Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: April 27, 2015

JM. Valin
Mozilla
C. Bran
Plantronics

October 24, 2014

WebRTC Audio Codec and Processing Requirements
draft-ietf-rtcweb-audio-07

Abstract

This document outlines the audio codec and processing requirements for WebRTC client application and endpoint devices.

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

An integral part of the success and adoption of the Web Real Time Communications (WebRTC) will be the voice and video interoperability between WebRTC applications. This specification will outline the audio processing and codec requirements for WebRTC client implementations.

2. Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [RFC2119].

3. Codec Requirements

To ensure a baseline level of interoperability between WebRTC clients, a minimum set of required codecs are specified below. If other suitable audio codecs are available for the browser to use, it is RECOMMENDED that they are also be included in the offer in order to maximize the possibility to establish the session without the need for audio transcoding.

WebRTC clients are REQUIRED to implement the following audio codecs:

- Opus [RFC6716] with the payload format specified in [I-D.ietf-payload-rtp-opus].

- G.711 PCMA and PCMU with the payload format specified in section 4.5.14 of [RFC3551].

- o [RFC3389] comfort noise (CN). Receivers MUST support RFC3389 CN for streams encoded with G.711 or any other supported codec that does not provide its own CN. Since Opus provides its own CN mechanism, the use of RFC3389 CN with Opus is NOT RECOMMENDED. Use of DTX/CN by senders is OPTIONAL.

- o The audio/telephone-event media format as specified in [RFC4733]. WebRTC clients are REQUIRED to be able to generate and consume the following events:

Event Code	Event Name	Reference
0	DTMF digit "0"	RFC4733
1	DTMF digit "1"	RFC4733
2	DTMF digit "2"	RFC4733
3	DTMF digit "3"	RFC4733
4	DTMF digit "4"	RFC4733
5	DTMF digit "5"	RFC4733
6	DTMF digit "6"	RFC4733
7	DTMF digit "7"	RFC4733
8	DTMF digit "8"	RFC4733
9	DTMF digit "9"	RFC4733
10	DTMF digit "*"	RFC4733
11	DTMF digit "#"	RFC4733

For all cases where the client is able to process audio at a sampling rate higher than 8 kHz, it is RECOMMENDED that Opus be offered before PCMA/PCMU. For Opus, all modes MUST be supported on the decoder side. The choice of encoder-side modes is left to the implementer. Clients MAY use the offer/answer mechanism to signal a preference for a particular mode or ptime.

For additional information on implementing codecs other than the mandatory-to-implement codecs listed above, refer to [I-D.ietf-rtcweb-audio-codecs-for-interop].

4. Audio Level

It is desirable to standardize the "on the wire" audio level for speech transmission to avoid users having to manually adjust the playback and to facilitate mixing in conferencing applications. It is also desirable to be consistent with ITU-T recommendations G.169 and G.115, which recommend an active audio level of -19 dBm0. However, unlike G.169 and G.115, the audio for WebRTC is not constrained to have a passband specified by G.712 and can in fact be sampled at any sampling rate from 8 kHz to 48 kHz and up. For this

reason, the level SHOULD be normalized by only considering frequencies above 300 Hz, regardless of the sampling rate used. The level SHOULD also be adapted to avoid clipping, either by lowering the gain to a level below -19 dBm0, or through the use of a compressor.

Assuming 16-bit PCM with a value of +/-32767, -19 dBm0 corresponds to a root mean square (RMS) level of 2600. Only active speech should be considered in the RMS calculation. If the client has control over the entire audio capture path, as is typically the case for a regular phone, then it is RECOMMENDED that the gain be adjusted in such a way that active speech have a level of 2600 (-19 dBm0) for an average speaker. If the client does not have control over the entire audio capture, as is typically the case for a software client, then the client SHOULD use automatic gain control (AGC) to dynamically adjust the level to 2600 (-19 dBm0) +/- 6 dB. For music or desktop sharing applications, the level SHOULD NOT be automatically adjusted and the client SHOULD allow the user to set the gain manually.

The RECOMMENDED filter for normalizing the signal energy is a second-order Butterworth filter with a 300 Hz cutoff frequency.

It is common for the audio output on some devices to be "calibrated" for playing back pre-recorded "commercial" music, which is typically around 12 dB louder than the level recommended in this section. Because of this, clients MAY increase the gain before playback.

5. Acoustic Echo Cancellation (AEC)

It is plausible that the dominant near to mid-term WebRTC usage model will be people using the interactive audio and video capabilities to communicate with each other via web browsers running on a notebook computer that has built-in microphone and speakers. The notebook-as-communication-device paradigm presents challenging echo cancellation problems, the specific remedy of which will not be mandated here. However, while no specific algorithm or standard will be required by WebRTC compatible clients, echo cancellation will improve the user experience and should be implemented by the endpoint device.

WebRTC clients SHOULD include an AEC or some other form of echo control and if that is not possible, the clients SHOULD ensure that the speaker-to-microphone gain is below unity at all frequencies to avoid instability when none of the client has echo control. For clients that do not control the audio capture and playback hardware, it is RECOMMENDED to support echo cancellation between devices running at slightly different sampling rates, such as when a webcam is used for microphone.

Clients SHOULD allow the entire AEC and/or the non-linear processing (NLP) to be turned off for applications, such as music, that do not behave well with the spectral attenuation methods typically used in NLPs. Similarly, clients SHOULD have the ability to detect the presence of a headset and disable echo cancellation.

For some applications where the remote client may not have an echo canceller, the local client MAY include a far-end echo canceller, but if that is the case, it SHOULD be disabled by default.

6. Legacy VoIP Interoperability

The codec requirements above will ensure, at a minimum, voice interoperability capabilities between WebRTC client applications and legacy phone systems that support G.711.

7. IANA Considerations

This document makes no request of IANA.

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

8. Security Considerations

Implementers should consider whether the use of VBR is appropriate for their application based on [RFC6562]. Encryption and authentication issues are beyond the scope of this document.

9. Acknowledgements

This draft incorporates ideas and text from various other drafts. In particularly we would like to acknowledge, and say thanks for, work we incorporated from Harald Alvestrand and Cullen Jennings.

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.

[RFC3551] Schulzrinne, H. and S. Casner, "RTP Profile for Audio and Video Conferences with Minimal Control", STD 65, RFC 3551, July 2003.

[RFC3389] Zopf, R., "Real-time Transport Protocol (RTP) Payload for Comfort Noise (CN)", RFC 3389, September 2002.

[RFC4733] Schulzrinne, H. and T. Taylor, "RTP Payload for DTMF Digits, Telephony Tones, and Telephony Signals", RFC 4733, December 2006.

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

[RFC6562] Perkins, C. and JM. Valin, "Guidelines for the Use of Variable Bit Rate Audio with Secure RTP", RFC 6562, March 2012.

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

10.2. Informative References

[I-D.ietf-rtcweb-audio-codecs-for-interop] Proust, S., Berger, E., Feiten, B., Bogineni, K., Lei, M., and E. Marocco, "Additional WebRTC audio codecs for interoperability with legacy networks.", draft-ietf-rtcweb-audio-codecs-for-interop-00 (work in progress), September 2014.

Authors' Addresses

Jean-Marc Valin
Mozilla
331 E. Evelyn Avenue
Mountain View, CA 94041
USA

Email: jmvalin@jmvalin.ca

Cary Bran
Plantronics
345 Encinial Street
Santa Cruz, CA 95060
USA

Phone: +1 206 661-2398
Email: cary.bran@plantronics.com

Network Working Group
Internet-Draft
Intended status: Informational
Expires: July 20, 2015

S. Proust
Orange
E. Berger
Cisco
B. Feiten
Deutsche Telekom
B. Burman
Ericsson
K. Bogineni
Verizon Wireless
M. Lei
Huawei
E. Marocco
Telecom Italia
January 16, 2015

Additional WebRTC audio codecs for interoperability.
draft-ietf-rtcweb-audio-codecs-for-interop-01

Abstract

To ensure a baseline level of interoperability between WebRTC clients, [I-D.ietf-rtcweb-audio] requires a minimum set of codecs. However, to maximize the possibility to establish the session without the need for audio transcoding, it is also recommended to include in the offer other suitable audio codecs that are available to the browser.

This document provides some guidelines on the suitable codecs to be considered for WebRTC clients to address the most relevant interoperability use cases.

Status of This Memo

This Internet-Draft will expire on July 20, 2015.

Copyright Notice

Table of Contents

1. Introduction

As indicated in [I-D.ietf-rtcweb-overview], it has been anticipated that WebRTC will not remain an isolated island and that some WebRTC endpoints will need to communicate with devices used in other existing networks with the help of a gateway. Therefore, in order to maximize the possibility to establish the session without the need for audio transcoding, it is recommended in [I-D.ietf-rtcweb-audio] to include in the offer other suitable audio codecs that are available to the browser. This document provides some guidelines on the suitable codecs to be considered for WebRTC clients to address the most relevant interoperability use cases.

The codecs considered in this document are recommended to be supported and included in the Offer only for WebRTC clients for which interoperability with other non WebRTC end points and non WebRTC based services is relevant as described in sections 5.1.2, 5.2.2 and 5.3.2. Other use cases may justify offering other additional codecs to avoid transcodings. It is the intent of this document to inventory and document any other additional interoperability use cases and codecs if needed.

2. Definitions

Legacy networks: In this draft, legacy networks encompass the conversational networks that are already deployed like the PSTN, the PLMN, the IMS, H.323 networks.

3. Rationale for additional WebRTC codecs

The mandatory implementation of OPUS [RFC6716] in WebRTC clients can guarantee the codec interoperability (without transcoding) at the state of the art voice quality (better than narrow band "PSTN" quality) between WebRTC clients. The WebRTC technology is however expected to be used to communicate with other types of clients using other technologies. It can be used for instance as an access technology to 3GPP IMS services (e.g. VoLTE, ViLTE) or to interoperate with fixed or mobile Circuit Switched or VoIP services like mobile 3GPP 3G/2G Circuit Switched voice or DECT based VoIP telephony. Consequently, a significant number of calls are likely to occur between terminals supporting WebRTC clients and other terminals like mobile handsets, fixed VoIP terminals, DECT terminals that do not support WebRTC clients nor implement OPUS. As a consequence, these calls are likely to be either of low narrow band PSTN quality using G.711 at both ends or affected by transcoding operations. The drawbacks of such transcoding operations are recalled below:

- o Degraded user experience with respect to voice quality: voice quality is significantly degraded by transcoding. For instance, the degradation is around 0.2 to 0.3 MOS for most of transcoding use cases with AMR-WB at 12.65 kbit/s and in the same range for other wideband transcoding cases. It should be stressed that if G.711 is used as a fall back codec for interoperation, wideband voice quality will be lost. Such bandwidth reduction effect down to narrow band clearly degrades the user perceived quality of service leading to shorter and less frequent calls. Such a switch to G.711 is less than desirable or acceptable choice for customers. If transcoding is performed between OPUS and any other wideband codec, wideband communication could be maintained but with degraded quality (MOS scores of transcoding between AMR-WB 12.65kbit/s and OPUS at 16 kbit/s in both directions are significantly lower than those of AMR-WB at 12.65kbit/s or OPUS at 16 kbit/s). Furthermore, in degraded conditions, the addition of defects, like audio artifacts due to packet losses, and the audio effects resulting from the cascading of different packet loss recovery algorithms may result in a quality below the acceptable limit for the customers.

- o Degraded user experience with respect to conversational interactivity: the degradation of conversational interactivity is due to the increase of end to end latency for both directions that is introduced by the transcoding operations. Transcoding requires full de-packetization for decoding of the media stream (including mechanisms of de-jitter buffering and packet loss recovery) then re-encoding, re-packetization and re-sending. The delays produced by all these operations are additive and may increase the end to end delay beyond acceptable limits like with more than 1s end to end latency.

- o Additional costs in networks: transcoding places important additional costs on network gateways mainly related to codec implementation, codecs license, deployments, testing and validation costs. It must be noted that transcoding of wideband to wideband would require more CPU and be more costly than between narrowband codecs.

4. Additional suitable codecs for WebRTC

The following codecs are considered as relevant suitable codecs with respect to the general purpose described in section 4. This list reflects the current status of WebRTC foreseen use cases. It is not limitative and opened to further inclusion of other codecs for which

relevant use cases can be identified. These additional codecs are recommended to be included in the offer in addition to OPUS and G.711 according to the foreseen interoperability cases to be addressed.

4.1. AMR-WB

4.1.1. AMR-WB General description

The Adaptive Multi-Rate WideBand (AMR-WB) is a 3GPP defined speech codec that is mandatory to implement in any 3GPP terminal that supports wideband speech communication. It is being used in circuit switched mobile telephony services and new multimedia telephony services over IP/IMS and 4G/VoLTE, specified by GSMA as voice IMS profile for VoLTE in [IR.92]. More detailed information on AMR-WB can be found in [IR.36]. [IR.36] includes references for all 3GPP AMR-WB related specifications including detailed codec description and Source code.

4.1.2. WebRTC relevant use case for AMR-WB

The market of voice personal communication is driven by mobile terminals. AMR-WB is now implemented in more than 200 devices models and 85 HD mobile networks in 60 countries with a customer base of more than 100 million. A high number of calls are consequently likely to occur between WebRTC clients and mobile 3GPP terminals. The use of AMR-WB by WebRTC clients would consequently allow transcoding free interoperation with all mobile 3GPP wideband terminal. Besides, WebRTC clients running on mobile terminals (smartphones) may reuse the AMR-WB codec already implemented on these devices.

4.1.3. Guidelines for AMR-WB usage and implementation with WebRTC

Guidelines for implementing and using AMR-WB and ensuring interoperability with 3GPP mobile services can be found in [TS26.114]. In order to ensure interoperability with 4G/VoLTE as specified by GSMA, the more specific IMS profile for voice derived from [TS26.114] should be considered in [IR.92]. In order to maximize the possibility of successful call establishment for WebRTC client offering AMR-WB it is important that the WebRTC client:

- Offer AMR in addition to AMR-WB with AMR-WB, being a wideband codec, listed first as preferred payload type with respect to other narrow band codecs (AMR, G.711...)and with Bandwidth Efficient payload format preferred.

- Be capable of operating AMR-WB with any subset of the nine codec modes and source controlled rate operation. Offer at least one

AMR-WB configuration with parameter settings as defined in Table 6.1 of [TS 26.114]. In order to maximize the interoperability and quality this offer does not restrict the codec modes offered. Restrictions in the use of codec modes may be included in the answer.

4.2. AMR

4.2.1. AMR General description

Adaptive Multi-Rate (AMR) is a 3GPP defined speech codec that is mandatory to implement in any 3GPP terminal that supports voice communication, i.e. several hundred millions of terminals. This include both mobile phone calls using GSM and 3G cellular systems as well as multimedia telephony services over IP/IMS and 4G/VoLTE, such as GSMA voice IMS profile for VoLTE in [IR.92]. In addition to impacts listed above, support of AMR can avoid degrading the high efficiency over mobile radio access.

4.2.2. WebRTC relevant use case for AMR

A user of a WebRTC endpoint on a device integrating an AMR module wants to communicate with another user that can only be reached on a mobile device that only supports AMR. Although more and more terminal devices are now "HD voice" and support AMR-WB; there is still a high number of legacy terminals supporting only AMR (terminals with no wideband / HD Voice capabilities) are still used. The use of AMR by WebRTC client would consequently allow transcoding free interoperation with all mobile 3GPP terminals. Besides, WebRTC client running on mobile terminals (smartphones) may reuse the AMR codec already implemented on these devices.

4.2.3. Guidelines for AMR usage and implementation with WebRTC

Guidelines for implementing and using AMR with purpose to ensure interoperability with 3GPP mobile services can be found in [TS26.114]. In order to ensure interoperability with 4G/VoLTE as specified by GSMA, the more specific IMS profile for voice derived from [TS26.114] should be considered in [IR.92]. In order to maximize the possibility of successful call establishment for WebRTC client offering AMR, it is important that the WebRTC client:

- Be capable of operating AMR with any subset of the eight codec modes and source controlled rate operation.

- Offer at least one configuration with parameter settings as defined in Table 6.1 and Table 6.2 of [TS26.114]. In order to maximize the interoperability and quality this offer shall not

restrict AMR codec modes offered. Restrictions in the use of codec modes may be included in the answer.

4.3. G.722

4.3.1. G.722 General description

G.722 is an ITU-T defined wideband speech codec. [G.722] was approved by ITU-T in 1988. It is a royalty free codec that is common in a wide range of terminals and end-points supporting wideband speech and requiring low complexity. The complexity of G.722 is estimated to 10 MIPS [EN300175-8] which is 2.5 to 3 times lower than AMR-WB. Especially, G.722 has been chosen by ETSI DECT as the mandatory wideband codec for New Generation DECT with purpose to greatly increase the voice quality by extending the bandwidth from narrow band to wideband. G.722 is the wideband codec required for CAT-iq DECT certified terminal and the V2.0 of CAT-iq specifications have been approved by GSMA as minimum requirements for HD voice logo usage on "fixed" devices; i.e., broadband connections using the G.722 codec.

4.3.2. WebRTC relevant use case for G.722

G.722 is the wideband codec required for DECT CAT-iq terminals. The market for DECT cordeless phones including DECT chipset is more than 150 Millions per year and CAT-IQ is a registered trade make in 47 countries worldwide. G.722 has also been specified by ETSI in [TS181005] as mandatory wideband codec for IMS multimedia telephony communication service and supplementary services using fixed broadband access. The support of G.722 would consequently allow transcoding free IP interoperation between WebRTC client and fixed VoIP terminals including DECT / CAT-IQ terminals supporting G.722. Besides, WebRTC client running on fixed terminals implementing G.722 may reuse the G.722 codec already implemented on these devices.

4.3.3. Guidelines for G.722 usage and implementation

Guidelines for implementing and using G.722 with purpose to ensure interoperability with Multimedia Telephony services overs IMS can be found in section 7 of [TS26.114]. Additional information of G.722 implementation in DECT can be found in [EN300175-8] and full codec description and C source code in [G.722].

4.4. Other codecs

Other interoperability use cases may justify the use of other codecs. Some further update of this Draft may provide under this section

additional use case description and codec implementation guidelines for these codecs.

5. Security Considerations

6. IANA Considerations

None.

7. Acknowledgements

Thanks to Milan Patel for his review.

8. References

8.1. Normative references

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

8.2. Informative references

[EN300175-8]
ETSI, "ETSI EN 300 175-8, v2.5.1: "Digital Enhanced Cordless Telecommunications (DECT); Common Interface (CI); Part 8: Speech and audio coding and transmission".", 2009.

[G.722] ITU, "Recommendation ITU-T G.722 (2012): "7 kHz audio-coding within 64 kbit/s".", 2012.

[I-D.ietf-rtcweb-audio]
Valin, J. and C. Bran, "WebRTC Audio Codec and Processing Requirements", draft-ietf-rtcweb-audio-07 (work in progress), October 2014.

[I-D.ietf-rtcweb-overview]
Alvestrand, H., "Overview: Real Time Protocols for Browser-based Applications", draft-ietf-rtcweb-overview-13 (work in progress), November 2014.

[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-15 (work in progress), December 2014.

[IR.36] GSMA, "Adaptive Multirate Wide Band", 2013.

[IR.92] GSMA, "IMS Profile for Voice and SMS", 2013.

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

[TS181005] ETSI, "Telecommunications and Internet converged Services and Protocols for Advanced Networking (TISPAN); Service and Capability Requirements V3.3.1 (2009-12)", 2009.

[TS26.114] 3GPP, "IP Multimedia Subsystem (IMS); Multimedia telephony; Media handling and interaction", 2011.

Authors' Addresses

Stephane Proust
Orange
2, avenue Pierre Marzin
Lannion 22307
France

Email: stephane.proust@orange.com

Espen Berger
Cisco

Email: espeberg@cisco.com

Bernhard Feiten
Deutsche Telekom

Email: Bernhard.Feiten@telekom.de

Bo Burman
Ericsson

Email: bo.burman@ericsson.com

Kalyani Bogineni
Verizon Wireless

Email: Kalyani.Bogineni@VerizonWireless.com

Miao Lei
Huawei

Email: lei.miao@huawei.com

Enrico Marocco
Telecom Italia

Email: enrico.marocco@telecomitalia.it

Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: July 8, 2015

R. Jesup
Mozilla
S. Loreto
Ericsson
M. Tuexen
Muenster Univ. of Appl. Sciences

January 4, 2015

WebRTC Data Channels
draft-ietf-rtcweb-data-channel-13.txt

Abstract

The WebRTC framework specifies 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 WebRTC framework. It provides an architectural overview of how the Stream Control Transmission Protocol (SCTP) is used in the WebRTC context as a generic transport service allowing WEB-browsers to exchange generic data from peer to peer.

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

In the WebRTC framework, communication between the parties consists of media (for example audio and video) and non-media data. Media is sent using SRTP, and is not specified further here. Non-media data is handled by using SCTP [RFC4960] encapsulated in DTLS. DTLS 1.0 is defined in [RFC4347] and the present latest version, DTLS 1.2, is defined in [RFC6347].

```
+----------+
|   SCTP   |
+----------+
|   DTLS   |
+----------+
| ICE/UDP  |
+----------+
```

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 SRTP media transports, and all of them can eventually share a single UDP port number.

SCTP as specified in [RFC4960] with the partial reliability extension defined in [RFC3758] and the additional policies defined in [I-D.ietf-tsvwg-sctp-prpolicies] provides multiple streams natively with reliable, and the relevant 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. Using [I-D.ietf-tsvwg-sctp-ndata] allows to interleave large messages to avoid the monopolization and adds the support of prioritizing of 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 discusses SCTP over DTLS over UDP; Section 6 provides the specification of how SCTP should be used by the WebRTC 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 defines use cases specific to data channels. Please note that this section is informational only.

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 SRTP media channels, or all SRTP 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 SRTP 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 configuration of 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. Please note that this section is informational only.

Req. 1: Multiple simultaneous data channels must be supported. Note that there may be 0 or more SRTP media streams in parallel with the data channels in the same PeerConnection, and the number and state (active/inactive) of these SRTP media streams may change at any time.

Req. 2: Both reliable and unreliable data channels must be supported.

Req. 3: Data channels of a PeerConnection must be congestion controlled; either individually, as a class, or in conjunction with the SRTP media streams of the PeerConnection, to ensure that data channels don’t cause congestion problems for these SRTP media streams, and that the WebRTC PeerConnection does not cause excessive problems when run in parallel with TCP connections.

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 SRTP media streams. 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 WebRTC context are:

- o Usage of a TCP-friendly congestion control.
- o The congestion control is modifiable for integration with the SRTP 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 messages by providing fragmentation and reassembly.
- o Support of PMTU-discovery.
- o Support of reliable or partially reliable message transport.

The WebRTC Data Channel mechanism does not support SCTP multihoming. The SCTP layer will simply act as if it were running on a single-homed host, since that is the abstraction that the DTLS 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 middlebox traversal in IPv4 and IPv6 based networks. SCTP as specified in [RFC4960] MUST be used in combination with the extension defined in [RFC3758] and provides the following 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 Payload Protocol Identifier (PPID) that is passed to SCTP by its upper layer on the sending side and provided to its upper layer on the receiving side. The PPID can be used to multiplex/demultiplex multiple upper layers over a single SCTP association. In the WebRTC context, the PPID is used to distinguish between UTF-8 encoded user data, binary encoded userdata and the Data Channel Establishment Protocol defined in

[I-D.ietf-rtcweb-data-protocol]. Please note that the PPID is not accessible via the Javascript API.

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.

```
                              +------+------+------+
                              | DCEP | UTF-8|Binary|
                              |      | data | data |
                              +------+------+------+
                              |        SCTP        |
                 +-----------------------------------+
                 | STUN | SRTP |        DTLS        |
                 +-----------------------------------+
                 |                ICE                |
                 +-----------------------------------+
                 | UDP1 | UDP2 | UDP3 | ...         |
                 +-----------------------------------+
```

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 SRTP media channels of the PeerConnection.
- o provides privacy for the SCTP control information.

Considering the protocol stack of Figure 2 the usage of DTLS 1.0 over UDP is specified in [RFC4347] and the usage of DTLS 1.2 over UDP in specified in [RFC6347], while the usage of SCTP on top of DTLS is specified in [I-D.ietf-tsvwg-sctp-dtls-encaps]. Please note that the demultiplexing STUN vs. SRTP vs. DTLS is done as described in Section 5.1.2 of [RFC5764] and SCTP is the only payload of DTLS.

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

The ICE/UDP layer can handle IP address changes during a session without needing interaction with the DTLS and SCTP layers. However, SCTP SHOULD be notified when an address changes has happened. In this case SCTP SHOULD retest the Path MTU and reset the congestion

state to the initial state. In case of a window based congestion control like the one specified in [RFC4960], this means setting the congestion window and slow start threshold to its initial values.

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 SHOULD NOT exceed 1200 bytes for IPv4 and 1280 for IPv6.

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 the 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.

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 than the standard one might improve the impact on the parallel SRTP media streams.

SCTP uses the same port number concept as TCP and UDP do. Therefore an SCTP association uses two port numbers, one at each SCTP end-point.

6. The Usage of SCTP for Data Channels

6.1. SCTP Protocol Considerations

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

This SCTP stack and its upper layer MUST support the usage of multiple SCTP streams. A user message can be sent ordered or unordered and with partial or full reliability.

The following SCTP protocol extensions are required:

- o The stream reconfiguration 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] are OPTIONAL.

- 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.ietf-tsvwg-sctp-prpolicies] MUST be supported. 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.

The support for message interleaving as defined in [I-D.ietf-tsvwg-sctp-ndata] SHOULD be used.

6.2. SCTP Association Management

In the WebRTC context, 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]. It will use the DTLS connection selected via ICE; typically this will be shared via BUNDLE or equivalent with DTLS connections used to key the SRTP media streams.

The number of streams negotiated during SCTP association setup SHOULD be 65535, which is the maximum number of streams that can be negotiated during the association setup.

SCTP supports two ways of terminating an SCTP association. A graceful one, using a procedure which ensures that no messages are lost during the shutdown of the association. The second method is a non-graceful one, where one side can just abort the association.

Each SCTP end-point supervises continuously the reachability of its peer by monitoring the number of retransmissions of user messages and test messages. In case of excessive retransmissions, the association is terminated in a non-graceful way.

If an SCTP association is closed in a graceful way, all of its data channels are closed. In case of a non-graceful teardown, all data channels are also closed, but an error indication SHOULD be provided if possible.

6.3. SCTP Streams

SCTP defines a stream as a unidirectional logical channel existing within an SCTP association 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 ordered or unordered. Ordering is preserved only for ordered messages sent on the same stream.

6.4. Data Channel Definition

Data channels are defined such that their accompanying application-level API can closely mirror the API for WebSockets, which implies bidirectional streams of data and a textual field called ’label’ used to identify the meaning of the data channel.

The realization of a data channel is a pair of one incoming stream and one outgoing SCTP stream having the same SCTP stream identifier. How these SCTP stream identifiers are selected is protocol and implementation dependent. This allows a bidirectional communication.

Additionally, each data channel has the following properties in each direction:

- o reliable or unreliable message transmission. In case of unreliable transmissions, the same level of unreliability is used. Please note that in SCTP this is a property of an SCTP user message and not of an SCTP stream.

- o in-order or out-of-order message delivery for message sent. Please note that in SCTP this is a property of an SCTP user message and not of an SCTP stream.

- o A priority, which is a 2 byte unsigned integer. These priorities MUST be interpreted as weighted-fair-queuing scheduling priorities per the definition of the corresponding stream scheduler supporting interleaving in [I-D.ietf-tsvwg-sctp-ndata]. For use in WebRTC, the values used SHOULD be one of 128 ("below normal"), 256 ("normal"), 512 ("high") or 1024 ("extra high").

- o an optional label.

- o an optional protocol.

Please note that for a data channel being negotiated with the protocol specified in [I-D.ietf-rtcweb-data-protocol] all of the above properties are the same in both directions.

6.5. Opening a Data Channel

Data channels can be opened by using negotiation within the SCTP association, called in-band negotiation, or out-of-band negotiation. Out-of-band negotiation is defined as any method which results in an

agreement as to the parameters of a channel and the creation thereof. The details are out of scope of this document. Applications using data channels need to use the negotiation methods consistently on both end-points.

A simple protocol for in-band negotiation is specified in [I-D.ietf-rtcweb-data-protocol].

When one side wants to open a channel using out-of-band negotiation, it picks a stream. Unless otherwise defined or negotiated, the streams are picked based on the DTLS role (the client picks even stream identifiers, the server odd stream identifiers). 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 data channel, the addition MUST fail. In addition to choosing a stream, the application SHOULD also determine the options to use for sending messages. The application MUST ensure in an application-specific manner that the application at the peer will also know the selected stream to be used, and the options for sending data from that side.

6.6. Transferring User Data on a Data Channel

All data sent on a data channel in both directions MUST be sent over the underlying stream using the reliability defined when the data channel was opened unless the options are changed, or per-message options are specified by a higher level.

The message-orientation of SCTP is used to preserve the message boundaries of user messages. Therefore, senders MUST NOT put more than one application message into an SCTP user message. Unless the deprecated PPID-based fragmentation and reassembly is used, the sender MUST include exactly one application message in each SCTP user message.

The SCTP Payload Protocol Identifiers (PPIDs) are used to signal the interpretation of the "Payload data". The following PPIDs MUST be used (see Section 8):

WebRTC String: to identify a non-empty JavaScript string encoded in UTF-8.

WebRTC String Empty: to identify an empty JavaScript string encoded in UTF-8.

WebRTC Binary: to identify a non-empty JavaScript binary data (ArrayBuffer, ArrayBufferView or Blob).

WebRTC Binary Empty: to identify an empty JavaScript binary data (ArrayBuffer, ArrayBufferView or Blob).

SCTP does not support the sending of empty user messages. Therefore, if an empty message has to be sent, the appropriate PPID (WebRTC String Empty or WebRTC Binary Empty) is used and the SCTP user message of one zero byte is sent. When receiving an SCTP user message with one of these PPIDs, the receiver MUST ignore the SCTP user message and process it as an empty message.

The usage of the PPIDs "WebRTC String Partial" and "WebRTC Binary Partial" is deprecated. They were used for a PPID-based fragmentation and reassembly of user messages belonging to reliable and ordered data channels.

If a message with an unsupported PPID is received or some error condition related to the received message is detected by the receiver (for example, illegal ordering), the receiver SHOULD close the corresponding data channel. This implies in particular that extensions using additional PPIDs can’t be used without prior negotiation.

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.ietf-tsvwg-sctp-ndata] defines an extension to support message interleaving, which SHOULD be used. As long as message interleaving is not supported, the sender SHOULD limit the maximum message size to 16 KB to avoid monopolization.

It is recommended that the message size be kept within certain size bounds as applications will not be able to support arbitrarily-large single messages. This limit has to be negotiated, for example by using [I-D.ietf-mmusic-sctp-sdp].

The sender SHOULD disable the Nagle algorithm (see [RFC1122]) to minimize the latency.

6.7. Closing a Data Channel

Closing of a data channel MUST be signaled by resetting the corresponding outgoing streams [RFC6525]. This means that if one side decides to close the data channel, it resets the corresponding outgoing stream. When the peer sees that an incoming stream was reset, it also resets its corresponding outgoing stream. Once this is completed, the data channel is closed. Resetting a stream sets 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 for reuse after a reset has been performed.

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

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].

It should be noted that a receiver must be prepared that the sender tries to send arbitrary large messages.

8. IANA Considerations

[NOTE to RFC-Editor:

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

]

This document uses six already registered SCTP Payload Protocol Identifiers (PPIDs): "DOMString Last", "Binary Data Partial", "Binary Data Last", "DOMString Partial", "WebRTC String Empty", and "WebRTC Binary Empty". [RFC4960] creates the registry "SCTP Payload Protocol Identifiers" from which these identifiers were assigned. IANA is requested to update the reference of these six assignments to point to this document and change the names of the first four PPIDs. The corresponding dates should be kept.

Therefore these six assignments should be updated to read:

Value	SCTP PPID	Reference	Date
WebRTC String	51	[RFCXXXX]	2013-09-20
WebRTC Binary Partial (Deprecated)	52	[RFCXXXX]	2013-09-20
WebRTC Binary	53	[RFCXXXX]	2013-09-20
WebRTC String Partial (Deprecated)	54	[RFCXXXX]	2013-09-20
WebRTC String Empty	56	[RFCXXXX]	2014-08-22
WebRTC Binary Empty	57	[RFCXXXX]	2014-08-22

9. Acknowledgments

Many thanks for comments, ideas, and text from Harald Alvestrand, Richard Barnes, Adam Bergkvist, Alissa Cooper, Benoit Claise, Spencer Dawkins, Gunnar Hellstrom, Christer Holmberg, Cullen Jennings, Paul Kyzivat, Eric Rescorla, Adam Roach, Irene Ruengeler, Randall Stewart, Martin Stiemerling, 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.

[RFC4347] Rescorla, E. and N. Modadugu, "Datagram Transport Layer Security", RFC 4347, April 2006.

[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.ietf-tsvwg-sctp-ndata]
Stewart, R., Tuexen, M., Loreto, S., and R. Seggelmann, "Stream Schedulers and a New Data Chunk for the Stream Control Transmission Protocol", draft-ietf-tsvwg-sctp-ndata-01 (work in progress), July 2014.

[I-D.ietf-rtcweb-data-protocol]
Jesup, R., Loreto, S., and M. Tuexen, "WebRTC Data Channel Establishment Protocol", draft-ietf-rtcweb-data-protocol-08 (work in progress), September 2014.

[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-07 (work in progress), December 2014.

[I-D.ietf-rtcweb-security]
Rescorla, E., "Security Considerations for WebRTC", draft-ietf-rtcweb-security-07 (work in progress), July 2014.

[I-D.ietf-rtcweb-security-arch]
Rescorla, E., "WebRTC Security Architecture", draft-ietf-rtcweb-security-arch-10 (work in progress), July 2014.

[I-D.ietf-rtcweb-jsep]
Uberti, J., Jennings, C., and E. Rescorla, "Javascript Session Establishment Protocol", draft-ietf-rtcweb-jsep-08 (work in progress), October 2014.

[I-D.ietf-tsvwg-sctp-prpolicies]
Tuexen, M., Seggelmann, R., Stewart, R., and S. Loreto, "Additional Policies for the Partial Reliability Extension of the Stream Control Transmission Protocol", draft-ietf-tsvwg-sctp-prpolicies-06 (work in progress), December 2014.

[I-D.ietf-mmusic-sctp-sdp]
Holmberg, C., Loreto, S., and G. Camarillo, "Stream Control Transmission Protocol (SCTP)-Based Media Transport in the Session Description Protocol (SDP)", draft-ietf-mmusic-sctp-sdp-11 (work in progress), December 2014.

10.2. Informative References

[RFC1122] Braden, R., "Requirements for Internet Hosts - Communication Layers", STD 3, RFC 1122, October 1989.

[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.

[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.

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: July 8, 2015

R. Jesup
Mozilla
S. Loreto
Ericsson
M. Tuexen
Muenster Univ. of Appl. Sciences

January 4, 2015

WebRTC Data Channel Establishment Protocol
draft-ietf-rtcweb-data-protocol-09.txt

Abstract

The WebRTC framework specifies protocol 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

Copyright Notice

Table of Contents

1. Introduction

The Data Channel Establishment Protocol (DCEP) 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) as described in [I-D.ietf-tsvwg-sctp-dtls-encaps] to benefit from their already standardized transport and security features. DTLS 1.0 is defined in [RFC4347] and the present latest version, DTLS 1.2, is defined in [RFC6347].

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 an SCTP stream identifier (0-65534). Note: the SCTP stream identifier 65535 is reserved due to SCTP INIT and INIT-ACK chunks only allowing a maximum of 65535 Streams to be negotiated (0-65534).

Stream Identifier: The SCTP stream identifier uniquely identifying a Stream.

Data Channel: Two Streams with the same Stream Identifier, one in each direction, which are managed together.

4. Protocol Overview

The Data Channel Establishment Protocol is a simple, low-overhead way to establish bidirectional Data Channels over an SCTP association with a consistent set of properties.

The set of consistent properties includes:

- reliable or unreliable message transmission. In case of unreliable transmissions, the same level of unreliability is used.
- in-order or out-of-order message delivery.
- the priority of the Data Channel.
- an optional label for the Data Channel.
- an optional protocol for the Data Channel.
- the Streams.

This protocol uses a two way handshake to open a Data Channel. The handshake pairs one incoming and one outgoing Stream, both having the same Stream Identifier, into a single bidirectional Data Channel. The peer that initiates opening a Data Channel selects a Stream Identifier for which the corresponding incoming and outgoing Streams are unused and sends a DATA_CHANNEL_OPEN message on the outgoing Stream. The peer responds with a DATA_CHANNEL_ACK message on its corresponding outgoing Stream. Then the Data Channel is open. Data Channel Establishment Protocol 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 (PPID), since the Data Channel Establishment Protocol uses a specific PPID.

Note: The opening side MAY send user messages before the DATA_CHANNEL_ACK is received.

To avoid collisions where both sides try to open a Data Channel with the same Stream Identifiers, each side MUST use Streams with either even or odd Stream Identifiers when sending a DATA_CHANNEL_OPEN message. When using SCTP over DTLS [I-D.ietf-tsvwg-sctp-dtls-encaps], the method used to determine which side uses odd or even is based on the underlying DTLS connection role: the side acting as the DTLS client MUST use Streams with even Stream Identifiers, the side acting as the DTLS server MUST use Streams with odd Stream Identifiers.

Note: There is no attempt to ensure uniqueness for the label; if both sides open a Data Channel labeled "x" at the same time, there will be two Data 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 user data being passed with an IANA-registered string (’WebSocket Subprotocol Name Registry’ defined in [RFC6455]), and may be useful for homogeneous applications which may create more than one type of Data Channel. Please note that there is also no attempt to ensure uniqueness for the protocol field.

5. Message Formats

Every Data Channel Establishment 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 8.2.1).

