN IP4 192.0.2.2
a=candidate:2299743422 1 udp 2113937151 192.0.2.2 20000
        typ host generation 0
a=ice-ufrag:6sFvz2gdLkEwjZEr
a=ice-pwd:cOTZKZNVlO9RSGsEGM63JXT2
a=fingerprint:sha-256 6B:8B:F0:65:5F:78:E2:51:3B:AC:6F:F3:3F:46:1B:35
        :DC:B8:5F:64:1A:24:C2:43:F0:A1:58:D0:A1:2C:19:08

a=setup:active
a=mid:data
a=sctpmap:5000 webrtc-datachannel 16

```

A.2.3. Call Flows

Example SDP for WebRTC call flows can be found in [I-D.nandakumar-rtcweb-sdp]. [TODO: should these call flows be merged into this section?]

Appendix B. Change log

Changes in draft-05:

- o Fixed several issues identified in the createOffer/Answer sections during document review.
- o Updated references.

Changes in draft-04:

- o Filled in sections on createOffer and createAnswer.
- o Added SDP examples.
- o Fixed references.

Changes in draft-03:

- o Added text describing relationship to W3C specification

Changes in draft-02:

- o Converted from nroff
- o Removed comparisons to old approaches abandoned by the working group
- o Removed stuff that has moved to W3C specification
- o Align SDP handling with W3C draft
- o Clarified section on forking.

Changes in draft-01:

- o Added diagrams for architecture and state machine.
- o Added sections on forking and rehydration.
- o Clarified meaning of "pranswer" and "answer".
- o Reworked how ICE restarts and media directions are controlled.
- o Added list of parameters that can be changed in a description.
- o Updated suggested API and examples to match latest thinking.
- o Suggested API and examples have been moved to an appendix.

Changes in draft -00:

- o Migrated from draft-uberti-rtcweb-jsep-02.

Authors' Addresses

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

Email: justin@uberti.name

Cullen Jennings
Cisco
170 West Tasman Drive
San Jose, CA 95134
USA

Email: fluffy@iii.ca

RTCWEB Working Group
Internet-Draft
Intended status: Standards Track
Expires: April 24, 2014

C. Perkins
University of Glasgow
M. Westerlund
Ericsson
J. Ott
Aalto University
October 21, 2013

Web Real-Time Communication (WebRTC): Media Transport and Use of RTP
draft-ietf-rtcweb-rtp-usage-10

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 April 24, 2014.

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. Rationale	4
3. Terminology	4
4. WebRTC Use of RTP: Core Protocols	5
4.1. RTP and RTCP	5
4.2. Choice of the RTP Profile	6
4.3. Choice of RTP Payload Formats	7
4.4. Use of RTP Sessions	9
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)	11
4.9. Generation of the RTCP Canonical Name (CNAME)	11
5. WebRTC Use of RTP: Extensions	12
5.1. Conferencing Extensions	12
5.1.1. Full Intra Request (FIR)	13
5.1.2. Picture Loss Indication (PLI)	13
5.1.3. Slice Loss Indication (SLI)	14
5.1.4. Reference Picture Selection Indication (RPSI)	14
5.1.5. Temporal-Spatial Trade-off Request (TSTR)	14
5.1.6. Temporary Maximum Media Stream Bit Rate Request (TMMBR)	14
5.2. Header Extensions	14
5.2.1. Rapid Synchronisation	15
5.2.2. Client-to-Mixer Audio Level	15
5.2.3. Mixer-to-Client Audio Level	16
5.2.4. Associating RTP Media Streams and Signalling Contexts	16
6. WebRTC Use of RTP: Improving Transport Robustness	16
6.1. Negative Acknowledgements and RTP Retransmission	16
6.2. Forward Error Correction (FEC)	17
7. WebRTC Use of RTP: Rate Control and Media Adaptation	18
7.1. Boundary Conditions and Circuit Breakers	18
7.2. RTCP Limitations for Congestion Control	19
7.3. Congestion Control Interoperability and Legacy Systems	20
8. WebRTC Use of RTP: Performance Monitoring	21
9. WebRTC Use of RTP: Future Extensions	21
10. Signalling Considerations	22
11. WebRTC API Considerations	23
12. RTP Implementation Considerations	24
12.1. Configuration and Use of RTP Sessions	24

12.1.1. Use of Multiple Media Flows Within an RTP Session	24
12.1.2. Use of Multiple RTP Sessions	26
12.1.3. Differentiated Treatment of Flows	31
12.2. Source, Flow, and Participant Identification	32
12.2.1. Media Streams	32
12.2.2. Media Streams: SSRC Collision Detection	33
12.2.3. Media Synchronisation Context	34
12.2.4. Correlation of Media Streams	34
13. Security Considerations	35
14. IANA Considerations	35
15. Open Issues	35
16. Acknowledgements	36
17. References	36
17.1. Normative References	36
17.2. Informative References	39
Authors' Addresses	41

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 endpoints, along with suggested extensions for enhanced functionality.

This memo specifies a protocol intended for use within the WebRTC framework, but is not restricted to that context. An overview of the WebRTC framework is given in [I-D.ietf-rtcweb-overview].

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 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. The memo concludes with security considerations (Section 13) and IANA considerations (Section 14).

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 20 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].

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, following [RFC3550] and [I-D.ietf-avtcore-rtp-multi-stream]. Support for the RTCP optimisations for multi-SSRC sessions defined in [I-D.ietf-avtcore-rtp-multi-stream-optimisation] is RECOMMENDED.

- * (tbd: do endpoints need to signal the maximum number of SSRCs that they support (e.g., draft-westerlund-mmusic-max-ssrc-01) and/or some constraint on the maximum number of simultaneous streams of various kinds that can be decoded?)
- o Random choice of SSRC on joining a session; collision detection and resolution for SSRC values (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. Note that additional RTCP Packet types are needed by the RTP/SAVPF Profile (Section 4.2) and the other RTCP extensions (Section 5).
- 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 RTCP-Based Feedback (RTP/SAVPF) [RFC5124], as extended by [RFC7007], 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.

The keying mechanism(s) to be used with the RTP/SAVPF profile are defined in Section 5.5 of [I-D.ietf-rtcweb-security-arch] or its replacement.

4.3. Choice of RTP Payload Formats

The set of mandatory to implement codecs and RTP payload formats for WebRTC is not specified in this memo. Implementations can support any codec for which an RTP payload format and associated signalling is defined. Implementation cannot assume that the other participants in an RTP session understand any RTP payload format, no matter how common; the mapping between RTP payload type numbers and specific configurations of particular RTP payload formats MUST be agreed before those payload types/formats can be used. In an SDP context, this can be done using the "a=rtpmap:" and "a=fmtp:" attributes associated with an "m=" line.

Endpoints can signal support for multiple RTP payload formats, or multiple configurations of a single RTP payload format, as long as each unique RTP payload format configuration uses a different RTP payload type number. As outlined in Section 4.8, the RTP payload type number is sometimes used to associate an RTP media stream with a signalling context. This association is possible provided unique RTP payload type numbers are used in each context. For example, an RTP media stream can be associated with an SDP "m=" line by comparing the RTP payload type numbers used by the media stream with payload types signalled in the "a=rtpmap:" lines in the media sections of the SDP. If RTP media streams are being associated with signalling contexts based on the RTP payload type, then the assignment of RTP payload type numbers MUST be unique across signalling contexts; if the same RTP payload format configuration is used in multiple contexts, then a different RTP payload type number has to be assigned in each context to ensure uniqueness. If the RTP payload type number is not being used to associated RTP media streams with a signalling context, then the same RTP payload type number can be used to indicate the exact same RTP payload format configuration in multiple contexts.

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 types of media (e.g., audio and video) 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 type of media 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. Use of RTP Sessions

An association amongst a set of participants communicating using RTP is known as an RTP session. A participant can be involved in several RTP sessions at the same time. In a multimedia session, each type of media has typically been carried in a separate RTP session (e.g., using one RTP session for the audio, and a separate RTP session using different transport addresses for the video). WebRTC implementations of RTP are REQUIRED to implement support for multimedia sessions in this way, separating each session using different transport-layer addresses (e.g., different UDP ports) for compatibility with legacy systems.

In modern day networks, however, with the widespread use of network address/port translators (NAT/NAPT) and firewalls, it is desirable to reduce the number of transport-layer flows used by RTP applications. This can be done by sending all the RTP media streams in a single RTP session, which will comprise a single transport-layer flow (this will prevent the use of some quality-of-service mechanisms, as discussed in Section 12.1.3). Implementations 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 multiple types of media are to be used in a single RTP session, all participants in that session MUST agree to this usage. In an SDP context, [I-D.ietf-mmusic-sdp-bundle-negotiation] can be used to signal this.

It is also possible to use a shim-based approach to run multiple RTP sessions on a single transport-layer flow. This gives advantages in some gateway scenarios, and makes it easy to distinguish groups of RTP media streams that might need distinct processing. One way of doing this is described in [I-D.westerlund-avtcore-transport-multiplexing]. At the time of this writing, there is no consensus to use a shim-based approach in WebRTC implementations.

Further discussion about when different RTP session structures and multiplexing methods are suitable can be found in [I-D.ietf-avtcore-multiplex-guidelines].

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 RTP and RTCP sent on separate transport-layer addresses.

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 non-compound RTCP feedback packets [RFC5506] 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]. 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 synchronisation source (SSRC) identifiers, using the "a=ssrc:" SDP attribute defined in Section 4.1 and Section 5 of [RFC5576]. Implementations MUST also support the "previous-ssrc" source attribute defined in Section 6.2 of [RFC5576]. Other per-SSRC attributes defined in [RFC5576] MAY be supported.

Use of the "a=ssrc:" attribute to signal SSRC identifiers in an RTP session is OPTIONAL. Implementations MUST be prepared to accept RTP and RTCP packets using SSRCs that have not been explicitly signalled ahead of time. Implementations MUST support random SSRC assignment, and MUST support SSRC collision detection and resolution, according to [RFC3550]. When using signalled SSRC values, collision detection MUST be performed as described in Section 5 of [RFC5576].

It is often desirable to associate an RTP media stream with a non-RTP context (e.g., to associate an RTP media stream with an "m=" line in a session description formatted using SDP). If SSRCs are signalled this is straightforward (in SDP the "a=ssrc:" line will be at the media level, allowing a direct association with an "m=" line). If SSRCs are not signalled, the RTP payload type numbers used in an RTP media stream are often sufficient to associate that media stream with a signalling context (e.g., if RTP payload type numbers are assigned as described in Section 4.3 of this memo, the RTP payload types used by an RTP media stream can be compared with values in SDP "a=rtpmap:" lines, which are at the media level in SDP, and so map to an "m=" line).

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 at least one unique RTCP CNAME value. An endpoint MAY have multiple CNAMEs, as the CNAME also identifies a particular synchronization context, i.e. all SSRC associated with a CNAME share a common reference clock, and if an endpoint have SSRCs associated with different reference clocks it will need to use multiple CNAMEs. This ought not be common, and if possible reference clocks ought to be mapped to each other and one chosen to be used with RTP and RTCP.

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 [RFC7022] 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 is popular in IPTV systems, overlay-based topologies dominate in interactive conferencing environments. Such overlay-based topologies typically use one or more central servers to connect end-points in a star or flat tree topology. These central servers can be implemented in a number of ways as discussed in the memo on RTP Topologies [I-D.ietf-avtcore-rtp-topologies-update].

Not all of the possible the overlay-based topologies are suitable for use in the WebRTC environment. Specifically:

- o The use of video switching MCUs makes the use of RTCP for congestion control and quality of service reports problematic (see Section 3.6.2 of [I-D.ietf-avtcore-rtp-topologies-update]).

- o The use of content modifying MCUs with RTCP termination breaks RTP loop detection, and prevents receivers from identifying active senders (see section 3.8 of [I-D.ietf-avtcore-rtp-topologies-update]).

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 in Section 5.1.1 to Section 5.1.6 are designed to be used with centralised conferencing, where an RTP middlebox (e.g., a conference bridge) receives a participant's RTP media streams and distributes them to the other participants. These extensions 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. WebRTC senders MUST understand and react to the FIR 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. WebRTC senders MUST 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. Support for 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. 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 in addition to RTCP SR-based synchronisation. 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 [RFC6904] 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 [RFC6904] since the information contained in these header extensions can be considered sensitive.

5.2.4. Associating RTP Media Streams and Signalling Contexts

(tbd: it seems likely that we need a mechanism to associate RTP media streams with signalling contexts. The mechanism by which this is done will likely be some combination of an RTP header extension, periodic transmission of a new RTCP SDES item, and some signalling extension. The semantics of those items are not yet settled; see draft-westerlund-avtext-rtcp-sdes-srname, draft-ietf-mmusic-msid, and draft-even-mmusic-application-token for discussion).

6. WebRTC Use of RTP: Improving Transport Robustness

There are tools that can make RTP media streams robust against packet loss and reduce the impact of loss on media quality. However, they all add extra bits compared to a non-robust stream. The overhead of these extra bits needs to be considered, and the aggregate bit-rate MUST be rate controlled to avoid causing network congestion (see Section 7). As a result, 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 can 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, as described at the end of Section 6.4.1 of [RFC3550]). 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 does not need 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 send one or more RTP media streams to each participant in a WebRTC conference, and there can be several participants. Each of these RTP media streams 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 with 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 RTP circuit breaker is designed to enable applications to recognise and react to situations of extreme network congestion. However, since the RTP circuit breaker might not be triggered until congestion becomes extreme, it cannot be considered a substitute for congestion control, and applications **MUST** also implement congestion control to allow them to adapt to changes in network capacity. 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 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.

7.3. Congestion Control Interoperability and 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.

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. This memo cannot mandate behaviour for proprietary congestion control algorithms, but implementations that use such algorithms ought to be aware of this issue, and try to ensure that both effective congestion control is negotiated for media flowing in both directions. If the IETF were to standardise both sender- and receiver-based congestion control algorithms for WebRTC traffic in the future, the issues of interoperability, control, and ensuring that both directions of media flow are congestion controlled would also need to be considered.

8. WebRTC Use of RTP: Performance Monitoring

As described in Section 4.1, implementations are REQUIRED to generate RTCP Sender Report (SR) and Reception Report (RR) packets relating to the RTP media streams they send and receive. These RTCP reports can be used for performance monitoring purposes, since they include basic packet loss and jitter statistics.

A large number of additional performance metrics are supported by the RTCP Extended Reports (XR) framework [RFC3611]. It is not yet clear what extended metrics are appropriate for use in the WebRTC context, so implementations are not expected to generate any RTCP XR packets. However, implementations that can use detailed performance monitoring data MAY generate RTCP XR packets as appropriate; the use of such packets SHOULD be signalled in advance.

All WebRTC implementations MUST be prepared to receive RTP XR report packets, whether or not they were signalled. There is no requirement that the data contained in such reports be used, or exposed to the Javascript application, however.

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 that an external signalling channel exists, and can be used to configure 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 for authentication in SRTP packets and cryptographic transformation of the payload. WebRTC requires the use of the RTP/SAVPF profile, and this MUST be signalled if SDP is used. Interworking functions might transform this into the RTP/SAVP profile for a legacy use case, by indicating to the WebRTC end-point that the RTP/SAVPF is used, 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 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). Section 4.3 of this memo discusses requirements for uniqueness of payload types.

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.

The API also needs to be capable of handling when new SSRCs are received but not previously signalled by signalling in some fashion. Note, that not all SSRCs carries media directly associated with a media source, instead they can be repair or redundancy information for one or a set of SSRCs.

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. Configuration and Use of RTP Sessions

A WebRTC end-point will be a simultaneous participant in one or more RTP sessions. Each RTP session can convey multiple media flows, and can include media data from multiple end-points. In the following, we outline some ways in which WebRTC end-points can configure and use RTP sessions.

12.1.1. Use of Multiple Media Flows Within an RTP Session

RTP is a group communication protocol, and in a WebRTC context every RTP session can potentially contain multiple media flows. There are several reasons why this might be desirable:

Multiple media types: Outside of WebRTC, it is common to use one RTP session for each type of media (e.g., one RTP session for audio and one for video, each sent on a different UDP port). However, to reduce the number of UDP ports used, the default in WebRTC is to send all types of media in a single RTP session, as described in Section 4.4, using RTP and RTCP multiplexing (Section 4.5) to further reduce the number of UDP ports needed. This RTP session then uses only one UDP flow, but will contain multiple RTP media streams, each containing a different type of media. A common example might be an end-point with a camera and microphone that sends two RTP streams, one video and one audio, into a single RTP session.

Multiple Capture Devices: A WebRTC end-point might have multiple cameras, microphones, or other media capture devices, and so might want to generate several RTP media streams of the same media type. Alternatively, it might want to send media from a single capture device in several different formats or quality settings at once. Both can result in a single end-point sending multiple RTP media streams of the same media type into a single RTP session at the same time.

Associated Repair Data: An end-point might send a media stream that is somehow associated with another stream. For example, it might send an RTP stream that contains FEC or retransmission data relating to another stream. Some RTP payload formats send this sort of associated repair data as part of the original media stream, while others send it as a separate stream.

Layered or Multiple Description Coding: An end-point can use a layered media codec, for example H.264 SVC, or a multiple description codec, that generates multiple media flows, each with a distinct RTP SSRC, within a single RTP session.

RTP Mixers, Translators, and Other Middleboxes: An RTP session, in the WebRTC context, is a point-to-point association between an end-point and some other peer device, where those devices share a common SSRC space. The peer device might be another WebRTC end-point, or it might be an RTP mixer, translator, or some other form of media processing middlebox. In the latter cases, the middlebox might send mixed or relayed RTP streams from several participants, that the WebRTC end-point will need to render. Thus, even though a WebRTC end-point might only be a member of a single RTP session, the peer device might be extending that RTP session to incorporate other end-points. WebRTC is a group communication environment and end-points need to be capable of receiving, decoding, and playing out multiple RTP media streams at once, even in a single RTP session.

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

(tbd: need to discuss signalling for the above here, preferably by referring to a separate document that describes SDP use for WebRTC)

12.1.2. Use of Multiple RTP Sessions

In addition to sending and receiving multiple media streams within a single RTP session, a WebRTC end-point might participate in multiple RTP sessions. There are several reasons why a WebRTC end-point might choose to do this:

To interoperate with legacy devices: The common practice in the non-WebRTC world is to send different types of media in separate RTP sessions, for example using one RTP session for audio and another RTP session, on a different UDP port, for video. All WebRTC end-points need to support the option of sending different types of media on different RTP sessions, so they can interwork with such legacy devices. This is discussed further in Section 4.4.

To provide enhanced quality of service: Some network-based quality of service mechanisms operate on the granularity of UDP 5-tuples. If it is desired to use these mechanisms to provide differentiated quality of service for some RTP flows, then those RTP flows need to be sent in a separate RTP session using a different UDP port number, and with appropriate quality of service marking. This is discussed further in Section 12.1.3.

To separate media with different purposes: An end-point might want to send media streams that have different purposes on different RTP sessions, to make it easy for the peer device to distinguish them. For example, some centralised multiparty conferencing systems display the active speaker in high resolution, but show low resolution "thumbnails" of other participants. Such systems might configure the end-points to send simulcast high- and low-resolution versions of their video using separate RTP sessions, to simplify the operation of the central mixer. In the WebRTC context this 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, and will produce an independently encoded version with the codec parameters agreed specifically in the context of that PeerConnection. The central mixer can always distinguish packets corresponding to the low- and high-resolution streams by inspecting their SSRC, RTP payload type, or some other information contained in RTP header extensions or RTCP packets, but it can be

easier to distinguish the flows if they arrive on separate RTP sessions on separate UDP ports.

To directly connect with multiple peers: A multi-party conference does not need to use a central mixer. Rather, a multi-unicast mesh can be created, comprising several distinct RTP sessions, with each participant sending RTP traffic over a separate RTP session (that is, using an independent `PeerConnection` object) to every other participant, as shown in Figure 1. This topology has the benefit of not requiring a central mixer node that is trusted to access and manipulate the media data. 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.

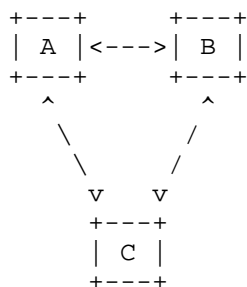


Figure 1: Multi-unicast using several RTP sessions

The multi-unicast topology could also 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 `PeerConnection` objects we recommend that this is implemented as several 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, since it will not see RTCP reports for the RTP session between B and C, whereas it would if all three participants were part of a single RTP session. Experience with the Mbone tools (experimental RTP-based multicast conferencing tools from the late 1990s) has showed that RTCP reception quality reports for third parties can usefully be presented to the users in a way that helps them understand asymmetric network problems, and the approach of using separate RTP sessions prevents this. However, an advantage of using separate RTP sessions is that it enables using different media bit-rates and RTP session configurations between 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. It is believed that these advantages outweigh the limitations in debugging power.

To indirectly connect with multiple peers: A common scenario in multi-party conferencing is to create indirect connections to multiple peers, using an RTP mixer, translator, or some other type of RTP middlebox. Figure 2 outlines a simple topology that might be used in a four-person centralised conference. The middlebox acts to optimise 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.

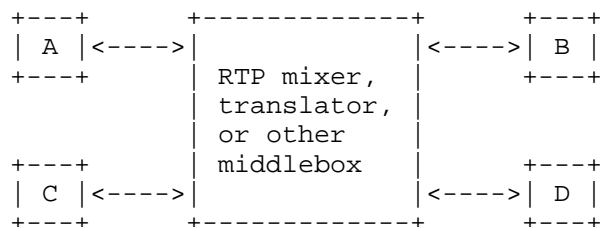


Figure 2: RTP mixer with only unicast paths

There are various methods of implementation for the middlebox. If implemented as a standard RTP mixer or translator, a single RTP session will extend across the middlebox and encompass all the end-points in one multi-party session. Other types of middlebox might use separate RTP sessions between each end-point and the middlebox. 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. The middlebox gets the significant responsibility to correctly perform congestion control, source identification, manage synchronization while providing the application with suitable media optimizations. The middlebox is also has to be a trusted node when it comes to security, since it manipulates either the RTP header or the media itself (or both) received from one end-point, 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.

RTP Mixers can create a situation where an end-point experiences a situation in-between a session with only two end-points and multiple RTP sessions. 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.