5.1. DATA_CHANNEL_OPEN Message

This message is sent initially on the Stream used for user messages using the Data Channel.

```
 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 |  Channel Type |            Priority           |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                     Reliability Parameter                     |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|         Label Length          |       Protocol Length         |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
\                                                               /
|                             Label                             |
/                                                               \
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
\                                                               /
|                           Protocol                            |
/                                                               \
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Message Type: 1 byte (unsigned integer)
This field holds the IANA defined message type for the DATA_CHANNEL_OPEN message. The value of this field is 0x03 as specified in Section 8.2.1.

Channel Type: 1 byte (unsigned integer)
This field specifies the type of the Data Channel to be opened and the values are managed by IANA (see Section 8.2.2):

DATA_CHANNEL_RELIABLE (0x00): The Data Channel provides a reliable in-order bi-directional communication.

DATA_CHANNEL_RELIABLE_UNORDERED (0x80): The Data Channel provides a reliable unordered bi-directional communication.

DATA_CHANNEL_PARTIAL_RELIABLE_REXMIT (0x01): The Data Channel provides a partially-reliable in-order bi-directional communication. User messages will not be retransmitted more times than specified in the Reliability Parameter.

DATA_CHANNEL_PARTIAL_RELIABLE_REXMIT_UNORDERED (0x81): The Data Channel provides a partial reliable unordered bi-directional communication. User messages will not be retransmitted more times than specified in the Reliability Parameter.

DATA_CHANNEL_PARTIAL_RELIABLE_TIMED (0x02): The Data Channel provides a partial reliable in-order bi-directional communication. 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 protocol stack.

DATA_CHANNEL_PARTIAL_RELIABLE_TIMED_UNORDERED (0x82): The Data Channel provides a partial reliable unordered bi-directional communication. 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 protocol stack.

Priority: 2 bytes (unsigned integer)
The priority of the Data Channel as described in [I-D.ietf-rtcweb-data-channel].

Reliability Parameter: 4 bytes (unsigned integer)
For reliable Data Channels this field MUST be set to 0 on the sending side and MUST be ignored on the receiving side. If a partial reliable Data Channel with limited number of retransmissions is used, this field specifies the number of retransmissions. If a partial reliable Data Channel with limited lifetime is used, this field specifies the maximum lifetime in milliseconds. The following table summarizes this:

Channel Type	Reliability Parameter
DATA_CHANNEL_RELIABLE	Ignored
DATA_CHANNEL_RELIABLE_UNORDERED	Ignored
DATA_CHANNEL_PARTIAL_RELIABLE_REXMIT	Number of RTX
DATA_CHANNEL_PARTIAL_RELIABLE_REXMIT_UNORDERED	Number of RTX
DATA_CHANNEL_PARTIAL_RELIABLE_TIMED	Lifetime in ms
DATA_CHANNEL_PARTIAL_RELIABLE_TIMED_UNORDERED	Lifetime in ms

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 Data Channel as a UTF-8 encoded string as specified in [RFC3629]. This may be an empty string.

Protocol: Variable Length (sequence of characters)
If this is an empty string the protocol is unspecified. If it is a non-empty string, it specifies a protocol registered in the

'WebSocket Subprotocol Name Registry' created in [RFC6455]. This string is UTF-8 encoded as specified in [RFC3629].

5.2. DATA_CHANNEL_ACK Message

This message is sent in response to a DATA_CHANNEL_OPEN_RESPONSE message on the stream used for user messages using the Data Channel. Reception of this message tells the opener that the Data 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 DATA_CHANNEL_ACK message. The value of this field is 0x02 as specified in Section 8.2.1.

6. Procedures

All Data Channel Establishment Protocol messages MUST be sent using ordered delivery and reliable transmission. They MUST be sent on the same outgoing 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 Establishment Protocol message MUST be sent with the assigned PPID for the Data Channel Establishment Protocol (see Section 8.1). Other messages MUST NOT be sent using this PPID.

The peer that initiates opening a Data Channel selects a Stream Identifier for which the corresponding incoming and outgoing Streams are unused. 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 Stream.

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 the DATA_CHANNEL_OPEN message doesn't satisfy the conditions above, for instance if a DATA_CHANNEL_OPEN message is received on an

already used Stream or there are any problems with parameters within the DATA_CHANNEL_OPEN message, the odd/even rule is violated or the DATA_CHANNEL_OPEN message itself is not well-formed, the receiver MUST close the corresponding Data Channel using the procedure described in [I-D.ietf-rtcweb-data-channel] 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. After also successfully resetting the corresponding outgoing Stream, which concludes the Data Channel closing initiated by the peer, a new DATA_CHANNEL_OPEN message can be sent on the Stream.

After the DATA_CHANNEL_OPEN message has been sent, the sender of the DATA_CHANNEL_OPEN MAY 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 a Data Channel, all other messages containing user data and belonging to this Data Channel MUST be sent ordered, no 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 sent ordered on ordered Data Channels and MUST be sent unordered on unordered Data Channels. Therefore receiving a message containing user data on an unused Stream indicates an error. The corresponding Data Channel MUST be closed as described in [I-D.ietf-rtcweb-data-channel].

7. Security Considerations

The DATA_CHANNEL_OPEN messages contains two variable length fields: the protocol and the label. A receiver must be prepared to receive DATA_CHANNEL_OPEN messages where these field have the maximum length of 65535 bytes. Error cases like the use of inconsistent lengths fields, unknown parameter values or violation the odd/even rule must also be handled by closing the corresponding Data Channel. An end-point must also be prepared that the peer open the maximum number of Data Channels.

This protocol does not provide privacy, integrity or authentication. It needs to be used as part of a protocol suite that contains all these things. Such a protocol suite is specified in [I-D.ietf-tsvwg-sctp-dtls-encaps].

For general considerations see [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 (Section 8.1) and to create a new standalone registry with its own URL for the DCEP (Section 8.2) containing two new registration tables (Section 8.2.1 and Section 8.2.2).

8.1. SCTP Payload Protocol Identifier

This document uses one already registered SCTP Payload Protocol Identifier (PPID) named "WebRTC Control". [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 and to update the name. The corresponding date should be kept.

Therefore this assignment should be updated to read:

Value	SCTP PPID	Reference	Date
WebRTC DCEP	50	[RFCXXXX]	2013-09-20

8.2. New Standalone Registry for the DCEP

IANA is requested to create a new standalone registry (aka a webpage) with its own URL for the Data Channel Establishment Protocol (DCEP). The title should be "Data Channel Establishment Protocol (DCEP) Parameters". It will contain the two tables as described in Section 8.2.1 and Section 8.2.2.

8.2.1. New Message Type Registry

IANA is requested to create a new registration table "Message Type Registry" for the Data Channel Establishment Protocol (DCEP) to manage the one byte "Message Type" field in DCEP messages (see Section 5). This registration table should be part of the registry described in Section 8.2.

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 Establishment 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]

Please note that the values 0x00 and 0x01 are reserved to avoid interoperability problems, since they have been used in earlier versions of the document. The value 0xff has been reserved for future extensibility. The range of possible values is from 0x00 to 0xff.

8.2.2. New Channel Type Registry

IANA is requested to create a new registration table "Channel Type Registry" for the Data Channel Establishment Protocol to manage the one byte "Channel Type" field in DATA_CHANNEL_OPEN messages (see Section 5.1). This registration table should be part of the registry described in Section 8.2.

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 MUST 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	

Please note that the values 0x7f and 0xff have been reserved for future extensibility. The range of possible values is from 0x00 to 0xff.

9. Acknowledgments

The authors wish to thank Harald Alvestrand, Richard Barnes, Adam Bergkvist, Spencer Dawkins, Barry Dingle, Stefan Haekansson, Cullen Jennings, Paul Kyzivat, Doug Leonard, Alexey Melnikov, Pete Resnick, Irene Ruengeler, Randall Stewart, Peter Thatcher, Martin Thompson, Justin Uberti, 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.

[RFC3629] Yergeau, F., "UTF-8, a transformation format of ISO 10646", STD 63, RFC 3629, November 2003.

[RFC4347] Rescorla, E. and N. Modadugu, "Datagram Transport Layer Security", RFC 4347, April 2006.

[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.

[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-07 (work in progress), December 2014.

[I-D.ietf-rtcweb-data-channel] Jesup, R., Loreto, S., and M. Tuexen, "WebRTC Data Channels", draft-ietf-rtcweb-data-channel-12 (work in progress), September 2014.

10.2. Informational References

[RFC6455] Fette, I. and A. Melnikov, "The WebSocket Protocol", RFC 6455, December 2011.

[I-D.ietf-rtcweb-security] Rescorla, E., "Security Considerations for WebRTC", draft-ietf-rtcweb-security-07 (work in progress), July 2014.

[I-D.ietf-rtcweb-security-arch] Rescorla, E., "WebRTC Security Architecture", draft-ietf-rtcweb-security-arch-10 (work in progress), July 2014.

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: August 9, 2015

J. Uberti
Google

February 5, 2015

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

Abstract

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

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

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

2. Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119].

3. Types of FEC

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

3.1. Separate FEC Stream

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

3.2. Redundant Encoding

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

3.3. Codec-Specific In-band FEC

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

4. FEC for Audio Content

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

4.1. Recommended Mechanism

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

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

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

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

4.2. Negotiating Support

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

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

5. FEC for Video Content

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

5.1. Recommended Mechanism

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

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

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

5.2. Negotiating Support

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

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

6. Implementation Requirements

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

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

7. Adaptive Use of FEC

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

8. Security Considerations

TODO

9. IANA Considerations

This document requires no actions from IANA.

10. Acknowledgements

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

11. References

11.1. Normative References

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

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

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

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

11.2. Informative References

[I-D.ietf-payload-rtp-opus]
Spittka, J., Vos, K., and J. Valin, "RTP Payload Format for Opus Speech and Audio Codec", draft-ietf-payload-rtp-opus-07 (work in progress), January 2015.

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

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

Appendix A. Change log

Changes in draft -00:

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

Author’s Address

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

Email: justin@uberti.name

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

J. Uberti
Google
C. Jennings
Cisco
E. Rescorla, Ed.
Mozilla

March 9, 2015

Javascript Session Establishment Protocol
draft-ietf-rtcweb-jsep-09

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

Copyright Notice

Table of Contents

1. Introduction

This document describes how the W3C WEBRTC RTCPeerConnection interface[W3C.WD-webrtc-20140617] 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.

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. JSEP removes the browser almost entirely from the core signaling flow, which is instead handled by the Javascript making use of two interfaces: (1) passing in local and remote session descriptions and (2) interacting 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.

To complete the offer/answer exchange, the remote party uses the createAnswer() API to generate an appropriate answer, applies it using the setLocalDescription() API, and sends the answer back to the initiator over the signaling channel. When the initiator gets that answer, it installs it using the setRemoteDescription() API, 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 session description can be sent immediately and the transport information can be sent when available. 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 ICE checks can start as soon as any transport information is available rather than waiting for all of it.

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, libraries exist to adapt the JSEP API into an API suitable for a SIP or XMPP. Thus, JSEP provides 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.

```
+-----------+                                +-----------+
|  Web App  |<--- App-Specific Signaling -->|  Web App  |
+-----------+                                +-----------+
      ^                                             ^
      |  SDP                                        |  SDP
      V                                             V
+-----------+                                 +-----------+
|  Browser  |<----------- Media ------------->|  Browser  |
+-----------+                                 +-----------+
```

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 DTLS-SRTP parameters [RFC5763] sent to a remote party indicate what certificate the local side will use in DTLS setup, and thereby what the remote party should expect to receive; the remote party will have to accept these parameters, with no option to choose different values.

In addition, various RFCs put different conditions on the format of offers versus answers. For example, an offer may propose an arbitrary number of media streams (i.e. m= sections), but an answer must contain the exact same number as the offer.

Lastly, while the exact media parameters are only known only after an 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 setLocalDescription and setRemoteDescription methods 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 at 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 via the createOffer() method 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:

```
                       setRemote(OFFER)               setLocal(PRANSWER)
                           /-----\                           /-----\
                           |     |                           |     |
                           v     |                           v     |
            +---------------+    |               +---------------+    |
            |               |----/               |               |----/
            |               | setLocal(PRANSWER)  |               |
            |  Remote-Offer |------------------- >| Local-Pranswer|
            |               |                     |               |
            |               |                     |               |
            +---------------+                     +---------------+
                 ^   |                                 |
                 |   | setLocal(ANSWER)                |
   setRemote(OFFER)  |                                 |
                 |   V                  setLocal(ANSWER) |
            +---------------+                         |
            |               |                         |
            |               |<--------------------------+
            |    Stable     |
            |               |<--------------------------+
            |               |                         |
            +---------------+          setRemote(ANSWER) |
                 ^   |                                 |
                 |   | setLocal(OFFER)                 |
   setRemote(ANSWER) |                                 |
                 |   V                                 |
            +---------------+                     +---------------+
            |               |                     |               |
            |               | setRemote(PRANSWER) |               |
            |  Local-Offer  |------------------- >|Remote-Pranswer|
            |               |                     |               |
            |               |----\                |               |----\
            +---------------+    |                +---------------+    |
                           ^     |                             ^     |
                           |     |                             |     |
                           \-----/                             \-----/
                       setLocal(OFFER)               setRemote(PRANSWER)
```

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 WebRTC API specification 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

3.4.1. ICE Gathering Overview

JSEP gathers ICE candidates as needed by the application. Collection of ICE candidates is referred to as a gathering phase, and this is triggered either by the addition of a new or recycled m= line to the local session description, or new ICE credentials in the description, indicating an ICE restart. Use of new ICE credentials can be triggered explicitly by the application, or implicitly by the browser in response to changes in the ICE configuration.

When a new gathering phase starts, the ICE Agent will notify the application that gathering is occurring through an event. Then, when each new ICE candidate becomes available, the ICE Agent will supply it to the application via an additional event; these candidates will also automatically be added to the local session description.

Finally, when all candidates have been gathered, an event will be dispatched to signal that the gathering process is complete.

Note that gathering phases only gather the candidates needed by new/recycled/restarting m= lines; other m= lines continue to use their existing candidates.

3.4.2. 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 media setup in cases where gathering is not performed prior to initiating the call.

JSEP supports optional candidate trickling by providing APIs, as described above, 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.

3.4.2.1. 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, with index N referring to the N+1th m= line in the SDP sent by the entity which sent the IceCandidate. The MID uses the "media stream identification" attribute, as defined in [RFC5888], Section 4, to identify the m= line. JSEP implementations creating an ICE Candidate object MUST populate both of these fields. Implementations receiving an ICE Candidate object MUST use the MID if present, or the m= line index, if not (as it could have come from a non-JSEP endpoint).

3.4.3. ICE Candidate Policy

Typically, when gathering ICE candidates, the browser will gather all possible forms of initial candidates - host, server reflexive, and relay. However, in certain cases, applications may want to have more specific control over the gathering process, due to privacy or related concerns. For example, one may want to suppress the use of host candidates, to avoid exposing information about the local network, or go as far as only using relay candidates, to leak as little location information as possible (note that these choices come with corresponding operational costs). To accomplish this, the browser MUST allow the application to restrict which ICE candidates are used in a session. In addition, administrators may also wish to control the set of ICE candidates, and so the browser SHOULD also allow control via local policy, with the most restrictive policy prevailing.

There may also be cases where the application wants to change which types of candidates are used while the session is active. A prime example is where a callee may initially want to use only relay candidates, to avoid leaking location information to an arbitrary caller, but then change to use all candidates (for lower operational cost) once the user has indicated they want to take the call. For this scenario, the browser MUST allow the candidate policy to be changed in mid-session, subject to the aforementioned interactions with local policy.

To administer the ICE candidate policy, the browser will determine the current setting at the start of each gathering phase. Then, during the gathering phase, the browser MUST NOT expose candidates disallowed by the current policy to the application, use them as the source of connectivity checks, or indirectly expose them via other fields, such as the raddr/rport attributes for other ICE candidates. Later, if a different policy is specified by the application, the application can apply it by kicking off a new gathering phase via an ICE restart.

3.4.4. ICE Candidate Pool

JSEP applications typically inform the browser to begin ICE gathering via the information supplied to setLocalDescription, as this is where the app specifies the number of media streams, and thereby ICE components, for which to gather candidates. However, to accelerate cases where the application knows the number of ICE components to use ahead of time, it may ask the browser to gather a pool of potential ICE candidates to help ensure rapid media setup.

When setLocalDescription is eventually called, and the browser goes to gather the needed ICE candidates, it SHOULD start by checking if any candidates are available in the pool. If there are candidates in the pool, they SHOULD be handed to the application immediately via the ICE candidate event. If the pool becomes depleted, either because a larger-than-expected number of ICE components is used, or because the pool has not had enough time to gather candidates, the remaining candidates are gathered as usual.

One example of where this concept is useful is an application that expects an incoming call at some point in the future, and wants to minimize the time it takes to establish connectivity, to avoid clipping of initial media. By pre-gathering candidates into the pool, it can exchange and start sending connectivity checks from these candidates almost immediately upon receipt of a call. Note though that by holding on to these pre-gathered candidates, which will be kept alive as long as they may be needed, the application will consume resources on the STUN/TURN servers it is using.

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 that 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:

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

- o 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. 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.

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. Constructor

The PeerConnection constructor allows the application to specify global parameters for the media session, such as the STUN/TURN servers and credentials to use when gathering candidates, as well as the initial ICE candidate policy and pool size, and also the BUNDLE policy to use.

If an ICE candidate policy is specified, it functions as described in Section 3.4.3, causing the browser to only surface the permitted candidates to the application, and only use those candidates for connectivity checks. The set of available policies is as follows:

all: All candidates will be gathered and used.

public: Candidates with private IP addresses [RFC1918] will be filtered out. This prevents exposure of internal network details, at the cost of requiring relay usage even for intranet calls, if the NAT does not allow hairpinning as described in [RFC4787], section 6.

relay: All candidates except relay candidates will be filtered out. This obfuscates the location information that might be ascertained by the remote peer from the received candidates. Depending on how the application deploys its relay servers, this could obfuscate location to a metro or possibly even global level.

Although it can be overridden by local policy, the default ICE candidate policy MUST be set to allow all candidates, as this minimizes use of application STUN/TURN server resources.

If a size is specified for the ICE candidate pool, this indicates the number of ICE components to pre-gather candidates for. Because pre-gathering results in utilizing STUN/TURN server resources for potentially long periods of time, this must only occur upon application request, and therefore the default candidate pool size MUST be zero.

The application can specify its preferred policy regarding use of BUNDLE, the multiplexing mechanism defined in [I-D.ietf-mmusic-sdp-bundle-negotiation]. By specifying a policy from the list below, the application can control how aggressively it will try to BUNDLE media streams together. The set of available policies is as follows:

balanced: The application will BUNDLE all media streams of the same type together. That is, if there are multiple audio and multiple video MediaStreamTracks attached to a PeerConnection, all but the first audio and video tracks will be marked as bundle-only, and candidates will only be gathered for N media streams, where N is the number of distinct media types. When talking to a non-BUNDLE-aware endpoint, only the non-bundle-only streams will be negotiated. This policy balances desire to multiplex with the need to ensure basic audio and video still works in legacy cases. Data channels will be in a separate bundle group.

max-compat: The application will offer BUNDLE, but mark none of its streams as bundle-only. This policy will allow all streams to be received by non-BUNDLE-aware endpoints, but require separate candidates to be gathered for each media stream.

max-bundle: The application will BUNDLE all of its media streams, including data channels, on a single transport. All streams other than the first will be marked as bundle-only. This policy aims to

minimize candidate gathering and maximize multiplexing, at the cost of less compatibility with legacy endpoints.

As it provides the best tradeoff between performance and compatibility with legacy endpoints, the default BUNDLE policy MUST be set to "balanced".

The application can specify its preferred policy regarding use of RTP/RTCP multiplexing [RFC5761] using one of the following policies:

negotiate: The browser will gather both RTP and RTCP candidates but also will offer "a=rtcp-mux", thus allowing for compatibility with either multiplexing or non-multiplexing endpoints.

require: The browser will only gather RTP candidates. [[OPEN ISSUE: how should the answerer behave. https://github.com/rtcweb-wg/jsep/issues/114]] This halves the number of candidates that the offerer needs to gather.

4.1.2. 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. An options parameter may be supplied to provide additional control over the generated offer. This options parameter should allow for the following manipulations to be performed:

- 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.)

- To trigger an ICE restart, for the purpose of reestablishing connectivity.

In the initial offer, the generated SDP will contain all desired functionality for the session (functionality that is 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 RFC 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.

4.1.3. 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 most recent call to setRemoteDescription, which MUST have been called prior to calling createAnswer. 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. An options 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.

4.1.4. SessionDescriptionType

Session description objects (RTCSessionDescription) may be of type "offer", "pranswer", or "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 "pranswer".

"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).

"rollback" is a special session description type implying that the state machine should be rolled back to the previous state, as described in Section 4.1.4.2. The contents MUST be empty.

4.1.4.1. Use of Provisional Answers

Most web applications will not need to create answers using the "pranswer" type. While it is good practice to send an immediate response to an "offer", in order to warm up the session transport and prevent media clipping, the preferred handling for a web application would be to create and send an "inactive" final answer immediately after receiving the offer. Later, when the called user actually accepts the call, the application can create a new "sendrecv" offer to update the previous offer/answer pair and start the media flow. While this could also be done with an inactive "pranswer", followed by a sendrecv "answer", the initial "pranswer" leaves the offer-answer exchange open, which means that neither side can send an updated offer during this time.

As an example, 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 triggered the new offer, it is likely that the ICE and DTLS handshaking for all the channels will already have finished.

Of course, some applications may not be able to perform this double offer-answer exchange, particularly ones that are attempting to gateway to legacy signaling protocols. In these cases, "pranswer" can still provide the application with a mechanism to warm up the transport.

4.1.4.2. 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 discards any proposed changes to the session, returning the state machine to the stable state, and setting the modified local and/or remote description back to their previous values. Any

resources or candidates that were allocated by the abandoned local description are discarded; any media that is received will be processed according to the previous local and remote descriptions. Rollback can only be used to cancel proposed changes; there is no support for rolling back from a stable state to a previous stable state. Note that this implies that once the answerer has performed setLocalDescription with his answer, this cannot be rolled back.

A rollback is performed by supplying a session description of type "rollback" with empty contents to either setLocalDescription or setRemoteDescription, depending on which was most recently used (i.e. if the new offer was supplied to setLocalDescription, the rollback should be done using setLocalDescription as well).

4.1.5. 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 previous 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. 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 setLocalDescription was previously called with an offer, and setRemoteDescription 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.

4.1.7. 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.

[[OPEN ISSUE: Do we need to expose accessors for both the current and proposed local description? https://github.com/rtcweb-wg/jsep/issues/16]]

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

4.1.8. 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.

[[OPEN ISSUE: Do we need to expose accessors for both the current and proposed remote description? https://github.com/rtcweb-wg/jsep/issues/16]]

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

4.1.9. canTrickleIceCandidates

The canTrickleIceCandidates property indicates whether the remote side supports receiving trickled candidates. There are three potential values:

null: No SDP has been received from the other side, so it is not known if it can handle trickle. This is the initial value before setRemoteDescription() is called.

true: SDP has been received from the other side indicating that it can support trickle.

false: SDP has been received from the other side indicating that it cannot support trickle.

As described in Section 3.4.2, JSEP implementations always provide candidates to the application individually, consistent with what is needed for Trickle ICE. However, applications can use the canTrickleIceCandidates property to determine whether their peer can actually do Trickle ICE, i.e., whether it is safe to send an initial offer or answer followed later by candidates as they are gathered. As "true" is the only value that definitively indicates remote Trickle ICE support, an application which compares canTrickleIceCandidates against "true" will by default attempt Half Trickle on initial offers and Full Trickle on subsequent interactions with a Trickle ICE-compatible agent.

4.1.10. setConfiguration

The setConfiguration method allows the global configuration of the PeerConnection, which was initially set by constructor parameters, to be changed during the session. The effects of this method call depend on when it is invoked, and differ depending on which specific parameters are changed:

- Any changes to the STUN/TURN servers to use affect the next gathering phase. If gathering has already occurred, this will cause the next call to createOffer to generate new ICE credentials, for the purpose of forcing an ICE restart and kicking off a new gathering phase, in which the new servers will be used. If the ICE candidate pool has a nonzero size, any existing candidates will be discarded, and new candidates will be gathered from the new servers.

- Any changes to the ICE candidate policy also affect the next gathering phase, in similar fashion to the server changes described above. Note though that changes to the policy have no effect on the candidate pool, because pooled candidates are not surfaced to the application until a gathering phase occurs, and so any necessary filtering can still be done on any pooled candidates.

- Any changes to the ICE candidate pool size take effect immediately; if increased, additional candidates are pre-gathered; if decreased, the now-superfluous candidates are discarded.

- o The BUNDLE and RTCP-multiplexing policies MUST NOT be changed after the construction of the PeerConnection.

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.

4.1.11. 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 [RFC5764] and TCP/DTLS/RTP/SAVPF [I-D.nandakumar-mmusic-proto-iana-registration] RTP profiles.

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 and grouping semantics.

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 [RFC4588] MUST be implemented to signal RTX payload type associations.

R-15 [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.

R-16 TODO: any others?

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 ICE, as specified in [RFC5245], MUST be used. Note that the remote endpoint may use a Lite implementation; implementations MUST properly handle remote endpoints which do ICE-Lite.

R-2 DTLS [RFC6347] or DTLS-SRTP [RFC5763], MUST be used, as appropriate for the media type, as specified in [I-D.ietf-rtcweb-security-arch]

5.1.3. Profile Names and Interoperability

For media m= sections, JSEP endpoints MUST support both the "UDP/TLS/RTP/SAVPF" and "TCP/DTLS/RTP/SAVPF" profiles and MUST indicate one of these two profiles for each media m= line they produce in an offer. For data m= sections, JSEP endpoints must support both the "UDP/DTLS/SCTP" and "TCP/DTLS/SCTP" profiles and MUST indicate one of these two profiles for each data m= line they produce in an offer. Because ICE can select either TCP or UDP transport depending on network conditions, both advertisements are consistent with ICE eventually selecting either either UDP or TCP.

Unfortunately, in an attempt at compatibility, some endpoints generate other profile strings even when they mean to support one of these profiles. For instance, an endpoint might generate "RTP/AVP" but supply "a=fingerprint" and "a=rtcp-fb" attributes, indicating its willingness to support "(UDP,TCP)/TLS/RTP/SAVPF". In order to simplify compatibility with such endpoints, JSEP endpoints MUST follow the following rules when processing the media m= sections in an offer:

- The profile in any "m=" line in any answer MUST exactly match the profile provided in the offer.

- Any profile matching the following patterns MUST be accepted: "RTP/[S]AVP[F]" and "(UDP/TCP)/TLS/RTP/SAVP[F]"

- Because DTLS-SRTP is REQUIRED, the choice of SAVP or AVP has no effect; support for DTLS-SRTP is determined by the presence of the "a=fingerprint" attribute. Note that lack of an "a=fingerprint" attribute will lead to negotiation failure.

- o The use of AVPF or AVP simply controls the timing rules used for RTCP feedback. If AVPF is provided, or an "a=rtcp-fb" attribute is present, assume AVPF timing, i.e. a default value of "trr-int=0". Otherwise, assume that AVPF is being used in an AVP compatible mode and use AVP timing, i.e., "trr-int=4".

- o For data m= sections, JSEP endpoints MUST support receiving the "UDP/ DTLS/SCTP", "TCP/DTLS/SCTP", or "DTLS/SCTP" (for backwards compatibility) profiles.

Note that re-offers by JSEP endpoints MUST use the correct profile strings even if the initial offer/answer exchange used an (incorrect) older profile string.

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=-". Note that this differs from the

advice in [RFC4566] which proposes a single space, but as both "o=" and "s=" are meaningless, having the same meaningless value seems clearer.

- 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, as specified in [RFC4566] Section 5.14, 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.) m=sections MUST be sorted first by the order in which the MediaStreams were added to the PeerConnection, and then by 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. If a second MediaStream containing an audio MediaStreamTrack was added, it would follow 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 as the m= section into which it is being bundled. Note that this means that for offers, any m= sections which are not bundle-only MUST have unique ICE credentials and candidates, since it is possible that the answerer will accept them without bundling them.

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, or this value MUST be a session-level attribute.

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, but given that no candidates have yet been gathered, the "dummy" port value of 9 (Discard) MUST be used, as indicated in [I-D.ietf-mmusic-trickle-ice], Section 5.1.

- o To properly indicate use of DTLS, the <proto> field MUST be set to "UDP/TLS/RTP/SAVPF", as specified in [RFC5764], Section 8, if the default candidate uses UDP transport, or "TCP/DTLS/RTP/SAVPF", as specified in[I-D.nandakumar-mmusic-proto-iana-registration] if the default candidate uses TCP transport.

The m= line MUST be followed immediately by a "c=" line, as specified in [RFC4566], Section 5.7. Again, as no candidates have yet been gathered, the "c=" line must contain the "dummy" value "IN IP6 ::", as defined in [I-D.ietf-mmusic-trickle-ice], Section 5.1.

Each m= section MUST include the following attribute lines:

- o An "a=mid" line, as specified in [RFC5888], Section 4. When generating mid values, it is RECOMMENDED that the values be 3 bytes or less, to allow them to efficiently fit into the RTP header extension defined in [I-D.ietf-mmusic-sdp-bundle-negotiation], Section 11.

- o An "a=rtcp" line, as specified in [RFC3605], Section 2.1, containing the dummy value "9 IN IP6 ::", because no candidates have yet been gathered.

- o An "a=msid" line, as specified in [I-D.ietf-mmusic-msid], Section 2.

- 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. The audio and video codecs that MUST be supported are specified in [I-D.ietf-rtcweb-audio] (see Section 3) and [I-D.ietf-rtcweb-video] (see Section 5).

- o If this m= section is for media with configurable frame sizes, e.g. audio, an "a=maxptime" line, indicating the smallest of the maximum supported frame sizes out of all codecs included above, as specified in [RFC4566], Section 6.

- 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 of the primary codec, as specified in [RFC4588], Section 8.1.

- For each supported FEC mechanism, "a=rtpmap" and "a=fmtp" lines, as specified in [RFC4566], Section 6. The FEC mechanisms that MUST be supported are specified in [I-D.ietf-rtcweb-fec], Section 6, and specific usage for each media type is outlined in Sections 4 and 5.

- "a=ice-ufrag" and "a=ice-passwd" lines, as specified in [RFC5245], Section 15.4.

- An "a=ice-options" line, with the "trickle" option, as specified in [I-D.ietf-mmusic-trickle-ice], Section 4.

- An "a=fingerprint" line, as specified in [RFC4572], Section 5; the algorithm used for the fingerprint MUST match that used in the certificate signature.

- 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".

- An "a=rtcp-mux" line, as specified in [RFC5761], Section 5.1.1.

- An "a=rtcp-rsize" line, as specified in [RFC5506], Section 5.

- 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.

- 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.

- 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 [RFC7022]. [OPEN ISSUE: How are CNAMEs specified for MSTs? Are they randomly generated for each

MediaStream? If so, can two MediaStreams be synced? See: https://github.com/rtcweb-wg/jsep/issues/4]

- 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 with semantics set to "FEC-FR" and including the primary and FEC SSRCs, as specified in [RFC5956], section 4.3. For simplicity, if both RTX and FEC are supported, the FEC SSRC MUST be the same as the RTX SSRC.

- o [OPEN ISSUE: Handling of a=imageattr]

- o If the BUNDLE policy for this PeerConnection is set to "max-bundle", and this is not the first m= section, or the BUNDLE policy is set to "balanced", and this is not the first m= section for this media type, an "a=bundle-only" line.

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 "UDP/DTLS/SCTP" if the default candidate uses UDP transport, or "TCP/DTLS/SCTP" if the default candidate uses TCP transport [I-D.ietf-mmusic-sctp-sdp]. The "fmt" value MUST be set to the SCTP port number, as specified in Section 4.1. [TODO: update this to use a=sctp-port, as indicated in the latest data channel docs]

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 -01 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 MUST include the mid identifiers of each m= section. The effect of this is that the browser offers all m= sections as one BUNDLE group. However, whether the m= sections are bundle-only or not depends on the BUNDLE policy.

Attributes which SDP permits to either be at the session level or the media level SHOULD generally be at the media level even if they are identical. This promotes readability, especially if one of a set of initially identical attributes is subsequently changed.

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:

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

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

Note that these requirements are in some cases stricter than those of SDP. Implementations MUST be prepared to accept compliant SDP even if it would not conform to the requirements for generating SDP in this specification.

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:

- 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:

- The "s=" and "t=" lines MUST stay the same.
- Each "m=" and c=" line MUST be filled in with the port, protocol, and address of the default candidate for the m= section, as described in [RFC5245], Section 4.3. Each "a=rtcp" attribute line MUST also be filled in with the port and address of the

appropriate default candidate, either the default RTP or RTCP candidate, depending on whether RTCP multiplexing is currently active or not. Note that if RTCP multiplexing is being offered, but not yet active, the default RTCP candidate MUST be used, as indicated in [RFC5761], section 5.1.3. In each case, if no candidates of the desired type have yet been gathered, dummy values MUST be used, as described above.

- o Each "a=mid" line MUST stay the same.

- o Each "a=ice-ufrag" and "a=ice-pwd" line MUST stay the same, unless the ICE configuration has changed (either changes to the supported STUN/TURN servers, or the ICE candidate policy), or the "IceRestart" option (Section 5.2.3.3 was specified.

- o Within each m= section, for each candidate that has been gathered during the most recent gathering phase (see Section 3.4.1), an "a=candidate" line MUST be added, as specified in [RFC5245], Section 4.3., paragraph 3. If candidate gathering for the section has completed, an "a=end-of-candidates" attribute MUST be added, as described in [I-D.ietf-mmusic-trickle-ice], Section 9.3.

- 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.

- o If any MediaStreamTracks have been added, and there exist m= sections of the appropriate media type with no associated MediaStreamTracks (i.e. as described in the preceding paragraph), those m= sections MUST be recycled by adding the new MediaStreamTrack to the m= section. This is done by adding the necessary "a=msid", "a=ssrc", and "a=ssrc-group" lines to the recycled m= section, and removing the "a=recvonly" attribute.

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 exists in the current local description, but does not have an associated local MediaStreamTrack (possibly because said MediaStreamTrack was removed since the last exchange), a m= section MUST still be generated in the new offer, as indicated in [RFC3264], Section 8. The disposition of this section will depend on the state of the remote MediaStreamTrack associated with this m= section. If one exists, and it is still in the "live" state, the new m= section MUST be marked as "a=recvonly", with no "a=msid" or related attributes present. If no remote MediaStreamTrack exists, or it is in the "ended" state, the m= section MUST be marked as rejected, by setting the port to zero, as indicated in [RFC3264], Section 8.2.

- o If any MediaStreamTracks have been added, and there exist recvonly m= sections of the appropriate media type with no associated MediaStreamTracks, or rejected m= sections of any media type, those m= sections MUST be recycled, and a local MediaStreamTrack associated with these recycled m= sections until all such existing m= sections have been used. This includes any recvonly or rejected m= sections created by the preceding paragraph.

In addition, for each non-recycled, 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.

The "a=group:BUNDLE" attribute MUST include the mid identifiers specified in the BUNDLE group in the most recent answer, minus any m= sections that have been marked as rejected, plus any newly added or re-enabled m= sections. In other words, the BUNDLE attribute must contain all m= sections that were previously bundled, as long as they are still alive, as well as any new m= sections.

5.2.3. Options Handling

The createOffer method takes as a parameter an RTCOfferOptions object. Special processing is performed when generating a SDP description if the following options are present.

5.2.3.1. OfferToReceiveAudio

If the "OfferToReceiveAudio" option is specified, with an integer value of N, and M audio MediaStreamTracks have been added to the PeerConnection, the offer MUST include N non-rejected m= sections with media type "audio", even if N is greater than M. This allows the offerer to receive audio, including multiple independent streams, even when not sending it; accordingly, the directional attribute on the N-M audio m= sections without associated MediaStreamTracks MUST be set to recvonly.

If N is set to a value less than M, the offer MUST mark the m= sections associated with the M-N most recently added (since the last setLocalDescription) MediaStreamTracks as sendonly. This allows the offerer to indicate that it does not want to receive audio on some or all of its newly created streams. For m= sections that have previously been negotiated, this setting has no effect. [TODO: refer to RTCRtpSender in the future]

For backwards compatibility with pre-standard versions of this specification, a value of "true" is interpreted as equivalent to N=1, and "false" as N=0.

5.2.3.2. OfferToReceiveVideo

If the "OfferToReceiveVideo" option is specified, with an integer value of N, and M video MediaStreamTracks have been added to the PeerConnection, the offer MUST include N non-rejected m= sections with media type "video", even if N is greater than M. This allows the offerer to receive video, including multiple independent streams, even when not sending it; accordingly, the directional attribute on the N-M video m= sections without associated MediaStreamTracks MUST be set to recvonly.

If N is set to a value less than M, the offer MUST mark the m= sections associated with the M-N most recently added (since the last setLocalDescription) MediaStreamTracks as sendonly. This allows the offerer to indicate that it does not want to receive video on some or all of its newly created streams. For m= sections that have previously been negotiated, this setting has no effect. [TODO: refer to RTCRtpSender in the future]

For backwards compatibility with pre-standard versions of this specification, a value of "true" is interpreted as equivalent to N=1, and "false" as N=0.

5.2.3.3. IceRestart

If the "IceRestart" option 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 option is specified on an initial offer, it has no effect (since a new ICE ufrag and pwd are already generated). Similarly, if the ICE configuration has changed, this option has no effect, since new ufrag and pwd attributes will be generated automatically. This option is primarily useful for reestablishing connectivity in cases where failures are detected by the application.

5.2.3.4. VoiceActivityDetection

If the "VoiceActivityDetection" option is specified, with a value of "true", the offer MUST indicate support for silence suppression in the audio it receives by including comfort noise ("CN") codecs for each offered audio codec, as specified in [RFC3389], Section 5.1, except for codecs that have their own internal silence suppression support. For codecs that have their own internal silence suppression support, the appropriate fmtp parameters for that codec MUST be specified to indicate that silence suppression for received audio is desired. For example, when using the Opus codec, the "usedtx=1" parameter would be specified in the offer. This option allows the endpoint to significantly reduce the amount of audio bandwidth it receives, at the cost of some fidelity, depending on the quality of the remote VAD algorithm.

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.

The next step is to go through each offered m= section. If there is a local MediaStreamTrack of the same type which has been added to the PeerConnection via addStream and not yet associated with a m= section, and the specific m= section is either sendrecv or recvonly, the MediaStreamTrack will be associated with the m= section at this time. MediaStreamTracks are assigned to m= sections using the canonical order described in Section 5.2.1. 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 compatible 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.

For each offered m= section, if the associated remote MediaStreamTrack has been stopped, and is therefore in state "ended", and no local MediaStreamTrack has been associated, 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 further processing for this m= section can be skipped.

Provided that is not the case, each m= section in the answer should then be generated as specified in [RFC3264], Section 6.1. For the m= line itself, the following rules must be followed:

- o The port value would normally be set to the port of the default ICE candidate for this m= section, but given that no candidates have yet been gathered, the "dummy" port value of 9 (Discard) MUST be used, as indicated in [I-D.ietf-mmusic-trickle-ice], Section 5.1.

- o The <proto> field MUST be set to exactly match the <proto> field for the corresponding m= line in the offer.

The m= line MUST be followed immediately by a "c=" line, as specified in [RFC4566], Section 5.7. Again, as no candidates have yet been gathered, the "c=" line must contain the "dummy" value "IN IP6 ::", as defined in [I-D.ietf-mmusic-trickle-ice], Section 5.1.

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 An "a=rtcp" line, as specified in [RFC3605], Section 2.1, containing the dummy value "9 IN IP6 ::", because no candidates have yet been gathered.

- o If a local MediaStreamTrack has been associated, an "a=msid" line, as specified in [I-D.ietf-mmusic-msid], Section 2.

- o Depending on the directionality of the offer, the disposition of any associated remote MediaStreamTrack, and the presence of an associated local MediaStreamTrack, the appropriate directionality attribute, as specified in [RFC3264], Section 6.1. If the offer was sendrecv, and the remote MediaStreamTrack is still "live", and there is a local MediaStreamTrack that has been associated, the directionality MUST be set as sendrecv. If the offer was sendonly, and the remote MediaStreamTrack is still "live", the directionality MUST be set as recvonly. If the offer was recvonly, and a local MediaStreamTrack has been associated, the directionality MUST be set as sendonly. If the offer was inactive, the directionality MUST be set as inactive.

- 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. The audio and video codecs that MUST be supported are specified in [I-D.ietf-rtcweb-audio] (see Section 3) and [I-D.ietf-rtcweb-video] (see Section 5). 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 this m= section is for media with configurable frame sizes, e.g. audio, an "a=maxptime" line, indicating the smallest of the

maximum supported frame sizes out of all codecs included above, as specified in [RFC4566], Section 6.

- 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 of the primary codec, as specified in [RFC4588], Section 8.1.

- o For each supported FEC mechanism, "a=rtpmap" and "a=fmtp" lines, as specified in [RFC4566], Section 6. The FEC mechanisms that MUST be supported are specified in [I-D.ietf-rtcweb-fec], Section 6, and specific usage for each media type is outlined in Sections 4 and 5.

- 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 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. If the "require" RTCP multiplexing policy is set and no "a=rtcp-mux" line is present in the offer, then the m=line MUST be marked as rejected by setting the port in the m= line to zero, as indicated in [RFC3264], Section 6.

- 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.

- 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.

- 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.

- 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.

- 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 with semantics set to "FEC-FR" and including the primary and FEC SSRCs, as specified in [RFC5956], section 4.3. For simplicity, if both RTX and FEC are supported, the FEC SSRC MUST be the same as the RTX SSRC.

- [OPEN ISSUE: Handling of a=imageattr]

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 exactly match the field in the offer; the "fmt" value MUST be set to the SCTP port number, as specified in Section 4.1. [TODO: update this to use a=sctp-port, as indicated in the latest data channel docs]

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 -01 of this document is missing this information.]

If "a=group" attributes with semantics of "BUNDLE" are offered, corresponding session-level "a=group" attributes MUST be added as specified in [RFC5888]. These attributes MUST have semantics "BUNDLE", and MUST include the all mid identifiers from the offered BUNDLE groups that have not been rejected. Note that regardless of the presence of "a=bundle-only" in the offer, no m= sections in the answer should have an "a=bundle-only" line.

Attributes that are common between all m= sections MAY be moved to session-level, if explicitly defined to be valid at session-level.

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

5.3.2. Subsequent Answers

When createAnswer 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 answer.

If the initial answer was not applied using setLocalDescription, meaning the PeerConnection is still in the "have-remote-offer" state, the steps for generating an initial answer 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 from the previously generated answer.

If any session description was previously supplied to setLocalDescription, an answer is generated by following the steps in the "have-remote-offer" state above, along with these exceptions:

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

- o Each "m=" and c=" line MUST be filled in with the port and address of the default candidate for the m= section, as described in [RFC5245], Section 4.3. Note, however, that the m= line protocol need not match the default candidate, because this protocol value must instead match what was supplied in the offer, as described above. Each "a=rtcp" attribute line MUST also be filled in with the port and address of the appropriate default candidate, either the default RTP or RTCP candidate, depending on whether RTCP multiplexing is enabled in the answer. In each case, if no candidates of the desired type have yet been gathered, dummy values MUST be used, as described in the initial answer section above.

- o Each "a=ice-ufrag" and "a=ice-pwd" line MUST stay the same.

- o Within each m= section, for each candidate that has been gathered during the most recent gathering phase (see Section 3.4.1), an "a=candidate" line MUST be added, as specified in [RFC5245], Section 4.3., paragraph 3. If candidate gathering for the section

has completed, an "a=end-of-candidates" attribute MUST be added, as described in [I-D.ietf-mmusic-trickle-ice], Section 9.3.

- For MediaStreamTracks that are still present, the "a=msid", "a=ssrc", and "a=ssrc-group" lines MUST stay the same.

5.3.3. Options Handling

The createAnswer method takes as a parameter an RTCAnswerOptions object. The set of parameters for RTCAnswerOptions is different than those supported in RTCOfferOptions; the OfferToReceiveAudio, OfferToReceiveVideo, and IceRestart options mentioned in Section 5.2.3 are meaningless in the context of generating an answer, as there is no need to generate extra m= lines in an answer, and ICE credentials will automatically be changed for all m= lines where the offerer chose to perform ICE restart.

The following options are supported in RTCAnswerOptions.

5.3.3.1. VoiceActivityDetection

Silence suppression in the answer is handled as described in Section 5.2.3.4.

5.4. Processing a Local Description

When a SessionDescription is supplied to setLocalDescription, the following steps MUST be performed:

- First, the type of the SessionDescription is checked against the current state of the PeerConnection:
  - If the type is "offer", the PeerConnection state MUST be either "stable" or "have-local-offer".
  - If the type is "pranswer" or "answer", the PeerConnection state MUST be either "have-remote-offer" or "have-local-pranswer".
- If the type is not correct for the current state, processing MUST stop and an error MUST be returned.
- Next, the SessionDescription is parsed into a data structure, as described in the Section 5.6 section below. If parsing fails for any reason, processing MUST stop and an error MUST be returned.
- Finally, the parsed SessionDescription is applied as described in the Section 5.7 section below.

5.5. Processing a Remote Description

When a SessionDescription is supplied to setRemoteDescription, the following steps MUST be performed:

- o First, the type of the SessionDescription is checked against the current state of the PeerConnection:
  - * If the type is "offer", the PeerConnection state MUST be either "stable" or "have-remote-offer".
  - * If the type is "pranswer" or "answer", the PeerConnection state MUST be either "have-local-offer" or "have-remote-pranswer".
- o If the type is not correct for the current state, processing MUST stop and an error MUST be returned.
- o Next, the SessionDescription is parsed into a data structure, as described in the Section 5.6 section below. If parsing fails for any reason, processing MUST stop and an error MUST be returned.
- o Finally, the parsed SessionDescription is applied as described in the Section 5.8 section below.

5.6. Parsing a Session Description

[The behavior described herein is a draft version, and needs more discussion to resolve various open issues.]

When a SessionDescription of any type is supplied to setLocal/RemoteDescription, the implementation must parse it and reject it if it is invalid. The exact details of this process are explained below.

The SDP contained in the session description object consists of a sequence of text lines, each containing a key-value expression, as described in [RFC4566], Section 5. The SDP is read, line-by-line, and converted to a data structure that contains the deserialized information. However, SDP allows many types of lines, not all of which are relevant to JSEP applications. For each line, the implementation will first ensure it is syntactically correct according its defining ABNF [TODO: reference], check that it conforms to [RFC4566] and [RFC3264] semantics, and then either parse and store or discard the provided value, as described below. [TODO: ensure that every line is listed below.] If the line is not well-formed, or cannot be parsed as described, the parser MUST stop with an error and reject the session description. This ensures that implementations do not accidentally misinterpret ambiguous SDP.

5.6.1. Session-Level Parsing

First, the session-level lines are checked and parsed. These lines MUST occur in a specific order, and with a specific syntax, as defined in [RFC4566], Section 5. Note that while the specific line types (e.g. "v=", "c=") MUST occur in the defined order, lines of the same type (typically "a=") can occur in any order, and their ordering is not meaningful.

For non-attribute (non-"a=") lines, their sequencing, syntax, and semantics, are checked, as mentioned above. The following lines are not meaningful in the JSEP context and MAY be discarded once they have been checked.

TODO

The remaining lines are processed as follows:

The "c=" line MUST be parsed and stored.

[OPEN ISSUE: For example, because session-level bandwidth is ambiguous when multiple media streams are present, a "b=" line at session level is not useful and its value SHOULD be ignored. [OPEN ISSUE: is this WG consensus? Are there other non-a= lines that we need to do more than just syntactical validation, e.g. v=?]

Specific processing MUST be applied for the following session-level attribute ("a=") lines:

- Any "a=group" lines are parsed as specified in [RFC5888], Section 5, and the group’s semantics and mids are stored.

- If present, a single "a=ice-lite" line is parsed as specified in [RFC5245], Section 15.3, and a value indicating the presence of ice-lite is stored.

- If present, a single "a=ice-ufrag" line is parsed as specified in [RFC5245], Section 15.4, and the ufrag value is stored.

- If present, a single "a=ice-pwd" line is parsed as specified in [RFC5245], Section 15.4, and the password value is stored.

- If present, a single "a=ice-options" line is parsed as specified in [RFC5245], Section 15.5, and the set of specified options is stored.

- o Any "a=fingerprint" lines are parsed as specified in [RFC4572], Section 5, and the set of fingerprint and algorithm values is stored.

- o If present, a single "a=setup" line is parsed as specified in [RFC4145], Section 4, and the setup value is stored.

- o Any "a=extmap" lines are parsed as specified in [RFC5285], Section 5, and their values are stored.

- o TODO: msid-semantic, identity, rtcp-rsize, rtcp-mux, and any other attribs valid at session level.

Once all the session-level lines have been parsed, processing continues with the lines in media sections.

5.6.2. Media Section Parsing

Like the session-level lines, the media session lines MUST occur in the specific order and with the specific syntax defined in [RFC4566], Section 5.

The "m=" line itself MUST be parsed as described in [RFC4566], Section 5.14, and the media, port, proto, and fmt values stored.

Following the "m=" line, specific processing MUST be applied for the following non-attribute lines:

- o The "c=" line, if present, MUST be parsed as specified in [RFC4566], Section 5.7, and its contents stored.

- o The "b=" line, if present, MUST be parsed as specified in [RFC4566], Section 5.8, and the bwtype and bandwidth values stored.

Specific processing MUST also be applied for the following attribute lines:

- o If present, a single "a=ice-lite" line is parsed as specified in [RFC5245], Section 15.3, and a value indicating the presence of ice-lite is stored.

- o If present, a single "a=ice-ufrag" line is parsed as specified in [RFC5245], Section 15.4, and the ufrag value is stored.

- o If present, a single "a=ice-pwd" line is parsed as specified in [RFC5245], Section 15.4, and the password value is stored.

- o If present, a single "a=ice-options" line is parsed as specified in [RFC5245], Section 15.5, and the set of specified options is stored.

- o Any "a=fingerprint" lines are parsed as specified in [RFC4572], Section 5, and the set of fingerprint and algorithm values is stored.

- o If present, a single "a=setup" line is parsed as specified in [RFC4145], Section 4, and the setup value is stored.

If the "m=" proto value indicates use of RTP, as decribed in the Section 5.1.3 section above, the following attribute lines MUST be processed:

- o The "m=" fmt value MUST be parsed as specified in [RFC4566], Section 5.14, and the individual values stored.

- o Any "a=rtpmap" or "a=fmtp" lines MUST be parsed as specified in [RFC4566], Section 6, and their values stored.

- o If present, a single "a=ptime" line MUST be parsed as described in [RFC4566], Section 6, and its value stored.

- o If present, a single direction attribute line (e.g. "a=sendrecv") MUST be parsed as described in [RFC4566], Section 6, and its value stored.

- o Any "a=ssrc" or "a=ssrc-group" attributes MUST be parsed as specified in [RFC5576], Sections 4.1-4.2, and their values stored.

- o Any "a=extmap" attributes MUST be parsed as specified in [RFC5285], Section 5, and their values stored.

- o Any "a=rtcp-fb" attributes MUST be parsed as specified in [RFC4585], Section 4.2., and their values stored.

- o If present, a single "a=rtcp-mux" line MUST be parsed as specified in [RFC5761], Section 5.1.1, and its presence or absence flagged and stored.

- o TODO: a=rtcp-rsize, a=rtcp, a=msid, a=candidate, a=end-of-candidates

Otherwise, if the "m=" proto value indicats use of SCTP, the following attribute lines MUST be processed:

- o The "m=" fmt value MUST be parsed as specified in [I-D.ietf-mmusic-sctp-sdp], Section 4.3, and the application protocol value stored.

- o An "a=sctp-port" attribute MUST be present, and it MUST be parsed as specified in [I-D.ietf-mmusic-sctp-sdp], Section 5.2, and the value stored.

- o TODO: max message size

5.6.3. Semantics Verification

Assuming parsing completes successfully, the parsed description is then evaluated to ensure internal consistency as well as proper support for mandatory features. Specifically, the following checks are performed:

- o For each m= section, valid values for each of the mandatory-to-use features enumerated in Section 5.1.2 MUST be present. These values MAY either be present at the media level, or inherited from the session level.

  - * ICE ufrag and password values

  - * DTLS fingerprint and setup values

If this session description is of type "pranswer" or "answer", the following additional checks are applied:

- o The session description must follow the rules defined in [RFC3264], Section 6.

- o For each m= section, the protocol value MUST exactly match the protocol value in the corresponding m= section in the associated offer.

5.7. Applying a Local Description

The following steps are performed at the media engine level to apply a local description.

First, the parsed parameters are checked to ensure that any modifications performed fall within those explicitly permitted by Section 6; otherwise, processing MUST stop and an error MUST be returned.

Next, media sections are processed. For each media section, the following steps MUST be performed; if any parameters are out of

bounds, or cannot be applied, processing MUST stop and an error MUST be returned.

- o TODO

Finally, if this description is of type "pranswer" or "answer", follow the processing defined in the Section 5.9 section below.

5.8. Applying a Remote Description

TODO

5.9. Applying an Answer

TODO

6. Configurable SDP Parameters

It is possible to change elements in the SDP returned from createOffer before passing it to setLocalDescription. When an implementation receives modified SDP it MUST either:

- o Accept the changes and adjust its behavior to match the SDP.
- o Reject the changes and return an error via the error callback.

Changes MUST NOT be silently ignored.

The following elements of the SDP media description MUST NOT be changed between the createOffer and the setLocalDescription (or between the createAnswer 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).
- o The DTLS fingerprint(s) (a=fingerprint).

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 options passed into createOffer/createAnswer. As an open issue, these changes may also be 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 [[OPEN ISSUE: This is a placeholder for other modifications, which we may continue adding as use cases appear.]]

Implementations MAY choose to either honor or reject any elements not listed in the above two categories, but must do so explicitly as described at the beginning of this section. Note that future standards may add new SDP elements to the list of elements which must be accepted or rejected, but due to version skew, applications must be prepared for implementations to accept changes which must be rejected and vice versa.

The application can also modify the SDP to reduce the capabilities in the offer it sends to the far side or the offer that it installs from 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 was in the original offer. This is safe because the answer is not permitted to expand capabilities and therefore will just respond to what is actually in the offer.

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. Examples

Note that this example section shows several SDP fragments. To format in 72 columns, some of the lines in SDP have been split into multiple lines, where leading whitespace indicates that a line is a continuation of the previous line. In addition, some blank lines have been added to improve readability but are not valid in SDP.

More examples of SDP for WebRTC call flows can be found in [I-D.nandakumar-rtcweb-sdp].

7.1. Simple Example

This section shows a very simple example that sets up a minimal audio / video call between two browsers and does not use trickle ICE. The example in the following section provides a more realistic example of what would happen in a normal browser to browser connection.

The flow shows Alice’s browser initiating the session to Bob’s browser. The messages from Alice’s JS to Bob’s JS are assumed to flow over some signaling protocol via a web server. The JS on both Alice’s side and Bob’s side waits for all candidates before sending the offer or answer, so the offers and answers are complete. Trickle ICE is not used. Both Alice and Bob are using the default policy of balanced.

```
//                  set up local media state
AliceJS->AliceUA:   create new PeerConnection
AliceJS->AliceUA:   addStream with stream containing audio and video
AliceJS->AliceUA:   createOffer to get offer
AliceJS->AliceUA:   setLocalDescription with offer
AliceUA->AliceJS:   multiple onicecandidate events with candidates

//                  wait for ICE gathering to complete
AliceUA->AliceJS:   onicecandidate event with null candidate
AliceJS->AliceUA:   get |offer-A1| from value of localDescription

//                  |offer-A1| is sent over signaling protocol to Bob
AliceJS->WebServer: signaling with |offer-A1|
WebServer->BobJS:   signaling with |offer-A1|

//                  |offer-A1| arrives at Bob
BobJS->BobUA:       create a PeerConnection
BobJS->BobUA:       setRemoteDescription with |offer-A1|
BobUA->BobJS:       onaddstream event with remoteStream

//                  Bob accepts call
BobJS->BobUA:       addStream with local media
BobJS->BobUA:       createAnswer
BobJS->BobUA:       setLocalDescription with answer
BobUA->BobJS:       multiple onicecandidate events with candidates

//                  wait for ICE gathering to complete
BobUA->BobJS:       onicecandidate event with null candidate
BobJS->BobUA:       get |answer-A1| from value of localDescription

//                  |answer-A1| is sent over signaling protocol to Alice
BobJS->WebServer:   signaling with |answer-A1|
WebServer->AliceJS: signaling with |answer-A1|

//                  |answer-A1| arrives at Alice
AliceJS->AliceUA:   setRemoteDescription with |answer-A1|
AliceUA->AliceJS:   onaddstream event with remoteStream

//                  media flows
BobUA->AliceUA:     media sent from Bob to Alice
AliceUA->BobUA:     media sent from Alice to Bob
```

The SDP for |offer-A1| looks like:

```
v=0
o=- 4962303333179871722 1 IN IP4 0.0.0.0
s=-
t=0 0
```

```
a=msid-semantic:WMS
a=group:BUNDLE a1 v1
m=audio 56500 UDP/TLS/RTP/SAVPF 96 0 8 97 98
c=IN IP4 192.0.2.1
a=mid:a1
a=rtcp:56501 IN IP4 192.0.2.1
a=msid:47017fee-b6c1-4162-929c-a25110252400
       f83006c5-a0ff-4e0a-9ed9-d3e6747be7d9
a=sendrecv
a=rtpmap:96 opus/48000/2
a=rtpmap:0 PCMU/8000
a=rtpmap:8 PCMA/8000
a=rtpmap:97 telephone-event/8000
a=rtpmap:98 telephone-event/48000
a=maxptime:120
a=ice-ufrag:ETEn1v9DoTMB9J4r
a=ice-pwd:OtSK0WpNtpUjkY4+86js7ZQl
a=ice-options:trickle
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=rtcp-mux
a=rtcp-rsize
a=extmap:1 urn:ietf:params:rtp-hdrext:ssrc-audio-level
a=extmap:2 urn:ietf:params:rtp-hdrext:sdes:mid
a=ssrc:1732846380 cname:EocUG1f0fcg/yvY7
a=candidate:3348148302 1 udp 2113937151 192.0.2.1 56500
            typ host
a=candidate:3348148302 2 udp 2113937151 192.0.2.1 56501
            typ host
a=end-of-candidates

m=video 56502 UDP/TLS/RTP/SAVPF 100 101
c=IN IP4 192.0.2.1
a=rtcp:56503 IN IP4 192.0.2.1
a=mid:v1
a=msid:61317484-2ed4-49d7-9eb7-1414322a7aae
       f30bdb4a-5db8-49b5-bcdc-e0c9a23172e0
a=sendrecv
a=rtpmap:100 VP8/90000
a=rtpmap:101 rtx/90000
a=fmtp:101 apt=100
a=ice-ufrag:BGKkWnG5GmiUpdIV
a=ice-pwd:mqyWsAjvtKwTGnvhPztQ9mIf
a=ice-options:trickle
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=rtcp-mux
a=rtcp-rsize
a=extmap:3 urn:ietf:params:rtp-hdrext:sdes:mid
a=rtcp-fb:100 ccm fir
a=rtcp-fb:100 nack
a=rtcp-fb:100 nack pli
a=ssrc:1366781083 cname:EocUG1f0fcg/yvY7
a=ssrc:1366781084 cname:EocUG1f0fcg/yvY7
a=ssrc-group:FID 1366781083 1366781084
a=candidate:3348148302 1 udp 2113937151 192.0.2.1 56502
            typ host
a=candidate:3348148302 2 udp 2113937151 192.0.2.1 56503
            typ host
a=end-of-candidates
```

The SDP for |answer-A1| looks like:

```
v=0
o=- 6729291447651054566 1 IN IP4 0.0.0.0
s=-
t=0 0
a=msid-semantic:WMS
m=audio 20000 UDP/TLS/RTP/SAVPF 96 0 8 97 98
c=IN IP4 192.0.2.2
a=mid:a1
a=rtcp:20000 IN IP4 192.0.2.2
a=msid:PI39StLS8W7ZbQl1sJsWUXkr3Zf12fJUvzQ1
       PI39StLS8W7ZbQl1sJsWUXkr3Zf12fJUvzQ1a0
a=sendrecv
a=rtpmap:96 opus/48000/2
a=rtpmap:0 PCMU/8000
a=rtpmap:8 PCMA/8000
a=rtpmap:97 telephone-event/8000
a=rtpmap:98 telephone-event/48000
a=maxptime:120
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=rtcp-mux
a=rtcp-rsize
a=extmap:1 urn:ietf:params:rtp-hdrext:ssrc-audio-level
a=ssrc:3429951804 cname:Q/NWs1ao1HmN4Xa5
a=candidate:2299743422 1 udp 2113937151 192.0.2.2 20000
            typ host
```

```
a=end-of-candidates

m=video 20001 UDP/TLS/RTP/SAVPF 100 101
c=IN IP4 192.0.2.2
a=rtcp 20001 IN IP4 192.0.2.2
a=mid:v1
a=msid:PI39StLS8W7ZbQl1sJsWUXkr3Zf12fJUvzQ1
       PI39StLS8W7ZbQl1sJsWUXkr3Zf12fJUvzQ1v0
a=sendrecv
a=rtpmap:100 VP8/90000
a=rtpmap:101 rtx/90000
a=fmtp:101 apt=100
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=rtcp-mux
a=rtcp-rsize
a=rtcp-fb:100 ccm fir
a=rtcp-fb:100 nack
a=rtcp-fb:100 nack pli
a=ssrc:3229706345 cname:Q/NWs1ao1HmN4Xa5
a=ssrc:3229706346 cname:Q/NWs1ao1HmN4Xa5
a=ssrc-group:FID 3229706345 3229706346
a=candidate:2299743422 1 udp 2113937151 192.0.2.2 20001
            typ host
a=end-of-candidates
```

7.2. Normal Examples

This section shows a typical example of a session between two browsers setting up an audio channel and a data channel. Trickle ICE is used in full trickle mode with a bundle policy of max-bundle, an RTCP mux policy of require, and a single TURN server. Later, two video flows, one for the presenter and one for screen sharing, are added to the session. This example shows Alice's browser initiating the session to Bob's browser. The messages from Alice's JS to Bob's JS are assumed to flow over some signaling protocol via a web server.

```
//                  set up local media state
AliceJS->AliceUA:   create new PeerConnection
AliceJS->AliceUA:   addStream that contains audio track
AliceJS->AliceUA:   createDataChannel to get data channel
AliceJS->AliceUA:   createOffer to get |offer-B1|
AliceJS->AliceUA:   setLocalDescription with |offer-B1|

//                  |offer-B1| is sent over signaling protocol to Bob
```

```
AliceJS->WebServer: signaling with |offer-B1|
WebServer->BobJS:   signaling with |offer-B1|

//                  |offer-B1| arrives at Bob
BobJS->BobUA:       create a PeerConnection
BobJS->BobUA:       setRemoteDescription with |offer-B1|
BobUA->BobJS:       onaddstream with audio track from Alice

//                  candidates are sent to Bob
AliceUA->AliceJS:   onicecandidate event with |candidate-B1| (host)
AliceJS->WebServer: signaling with |candidate-B1|
AliceUA->AliceJS:   onicecandidate event with |candidate-B2| (srflx)
AliceJS->WebServer: signaling with |candidate-B2|
AliceUA->AliceJS:   onicecandidate event with |candidate-B3| (relay)
AliceJS->WebServer: signaling with |candidate-B3|

WebServer->BobJS:   signaling with |candidate-B1|
BobJS->BobUA:       addIceCandidate with |candidate-B1|
WebServer->BobJS:   signaling with |candidate-B2|
BobJS->BobUA:       addIceCandidate with |candidate-B2|
WebServer->BobJS:   signaling with |candidate-B3|
BobJS->BobUA:       addIceCandidate with |candidate-B3|

//                  Bob accepts call
BobJS->BobUA:       addStream with local audio stream
BobJS->BobUA:       createDataChannel to get data channel
BobJS->BobUA:       createAnswer to get |answer-B1|
BobJS->BobUA:       setLocalDescription with |answer-B1|

//                  |answer-B1| is sent to Alice
BobJS->WebServer:   signaling with |answer-B1|
WebServer->AliceJS: signaling with |answer-B1|
AliceJS->AliceUA:   setRemoteDescription with |answer-B1|
AliceUA->AliceJS:   onaddstream event with audio track from Bob

//                  candidates are sent to Alice
BobUA->BobJS:       onicecandidate event with |candidate-B4| (host)
BobJS->WebServer:   signaling with |candidate-B4|
BobUA->BobJS:       onicecandidate event with |candidate-B5| (srflx)
BobJS->WebServer:   signaling with |candidate-B5|
BobUA->BobJS:       onicecandidate event with |candidate-B6| (relay)
BobJS->WebServer:   signaling with |candidate-B6|

WebServer->AliceJS: signaling with |candidate-B4|
AliceJS->AliceUA:   addIceCandidate with |candidate-B4|
WebServer->AliceJS: signaling with |candidate-B5|
AliceJS->AliceUA:   addIceCandidate with |candidate-B5|
WebServer->AliceJS: signaling with |candidate-B6|
```

```
AliceJS->AliceUA:   addIceCandidate with |candidate-B6|

//                  data channel opens
BobUA->BobJS:       ondatachannel event
AliceUA->AliceJS:   ondatachannel event
BobUA->BobJS:       onopen
AliceUA->AliceJS:   onopen

//                  media is flowing between browsers
BobUA->AliceUA:     audio+data sent from Bob to Alice
AliceUA->BobUA:     audio+data sent from Alice to Bob

//                  some time later Bob adds two video streams
//                  note, no candidates exchanged, because of BUNDLE
BobJS->BobUA:       addStream with first video stream
BobJS->BobUA:       addStream with second video stream
BobJS->BobUA:       createOffer to get |offer-B2|
BobJS->BobUA:       setLocalDescription with |offer-B2|

//                  |offer-B2| is sent to Alice
BobJS->WebServer:   signaling with |offer-B2|
WebServer->AliceJS: signaling with |offer-B2|
AliceJS->AliceUA:   setRemoteDescription with |offer-B2|
AliceUA->AliceJS:   onaddstream event with first video stream
AliceUA->AliceJS:   onaddstream event with second video stream
AliceJS->AliceUA:   createAnswer to get |answer-B2|
AliceJS->AliceUA:   setLocalDescription with |answer-B2|

//                  |answer-B2| is sent over signaling protocol to Bob
AliceJS->WebServer: signaling with |answer-B2|
WebServer->BobJS:   signaling with |answer-B2|
BobJS->BobUA:       setRemoteDescription with |answer-B2|

//                  media is flowing between browsers
BobUA->AliceUA:     audio+video+data sent from Bob to Alice
AliceUA->BobUA:     audio+video+data sent from Alice to Bob
```

The SDP for |offer-B1| looks like:

```
v=0
o=- 4962303333179871723 1 IN IP4 0.0.0.0
s=-
t=0 0
a=msid-semantic:WMS
a=group:BUNDLE a1 d1
m=audio 9 UDP/TLS/RTP/SAVPF 96 0 8 97 98
c=IN IP6 ::
a=rtcp:9 IN IP6 ::
a=mid:a1
a=msid:57017fee-b6c1-4162-929c-a25110252400
       e83006c5-a0ff-4e0a-9ed9-d3e6747be7d9
a=sendrecv
a=rtpmap:96 opus/48000/2
a=rtpmap:0 PCMU/8000
a=rtpmap:8 PCMA/8000
a=rtpmap:97 telephone-event/8000
a=rtpmap:98 telephone-event/48000
a=maxptime:120
a=ice-ufrag:ATEn1v9DoTMB9J4r
a=ice-pwd:AtSK0WpNtpUjkY4+86js7ZQl
a=ice-options:trickle
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=rtcp-mux
a=rtcp-rsize
a=extmap:1 urn:ietf:params:rtp-hdrext:ssrc-audio-level
a=extmap:2 urn:ietf:params:rtp-hdrext:sdes:mid
a=ssrc:1732846380 cname:FocUG1f0fcg/yvY7

m=application 9 UDP/DTLS/SCTP webrtc-datachannel
c=IN IP6 ::
a=mid:d1
a=fmtp:webrtc-datachannel max-message-size=65536
a=sctp-port 5000
a=ice-ufrag:ATEn1v9DoTMB9J4r
a=ice-pwd:AtSK0WpNtpUjkY4+86js7ZQl
a=ice-options:trickle
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
```

The SDP for |candidate-B1| looks like:

```
candidate:109270923 1 udp 2122194687 192.168.1.2 51556 typ host
```

The SDP for |candidate-B2| looks like:

```
candidate:4036177503 1 udp 1685987071 11.22.33.44 52546 typ srflx
          raddr 192.168.1.2 rport 51556
```

The SDP for |candidate-B3| looks like:

```
candidate:3671762466 1 udp 41819903 22.33.44.55 61405 typ relay
          raddr 11.22.33.44 rport 52546
```

The SDP for |answer-B1| looks like:

```
v=0
o=- 7729291447651054566 1 IN IP4 0.0.0.0
s=-
t=0 0
a=msid-semantic:WMS
a=group:BUNDLE a1 d1
m=audio 9 UDP/TLS/RTP/SAVPF 96 0 8 97 98
c=IN IP6 ::
a=rtcp:9 IN IP6 ::
a=mid:a1
a=msid:QI39StLS8W7ZbQl1sJsWUXkr3Zf12fJUvzQ1
       QI39StLS8W7ZbQl1sJsWUXkr3Zf12fJUvzQ1a0
a=sendrecv
a=rtpmap:96 opus/48000/2
a=rtpmap:0 PCMU/8000
a=rtpmap:8 PCMA/8000
a=rtpmap:97 telephone-event/8000
a=rtpmap:98 telephone-event/48000
a=maxptime:120
a=ice-ufrag:7sFvz2gdLkEwjZEr
a=ice-pwd:dOTZKZNVlO9RSGsEGM63JXT2
a=ice-options:trickle
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=rtcp-mux
a=rtcp-rsize
a=extmap:1 urn:ietf:params:rtp-hdrext:ssrc-audio-level
a=extmap:2 urn:ietf:params:rtp-hdrext:sdes:mid
a=ssrc:4429951804 cname:Q/NWs1ao1HmN4Xa5

m=application 9 UDP/DTLS/SCTP webrtc-datachannel
c=IN IP6 ::
a=mid:d1
a=fmtp:webrtc-datachannel max-message-size=65536
a=sctp-port 5000
a=ice-ufrag:7sFvz2gdLkEwjZEr
a=ice-pwd:dOTZKZNVlO9RSGsEGM63JXT2
a=ice-options:trickle
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
```

The SDP for |candidate-B4| looks like:

```
candidate:109270924 1 udp 2122194687 192.168.2.3 61665 typ host
```

The SDP for |candidate-B5| looks like:

```
candidate:4036177504 1 udp 1685987071 55.66.77.88 64532 typ srflx
          raddr 192.168.2.3 rport 61665
```

The SDP for |candidate-B6| looks like:

```
candidate:3671762467 1 udp 41819903 66.77.88.99 50416 typ relay
          raddr 55.66.77.88 rport 64532
```

The SDP for |offer-B2| looks like: (note the increment of the version number in the o= line, and the c= and a=rtcp lines, which indicate the local candidate that was selected)

```
v=0
o=- 7729291447651054566 2 IN IP4 0.0.0.0
s=-
t=0 0
a=msid-semantic:WMS
a=group:BUNDLE a1 d1 v1 v2
m=audio 64532 UDP/TLS/RTP/SAVPF 96 0 8 97 98
c=IN IP4 55.66.77.88
a=rtcp:64532 IN IP4 55.66.77.88
a=mid:a1
a=msid:QI39StLS8W7ZbQl1sJsWUXkr3Zf12fJUvzQ1
       QI39StLS8W7ZbQl1sJsWUXkr3Zf12fJUvzQ1a0
a=sendrecv
a=rtpmap:96 opus/48000/2
a=rtpmap:0 PCMU/8000
a=rtpmap:8 PCMA/8000
a=rtpmap:97 telephone-event/8000
a=rtpmap:98 telephone-event/48000
a=maxptime:120
a=ice-ufrag:7sFvz2gdLkEwjZEr
a=ice-pwd:dOTZKZNVlO9RSGsEGM63JXT2
a=ice-options:trickle
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:actpass
a=rtcp-mux
a=rtcp-rsize
a=extmap:1 urn:ietf:params:rtp-hdrext:ssrc-audio-level
a=extmap:2 urn:ietf:params:rtp-hdrext:sdes:mid
a=ssrc:4429951804 cname:Q/NWs1ao1HmN4Xa5
a=candidate:109270924 1 udp 2122194687 192.168.2.3 61665 typ host
a=candidate:4036177504 1 udp 1685987071 55.66.77.88 64532 typ srflx
            raddr 192.168.2.3 rport 61665
a=candidate:3671762467 1 udp 41819903 66.77.88.99 50416 typ relay
            raddr 55.66.77.88 rport 64532
a=end-of-candidates
```

```
m=application 64532 UDP/DTLS/SCTP webrtc-datachannel
c=IN IP4 55.66.77.88
a=mid:d1
a=fmtp:webrtc-datachannel max-message-size=65536
a=sctp-port 5000
a=ice-ufrag:7sFvz2gdLkEwjZEr
a=ice-pwd:dOTZKZNVlO9RSGsEGM63JXT2
a=ice-options:trickle
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:actpass
a=candidate:109270924 1 udp 2122194687 192.168.2.3 61665 typ host
a=candidate:4036177504 1 udp 1685987071 55.66.77.88 64532 typ srflx
            raddr 192.168.2.3 rport 61665
a=candidate:3671762467 1 udp 41819903 66.77.88.99 50416 typ relay
            raddr 55.66.77.88 rport 64532
a=end-of-candidates

m=video 64532 UDP/TLS/RTP/SAVPF 100 101
c=IN IP4 55.66.77.88
a=rtcp:64532 IN IP4 55.66.77.88
a=mid:v1
a=msid:61317484-2ed4-49d7-9eb7-1414322a7aae
       f30bdb4a-5db8-49b5-bcdc-e0c9a23172e0
a=sendrecv
a=rtpmap:100 VP8/90000
a=rtpmap:101 rtx/90000
a=fmtp:101 apt=100
a=ice-ufrag:7sFvz2gdLkEwjZEr
a=ice-pwd:dOTZKZNVlO9RSGsEGM63JXT2
a=ice-options:trickle
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=rtcp-mux
a=rtcp-rsize
a=extmap:2 urn:ietf:params:rtp-hdrext:sdes:mid
a=rtcp-fb:100 ccm fir
a=rtcp-fb:100 nack
a=rtcp-fb:100 nack pli
a=ssrc:1366781083 cname:Q/NWs1ao1HmN4Xa5
a=ssrc:1366781084 cname:Q/NWs1ao1HmN4Xa5
a=ssrc-group:FID 1366781083 1366781084
a=candidate:109270924 1 udp 2122194687 192.168.2.3 61665 typ host
a=candidate:4036177504 1 udp 1685987071 55.66.77.88 64532 typ srflx
            raddr 192.168.2.3 rport 61665
a=candidate:3671762467 1 udp 41819903 66.77.88.99 50416 typ relay
```

```
                raddr 55.66.77.88 rport 64532
   a=end-of-candidates

   m=video 64532 UDP/TLS/RTP/SAVPF 100 101
   c=IN IP4 55.66.77.88
   a=rtcp:64532 IN IP4 55.66.77.88
   a=mid:v1
   a=msid:71317484-2ed4-49d7-9eb7-1414322a7aae
          f30bdb4a-5db8-49b5-bcdc-e0c9a23172e0
   a=sendrecv
   a=rtpmap:100 VP8/90000
   a=rtpmap:101 rtx/90000
   a=fmtp:101 apt=100
   a=ice-ufrag:7sFvz2gdLkEwjZEr
   a=ice-pwd:dOTZKZNVlO9RSGsEGM63JXT2
   a=ice-options:trickle
   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=rtcp-mux
   a=rtcp-rsize
   a=extmap:2 urn:ietf:params:rtp-hdrext:sdes:mid
   a=rtcp-fb:100 ccm fir
   a=rtcp-fb:100 nack
   a=rtcp-fb:100 nack pli
   a=ssrc:2366781083 cname:Q/NWs1ao1HmN4Xa5
   a=ssrc:2366781084 cname:Q/NWs1ao1HmN4Xa5
   a=ssrc-group:FID 2366781083 2366781084
   a=candidate:109270924 1 udp 2122194687 192.168.2.3 61665 typ host
   a=candidate:4036177504 1 udp 1685987071 55.66.77.88 64532 typ srflx
                raddr 192.168.2.3 rport 61665
   a=candidate:3671762467 1 udp 41819903 66.77.88.99 50416 typ relay
                raddr 55.66.77.88 rport 64532
   a=end-of-candidates
```

The SDP for |answer-B2| looks like: (note the use of setup:passive to maintain the existing DTLS roles, and the use of a=recvonly to indicate that the video streams are one-way)

```
   v=0
   o=- 4962303333179871723 2 IN IP4 0.0.0.0
   s=-
   t=0 0
   a=msid-semantic:WMS
   a=group:BUNDLE a1 d1 v1 v2
   m=audio 52546 UDP/TLS/RTP/SAVPF 96 0 8 97 98
```

```
c=IN IP4 11.22.33.44
a=rtcp:52546 IN IP4 11.22.33.44
a=mid:a1
a=msid:57017fee-b6c1-4162-929c-a25110252400
       e83006c5-a0ff-4e0a-9ed9-d3e6747be7d9
a=sendrecv
a=rtpmap:96 opus/48000/2
a=rtpmap:0 PCMU/8000
a=rtpmap:8 PCMA/8000
a=rtpmap:97 telephone-event/8000
a=rtpmap:98 telephone-event/48000
a=maxptime:120
a=ice-ufrag:ATEn1v9DoTMB9J4r
a=ice-pwd:AtSK0WpNtpUjkY4+86js7ZQl
a=ice-options:trickle
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:passive
a=rtcp-mux
a=rtcp-rsize
a=extmap:1 urn:ietf:params:rtp-hdrext:ssrc-audio-level
a=extmap:2 urn:ietf:params:rtp-hdrext:sdes:mid
a=ssrc:1732846380 cname:FocUG1f0fcg/yvY7
a=candidate:109270923 1 udp 2122194687 192.168.1.2 51556 typ host
a=candidate:4036177503 1 udp 1685987071 11.22.33.44 52546 typ srflx
            raddr 192.168.1.2 rport 51556
a=candidate:3671762466 1 udp 41819903 22.33.44.55 61405 typ relay
            raddr 11.22.33.44 rport 52546
a=end-of-candidates

m=application 52546 UDP/DTLS/SCTP webrtc-datachannel
c=IN IP4 11.22.33.44
a=mid:d1
a=fmtp:webrtc-datachannel max-message-size=65536
a=sctp-port 5000
a=ice-ufrag:ATEn1v9DoTMB9J4r
a=ice-pwd:AtSK0WpNtpUjkY4+86js7ZQl
a=ice-options:trickle
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:passive
a=candidate:109270923 1 udp 2122194687 192.168.1.2 51556 typ host
a=candidate:4036177503 1 udp 1685987071 11.22.33.44 52546 typ srflx
            raddr 192.168.1.2 rport 51556
a=candidate:3671762466 1 udp 41819903 22.33.44.55 61405 typ relay
            raddr 11.22.33.44 rport 52546
a=end-of-candidates
```

```
m=video 52546 UDP/TLS/RTP/SAVPF 100 101
c=IN IP4 11.22.33.44
a=rtcp:52546 IN IP4 11.22.33.44
a=mid:v1
a=recvonly
a=rtpmap:100 VP8/90000
a=rtpmap:101 rtx/90000
a=fmtp:101 apt=100
a=ice-ufrag:ATEn1v9DoTMB9J4r
a=ice-pwd:AtSK0WpNtpUjkY4+86js7ZQl
a=ice-options:trickle
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:passive
a=rtcp-mux
a=rtcp-rsize
a=extmap:2 urn:ietf:params:rtp-hdrext:sdes:mid
a=rtcp-fb:100 ccm fir
a=rtcp-fb:100 nack
a=rtcp-fb:100 nack pli
a=candidate:109270923 1 udp 2122194687 192.168.1.2 51556 typ host
a=candidate:4036177503 1 udp 1685987071 11.22.33.44 52546 typ srflx
          raddr 192.168.1.2 rport 51556
a=candidate:3671762466 1 udp 41819903 22.33.44.55 61405 typ relay
          raddr 11.22.33.44 rport 52546
a=end-of-candidates

m=video 52546 UDP/TLS/RTP/SAVPF 100 101
c=IN IP4 11.22.33.44
a=rtcp:52546 IN IP4 11.22.33.44
a=mid:v2
a=recvonly
a=rtpmap:100 VP8/90000
a=rtpmap:101 rtx/90000
a=fmtp:101 apt=100
a=ice-ufrag:ATEn1v9DoTMB9J4r
a=ice-pwd:AtSK0WpNtpUjkY4+86js7ZQl
a=ice-options:trickle
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:passive
a=rtcp-mux
a=rtcp-rsize
a=extmap:2 urn:ietf:params:rtp-hdrext:sdes:mid
a=rtcp-fb:100 ccm fir
a=rtcp-fb:100 nack
```