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, for example TESLA for SRTP [RFC4383], that are not part of the base WebRTC standards.

To forward media between multiple peers: It might be desirable for an end-point that receives an RTP media stream to be able to forward that media stream to a third party. There are obvious security and privacy implications in this, but also potential uses. If it is to be allowed, there are two implementation strategies: either the browser can relay the flow at the RTP layer, or it transcode and forward the media at the application layer.

A relay approach will result in the RTP session be extended beyond the PeerConnection, making both the original end-point and the destination to which the media is forwarded part of the RTP session. These end-points can have different path characteristics, and hence different reception quality. Thus sender's congestion control needs to be capable of handling this. The security solution can either support mechanism that the sender informs both receivers of the key; alternatively the end-point that is forwarding the media needs to decrypt and then re-encrypt using a new key. The relay based approach has the advantage that the forwarding end-point does not need to transcode the media, thus maintaining the quality of the encoding and reducing the computational complexity requirements. If the right security solutions are supported then the end-point that receives the forwarded media will be able to verify the authenticity of the media coming from the original sender. A downside is that the original sender is forced to take both receivers into consideration when delivering content.

The media transcoder approach is similar to having the forwarding end-point act as Mixer, terminating the RTP session, combined with a transcoder. The original sender will only see a single receiver of its media. The receiving end-point will be responsible to produce a RTP media stream suitable for onwards transmission. This might require media transcoding for congestion control purpose to produce a suitable bit-rate. Thus losing media quality in the transcoding and forcing the forwarding end-point to spend the resource on the transcoding. The media transcoding does result in a separation of the two different legs removing almost all dependencies, and allowing the forwarding end-point to optimize its media transcoding operation. It also allows forwarding without the original sender being aware of the forwarding. The cost is greatly increased computational complexity on the forwarding node.

(tbd: ought media forwarding be allowed?)

12.1.3. 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.dhesikan-tsvwg-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.

12.2. Source, Flow, and Participant Identification

12.2.1. Media Streams

Each RTP media stream is identified by a unique synchronisation source (SSRC) identifier. The SSRC identifier is carried in the RTP data packets comprising a media stream, and is also used to identify that stream in the corresponding RTCP reports. The SSRC is chosen as discussed in Section 4.8. The first stage in demultiplexing RTP and RTCP packets received at a WebRTC end-point is to separate the media streams based on their SSRC value; once that is done, additional demultiplexing steps can determine how and where to render the media.

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 can 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.

12.2.2. Media Streams: 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 it 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.

12.2.3. Media Synchronisation Context

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 RTCP 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.

12.2.4. Correlation of Media Streams

(tbd: this need to outline the approach to mapping media streams to the signalling context defined in the unified plan)

(tbd: need to discuss correlation between associated RTP streams, for example between a media stream and its associated FEC stream)

13. 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 created by combining this secured RTP profile and DTLS-SRTP keying [RFC5764] as defined by Section 5.5 of [I-D.ietf-rtcweb-security-arch].

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).

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. Open Issues

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

1. tbd: The API mapping to RTP level concepts has to be agreed and documented in Section 11. This include both SSRC to API constructs, but also how different SSRC are related in this context.
2. tbd: 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 4.1.
3. tbd: The method for achieving simulcast of a media source has to be decided.
4. tbd: 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 12.1.3.
5. tbd: There are various reasons for having multiple SSRCs of the same media type in the PeerConnections RTP session(s) (Section 12.1.1). The signalling separating these cases needs clarifications, preferably just by pointing to relevant signalling section taking care of it. Related to Open Issue 1.
6. tbd: The section on usage of multiple RTP sessions (Section 12.1.2) raised the question: ought media forwarding be allowed?

16. Acknowledgements

The authors would like to thank Harald Alvestrand, Cary Bran, Charles Eckel, Cullen Jennings, Bernard Aboba, and the other members of the IETF RTCWEB working group for their valuable feedback.

17. References

17.1. Normative References

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

Westerlund, M., Perkins, C., and J. Lennox, "Sending Multiple Types of Media in a Single RTP Session", draft-ietf-avtcore-multi-media-rtp-session-03 (work in progress), July 2013.

[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-03 (work in progress), July 2013.

[I-D.ietf-avtcore-rtp-multi-stream-optimisation]
Lennox, J., Westerlund, M., Wu, W., and C. Perkins,
"Sending Multiple Media Streams in a Single RTP Session:
Grouping RTCP Reception Statistics and Other Feedback",
draft-ietf-avtcore-rtp-multi-stream-optimisation-00 (work
in progress), July 2013.

[I-D.ietf-avtcore-rtp-multi-stream]
Lennox, J., Westerlund, M., Wu, W., and C. Perkins,
"Sending Multiple Media Streams in a Single RTP Session",
draft-ietf-avtcore-rtp-multi-stream-01 (work in progress),
July 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-10 (work in progress), September
2013.

[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-05 (work in progress), October 2013.

[I-D.ietf-rtcweb-security-arch]
Rescorla, E., "WebRTC Security Architecture", draft-ietf-
rtcweb-security-arch-07 (work in progress), July 2013.

[I-D.ietf-rtcweb-security]
Rescorla, E., "Security Considerations for WebRTC", draft-
ietf-rtcweb-security-05 (work in progress), July 2013.

[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., Iovov, E., and E. Marocco, "A Real-time Transport Protocol (RTP) Header Extension for Client-to-Mixer Audio Level Indication", RFC 6464, December 2011.
- [RFC6465] Iovov, 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.
- [RFC6904] Lennox, J., "Encryption of Header Extensions in the Secure Real-time Transport Protocol (SRTP)", RFC 6904, April 2013.
- [RFC7007] Terriberry, T., "Update to Remove DVI4 from the Recommended Codecs for the RTP Profile for Audio and Video Conferences with Minimal Control (RTP/AVP)", RFC 7007, August 2013.
- [RFC7022] Begen, A., Perkins, C., Wing, D., and E. Rescorla, "Guidelines for Choosing RTP Control Protocol (RTCP) Canonical Names (CNAMES)", RFC 7022, September 2013.

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.dhesikan-tsvwg-rtcweb-qos]
Dhesikan, S., Druta, D., Jones, P., and J. Polk, "DSCP and other packet markings for RTCWeb QoS", draft-dhesikan-tsvwg-rtcweb-qos-02 (work in progress), July 2013.
- [I-D.ietf-avtcore-multiplex-guidelines]
Westerlund, M., Perkins, C., and H. Alvestrand, "Guidelines for using the Multiplexing Features of RTP to Support Multiple Media Streams", draft-ietf-avtcore-multiplex-guidelines-01 (work in progress), July 2013.
- [I-D.ietf-avtcore-rtp-topologies-update]
Westerlund, M. and S. Wenger, "RTP Topologies", draft-ietf-avtcore-rtp-topologies-update-00 (work in progress), April 2013.

- [I-D.ietf-rtcweb-overview]
Alvestrand, H., "Overview: Real Time Protocols for Browser-based Applications", draft-ietf-rtcweb-overview-08 (work in progress), September 2013.
- [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-12 (work in progress), October 2013.
- [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-transport-multiplexing]
Westerlund, M. and C. Perkins, "Multiple RTP Sessions on a Single Lower-Layer Transport", draft-westerlund-avtcore-transport-multiplexing-06 (work in progress), August 2013.
- [RFC3611] Friedman, T., Caceres, R., and A. Clark, "RTP Control Protocol Extended Reports (RTCP XR)", RFC 3611, November 2003.
- [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.

Authors' Addresses

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

Email: csp@csp Perkins.org
URI: <http://csp Perkins.org/>

Magnus Westerlund
Ericsson
Farogatan 6
SE-164 80 Kista
Sweden

Phone: +46 10 714 82 87
Email: magnus.westerlund@ericsson.com

Joerg Ott
Aalto University
School of Electrical Engineering
Espoo 02150
Finland

Email: jorg.ott@aalto.fi

RTC-Web
Internet-Draft
Intended status: Standards Track
Expires: January 16, 2014

E. Rescorla
RTFM, Inc.
July 15, 2013

Security Considerations for WebRTC
draft-ietf-rtcweb-security-05

Abstract

The Real-Time Communications on the Web (RTCWEB) working group is tasked with standardizing protocols for real-time communications between Web browsers, generally called "WebRTC". 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 soft phones) WebRTC communications are directly controlled by a Web server, which poses new security challenges. For instance, a Web browser might expose a JavaScript API which allows a server to place a video call. Unrestricted access to such an API would allow any site which a user visited to "bug" a user's computer, capturing any activity which passed in front of their camera. This document defines the WebRTC threat model and analyzes the security threats of WebRTC in that model.

Legal

THIS DOCUMENT AND THE INFORMATION CONTAINED THEREIN ARE PROVIDED ON AN "AS IS" BASIS AND THE CONTRIBUTOR, THE ORGANIZATION HE/SHE REPRESENTS OR IS SPONSORED BY (IF ANY), THE INTERNET SOCIETY, THE IETF TRUST, AND THE INTERNET ENGINEERING TASK FORCE, DISCLAIM ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO ANY WARRANTY THAT THE USE OF THE INFORMATION THEREIN WILL NOT INFRINGE ANY RIGHTS OR ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

Status of this Memo

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

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

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any

time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on January 16, 2014.

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.

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	4
2. Terminology	5
3. The Browser Threat Model	5
3.1. Access to Local Resources	6
3.2. Same Origin Policy	6
3.3. Bypassing SOP: CORS, WebSockets, and consent to communicate	7
4. Security for WebRTC Applications	7
4.1. Access to Local Devices	8
4.1.1. Threats from Screen Sharing	9
4.1.2. Calling Scenarios and User Expectations	9
4.1.2.1. Dedicated Calling Services	9
4.1.2.2. Calling the Site You're On	10
4.1.3. Origin-Based Security	10
4.1.4. Security Properties of the Calling Page	12
4.2. Communications Consent Verification	13
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	20
4.4.2. Browser Fingerprinting	21
5. Security Considerations	21
6. Acknowledgements	21
7. Changes Since -04	21
8. References	21
8.1. Normative References	21
8.2. Informative References	22
Author's Address	24

1. Introduction

The Real-Time Communications on the Web (RTCWEB) working group is tasked with standardizing protocols for real-time communications between Web browsers, generally called "WebRTC" [I-D.ietf-rtcweb-overview]. The major use cases for WebTC 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.

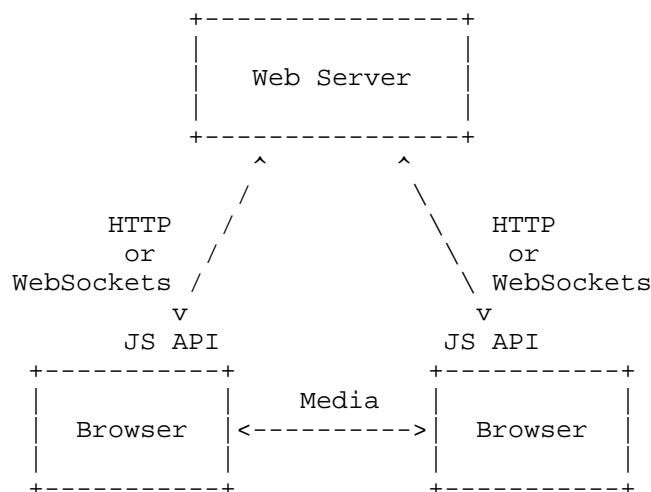


Figure 1: A simple WebRTC system

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", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [RFC2119].