```
a=rtcp-fb:100 nack pli
a=candidate:109270923 1 udp 2122194687 192.168.1.2 51556 typ host
a=candidate:4036177503 1 udp 1685987071 11.22.33.44 52546 typ srflx
            raddr 192.168.1.2 rport 51556
a=candidate:3671762466 1 udp 41819903 22.33.44.55 61405 typ relay
            raddr 11.22.33.44 rport 52546
a=end-of-candidates
```

8. Security Considerations

The IETF has published separate documents [I-D.ietf-rtcweb-security-arch] [I-D.ietf-rtcweb-security] describing the security architecture for WebRTC as a whole. The remainder of this section describes security considerations for this document.

While formally the JSEP interface is an API, it is better to think of it is an Internet protocol, with the JS being untrustworthy from the perspective of the browser. Thus, the threat model of [RFC3552] applies. In particular, JS can call the API in any order and with any inputs, including malicious ones. This is particularly relevant when we consider the SDP which is passed to setLocalDescription(). While correct API usage requires that the application pass in SDP which was derived from createOffer() or createAnswer() (perhaps suitably modified as described in Section 6, there is no guarantee that applications do so. The browser MUST be prepared for the JS to pass in bogus data instead.

Conversely, the application programmer MUST recognize that the JS does not have complete control of browser behavior. One case that bears particular mention is that editing ICE candidates out of the SDP or suppressing trickled candidates does not have the expected behavior: implementations will still perform checks from those candidates even if they are not sent to the other side. Thus, for instance, it is not possible to prevent the remote peer from learning your public IP address by removing server reflexive candidates. Applications which wish to conceal their public IP address should instead configure the ICE agent to use only relay candidates.

9. IANA Considerations

This document requires no actions from IANA.

10. 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, Adam Bergkvist and Matthew Kaufman all provided valuable feedback on this proposal.

11. References

11.1. 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-mmusic-sdp-mux-attributes]
Nandakumar, S., "A Framework for SDP Attributes when Multiplexing", draft-ietf-mmusic-sdp-mux-attributes-01 (work in progress), February 2014.

[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-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-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.ietf-rtcweb-fec]
Uberti, J., "WebRTC Forward Error Correction Requirements", draft-ietf-rtcweb-fec-00 (work in progress), February 2015.

[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.ietf-rtcweb-security]
Rescorla, E., "Security Considerations for WebRTC", draft-ietf-rtcweb-security-06 (work in progress), January 2014.

[I-D.ietf-rtcweb-security-arch]
Rescorla, E., "WebRTC Security Architecture", draft-ietf-rtcweb-security-arch-09 (work in progress), February 2014.

[I-D.ietf-rtcweb-video]
Roach, A., "WebRTC Video Processing and Codec Requirements", draft-ietf-rtcweb-video-00 (work in progress), July 2014.

[I-D.nandakumar-mmusic-proto-iana-registration]
Nandakumar, S., "IANA registration of SDP ’proto’ attribute for transporting RTP Media over TCP under various RTP profiles.", September 2014.

[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.

[RFC3552] Rescorla, E. and B. Korver, "Guidelines for Writing RFC Text on Security Considerations", BCP 72, RFC 3552, July 2003.

[RFC3605] Huitema, C., "Real Time Control Protocol (RTCP) attribute in Session Description Protocol (SDP)", RFC 3605, October 2003.

[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.

[RFC6347] Rescorla, E. and N. Modadugu, "Datagram Transport Layer Security Version 1.2", RFC 6347, January 2012.

[RFC6904] Lennox, J., "Encryption of Header Extensions in the Secure Real-time Transport Protocol (SRTP)", RFC 6904, April 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.

11.2. Informative References

[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.

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

[W3C.WD-webrtc-20140617]
Bergkvist, A., Burnett, D., Narayanan, A., and C. Jennings, "WebRTC 1.0: Real-time Communication Between Browsers", World Wide Web Consortium WD WD-webrtc-20140617, June 2014, <http://www.w3.org/TR/2011/WD-webrtc-20140617>.

Appendix A. Change log

Note: This section will be removed by RFC Editor before publication.

Changes in draft-09:">

- o Don’t return null for {local,remote}Description after close().
- o Changed TCP/TLS to UDP/DTLS in RTP profile names.
- o Separate out bundle and mux policy.
- o Added specific references to FEC mechanisms.
- o Added canTrickle mechanism.
- o Added section on subsequent answers and, answer options.
- o Added text defining set{Local,Remote}Description behavior.

Changes in draft-08:

- o Added new example section and removed old examples in appendix.
- o Fixed <proto> field handling.
- o Added text describing a=rtcp attribute.
- o Reworked handling of OfferToReceiveAudio and OfferToReceiveVideo per discussion at IETF 90.
- o Reworked trickle ICE handling and its impact on m= and c= lines per discussion at interim.
- o Added max-bundle-and-rtcp-mux policy.
- o Added description of maxptime handling.
- o Updated ICE candidate pool default to 0.
- o Resolved open issues around AppID/receiver-ID.

- o Reworked and expanded how changes to the ICE configuration are handled.
- o Some reference updates.
- o Editorial clarification.

Changes in draft-07:

- o Expanded discussion of VAD and Opus DTX.
- o Added a security considerations section.
- o Rewrote the section on modifying SDP to require implementations to clearly indicate whether any given modification is allowed.
- o Clarified impact of IceRestart on CreateOffer in local-offer state.
- o Guidance on whether attributes should be defined at the media level or the session level.
- o Renamed "default" bundle policy to "balanced".
- o Removed default ICE candidate pool size and clarify how it works.
- o Defined a canonical order for assignment of MSTs to m= lines.
- o Removed discussion of rehydration.
- o Added Eric Rescorla as a draft editor.
- o Cleaned up references.
- o Editorial cleanup

Changes in draft-06:

- o Reworked handling of m= line recycling.
- o Added handling of BUNDLE and bundle-only.
- o Clarified handling of rollback.
- o Added text describing the ICE Candidate Pool and its behavior.
- o Allowed OfferToReceiveX to create multiple recvonly m= sections.

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

Eric Rescorla (editor)
Mozilla
331 Evelyn Ave
Mountain View, CA 94041
USA

Email: ekr@rtfm.com

Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: June 1, 2015

H. Alvestrand
Google

November 28, 2014

Overview: Real Time Protocols for Browser-based Applications
draft-ietf-rtcweb-overview-13

Abstract

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

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

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

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

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

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

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

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

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

Development of The Universal Solution has proved hard, however, for all the usual reasons.

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

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

This memo describes a set of building blocks that can be made accessible and controllable through a Javascript API in a browser, and which together form a sufficient set of functions to allow the use of interactive audio and video in applications that communicate

directly between browsers across the Internet. The resulting protocol suite is intended to enable all the applications that are described as required scenarios in the use cases document [I-D.ietf-rtcweb-use-cases-and-requirements].

Other efforts, for instance the W3C WEBRTC, Web Applications and Device API working groups, focus on making standardized APIs and interfaces available, within or alongside the HTML5 effort, for those functions; this memo concentrates on specifying the protocols and subprotocols that are needed to specify the interactions that happen across the network.

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

2. Principles and Terminology

2.1. Goals of this document

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

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

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

2.2. Relationship between API and protocol

The total WebRTC effort consists of two pieces:

- o A protocol specification, done in the IETF

- o A Javascript API specification, done in the W3C [W3C.WD-webrtc-20120209][W3C.WD-mediacapture-streams-20120628]

Together, these two specifications aim to provide an environment where Javascript embedded in any page, viewed in any compatible browser, when suitably authorized by its user, is able to set up communication using audio, video and auxiliary data, where the

browser environment does not constrain the types of application in which this functionality can be used.

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

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

For the purpose of this document, we define the following terminology to talk about WebRTC things:

- o A WebRTC browser (also called a WebRTC User Agent or WebRTC UA) is something that conforms to both the protocol specification and the Javascript API defined above.

- o A WebRTC non-browser is something that conforms to the protocol specification, but does not claim to implement the Javascript API. This can also be called a "WebRTC device" or "WebRTC native application".

- o A WebRTC endpoint is either a WebRTC browser or a WebRTC non-browser. It conforms to the protocol specification.

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

- o A WebRTC gateway is a WebRTC-compatible endpoint that mediates media traffic to non-WebRTC entities.

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

A WebRTC non-browser may be capable of hosting applications in a similar way to the way in which a browser can host Javascript

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

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

2.3. On interoperability and innovation

The "Mission statement of the IETF" [RFC3935] states that "The benefit of a standard to the Internet is in interoperability - that multiple products implementing a standard are able to work together in order to deliver valuable functions to the Internet’s users."

Communication on the Internet frequently occurs in two phases:

- Two parties communicate, through some mechanism, what functionality they both are able to support

- They use that shared communicative functionality to communicate, or, failing to find anything in common, give up on communication.

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

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

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

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

2.4. Terminology

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

The list is in alphabetical order.

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

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

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

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

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

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

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

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

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

SDP Agent: The protocol implementation involved in the SDP offer/ answer exchange, as defined in [RFC3264] section 3.

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

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

NOTE: Where common definitions exist for these terms, those definitions should be used to the greatest extent possible.

3. Architecture and Functionality groups

The model of real-time support for browser-based applications does not assume that the browser will contain all the functions that need to be performed in order to have a function such as a telephone or a video conferencing unit; the vision is that the browser will have the functions that are needed for a Web application, working in conjunction with its backend servers, to implement these functions.

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

```
                        +-------------------------+  On-the-wire
                        |                         |  Protocols
                        |      Servers            |--------->
                        |                         |
                        |                         |
                        +-------------------------+
                                    ^
                                    |
                                    |
                                    | HTTP/
                                    | Websockets
                                    |
                                    |
                     +-----------------------------+
                     |    Javascript/HTML/CSS      |
                     +-----------------------------+
                  Other  ^                  ^RTC
                  APIs   |                  |APIs
                     +---|------------------|------+
                     |   |                  |      |
                     |                  +---------+|
                     |                  | Browser ||  On-the-wire
                     | Browser          | RTC     ||  Protocols
                     |                  | Function|----------->
                     |                  |         ||
                     |                  |         ||
                     |                  +---------+|
                     +---------------------|------+
                                           |
                                           V
                                     Native OS Services
```

Figure 1: Browser Model

Note that HTTP and Websockets are also offered to the Javascript application through browser APIs.

As for all protocol and API specifications, there is no restriction that the protocols can only be used to talk to another browser; since they are fully specified, any endpoint that implements the protocols

faithfully should be able to interoperate with the application running in the browser.

A commonly imagined model of deployment is the one depicted below.

```
                +-----------+             +-----------+
                |   Web     |             |   Web     |
                |           |  Signaling  |           |
                |           |-------------|           |
                |  Server   |   path      |  Server   |
                |           |             |           |
                +-----------+             +-----------+
                     /                           \
                    /                             \ Application-defined
                   /                               \ over
                  /                                 \ HTTP/Websockets
                 /  Application-defined over         \
                /   HTTP/Websockets                   \
               /                                       \
        +-----------+                           +-----------+
        |JS/HTML/CSS|                           |JS/HTML/CSS|
        +-----------+                           +-----------+
        +-----------+                           +-----------+
        |           |                           |           |
        |           |                           |           |
        |  Browser  | -------------------------- |  Browser  |
        |           |          Media path       |           |
        |           |                           |           |
        +-----------+                           +-----------+
```

Figure 2: Browser RTC Trapezoid

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

If the two Web servers are operated by different entities, the inter-server signaling mechanism needs to be agreed upon, either by standardization or by other means of agreement. Existing protocols (for example SIP [RFC3261] or XMPP [RFC6120]) could be used between servers, while either a standards-based or proprietary protocol could be used between the browser and the web server.

For example, if both operators' servers implement SIP, SIP could be used for communication between servers, along with either a

standardized signaling mechanism (e.g. SIP over Websockets) or a proprietary signaling mechanism used between the application running in the browser and the web server. Similarly, if both operators' servers implement XMPP, XMPP could be used for communication between XMPP servers, with either a standardized signaling mechanism (e.g. XMPP over Websockets or BOSH) or a proprietary signaling mechanism used between the application running in the browser and the web server.

The choice of protocols, and definition of the translation between them, is outside the scope of the WebRTC protocol suite described in the document.

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

- Data transport: TCP, UDP and the means to securely set up connections between entities, as well as the functions for deciding when to send data: Congestion management, bandwidth estimation and so on.

- Data framing: RTP and other data formats that serve as containers, and their functions for data confidentiality and integrity.

- Data formats: Codec specifications, format specifications and functionality specifications for the data passed between systems. Audio and video codecs, as well as formats for data and document sharing, belong in this category. In order to make use of data formats, a way to describe them, a session description, is needed.

- Connection management: Setting up connections, agreeing on data formats, changing data formats during the duration of a call; SIP and Jingle/XMPP belong in this category.

- Presentation and control: What needs to happen in order to ensure that interactions behave in a non-surprising manner. This can include floor control, screen layout, voice activated image switching and other such functions - where part of the system require the cooperation between parties. XCON and Cisco/ Tandberg's TIP were some attempts at specifying this kind of functionality; many applications have been built without standardized interfaces to these functions.

- Local system support functions: These are things that need not be specified uniformly, because each participant may choose to do these in a way of the participant's choosing, without affecting the bits on the wire in a way that others have to be cognizant of. Examples in this category include echo cancellation (some forms of

it), local authentication and authorization mechanisms, OS access control and the ability to do local recording of conversations.

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

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

4. Data transport

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

It includes necessary functions for congestion control: When not to send data.

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

5. Data framing and securing

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

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

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

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

6. Data formats

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

WebRTC endpoints that support audio and/or video MUST implement the codecs and profiles required in [I-D.ietf-rtcweb-audio] and [I-D.ietf-rtcweb-video].

7. Connection management

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

The following principles apply:

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

2. It will be possible to gateway between legacy SIP devices that support ICE and appropriate RTP / SDP mechanisms, codecs and security mechanisms without using a media gateway. A signaling gateway to convert between the signaling on the web side to the SIP signaling may be needed.

3. When a new codec is specified, and the SDP for the new codec is specified in the MMUSIC WG, no other standardization should be required for it to be possible to use that in the web browsers. Adding new codecs which might have new SDP parameters should not

change the APIs between the browser and Javascript application. As soon as the browsers support the new codecs, old applications written before the codecs were specified should automatically be able to use the new codecs where appropriate with no changes to the JS applications.

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

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

WebRTC endpoints MUST implement the functions described in that document that relate to the network layer (for example Bundle, RTCP-mux and Trickle ICE), but do not need to support the API functionality described there.

8. Presentation and control

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

WebRTC browsers MUST implement these two specifications.

9. Local system support functions

These are characterized by the fact that the quality of these functions strongly influence the user experience, but the exact algorithm does not need coordination. In some cases (for instance echo cancellation, as described below), the overall system definition may need to specify that the overall system needs to have some characteristics for which these facilities are useful, without requiring them to be implemented a certain way.

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

Certain parts of the system SHOULD conform to certain properties, for instance:

- o Echo cancellation should be good enough to achieve the suppression of acoustical feedback loops below a perceptually noticeable level.

- o Privacy concerns MUST be satisfied; for instance, if remote control of camera is offered, the APIs should be available to let the local participant figure out who’s controlling the camera, and possibly decide to revoke the permission for camera usage.

- o Automatic gain control, if present, should normalize a speaking voice into a reasonable dB range.

The requirements on WebRTC systems with regard to audio processing are found in [I-D.ietf-rtcweb-audio]; the proposed API for control of local devices are found in [W3C.WD-mediacapture-streams-20120628].

WebRTC endpoints MUST implement the processing functions in [I-D.ietf-rtcweb-audio]. (Together with the requirement inSection 6, this means that WebRTC endpoints MUST implement the whole document.)

10. IANA Considerations

This document makes no request of IANA.

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

11. Security Considerations

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

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

- o Security of the communication channels: It should be easy for a participant to reassure himself of the security of his communication - by verifying the crypto parameters of the links he himself participates in, and to get reassurances from the other parties to the communication that they promise that appropriate measures are taken.

- o Security of the partners’ identity: verifying that the participants are who they say they are (when positive identification is appropriate), or that their identity cannot be uncovered (when anonymity is a goal of the application).

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

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

12. Acknowledgements

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

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

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

Thanks to Bjoern Hoehrmann, Colin Perkins, Colton Shields, Eric Rescorla, Heath Matlock, Henry Sinnreich, Justin Uberti, Keith Drage and Simon Leinen for document review.

13. References

13.1. Normative References

[I-D.ietf-rtcweb-audio]
Valin, J. and C. Bran, "WebRTC Audio Codec and Processing Requirements", draft-ietf-rtcweb-audio-05 (work in progress), February 2014.

[I-D.ietf-rtcweb-data-channel]
Jesup, R., Loreto, S., and M. Tuexen, "WebRTC Data Channels", draft-ietf-rtcweb-data-channel-11 (work in progress), July 2014.

[I-D.ietf-rtcweb-data-protocol]
Jesup, R., Loreto, S., and M. Tuexen, "WebRTC Data Channel Establishment Protocol", draft-ietf-rtcweb-data-protocol-07 (work in progress), July 2014.

[I-D.ietf-rtcweb-jsep]
Uberti, J., Jennings, C., and E. Rescorla, "Javascript Session Establishment Protocol", draft-ietf-rtcweb-jsep-07 (work in progress), July 2014.

[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-16 (work in progress), July 2014.

[I-D.ietf-rtcweb-security]
Rescorla, E., "Security Considerations for WebRTC", draft-ietf-rtcweb-security-07 (work in progress), July 2014.

[I-D.ietf-rtcweb-security-arch]
Rescorla, E., "WebRTC Security Architecture", draft-ietf-rtcweb-security-arch-10 (work in progress), July 2014.

[I-D.ietf-rtcweb-transports]
Alvestrand, H., "Transports for WebRTC", draft-ietf-rtcweb-transports-06 (work in progress), August 2014.

[I-D.ietf-rtcweb-video]
Roach, A., "WebRTC Video Processing and Codec Requirements", draft-ietf-rtcweb-video-00 (work in progress), July 2014.

[RFC3264] Rosenberg, J. and H. Schulzrinne, "An Offer/Answer Model with Session Description Protocol (SDP)", RFC 3264, June 2002.

[RFC3550] Schulzrinne, H., Casner, S., Frederick, R., and V. Jacobson, "RTP: A Transport Protocol for Real-Time Applications", STD 64, RFC 3550, 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.

[RFC5245] Rosenberg, J., "Interactive Connectivity Establishment (ICE): A Protocol for Network Address Translator (NAT) Traversal for Offer/Answer Protocols", RFC 5245, April 2010.

[W3C.WD-mediacapture-streams-20120628]
Burnett, D. and A. Narayanan, "Media Capture and Streams", World Wide Web Consortium WD WD-mediacapture-streams-20120628, June 2012, <http://www.w3.org/TR/2012/WD-mediacapture-streams-20120628>.

[W3C.WD-webrtc-20120209]
Bergkvist, A., Burnett, D., Jennings, C., and A. Narayanan, "WebRTC 1.0: Real-time Communication Between Browsers", World Wide Web Consortium WD WD-webrtc-20120209, February 2012, <http://www.w3.org/TR/2012/WD-webrtc-20120209>.

13.2. Informative References

[I-D.alvestrand-rtcweb-gateways]
Alvestrand, H., "WebRTC Gateways", draft-alvestrand-rtcweb-gateways-00 (work in progress), August 2014.

[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-14 (work in progress), February 2014.

[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.

[RFC3935] Alvestrand, H., "A Mission Statement for the IETF", BCP 95, RFC 3935, October 2004.

[RFC6120] Saint-Andre, P., "Extensible Messaging and Presence Protocol (XMPP): Core", RFC 6120, March 2011.

[W3C.WD-html5-20110525]
Hickson, I., "HTML5", World Wide Web Consortium LastCall WD-html5-20110525, May 2011, <http://www.w3.org/TR/2011/WD-html5-20110525>.

Appendix A. Change log

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

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

Added section "On interoperability and innovation"

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

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

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

Strengthened disclaimer that listed codecs are placeholders, not decisions.

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

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

Added section on "Relationship between API and protocol"

Added terminology section

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

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

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

Added content to acknowledgments section.

Added change log.

Spell-checked document.

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

Changed draft name and document date.

Removed unused references

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

Added architecture figures to section 2.

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

Added a few more definitions.

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

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

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

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

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

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

Added references to a number of new drafts.

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

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

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

Added a definition for "browser".

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

Made introduction more normative.

Several wording changes in response to review comments from EKR

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

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

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

A.10. Changes from -05 to -06

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

A.11. Changes from -06 to -07

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

Otherwise, it’s a "keepalive" draft.

A.12. Changes from -07 to -08

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

A.13. Changes from -08 to -09

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

A.14. Changes from -09 to -10

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

Updated reference to data-protocol draft

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

Deleted references to -unified-plan

Deleted reference to -generic-idp (draft expired)

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

Added pointer to the security section of the API drafts.

A.15. Changes from -10 to -11

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

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

A.16. Changes from -11 to -12

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

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

A.17. Changes from -12 to -13

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

Added words about WebRTC APIs in languages other than Javascript.

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

Author’s Address

Harald T. Alvestrand
Google
Kungsbron 2
Stockholm 11122
Sweden

Email: harald@alvestrand.no

RTCWEB Working Group
Internet-Draft
Intended status: Standards Track
Expires: August 13, 2015

C. S. Perkins
University of Glasgow
M. Westerlund
Ericsson
J. Ott
Aalto University

February 09, 2015

Web Real-Time Communication (WebRTC): Media Transport and Use of RTP
draft-ietf-rtcweb-rtp-usage-22

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

Copyright Notice

Table of Contents

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 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. 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 to review the available RTP features and extensions, and to define a common baseline RTP feature set for all WebRTC Endpoints. 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.

RTP and RTCP extensions that are not discussed in this document can be implemented by WebRTC Endpoints if they are beneficial for new use cases. However, they are not necessary to address the WebRTC use cases and requirements identified in [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 additional terms:

WebRTC MediaStream: The MediaStream concept defined by the W3C in the WebRTC API [W3C.WD-mediacapture-streams-20130903].

Transport-layer Flow: A uni-directional flow of transport packets that are identified by having a particular 5-tuple of source IP address, source port, destination IP address, destination port, and transport protocol used.

Bi-directional Transport-layer Flow: A bi-directional transport-layer flow is a transport-layer flow that is symmetric. That is, the transport-layer flow in the reverse direction has a 5-tuple where the source and destination address and ports are swapped compared to the forward path transport-layer flow, and the transport protocol is the same.

This document uses the terminology from [I-D.ietf-avtext-rtp-grouping-taxonomy] and [I-D.ietf-rtcweb-overview]. Other terms are used according to their definitions from the RTP Specification [RFC3550]. Especially note the following frequently used terms: RTP Packet Stream, RTP Session, and End-point.

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. Also described are the core extensions providing essential features that all WebRTC Endpoints 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 and used in all WebRTC Endpoints.

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

- 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]. The RTCP optimisations for multi-SSRC sessions defined in [I-D.ietf-avtcore-rtp-multi-stream-optimisation] MAY be supported; if supported the usage MUST be signalled.

- 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 Sending correct synchronisation information in the RTCP Sender Reports, to allow receivers to implement lip-synchronisation; see Section 5.2.1 regarding support for the rapid RTP synchronisation extensions.

- o Support for multiple synchronisation contexts. Participants that send multiple simultaneous RTP packet streams SHOULD do so as part of a single synchronisation context, using a single RTCP CNAME for all streams and allowing receivers to play the streams out in a synchronised manner. For compatibility with potential future versions of this specification, or for interoperability with non-WebRTC devices through a gateway, receivers MUST support multiple synchronisation contexts, indicated by the use of multiple RTCP CNAMEs in an RTP session. This specification requires the usage of a single CNAME when sending RTP Packet Streams in some circumstances, see Section 4.9.

- o Support for sending and receiving RTCP SR, RR, SDES, and BYE packet types, with OPTIONAL support for other RTCP packet types unless mandated by other parts of this specification. Note that additional RTCP Packet types are used by the RTP/SAVPF Profile (Section 4.2) and the other RTCP extensions (Section 5). WebRTC endpoints that implement the SDP bundle negotiation extension will use the SDP grouping framework ’mid’ attribute to identify media streams. Such endpoints MUST implement the RTCP SDES MID item described in [I-D.ietf-mmusic-sdp-bundle-negotiation].

- 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 (Section 6.3.6 of [RFC3550]) and reverse reconsideration (Section 6.3.4 of [RFC3550]).

- 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 [RFC4566][RFC3556].

- o Support for the reduced minimum RTCP reporting interval described in Section 6.2 of [RFC3550] is REQUIRED. When using the reduced minimum RTCP reporting interval, the fixed (non-reduced) minimum interval MUST be used when calculating the participant timeout interval (see Sections 6.2 and 6.3.5 of [RFC3550]). The delay before sending the initial compound RTCP packet can be set to zero (see Section 6.2 of [RFC3550] as updated by [I-D.ietf-avtcore-rtp-multi-stream]).

- o Support for discontinuous transmission. RTP allows endpoints to pause and resume transmission at any time. When resuming, the RTP sequence number will increase by one, as usual, while the increase in the RTP timestamp value will depend on the duration of the pause. Discontinuous transmission is most commonly used with some audio payload formats, but is not audio specific, and can be used with any RTP payload format.

- o Ignore unknown RTCP packet types and RTP header extensions. This to ensure robust handling of future extensions, middlebox behaviours, etc., that can result in not signalled RTCP packet types or RTP header extensions being received. If a compound RTCP packet is received that contains a mixture of known and unknown RTCP packet types, the known packets types need to be processed as usual, with only the unknown packet types being discarded.

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. The RTP/SAVPF profile is the combination of 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. This allows more flexible transmission of RTCP packets in response to events, rather than strictly according to bandwidth, and is vital for being able to report congestion signals as well as media events. These extensions also allow saving RTCP bandwidth, and an end-point will commonly only use the full RTCP bandwidth allocation if there are many events that require feedback. The timer rules 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 RTP/AVP or RTP/SAVP profile, given some constraints on parameter configuration such as the RTCP bandwidth value and "trr-int" (the most important factor for interworking with RTP/(S)AVP end-points via a gateway is to set the trr-int parameter to a value representing 4 seconds, see Section 6.1 in [I-D.ietf-avtcore-rtp-multi-stream]).

The secure RTP (SRTP) profile extensions [RFC3711] are needed to provide media encryption, integrity protection, replay protection and a limited form of source authentication. WebRTC Endpoints 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 (i.e., implementations MUST use SRTP and SRTCP). The RTP/SAVPF profile MUST be configured using the cipher suites, DTLS-SRTP protection profiles, keying mechanisms, and other parameters described in [I-D.ietf-rtcweb-security-arch].

4.3. Choice of RTP Payload Formats

Mandatory to implement audio codecs and RTP payload formats for WebRTC endpoints are defined in [I-D.ietf-rtcweb-audio]. Mandatory to implement video codecs and RTP payload formats for WebRTC endpoints are defined in [I-D.ietf-rtcweb-video]. WebRTC endpoints MAY additionally implement any other codec for which an RTP payload format and associated signalling has been defined.

WebRTC Endpoints 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, along with any other SDP attributes needed to configure the RTP payload format.

End-points 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 packet stream with a signalling context. This association is possible provided unique RTP payload type numbers are used in each context. For example, an RTP packet stream can be associated with an SDP "m=" line by comparing the RTP payload type numbers used by the RTP packet stream with payload types signalled in the "a=rtpmap:" lines in the media sections of the SDP. This leads to the following considerations:

If RTP packet 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 associate RTP packet 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.

A single RTP payload type number MUST NOT be assigned to different RTP payload formats, or different configurations of the same RTP payload format, within a single RTP session (note that the "m=" lines in an SDP bundle group [I-D.ietf-mmusic-sdp-bundle-negotiation] form a single RTP session).

An end-point that has signalled support for multiple RTP payload formats MUST be able to 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 used by a single SSRC that were signalled during session set-up.

If performing changes between two RTP payload types that use different RTP clock rates, an RTP sender MUST follow the recommendations in Section 4.1 of [RFC7160]. RTP receivers MUST follow the recommendations in Section 4.3 of [RFC7160] 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 end-points communicating using RTP is known as an RTP session [RFC3550]. An end-point 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 a different transport-layer flow for the video). WebRTC Endpoints are REQUIRED to implement support for multimedia sessions in this way, separating each RTP session using different transport-layer flows for compatibility with legacy systems (this is sometimes called session multiplexing).

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 packet 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 therefore also REQUIRED to support transport of all RTP packet streams, independent of media type, in a single RTP session using a single transport layer flow, according to [I-D.ietf-avtcore-multi-media-rtp-session] (this is sometimes called SSRC multiplexing). If multiple types of media are to be used in a single RTP session, all participants in that RTP session MUST agree to this usage. In an SDP context, [I-D.ietf-mmusic-sdp-bundle-negotiation] can be used to signal such a bundle of RTP packet streams forming a single RTP session.

Further discussion about the suitability of different RTP session structures and multiplexing methods to different scenarios 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 flows (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 (NAT/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, implementations are REQUIRED to support multiplexing RTP data packets and RTCP control packets on a single transport-layer flow [RFC5761]. Such RTP and RTCP multiplexing MUST be negotiated in the signalling

channel before it is used. If SDP is used for signalling, this negotiation MUST use the attributes defined in [RFC5761]. For backwards compatibility, implementations are also REQUIRED to support RTP and RTCP sent on separate transport-layer flows.

Note that the use of RTP and RTCP multiplexed onto a single transport-layer flow 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 [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. Implementations MUST support sending and receiving non-compound RTCP feedback packets [RFC5506]. Use of non-compound RTCP packets MUST be negotiated using the signalling channel. If SDP is used for signalling, this negotiation MUST use the attributes defined in [RFC5506]. For backwards compatibility, implementations are also REQUIRED to support the use of compound RTCP feedback packets if the remote end-point 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 reason for using symmetric RTP is primarily to avoid issues with NATs and Firewalls by ensuring that the send and receive RTP packet streams, as well as RTCP, are actually bi-directional transport-layer flows. This will keep alive the NAT and firewall pinholes, and help indicate consent that the receive direction is a transport-layer 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. The amount of per flow QoS state kept in the network is also reduced.

4.8. Choice of RTP Synchronisation Source (SSRC)

Implementations are REQUIRED to support signalled RTP synchronisation source (SSRC) identifiers. If SDP is used, this MUST be done using the "a=ssrc:" SDP attribute defined in Section 4.1 and Section 5 of [RFC5576] and the "previous-ssrc" source attribute defined in Section 6.2 of [RFC5576]; other per-SSRC attributes defined in [RFC5576] MAY be supported.

While support for signalled SSRC identifiers is mandated, their use 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 packet stream with a non-RTP context. For users of the WebRTC API a mapping between SSRCs and MediaStreamTracks are provided per Section 11. For gateways or other usages it is possible to associate an RTP packet 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 packet stream are often sufficient to associate that packet 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 packet 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 end-point. While the Synchronisation Source (SSRC) identifier for an RTP end-point can change if a collision is detected, or when the RTP application is restarted, its RTCP CNAME is meant to stay unchanged for the duration of a RTCPeerConnection [W3C.WD-webrtc-20130910], so that RTP end-points can be uniquely identified and associated with their RTP packet streams within a set of related RTP sessions.

Each RTP end-point MUST have at least one RTCP CNAME, and that RTCP CNAME MUST be unique within the RTCPeerConnection. RTCP CNAMEs identify a particular synchronisation context, i.e., all SSRCs associated with a single RTCP CNAME share a common reference clock. If an end-point has SSRCs that are associated with several

unsynchronised reference clocks, and hence different synchronisation contexts, it will need to use multiple RTCP CNAMEs, one for each synchronisation context.

Taking the discussion in Section 11 into account, a WebRTC Endpoint MUST NOT use more than one RTCP CNAME in the RTP sessions belonging to single RTCPeerConnection (that is, an RTCPeerConnection forms a synchronisation context). RTP middleboxes MAY generate RTP packet streams associated with more than one RTCP CNAME, to allow them to avoid having to resynchronize media from multiple different end-points part of a multi-party RTP session.

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 (Section 13). Accordingly, a WebRTC Endpoint MUST generate a new, unique, short-term persistent RTCP CNAME for each RTCPeerConnection, following [RFC7022], with a single exception; if explicitly requested at creation an RTCPeerConnection MAY use the same CNAME as as an existing RTCPeerConnection within their common same-origin context.

An WebRTC Endpoint 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.

4.10. Handling of Leap Seconds

The guidelines regarding handling of leap seconds to limit their impact on RTP media play-out and synchronization given in [RFC7164] SHOULD be followed.

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 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 WebRTC.

5.1. Conferencing Extensions and Topologies

RTP is a protocol that inherently supports group communication. Groups can be implemented by having each endpoint send its RTP packet streams to an RTP middlebox that redistributes the traffic, by using a mesh of unicast RTP packet streams between endpoints, or by using

an IP multicast group to distribute the RTP packet streams. These topologies can be implemented in a number of ways as discussed in [I-D.ietf-avtcore-rtp-topologies-update].

While the use of IP multicast groups is popular in IPTV systems, the topologies based on RTP middleboxes are dominant in interactive video conferencing environments. Topologies based on a mesh of unicast transport-layer flows to create a common RTP session have not seen widespread deployment to date. Accordingly, WebRTC Endpoints are not expected to support topologies based on IP multicast groups or to support mesh-based topologies, such as a point-to-multipoint mesh configured as a single RTP session (Topo-Mesh in the terminology of [I-D.ietf-avtcore-rtp-topologies-update]). However, a point-to-multipoint mesh constructed using several RTP sessions, implemented in WebRTC using independent RTCPeerConnections [W3C.WD-webrtc-20130910], can be expected to be used in WebRTC, and needs to be supported.

WebRTC Endpoints implemented according to this memo are expected to support all the topologies described in [I-D.ietf-avtcore-rtp-topologies-update] where the RTP endpoints send and receive unicast RTP packet streams to and from some peer device, provided that peer can participate in performing congestion control on the RTP packet streams. The peer device could be another RTP endpoint, or it could be an RTP middlebox that redistributes the RTP packet streams to other RTP endpoints. This limitation means that some of the RTP middlebox-based topologies are not suitable for use in WebRTC. Specifically:

- o Video switching MCUs (Topo-Video-switch-MCU) SHOULD NOT be used, since they make the use of RTCP for congestion control and quality of service reports problematic (see Section 3.8 of [I-D.ietf-avtcore-rtp-topologies-update]).

- o The Relay-Transport Translator (Topo-PtM-Trn-Translator) topology SHOULD NOT be used because its safe use requires a congestion control algorithm or RTP circuit breaker that handles point to multipoint, which has not yet been standardised.

The following topology can be used, however it has some issues worth noting:

- o Content modifying MCUs with RTCP termination (Topo-RTCP-terminating-MCU) MAY be used. Note that in this RTP Topology, RTP loop detection and identification of active senders is the responsibility of the WebRTC application; since the clients are isolated from each other at the RTP layer, RTP cannot assist with these functions (see section 3.9 of [I-D.ietf-avtcore-rtp-topologies-update]).

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 packet streams and distributes them to the other participants. These extensions are not necessary for interoperability; an RTP end-point 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 of these extensions are mandatory to be supported by WebRTC Endpoints.

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

5.1.1. Full Intra Request (FIR)

The Full Intra Request message is defined in Sections 3.5.1 and 4.3.1 of the Codec Control Messages [RFC5104]. It 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 Endpoints that are sending media MUST understand and react to FIR feedback messages they receive, since this greatly improves the user experience when using centralised mixer-based conferencing. Support for sending FIR messages is OPTIONAL.

5.1.2. Picture Loss Indication (PLI)

The Picture Loss Indication message 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 Endpoints that are sending media MUST understand and react to PLI feedback messages as a loss tolerance mechanism. Receivers MAY send PLI messages.

5.1.3. Slice Loss Indication (SLI)

The Slice Loss Indication message 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. It is RECOMMENDED that receivers generate SLI feedback messages if slices are lost when using a codec that supports the concept of macro blocks. A sender that receives an SLI feedback message SHOULD attempt to repair the lost slice(s).

5.1.4. Reference Picture Selection Indication (RPSI)

Reference Picture Selection Indication (RPSI) messages are defined in Section 6.3.3 of the RTP/AVPF profile [RFC4585]. Some video encoding standards allow the use of older reference pictures than the most recent one for predictive coding. If such a codec is in use, and if the encoder has learnt that encoder-decoder synchronisation has been lost, then a known as correct reference picture can be used as a base for future coding. The RPSI message allows this to be signalled. Receivers that detect that encoder-decoder synchronisation has been lost SHOULD generate an RPSI feedback message if codec being used supports reference picture selection. A RTP packet stream sender that receives such an RPSI message SHOULD act on that messages to change the reference picture, if it is possible to do so within the available bandwidth constraints, and with the codec being used.

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)

The TMMBR feedback message is defined in Sections 3.5.4 and 4.2.1 of the Codec Control Messages [RFC5104]. This request 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 Endpoints that are sending media 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 WebRTC, 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 of RTP header extensions might include metadata that is additional to the usual RTP information, but that can safely be ignored without compromising interoperability.

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.

This header extension uses the [RFC5285] generic header extension framework, and so needs to be negotiated before it can be used.

5.2.2. Client-to-Mixer Audio Level

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

The Client-to-Mixer Audio Level [RFC6464] header extension MUST be implemented. It is REQUIRED that implementations are capable of encrypting the header extension according to [RFC6904] since the information contained in these header extensions can be considered sensitive. The use of this encryption is RECOMMENDED, however usage of the encryption can be explicitly disabled through API or signalling.

This header extension uses the [RFC5285] generic header extension framework, and so needs to be negotiated before it can be used.

5.2.3. Mixer-to-Client Audio Level

The Mixer to Client Audio Level header extension [RFC6465] provides an endpoint with the audio level of the different sources mixed into a common source stream 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 optimisation of non critical functions, and is hence OPTIONAL to implement. If this header extension is implemented, it is REQUIRED that implementations are capable of encrypting the header extension according to [RFC6904] since the information contained in these header extensions can be considered sensitive. It is further RECOMMENDED that this encryption is used, unless the encryption has been explicitly disabled through API or signalling.

This header extension uses the [RFC5285] generic header extension framework, and so needs to be negotiated before it can be used.

5.2.4. Media Stream Identification

WebRTC endpoints that implement the SDP bundle negotiation extension will use the SDP grouping framework ’mid’ attribute to identify media streams. Such endpoints MUST implement the RTP MID header extension described in [I-D.ietf-mmusic-sdp-bundle-negotiation].

This header extension uses the [RFC5285] generic header extension framework, and so needs to be negotiated before it can be used.

5.2.5. Coordination of Video Orientation

WebRTC endpoints that send or receive video MUST implement the coordination of video orientation (CVO) RTP header extension as described in Section 4 of [I-D.ietf-rtcweb-video].

This header extension uses the [RFC5285] generic header extension framework, and so needs to be negotiated before it can be used.

6. WebRTC Use of RTP: Improving Transport Robustness

There are tools that can make RTP packet streams robust against packet loss and reduce the impact of loss on media quality. However, they generally add some overhead compared to a non-robust stream. The overhead 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 send 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.

RTP packet stream senders are REQUIRED to understand the Generic NACK message defined in Section 6.2.1 of [RFC4585], but MAY choose to ignore some or all of this feedback (following Section 4.2 of [RFC4585]). Receivers MAY send NACKs for missing RTP packets. Guidelines on when to send NACKs are provided in [RFC4585]. 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 caused increased packet loss if the original packet loss was caused by network congestion. 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] sent using SSRC multiplexing, and MAY also support RTP retransmission packets sent using session multiplexing. Senders MAY send RTP retransmission packets in response to NACKs if support for the RTP retransmission payload format has been negotiated, and if the sender believes it is useful to send a retransmission of the packet(s) referenced in the NACK. Senders do 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 FEC schemes and their parameters.

WebRTC endpoints MUST follow the recommendations for FEC use given in [I-D.ietf-rtcweb-fec]. WebRTC endpoints MAY support other types of FEC, but these MUST be negotiated before they are used.

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 packet streams to each participant, and there can be several participants. Each of these RTP packet streams can contain different types of media, and the type of media, bit rate, and number of RTP packet streams as well as transport-layer flows can be highly asymmetric. Non-RTP traffic can share the network paths with RTP transport-layer 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 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's flows. Some requirements for congestion control algorithms for RTCPeerConnections are discussed in [I-D.ietf-rmcat-cc-requirements]. 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

WebRTC Endpoints MUST implement the RTP circuit breaker algorithm that is described in [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 packetisation choices that exist. In addition, the signalling

channel can establish maximum media bit-rate boundaries using, for example, 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). Signalled bandwidth limitations, such as SDP "b=AS:" or "b=CT:" lines received from the peer, MUST be followed when sending RTP packet streams. A WebRTC Endpoint receiving media SHOULD signal its bandwidth limitations, these limitations have to be based on known bandwidth limitations, for example the capacity of the edge links.

7.2. Congestion Control Interoperability and Legacy Systems

There are legacy RTP implementations that do not implement RTCP, and hence do not provide any congestion feedback. Congestion control cannot be performed with these end-points. WebRTC Endpoints 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 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 packet 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][RFC6792]. At the time of this writing, it is not clear what extended metrics are suitable for use in WebRTC, so there is no requirement that implementations generate 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.

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. In this case, future updates to this memo MUST be made following the Guidelines for Writers of RTP Payload Format Specifications [RFC2736], How to Write an RTP Payload Format [I-D.ietf-payload-rtp-howto] 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. Interworking functions might transform this into the RTP/SAVP profile for a legacy use case, by indicating to the WebRTC Endpoint that the RTP/SAVPF is used and configuring 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 [RFC5245] 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, i.e. transport-layer flow, is used for RTP and RTCP flows, this MUST be signalled (see Section 4.5).

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 use of any additional RTP header extensions and RTCP packet types, including any necessary parameters, MUST be signalled. This signalling is to ensure that a WebRTC Endpoint’s behaviour, especially when sending, of any extensions is predictable and consistent. For robustness, and for compatibility with non-WebRTC systems that might be connected to a WebRTC session via a gateway, implementations are REQUIRED to ignore unknown RTCP packets and RTP header extensions (see also Section 4.1).

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] if using SDP, or something semantically equivalent. This also ensures that the end-points have a common view of the RTCP bandwidth. A common RTCP bandwidth is important as a 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. Note that in WebRTC it will depend on the signalling model and API how these parameters need to be configured but they will be need to either be set in the API or explicitly signalled between the peers.

11. WebRTC API Considerations

The WebRTC API [W3C.WD-webrtc-20130910] and the Media Capture and Streams API [W3C.WD-mediacapture-streams-20130903] defines and uses the concept of a MediaStream that consists of zero or more MediaStreamTracks. A MediaStreamTrack 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 MediaStreamTracks within a MediaStream need to be possible to play out synchronised.

A MediaStreamTrack’s realisation in RTP in the context of an RTCPeerConnection consists of a source packet stream identified with an SSRC within an RTP session part of the RTCPeerConnection. The MediaStreamTrack can also result in additional packet streams, and thus SSRCs, in the same RTP session. These can be dependent packet streams from scalable encoding of the source stream associated with the MediaStreamTrack, if such a media encoder is used. They can also be redundancy packet streams, these are created when applying Forward Error Correction (Section 6.2) or RTP retransmission (Section 6.1) to the source packet stream.

It is important to note that the same media source can be feeding multiple MediaStreamTracks. As different sets of constraints or other parameters can be applied to the MediaStreamTrack, each MediaStreamTrack instance added to a RTCPeerConnection SHALL result in an independent source packet stream, with its own set of associated packet streams, and thus different SSRC(s). It will depend on applied constraints and parameters if the source stream and the encoding configuration will be identical between different MediaStreamTracks sharing the same media source. If the encoding parameters and constraints are the same, an implementation could choose to use only one encoded stream to create the different RTP packet streams. Note that such optimisations would need to take into account that the constraints for one of the MediaStreamTracks can at any moment change, meaning that the encoding configurations might no longer be identical and two different encoder instances would then be needed.

The same MediaStreamTrack can also be included in multiple MediaStreams, thus multiple sets of MediaStreams can implicitly need to use the same synchronisation base. To ensure that this works in all cases, and does not force an end-point to to disrupt the media by changing synchronisation base and CNAME during delivery of any ongoing packet streams, all MediaStreamTracks and their associated SSRCs originating from the same end-point need to be sent using the same CNAME within one RTCPeerConnection. This is motivating the discussion in Section 4.9 to only use a single CNAME.

> The requirement on using the same CNAME for all SSRCs that originate from the same end-point, does not require a middlebox that forwards traffic from multiple end-points to only use a single CNAME.

Different CNAMEs normally need to be used for different RTCPeerConnection instances, as specified in Section 4.9. Having two communication sessions with the same CNAME could enable tracking of a user or device across different services (see Section 4.4.1 of [I-D.ietf-rtcweb-security] for details). A web application can request that the CNAMEs used in different RTCPeerConnections (within a same-orign context) be the same, this allows for synchronization of the endpoint’s RTP packet streams across the different RTCPeerConnections.

> Note: this doesn’t result in a tracking issue, since the creation of matching CNAMEs depends on existing tracking.

The above will currently force a WebRTC Endpoint that receives a MediaStreamTrack on one RTCPeerConnection and adds it as an outgoing on any RTCPeerConnection to perform resynchronisation of the stream. This, as the sending party needs to change the CNAME to the one it uses, which implies that the sender has to use a local system clock as timebase for the synchronisation. Thus, the relative relation between the timebase of the incoming stream and the system sending out needs to defined. This relation also needs monitoring for clock drift and likely adjustments of the synchronisation. The sending entity is also responsible for congestion control for its sent streams. In cases of packet loss the loss of incoming data also needs to be handled. This leads to the observation that the method that is least likely to cause issues or interruptions in the outgoing source packet stream is a model of full decoding, including repair etc., followed by encoding of the media again into the outgoing packet stream. Optimisations of this method is clearly possible and implementation specific.

A WebRTC Endpoint MUST support receiving multiple MediaStreamTracks, where each of different MediaStreamTracks (and their sets of

associated packet streams) uses different CNAMEs. However, MediaStreamTracks that are received with different CNAMEs have no defined synchronisation.

> Note: The motivation for supporting reception of multiple CNAMEs is to allow for forward compatibility with any future changes that enables more efficient stream handling when end-points relay/ forward streams. It also ensures that end-points can interoperate with certain types of multi-stream middleboxes or end-points that are not WebRTC.

The binding between the WebRTC MediaStreams, MediaStreamTracks and the SSRC is done as specified in "Cross Session Stream Identification in the Session Description Protocol" [I-D.ietf-mmusic-msid]. This document [I-D.ietf-mmusic-msid] also defines, in section 4.1, how to map unknown source packet stream SSRCs to MediaStreamTracks and MediaStreams. This later is relevant to handle some cases of legacy interop. Commonly the RTP Payload Type of any incoming packets will reveal if the packet stream is a source stream or a redundancy or dependent packet stream. The association to the correct source packet stream depends on the payload format in use for the packet stream.

Finally this specification puts a requirement on the WebRTC API to realize a method for determining the CSRC list (Section 4.1) as well as the Mixer-to-Client audio levels (Section 5.2.3) (when supported) and the basic requirements for this is further discussed in Section 12.2.1.

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 Endpoint 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 Endpoint will be a simultaneous participant in one or more RTP sessions. Each RTP session can convey multiple media sources, and can include media data from multiple end-points. In the following, some ways in which WebRTC Endpoints can configure and use RTP sessions is outlined.

12.1.1. Use of Multiple Media Sources Within an RTP Session

RTP is a group communication protocol, and every RTP session can potentially contain multiple RTP packet streams. 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 sources (e.g., one RTP session for audio sources and one for video sources, each sent over different transport layer flows). 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 bi-directional transport-layer flow, but will contain multiple RTP packet 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 packet streams, one video and one audio, into a single RTP session.

Multiple Capture Devices: A WebRTC Endpoint might have multiple cameras, microphones, or other media capture devices, and so might want to generate several RTP packet 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 packet streams of the same media type into a single RTP session at the same time.

Associated Repair Data: An end-point might send a RTP packet stream that is somehow associated with another stream. For example, it might send an RTP packet 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 source packet stream, while others send it as a separate packet 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 RTP packet streams, 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 Endpoint, 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 Endpoint will need to render. Thus,

even though a WebRTC Endpoint 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 packet streams at once, even in a single RTP session.

12.1.2. Use of Multiple RTP Sessions

In addition to sending and receiving multiple RTP packet streams within a single RTP session, a WebRTC Endpoint might participate in multiple RTP sessions. There are several reasons why a WebRTC Endpoint 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 separate transport layer flow, for video. All WebRTC Endpoints 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 transport layer flows. If it is desired to use these mechanisms to provide differentiated quality of service for some RTP packet streams, then those RTP packet streams need to be sent in a separate RTP session using a different transport-layer flow, 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 RTP packet 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 RTP middlebox. In the WebRTC context this is currently possible by establishing multiple WebRTC MediaStreamTracks that have the same media source in one (or more) RTCPeerConnection. Each MediaStreamTrack 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 RTCPeerConnection. The RTP middlebox can

distinguish packets corresponding to the low- and high-resolution streams by inspecting their SSRC, RTP payload type, or some other information contained in RTP payload, RTP header extension or RTCP packets, but it can be easier to distinguish the RTP packet streams if they arrive on separate RTP sessions on separate transport-layer flows.

To directly connect with multiple peers: A multi-party conference does not need to use an RTP middlebox. 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 RTCPeerConnection object) to every other participant, as shown in Figure 1. This topology has the benefit of not requiring an RTP middlebox 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 packet streams for each participant that are part of the same session beyond the sender itself.

```
+---+     +---+
| A |<--->| B |
+---+     +---+
  ^         ^
   \       /
    \     /
     v   v
     +---+
     | C |
     +---+
```

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 RTCPeerConnection objects it is 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 it 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 be presented to 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 packet streams from certain perspectives, either by only sending some of the received RTP packet stream to any given receiver, or by providing a combined RTP packet stream out of a set of contributing streams.

```
+---+      +-------------+      +---+
| A |<---->|             |<---->| B |
+---+      | RTP mixer,  |      +---+
           | translator, |
           | or other    |
+---+      | middlebox   |      +---+
| C |<---->|             |<---->| D |
+---+      +-------------+      +---+
```

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 RTP middleboxes can use a number of tools to control the media encoding provided by a WebRTC Endpoint. This includes functions like requesting the breaking of 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 has the responsibility to correctly perform congestion control, source identification, manage synchronisation while providing the application with suitable media optimisations. The middlebox 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 re-encrypt the RTP packet stream 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 packet streams across themselves. This is due to the difference in the RTP packet 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 is sometimes desirable for an end-point that receives an RTP packet stream to be able to forward that RTP packet stream to a third party. The are some obvious security and privacy implications in supporting this, but also potential uses. This is supported in the W3C API by taking the received and decoded media and using it as media source that is re-encoding and transmitted as a new stream.

At the RTP layer, media forwarding acts as a back-to-back RTP receiver and RTP sender. The receiving side terminates the RTP session and decodes the media, while the sender side re-encodes and transmits the media using an entirely separate RTP session. The original sender will only see a single receiver of the media,

and will not be able to tell that forwarding is happening based on RTP-layer information since the RTP session that is used to send the forwarded media is not connected to the RTP session on which the media was received by the node doing the forwarding.

The end-point that is performing the forwarding is responsible for producing an RTP packet stream suitable for onwards transmission. The outgoing RTP session that is used to send the forwarded media is entirely separate to the RTP session on which the media was received. This will require media transcoding for congestion control purpose to produce a suitable bit-rate for the outgoing RTP session, reducing media quality 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 optimise its media transcoding operation. The cost is greatly increased computational complexity on the forwarding node. Receivers of the forwarded stream will see the forwarding device as the sender of the stream, and will not be able to tell from the RTP layer that they are receiving a forwarded stream rather than an entirely new RTP packet stream generated by the forwarding device.

12.1.3. Differentiated Treatment of RTP Packet Streams

There are use cases for differentiated treatment of RTP packet 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 packet 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 [W3C.WD-webrtc-20130910] 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 packet streams. Any changes in relative priority will also need to be considered for RTP packet streams that are associated with the main RTP packet streams, such as redundant streams for RTP retransmission and FEC. The importance of such redundant RTP packet streams 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 to be to use the same priority for redundant RTP packet stream as for the source RTP packet stream.

Secondly, the network can prioritize transport-layer flows and sub-flows, including RTP packet 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 consisting of 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 transport-layer 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 RTP packet streams used in a WebRTC session. When flow-based differentiation is available, the WebRTC Endpoint needs to know about it so that it can provide the separation of the RTP packet 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 Endpoint has to know which DSCP to use and that it can use them on some set of RTP packet streams. 2) The information needs to be propagated to the operating system when transmitting the packet. Details of this process are outside the scope of this memo and are further discussed in "DSCP and other packet markings for RTCWeb QoS" [I-D.ietf-tsvwg-rtcweb-qos].

For packet based marking schemes it might be possible to mark individual RTP packets differently based on the relative priority of the RTP payload. For example video codecs that have I, P, and B pictures could prioritise any payloads carrying only B frames less, as these are less damaging to loose. However, depending on the QoS mechanism and what markings that are applied, this can result in not only different packet drop probabilities but also packet reordering, see [I-D.ietf-tsvwg-rtcweb-qos] for further discussion. As a default policy all RTP packets related to a RTP packet stream ought to be provided with the same prioritization; per-packet prioritization is

outside the scope of this memo, but might be specified elsewhere in future.

It is also important to consider how RTCP packets associated with a particular RTP packet stream need to be marked. RTCP compound packets with Sender Reports (SR), ought to be marked with the same priority as the RTP packet stream itself, so the RTCP-based round-trip time (RTT) measurements are done using the same transport-layer flow priority as the RTP packet stream experiences. RTCP compound packets containing RR packet ought to be sent with the priority used by the majority of the RTP packet streams 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. Media Source, RTP Packet Streams, and Participant Identification

12.2.1. Media Source Identification

Each RTP packet stream is identified by a unique synchronisation source (SSRC) identifier. The SSRC identifier is carried in each of the RTP packets comprising a RTP packet 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 on a single transport layer flow at a WebRTC Endpoint is to separate the RTP packet 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 encoded streams from multiple media sources to form a new encoded stream from a new media source (the mixer). The RTP packets in that new RTP packet stream can include a Contributing Source (CSRC) list, indicating which original SSRCs contributed to the combined source stream. 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 media 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 WebRTC 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. It is desirable to expose this information to the WebRTC application using some API, after mapping the CSRC values to WebRTC MediaStream identities, so it can be exposed in the user interface.

12.2.2. SSRC Collision Detection

The RTP standard requires RTP implementations to have support for detecting and handling SSRC collisions, i.e., resolve the conflict when two different end-points use the same SSRC value (see section 8.2 of [RFC3550]). This requirement also applies to WebRTC Endpoints. 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. If SDP is used, this information is provided by Source-Specific SDP Attributes [RFC5576]. Still, collisions can occur if both end-points start using a new SSRC identifier prior to having signalled it to the peer and received acknowledgement on the signalling message. The Source-Specific SDP Attributes [RFC5576] contains a mechanism to signal how the end-point resolved the SSRC collision.

- SSRC values that have not been signalled could also appear in an RTP session. This is more likely than it appears, since some RTP functions use extra SSRCs to provide their functionality. For example, retransmission data might be transmitted using a separate RTP packet stream that requires its own SSRC, separate to the SSRC of the source RTP packet stream [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 RTCPeerConnection level.

- Multiple end-points in a multiparty conference can create new sources and signal those towards the RTP middlebox. In cases where the SSRC/CSRC are propagated between the different end-points from the RTP middlebox collisions can occur.

- o An RTP middlebox could connect an end-point's RTCPeerConnection to another RTCPeerConnection from the same end-point, 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 RTCPeerConnections by a RTP middlebox. To resolve the more generic case where multiple RTCPeerConnections are interconnected, identification of the media source(s) part of a MediaStreamTrack being propagated across multiple interconnected RTCPeerConnection 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 packet 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 packet 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.

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 also apply to this memo.

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. It is not believed 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 combing 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 RTP packet streams 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.

Some potential denial of service attacks exist if the RTCP reporting interval is configured to an inappropriate value. This could be done by configuring the RTCP bandwidth fraction to an excessively large or small value using the SDP "b=RR:" or "b=RS:" lines [RFC3556], or some similar mechanism, or by choosing an excessively large or small value for the RTP/AVPF minimal receiver report interval (if using SDP, this is the "a=rtcp-fb:... trr-int" parameter) [RFC4585]. The risks are as follows:

1. the RTCP bandwidth could be configured to make the regular reporting interval so large that effective congestion control cannot be maintained, potentially leading to denial of service due to congestion caused by the media traffic;

2. the RTCP interval could be configured to a very small value, causing endpoints to generate high rate RTCP traffic, potentially leading to denial of service due to the non-congestion controlled RTCP traffic; and

3. RTCP parameters could be configured differently for each endpoint, with some of the endpoints using a large reporting interval and some using a smaller interval, leading to denial of service due to premature participant timeouts due to mismatched timeout periods which are based on the reporting interval (this is a particular concern if endpoints use a small but non-zero value for the RTP/AVPF minimal receiver report interval (trr-int) [RFC4585], as discussed in Section 6.1 of [I-D.ietf-avtcore-rtp-multi-stream]).

Premature participant timeout can be avoided by using the fixed (non-reduced) minimum interval when calculating the participant timeout (see Section 4.1 of this memo and Section 6.1 of

[I-D.ietf-avtcore-rtp-multi-stream]). To address the other concerns, endpoints SHOULD ignore parameters that configure the RTCP reporting interval to be significantly longer than the default five second interval specified in [RFC3550] (unless the media data rate is so low that the longer reporting interval roughly corresponds to 5% of the media data rate), or that configure the RTCP reporting interval small enough that the RTCP bandwidth would exceed the media bandwidth.

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). The 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). The use of the encryption of the header extensions are RECOMMENDED, unless there are known reasons, like RTP middleboxes or third party monitoring that will greatly benefit from the information, and this has been expressed using API or signalling. If further evidence are produced to show that information leakage is significant from audio level indications, then use of encryption needs to be mandated at that time.

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. Acknowledgements

The authors would like to thank Bernard Aboba, Harald Alvestrand, Cary Bran, Ben Campbell, Charles Eckel, Alex Eleftheriadis, Christian Groves, Cullen Jennings, Olle Johansson, Suhas Nandakumar, Dan Romascanu, Jim Spring, Martin Thomson, and the other members of the IETF RTCWEB working group for their valuable feedback.

16. References

16.1. 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-06 (work in progress), October 2014.

[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-08 (work in progress), December 2014.

[I-D.ietf-avtcore-rtp-multi-stream-optimisation]
Lennox, J., Westerlund, M., Wu, Q., 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-06 (work in progress), October 2014.

[I-D.ietf-mmusic-sdp-bundle-negotiation]
Holmberg, C., Alvestrand, H., and C. Jennings, "Negotiating Media Multiplexing Using the Session Description Protocol (SDP)", draft-ietf-mmusic-sdp-bundle-negotiation-16 (work in progress), January 2015.

[I-D.ietf-rtcweb-audio]
Valin, J. and C. Bran, "WebRTC Audio Codec and Processing Requirements", draft-ietf-rtcweb-audio-07 (work in progress), October 2014.

[I-D.ietf-rtcweb-fec]
Uberti, J., "WebRTC Forward Error Correction Requirements", draft-ietf-rtcweb-fec-00 (work in progress), February 2015.

[I-D.ietf-rtcweb-security-arch]
Rescorla, E., "WebRTC Security Architecture", draft-ietf-rtcweb-security-arch-10 (work in progress), July 2014.

[I-D.ietf-rtcweb-security]
Rescorla, E., "Security Considerations for WebRTC", draft-ietf-rtcweb-security-07 (work in progress), July 2014.

[I-D.ietf-rtcweb-video]
Roach, A., "WebRTC Video Processing and Codec Requirements", draft-ietf-rtcweb-video-03 (work in progress), November 2014.

[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.

[RFC4566] Handley, M., Jacobson, V., and C. Perkins, "SDP: Session Description Protocol", RFC 4566, 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.

[RFC4588] Rey, J., Leon, D., Miyazaki, A., Varsa, V., and R. Hakenberg, "RTP Retransmission Payload Format", RFC 4588, July 2006.

[RFC4961] Wing, D., "Symmetric RTP / RTP Control Protocol (RTCP)", BCP 131, RFC 4961, July 2007.

[RFC5104] Wenger, S., Chandra, U., Westerlund, M., and B. Burman, "Codec Control Messages in the RTP Audio-Visual Profile with Feedback (AVPF)", RFC 5104, February 2008.

[RFC5124] Ott, J. and E. Carrara, "Extended Secure RTP Profile for Real-time Transport Control Protocol (RTCP)-Based Feedback (RTP/SAVPF)", RFC 5124, February 2008.

[RFC5285] Singer, D. and H. Desineni, "A General Mechanism for RTP Header Extensions", RFC 5285, July 2008.

[RFC5506] Johansson, I. and M. Westerlund, "Support for Reduced-Size Real-Time Transport Control Protocol (RTCP): Opportunities and Consequences", RFC 5506, April 2009.

[RFC5761] Perkins, C. and M. Westerlund, "Multiplexing RTP Data and Control Packets on a Single Port", RFC 5761, April 2010.

[RFC5764] McGrew, D. and E. Rescorla, "Datagram Transport Layer Security (DTLS) Extension to Establish Keys for the Secure Real-time Transport Protocol (SRTP)", RFC 5764, May 2010.

[RFC6051] Perkins, C. and T. Schierl, "Rapid Synchronisation of RTP Flows", RFC 6051, November 2010.

[RFC6464] Lennox, J., Ivov, E., and E. Marocco, "A Real-time Transport Protocol (RTP) Header Extension for Client-to-Mixer Audio Level Indication", RFC 6464, December 2011.

[RFC6465] Ivov, E., Marocco, E., and J. Lennox, "A Real-time Transport Protocol (RTP) Header Extension for Mixer-to-Client Audio Level Indication", RFC 6465, December 2011.

[RFC6562] Perkins, C. and JM. Valin, "Guidelines for the Use of Variable Bit Rate Audio with Secure RTP", RFC 6562, March 2012.

[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.

[RFC7160] Petit-Huguenin, M. and G. Zorn, "Support for Multiple Clock Rates in an RTP Session", RFC 7160, April 2014.

[RFC7164] Gross, K. and R. Brandenburg, "RTP and Leap Seconds", RFC 7164, March 2014.

16.2. Informative References

[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-03 (work in progress), October 2014.

[I-D.ietf-avtcore-rtp-topologies-update]
Westerlund, M. and S. Wenger, "RTP Topologies", draft-ietf-avtcore-rtp-topologies-update-05 (work in progress), November 2014.

[I-D.ietf-avtext-rtp-grouping-taxonomy]
Lennox, J., Gross, K., Nandakumar, S., and G. Salgueiro, "A Taxonomy of Grouping Semantics and Mechanisms for Real-Time Transport Protocol (RTP) Sources", draft-ietf-avtext-rtp-grouping-taxonomy-05 (work in progress), January 2015.

[I-D.ietf-mmusic-msid]
Alvestrand, H., "WebRTC MediaStream Identification in the Session Description Protocol", draft-ietf-mmusic-msid-07 (work in progress), October 2014.

[I-D.ietf-mmusic-sdp-bundle-negotiation]
Holmberg, C., Alvestrand, H., and C. Jennings, "Negotiating Media Multiplexing Using the Session Description Protocol (SDP)", draft-ietf-mmusic-sdp-bundle-negotiation-16 (work in progress), January 2015.

[I-D.ietf-payload-rtp-howto]
Westerlund, M., "How to Write an RTP Payload Format", draft-ietf-payload-rtp-howto-13 (work in progress), January 2014.

[I-D.ietf-rmcat-cc-requirements]
Jesup, R. and Z. Sarker, "Congestion Control Requirements for Interactive Real-Time Media", draft-ietf-rmcat-cc-requirements-09 (work in progress), December 2014.

[I-D.ietf-rtcweb-overview]
Alvestrand, H., "Overview: Real Time Protocols for Browser-based Applications", draft-ietf-rtcweb-overview-13 (work in progress), November 2014.

[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-16 (work in progress), January 2015.

[I-D.ietf-tsvwg-rtcweb-qos]
Dhesikan, S., Jennings, C., Druta, D., Jones, P., and J. Polk, "DSCP and other packet markings for RTCWeb QoS", draft-ietf-tsvwg-rtcweb-qos-03 (work in progress), November 2014.

[RFC3611] Friedman, T., Caceres, R., and A. Clark, "RTP Control Protocol Extended Reports (RTCP XR)", RFC 3611, November 2003.

[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.

[RFC5245] Rosenberg, J., "Interactive Connectivity Establishment (ICE): A Protocol for Network Address Translator (NAT) Traversal for Offer/Answer Protocols", RFC 5245, April 2010.

[RFC5576] Lennox, J., Ott, J., and T. Schierl, "Source-Specific Media Attributes in the Session Description Protocol (SDP)", RFC 5576, June 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.

[RFC6792] Wu, Q., Hunt, G., and P. Arden, "Guidelines for Use of the RTP Monitoring Framework", RFC 6792, November 2012.

[W3C.WD-mediacapture-streams-20130903]
Burnett, D., Bergkvist, A., Jennings, C., and A. Narayanan, "Media Capture and Streams", World Wide Web Consortium WD WD-mediacapture-streams-20130903, September 2013, <http://www.w3.org/TR/2013/WD-mediacapture-streams-20130903>.

[W3C.WD-webrtc-20130910]
Bergkvist, A., Burnett, D., Jennings, C., and A. Narayanan, "WebRTC 1.0: Real-time Communication Between Browsers", World Wide Web Consortium WD WD-webrtc-20130910, September 2013, <http://www.w3.org/TR/2013/WD-webrtc-20130910>.

Authors' Addresses

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

Email: csp@csperkins.org
URI: http://csperkins.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

RTCWEB
Internet-Draft
Intended status: Standards Track
Expires: June 20, 2015

M. Perumal
Ericsson
D. Wing
R. Ravindranath
T. Reddy
Cisco Systems
M. Thomson
Mozilla

December 17, 2014

STUN Usage for Consent Freshness
draft-ietf-rtcweb-stun-consent-freshness-11

Abstract

To prevent sending excessive traffic to an endpoint, periodic consent needs to be obtained from that remote endpoint.

This document describes a consent mechanism using a new Session Traversal Utilities for NAT (STUN) usage.

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

To prevent attacks on peers, endpoints have to ensure the remote peer is willing to receive traffic. This is performed both when the session is first established to the remote peer using Interactive Connectivity Establishment ICE [RFC5245] connectivity checks, and periodically for the duration of the session using the procedures defined in this document.

When a session is first established, ICE implementations obtain an initial consent to send by performing STUN connectivity checks. This document describes a new STUN usage with exchange of request and response messages that verifies the remote peer's ongoing consent to receive traffic. This consent expires after a period of time and needs to be continually renewed, which ensures that consent can be terminated.

This document defines what it takes to obtain, maintain, and lose consent to send. Consent to send applies to a single 5-tuple. How applications react to changes in consent is not described in this document.

Consent is obtained only by full ICE implementations. An ICE-lite implementation will not generate consent checks, but will just respond to consent checks it receives. No changes are required to ICE-lite implementations in order to respond to consent checks, as they are processed as normal ICE connectivity checks.

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].

Consent: The mechanism of obtaining permission to send to a remote transport address. Initial consent is obtained using ICE.

Consent Freshness: Maintaining and renewing consent over time.

Transport Address: The remote peer’s IP address and UDP or TCP port number.

3. Design Considerations

Although ICE requires periodic keepalive traffic to keep NAT bindings alive (Section 10 of [RFC5245], [RFC6263]), those keepalives are sent as STUN Indications which are send-and-forget, and do not evoke a response. A response is necessary for consent to continue sending traffic. Thus, we need a request/response mechanism for consent freshness. ICE can be used for that mechanism because ICE implementations are already required to continue listening for ICE messages, as described in section 10 of [RFC5245]. If consent is performed then there is no need to send keepalive messages.

4. Solution

There are two ways consent to send traffic is revoked: expiration of consent and immediate revocation of consent, which are discussed in the following sections.

4.1. Expiration of Consent

A full ICE implementation performs consent freshness test using STUN request/response as described below:

An endpoint MUST NOT send data other than paced STUN connectivity checks or responses toward any transport address unless the receiving endpoint consents to receive data. That is, no application data (e.g., RTP or DTLS) can be sent until consent is obtained. After a successful ICE connectivity check on a particular transport address,

consent MUST be maintained following the procedure described in this document.

Explicit consent to send is obtained and maintained by sending an STUN binding request to the remote peer’s transport address and receiving a matching, authenticated, non-error STUN binding response from the remote peer’s transport address. These STUN binding requests and responses are authenticated using the same short-term credentials as the initial ICE exchange.

Note: Although TCP has its own consent mechanism (TCP acknowledgements), consent is necessary over a TCP connection because it could be translated to a UDP connection (e.g., [RFC6062]).

Initial consent to send traffic is obtained using ICE. Consent expires after 30 seconds. That is, if a valid STUN binding response corresponding to any STUN request sent in the last 30 seconds has not been received from the remote peer’s transport address, the endpoint MUST cease transmission on that 5-tuple. STUN consent responses received after consent expiry do not re-establish consent, and may be discarded or cause an ICMP error.

To prevent expiry of consent, a STUN binding request can be sent periodically. To prevent synchronization of consent checks, each interval MUST be randomized from between 0.8 and 1.2 times the basic period. Implementations SHOULD set a default interval of 5 seconds, resulting in a period between checks of 4 to 6 seconds.

Each STUN binding request for consent MUST use a new cryptographically strong [RFC4086] STUN transaction ID. Each STUN binding requests for consent is transmitted once only. Hence, the sender cannot assume that it will receive a response for each consent request, and a response might be for a previous request (rather than for the most recently sent request). Consent expiration causes immediate termination of all outstanding STUN consent transactions. Each STUN transaction is maintained until one of the following criteria is fulfilled:

- A STUN response associated with the transaction is received; or

- A STUN response associated to a newer transaction is received.

To meet the security needs of consent, an untrusted application (e.g., JavaScript or signaling servers) MUST NOT be able to obtain or control the STUN transaction ID, because that enables spoofing of STUN responses, falsifying consent.

To prevent attacks on the peer during ICE restart, an endpoint that continues to send traffic on the previously validated candidate pair during ICE restart MUST continue to perform consent freshness on that candidate pair as described earlier.

While TCP affords some protection from off-path attackers ([RFC5961], [RFC4953]), there is still a risk an attacker could cause a TCP sender to send forever by spoofing ACKs. To prevent such an attack, consent checks MUST be performed over all transport connections, including TCP. In this way, an off-path attacker spoofing TCP segments can not cause a TCP sender to send once the consent timer expires (30 seconds).

An endpoint that is not sending any application data does not need to maintain consent. However, failure to send could cause any NAT or firewall mappings for the flow to expire. Furthermore, having one peer unable to send is detrimental to many protocols.

After consent is lost for any reason, the same ICE credentials MUST NOT be used on the affected 5-tuple again. That means that a new session, or an ICE restart, is needed to obtain consent to send.

4.2. Immediate Revocation of Consent

In some cases it is useful to signal that consent is terminated rather than relying on a timeout.

Consent for sending application data is immediately revoked by receipt of an authenticated message that closes the connection (e.g., a TLS fatal alert) or receipt of a valid and authenticated STUN response with error code Forbidden (403). Note however that consent revocation messages can be lost on the network, so an endpoint could resend these messages, or wait for consent to expire.

Receipt of an unauthenticated message that closes a connection (e.g., TCP FIN) does not indicate revocation of consent. Thus, an endpoint receiving an unauthenticated end-of-session message SHOULD continue sending media (over connectionless transport) or attempt to re-establish the connection (over connection-oriented transport) until consent expires or it receives an authenticated message revoking consent.

Note that an authenticated SRTCP BYE does not terminate consent; it only indicates the associated SRTP source has quit.

5. DiffServ Treatment for Consent

It is RECOMMENDED that STUN consent checks use the same Diffserv Codepoint markings as the ICE connectivity checks described in Section 7.1.2.4 of [RFC5245] for a given 5-tuple.

Note: It is possible that different Diffserv Codepoints are used by different media over the same transport address [I-D.ietf-tsvwg-rtcweb-qos]. Such a case is outside the scope of this document.

6. DTLS applicability

The DTLS applicability is identical to what is described in Section 4.2 of [RFC7350].

7. API Recommendations

The W3C specification MAY provide the following API points to provide feedback and control over consent:

1. Generate an event when consent has expired for a given 5-tuple, meaning that transmission of data has ceased. This could indicate what application data is affected, such as media or data channels.

8. Security Considerations

This document describes a security mechanism.

The security considerations discussed in [RFC5245] should also be taken into account.

SRTP is encrypted and authenticated with symmetric keys; that is, both sender and receiver know the keys. With two party sessions, receipt of an authenticated packet from the single remote party is a strong assurance the packet came from that party. However, when a session involves more than two parties, all of whom know each others keys, any of those parties could have sent (or spoofed) the packet. Such shared key distributions are possible with some MIKEY [RFC3830] modes, Security Descriptions [RFC4568], and EKT [I-D.ietf-avtcore-srtp-ekt]. Thus, in such shared keying distributions, receipt of an authenticated SRTP packet is not sufficient to verify consent.

9. IANA Considerations

This document does not require any action from IANA.

10. Acknowledgement

Thanks to Eric Rescorla, Harald Alvestrand, Bernard Aboba, Magnus Westerland, Cullen Jennings, Christer Holmberg, Simon Perreault, Paul Kyzivat, Emil Ivov, Jonathan Lennox, Inaki Baz Castillo, Rajmohan Banavi and Christian Groves for their valuable inputs and comments. Thanks to Christer Holmberg for doing a through review.

11. References

11.1. Normative References

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

[RFC4086] Eastlake, D., Schiller, J., and S. Crocker, "Randomness Requirements for Security", BCP 106, RFC 4086, June 2005.

[RFC5245] Rosenberg, J., "Interactive Connectivity Establishment (ICE): A Protocol for Network Address Translator (NAT) Traversal for Offer/Answer Protocols", RFC 5245, April 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.

11.2. Informative References

[I-D.ietf-avtcore-srtp-ekt]
Mattsson, J., McGrew, D., and D. Wing, "Encrypted Key Transport for Secure RTP", draft-ietf-avtcore-srtp-ekt-03 (work in progress), October 2014.

[I-D.ietf-rtcweb-overview]
Alvestrand, H., "Overview: Real Time Protocols for Browser-based Applications", draft-ietf-rtcweb-overview-13 (work in progress), November 2014.

[I-D.ietf-tsvwg-rtcweb-qos]
Dhesikan, S., Jennings, C., Druta, D., Jones, P., and J. Polk, "DSCP and other packet markings for RTCWeb QoS", draft-ietf-tsvwg-rtcweb-qos-03 (work in progress), November 2014.

[RFC3830] Arkko, J., Carrara, E., Lindholm, F., Naslund, M., and K. Norrman, "MIKEY: Multimedia Internet KEYing", RFC 3830, August 2004.

[RFC4568] Andreasen, F., Baugher, M., and D. Wing, "Session Description Protocol (SDP) Security Descriptions for Media Streams", RFC 4568, July 2006.

[RFC4953] Touch, J., "Defending TCP Against Spoofing Attacks", RFC 4953, July 2007.

[RFC5961] Ramaiah, A., Stewart, R., and M. Dalal, "Improving TCP's Robustness to Blind In-Window Attacks", RFC 5961, August 2010.

[RFC6062] Perreault, S. and J. Rosenberg, "Traversal Using Relays around NAT (TURN) Extensions for TCP Allocations", RFC 6062, November 2010.

[RFC7350] Petit-Huguenin, M. and G. Salgueiro, "Datagram Transport Layer Security (DTLS) as Transport for Session Traversal Utilities for NAT (STUN)", RFC 7350, August 2014.

Authors' Addresses

Muthu Arul Mozhi Perumal
Ericsson
Ferns Icon
Doddanekundi, Mahadevapura
Bangalore, Karnataka 560037
India

Email: muthu.arul@gmail.com

Dan Wing
Cisco Systems
821 Alder Drive
Milpitas, California 95035
USA

Email: dwing@cisco.com

Ram Mohan Ravindranath
Cisco Systems
Cessna Business Park
Sarjapur-Marathahalli Outer Ring Road
Bangalore, Karnataka 560103
India

Email: rmohanr@cisco.com

Tirumaleswar Reddy
Cisco Systems
Cessna Business Park, Varthur Hobli
Sarjapur Marathalli Outer Ring Road
Bangalore, Karnataka 560103
India

Email: tireddy@cisco.com

Martin Thomson
Mozilla
Suite 300
650 Castro Street
Mountain View, California 94041
US

Email: martin.thomson@gmail.com

Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: April 25, 2015

H. Alvestrand
Google

October 22, 2014

Transports for WebRTC
draft-ietf-rtcweb-transports-07

Abstract

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

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

WebRTC is a protocol suite aimed at real time multimedia exchange between browsers, and between browsers and other entities.

WebRTC is described in the WebRTC overview document, [I-D.ietf-rtcweb-overview], which also defines terminology used in this document.

This document focuses on the data transport protocols that are used by conforming implementations, including the protocols used for interaction with intermediate boxes such as firewalls, relays and NAT boxes.

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

This document describes requirements that apply to all WebRTC devices. When there are requirements that apply only to WebRTC User Agents (also called browsers) , this is called out.

The form "WebRTC endpoint" is used as a synonym for "WebRTC device" in contexts where other text talks about endpoints.

2. 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].

3. Transport and Middlebox specification

3.1. System-provided interfaces

The protocol specifications used here assume that the following protocols are available to the WebRTC devices:

- 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, IPv4 and IPv6 support is assumed.

For UDP, 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-tsvwg-rtcweb-qos] (see Section 4.1) when multiple media types are multiplexed. 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.

Platforms that do not give access to these interfaces will not be able to support a conforming WebRTC implementation.

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

3.2. Ability to use IPv4 and IPv6

Web applications running in a WebRTC browser MUST be able to utilize both IPv4 and IPv6 where available - that is, when two peers have only IPv4 connectivity to each other, or they have only IPv6

connectivity to each other, applications running in the WebRTC browser MUST be able to communicate.

WebRTC devices, when attached to networks with appropriate protocol support MUST also be able to communicate using IPv6 and IPv4.

When TURN is used, and the TURN server has IPv4 or IPv6 connectivity to the peer or its TURN server, candidates of the appropriate types MUST be supported. The "Happy Eyeballs" specification for ICE [I-D.reddy-mmusic-ice-happy-eyeballs] SHOULD be supported.

3.3. Usage of temporary IPv6 addresses

The IPv6 default address selection specification [RFC6724] specifies that temporary addresses [RFC4941] are to be preferred over permanent addresses. This is a change from the rules specified by [RFC3484]. For applications that select a single address, this is usually done by the IPV6_PREFER_SRC_TMP preference flag specified in [RFC5014]. However, this rule is not completely obvious in the ICE scope. This is therefore clarified as follows:

When a WebRTC endpoint gathers all IPv6 addresses on a host, and both temporary addresses and permanent addresses of the same scope are present, the client SHOULD discard the permanent addresses before forming pairs. This is consistent with the default policy described in [RFC6724].

3.4. Middle box related functions

Except when called out, all requirements in this section apply to all WebRTC devices.

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 is in response to inside traffic (simple stateful firewalls).

WebRTC endpoints MUST support ICE [RFC5245]. The implementation MUST be a full ICE implementation, not ICE-Lite. A full ICE implementation allows interworking with both ICE and ICE-Lite implementations when they are deployed appropriately.

In order to deal with situations where both parties are behind NATs of the type that perform endpoint-dependent mapping (as defined in [RFC5128] section 2.4), WebRTC endpoints MUST support TURN [RFC5766].

WebRTC browsers MUST support configuration of STUN and TURN servers, both from browser configuration and from an application.

In order to deal with firewalls that block all UDP traffic, the mode of TURN that uses TCP between the client and the server MUST be supported, and the mode of TURN that uses TLS over TCP between the client and the server MUST be supported. See [RFC5766] section 2.1 for details.

In order to deal with situations where one party is on an IPv4 network and the other party is on an IPv6 network, TURN extensions for IPv6 [RFC6156] MUST be supported.

TURN TCP candidates, where the connection from the client’s TURN server to the peer is a TCP connection, [RFC6062] MAY be supported.

However, such candidates are not seen as providing any significant benefit, for the following reasons.

First, use of TURN TCP candidates would only be relevant in cases which both peers are required to use TCP to establish a PeerConnection.

Second, that use case is supported in a different way by both sides establishing UDP relay candidates using TURN over TCP to connect to their respective relay servers.

Third, using TCP only between the endpoint and its relay may result in less issues with TCP in regards to real-time constraints, e.g. due to head of line blocking.

ICE-TCP candidates [RFC6544] MUST be supported; this may allow applications to communicate to peers with public IP addresses across UDP-blocking firewalls without using a TURN server.

If ICE-TCP connections are used, RTP framing according to [RFC4571] MUST be used for all content that doesn’t have its own framing mechanism.

The ALTERNATE-SERVER mechanism specified in [RFC5389] (STUN) section 11 (300 Try Alternate) MUST be supported.

In order to deal with the scenario in which the media must traverse a HTTP Proxy, WebRTC browser MUST support the HTTP CONNECT request (Section 4.3.6 of [RFC7231]). WebRTC devices SHOULD support this request.

The HTTP Proxy may require authentication and therefore, if HTTP CONNECT request is supported, proxy authentication as described in Section 4.3.6 of [RFC7231] and [RFC7235] MUST also be supported.

In addition, the HTTP CONNECT MUST include an indication of the protocol being used with the HTTP CONNECT initiated tunnel as described in [I-D.ietf-httpbis-tunnel-protocol]

3.5. Transport protocols implemented

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]. Key exchange MUST be done using DTLS-SRTP, as described in [I-D.ietf-rtcweb-security-arch].

For data transport over the WebRTC data channel [I-D.ietf-rtcweb-data-channel], WebRTC endpoints MUST support SCTP over DTLS over ICE. This encapsulation is specified in [I-D.ietf-tsvwg-sctp-dtls-encaps]. Negotiation of this transport in SDP is defined in [I-D.ietf-mmusic-sctp-sdp]. The SCTP extension for NDATA, [I-D.ietf-tsvwg-sctp-ndata], MUST be supported.

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

WebRTC devices MUST support multiplexing of DTLS and RTP over the same port pair, as described in the DTLS_SRTP specification [RFC5764], section 5.1.2. All application layer protocol payloads over this DTLS connection are SCTP packets.

Protocol identification MUST be supplied as part of the DTLS handshake, as specified in [I-D.thomson-rtcweb-alpn].

4. Media Prioritization

The WebRTC prioritization model is that the application tells the WebRTC browser about the priority of media and data flows through an API.

The priority associated with a media or data flow is classified as "normal", "below normal", "high" or "very high". There are only four priority levels at the API.

The priority settings affect two pieces of behavior: Packet markings and packet send sequence decisions. Each is described in its own section below.

4.1. Usage of Quality of Service - DSCP and Multiplexing

WebRTC endpoints SHOULD attempt to set QoS on the packets sent, according to the guidelines in [I-D.ietf-tsvwg-rtcweb-qos]. It is

appropriate to depart from this recommendation when running on platforms where QoS marking is not implemented.

The WebRTC endpoint MAY turn off use of DSCP markings if it detects symptoms of unexpected behaviour like priority inversion or blocking of packets with certain DSCP markings. The detection of these conditions is implementation dependent. (Question: Does there need to be an API knob to turn off DSCP markings?)

All packets carrying data from the SCTP association supporting the data channels MUST use a single DSCP code point.

All packets on one TCP connection, no matter what it carries, MUST use a single DSCP code point.

More advice on the use of DSCP code points with RTP is given in [I-D.ietf-dart-dscp-rtp].

There exist a number of schemes for achieving quality of service that do not depend solely on DSCP code points. Some of these schemes depend on classifying the traffic into flows based on 5-tuple (source address, source port, protocol, destination address, destination port) or 6-tuple (5-tuple + DSCP code point). Under differing conditions, it may therefore make sense for a sending application to choose any of the configurations:

- o Each media stream carried on its own 5-tuple
- o Media streams grouped by media type into 5-tuples (such as carrying all audio on one 5-tuple)
- o All media sent over a single 5-tuple, with or without differentiation into 6-tuples based on DSCP code points

In each of the configurations mentioned, data channels may be carried in its own 5-tuple, or multiplexed together with one of the media flows.

More complex configurations, such as sending a high priority video stream on one 5-tuple and sending all other video streams multiplexed together over another 5-tuple, can also be envisioned. More information on mapping media flows to 5-tuples can be found in [I-D.ietf-rtcweb-rtp-usage].

A sending WebRTC endpoint MUST be able to support the following configurations:

- o multiplex all media and data on a single 5-tuple (fully bundled)

- o send each media stream on its own 5-tuple and data on its own 5-tuple (fully unbundled)

- o bundle each media type (audio, video or data) into its own 5-tuple (bundling by media type)

It MAY choose to support other configurations.

Sending data over multiple 5-tuples is not supported.

A receiving WebRTC endpoint MUST be able to receive media and data in all these configurations.

4.2. Local prioritization

When an WebRTC endpoint has packets to send on multiple streams (with each media stream and each data channel considered as one "stream" for this purpose) that are congestion-controlled under the same congestion controller, the WebRTC endpoint SHOULD cause data to be emitted in such a way that each stream at each level of priority is being given approximately twice the transmission capacity (measured in payload bytes) of the level below.

Thus, when congestion occurs, a "very high" priority flow will have the ability to send 8 times as much data as a "below normal" flow if both have data to send. This prioritization is independent of the media type. The details of which packet to send first are implementation defined.

For example: If there is a very high priority audio flow sending 100 byte packets, and a normal priority video flow sending 1000 byte packets, and outgoing capacity exists for sending >5000 payload bytes, it would be appropriate to send 4000 bytes (40 packets) of audio and 1000 bytes (one packet) of video as the result of a single pass of sending decisions.

Conversely, if the audio flow is marked normal priority and the video flow is marked very high priority, the scheduler may decide to send 2 video packets (2000 bytes) and 5 audio packets (500 bytes) when outgoing capacity exists for sending > 2500 payload bytes.

If there are two very high priority audio flows, each will be able to send 4000 bytes in the same period where a normal priority video flow is able to send 1000 bytes.

Two example implementation strategies are:

- o When the available bandwidth is known from the congestion control algorithm, configure each codec and each data channel with a target send rate that is appropriate to its share of the available bandwidth.

- o When congestion control indicates that a specified number of packets can be sent, send packets that are available to send using a weighted round robin scheme across the connections.

Any combination of these, or other schemes that have the same effect, is valid, as long as the distribution of transmission capacity is approximately correct.

For media, it is usually inappropriate to use deep queues for sending; it is more useful to, for instance, skip intermediate frames that have no dependencies on them in order to achieve a lower bitrate. For reliable data, queues are useful.

5. IANA Considerations

This document makes no request of IANA.

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

6. Security Considerations

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

7. 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.

Special thanks for reviews of earlier versions of this draft go to Eduardo Gueiros, Magnus Westerlund, Markus Isomaki and Dan Wing; the contributions from Andrew Hutton also deserve special mention.

8. References

8.1. Normative References

[I-D.ietf-httpbis-tunnel-protocol]
Hutton, A., Uberti, J., and M. Thomson, "The Tunnel-Protocol HTTP Request Header Field", draft-ietf-httpbis-tunnel-protocol-00 (work in progress), August 2014.