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 for HTTP traffic, a network attacker is also a Web attacker, since it can inject traffic as if it were any non-HTTPS Web site. Thus, 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 screw yourself", as extensive research 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 those requests are not allowed to be made to any server, but rather solely to the same ORIGIN from whence the script came xref target="RFC6454"/> (although CORS [CORS] and WebSockets [RFC6455] provide a 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:

You are sending your media to entity A because you want to talk to Entity A (e.g., your mother).
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, identity 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 case, 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 Gmail 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 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 seen another use case, 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 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 apparently `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 sites, if the site embeds active content (e.g., JavaScript) that is fetched over HTTP or from an untrusted site, because that JavaScript is executed in the security context of the page [finer-grained]. Thus, it is also dangerous to allow WebRTC functionality from HTTPS origins that embed mixed content. Note: this issue is not restricted to PAGES which contain mixed content. If a page from a given origin ever loads mixed content then it is possible for a network 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.

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. ICE [RFC5245] 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, so some other mechanism is needed as described in [I-D.muthu-behave-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]. 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.).

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 or 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 is 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 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 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.

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 [RFC4347] and DTLS-SRTP [RFC5763]) is well understood. However, we must examine this technology to 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 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 [crantor-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 close to 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 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 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, OpenID, WebFinger), 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., I must be able to verify that the person I am calling has engaged a secure media mode. 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.

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

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.

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 Westerland.

7. 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-muthu.
- o Added a section about "malicious peers".
- o Added a section describing screen sharing threats.
- o Assorted editorial changes.

8. References

8.1. Normative References

- [I-D.ietf-rtcweb-overview]
Alvestrand, H., "Overview: Real Time Protocols for Brower-based Applications", draft-ietf-rtcweb-overview-06 (work in progress), February 2013.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate

Requirement Levels", BCP 14, RFC 2119, March 1997.

8.2. Informative References

- [CORS] van Kesteren, A., "Cross-Origin Resource Sharing".
- [I-D.ietf-avtc core-6222bis]
Begen, A., Perkins, C., Wing, D., and E. Rescorla,
"Guidelines for Choosing RTP Control Protocol (RTCP)
Canonical Names (CNAMES)", draft-ietf-avtc core-6222bis-06
(work in progress), July 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.kaufman-rtcweb-security-ui]
Kaufman, M., "Client Security User Interface Requirements
for RTCWEB", draft-kaufman-rtcweb-security-ui-00 (work in
progress), June 2011.
- [I-D.muthu-behave-consent-freshness]
Perumal, M., Wing, D., R, R., and H. Kaplan, "STUN Usage
for Consent Freshness",
draft-muthu-behave-consent-freshness-03 (work in
progress), February 2013.
- [RFC2818] Rescorla, E., "HTTP Over TLS", RFC 2818, May 2000.
- [RFC3261] Rosenberg, J., Schulzrinne, H., Camarillo, G., Johnston,
A., Peterson, J., Sparks, R., Handley, M., and E.
Schooler, "SIP: Session Initiation Protocol", RFC 3261,
June 2002.
- [RFC3552] Rescorla, E. and B. Korver, "Guidelines for Writing RFC
Text on Security Considerations", BCP 72, RFC 3552,
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.
- [RFC3760] Gustafson, D., Just, M., and M. Nystrom, "Securely
Available Credentials (SACRED) - Credential Server
Framework", RFC 3760, April 2004.
- [RFC4251] Ylonen, T. and C. Lonvick, "The Secure Shell (SSH)

Protocol Architecture", RFC 4251, January 2006.

- [RFC4347] Rescorla, E. and N. Modadugu, "Datagram Transport Layer Security", RFC 4347, April 2006.
- [RFC4568] Andreasen, F., Baugher, M., and D. Wing, "Session Description Protocol (SDP) Security Descriptions for Media Streams", RFC 4568, July 2006.
- [RFC5245] Rosenberg, J., "Interactive Connectivity Establishment (ICE): A Protocol for Network Address Translator (NAT) Traversal for Offer/Answer Protocols", RFC 5245, April 2010.
- [RFC5479] Wing, D., Fries, S., Tschofenig, H., and F. Audet, "Requirements and Analysis of Media Security Management Protocols", RFC 5479, April 2009.
- [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, May 2010.
- [RFC6189] Zimmermann, P., Johnston, A., and J. Callas, "ZRTP: Media Path Key Agreement for Unicast Secure RTP", RFC 6189, April 2011.
- [RFC6222] Begen, A., Perkins, C., and D. Wing, "Guidelines for Choosing RTP Control Protocol (RTCP) Canonical Names (CNAMES)", RFC 6222, April 2011.
- [RFC6454] Barth, A., "The Web Origin Concept", RFC 6454, December 2011.
- [RFC6455] Fette, I. and A. Melnikov, "The WebSocket Protocol", RFC 6455, December 2011.
- [SWF] Adobe, "SWF File Format Specification Version 19".
- [abarth-rtcweb] Barth, A., "Prompting the user is security failure", RTC-Web Workshop.
- [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.

[farus-conversion]

Farrus, M., Erro, D., and J. Hernando, "Speaker Recognition Robustness to Voice Conversion".

[finer-grained]

Barth, A. and C. Jackson, "Beware of Finer-Grained Origins", W2SP, 2008.

[huang-w2sp]

Huang, L-S., Chen, E., Barth, A., Rescorla, E., and C. Jackson, "Talking to Yourself for Fun and Profit", W2SP, 2011.

[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.

[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.

Author's Address

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

Phone: +1 650 678 2350
Email: ekr@rtfm.com

RTCWEB
Internet-Draft
Intended status: Standards Track
Expires: January 15, 2014

E. Rescorla
RTFM, Inc.
July 14, 2013

WebRTC Security Architecture
draft-ietf-rtcweb-security-arch-07

Abstract

The Real-Time Communications on the Web (RTCWEB) working group is tasked with standardizing protocols for enabling real-time communications within user-agents using web technologies (commonly called "WebRTC"). This document defines the security architecture for

Legal

THIS DOCUMENT AND THE INFORMATION CONTAINED THEREIN ARE PROVIDED ON AN "AS IS" BASIS AND THE CONTRIBUTOR, THE ORGANIZATION HE/SHE REPRESENTS OR IS SPONSORED BY (IF ANY), THE INTERNET SOCIETY, THE IETF TRUST, AND THE INTERNET ENGINEERING TASK FORCE, DISCLAIM ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO ANY WARRANTY THAT THE USE OF THE INFORMATION THEREIN WILL NOT INFRINGE ANY RIGHTS OR ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

Status of this Memo

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

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

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

This Internet-Draft will expire on January 15, 2014.

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.

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	5
2. Terminology	6
3. Trust Model	6
3.1. Authenticated Entities	7
3.2. Unauthenticated Entities	7
4. Overview	7
4.1. Initial Signaling	10
4.2. Media Consent Verification	12
4.3. DTLS Handshake	13
4.4. Communications and Consent Freshness	13
5. Detailed Technical Description	14
5.1. Origin and Web Security Issues	14
5.2. Device Permissions Model	14
5.3. Communications Consent	16
5.4. IP Location Privacy	17
5.5. Communications Security	18
5.6. Web-Based Peer Authentication	19
5.6.1. Trust Relationships: IdPs, APs, and RPs	20
5.6.2. Overview of Operation	22
5.6.3. Items for Standardization	23
5.6.4. Binding Identity Assertions to JSEP Offer/Answer Transactions	23
5.6.4.1. Input to Assertion Generation Process	23
5.6.4.2. Carrying Identity Assertions	24
5.6.5. IdP Interaction Details	25
5.6.5.1. General Message Structure	25
5.6.5.2. IdP Proxy Setup	26
5.7. Security Considerations	30
5.7.1. Communications Security	30
5.7.2. Privacy	31
5.7.3. Denial of Service	32
5.7.4. IdP Authentication Mechanism	33
5.7.4.1. PeerConnection Origin Check	33
5.7.4.2. IdP Well-known URI	34
5.7.4.3. Privacy of IdP-generated identities and the hosting site	34
5.7.4.4. Security of Third-Party IdPs	35
5.7.4.5. Web Security Feature Interactions	35
5.8. IANA Considerations	35
6. Acknowledgements	36
7. Changes	36
7.1. Changes since -06	36
7.2. Changes since -05	36
7.3. Changes since -03	36
7.4. Changes since -03	36
7.5. Changes since -02	37

8. References	37
8.1. Normative References	37
8.2. Informative References	38
Appendix A. Example IdP Bindings to Specific Protocols	39
A.1. BrowserID	39
A.2. OAuth	42
Author's Address	43

1. Introduction

The Real-Time Communications on the Web (WebRTC) working group is tasked with standardizing protocols for real-time communications between Web browsers. 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.

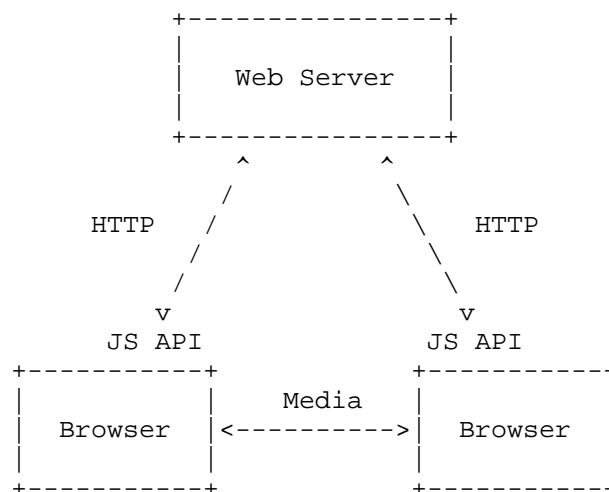


Figure 1: A simple WebRTC system

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.

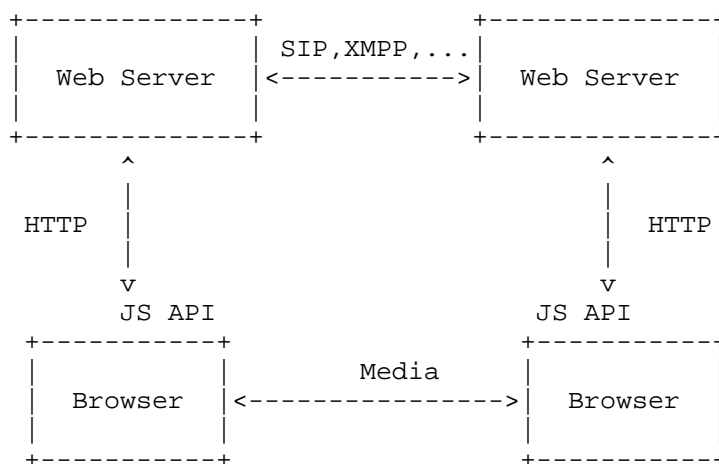


Figure 2: A multidomain WebRTC system

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", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [RFC2119].

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 are partly trusted--though to the minimum extent necessary--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.evil.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.

It's also worth noting that there are settings where authentication is non-cryptographic, such as other machines behind a firewall. Naturally, the level of trust one can have in identities verified in this way depends on how strong the topology enforcement is.

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 RTCWeb 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 or BrowserID 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 5.6.1 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.

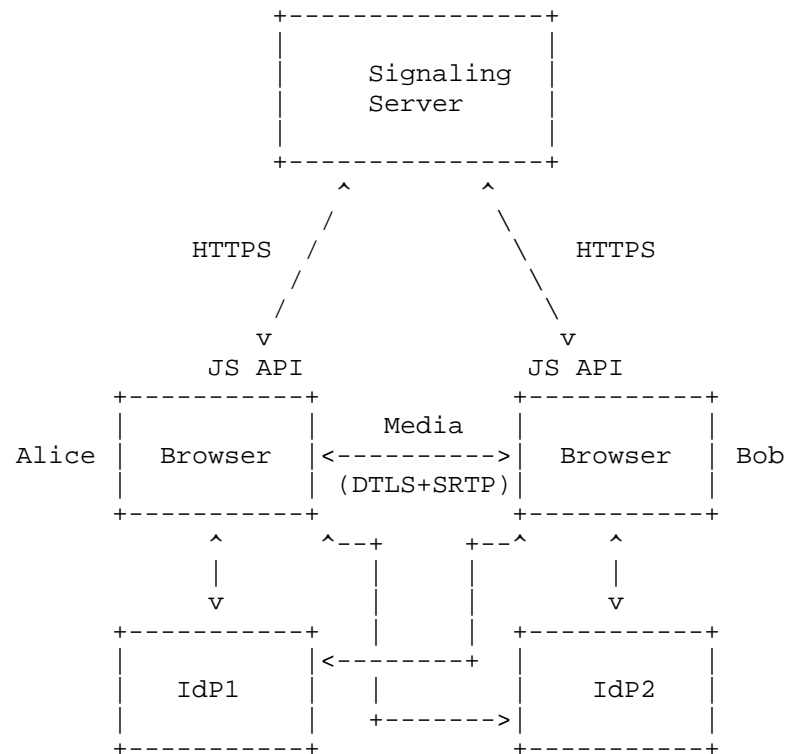


Figure 3: A call with IdP-based identity

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.

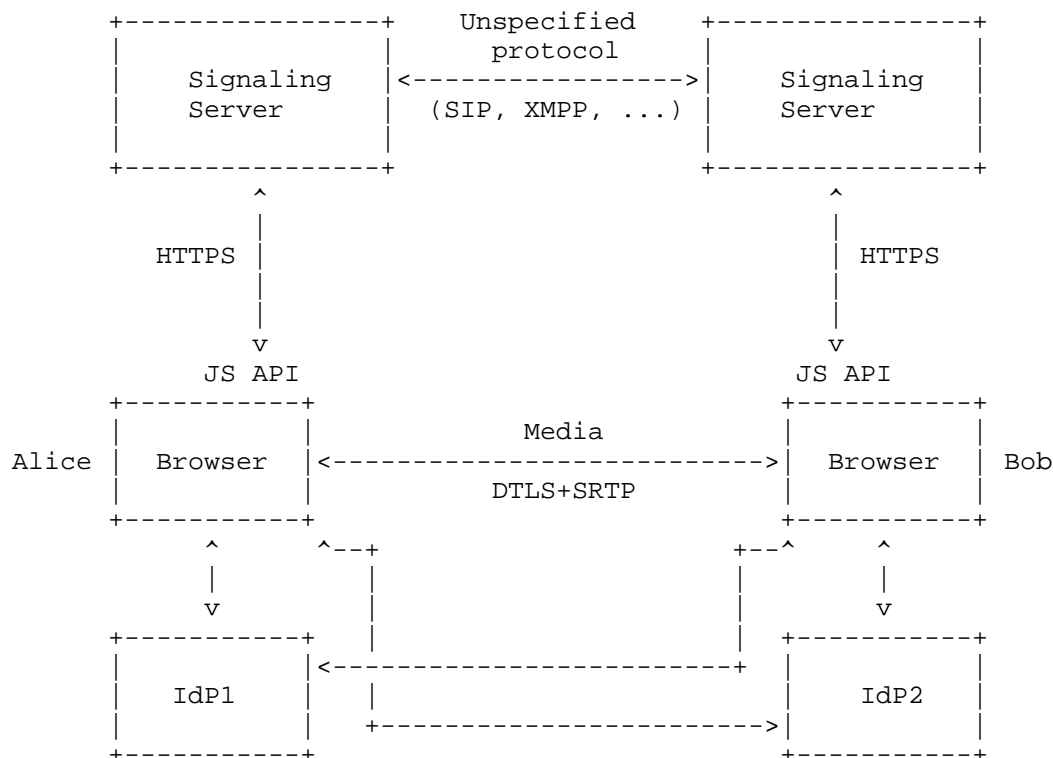


Figure 4: A federated call with IdP-based identity

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 till 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 in technologies such (BrowserID, 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) [RFC5245] 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 5.6 `PeerConnection` can be agnostic to them), but for now it's easiest to think of as a `BrowserID` assertion. 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], preferably 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. 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/calllee'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 ICE checks have completed [more specifically, once some ICE checks have completed], Alice and Bob can set up a secure channel or channels. This is performed via DTLS [RFC4347] (for the data channel) and DTLS-SRTP [RFC5763] keying for SRTP [RFC3711] for the media channel and SCTP over DTLS [I-D.ietf-tsvwg-sctp-dtls-encaps] for data channels. Specifically, Alice and Bob perform a DTLS handshake on every channel 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 RTCWeb implementations to periodically send keepalives. As described in Section 5.3, these keepalives MUST be based on the consent freshness mechanism specified in [I-D.muthu-behave-consent-freshness]. If a keepalive fails and no new ICE channels can be established, then the session is terminated.

5. Detailed Technical Description

5.1. 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.

5.2. 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 NOT** allow the setting of permanent access permissions for HTTP origins. Implementations **MAY** also opt to refuse all permissions grants for HTTP origins, but it is **RECOMMENDED** that currently they support one-time camera/microphone access.

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). E.g., "Call customerservice@ford.com". 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 an X.509 certificate, 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 indicate which of these forms of permissions it is requesting. This allows the browser client to know what sort of user interface experience to provide to the user, including what permissions to request from the user and hence what to enforce later. For instance, browsers might display a non-invasive door hanger ("some features of this site may not work..." when asking for long-term permissions) but a more invasive UI ("here is your own video") for single-call permissions. The API MAY grant weaker permissions than the JS asked for if the user chooses to authorize only those permissions, but if it intends to grant stronger ones it SHOULD display the appropriate UI for those permissions and MUST clearly indicate what permissions are being requested.

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 conversation.

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.]

[[OPEN ISSUE: This section does not have WG consensus. Because screen/application sharing presents a more significant risk than

camera and microphone access (see the discussion in [I-D.ietf-rtcweb-security] S 4.1.1), we require a higher level of user consent.

- 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.

-- END OF OPEN ISSUE]]

Clients 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. (though see Section 5.3 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 5.5 provides more on this.

5.3. 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 [RFC5245].

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, that ICE [RFC5245]; Section 10 keepalives STUN Binding Indications 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 [I-D.muthu-behave-consent-freshness].

5.4. 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., Torbutton [<https://www.torproject.org/torbutton/>]) 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.

5.5. Communications Security

Implementations MUST implement SRTP [RFC3711]. Implementations MUST implement DTLS [RFC4347] and DTLS-SRTP [RFC5763][RFC5764] for SRTP keying. Implementations MUST implement [I-D.ietf-tsvwg-sctp-dtls-encaps].

All media channels MUST be secured via SRTP. Media traffic MUST NOT be sent over plain (unencrypted) RTP. DTLS-SRTP MUST be offered for every media channel and MUST be the default; i.e., if an implementation receives an offer for DTLS-SRTP and SDES, DTLS-SRTP MUST be selected.

All data channels MUST be secured via DTLS.

[OPEN ISSUE: What should the settings be here? MUST?]
Implementations MAY support SDES for media traffic for backward compatibility purposes.

API Requirement: The API MUST provide a mechanism to indicate that a fresh DTLS key pair is to be generated for a specific call. This is intended to allow for unlinkability. Note that there are also settings where it is attractive to use the same keying material repeatedly, especially those with key continuity-based authentication. 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.
- * The "security characteristics" MUST include an indication as to whether the cryptographic keys were delivered out-of-band (from a server) or were generated as a result of a pairwise negotiation.
- * If the far endpoint was directly verified, either via a third-party verifiable X.509 certificate or via a Web IdP mechanism (see Section 5.6) 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" or "Null Cipher".) However, if Null ciphers are used, that MUST be presented to the user at the top-level UI.
- * 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 an SAS.

5.6. 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 only 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,

BrowserID, 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 your identity. For instance, if I have an account at example.org, I could use the example.org identity provider to prove to others that I was alice@example.org. The development of these technologies allows us to separate calling from identity provision: I could call you on Poker Galaxy but identify myself 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 securely generate the input to the IdP assertion process and MUST securely 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.

5.6.1. 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 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.

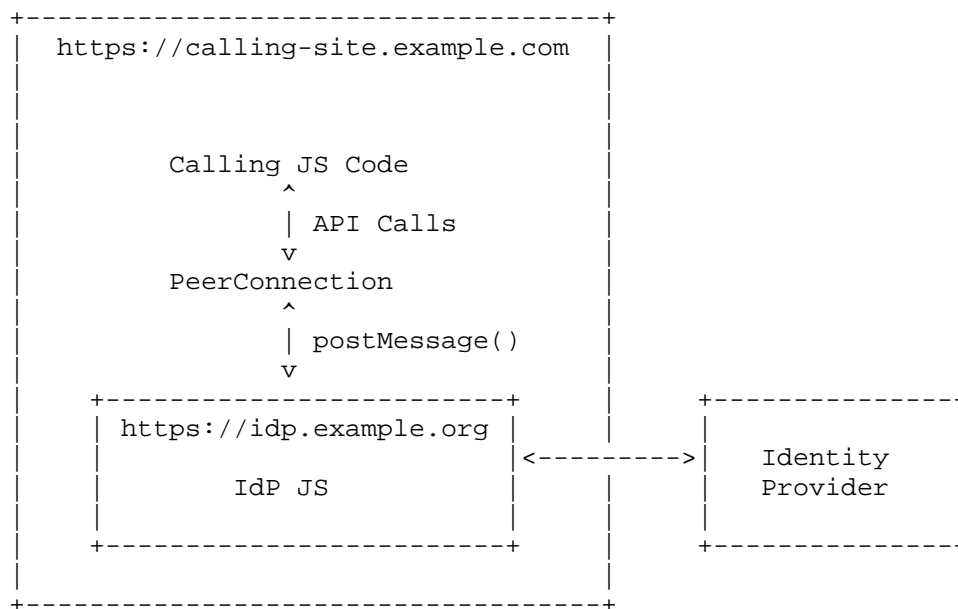
5.6.2. Overview of Operation

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 with its source at the IdP. This allows the IdP to load whatever JS is necessary into the proxy, which runs in the IdP's

- security context.
2. If the user is not already logged in, the IdP does whatever is required to log them in, such as soliciting a username and password.