[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-07 (work in progress), July 2014.

[I-D.ietf-rtcweb-data-channel]
Jesup, R., Loreto, S., and M. Tuexen, "WebRTC Data Channels", draft-ietf-rtcweb-data-channel-11 (work in progress), July 2014.

[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-15 (work in progress), May 2014.

[I-D.ietf-rtcweb-security]
Rescorla, E., "Security Considerations for WebRTC", draft-ietf-rtcweb-security-07 (work in progress), July 2014.

[I-D.ietf-rtcweb-security-arch]
Rescorla, E., "WebRTC Security Architecture", draft-ietf-rtcweb-security-arch-10 (work in progress), July 2014.

[I-D.ietf-tsvwg-rtcweb-qos]
Dhesikan, S., Jennings, C., Druta, D., Jones, P., and J. Polk, "DSCP and other packet markings for RTCWeb QoS", draft-ietf-tsvwg-rtcweb-qos-02 (work in progress), June 2014.

[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-05 (work in progress), July 2014.

[I-D.ietf-tsvwg-sctp-ndata]
Stewart, R., Tuexen, M., Loreto, S., and R. Seggelmann, "Stream Schedulers and a New Data Chunk for the Stream Control Transmission Protocol", draft-ietf-tsvwg-sctp-ndata-01 (work in progress), July 2014.

[I-D.reddy-mmusic-ice-happy-eyeballs]
Reddy, T., Patil, P., and P. Martinsen, "Happy Eyeballs Extension for ICE", draft-reddy-mmusic-ice-happy-eyeballs-07 (work in progress), June 2014.

[I-D.thomson-rtcweb-alpn]
Thomson, M., "Application Layer Protocol Negotiation for Web Real-Time Communications (WebRTC)", draft-thomson-rtcweb-alpn-00 (work in progress), April 2014.

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

[RFC4571] Lazzaro, J., "Framing Real-time Transport Protocol (RTP) and RTP Control Protocol (RTCP) Packets over Connection-Oriented Transport", RFC 4571, July 2006.

[RFC4941] Narten, T., Draves, R., and S. Krishnan, "Privacy Extensions for Stateless Address Autoconfiguration in IPv6", RFC 4941, 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.

[RFC5389] Rosenberg, J., Mahy, R., Matthews, P., and D. Wing, "Session Traversal Utilities for NAT (STUN)", RFC 5389, October 2008.

[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.

[RFC6156] Camarillo, G., Novo, O., and S. Perreault, "Traversal Using Relays around NAT (TURN) Extension for IPv6", RFC 6156, April 2011.

[RFC6544] Rosenberg, J., Keranen, A., Lowekamp, B., and A. Roach, "TCP Candidates with Interactive Connectivity Establishment (ICE)", RFC 6544, March 2012.

[RFC6724] Thaler, D., Draves, R., Matsumoto, A., and T. Chown, "Default Address Selection for Internet Protocol Version 6 (IPv6)", RFC 6724, September 2012.

[RFC7231] Fielding, R. and J. Reschke, "Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content", RFC 7231, June 2014.

[RFC7235] Fielding, R. and J. Reschke, "Hypertext Transfer Protocol (HTTP/1.1): Authentication", RFC 7235, June 2014.

8.2. Informative References

[I-D.ietf-dart-dscp-rtp]
Black, D. and P. Jones, "Differentiated Services (DiffServ) and Real-time Communication", draft-ietf-dart-dscp-rtp-08 (work in progress), October 2014.

[I-D.ietf-rtcweb-overview]
Alvestrand, H., "Overview: Real Time Protocols for Browser-based Applications", draft-ietf-rtcweb-overview-10 (work in progress), June 2014.

[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.

[RFC3484] Draves, R., "Default Address Selection for Internet Protocol version 6 (IPv6)", RFC 3484, February 2003.

[RFC5014] Nordmark, E., Chakrabarti, S., and J. Laganier, "IPv6 Socket API for Source Address Selection", RFC 5014, September 2007.

[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

This section should be removed before publication as an RFC.

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

A.2. Changes from -01 to -02

- o Required support for 300 Alternate Server from STUN.
- o Separated the ICE-TCP candidate requirement from the TURN-TCP requirement.
- o Added new sections on using QoS functions, and on multiplexing considerations.
- o Removed all mention of RTP profiles. Those are the business of the RTP usage draft, not this one.
- o Required support for TURN IPv6 extensions.
- o Removed reference to the TURN URI scheme, as it was unnecessary.
- o Made an explicit statement that multiplexing (or not) is an application matter.

.

A.3. Changes from -02 to -03

- o Added required support for draft-ietf-tsvwg-sctp-ndata
- o Removed discussion of multiplexing, since this is present in rtp-usage.
- o Added RFC 4571 reference for framing RTP packets over TCP.
- o Downgraded TURN TCP candidates from SHOULD to MAY, and added more language discussing TCP usage.
- o Added language on IPv6 temporary addresses.
- o Added language describing multiplexing choices.
- o Added a separate section detailing what it means when we say that an WebRTC implementation MUST support both IPv4 and IPv6.

A.4. Changes from -03 to -04

- Added a section on prioritization, moved the DSCP section into it, and added a section on local prioritization, giving a specific algorithm for interpreting "priority" in local prioritization.

- ICE-TCP candidates was changed from MAY to MUST, in recognition of the sense of the room at the London IETF.

A.5. Changes from -04 to -05

- Reworded introduction

- Removed all references to "WebRTC". It now uses only the term RTCWEB.

- Addressed a number of clarity / language comments

- Rewrote the prioritization to cover data channels and to describe multiple ways of prioritizing flows

- Made explicit reference to "MUST do DTLS-SRTP", and referred to security-arch for details

A.6. Changes from -05 to -06

- Changed all references to "RTCWEB" to "WebRTC", except one reference to the working group

- Added reference to the httpbis "connect" protocol (being adopted by HTTPBIS)

- Added reference to the ALPN header (being adopted by RTCWEB)

- Added reference to the DART RTP document

- Said explicitly that SCTP for data channels has a single DSCP codepoint

A.7. Changes from -06 to -07

- Updated terminology in accordance with -overview. Got rid of all occurences of "WebRTC implementation".

- Modified description of ICE-TCP encapsulation in accordance with list discussion.

- Added HTTP CONNECT requirement in accordance with list discussion.

Author’s Address

Harald Alvestrand
Google

Email: harald@alvestrand.no

Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: April 29, 2017

H. Alvestrand
Google

October 26, 2016

Transports for WebRTC
draft-ietf-rtcweb-transports-17

Abstract

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

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

WebRTC is a protocol suite aimed at real time multimedia exchange between browsers, and between browsers and other entities.

WebRTC is described in the WebRTC overview document, [I-D.ietf-rtcweb-overview], which also defines terminology used in this document, including the terms "WebRTC endpoint" and "WebRTC browser".

Terminology for RTP sources is taken from[RFC7656] .

This document focuses on the data transport protocols that are used by conforming implementations, including the protocols used for interaction with intermediate boxes such as firewalls, relays and NAT boxes.

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

This document describes requirements that apply to all WebRTC endpoints. When there are requirements that apply only to WebRTC browsers, this is called out explicitly.

2. 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].

3. Transport and Middlebox specification

3.1. System-provided interfaces

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

- o UDP [RFC0768]. This is the protocol assumed by most protocol elements described.

- o TCP [RFC0793]. This is used for HTTP/WebSockets, as well as for TURN/TLS and ICE-TCP.

For both protocols, IPv4 and IPv6 support is assumed.

For UDP, 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-tsvwg-rtcweb-qos] (see Section 4.2) when multiple media types are multiplexed. 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.

Platforms that do not give access to these interfaces will not be able to support a conforming WebRTC endpoint.

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

The following protocols may be used, but can be implemented by a WebRTC endpoint, and are therefore not defined as "system-provided interfaces":

- o TURN - Traversal Using Relays Around NAT, [RFC5766]
- o STUN - Session Traversal Utilities for NAT, [RFC5389]
- o ICE - Interactive Connectivity Establishment, [I-D.ietf-ice-rfc5245bis]
- o TLS - Transport Layer Security, [RFC5246]
- o DTLS - Datagram Transport Layer Security, [RFC6347].

3.2. Ability to use IPv4 and IPv6

Web applications running in a WebRTC browser MUST be able to utilize both IPv4 and IPv6 where available - that is, when two peers have only IPv4 connectivity to each other, or they have only IPv6 connectivity to each other, applications running in the WebRTC browser MUST be able to communicate.

When TURN is used, and the TURN server has IPv4 or IPv6 connectivity to the peer or the peer’s TURN server, candidates of the appropriate types MUST be supported. The "Happy Eyeballs" specification for ICE [I-D.ietf-mmusic-ice-dualstack-fairness] SHOULD be supported.

3.3. Usage of temporary IPv6 addresses

The IPv6 default address selection specification [RFC6724] specifies that temporary addresses [RFC4941] are to be preferred over permanent addresses. This is a change from the rules specified by [RFC3484]. For applications that select a single address, this is usually done by the IPV6_PREFER_SRC_TMP preference flag specified in [RFC5014]. However, this rule, which is intended to ensure that privacy-enhanced addresses are used in preference to static addresses, doesn’t have the right effect in ICE, where all addresses are gathered and therefore revealed to the application. Therefore, the following rule is applied instead:

When a WebRTC endpoint gathers all IPv6 addresses on its host, and both non-deprecated temporary addresses and permanent addresses of the same scope are present, the WebRTC endpoint SHOULD discard the permanent addresses before exposing addresses to the application or

using them in ICE. This is consistent with the default policy described in [RFC6724].

If some of the temporary IPv6 addresses, but not all, are marked deprecated, the WebRTC endpoint SHOULD discard the deprecated addresses, unless they are used by an ongoing connection. In an ICE restart, deprecated addresses that are currently in use MAY be retained.

3.4. 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 is in response to inside traffic (simple stateful firewalls).

ICE [I-D.ietf-ice-rfc5245bis] MUST be supported. The implementation MUST be a full ICE implementation, not ICE-Lite. A full ICE implementation allows interworking with both ICE and ICE-Lite implementations when they are deployed appropriately.

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

WebRTC browsers MUST support configuration of STUN and TURN servers, both from browser configuration and from an application.

Note that there is other work around STUN and TURN sever discovery and management, including [I-D.ietf-tram-turn-server-discovery] for server discovery, as well as [I-D.ietf-rtcweb-return].

In order to deal with firewalls that block all UDP traffic, the mode of TURN that uses TCP between the WebRTC endpoint and the TURN server MUST be supported, and the mode of TURN that uses TLS over TCP between the WebRTC endpoint and the TURN server MUST be supported. See [RFC5766] section 2.1 for details.

In order to deal with situations where one party is on an IPv4 network and the other party is on an IPv6 network, TURN extensions for IPv6 [RFC6156] MUST be supported.

TURN TCP candidates, where the connection from the WebRTC endpoint’s TURN server to the peer is a TCP connection, [RFC6062] MAY be supported.

However, such candidates are not seen as providing any significant benefit, for the following reasons.

First, use of TURN TCP candidates would only be relevant in cases which both peers are required to use TCP to establish a PeerConnection.

Second, that use case is supported in a different way by both sides establishing UDP relay candidates using TURN over TCP to connect to their respective relay servers.

Third, using TCP between the WebRTC endpoint’s TURN server and the peer may result in more performance problems than using UDP, e.g. due to head of line blocking.

ICE-TCP candidates [RFC6544] MUST be supported; this may allow applications to communicate to peers with public IP addresses across UDP-blocking firewalls without using a TURN server.

If TCP connections are used, RTP framing according to [RFC4571] MUST be used for all packets. This includes the RTP packets, DTLS packets used to carry data channels, and STUN connectivity check packets.

The ALTERNATE-SERVER mechanism specified in [RFC5389] (STUN) section 11 (300 Try Alternate) MUST be supported.

The WebRTC endpoint MAY support accessing the Internet through an HTTP proxy. If it does so, it MUST include the "ALPN" header as specified in [RFC7639], and proxy authentication as described in Section 4.3.6 of [RFC7231] and [RFC7235] MUST also be supported.

3.5. Transport protocols implemented

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], which mandates the use of a circuit breaker [I-D.ietf-avtcore-rtp-circuit-breakers] and congstion control (see [I-D.ietf-rmcat-cc-requirements] for further guidance).

Key exchange MUST be done using DTLS-SRTP, as described in [I-D.ietf-rtcweb-security-arch].

For data transport over the WebRTC data channel [I-D.ietf-rtcweb-data-channel], WebRTC endpoints MUST support SCTP over DTLS over ICE. This encapsulation is specified in [I-D.ietf-tsvwg-sctp-dtls-encaps]. Negotiation of this transport in SDP is defined in [I-D.ietf-mmusic-sctp-sdp]. The SCTP extension for NDATA, [I-D.ietf-tsvwg-sctp-ndata], MUST be supported.

The setup protocol for WebRTC data channels described in [I-D.ietf-rtcweb-data-protocol] MUST be supported.

Note: DTLS-SRTP as defined in [RFC5764] section 6.7.1 defines the interaction between DTLS and ICE ([I-D.ietf-ice-rfc5245bis]). The effect of this specification is that all ICE candidate pairs associated with a single component are part of the same DTLS association. Thus, there will only be one DTLS handshake even if there are multiple valid candidate pairs.

WebRTC endpoints MUST support multiplexing of DTLS and RTP over the same port pair, as described in the DTLS-SRTP specification [RFC5764], section 5.1.2, with clarifications in [I-D.ietf-avtcore-rfc5764-mux-fixes]. All application layer protocol payloads over this DTLS connection are SCTP packets.

Protocol identification MUST be supplied as part of the DTLS handshake, as specified in [I-D.ietf-rtcweb-alpn].

4. Media Prioritization

The WebRTC prioritization model is that the application tells the WebRTC endpoint about the priority of media and data that is controlled from the API.

In this context, a "flow" is used for the units that are given a specific priority through the WebRTC API.

For media, a "media flow", which can be an "audio flow" or a "video flow", is what [RFC7656] calls a "media source", which results in a "source RTP stream" and one or more "redundancy RTP streams". This specification does not describe prioritization between the RTP streams that come from a single "media source".

All media flows in WebRTC are assumed to be interactive, as defined in [RFC4594]; there is no browser API support for indicating whether media is interactive or non-interactive.

A "data flow" is the outgoing data on a single WebRTC data channel.

The priority associated with a media flow or data flow is classified as "very-low", "low", "medium or "high". There are only four priority levels at the API.

The priority settings affect two pieces of behavior: Packet send sequence decisions and packet markings. Each is described in its own section below.

4.1. Local prioritization

Local prioritization is applied at the local node, before the packet is sent. This means that the prioritization has full access to the data about the individual packets, and can choose differing treatment based on the stream a packet belongs to.

When an WebRTC endpoint has packets to send on multiple streams that are congestion-controlled under the same congestion control regime, the WebRTC endpoint SHOULD cause data to be emitted in such a way that each stream at each level of priority is being given approximately twice the transmission capacity (measured in payload bytes) of the level below.

Thus, when congestion occurs, a "high" priority flow will have the ability to send 8 times as much data as a "very-low" priority flow if both have data to send. This prioritization is independent of the media type. The details of which packet to send first are implementation defined.

For example: If there is a high priority audio flow sending 100 byte packets, and a low priority video flow sending 1000 byte packets, and outgoing capacity exists for sending >5000 payload bytes, it would be appropriate to send 4000 bytes (40 packets) of audio and 1000 bytes (one packet) of video as the result of a single pass of sending decisions.

Conversely, if the audio flow is marked low priority and the video flow is marked high priority, the scheduler may decide to send 2 video packets (2000 bytes) and 5 audio packets (500 bytes) when outgoing capacity exists for sending > 2500 payload bytes.

If there are two high priority audio flows, each will be able to send 4000 bytes in the same period where a low priority video flow is able to send 1000 bytes.

Two example implementation strategies are:

- When the available bandwidth is known from the congestion control algorithm, configure each codec and each data channel with a target send rate that is appropriate to its share of the available bandwidth.

- When congestion control indicates that a specified number of packets can be sent, send packets that are available to send using a weighted round robin scheme across the connections.

Any combination of these, or other schemes that have the same effect, is valid, as long as the distribution of transmission capacity is approximately correct.

For media, it is usually inappropriate to use deep queues for sending; it is more useful to, for instance, skip intermediate frames that have no dependencies on them in order to achieve a lower bitrate. For reliable data, queues are useful.

Note that this specification doesn’t dictate when disparate streams are to be "congestion controlled under the same congestion control regime". The issue of coupling congestion controllers is explored further in [I-D.ietf-rmcat-coupled-cc].

4.2. Usage of Quality of Service - DSCP and Multiplexing

When the packet is sent, the network will make decisions about queueing and/or discarding the packet that can affect the quality of the communication. The sender can attempt to set the DSCP field of the packet to influence these decisions.

Implementations SHOULD attempt to set QoS on the packets sent, according to the guidelines in [I-D.ietf-tsvwg-rtcweb-qos]. It is appropriate to depart from this recommendation when running on platforms where QoS marking is not implemented.

The implementation MAY turn off use of DSCP markings if it detects symptoms of unexpected behaviour like priority inversion or blocking of packets with certain DSCP markings. Some examples of such behaviors are described in [ANRW16]. The detection of these conditions is implementation dependent.

A particularly hard problem is when one media transport uses multiple DSCP code points, where one may be blocked and another may be allowed. This is allowed even within a single media flow for video in [I-D.ietf-tsvwg-rtcweb-qos]. Implementations need to diagnose this scenario; one possible implementation is to send initial ICE probes with DSCP 0, and send ICE probes on all the DSCP code points that are intended to be used once a candidate pair has been selected. If one or more of the DSCP-marked probes fail, the sender will switch the media type to using DSCP 0. This can be carried out simultaneously with the initial media traffic; on failure, the initial data may need to be resent. This switch will of course invalidate any congestion information gathered up to that point.

Failures can also start happening during the lifetime of the call; this case is expected to be rarer, and can be handled by the normal mechanisms for transport failure, which may involve an ICE restart.

Note that when a DSCP code point causes non-delivery, one has to switch the whole media flow to DSCP 0, since all traffic for a single media flow needs to be on the same queue for congestion control purposes. Other flows on the same transport, using different DSCP code points, don’t need to change.

All packets carrying data from the SCTP association supporting the data channels MUST use a single DSCP code point. The code point used SHOULD be that recommended by [I-D.ietf-tsvwg-rtcweb-qos] for the highest priority data channel carried. Note that this means that all data packets, no matter what their relative priority is, will be treated the same by the network.

All packets on one TCP connection, no matter what it carries, MUST use a single DSCP code point.

More advice on the use of DSCP code points with RTP and on the relationship between DSCP and congestion control is given in [RFC7657].

There exist a number of schemes for achieving quality of service that do not depend solely on DSCP code points. Some of these schemes depend on classifying the traffic into flows based on 5-tuple (source address, source port, protocol, destination address, destination port) or 6-tuple (5-tuple + DSCP code point). Under differing conditions, it may therefore make sense for a sending application to choose any of the configurations:

- Each media stream carried on its own 5-tuple

- Media streams grouped by media type into 5-tuples (such as carrying all audio on one 5-tuple)

- All media sent over a single 5-tuple, with or without differentiation into 6-tuples based on DSCP code points

In each of the configurations mentioned, data channels may be carried in its own 5-tuple, or multiplexed together with one of the media flows.

More complex configurations, such as sending a high priority video stream on one 5-tuple and sending all other video streams multiplexed together over another 5-tuple, can also be envisioned. More information on mapping media flows to 5-tuples can be found in [I-D.ietf-rtcweb-rtp-usage].

A sending implementation MUST be able to support the following configurations:

- o Multiplex all media and data on a single 5-tuple (fully bundled)

- o Send each media stream on its own 5-tuple and data on its own 5-tuple (fully unbundled)

It MAY choose to support other configurations, such as bundling each media type (audio, video or data) into its own 5-tuple (bundling by media type).

Sending data channel data over multiple 5-tuples is not supported.

A receiving implementation MUST be able to receive media and data in all these configurations.

5. IANA Considerations

This document makes no request of IANA.

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

6. Security Considerations

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

Security considerations pertaining to the use of DSCP are enumerated in [I-D.ietf-tsvwg-rtcweb-qos].

7. 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.

Special thanks for reviews of earlier versions of this draft go to Eduardo Gueiros, Magnus Westerlund, Markus Isomaki and Dan Wing; the contributions from Andrew Hutton also deserve special mention.

8. References

8.1. Normative References

[I-D.ietf-avtcore-rfc5764-mux-fixes]
Petit-Huguenin, M. and G. Salgueiro, "Multiplexing Scheme Updates for Secure Real-time Transport Protocol (SRTP) Extension for Datagram Transport Layer Security (DTLS)", draft-ietf-avtcore-rfc5764-mux-fixes-11 (work in progress), September 2016.

[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-06 (work in progress), July 2014.

[I-D.ietf-ice-rfc5245bis]
Keranen, A., Holmberg, C., and J. Rosenberg, "Interactive Connectivity Establishment (ICE): A Protocol for Network Address Translator (NAT) Traversal", draft-ietf-ice-rfc5245bis-04 (work in progress), June 2016.

[I-D.ietf-mmusic-ice-dualstack-fairness]
Martinsen, P., Reddy, T., and P. Patil, "ICE Multihomed and IPv4/IPv6 Dual Stack Fairness", draft-ietf-mmusic-ice-dualstack-fairness-02 (work in progress), September 2015.

[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-07 (work in progress), July 2014.

[I-D.ietf-rmcat-cc-requirements]
Jesup, R., "Congestion Control Requirements For RMCAT", draft-ietf-rmcat-cc-requirements-06 (work in progress), October 2014.

[I-D.ietf-rtcweb-alpn]
Thomson, M., "Application Layer Protocol Negotiation for Web Real-Time Communications (WebRTC)", draft-ietf-rtcweb-alpn-00 (work in progress), July 2014.

[I-D.ietf-rtcweb-data-channel]
Jesup, R., Loreto, S., and M. Tuexen, "WebRTC Data Channels", draft-ietf-rtcweb-data-channel-12 (work in progress), September 2014.

[I-D.ietf-rtcweb-data-protocol]
Jesup, R., Loreto, S., and M. Tuexen, "WebRTC Data Channel Establishment Protocol", draft-ietf-rtcweb-data-protocol-08 (work in progress), September 2014.

[I-D.ietf-rtcweb-overview]
Alvestrand, H., "Overview: Real Time Protocols for Browser-based Applications", draft-ietf-rtcweb-overview-11 (work in progress), August 2014.

[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-17 (work in progress), August 2014.

[I-D.ietf-rtcweb-security]
Rescorla, E., "Security Considerations for WebRTC", draft-ietf-rtcweb-security-07 (work in progress), July 2014.

[I-D.ietf-rtcweb-security-arch]
Rescorla, E., "WebRTC Security Architecture", draft-ietf-rtcweb-security-arch-10 (work in progress), July 2014.

[I-D.ietf-tsvwg-rtcweb-qos]
Dhesikan, S., Jennings, C., Druta, D., Jones, P., and J. Polk, "DSCP and other packet markings for RTCWeb QoS", draft-ietf-tsvwg-rtcweb-qos-02 (work in progress), June 2014.

[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-05 (work in progress), July 2014.

[I-D.ietf-tsvwg-sctp-ndata]
Stewart, R., Tuexen, M., Loreto, S., and R. Seggelmann, "Stream Schedulers and a New Data Chunk for the Stream Control Transmission Protocol", draft-ietf-tsvwg-sctp-ndata-01 (work in progress), July 2014.

[RFC0768] Postel, J., "User Datagram Protocol", STD 6, RFC 768, August 1980.

[RFC0793] Postel, J., "Transmission Control Protocol", STD 7, RFC 793, September 1981.

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

[RFC4571] Lazzaro, J., "Framing Real-time Transport Protocol (RTP) and RTP Control Protocol (RTCP) Packets over Connection-Oriented Transport", RFC 4571, July 2006.

[RFC4594] Babiarz, J., Chan, K., and F. Baker, "Configuration Guidelines for DiffServ Service Classes", RFC 4594, August 2006.

[RFC4941] Narten, T., Draves, R., and S. Krishnan, "Privacy Extensions for Stateless Address Autoconfiguration in IPv6", RFC 4941, September 2007.

[RFC5246] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", RFC 5246, August 2008.

[RFC5389] Rosenberg, J., Mahy, R., Matthews, P., and D. Wing, "Session Traversal Utilities for NAT (STUN)", RFC 5389, October 2008.

[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.

[RFC6156] Camarillo, G., Novo, O., and S. Perreault, "Traversal Using Relays around NAT (TURN) Extension for IPv6", RFC 6156, April 2011.

[RFC6347] Rescorla, E. and N. Modadugu, "Datagram Transport Layer Security Version 1.2", RFC 6347, January 2012.

[RFC6544] Rosenberg, J., Keranen, A., Lowekamp, B., and A. Roach, "TCP Candidates with Interactive Connectivity Establishment (ICE)", RFC 6544, March 2012.

[RFC6724] Thaler, D., Draves, R., Matsumoto, A., and T. Chown, "Default Address Selection for Internet Protocol Version 6 (IPv6)", RFC 6724, September 2012.

[RFC7231] Fielding, R. and J. Reschke, "Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content", RFC 7231, June 2014.

[RFC7235] Fielding, R. and J. Reschke, "Hypertext Transfer Protocol (HTTP/1.1): Authentication", RFC 7235, June 2014.

[RFC7639] Hutton, A., Uberti, J., and M. Thomson, "The ALPN HTTP Header Field", RFC 7639, DOI 10.17487/RFC7639, August 2015, <http://www.rfc-editor.org/info/rfc7639>.

[RFC7656] Lennox, J., Gross, K., Nandakumar, S., Salgueiro, G., and B. Burman, Ed., "A Taxonomy of Semantics and Mechanisms for Real-Time Transport Protocol (RTP) Sources", RFC 7656, DOI 10.17487/RFC7656, November 2015, <http://www.rfc-editor.org/info/rfc7656>.

8.2. Informative References

[ANRW16] Barik, R., Welzl, M., and A. Elmokashfi, "How to say that you’re special: Can we use bits in the IPv4 header?", ACM, IRTF, ISOC Applied Networking Research Workshop (ANRW 2016), Berlin , July 2016.

[I-D.ietf-rmcat-coupled-cc]
Islam, S., Welzl, M., and S. Gjessing, "Coupled congestion control for RTP media", draft-ietf-rmcat-coupled-cc-03 (work in progress), July 2016.

[I-D.ietf-rtcweb-return]
Schwartz, B. and J. Uberti, "Recursively Encapsulated TURN (RETURN) for Connectivity and Privacy in WebRTC", draft-ietf-rtcweb-return-01 (work in progress), January 2016.

[I-D.ietf-tram-turn-server-discovery]
Patil, P., Reddy, T., and D. Wing, "TURN Server Auto Discovery", draft-ietf-tram-turn-server-discovery-00 (work in progress), July 2014.

[RFC3484] Draves, R., "Default Address Selection for Internet Protocol version 6 (IPv6)", RFC 3484, February 2003.

[RFC5014] Nordmark, E., Chakrabarti, S., and J. Laganier, "IPv6 Socket API for Source Address Selection", RFC 5014, September 2007.

[RFC5128] Srisuresh, P., Ford, B., and D. Kegel, "State of Peer-to-Peer (P2P) Communication across Network Address Translators (NATs)", RFC 5128, March 2008.

[RFC7657] Black, D., Ed. and P. Jones, "Differentiated Services (Diffserv) and Real-Time Communication", RFC 7657, DOI 10.17487/RFC7657, November 2015, <http://www.rfc-editor.org/info/rfc7657>.

Appendix A. Change log

This section should be removed before publication as an RFC.

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

A.2. Changes from -01 to -02

- o Required support for 300 Alternate Server from STUN.
- o Separated the ICE-TCP candidate requirement from the TURN-TCP requirement.
- o Added new sections on using QoS functions, and on multiplexing considerations.
- o Removed all mention of RTP profiles. Those are the business of the RTP usage draft, not this one.
- o Required support for TURN IPv6 extensions.
- o Removed reference to the TURN URI scheme, as it was unnecessary.
- o Made an explicit statement that multiplexing (or not) is an application matter.

.

A.3. Changes from -02 to -03

- Added required support for draft-ietf-tsvwg-sctp-ndata
- Removed discussion of multiplexing, since this is present in rtp-usage.
- Added RFC 4571 reference for framing RTP packets over TCP.
- Downgraded TURN TCP candidates from SHOULD to MAY, and added more language discussing TCP usage.
- Added language on IPv6 temporary addresses.
- Added language describing multiplexing choices.
- Added a separate section detailing what it means when we say that an WebRTC implementation MUST support both IPv4 and IPv6.

A.4. Changes from -03 to -04

- Added a section on prioritization, moved the DSCP section into it, and added a section on local prioritization, giving a specific algorithm for interpreting "priority" in local prioritization.
- ICE-TCP candidates was changed from MAY to MUST, in recognition of the sense of the room at the London IETF.

A.5. Changes from -04 to -05

- Reworded introduction
- Removed all references to "WebRTC". It now uses only the term RTCWEB.
- Addressed a number of clarity / language comments
- Rewrote the prioritization to cover data channels and to describe multiple ways of prioritizing flows
- Made explicit reference to "MUST do DTLS-SRTP", and referred to security-arch for details

A.6. Changes from -05 to -06

- Changed all references to "RTCWEB" to "WebRTC", except one reference to the working group

- o Added reference to the httpbis "connect" protocol (being adopted by HTTPBIS)
- o Added reference to the ALPN header (being adopted by RTCWEB)
- o Added reference to the DART RTP document
- o Said explicitly that SCTP for data channels has a single DSCP codepoint

A.7. Changes from -06 to -07

- o Updated references
- o Removed reference to draft-hutton-rtcweb-nat-firewall-considerations

A.8. Changes from -07 to -08

- o Updated references
- o Deleted "bundle each media type (audio, video or data) into its own 5-tuple (bundling by media type)" from MUST support configuration, since JSEP does not have a means to negotiate this configuration

A.9. Changes from -08 to -09

- o Added a clarifying note about DTLS-SRTP and ICE interaction.

A.10. Changes from -09 to -10

- o Re-added references to proxy authentication lost in 07-08 transition (Bug #5)
- o Rearranged and rephrased text in section 4 about prioritization to reflect discussions in TSVWG.
- o Changed the "Connect" header to "ALPN", and updated reference. (Bug #6)

A.11. Changes from -10 to -11

- o Added a definition of the term "flow" used in the prioritization chapter
- o Changed the names of the four priority levels to conform to other specs.

A.12. Changes from -11 to -12

- o Added a SHOULD NOT about using deprecated temporary IPv6 addresses.
- o Updated draft-ietf-dart-dscp-rtp reference to RFC 7657

A.13. Changes from -12 to -13

- o Clarify that the ALPN header needs to be sent.
- o Mentioned that RFC 7657 also talks about congestion control

A.14. Changes from -13 to -14

- o Add note about non-support for marking flows as interactive or non-interactive.

A.15. Changes from -14 to -15

- o Various text clarifications based on comments in Last Call and IESG review
- o Clarified that only non-deprecated IPv6 addresses are used
- o Described handling of downgrading of DSCP markings when blackholes are detected
- o Expanded acronyms in a new protocol list

A.16. Changes from -15 to -16

These changes are done post IESG approval, and address IESG comments and other late comments. Issue numbers refer to https://github.com/rtcweb-wg/rtcweb-transport/issues.

- o Moved RFC 4594, 7656 and -overview to normative (issue #28)
- o Changed the terms "client", "WebRTC implementation" and "WebRTC device" to consistently be "WebRTC endpoint", as defined in -overview. (issue #40)
- o Added a note mentioning TURN service discovery and RETURN (issue #42)
- o Added a note mentioning that rtp-usage requires circut breaker and congestion control (issue #43)

- o Added mention of the "don’t discard temporary IPv6 addresses that are in use" (issue #44)

- o Added a reference to draft-ietf-rmcat-coupled-cc (issue #46)

A.17. Changes from -16 to -17

- o Added an informative reference to the "DSCP blackholing" paper

- o Changed the reference for ICE from RFC 5245 to draft-ietf-ice-rfc5245bis

Author’s Address

Harald Alvestrand
Google

Email: harald@alvestrand.no

Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: May 29, 2015

A.B. Roach
Mozilla

November 25, 2014

WebRTC Video Processing and Codec Requirements
draft-ietf-rtcweb-video-03

Abstract

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

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

One of the major functions of WebRTC endpoints is the ability to send and receive interactive video. The video might come from a camera, a screen recording, a stored file, or some other source. This specification defines how the video is used and discusses special considerations for processing the video. It also covers the video-related algorithms WebRTC devices need to support.

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

2. Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119].

3. Pre and Post Processing

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

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

TODO: I’m just throwing this out there to see if a specific proposal, even if wrong, might draw more comment than "TBD". If you don’t like sRGB for this purpose, comment on the rtcweb@ietf.org mailing list. It has been suggested that the MPEG "Coding independent media description code points" specification [IEC23001-8] may have applicability here.

3.1. Camera Source Video

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

- o Automatic focus, if applicable for the camera in use
- o Automatic white balance
- o Automatic light level control

3.2. Screen Source Video

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

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

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

4. Stream Orientation

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

While the sender may elect to simply change the pre-encoding orientation of frames, this may not be practical or efficient (in particular, in cases where the interface to the camera returns pre-

compressed video frames). Note that the potential for this behavior adds another set of circumstances under which the resolution of a screen might change in the middle of a video stream, in addition to those mentioned under "Screen Sourced Video," above.

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

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

5. Mandatory to Implement Video Codec

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

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

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

> To promote the use of non-royalty bearing video codecs, participants in the RTCWEB working group, and any successor working groups in the IETF, intend to monitor the evolving licensing landscape as it pertains to the two mandatory-to-implement codecs. If compelling evidence arises that one of the codecs is available for use on a royalty-free basis, the working group plans to revisit the question of which codecs are required for Non-Browsers, with the intention being that the royalty-free codec will remain mandatory to implement, and the other will become optional.
>
> These provisions apply to WebRTC Non-Browsers only. There is no plan to revisit the codecs required for WebRTC Browsers.

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

6. Codec-Specific Considerations

SDP allows for codec-independent indication of preferred video resolutions using the mechanism described in [RFC6236]. If a recipient of video indicates a receiving resolution, the sender SHOULD accommodate this resolution, as the receiver may not be capable of handling higher resolutions.

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

Unless otherwise signaled in SDP, recipients of video streams are MUST be able to decode video at a rate of at least 20 fps at a resolution of at least 320x240. These values are selected based on the recommendations in [HSUP1].

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

6.1. VP8

For the VP8 codec, defined in [RFC6386], endpoints MUST support the payload formats defined in [I-D.ietf-payload-vp8]. In addition they MUST support the ’bilinear’ and ’none’ reconstruction filters.

TODO: There have been claims that VP8 already requires supporting both filters; if true, these do not need to be reiterated here.

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

6.2. H.264

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

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

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

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

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

sprop-parameter-sets: H.264 allows sequence and picture information to be sent both in-band, and out-of-band. WebRTC implementations MUST signal this information in-band; as a result, this parameter will not be present in SDP.

TODO: Do we need to require the handling of specific SEI messages? One example that has been raised is freeze-frame messages.

7. Security Considerations

This specification does not introduce any new mechanisms or security concerns beyond what the other documents it references. In WebRTC, video is protected using DTLS/SRTP. A complete discussion of the security can be found in [I-D.ietf-rtcweb-security] and [I-D.ietf-rtcweb-security-arch]. Implementers should consider whether the use of variable bit rate video codecs are appropriate for their application based on [RFC6562].

8. IANA Considerations

This document requires no actions from IANA.

9. Acknowledgements

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

10. References

10.1. Normative References

[H264] ITU-T Recommendation H.264, "Advanced video coding for generic audiovisual services", April 2013.

[HSUP1] ITU-T Recommendation H.Sup1, "Application profile - Sign language and lip-reading real-time conversation using low bit rate video communication", May 1999.

[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-vp8-11 (work in progress), February 2014.

[I-D.ietf-rtcweb-overview] Alvestrand, H., "Overview: Real Time Protocols for Browser-based Applications", draft-ietf-rtcweb-overview-12 (work in progress), October 2014.

[IEC23001-8] ISO/IEC 23001-8:2013/DCOR1, "Coding independent media description code points", 2013.

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

[RFC4175] Gharai, L. and C. Perkins, "RTP Payload Format for Uncompressed Video", RFC 4175, September 2005.

[RFC4421] Perkins, C., "RTP Payload Format for Uncompressed Video: Additional Colour Sampling Modes", RFC 4421, February 2006.

[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.

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

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

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

[RFC6562] Perkins, C. and JM. Valin, "Guidelines for the Use of Variable Bit Rate Audio with Secure RTP", RFC 6562, March 2012.

[SRGB] IEC 61966-2-1, "Multimedia systems and equipment - Colour measurement and management - Part 2-1: Colour management - Default RGB colour space - sRGB.", October 1999.

[TS26.114] 3GPP TS 26.114 V12.7.0, "3rd Generation Partnership Project; Technical Specification Group Services and System Aspects; IP Multimedia Subsystem (IMS); Multimedia Telephony; Media handling and interaction (Release 12)", September 2014.

10.2. Informative References

[I-D.ietf-rtcweb-rtp-usage] Perkins, C., Westerlund, M., and J. Ott, "Web Real-Time Communication (WebRTC): Media Transport and Use of RTP", draft-ietf-rtcweb-rtp-usage-06 (work in progress), February 2013.

[I-D.ietf-rtcweb-security-arch] Rescorla, E., "WebRTC Security Architecture", draft-ietf-rtcweb-security-arch-09 (work in progress), February 2014.

[I-D.ietf-rtcweb-security] Rescorla, E., "Security Considerations for WebRTC", draft-ietf-rtcweb-security-06 (work in progress), January 2014.

Author’s Address

Adam Roach
Mozilla
\
Dallas
US

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

Network Working Group
Internet-Draft
Intended status: Informational
Expires: August 16, 2015

S. Nandakumar
C. Jennings
Cisco

February 12, 2015

SDP for the WebRTC
draft-nandakumar-rtcweb-sdp-07

Abstract

The Web Real-Time Communication [WebRTC] working group is charged to provide protocol support for direct interactive rich communication using audio, video and data between two peers' web browsers. With in the WebRTC framework, Session Description protocol (SDP) [RFC4566] is used for negotiating session capabilities between the peers. Such a negotiation happens based on the SDP Offer/Answer exchange mechanism described in [RFC3264].

This document provides an informational reference in describing the role of SDP and the Offer/Answer exchange mechanism for the most common WebRTC use-cases.

This SDP examples provided in this document is still a work in progress, but it aims to align closest to the evolving standards work.

Copyright Notice

Table of Contents

1. Introduction

Javascript Session Exchange Protocol(JSEP) [I-D.ietf-rtcweb-jsep] specifies a generic protocol needed to generate [RFC3264] Offers and Answers negotiated between the WebRTC peers for setting up, updating and tearing down a WebRTC session. For this purpose, SDP is used to construct [RFC3264] Offers/Answers for describing (media and non-media) streams as appropriate for the recipients of the session description to participate in the session.

The remainder of this document is organized as follows: Sections 3 and 4 provides an overview of SDP and the Offer/Answer exchange mechanism. Section 5 provides sample SDP generated for the most common WebRTC use-cases.

2. Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119].

3. SDP and the WebRTC

The purpose of this section is to provide a general overview of SDP and its components. For a more in-depth understanding, the readers are advised to refer to [RFC4566].

The Session Description Protocol (SDP) [RFC4566] describes multimedia sessions, which can contain audio, video, whiteboard, fax, modem, and other streams. SDP provides a general purpose, standard representation to describe various aspects of multimedia session such as media capabilities, transport addresses and related metadata in a transport agnostic manner, for the purposes of session announcement, session invitation and parameter negotiation.

As of today SDP is widely used in the context of Session Initiation Protocol [RFC3261], Real-time Transport Protocol [RFC3550] and Real-time Streaming Protocol applications [RFC2326].

Below figure introduces high-level breakup of SDP into components that semantically describe a multimedia session, in our case, a

WebRTC session [WebRTC]. It by no means captures everything about SDP and hence, should be used for informational purposes only.