 3. Once the user is logged in, the IdP proxy notifies the browser that it is ready.
 4. The browser and the IdP proxy communicate via a standardized series of messages delivered via `postMessage`. For instance, the browser might request the IdP proxy to sign or verify a given identity assertion.

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--which is deterministic from the IdP's identity--and the generic protocol for requesting and verifying 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.

5.6.3. Items for Standardization

In order to make this work, we must standardize the following items:

- 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. Section 5.6.4
- o The interface to the IdP. Section 5.6.5 specifies a specific protocol mechanism which allows the use of any identity protocol without requiring specific further protocol support in the browser
- o The JavaScript interfaces which the calling application can use to specify the IdP to use to generate assertions and to discover what assertions were received.

The first two items are defined in this document. The final one is defined in the companion W3C WebRTC API specification [webrtc-api].

5.6.4. Binding Identity Assertions to JSEP Offer/Answer Transactions

5.6.4.1. Input to Assertion Generation Process

As discussed above, an identity assertion binds the user's identity (as asserted by the IdP) to the JSEP offer/exchange transaction 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 in a JSON structure for extensibility, as shown below:


```

{
  "fingerprint" :
  {
    "algorithm": "SHA-1",
    "digest": "4A:AD:B9:B1:3F:...:E5:7C:AB"
  }
}

```

The "algorithm" and digest values correspond directly to the algorithm and digest values in the a=fingerprint line of the SDP. [RFC4572].

Note: 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.

5.6.4.2. Carrying Identity Assertions

Once an IdP has generated an assertion, it is attached to the JSEP message. This is done by adding a new a-line to the SDP, of the form a=identity. The sole contents of this value are a base-64-encoded version of the identity assertion. For example:

```

v=0
o=- 1181923068 1181923196 IN IP4 ual.example.com
s=example1
c=IN IP4 ual.example.com
a=setup:actpass
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: \
  ImlkcCI6eyJkb2lhaW4iOiAiZXhhbXBsZS5vcmcilCAicHJvdG9jb2wiOiAiYm9n \
  dXMifSwiYXNzZXJ0aW9uIjpcIntcImlkZW50aXR5XCI6XCJib2JAZXhhbXBsZS5v \
  cmdcIixcImNvbnRlbnRzXCI6XCJhYmNkZWZnaGlqa2xtbm9wcXJzdHV2d3l6XCIs \
  XCJzaWduYXRlcmVcIjpcIjAxMDIwMzA0MDUwNlwiSJ9Cg==
t=0 0
m=audio 6056 RTP/SAVP 0
a=sendrecv

```

Each identity attribute should be paired (and attests to) with an a=fingerprint attribute and therefore can exist either at the session or media level. Multiple identity attributes may appear at either level, though it is RECOMMENDED that implementations not do this, because it becomes very unclear what security claim that they are making and the UI guidelines above become unclear. Browsers MAY

choose refuse to display any identity indicators in the face of multiple identity attributes with different identities but SHOULD process multiple attributes with the same identity as described above.

5.6.5. IdP Interaction Details

5.6.5.1. General Message Structure

Messages between the PeerConnection object and the IdP proxy are formatted using JSON [RFC4627]. For instance, the PeerConnection would request a signature with the following "SIGN" message:

```
{
  "type": "SIGN",
  "id": "1",
  "origin": "https://calling-site.example.com",
  "message": "012345678abcdefghijkl"
}
```

All messages MUST contain a "type" field which indicates the general meaning of the message.

All requests from the PeerConnection object MUST contain an "id" field which MUST be unique for that PeerConnection object. Any responses from the IdP proxy MUST contain the same id in response, which allows the PeerConnection to correlate requests and responses, in case there are multiple requests/responses outstanding to the same proxy.

All requests from the PeerConnection object MUST contain an "origin" field containing the origin of the JS which initiated the PC (i.e., the URL of the calling site). This origin value can be used by the IdP to make access control decisions. For instance, an IdP might only issue identity assertions for certain calling services in the same way that some IdPs require that relying Web sites have an API key before learning user identity.

Any message-specific data is carried in a "message" field. Depending on the message type, this may either be a string or a richer JSON object.

5.6.5.1.1. Errors

If an error occurs, the IdP sends a message of type "ERROR". The message MAY have an "error" field containing freeform text data which containing additional information about what happened. For instance:

```
{
  "id": "1",
  "type": "ERROR",
  "error": "Signature verification failed"
}
```

Figure 5: Example error

5.6.5.2. IdP Proxy Setup

In order to perform an identity transaction, the PeerConnection must first create an IdP proxy. While the details of this are specified in the W3C API document, from the perspective of this specification, however, the relevant facts are:

- o The JS runs in the IdP's security context with the base page retrieved from the URL specified in Section 5.6.5.2.1
- o The usual browser sandbox isolation mechanisms MUST be enforced with respect to the IdP proxy.
- o JS running in the IdP proxy MUST be able to send and receive messages to the PeerConnection and the PC and IdP proxy are able to verify the source and destination of these messages.

Initially the IdP proxy is in an unready state; the IdP JS must be loaded and there may be several round trips to the IdP server, for instance to log the user in. When the IdP proxy is ready to receive commands, it delivers a "ready" message. As this message is unsolicited, it simply contains:

```
{ "type": "READY" }
```

Once the PeerConnection object receives the ready message, it can send commands to the IdP proxy.

5.6.5.2.1. 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:

domain name: The IdP's domain name

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

Each IdP MUST serve its initial entry page (i.e., the one loaded by

the IdP proxy) from the well-known URI specified in `"/.well-known/idp-proxy/<protocol>"` on the IdP's web site. This URI MUST be loaded via HTTPS [RFC2818]. For example, for the IdP `"identity.example.com"` and the protocol `"example"`, the URL would be:

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

5.6.5.2.1.1. 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"

5.6.5.2.1.2. 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, 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 5.6.5.2.3.1.

5.6.5.2.2. Requesting Assertions

In order to request an assertion, the PeerConnection sends a "SIGN" message. Aside from the mandatory fields, this message has a "message" field containing a string. The contents of this string are defined above, but are opaque from the perspective of the IdP.

A successful response to a "SIGN" message contains a message field which is a JS dictionary consisting of two fields:

idp: A dictionary containing the domain name of the provider and the protocol string
assertion: An opaque field containing the assertion itself. This is only interpretable by the idp or its proxy.

Figure 6 shows an example transaction, with the message `"abcde..."` (remember, the messages are opaque at this layer) being signed and bound to identity `"ekr@example.org"`. 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. Line breaks are inserted solely for readability.

```
PeerConnection -> IdP proxy:
{
  "type": "SIGN",
  "id": 1,
  "origin": "https://calling-service.example.com/",
  "message": "abcdefghijklmnopqrstuvwyz"
}

IdPProxy -> PeerConnection:
{
  "type": "SUCCESS",
  "id": 1,
  "message": {
    "idp": {
      "domain": "example.org"
      "protocol": "bogus"
    },
    "assertion": "\\{"identity\\":\\"bob@example.org\\",
                  \\\"contents\\":\\"abcdefghijklmnopqrstuvwyz\\",
                  \\\"request_origin\\":\\"rtcweb://peerconnection\\",
                  \\\"signature\\":\\"010203040506\\"}"
  }
}
```

Figure 6: Example assertion request

The message structure is serialized, base64-encoded, and placed in an `a=identity` attribute.

5.6.5.2.3. Verifying Assertions

In order to verify an assertion, an RP sends a "VERIFY" message to the IdP proxy containing the assertion supplied by the AP in the "message" field.

The IdP proxy verifies the assertion. Depending on the identity protocol, this may require one or more round trips to the IdP. For instance, an OAuth-based protocol will likely require using the IdP as an oracle, whereas with BrowserID the IdP proxy can likely verify the signature on the assertion without contacting the IdP, provided that it has cached the IdP's public key.

Regardless of the mechanism, if verification succeeds, a successful response from the IdP proxy MUST contain a message field consisting of a dictionary/hash with the following fields:

identity The identity of the AP from the IdP's perspective. Details of this are provided in Section 5.6.5.2.3.1

contents The original unmodified string provided by the AP in the original SIGN request.

request_origin The original origin of the SIGN request on the AP side as determined by the origin of the PostMessage call. The IdP MUST somehow arrange to propagate this information as part of the assertion. The receiving PeerConnection MUST verify that this value is "rtcweb://peerconnection" (which implies that PeerConnection must arrange that its messages to the IdP proxy are from this origin.) See Section 5.7.4.1 for the security purpose of this field. [[OPEN ISSUE: Can a URI person help make a better URI.]]

Figure 7 shows an example transaction. Line breaks are inserted solely for readability.

```
PeerConnection -> IdP Proxy:
{
  "type": "VERIFY",
  "id": 2,
  "origin": "https://calling-service.example.com/",
  "message": {
    "identity": "bob@example.org",
    "contents": "abcdefghijklmnopqrstuvwyz",
    "request_origin": "rtcweb://peerconnection",
    "signature": "010203040506"
  }
}

IdP Proxy -> PeerConnection:
{
  "type": "SUCCESS",
  "id": 2,
  "message": {
    "identity": {
      "name": "bob@example.org",
      "displayname": "Bob"
    },
    "request_origin": "rtcweb://peerconnection",
    "contents": "abcdefghijklmnopqrstuvwyz"
  }
}
```

Figure 7: Example verification request

5.6.5.2.3.1. Identity Formats

Identities passed from the IdP proxy to the PeerConnection are structured as JSON dictionaries with one mandatory field: "name". This field MUST consist of an RFC822-formatted string representing the user's identity. [[OPEN ISSUE: Would it be better to have a typed field?]] The PeerConnection API MUST check this string as follows:

1. If the RHS 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.
2. If the RHS of the string is not equal to the domain name of the IdP proxy, then the PeerConnection object MUST reject the assertion unless (a) the IdP domain is listed as an acceptable third-party IdP and (b) local policy is configured to trust this IdP domain for the RHS of the identity string.

Sites which have identities that do not fit into the RFC822 style (for instance, Facebook ids are simple numeric values) SHOULD convert them to this form by appending their IdP domain (e.g., 12345@identity.facebook.com), thus ensuring that they are authoritative for the identity.

The IdP proxy MAY also include a "displayname" field which contains a more user-friendly identity assertion. Browsers SHOULD take care in the UI to distinguish the "name" assertion which is verifiable directly from the "displayname" which cannot be verified and thus relies on trust in the IdP. In future, we may define other fields to allow the IdP to provide more information to the browser. [[OPEN ISSUE: Should this field exist? Is it confusing?]]

5.7. 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.

5.7.1. Communications Security

While this document favors DTLS-SRTP, it permits a variety of communications security mechanisms and thus the level of communications security actually provided varies considerably. Any

pair of implementations which have multiple security mechanisms in common are subject to being downgraded to the weakest of those common mechanisms by any attacker who can modify the signaling traffic. If communications are over HTTP, this means any on-path attacker. If communications are over HTTPS, this means the signaling server. Implementations which wish to avoid downgrade attack should only offer the strongest available mechanism, which is DTLS/DTLS-SRTP. Note that the implication of this choice will be that interop to non-DTLS-SRTP devices will need to happen through gateways.

Even if only DTLS/DTLS-SRTP are 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 5.5 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 5.6 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 your 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 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.

5.7.2. 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. Additionally, browsers SHOULD implement the privacy-preserving CNAME generation mode of [I-D.ietf-avtcore-6222bis].

5.7.3. 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 RTCWeb. 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).

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.

5.7.4. 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.

5.7.4.1. 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. In order to prevent this attack, we require that all signatures be tied to a specific origin ("rtcweb://...") which cannot be produced by content JavaScript. Thus, while an attacker can instantiate the IdP proxy, they cannot send messages from an

appropriate origin and so cannot create acceptable assertions. I.e., the assertion request must have come from the browser. This origin check is enforced on the relying party side, not on the authenticating party side. The reason for this is to take the burden of knowing which origins are valid off of the IdP, thus making this mechanism extensible to other applications besides WebRTC. The IdP simply needs to gather the origin information (from the posted message) and attach it to the assertion.

Note that although this origin check is enforced on the RP side and not at the IdP, it is absolutely imperative that it be done. The mechanisms in this document rely on the browser enforcing access restrictions on the DTLS keys and assertion requests which do not come with the right origin may be from content JS rather than from browsers, and therefore those access restrictions cannot be assumed.

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. However, attaching one's identity to a key that the user does not control does not appear to provide substantial leverage to an attacker, so a proof of possession is omitted for simplicity.

5.7.4.2. IdP Well-known URI

As described in Section 5.6.5.2.1 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.

5.7.4.3. 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 "origin" field of the signature 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.

5.7.4.4. 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.

5.7.4.5. Web Security Feature Interactions

A number of optional Web security features have the potential to cause issues for this mechanism, as discussed below.

5.7.4.5.1. Popup Blocking

If the user is not already logged into the IdP, the IdP proxy may need to pop up a top level window in order to prompt the user for their authentication information (it is bad practice to do this in an IFRAME inside the window because then users have no way to determine the destination for their password). If the user's browser is configured to prevent popups, this may fail (depending on the exact algorithm that the popup blocker uses to suppress popups). It may be necessary to provide a standardized mechanism to allow the IdP proxy to request popping of a login window. Note that care must be taken here to avoid PeerConnection becoming a general escape hatch from popup blocking. One possibility would be to only allow popups when the user has explicitly registered a given IdP as one of theirs (this is only relevant at the AP side in any case).

5.7.4.5.2. 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.

5.8. IANA Considerations

[TODO: IANA registration for Identity header. Or should this be in MMUSIC?]

6. 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 5.5.

7. Changes

7.1. 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.

7.2. 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.

7.3. 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.

7.4. Changes since -03

7.5. 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

8. References

8.1. Normative References

- [I-D.ietf-avtcore-6222bis]
Begen, A., Perkins, C., Wing, D., and E. Rescorla,
"Guidelines for Choosing RTP Control Protocol (RTCP)
Canonical Names (CNAMEs)", draft-ietf-avtcore-6222bis-06
(work in progress), July 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-tsvwg-sctp-dtls-encaps]
Jesup, R., Loreto, S., Stewart, R., and M. Tuexen, "DTLS
Encapsulation of SCTP Packets for RTCWEB",
draft-ietf-tsvwg-sctp-dtls-encaps-00 (work in progress),
February 2013.
- [I-D.muthu-behave-consent-freshness]
Perumal, M., Wing, D., R, R., and H. Kaplan, "STUN Usage
for Consent Freshness",
draft-muthu-behave-consent-freshness-03 (work in
progress), February 2013.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate
Requirement Levels", BCP 14, RFC 2119, March 1997.
- [RFC2818] Rescorla, E., "HTTP Over TLS", RFC 2818, May 2000.
- [RFC3711] Baugher, M., McGrew, D., Naslund, M., Carrara, E., and K.
Norrman, "The Secure Real-time Transport Protocol (SRTP)",
RFC 3711, March 2004.

- [RFC4347] Rescorla, E. and N. Modadugu, "Datagram Transport Layer Security", RFC 4347, April 2006.
- [RFC4572] Lennox, J., "Connection-Oriented Media Transport over the Transport Layer Security (TLS) Protocol in the Session Description Protocol (SDP)", RFC 4572, July 2006.
- [RFC4627] Crockford, D., "The application/json Media Type for JavaScript Object Notation (JSON)", RFC 4627, July 2006.
- [RFC5245] Rosenberg, J., "Interactive Connectivity Establishment (ICE): A Protocol for Network Address Translator (NAT) Traversal for Offer/Answer Protocols", RFC 5245, April 2010.
- [RFC5246] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", RFC 5246, August 2008.
- [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, May 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.
- [RFC6454] Barth, A., "The Web Origin Concept", RFC 6454, December 2011.
- [webcrypto]
Dahl, Sleevi, "Web Cryptography API", June 2013.

Available at <http://www.w3.org/TR/WebCryptoAPI/>
- [webrtc-api]
Bergkvist, Burnett, Jennings, Narayanan, "WebRTC 1.0: Real-time Communication Between Browsers", October 2011.

Available at
<http://dev.w3.org/2011/webrtc/editor/webrtc.html>

8.2. Informative References

- [I-D.ietf-rtcweb-jsep]
Uberti, J. and C. Jennings, "Javascript Session Establishment Protocol", draft-ietf-rtcweb-jsep-03 (work in progress), February 2013.

- [I-D.jennings-rtcweb-signaling]
Jennings, C., Rosenberg, J., and R. Jesup, "RTCWeb Offer/Answer Protocol (ROAP)",
draft-jennings-rtcweb-signaling-01 (work in progress),
October 2011.
- [I-D.kaufman-rtcweb-security-ui]
Kaufman, M., "Client Security User Interface Requirements for RTCWEB", draft-kaufman-rtcweb-security-ui-00 (work in progress), June 2011.
- [RFC3261] Rosenberg, J., Schulzrinne, H., Camarillo, G., Johnston, A., Peterson, J., Sparks, R., Handley, M., and E. Schooler, "SIP: Session Initiation Protocol", RFC 3261, June 2002.
- [RFC5705] Rescorla, E., "Keying Material Exporters for Transport Layer Security (TLS)", RFC 5705, March 2010.
- [RFC6455] Fette, I. and A. Melnikov, "The WebSocket Protocol", RFC 6455, December 2011.
- [XmlHttpRequest]
van Kesteren, A., "XMLHttpRequest Level 2".

Appendix A. Example IdP Bindings to Specific Protocols

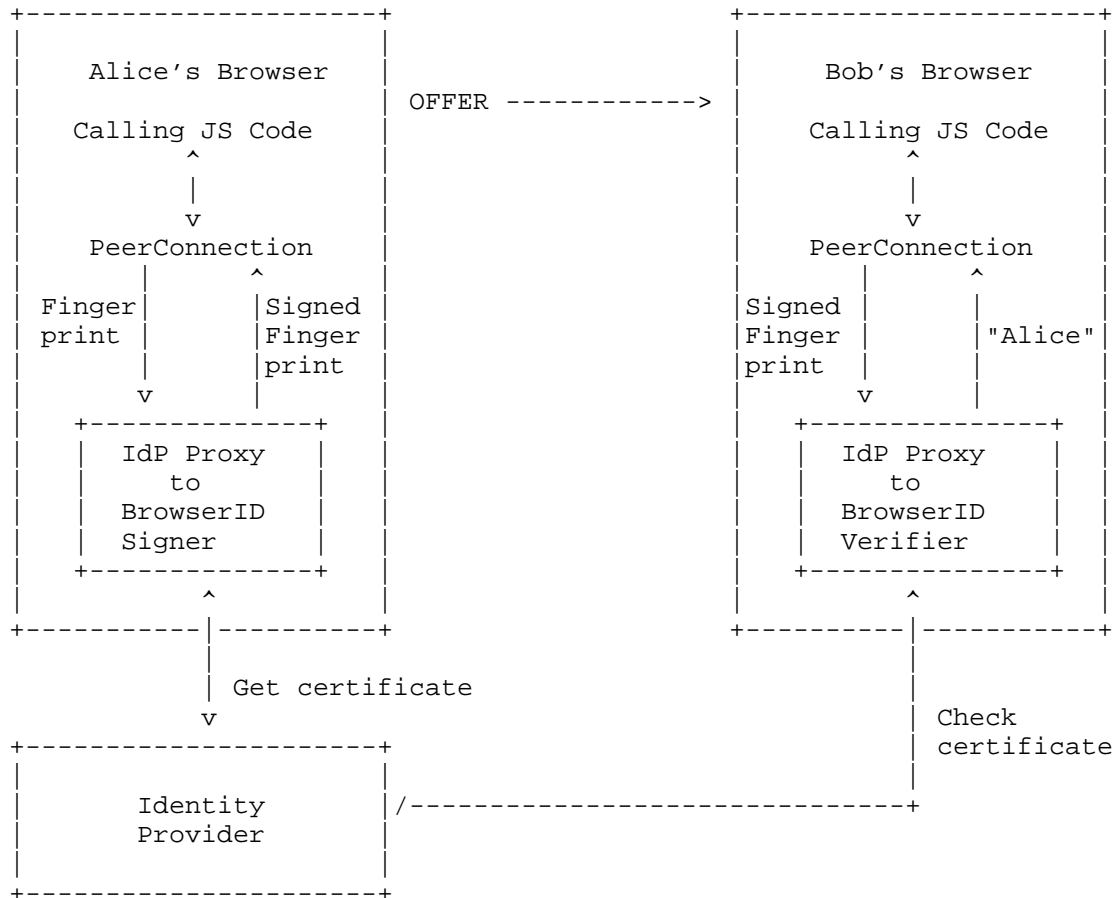
[[TODO: These still need some cleanup.]]

This section provides some examples of how the mechanisms described in this document could be used with existing authentication protocols such as BrowserID or OAuth. Note that this does not require browser-level support for either protocol. Rather, the protocols can be fit into the generic framework. (Though BrowserID in particular works better with some client side support).

A.1. BrowserID

BrowserID [<https://browserid.org/>] is a technology which allows a user with a verified email address to generate an assertion (authenticated by their identity provider) attesting to their identity (phrased as an email address). The way that this is used in practice is that the relying party embeds JS in their site which talks to the BrowserID code (either hosted on a trusted intermediary or embedded in the browser). That code generates the assertion which is passed back to the relying party for verification. The assertion can be verified directly or with a Web service provided by the

identity provider. It's relatively easy to extend this functionality to authenticate WebRTC calls, as shown below.



The way this mechanism works is as follows. On Alice's side, Alice goes to initiate a call.

1. The calling JS instantiates a PeerConnection and tells it that it is interested in having it authenticated via BrowserID (i.e., it provides "browserid.org" as the IdP name.)
2. The PeerConnection instantiates the BrowserID signer in the IdP proxy
3. The BrowserID signer contacts Alice's identity provider, authenticating as Alice (likely via a cookie).
4. The identity provider returns a short-term certificate attesting to Alice's identity and her short-term public key.

5. The Browser-ID code signs the fingerprint and returns the signed assertion + certificate to the PeerConnection.
6. The PeerConnection returns the signed information to the calling JS code.
7. The signed assertion gets sent over the wire to Bob's browser (via the signaling service) as part of the call setup.

The offer might look something like:

```
{
  "type": "OFFER",
  "sdp":
    "v=0\n
    o=- 2890844526 2890842807 IN IP4 192.0.2.1\n
    s= \n
    c=IN IP4 192.0.2.1\n
    t=2873397496 2873404696\n
    m=audio 49170 RTP/AVP 0\n
    a=fingerprint: SHA-1 \n
    a=identity [[base-64 encoding of...