```
                                                +---------------------+
                                                |        v=           |
                                                +---------------------+
                  +---------------------+       +---------------------+
          ====    |   Session Metadata  |  =====|        o=           |
          |       +---------------------+       +----------------------
          |                                     +---------------------+
          |                                     |        t=           |
          |                                     +---------------------+
          |
          |
          |                                     +---------------------+
          |                                     |        c=           |
          |                                     +---------------------+
          |       +---------------------+
          ====    | Network Description |   =====
          |       +---------------------+
          |                                     +---------------------+
          |                                     |    a=candidate      |
          |                                     +---------------------+
          |
          |
          |                                     +---------------------+
          |                                     |        m=           |
          |                                     +---------------------+
          |        +---------------------+       +---------------------+
          ====     | Stream Description  |  =====|      a=rtpmap       |
          |        +---------------------+       +----------------------
          |                                     +---------------------+
          |                                     |      a=fmtp         |
          |                                     +---------------------+
          |                                     +---------------------+
          |                                     |      a=sendrecv..   |
          |                                     +---------------------+
 +---------------+
 |    SEMANTIC   |
 | COMPONENTS OF |
 |     SDP       |
 +---------------+
          |                                     +---------------------+
          |                                     |      a=crypto       |
          |                                     +---------------------+
          |         +---------------------+     +---------------------+
          ====      |Security Descriptions|  =====|      a=ice-frag     |
          |         +---------------------+     +----------------------
```

```
         |                                      +---------------------+
         |                                      |      a=ice-pwd      |
         |                                      +---------------------+
         |                                      +---------------------+
         |                                      |     a=fingerprint   |
         |                                      +---------------------+
         |
         |
         |
         |                                      +---------------------+
         |                                      |      a=rtcp-fb      |
         |                                      +---------------------+
         |         +---------------------+      +---------------------+
         ====      |   Qos,Grouping      |      |                     |
                   |   Descriptions      |  =====|       a=group       |
                   +---------------------+      +----------------------
                                                +---------------------+
                                                |       a=rtcpmux     |
                                                +---------------------+
```

Figure 1: Semantic Components of SDP

[WebRTC] proposes JavaScript application to fully specify and control the signaling plane of a multimedia session as described in the JSEP specification [I-D.ietf-rtcweb-jsep]. JSEP provides mechanisms to create session characterization and media definition information to conduct the session based on SDP exchanges.

In this context, SDP serves two purposes:

1. Provide grammatical structure syntactically.

2. Semantically convey partipant’s intention and capabilities required to successfully negotiate a session.

4. Offer/Answer and the WebRTC

This section introduces SDP Offer/Answer Exchange mechanism mandated by WebRTC for negotiating session capabilities while setting up, updating and tearing down a WebRTC session. This section is intentionally brief in nature and interested readers are recommended to refer [RFC3264] for specific details on the protocol operation.

The Offer/Answer [RFC3264] model specifies rule for the bilateral exchange of Session Description Protocol (SDP) messages for creation of multimedia streams. It defines protocol with involved participants exchanging desired session characteristics from each others perspective constructed as SDP to negotiate the session between them.

In the most basic form,the protocol operation begins by one of the participants sending an initial SDP Offer describing its intent to start a multimedia communication session. The participant receiving the offer MAY generate an SDP Answer accepting the offer or it MAY reject the offer. If the session is accepted the Offer/Answer model guarantees a common view of the multimedia session between the participants.

At any time, either participant MAY generate a new SDP offer that updates the session in progress.

With in the context of WebRTC, the Offer/Answer model defines the state-machinery for WebRTC peers to negotiate session descriptions between them during the initial setup stages as well as for eventual session updates. Javascript Session Establishment Protocol specification [I-D.ietf-rtcweb-jsep] for WebRTC provides the mechanism for generating [RFC3264] SDP Offers and Answers in order for both sides of the session to agree upon details such as list of media formats to be sent/received, bandwidth information, crypto parameters, transport parameters, for example.

5. WebRTC Session Description Examples

A typical web based real-time multimedia communication session can be characterized as below:

- It has zero or more Audio only, Video only or Audio/Video RTP Sessions,
- MAY contain zero or more non-media data sessions,
- All the sessions are secured with DTLS-SRTP,
- Supports NAT traversal using ICE mechanism,
- Provides RTCP based feedback mechanisms,
- Sessions can be over IPv4-only, IPv6-only, dual-stack based clients.

5.1. Some Conventions

The examples given in this document follow the conventions listed below:

- o In all the examples, Alice and Bob are assumed to be the WebRTC peers.

- o [I-D.ietf-mmusic-sdp-bundle-negotiation] support for multiplexing several media streams over a single underlying transport is assumed by default unless explicitly specified otherwise.

- o Call flow diagrams that accompany the use-cases capture only the prominent aspects of the system behavior and intentionally is not detailed to improve readability.

- o The SDP examples deviate from actual on-the-wire SDP notation in several ways. This is done to facilitate readability and to conform to the restrictionsimposed by the RFC formatting rules.

  - * Any SDP line that is indented (compared to the initial line in the SDP block) is a continuation of the preceding line. The line break and indent are to be interpreted as a single space character.

  - * Empty lines in any SDP example are inserted to make functional divisions in the SDP clearer, and are not actually part of the SDP syntax.

  - * Excepting the above two conventions, line endings are to be interpreted as <CR><LF> pairs (that is, an ASCII 13 followed by an ASCII 10).

- o Against each SDP line, pointers to the appropriate RFCs are provide for further informational reference. Also an attempt has been made to provide explanatory notes to enable better understanding of the SDP usage, wherever appropriate.

- o Following SDP details are common across all the use-cases defined in this document unless mentioned otherwise.

  - * DTLS fingerprint for SRTP (a=fingerprint)

  - * RTP/RTCP Multiplexing (a=rtcp-mux)

  - * RTCP Feedback support (a=rtcp-fb)

  - * Host and server-reflexive candidate lines (a=candidate)

* SRTP Setup framework parameters (a=setup)

* RTCP attribute (a=rtcp)

* RTP header extension indicating audio-levels from client to the mixer

For more details, readers are recommended to refer to [I-D.ietf-rtcweb-jsep] specification.

- o The term "Session" is used rather loosely in this document to refer to either a "Communication Session" or a "RTP Session" or a "RTP Stream" depending on the context.

- o Payload type 109 is usually used for OPUS, 0 for PCMU, 8 for PCMA, 99 for H.264 and 120 for VP8 in most of the examples to maintain uniformity.

- o In the actual use the values that represent SSRCs, ICE candidate foundations, WebRTC Mediastream and MediaStreamTrack Ids shall be much larger and random than the ones shown in the examples.

  [OPEN ISSUE-1]: SDP Examples for Data Channel, Simulcast, SVC are still being discussed and doesn’t represent the final solution.

5.2. Basic Examples

5.2.1. Audio Only Session

This common scenario shows SDP for secure two-way audio session with Alice offering Opus, PCMU, PCMA and Bob accepting all the offered audio codecs.

Two Way Audio Only Session

```
Alice                                 Bob
|                                      |
|                                      |
|    Offer(Audio:Opus,PCMU,PCMA)       |
|------------------------------------->|
|                                      |
|                                      |
|   Answer(Audio:Opus,PCMU,PCMA)       |
|<-------------------------------------|
|                                      |
|                                      |
|Two-way Opus Audio (preferred-codec)  |
|......................................|
|                                      |
|                                      |
```

```
+----------------------------------+-------------------------------+
| SDP Contents                     | RFC#/Notes                    |
+----------------------------------+-------------------------------+
| v=0                              | [RFC4566]                     |
| o=- 20518 0 IN IP4 0.0.0.0       | [RFC4566] - Session Origin    |
|                                  | Information                   |
| s=-                              | [RFC4566]                     |
| t=0 0                            | [RFC4566]                     |
| a=msid-semantic:WMS ma           | [I-D.ietf-mmusic-msid]        |
| a=group:BUNDLE audio             | [I-D.ietf-mmusic-sdp-bundle-ne |
|                                  | gotiation]                    |
| m=audio 54609 UDP/TLS/RTP/SAVPF  | [RFC4566]                     |
| 109 0 8                          |                               |
| c=IN IP4 24.23.204.141           | [RFC4566]                     |
| a=mid:audio                      | [RFC5888]                     |
| a=msid:ma ta                     | Identifies RTCMediaStream ID  |
|                                  | (ma) and RTCMediaStreamTrack  |
|                                  | ID (ta)                       |
| a=rtcp-mux                       | [RFC5761] - Alice can perform |
|                                  | RTP/RTCP Muxing               |
| a=rtcp:54609 IN IP4              | [RFC3605] - Port for RTCP data |
| 24.23.204.141                    |                               |
| a=rtpmap:109 opus/48000/2        | [I-D.ietf-payload-rtp-opus] - |
|                                  | Opus Codec 48khz, 2 channels  |
| a=ptime:60                       | [I-D.ietf-payload-rtp-opus] - |
|                                  | Opus packetization of 60ms    |
| a=rtpmap:0 PCMU/8000             | [RFC3551] PCMU Audio Codec    |
| a=rtpmap:8 PCMA/8000             | [RFC3551] PCMA Audio Codec    |
| a=extmap:1 urn:ietf:params:rtp-  | [RFC6464] Alice supports RTP  |
```

SDP Contents	RFC#/Notes
hdrext:ssrc-audio-level	header extension to indicate audio levels
a=sendrecv	[RFC3264] - Alice can send and recv audio
a=setup:actpass	[RFC4145] - Alice can perform DTLS before Answer arrives
a=fingerprint:sha-1 99:41:49:83:4a:97:0e:1f:ef:6d:f7:c9:c7:70:9d:1f:66:79:a8:07	[RFC5245] - DTLS Fingerprint for SRTP
a=ice-ufrag:074c6550	[RFC5245] - ICE user fragment
a=ice-pwd:a28a397a4c3f31747d1ee3474af08a068	[RFC5245] - ICE password
a=candidate:0 1 UDP 2122194687 192.168.1.4 54609 typ host	[RFC5245] - RTP Host Candidate
a=candidate:0 2 UDP 2122194687 192.168.1.4 54609 typ host	[RFC5245] - RTCP Host Candidate
a=candidate:1 1 UDP 1685987071 24.23.204.141 64678 typ srflx raddr 192.168.1.4 rport 54609	[RFC5245] - RTP Server Reflexive ICE Candidate
a=candidate:1 2 UDP 1685987071 24.23.204.141 64678 typ srflx raddr 192.168.1.4 rport 54609	[RFC5245] - RTCP Server Reflexive Candidate
a=rtcp-fb:109 nack	[RFC5104] - Indicates NACK RTCP feedback support
a=ssrc:12345 cname:EocUG1f0fcg/yvY7	[RFC5576]
a=rtcp-rsize	[RFC5506] - Alice intends to use reduced size RTCP for this session
a=ice-options:trickle	[I-D.ietf-mmusic-trickle-ice]

Table 1: 5.2.1 SDP Offer

SDP Contents	RFC#/Notes
v=0	[RFC4566]
o=- 16833 0 IN IP4 0.0.0.0	[RFC4566] - Session Origin Information
s=-	[RFC4566]
t=0 0	[RFC4566]
a=msid-semantic:WMS ma	[I-D.ietf-mmusic-msid]
a=group:BUNDLE audio	[I-D.ietf-mmusic-sdp-bundle-negotiation]
m=audio 49203 UDP/TLS/RTP/SAVPF 109 0 8	[RFC4566]
c=IN IP4 98.248.92.77	[RFC4566]

a=mid:audio	[RFC5888]
a=msid:ma ta	Identifies RTCMediaStream ID (ma) and RTCMediaStreamTrack ID (ta)
a=rtpmap:109 opus/48000/2	[I-D.ietf-payload-rtp-opus] Opus Codec
a=ptime:60	[I-D.ietf-payload-rtp-opus] Packetization of 60ms
a=rtpmap:0 PCMU/8000	[RFC3551] PCMU Audio Codec
a=rtpmap:8 PCMA/8000	[RFC3551] PCMA Audio Codec
a=extmap:1 urn:ietf:params:rtp-hdrext:ssrc-audio-level	[RFC6464] Bob supports audio level RTP header extension as well
a=sendrecv	[RFC3264] - Bob can send and recv audio
a=setup:active	[RFC4145] - Bob carries out DTLS Handshake in parallel
a=rtcp-mux	[RFC5761] - Bob can perform RTP/RTCP Muxing on port 49203
a=fingerprint:sha-1 c9:c7:70:9d:1f:66:79:a8:07:99:41:49:83:4a:97:0e:1f:ef:6d:f7	[RFC5245] - DTLS Fingerprint for SRTP
a=ice-ufrag:05067423	[RFC5245] - ICE user fragment
a=ice-pwd:1747d1ee3474a28a397a4c3f3af08a068	[RFC5245] - ICE password parameter
a=candidate:0 1 UDP 2122194687 192.168.1.7 49203 typ host	[RFC5245] - RTP/RTCP Host ICE Candidate
a=candidate:1 1 UDP 1685987071 98.248.92.77 60654 typ srflx raddr 192.168.1.7 rport 49203	[RFC5245] - RTP/RTCP Server Reflexive ICE Candidate
a=rtcp-fb:109 nack	[RFC5104] - Indicates NACK RTCP feedback support
a=ssrc:54321 cname:NWs1ao1HmN4Xa5/yvY7	[RFC5576]
a=rtcp-rsize	[RFC5506] - Bob intends to use reduced size RTCP for this session
a=ice-options:trickle	[I-D.ietf-mmusic-trickle-ice]

Table 2: 5.2.1 SDP Answer

5.2.2. Audio/Video Session

Alice and Bob establish a two-way audio and video session with Opus as the audio codec and H.264 as the video codec.

Two Way Audio,Video Session

```
Alice                                          Bob
|                                               |
|                                               |
|Offer(Audio:Opus,PCMU,PCMA Video:H.264,VP8)    |
|---------------------------------------------->|
|                                               |
|                                               |
|      Answer(Audio:Opus,Video:H.264)           |
|<----------------------------------------------|
|                                               |
|                                               |
|     Two-way Opus Audio, H.264 Video           |
|...............................................|
|                                               |
```

SDP Contents	RFC#/Notes
v=0	[RFC4566]
o=- 20518 0 IN IP4 0.0.0.0	[RFC4566] - Session Origin Information
s=-	[RFC4566]
t=0 0	[RFC4566]
a=msid-semantic:WMS ma	[I-D.ietf-mmusic-msid]
a=group:BUNDLE audio video	[I-D.ietf-mmusic-sdp-bundle-negotiation]
m=audio 54609 UDP/TLS/RTP/SAVPF 109 0 8	[RFC4566]
c=IN IP4 24.23.204.141	[RFC4566]
a=mid:audio	[RFC5888]
a=msid:ma ta	Identifies RTCMediaStream ID (ma) and RTCMediaStreamTrack ID (ta)
a=rtcp-mux	[RFC5761] - Alice can perform RTP/RTCP Muxing
a=rtcp:54609 IN IP4 24.23.204.141	[RFC3605] - Port for RTCP data
a=rtpmap:109 opus/48000/2	[I-D.ietf-payload-rtp-opus] - Opus Codec 48khz, 2 channels
a=ptime:60	[I-D.ietf-payload-rtp-opus] - Opus packetization of 60ms
a=rtpmap:0 PCMU/8000	[RFC3551] PCMU Audio Codec
a=rtpmap:8 PCMA/8000	[RFC3551] PCMA Audio Codec
a=extmap:1 urn:ietf:params:rtp-hdrext:ssrc-audio-level	[RFC6464]

```
| a=sendrecv                        | [RFC3264] - Alice can send and |
|                                   | recv audio                     |
| a=setup:actpass                   | [RFC4145] - Alice can perform  |
|                                   | DTLS before Answer arrives     |
| a=ice-ufrag:074c6550              | [RFC5245] - ICE user fragment  |
| a=ice-pwd:a28a397a4c3f31747d1ee3 | [RFC5245] - ICE password       |
| 474af08a068                       | parameter                      |
| a=fingerprint:sha-1 99:41:49:83:  | [RFC5245] - DTLS Fingerprint   |
| 4a:97:0e:1f:ef:6d:f7:c9:c7:70:9d  | for SRTP                       |
| : 1f:66:79:a8:07                  |                                |
| a=candidate:0 1 UDP  2122194687   | [RFC5245] - RTP Host Candidate |
| 192.168.1.4 54609 typ host        |                                |
| a=candidate:0 2 UDP 2122194687    | [RFC5245] - RTCP Host          |
| 192.168.1.4 54609 typ host        | Candidate                      |
| a=candidate:1 1 UDP  1685987071   | [RFC5245] - RTP Server         |
| 24.23.204.141 54609 typ srflx     | Reflexive ICE Candidate        |
| raddr 192.168.1.4 rport 54609     |                                |
| a=candidate:1 2 UDP  1685987071   | [RFC5245] - RTCP Server        |
| 24.23.204.141 54609 typ srflx     | Reflexive Candidate.           |
| raddr 192.168.1.4 rport 54609     |                                |
| a=rtcp-fb:109 nack                | [RFC5104] - Indicates NACK     |
|                                   | RTCP feedback support          |
| a=ssrc:12345                      | [RFC5576]                      |
| cname:EocUG1f0fcg/yvY7            |                                |
| a=rtcp-rsize                      | [RFC5506] - Alice intends to   |
|                                   | use reduced size RTCP for this |
|                                   | session                        |
| a=ice-options:trickle             | [I-D.ietf-mmusic-trickle-ice]  |
| m=video 54609 UDP/TLS/RTP/SAVPF   | [RFC4566]                      |
| 99 120                            |                                |
| c=IN IP4 24.23.204.141            | [RFC4566]                      |
| a=mid:video                       | [RFC5888]                      |
| a=msid:ma tb                      | Identifies RTCMediaStream ID   |
|                                   | (ma) and RTCMediaStreamTrack   |
|                                   | ID (tb)                        |
| a=rtcp-mux                        | [RFC5761] - Alice can perform  |
|                                   | RTP/RTCP Muxing                |
| a=rtcp:54609 IN IP4               | [RFC3605] - Port for RTCP data |
| 24.23.204.141                     |                                |
| a=rtpmap:99 H264/90000            | [RFC3984] - H.264 Video Codec  |
| a=fmtp:99 profile-level-          | [RFC3984]                      |
| id=4d0028;packetization-mode=1    |                                |
| a=rtpmap:120 VP8/90000            | [I-D.ietf-payload-vp8] - VP8   |
|                                   | video codec                    |
| a=sendrecv                        | [RFC3264] - Alice can send and |
|                                   | recv video                     |
| a=setup:actpass                   | [RFC4145] - Alice can perform  |
|                                   | DTLS before Answer arrives     |
```

a=ice-ufrag:074c6550	[RFC5245] - ICE user fragment
a=ice-pwd:a28a397a4c3f31747d1ee3 474af08a068	[RFC5245] - ICE password parameter
a=fingerprint:sha-1 99:41:49:83: 4a:97:0e:1f:ef:6d:f7:c9:c7:70: 9d:1f:66:79:a8:07	[RFC5245] - DTLS Fingerprint for SRTP
a=candidate:0 1 UDP 2122194687 192.168.1.4 54609 typ host	[RFC5245] - RTP Host ICE Candidate
a=candidate:0 2 UDP 2122194687 192.168.1.4 54609 typ host	[RFC5245] - RTCP Host Candidate
a=candidate:1 1 UDP 1685987071 24.23.204.141 54609 typ srflx raddr 192.168.1.4 rport 54609	[RFC5245] - RTP Server Reflexive ICE Candidate
a=candidate:1 2 UDP 1685987071 24.23.204.141 54609 typ srflx raddr 192.168.1.4 rport 54609	[RFC5245] - RTCP Server Reflexive Candidate
a=rtcp-fb:99 nack	[RFC5104] - Indicates NACK RTCP feedback support
a=rtcp-fb:99 nack pli	[RFC5104] - Indicates support for Picture loss Indication and NACK
a=rtcp-fb:99 ccm fir	[RFC5104] - Full Intra Frame Request-Codec Control Message support
a=rtcp-fb:120 nack	[RFC5104] - Indicates NACK RTCP feedback support
a=rtcp-fb:120 nack pli	[RFC5104] - Indicates support for Picture loss Indication and NACK
a=rtcp-fb:120 ccm fir	[RFC5104] - Full Intra Frame Request-Codec Control Message support
a=ssrc:1366781083 cname:EocUG1f0fcg/yvY7	[RFC5576]
a=rtcp-rsize	[RFC5506] - Alice intends to use reduced size RTCP for this session
a=ice-options:trickle	[I-D.ietf-mmusic-trickle-ice]

Table 3: 5.2.2 SDP Offer

SDP Contents	RFC#/Notes
v=0	[RFC4566]
o=- 16833 0 IN IP4 0.0.0.0	[RFC4566] - Session Origin Information

```
| s=-                                | [RFC4566]                       |
| t=0 0                              | [RFC4566]                       |
| a=msid-semantic:WMS ma             | [I-D.ietf-mmusic-msid]          |
| a=group:BUNDLE audio video         | [I-D.ietf-mmusic-sdp-bundle-ne  |
|                                    | gotiation]                      |
| m=audio 49203 UDP/TLS/RTP/SAVPF    | [RFC4566]                       |
| 109                                |                                 |
| c=IN IP4 98.248.92.77              | [RFC4566]                       |
| a=mid:audio                        | [RFC5888]                       |
| a=msid:ma ta                       | Identifies RTCMediaStream ID    |
|                                    | (ma) and RTCMediaStreamTrack    |
|                                    | ID (ta)                         |
| a=rtcp-mux                         | [RFC5761] - Bob can perform     |
|                                    | RTP/RTCP Muxing                 |
| a=rtpmap:109 opus/48000/2          | [I-D.ietf-payload-rtp-opus] -   |
|                                    | Bob accepts only Opus Codec     |
| a=extmap:1 urn:ietf:params:rtp-    | [RFC6464]                       |
| hdrext:ssrc-audio-level            |                                 |
| a=ptime:60                         | [I-D.ietf-payload-rtp-opus]     |
| a=sendrecv                         | [RFC3264] - Bob can send and    |
|                                    | recv audio                      |
| a=setup:active                     | [RFC4145] - Bob carries out     |
|                                    | DTLS Handshake in parallel      |
| a=ice-ufrag:c300d85b               | [RFC5245] -  ICE username frag  |
| a=ice-pwd:de4e99bd291c325921d5d4   | [RFC5245] - ICE password        |
| 7efbabd9a2                         |                                 |
| a=fingerprint:sha-1 99:41:49:83:   | [RFC5245] - DTLS Fingerprint    |
| 4a:97:0e:1f:ef:6d:f7:c9:c7:70:     | for SRTP                        |
| 9d:1f:66:79:a8:07                  |                                 |
| a=candidate:0 1 UDP 3618095783     | [RFC5245] - RTP/RTCP Host ICE   |
| 192.168.1.7 49203 typ host         | Candidate                       |
| a=candidate:1 1 UDP 565689203      | [RFC5245] - RTP/RTCP Server     |
| 98.248.92.77 60065 typ srflx       | Reflexive ICE Candidate         |
| raddr 192.168.1.7 rport 49203      |                                 |
| a=ssrc:1366788312                  | [RFC5576]                       |
| cname:1f0fcgEocUG/yvY7             |                                 |
| a=rtcp-rsize                       | [RFC5506] - Bob intends to use  |
|                                    | reduced size RTCP for this      |
|                                    | session                         |
| a=ice-options:trickle              | [I-D.ietf-mmusic-trickle-ice]   |
| m=video 49203 UDP/TLS/RTP/SAVPF    | [RFC4566]                       |
| 99                                 |                                 |
| c=IN IP4 98.248.92.77              | [RFC4566]                       |
| a=mid:video                        | [RFC5888]                       |
| a=msid:ma tb                       | Identifies RTCMediaStream ID    |
|                                    | (ma) and RTCMediaStreamTrack    |
|                                    | ID (tb)                         |
| a=rtpmap:99 H264/90000             | [RFC3984] - Bob accepts H.264   |
```

	Video Codec.
a=fmtp:99 profile-level-id=4d0028;packetization-mode=1	[RFC3984]
a=rtcp-mux	[RFC5761] - Bob can perform RTP/RTCP Muxing
a=sendrecv	[RFC3264] - Bob can send and recv video
a=setup:active	[RFC4145] - Bob carries out DTLS Handshake in parallel
a=ice-ufrag:c300d85b	[RFC5245] - ICE username frag
a=ice-pwd:de4e99bd291c325921d5d47efbabd9a2	[RFC5245] - ICE password
a=fingerprint:sha-1 99:41:49:83:4a:97:0e:1f:ef:6d:f7:c9:c7:70:9d:1f:66:79:a8:07	[RFC5245] - DTLS Fingerprint for SRTP
a=candidate:0 1 UDP 2113667327 192.168.1.7 49203 typ host	[RFC5245] - RTP/RTCP Host ICE Candidate
a=candidate:1 1 UDP 1694302207 98.248.92.77 60065 typ srflx raddr 192.168.1.7 rport 49203	[RFC5245] - Server Reflexive ICE Candidate for the above host candidate
a=rtcp-fb:99 nack	[RFC5104] - Indicates support for NACK based RTCP feedback
a=rtcp-fb:99 nack pli	[RFC5104] - Indicates support for Picture loss Indication and NACK
a=rtcp-fb:99 ccm fir	[RFC5104] - Full Intra Frame Request- Codec Control Message support
a=ssrc:3229706345 cname:Q/NWs1ao1HmN4Xa5	[RFC5576]
a=rtcp-rsize	[RFC5506] - Bob intends to use reduced size RTCP for this session
a=ice-options:trickle	[I-D.ietf-mmusic-trickle-ice]

Table 4: 5.2.2 SDP Answer

5.2.3. Data Only Session

This scenario illustrates SDP negotiated to setup a data-only session based on SCTP Data Channel, thus enabling use-cases such as file-transfer for example.

2-Way DataChannel Session

```
Alice                              Bob
|                                   |
|                                   |
|                                   |
|      Offer(DataChannel)           |
|---------------------------------->|
|                                   |
|                                   |
|      Answer(DataChannel)          |
|<----------------------------------|
|                                   |
|                                   |
| Two-way SCTP based DataChannel    |
|...................................|
|                                   |
|                                   |
```

SDP Contents	RFC#/Notes
v=0	[RFC4566]
o=- 20518 0 IN IP4 0.0.0.0	[RFC4566] - Session Origin Information
s=-	[RFC4566]
t=0 0	[RFC4566]
a=group:BUNDLE data	[I-D.ietf-mmusic-sdp-bundle-negotiation]
a=ice-ufrag:074c6550	[RFC5245] - Session Level ICE parameter
a=ice-pwd:a28a397a4c3f31747d1ee3474af08a068	[RFC5245] - Session Level ICE parameter
a=fingerprint:sha-1 99:41:49:83:4a:97:0e:1f:ef:6d:f7:c9:c7:70:9d:1f:66:79:a8:07	[RFC5245] - Session DTLS Fingerprint for SRTP
m=application 56966 DTLS/SCTP 5000	[I-D.ietf-rtcweb-data-channel]
c=IN IP4 24.23.204.141	[RFC4566]
a=mid:data	[RFC5888]
a=sctpmap:5000 webrtc-DataChannel streams=16;label="channel 1"; subprotocol="chat";	[I-D.ietf-mmusic-sctp-sdp]
a=setup:actpass	[RFC4145] - Alice can perform DTLS before Answer arrives
a=sendrecv	[RFC3264] - Alice can send and recv non-media data
a=candidate:0 1 UDP 2113667327 192.168.1.7 56966 typ host	[RFC5245]
a=candidate:1 1 UDP 1694302207 24.23.204.141 56966 typ srflx raddr 192.168.1.7 rport 56966	[RFC5245]
a=ice-options:trickle	[I-D.ietf-mmusic-trickle-ice]

Table 5: 5.2.3 SDP Offer

SDP Contents	RFC#/Notes
v=0	[RFC4566]
o=- 16833 0 IN IP4 0.0.0.0	[RFC4566] - Session Origin Information
s=-	[RFC4566]
t=0 0	[RFC4566]
a=group:BUNDLE data	[I-D.ietf-mmusic-sdp-bundle-negotiation]
m=application 55700 DTLS/SCTP 5000	[I-D.ietf-mmusic-sctp-sdp]
c=IN IP4 98.248.92.771	[RFC4566]
a=mid:data	[RFC5888]
a=sctpmap:5000 webrtc-DataChannel:5000 streams=1;label="channel 1" ;subprotocol="chat";	[I-D.ietf-mmusic-sctp-sdp]
a=setup:active	[RFC4145] - Bob carries out DTLS Handshake in parallel
a=sendrecv	[RFC3264] - Bob can send and recv non-media data
a=ice-ufrag:c300d85b	[RFC5245] - Session Level ICE username frag
a=ice-pwd:de4e99bd291c325921d5d47efbabd9a2	[RFC5245] - Session Level ICE password
a=fingerprint:sha-1 99:41:49:83:4a:97:0e:1f:ef:6d:f7:c9:c7:70:9d:1f:66:79:a8:07	[RFC5245] - Session DTLS Fingerprint for SRTP
a=candidate:0 1 UDP 2113667327 192.168.1.7 55700 typ host	[RFC5245]
a=candidate:1 1 UDP 1694302207 98.248.92.77 55700 typ srflx raddr 192.168.1.7 rport 55700	[RFC5245]
a=ice-options:trickle	[I-D.ietf-mmusic-trickle-ice]

Table 6: 5.2.3 SDP Answer

5.2.4. Audio Call On Hold

Alice calls Bob, but when Bob answers he places Alice on hold by setting the SDP direction attribute to a=sendonly in the Answer.

Audio On Hold

```
Alice                              Bob
|                                   |
|                                   |
|      Offer(Audio:Opus)            |
|---------------------------------->|
|                                   |
|                                   |
|  Answer(Audio:Opus,a=sendonly)    |
|<----------------------------------|
|                                   |
|                                   |
|      One-way Opus Audio           |
|...................................|
|                                   |
|
```

SDP Contents	RFC#/Notes
v=0	[RFC4566]
o=- 20518 0 IN IP4 0.0.0.0	[RFC4566] - Session Origin Information
s=-	[RFC4566]
t=0 0	[RFC4566]
a=msid-semantic:WMS ma	[I-D.ietf-mmusic-msid]
a=group:BUNDLE audio	[I-D.ietf-mmusic-sdp-bundle-negotiation]
m=audio 54609 UDP/TLS/RTP/SAVPF 109	[RFC4566]
c=IN IP4 24.23.204.141	[RFC4566]
a=mid:audio	[RFC5888]
a=msid:ma ta	Identifies RTCMediaStream ID (ma) and RTCMediaStreamTrack ID (ta)
a=rtcp-mux	[RFC5761] - Alice can perform RTP/RTCP Muxing
a=rtcp:54609 IN IP4 24.23.204.141	[RFC3605] - Port for RTCP data
a=rtpmap:109 opus/48000/2	[I-D.ietf-payload-rtp-opus] - Opus Codec 48khz, 2 channels
a=extmap:1 urn:ietf:params:rtp-hdrext:ssrc-audio-level	[RFC6464]
a=ptime:20	[I-D.ietf-payload-rtp-opus] - Opus packetization of 20ms
a=sendrecv	[RFC3264] - Alice can send and recv audio

a=setup:actpass	[RFC4145] - Alice can perform DTLS before Answer arrives
a=ice-ufrag:074c6550	[RFC5245] - ICE user fragment
a=ice-pwd:a28a397a4c3f31747d1ee3474af08a068	[RFC5245] - ICE password
a=fingerprint:sha-1 99:41:49:83:4a:97:0e:1f:ef:6d:f7:c9:c7:70:9d:1f:66:79:a8:07	[RFC5245] - DTLS Fingerprint for SRTP
a=candidate:0 1 UDP 2113667327 192.168.1.4 54609 typ host	[RFC5245]
a=candidate:0 2 UDP 2113667327 192.168.1.4 54609 typ host	[RFC5245]
a=candidate:1 1 UDP 1685987071 24.23.204.141 54609 typ srflx raddr 192.168.1.4 rport 54609	[RFC5245]
a=candidate:1 2 UDP 1685987071 24.23.204.141 54609 typ srflx raddr 192.168.1.4 rport 54609	[RFC5245]
a=rtcp-fb:109 nack	[RFC5104] - Indicates NACK RTCP feedback support
a=ssrc:3229706345 cname:Q/NWs1ao1HmN4Xa5	[RFC5576]
a=rtcp-rsize	[RFC5506]
a=ice-options:trickle	[I-D.ietf-mmusic-trickle-ice]

Table 7: 5.2.4 SDP Offer

SDP Contents	RFC#/Notes
v=0	[RFC4566]
o=- 16833 0 IN IP4 0.0.0.0	[RFC4566] - Session Origin Information
s=-	[RFC4566]
t=0 0	[RFC4566]
a=msid-semantic:WMS ma	[I-D.ietf-mmusic-msid]
a=group:BUNDLE audio	[I-D.ietf-mmusic-sdp-bundle-negotiation]
m=audio 49203 UDP/TLS/RTP/SAVPF 109	[RFC4566]
c=IN IP4 98.248.92.77	[RFC4566]
a=mid:audio	[RFC5888]
a=msid:ma ta	Identifies RTCMediaStream ID (ma) and RTCMediaStreamTrack ID (ta)
a=rtpmap:109 opus/48000/2	[I-D.ietf-payload-rtp-opus] - Bob accepts Opus Codec
a=extmap:1 urn:ietf:params:rtp-hdrext:ssrc-audio-level	[RFC6464]
a=ptime:20	[I-D.ietf-payload-rtp-opus]
a=sendonly	[RFC3264] - Bob puts call On Hold
a=setup:active	[RFC4145] - Bob carries out DTLS Handshake in parallel
a=rtcp-mux	[RFC5761] - Bob can perform RTP/RTCP Muxing
a=ice-ufrag:c300d85b	[RFC5245] - ICE username frag
a=ice-pwd:de4e99bd291c325921d5d47efbabd9a2	[RFC5245] - ICE password
a=fingerprint:sha-1 99:41:49:83:4a:97:0e:1f:ef:6d:f7:c9:c7:70:9d: 1f:66:79:a8:07	[RFC5245] - DTLS Fingerprint for SRTP
a=candidate:0 1 UDP 2122194687 192.168.1.7 49203 typ host	[RFC5245]
a=candidate:1 1 UDP 1685987071 98.248.92.77 49203 typ srflx raddr 192.168.1.7 rport 49203	[RFC5245]
a=ssrc:1366781083 cname:EocUG1f0fcg/yvY7	[RFC5576]
a=rtcp-rsize	[RFC5506]
a=ice-options:trickle	[I-D.ietf-mmusic-trickle-ice]

Table 8: 5.2.4 SDP Answer

5.2.5. Audio with DTMF Session

In this example, Alice wishes to establish two separate audio streams, one for normal audio and the other for telephone-events. Alice offers first audio stream with three codecs and the other with [RFC2833] tones (for DTMF). Bob accepts both the audio streams by choosing Opus as the audio codec and telephone-event for the other stream.

Audio Session with DTMF

```
Alice                                                Bob
|                                                     |
|                                                     |
|                                                     |
|  Offer(Audio:Opus,PCMU,PCMA Audio:telephone-event)|
|----------------------------------------------------->|
|                                                     |
|                                                     |
|    Answer(Audio:Opus, Audio:telephone-event)        |
|<-----------------------------------------------------|
|                                                     |
|                                                     |
|   Opus audio stream and telephone-event stream      |
|.....................................................|
|                                                     |
```

SDP Contents	RFC#/Notes
v=0	[RFC4566]
o=- 20518 0 IN IP4 0.0.0.0	[RFC4566] - Session Origin Information
s=-	[RFC4566]
t=0 0	[RFC4566]
a=msid-semantic:WMS ma	[I-D.ietf-mmusic-msid]
a=group:BUNDLE audio dtmf	[I-D.ietf-mmusic-sdp-bundle-negotiation]
m=audio 54609 UDP/TLS/RTP/SAVPF 109 0 8	[RFC4566]
c=IN IP4 24.23.204.141	[RFC4566]
a=mid:audio	[RFC5888]
a=msid:ma ta	Identifies RTCMediaStream ID (ma) and RTCMediaStreamTrack ID (ta)
a=rtcp:54609 IN IP4 24.23.204.141	[RFC3605] - Port for RTCP data

a=rtcp-mux	[RFC5761] - Alice can perform RTP/RTCP Muxing
a=rtpmap:109 opus/48000/2	[I-D.ietf-payload-rtp-opus] - Opus Codec 48khz, 2 channels
a=ptime:20	[I-D.ietf-payload-rtp-opus] - Opus packetization of 20ms
a=rtpmap:0 PCMU/8000	[RFC3551] PCMU Audio Codec
a=rtpmap:8 PCMA/8000	[RFC3551] PCMA Audio Codec
a=extmap:1 urn:ietf:params:rtp-hdrext:ssrc-audio-level	[RFC6464]
a=sendrecv	[RFC3264] - Alice can send and recv audio
a=setup:actpass	[RFC4145] - Alice can perform DTLS before Answer arrives
a=ice-ufrag:074c6550	[RFC5245] - ICE user fragment
a=ice-pwd:a28a397a4c3f31747d1ee3474af08a068	[RFC5245] - ICE password parameter
a=fingerprint:sha-1 99:41:49:83:4a:97:0e:1f:ef:6d:f7:c9:c7:70:9d:1f:66:79:a8:07	[RFC5245] - DTLS Fingerprint for SRTP
a=candidate:0 1 UDP 2122194687 192.168.1.4 54609 typ host	[RFC5245]
a=candidate:0 2 UDP 2122194687 192.168.1.4 54609 typ host	[RFC5245]
a=candidate:1 1 UDP 1685987071 24.23.204.141 54609 typ srflx raddr 192.168.1.4 rport 54609	[RFC5245]
a=candidate:1 2 UDP 1685987071 24.23.204.141 54609 typ srflx raddr 192.168.1.4 rport 54609	[RFC5245]
a=rtcp-fb:109 nack	[RFC5104] - Indicates NACK RTCP feedback support
a=ssrc:3229706345 cname:Q/NWs1ao1HmN4Xa5	[RFC5576]
a=rtcp-rsize	[RFC5506]
a=ice-options:trickle	[I-D.ietf-mmusic-trickle-ice]
m=audio 54609 UDP/TLS/RTP/SAVPF 126	[RFC4566]
c=IN IP4 24.23.204.141	[RFC4566]
a=mid:dtmf	[RFC5888]
a=msid:ma tb	Identifies RTCMediaStream ID (ma) and RTCMediaStreamTrack ID (tb)
a=rtcp-mux	[RFC5761]
a=rtcp:54609 IN IP4 24.23.204.141	[RFC3605] - Port for RTCP data
a=rtpmap:126 telephone-event/8000	[RFC2833]

SDP Contents	RFC#/Notes
a=sendonly	[RFC3264] - Alice can send DTMF Events
a=setup:actpass	[RFC4145] - Alice can perform DTLS before Answer arrives
a=ice-ufrag:074c6550	[RFC5245] - ICE user fragment
a=ice-pwd:a28a397a4c3f31747d1ee3474af08a068	[RFC5245] - ICE password parameter
a=fingerprint:sha-1 99:41:49:83:4a:97:0e:1f:ef:6d:f7:c9:c7:70:9d:1f:66:79:a8:07	[RFC5245] - DTLS Fingerprint for SRTP
a=candidate:0 1 UDP 2122194687 192.168.1.4 54609 typ host	[RFC5245]
a=candidate:0 2 UDP 2122194687 192.168.1.4 54609 typ host	[RFC5245]
a=candidate:1 1 UDP 1685987071 24.23.204.141 54609 typ srflx raddr 192.168.1.4 rport 54609	[RFC5245]
a=candidate:1 2 UDP 1685987071 24.23.204.141 54609 typ srflx raddr 192.168.1.4 rport 54609	[RFC5245]
a=rtcp-fb:109 nack	[RFC5104] - Indicates NACK RTCP feedback support
a=ssrc:9032206345 cname:L/N9lk1ao1HmN4Xa5	[RFC5576]
a=rtcp-rsize	[RFC5506]
a=ice-options:trickle	[I-D.ietf-mmusic-trickle-ice]

Table 9: 5.2.5 SDP Offer

SDP Contents	RFC#/Notes
v=0	[RFC4566]
o=- 16833 0 IN IP4 0.0.0.0	[RFC4566] - Session Origin Information
s=-	[RFC4566]
t=0 0	[RFC4566]
a=msid-semantic:WMS ma	[I-D.ietf-mmusic-msid]
a=group:BUNDLE audio dtmf	[I-D.ietf-mmusic-sdp-bundle-negotiation]
m=audio 49203 UDP/TLS/RTP/SAVPF 109	[RFC4566]
c=IN IP4 98.248.92.77	[RFC4566]
a=mid:audio	[RFC5888]
a=msid:ma ta	Identifies RTCMediaStream ID (ma) and RTCMediaStreamTrack ID (ta)