    4A:AD:B9:B1:3F:82:18:3B:54:02:12:DF:3E:5D:49:6B:19:E5:7C:AB\n",
    "identity": {
      "idp": { // Standardized
        "domain": "browserid.org",
        "method": "default"
      },
      "assertion": // Contents are browserid-specific
      "\"assertion\": {
        \"digest\": \"<hash of the contents from the browser>\",
        \"audience\": \"[TBD]\",
        \"valid-until\": 1308859352261,
      },
      \"certificate\": {
        \"email\": \"rescorla@example.org\",
        \"public-key\": \"<ekrs-public-key>\",
        \"valid-until\": 1308860561861,
      }" // certificate is signed by example.org
    }]]"
}
```

Note that while the IdP here is specified as "browserid.org", the actual certificate is signed by example.org. This is because BrowserID is a combined authoritative/third-party system in which browserid.org delegates the right to be authoritative (what BrowserID calls primary) to individual domains.

On Bob's side, he receives the signed assertion as part of the call setup message and a similar procedure happens to verify it.

1. The calling JS instantiates a PeerConnection and provides it the relevant signaling information, including the signed assertion.
2. The PeerConnection instantiates the IdP proxy which examines the IdP name and brings up the BrowserID verification code.
3. The BrowserID verifier contacts the identity provider to verify the certificate and then uses the key to verify the signed fingerprint.
4. Alice's verified identity is returned to the PeerConnection (it already has the fingerprint).
5. At this point, Bob's browser can display a trusted UI indication that Alice is on the other end of the call.

When Bob returns his answer, he follows the converse procedure, which provides Alice with a signed assertion of Bob's identity and keying material.

A.2. OAuth

While OAuth is not directly designed for user-to-user authentication, with a little lateral thinking it can be made to serve. We use the following mapping of OAuth concepts to WebRTC concepts:

OAuth	WebRTC
Client	Relying party
Resource owner	Authenticating party
Authorization server	Identity service
Resource server	Identity service

Table 1

The idea here is that when Alice wants to authenticate to Bob (i.e., for Bob to be aware that she is calling). In order to do this, she allows Bob to see a resource on the identity provider that is bound to the call, her identity, and her public key. Then Bob retrieves the resource from the identity provider, thus verifying the binding between Alice and the call.

Alice	IdP	Bob
<hr/>		
Call-Id, Fingerprint	----->	
<-----	Auth Code	
Auth Code	----->	
	<----- Get Token + Auth Code	
	Token ----->	
	<----- Get call-info	

Call-Id, Fingerprint ----->

This is a modified version of a common OAuth flow, but omits the redirects required to have the client point the resource owner to the IdP, which is acting as both the resource server and the authorization server, since Alice already has a handle to the IdP.

Above, we have referred to "Alice", but really what we mean is the PeerConnection. Specifically, the PeerConnection will instantiate an IFRAME with JS from the IdP and will use that IFRAME to communicate with the IdP, authenticating with Alice's identity (e.g., cookie). Similarly, Bob's PeerConnection instantiates an IFRAME to talk to the IdP.

Author's Address

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

Phone: +1 650 678 2350
Email: ekr@rtfm.com

Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: March 7, 2014

H. Alvestrand
Google
September 3, 2013

Transports for RTCWEB
draft-ietf-rtcweb-transports-01

Abstract

This document describes the data transport protocols used by RTCWEB, including the protocols used for interaction with intermediate boxes such as firewalls, relays and NAT boxes.

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 March 7, 2014.

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. Transport and Middlebox specification	3
2.1. System-provided interfaces	3
2.2. Middle box related functions	4
2.3. Transport protocols implemented	4
3. IANA Considerations	5
4. Security Considerations	5
5. Acknowledgements	5
6. References	5
6.1. Normative References	5
6.2. Informative References	7
Appendix A. Change log	7
A.1. Changes from -00 to -01	7
Author's Address	7

1. Introduction

The IETF RTCWEB effort, part of the WebRTC effort carried out in cooperation between the IETF and the W3C, is aimed at specifying a protocol suite that is useful for real time multimedia exchange between browsers.

The overall effort is described in the RTCWEB overview document, [I-D.ietf-rtcweb-overview]. This document focuses on the data transport protocols that are used by conforming implementations.

This protocol suite is designed for WebRTC, and intends to satisfy the security considerations described in the WebRTC security documents, [I-D.ietf-rtcweb-security] and [I-D.ietf-rtcweb-security-arch].

2. Transport and Middlebox specification

2.1. System-provided interfaces

The protocol specifications used here assume that the following protocols are available to the implementations of the RTCWEB protocols:

- o UDP. This is the protocol assumed by most protocol elements described.
- o TCP. This is used for HTTP/WebSockets, as well as for TURN/SSL and ICE-TCP.

For both protocols, this specification assumes the ability to set the DSCP code point of the sockets opened on a per-packet basis, in order to achieve the prioritizations described in [I-D.ietf-rtcweb-qos]. It does not assume that the DSCP codepoints will be honored, and does assume that they may be zeroed or changed, since this is a local configuration issue.

If DSCP code points can only be set on a per-socket basis, not per-packet, one loses the ability to have the network discriminate reliably between classes of traffic sent over the same transport, but this does not prevent communication.

This specification does not assume that the implementation will have access to ICMP or raw IP.

2.2. Middle box related functions

The primary mechanism to deal with middle boxes is ICE, which is an appropriate way to deal with NAT boxes and firewalls that accept traffic from the inside, but only from the outside if it's in response to inside traffic (simple stateful firewalls).

In order to deal with situations where both parties are behind NATs which perform endpoint-dependent mapping (as defined in [RFC5128] section 2.4), TURN [RFC5766] MUST be supported.

In order to deal with firewalls that block all UDP traffic, TURN using TCP between the client and the server MUST be supported, and TURN using TLS between the client and the server MUST be supported.

ICE TCP candidates [RFC6062] MAY be supported; this may allow applications to achieve peer-to-peer communication across UDP-blocking firewalls, but this also requires use of the SRTP/AVPF/TCP profile of RTP.

The following specifications MUST be supported:

- o ICE [RFC5245]
- o TURN, including TURN over TCP[RFC5766].

TURN over TLS over TCP MAY be supported. (QUESTION: SHOULD? MUST?)

For referring to STUN and TURN servers, this specification depends on the STUN URI, [I-D.nandakumar-rtcweb-stun-uri].

Further discussion of the interaction of RTCWEB with firewalls is contained in [I-D.hutton-rtcweb-nat-firewall-considerations]. This document makes no requirements on interacting with HTTP proxies or HTTP proxy configuration methods.

2.3. Transport protocols implemented

For data transport over the RTCWEB data channel [I-D.ietf-rtcweb-data-channel], RTCWEB implementations support SCTP over DTLS over ICE. This is specified in [I-D.ietf-tsvwg-sctp-dtls-encaps]. Negotiation of this transport in SCTP is defined in [I-D.ietf-mmusic-sctp-sdp].

The setup protocol for RTCWEB data channels is described in [I-D.jesup-rtcweb-data-protocol].

For transport of media, secure RTP is used. The details of the

profile of RTP used are described in "RTP Usage" [I-D.ietf-rtcweb-rtp-usage].

RTCWEB implementations MUST support multiplexing of DTLS and RTP over the same port pair, as described in the DTLS_SRTCP specification [RFC5764], section 5.1.2. Further separation of the DTLS traffic into SCTP and "other" is described in <need reference>.

3. IANA Considerations

This document makes no request of IANA.

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

4. Security Considerations

Security considerations are enumerated in [I-D.ietf-rtcweb-security].

5. Acknowledgements

This document is based on earlier versions embedded in [I-D.ietf-rtcweb-overview], which were the results of contributions from many RTCWEB WG members.

6. References

6.1. Normative References

[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-04 (work in progress), June 2013.

[I-D.ietf-rtcweb-data-channel]

Jesup, R., Loreto, S., and M. Tuexen, "RTCWeb Data Channels", draft-ietf-rtcweb-data-channel-05 (work in progress), July 2013.

[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-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-07 (work in progress), July 2013.
- [I-D.ietf-rtcweb-security]
Rescorla, E., "Security Considerations for WebRTC", draft-ietf-rtcweb-security-05 (work in progress), July 2013.
- [I-D.ietf-rtcweb-security-arch]
Rescorla, E., "WebRTC Security Architecture", draft-ietf-rtcweb-security-arch-07 (work in progress), July 2013.
- [I-D.ietf-tsvwg-sctp-dtls-encaps]
Jesup, R., Loreto, S., Stewart, R., and M. Tuexen, "DTLS Encapsulation of SCTP Packets", draft-ietf-tsvwg-sctp-dtls-encaps-01 (work in progress), July 2013.
- [I-D.nandakumar-rtcweb-stun-uri]
Nandakumar, S., Salgueiro, G., Jones, P., and M. Petit-Huguenin, "URI Scheme for Session Traversal Utilities for NAT (STUN) Protocol", draft-nandakumar-rtcweb-stun-uri-05 (work in progress), July 2013.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.
- [RFC5245] Rosenberg, J., "Interactive Connectivity Establishment (ICE): A Protocol for Network Address Translator (NAT) Traversal for Offer/Answer Protocols", RFC 5245, 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.
- [RFC5766] Mahy, R., Matthews, P., and J. Rosenberg, "Traversal Using Relays around NAT (TURN): Relay Extensions to Session Traversal Utilities for NAT (STUN)", RFC 5766, April 2010.
- [RFC6062] Perreault, S. and J. Rosenberg, "Traversal Using Relays around NAT (TURN) Extensions for TCP Allocations", RFC 6062, November 2010.

6.2. Informative References

- [I-D.hutton-rtcweb-nat-firewall-considerations]
Stach, T., Hutton, A., and J. Uberti, "RTCWEB Considerations for NATs, Firewalls and HTTP proxies", draft-hutton-rtcweb-nat-firewall-considerations-01 (work in progress), June 2013.
- [I-D.ietf-rtcweb-overview]
Alvestrand, H., "Overview: Real Time Protocols for Browser-based Applications", draft-ietf-rtcweb-overview-07 (work in progress), August 2013.
- [I-D.jesup-rtcweb-data-protocol]
Jesup, R., Loreto, S., and M. Tuexen, "WebRTC Data Channel Protocol", draft-jesup-rtcweb-data-protocol-04 (work in progress), February 2013.
- [RFC5128] Srisuresh, P., Ford, B., and D. Kegel, "State of Peer-to-Peer (P2P) Communication across Network Address Translators (NATs)", RFC 5128, March 2008.

Appendix A. Change log

A.1. Changes from -00 to -01

- o Clarified DSCP requirements, with reference to -qos-
- o Clarified "symmetric NAT" -> "NATs which perform endpoint-dependent mapping"
- o Made support of TURN over TCP mandatory
- o Made support of TURN over TLS a MAY, and added open question
- o Added an informative reference to -firewalls-
- o Called out that we don't make requirements on HTTP proxy interaction (yet)

Author's Address

Harald Alvestrand
Google

Email: harald@alvestrand.no