```
| a=rtpmap:109 opus/48000/2        | [I-D.ietf-payload-rtp-opus] -  |
|                                  | Bob accepts Opus Codec         |
| a=extmap:1 urn:ietf:params:rtp-  | [RFC6464]                      |
| hdrext:ssrc-audio-level          |                                |
| a=ptime:20                       | [I-D.ietf-payload-rtp-opus]    |
| a=sendrecv                       | [RFC3264] - Bob can send and   |
|                                  | receive Opus audio             |
| a=setup:active                   | [RFC4145] - Bob carries out    |
|                                  | DTLS Handshake in parallel     |
| a=rtcp-mux                       | [RFC5761] - Bob can perform    |
|                                  | RTP/RTCP Muxing on port 49203  |
| a=ice-ufrag:c300d85b             | [RFC5245] -  ICE username frag |
| a=ice-pwd:de4e99bd291c325921d5d4 | [RFC5245] -  ICE password      |
| 7efbabd9a2                       |                                |
| a=fingerprint:sha-1 99:41:49:83: | [RFC5245] -  Fingerprint for   |
| 4a:97:0e:1f:ef:6d:f7:c9:c7:70:   | SRTP                           |
| 9d:1f:66:79:a8:07                |                                |
| a=candidate:0 1 UDP 2122194687   | [RFC5245]                      |
| 192.168.1.7 49203 typ host       |                                |
| a=candidate:1 1 UDP 1685987071   | [RFC5245]                      |
| 98.248.92.77 49203 typ srflx     |                                |
| raddr 192.168.1.7 rport 49203    |                                |
| a=ssrc:0634322975                | [RFC5576]                      |
| cname:Q/o1HmN4XNWs1aa5           |                                |
| a=rtcp-rsize                     | [RFC5506] - Alice intends to   |
|                                  | use reduced size RTCP for this |
|                                  | session                        |
| a=ice-options:trickle            | [I-D.ietf-mmusic-trickle-ice]  |
| m=audio 49203 UDP/TLS/RTP/SAVPF  | [RFC4566]                      |
| 126                              |                                |
| c=IN IP4 98.248.92.77            | [RFC4566]                      |
| a=mid:dtmf                       | [RFC5888]                      |
| a=msid:ma tb                     | Identifies RTCMediaStream ID   |
|                                  | (ma) and RTCMediaStreamTrack   |
|                                  | ID (tb)                        |
| a=rtpmap:126 telephone-          | [RFC2833]                      |
| event/8000                       |                                |
| a=recvonly                       | [RFC3264] - Alice can receive  |
|                                  | DTMF events                    |
| a=setup:active                   | [RFC4145] - Bob carries out    |
|                                  | DTLS Handshake in parallel     |
| a=rtcp-mux                       | [RFC5761] - Alice can perform  |
|                                  | RTP/RTCP Muxing on port 54690  |
| a=ice-ufrag:c300d85b             | [RFC5245] -  ICE username frag |
| a=ice-pwd:de4e99bd291c325921d5d4 | [RFC5245] -  ICE password      |
| 7efbabd9a2                       |                                |
| a=fingerprint:sha-1 99:41:49:83: | [RFC5245] -  Fingerprint for   |
| 4a:97:0e:1f:ef:6d:f7:c9:c7:70:9d | SRTP                           |
```

```
| : 1f:66:79:a8:07                 |                                |
| a=candidate:0 1 UDP 2122194687   | [RFC5245]                      |
| 192.168.1.7 49203 typ host       |                                |
| a=candidate:1 1 UDP 1685987071   | [RFC5245]                      |
| 98.248.92.77 49203 typ srflx     |                                |
| raddr 192.168.1.7 rport 49203    |                                |
| a=ssrc:6345903220                | [RFC5576]                      |
| cname:L/k1aN9lo1HmN4Xa5          |                                |
| a=rtcp-rsize                     | [RFC5506] - Alice intends to   |
|                                  | use reduced size RTCP for this |
|                                  | session                        |
| a=ice-options:trickle            | [I-D.ietf-mmusic-trickle-ice]  |
+----------------------------------+--------------------------------+
```

Table 10: 5.2.5 SDP Answer

5.2.6. One Way Audio/Video Session - Document Camera

In this scenario Alice and Bob engage in a 1 way audio and video session with Bob receiving Alice’s audio and her presentation slides as video stream.

One Way Audio & Video Session - Document Camera

```
Alice                                                  Bob
|                                                       |
|                                                       |
|                                                       |
|   Alice Offers sendonly audio and video streams.      |
|  The video stream corresponds to her presentation     |
|                                                       |
|           Offer(Audio:Opus, Video: VP8)               |
|------------------------------------------------------>|
|                                                       |
|                                                       |
|            (Audio:Opus, Video: VP8)                   |
|<------------------------------------------------------|
|                                                       |
|                                                       |
|          One-way Opus Audio, VP8 Video                |
|.......................................................|
|   Bob can hear Alice and see her presentation slides.|
|                                                       |
|                                                       |
```

SDP Contents	RFC#/Notes
v=0	[RFC4566]
o=- 20519 0 IN IP4 0.0.0.0	[RFC4566]
s=-	[RFC4566]
t=0 0	[RFC4566]
a=msid-semantic:WMS ma	[I-D.ietf-mmusic-msid]
a=group:BUNDLE audio video	[I-D.ietf-mmusic-sdp-bundle-negotiation]
m=audio 54609 UDP/TLS/RTP/SAVPF 109	[RFC4566]
c=IN IP4 24.23.204.141	[RFC4566]
a=mid:audio	[RFC5888]
a=msid:ma ta	Identifies RTCMediaStream ID (ma) and RTCMediaStreamTrack ID (ta)
a=rtcp-mux	[RFC5761]
a=rtcp:54609 IN IP4 24.23.204.141	[RFC3605] - Port for RTCP data
a=rtpmap:109 opus/48000/2	[I-D.ietf-payload-rtp-opus]
a=extmap:1 urn:ietf:params:rtp-hdrext:ssrc-audio-level	[RFC6464]
a=ptime:20	[I-D.ietf-payload-rtp-opus]
a=sendonly	[RFC3264] - Send only audio stream
a=setup:actpass	[RFC4145] - Alice can perform DTLS before Answer arrives
a=ice-ufrag:074c6550	[RFC5245]
a=ice-pwd:a28a397a4c3f31747d1ee3474af08a068	[RFC5245]
a=fingerprint:sha-1 99:41:49:83:4a:97:0e:1f:ef:6d:f7:c9:c7:70:9d:1f:66:79:a8:07	[RFC5245]
a=candidate:0 1 UDP 2113667327 192.168.1.4 54609 typ host	[RFC5245]
a=candidate:0 2 UDP 2113667327 192.168.1.4 54609 typ host	[RFC5245]
a=candidate:1 1 UDP 1685987071 24.23.204.141 54609 typ srflx raddr 192.168.1.4 rport 54609	[RFC5245]
a=candidate:1 2 UDP 1685987071 24.23.204.141 54609 typ srflx raddr 192.168.1.4 rport 54609	[RFC5245]
a=rtcp-fb:109 nack	[RFC5104]
a=ssrc:6345903220 cname:L/k1aN9lo1HmN4Xa5	[RFC5576]
a=rtcp-rsize	[RFC5506]

SDP Contents	RFC#/Notes
a=ice-options:trickle	[I-D.ietf-mmusic-trickle-ice]
m=video 54609 UDP/TLS/RTP/SAVPF 120	[RFC4566]
c=IN IP4 24.23.204.141	[RFC4566]
a=mid:video	[RFC5888]
a=msid:ma tb	Identifies RTCMediaStream ID (ma) and RTCMediaStreamTrack ID (tb)
a=rtcp-mux	[RFC5761]
a=rtcp:54609 IN IP4 24.23.204.141	[RFC3605] - Port for RTCP data
a=rtpmap:120 VP8/90000	[I-D.ietf-payload-vp8]
a=content:slides	[RFC4796] -Alice’s presentation video stream
a=sendonly	[RFC3264] - Send only video stream
a=setup:actpass	[RFC4145] - Alice can perform DTLS before Answer arrives
a=ice-ufrag:074c6550	[RFC5245]
a=ice-pwd:a28a397a4c3f31747d1ee3474af08a068	[RFC5245]
a=fingerprint:sha-1 99:41:49:83:4a:97:0e:1f:ef:6d:f7:c9:c7:70:9d:1f:66:79:a8:07	[RFC5245]
a=candidate:0 1 UDP 2113667327 192.168.1.4 54609 typ host	[RFC5245]
a=candidate:0 2 UDP 2113667327 192.168.1.4 54609 typ host	[RFC5245]
a=candidate:1 1 UDP 1685987071 24.23.204.141 54609 typ srflx raddr 192.168.1.4 rport 54609	[RFC5245]
a=candidate:1 2 UDP 1685987071 24.23.204.141 54609 typ srflx raddr 192.168.1.4 rport 54609	[RFC5245]
a=rtcp-fb:120 nack	[RFC5104]
a=rtcp-fb:120 nack pli	[RFC5104]
a=rtcp-fb:120 ccm fir	[RFC5104]
a=ssrc:3429951804 cname:Q/NWs1ao1HmN4Xa5	[RFC5576]
a=rtcp-rsize	[RFC5506]
a=ice-options:trickle	[I-D.ietf-mmusic-trickle-ice]

Table 11: 5.2.6 SDP Offer

SDP Contents	RFC#/Notes

```
| v=0                                | [RFC4566]                      |
| o=- 16833 0 IN IP4 0.0.0.0         | [RFC4566]                      |
| s=-                                | [RFC4566]                      |
| t=0 0                              | [RFC4566]                      |
| a=msid-semantic:WMS ma             | [I-D.ietf-mmusic-msid]         |
| a=group:BUNDLE audio video         | [I-D.ietf-mmusic-sdp-bundle-ne |
|                                    | gotiation]                     |
| m=audio 49203 UDP/TLS/RTP/SAVPF    | [RFC4566]                      |
| 109                                |                                |
| c=IN IP4 98.248.92.77              | [RFC4566]                      |
| a=mid:audio                        | [RFC5888]                      |
| a=msid:ma ta                       | Identifies RTCMediaStream ID   |
|                                    | (ma) and RTCMediaStreamTrack   |
|                                    | ID (ta)                        |
| a=rtpmap:109 opus/48000/2          | [I-D.ietf-payload-rtp-opus]    |
| a=extmap:1 urn:ietf:params:rtp-    | [RFC6464]                      |
| hdrext:ssrc-audio-level            |                                |
| a=ptime:20                         | [I-D.ietf-payload-rtp-opus]    |
| a=recvonly                         | [RFC3264] - Receive only audio |
|                                    | stream                         |
| a=setup:active                     | [RFC4145] - Bob carries out    |
|                                    | DTLS Handshake in parallel     |
| a=rtcp-mux                         | [RFC5761]                      |
| a=ice-ufrag:c300d85b               | [RFC5245]                      |
| a=ice-pwd:de4e99bd291c325921d5d4   | [RFC5245]                      |
| 7efbabd9a2                         |                                |
| a=fingerprint:sha-1 99:41:49:83:   | [RFC5245]                      |
| 4a:97:0e:1f:ef:6d:f7:c9:c7:70:9d   |                                |
| : 1f:66:79:a8:07                   |                                |
| a=candidate:0 1 UDP 2122194687     | [RFC5245]                      |
| 192.168.1.7 49203 typ host         |                                |
| a=candidate:1 1 UDP 1685987071     | [RFC5245]                      |
| 98.248.92.77 49203 typ srflx       |                                |
| raddr 192.168.1.7 rport 49203      |                                |
| a=ssrc:9513429804                  | [RFC5576]                      |
| cname:Q/o1HmNWs1aN4Xa5             |                                |
| a=rtcp-rsize                       | [RFC5506]                      |
| a=ice-options:trickle              | [I-D.ietf-mmusic-trickle-ice]  |
| m=video 49203 UDP/TLS/RTP/SAVPF    | [RFC4566]                      |
| 120                                |                                |
| c=IN IP4 98.248.92.77              | [RFC4566]                      |
| a=mid:video                        | [RFC5888]                      |
| a=msid:ma tb                       | Identifies RTCMediaStream ID   |
|                                    | (ma) and RTCMediaStreamTrack   |
|                                    | ID (tb)                        |
| a=rtpmap:120 VP8/90000             | [I-D.ietf-payload-vp8]         |
| a=content:slides                   | [RFC4796]                      |
| a=recvonly                         | [RFC3264] - Receive Only       |
```

	Alice’s presentation stream
a=setup:active	[RFC4145] - Bob carries out DTLS Handshake in parallel
a=rtcp-mux	[RFC5761]
a=ice-ufrag:c300d85b	[RFC5245]
a=ice-pwd:de4e99bd291c325921d5d4 7efbabd9a2	[RFC5245]
a=fingerprint:sha-1 99:41:49:83: 4a:97:0e:1f:ef:6d:f7:c9:c7:70: 9d:1f:66:79:a8:07	[RFC5245]
a=candidate:0 1 UDP 2122194687 192.168.1.7 49203 typ host	[RFC5245]
a=candidate:1 1 UDP 1685987071 98.248.92.77 49203 typ srflx raddr 192.168.1.7 rport 49203	[RFC5245]
a=ssrc:1366781083 cname:EocUG1f0fcg/yvY7	[RFC5576]
a=rtcp-rsize	[RFC5506]
a=ice-options:trickle	[I-D.ietf-mmusic-trickle-ice]

Table 12: 5.2.6 SDP Answer

5.2.7. Audio, Video Session with BUNDLE Support Unknown

In this example, since Alice is unsure of the Bob’s support of the BUNDLE framework, following 3 step procedures are performed in order to negotiate and setup a BUNDLE Address for the session

- An SDP Offer, in which the Alice assigns unique addresses to each "m=" line in the BUNDLE group, and requests the Answerer to select the Offerer’s BUNDLE address.

- An SDP Answer, in which the Bob indicates its support for BUNDLE, and assigns its own BUNDLE address for the BUNDLED m= lines.

- A subsequent SDP Offer from Alice, which is used to perform BUNDLE Address Synchronization (BAS).

Once the Offer/Answer exchange completes, both Alice and Bob each end up using single RTP Session for both the Media Streams.

Two-Way Secure Audio,Video with BUNDLE support unknown

```
   Alice                                                   Bob
   |                                                        |
   |                                                        |
   |   Alice offers BUNDLE support with unique address      |
   |        for the audio and video m-line                  |
   |                                                        |
   |                                                        |
   |          Offer(Audio:Opus Video:VP8)                   |
   |------------------------------------------------------>|
   |                                                        |Bob
   |                                                        |supports
   |                                                        |BUNDLE
   |          Answer(Audio:Opus Video:VP8)                  |
   |<------------------------------------------------------|
   |                                                        |
   |                                                        |Bob uses
   |                                                        |identical
   |                                                        |addresses
   |                                                        |
   |  Updated Offer for Bundle Address Synchronization.     |
   |------------------------------------------------------>|
   |                                                        |
   |                                                        |
   |       Answer (Bob accepts the updated offer)           |
   |<------------------------------------------------------|
   |                                                        |
   |                                                        |
   |    2 Way Call with Audio and Video Multiplexed         |
   |.......................................................|
   |                                                        |


   +----------------------------------+--------------------------------+
   | SDP Contents                     | RFC#/Notes                     |
   +----------------------------------+--------------------------------+
   | v=0                              | [RFC4566]                      |
   | o=- 20518 0 IN IP4 0.0.0.0       | [RFC4566]                      |
   | s=-                              | [RFC4566]                      |
   | t=0 0                            | [RFC4566]                      |
   | a=msid-semantic:WMS ma           | [I-D.ietf-mmusic-msid]         |
   | a=group:BUNDLE audio video       | [I-D.ietf-mmusic-sdp-bundle-ne |
   |                                  | gotiation] Alice supports      |
   |                                  | grouping of m=lines under      |
   |                                  | BUNDLE semantics               |
   | m=audio 54609 UDP/TLS/RTP/SAVPF  | [RFC4566]                      |
   | 109                              |                                |
```

```
| c=IN IP4 24.23.204.141           | [RFC4566]                      |
| a=mid:audio                      | [RFC5888] Audio m=line part of |
|                                  | BUNDLE group with a unique     |
|                                  | port number                    |
| a=msid:ma ta                     | Identifies RTCMediaStream ID   |
|                                  | (ma) and RTCMediaStreamTrack   |
|                                  | ID (ta)                        |
| a=rtcp-mux                       | [RFC5761]                      |
| a=rtcp:54609 IN IP4              | [RFC3605]                      |
| 24.23.204.141                    |                                |
| a=rtpmap:109 opus/48000/2        | [I-D.ietf-payload-rtp-opus]    |
| a=extmap:1 urn:ietf:params:rtp-  | [RFC6464]                      |
| hdrext:ssrc-audio-level          |                                |
| a=ptime:20                       | [I-D.ietf-payload-rtp-opus]    |
| a=sendrecv                       | [RFC3264]                      |
| a=setup:actpass                  | [RFC4145] - Alice can perform  |
|                                  | DTLS before Answer arrives     |
| a=ssrc:11111                     | [RFC5576]                      |
| cname:EocUG1f0fcg/yvY7           |                                |
| a=ice-ufrag:074c6550             | [RFC5245]                      |
| a=ice-pwd:a28a397a4c3f31747d1ee3 | [RFC5245]                      |
| 474af08a068                      |                                |
| a=fingerprint:sha-1 99:41:49:83: | [RFC5245]                      |
| 4a:97:0e:1f:ef:6d:f7:c9:c7:70:   |                                |
| 9d:1f:66:79:a8:07                |                                |
| a=candidate:0 1 UDP  2122194687  | [RFC5245]                      |
| 192.168.1.4 54609 typ host       |                                |
| a=candidate:0 2 UDP 2122194687   | [RFC5245]                      |
| 192.168.1.4 54609 typ host       |                                |
| a=candidate:1 1 UDP  1685987071  | [RFC5245]                      |
| 24.23.204.141 54609 typ srflx    |                                |
| raddr 192.168.1.4 rport 54609    |                                |
| a=candidate:1 2 UDP  1685987071  | [RFC5245]                      |
| 24.23.204.141 54609 typ srflx    |                                |
| raddr 192.168.1.4 rport 54609    |                                |
| a=rtcp-fb:109 nack               | [RFC5104]                      |
| a=rtcp-rsize                     | [RFC5506]                      |
| a=ice-options:trickle            | [I-D.ietf-mmusic-trickle-ice]  |
| m=video 62537 UDP/TLS/RTP/SAVPF  | [RFC4566]                      |
| 120                              |                                |
| c=IN IP4 24.23.204.141           | [RFC4566]                      |
| a=mid:video                      | [RFC5888] Video m=line part of |
|                                  | the Bundle group with a unique |
|                                  | port number                    |
| a=msid:ma tb                     | Identifies RTCMediaStream ID   |
|                                  | (ma) and RTCMediaStreamTrack   |
|                                  | ID (tb)                        |
| a=rtcp-mux                       | [RFC5761]                      |
```

a=rtcp:62537 IN IP4 24.23.204.141	[RFC3605] - Port for RTCP data
a=rtpmap:120 VP8/90000	[I-D.ietf-payload-vp8]
a=sendrecv	[RFC3264]
a=setup:actpass	[RFC4145] - Alice can perform DTLS before Answer arrives
a=ssrc:22222 cname:Q/NWs1ao1HmN4Xa5	[RFC5576]
a=ice-ufrag:6550074c	[RFC5245]
a=ice-pwd:74af08a068a28a397a4c3f 31747d1ee34	[RFC5245]
a=fingerprint:sha-1 1f:ef:6d:f7: c9:c7:70:9d:1f:66:99:41:49:83: 4a:97:0e79:a8:07	[RFC5245]
a=candidate:0 1 UDP 2122194687 192.168.1.4 62537 typ host	[RFC5245]
a=candidate:0 2 2122194687 192.168.1.4 62537 typ host	[RFC5245]
a=candidate:1 1 UDP 1685987071 24.23.204.141 62537 typ srflx raddr 192.168.1.4 rport 62537	[RFC5245]
a=candidate:1 2 UDP 1685987071 24.23.204.141 62537 typ srflx raddr 192.168.1.4 rport 62537	[RFC5245]
a=rtcp-fb:120 nack	[RFC5104]
a=rtcp-fb:120 nack pli	[RFC5104]
a=rtcp-fb:120 ccm fir	[RFC5104]
a=rtcp-rsize	[RFC5506]
a=ice-options:trickle	[I-D.ietf-mmusic-trickle-ice]

Table 13: 5.2.7 SDP Offer w/BUNDLE

SDP Contents	RFC#/Notes
v=0	[RFC4566]
o=- 16833 0 IN IP4 0.0.0.0	[RFC4566]
s=-	[RFC4566]
t=0 0	[RFC4566]
a=msid-semantic:WMS ma	[I-D.ietf-mmusic-msid]
a=group:BUNDLE audio video	[I-D.ietf-mmusic-sdp-bundle-negotiation] Bob supports BUNDLE semantics.
m=audio 49203 UDP/TLS/RTP/SAVPF 109	[RFC4566]
c=IN IP4 98.248.92.77	[RFC4566]
a=msid:ma ta	Identifies RTCMediaStream ID

```
   |                                  | (ma) and RTCMediaStreamTrack   |
   |                                  | ID (ta)                        |
   | a=mid:audio                      | [RFC5888] Audio m=line part of |
   |                                  | the BUNDLE group               |
   | a=rtpmap:109 opus/48000/2        | [I-D.ietf-payload-rtp-opus]    |
   | a=ptime:20                       | [I-D.ietf-payload-rtp-opus]    |
   | a=extmap:1 urn:ietf:params:rtp-  | [RFC6464]                      |
   | hdrext:ssrc-audio-level          |                                |
   | a=sendrecv                       | [RFC3264]                      |
   | a=setup:active                   | [RFC4145] - Bob carries out    |
   |                                  | DTLS Handshake in parallel     |
   | a=rtcp-fb:109 nack               | [RFC5104]                      |
   | a=rtcp-mux                       | [RFC5761]                      |
   | a=ssrc:33333                     | [RFC5576]                      |
   | cname:Q/1HmN4Xa5NWs1ao           |                                |
   | a=ice-ufrag:c300d85b             | [RFC5245]                      |
   | a=ice-pwd:de4e99bd291c325921d5d4 | [RFC5245]                      |
   | 7efbabd9a2                       |                                |
   | a=fingerprint:sha-1 99:41:49:83: | [RFC5245]                      |
   | 4a:97:0e:1f:ef:6d:f7:c9:c7:70:   |                                |
   | 9d:1f:66:79:a8:07                |                                |
   | a=candidate:0 1 UDP 2122194687   | [RFC5245]                      |
   | 192.168.1.7 49203 typ host       |                                |
   | a=candidate:1 1 UDP 1685987071   | [RFC5245]                      |
   | 98.248.92.77 49203 typ srflx     |                                |
   | raddr 192.168.1.7 rport 49203    |                                |
   | a=rtcp-rsize                     | [RFC5506]                      |
   | a=ice-options:trickle            | [I-D.ietf-mmusic-trickle-ice]  |
   | m=video 49203 UDP/TLS/RTP/SAVPF  | [RFC4566]                      |
   | 120                              |                                |
   | c=IN IP4 98.248.92.77            | [RFC4566]                      |
   | a=mid:video                      | [RFC5888] Video m=line part of |
   |                                  | the BUNDLE group with the port |
   |                                  | from audio line repeated       |
   | a=msid:ma tb                     | Identifies RTCMediaStream ID   |
   |                                  | (ma) and RTCMediaStreamTrack   |
   |                                  | ID (tb)                        |
   | a=rtpmap:120 VP8/90000           | [I-D.ietf-payload-vp8]         |
   | a=sendrecv                       | [RFC3264]                      |
   | a=setup:active                   | [RFC4145] - Bob carries out    |
   |                                  | DTLS Handshake in parallel     |
   | a=rtcp-mux                       | [RFC5761]                      |
   | a=ssrc:44444                     | [RFC5576]                      |
   | cname:Q/2AqlmN4Xa5NWs            |                                |
   | a=ice-ufrag:85bc300d             | [RFC5245]                      |
   | a=ice-pwd:bd2de4e9991c325921d5d4 | [RFC5245]                      |
   | 7efbabd9a2                       |                                |
   | a=fingerprint:sha-1 41:49:83:4a: | [RFC5245]                      |
```

99:97:0e:1f:ef:6d:f7:c9:c7:70: 9d:1f:66:79:a8:07	
a=candidate:0 1 UDP 2122194687 192.168.1.7 49203 typ host	[RFC5245]
a=candidate:1 1 UDP 1685987071 98.248.92.77 49203 typ srflx raddr 192.168.1.7 rport 49203	[RFC5245]
a=rtcp-fb:120 nack	[RFC5104]
a=rtcp-fb:120 nack pli	[RFC5104]
a=rtcp-fb:120 ccm fir	[RFC5104]
a=rtcp-rsize	[RFC5506]
a=ice-options:trickle	[I-D.ietf-mmusic-trickle-ice]

Table 14: 5.2.7 SDP Answer w/BUNDLE

SDP Contents	RFC#/Notes
v=0	[RFC4566]
o=- 20518 0 IN IP4 0.0.0.0	[RFC4566]
s=-	[RFC4566]
t=0 0	[RFC4566]
a=msid-semantic:WMS ma	[I-D.ietf-mmusic-msid]
a=group:BUNDLE audio video	[I-D.ietf-mmusic-sdp-bundle-negotiation]
m=audio 54609 UDP/TLS/RTP/SAVPF 109	[RFC4566]
c=IN IP4 24.23.204.141	[RFC4566]
a=msid:ma ta	Identifies RTCMediaStream ID (ma) and RTCMediaStreamTrack ID (ta)
a=mid:audio	[RFC5888] - Port number finalized as Bundle Address.
a=rtcp-mux	[RFC5761]
a=rtcp:54609 IN IP4 24.23.204.141	[RFC3605]
a=rtpmap:109 opus/48000/2	[I-D.ietf-payload-rtp-opus]
a=extmap:1 urn:ietf:params:rtp-hdrext:ssrc-audio-level	[RFC6464]
a=ptime:20	[I-D.ietf-payload-rtp-opus]
a=sendrecv	[RFC3264]
a=setup:actpass	[RFC4145]
a=ssrc:11111 cname:EocUG1f0fcg/yvY7	[RFC5576]
a=ice-ufrag:074c6550	[RFC5245]
a=ice-pwd:a28a397a4c3f31747d1ee3474af08a068	[RFC5245]

a=fingerprint:sha-1 99:41:49:83: 4a:97:0e:1f:ef:6d:f7:c9:c7:70: 9d:1f:66:79:a8:07	[RFC5245]
a=candidate:0 1 UDP 2122194687 192.168.1.4 54609 typ host	[RFC5245]
a=candidate:1 1 UDP 1685987071 24.23.204.141 54609 typ srflx raddr 192.168.1.4 rport 54609	[RFC5245]
a=rtcp-fb:109 nack	[RFC5104]
a=rtcp-rsize	[RFC5506]
a=ice-options:trickle	[I-D.ietf-mmusic-trickle-ice]
m=video 54609 UDP/TLS/RTP/SAVPF 120	[RFC4566]
c=IN IP4 24.23.204.141	[RFC4566]
a=msid:ma tb	Identifies RTCMediaStream ID (ma) and RTCMediaStreamTrack ID (tb)
a=mid:video	[RFC5888]
a=rtpmap:120 VP8/90000	[I-D.ietf-payload-vp8]
a=sendrecv	[RFC3264]
a=setup:actpass	[RFC4145]
a=rtcp-mux	[RFC5761]
a=ssrc:22222 cname:Q/NWs1ao1HmN4Xa5	[RFC5576]
a=ice-ufrag:074c6550	[RFC5245]
a=ice-pwd:a28a397a4c3f31747d1ee3 474af08a068	[RFC5245]
a=fingerprint:sha-1 99:41:49:83: 4a:97:0e:1f:ef:6d:f7:c9:c7:70: 9d:1f:66:79:a8:07	[RFC5245]
a=candidate:0 1 UDP 2122194687 192.168.1.4 54609 typ host	[RFC5245]
a=candidate:1 1 UDP 1685987071 24.23.204.141 54609 typ srflx raddr 192.168.1.4 rport 54609	[RFC5245]
a=rtcp-fb:120 nack	[RFC5104]
a=rtcp-fb:120 nack pli	[RFC5104]
a=rtcp-fb:120 ccm fir	[RFC5104]
a=rtcp-rsize	[RFC5506]
a=ice-options:trickle	[I-D.ietf-mmusic-trickle-ice]

Table 15: 5.2.7 SDP Offer for BAS

5.2.8. Audio, Video and Data Session

This example shows SDP for negotiating a session with Audio, Video and data streams between Alice and Bob with BUNDLE support known.

```
       Audio,Video,Data with BUNDLE support known

   Alice                                        Bob
   |                                             |
   |                                             |
   |Alice indicates BUNDLE support with          |
   |identical address across all the m=lines     |
   |                                             |
   |                                             |
   |     Offer(Audio:Opus Video:VP8 Data)        |
   |--------------------------------------------->|
   |                                             |Bob does
   |                                             |the same
   |    Answer(Audio:Opus,Video:VP8 Data)        |
   |<---------------------------------------------|
   |                                             |
   |                                             |
   |                                             |
   |    Two-way Audio,Video, Data multiplexed    |
   |.............................................|
   |                                             |
   |                                             |


   +----------------------------------+-------------------------------+
   | SDP Contents                     | RFC#/Notes                    |
   +----------------------------------+-------------------------------+
   | v=0                              | [RFC4566]                     |
   | o=- 20518 0 IN IP4 0.0.0.0       | [RFC4566]                     |
   | s=-                              | [RFC4566]                     |
   | t=0 0                            | [RFC4566]                     |
   | a=msid-semantic:WMS ma           | [I-D.ietf-mmusic-msid]        |
   | a=group:BUNDLE audio video data  | [I-D.ietf-mmusic-sdp-bundle-ne |
   |                                  | gotiation]                    |
   | m=audio 54609 UDP/TLS/RTP/SAVPF  | [RFC4566]                     |
   | 109                              |                               |
   | c=IN IP4 24.23.204.141           | [RFC4566]                     |
   | a=msid:ma ta                     | Identifies RTCMediaStream ID  |
   |                                  | (ma) and RTCMediaStreamTrack  |
   |                                  | ID (ta)                       |
   | a=rtcp:54609 IN IP4              | [RFC3605]                     |
```

24.23.204.141	
a=mid:audio	[RFC5888]
a=rtpmap:109 opus/48000/2	[I-D.ietf-payload-rtp-opus]
a=extmap:1 urn:ietf:params:rtp-hdrext:ssrc-audio-level	[RFC6464]
a=ptime:20	[I-D.ietf-payload-rtp-opus]
a=sendrecv	[RFC3264]
a=setup:actpass	[RFC4145]
a=rtcp-mux	[RFC5761]
a=ice-ufrag:074c6550	[RFC5245]
a=ice-pwd:a28a397a4c3f31747d1ee3474af08a068	[RFC5245]
a=fingerprint:sha-1 99:41:49:83:4a:97:0e:1f:ef:6d:f7:c9:c7:70:9d:1f:66:79:a8:07	[RFC5245]
a=candidate:0 1 UDP 2122194687 192.168.1.4 54609 typ host	[RFC5245]
a=candidate:0 2 UDP 2122194687 192.168.1.4 54609 typ host	[RFC5245]
a=candidate:1 1 UDP 1685987071 24.23.204.141 54609 typ srflx raddr 192.168.1.4 rport 54609	[RFC5245]
a=candidate:1 2 UDP 1685987071 24.23.204.141 54609 typ srflx raddr 192.168.1.4 rport 54609	[RFC5245]
a=rtcp-fb:109 nack	[RFC5104]
a=ssrc:11111 cname:Q/NWs1ao1HmN4Xa5	[RFC5576]
a=rtcp-rsize	[RFC5506]
a=ice-options:trickle	[I-D.ietf-mmusic-trickle-ice]
m=video 54609 UDP/TLS/RTP/SAVPF 120	[RFC4566]
c=IN IP4 24.23.204.141	[RFC4566]
a=msid:ma tb	Identifies RTCMediaStream ID (ma) and RTCMediaStreamTrack ID (tb)
a=rtcp:54609 IN IP4 24.23.204.141	[RFC3605]
a=mid:video	[RFC5888]
a=rtpmap:120 VP8/90000	[I-D.ietf-payload-vp8]
a=sendrecv	[RFC3264]
a=setup:actpass	[RFC4145]
a=rtcp-mux	[RFC5761]
a=ice-ufrag:074c6550	[RFC5245]
a=ice-pwd:a28a397a4c3f31747d1ee3474af08a068	[RFC5245]
a=fingerprint:sha-1 99:41:49:83:4a:97:0e:1f:ef:6d:f7:c9:c7:70:	[RFC5245]

9d:1f:66:79:a8:07	
a=candidate:0 1 UDP 2122194687	[RFC5245]
192.168.1.4 54609 typ host	
a=candidate:0 2 UDP 2122194687	[RFC5245]
192.168.1.4 54609 typ host	
a=candidate:1 1 UDP 1685987071	[RFC5245]
24.23.204.141 54609 typ srflx	
raddr 192.168.1.4 rport 54609	
a=candidate:1 2 UDP 1685987071	[RFC5245]
24.23.204.141 54609 typ srflx	
raddr 192.168.1.4 rport 54609	
a=rtcp-fb:120 nack	[RFC5104]
a=rtcp-fb:120 nack pli	[RFC5104]
a=rtcp-fb:120 ccm fir	[RFC5104]
a=ssrc:22222	[RFC5576]
cname:Q/aoNWs11HmN4Xa5	
a=rtcp-rsize	[RFC5506]
a=ice-options:trickle	[I-D.ietf-mmusic-trickle-ice]
m=application 54609	[I-D.ietf-rtcweb-data-channel]
UDP/DTLS/SCTP 5000	
c=IN IP4 24.23.204.141	[RFC4566]
a=mid:data	[RFC5888]
a=sctpmap:5000 webrtc-	[I-D.ietf-mmusic-sctp-sdp]
DataChannel	
streams=1;label="channel 1";	
subprotocol="chat";	
a=sendrecv	[RFC3264]
a=setup:actpass	[RFC4145]
a=ice-ufrag:074c6550	[RFC5245]
a=ice-pwd:a28a397a4c3f31747d1ee3	[RFC5245]
474af08a068	
a=fingerprint:sha-1 99:41:49:83:	[RFC5245]
4a:97:0e:1f:ef:6d:f7:c9:c7:70:	
9d:1f:66:79:a8:07	
a=candidate:0 1 UDP 2122194687	[RFC5245]
192.168.1.4 54609 typ host	
a=candidate:1 1 UDP 1685987071	[RFC5245]
24.23.204.141 54609 typ srflx	
raddr 192.168.1.4 rport 54609	
a=ice-options:trickle	[I-D.ietf-mmusic-trickle-ice]

Table 16: 5.2.8 SDP Offer

SDP Contents	RFC#/Notes
v=0	[RFC4566]

```
| o=-  16833 0 IN IP4 0.0.0.0      | [RFC4566] - Session Origin     |
|                                  | Information                    |
| s=-                              | [RFC4566]                      |
| t=0 0                            | [RFC4566]                      |
| a=msid-semantic:WMS ma           | [I-D.ietf-mmusic-msid]         |
| a=group:BUNDLE audio video data  | [I-D.ietf-mmusic-sdp-bundle-ne |
|                                  | gotiation]                     |
| a=ice-options:trickle            | [I-D.ietf-mmusic-trickle-ice]  |
|                                  | Bob’s trickle support support  |
|                                  | is indicated at the session    |
|                                  | level                          |
| m=audio 49203 UDP/TLS/RTP/SAVPF  | [RFC4566]                      |
| 109                              |                                |
| c=IN IP4 98.248.92.77            | [RFC4566]                      |
| a=msid:ma ta                     | Identifies RTCMediaStream ID   |
|                                  | (ma) and RTCMediaStreamTrack   |
|                                  | ID (ta)                        |
| a=mid:audio                      | [RFC5888]                      |
| a=rtpmap:109 opus/48000/2        | [I-D.ietf-payload-rtp-opus]    |
| a=extmap:1 urn:ietf:params:rtp-  | [RFC6464]                      |
| hdrext:ssrc-audio-level          |                                |
| a=ptime:20                       | [I-D.ietf-payload-rtp-opus]    |
| a=sendrecv                       | [RFC3264]                      |
| a=setup:active                   | [RFC4145]                      |
| a=rtcp-mux                       | [RFC5761]                      |
| a=rtcp-fb:109 nack               | [RFC5104]                      |
| a=ice-ufrag:c300d85b             | [RFC5245]                      |
| a=ice-pwd:de4e99bd291c325921d5d4 | [RFC5245]                      |
| 7efbabd9a2                       |                                |
| a=fingerprint:sha-1 99:41:49:83: | [RFC5245]                      |
| 4a:97:0e:1f:ef:6d:f7:c9:c7:70:   |                                |
| 9d:1f:66:79:a8:07                |                                |
| a=candidate:0 1 UDP 2122194687   | [RFC5245]                      |
| 192.168.1.7 49203 typ host       |                                |
| a=candidate:1 1 UDP 1685987071   | [RFC5245]                      |
| 98.248.92.77 49203 typ srflx     |                                |
| raddr 192.168.1.7 rport 49203    |                                |
| a=ssrc:33333                     | [RFC5576]                      |
| cname:L/aoNWs11HmN4Xa5           |                                |
| a=rtcp-rsize                     | [RFC5506]                      |
| m=video 49203 UDP/TLS/RTP/SAVPF  | [RFC4566]                      |
| 120                              |                                |
| c=IN IP4 98.248.92.77            | [RFC4566]                      |
| a=msid:ma tb                     | Identifies RTCMediaStream ID   |
|                                  | (ma) and RTCMediaStreamTrack   |
|                                  | ID (tb)                        |
| a=mid:video                      | [RFC5888]                      |
| a=rtpmap:120 VP8/90000           | [I-D.ietf-payload-vp8]         |
```

a=sendrecv	[RFC3264]
a=setup:active	[RFC4145]
a=rtcp-mux	[RFC5761]
a=ice-ufrag:c300d85b	[RFC5245]
a=ice-pwd:de4e99bd291c325921d5d4 7efbabd9a2	[RFC5245]
a=fingerprint:sha-1 99:41:49:83: 4a:97:0e:1f:ef:6d:f7:c9:c7:70: 9d:1f:66:79:a8:07	[RFC5245]
a=candidate:0 1 UDP 2122194687 192.168.1.7 49203 typ host	[RFC5245]
a=candidate:1 1 UDP 1685987071 98.248.92.77 49203 typ srflx raddr 192.168.1.7 rport 49203	[RFC5245]
a=rtcp-fb:120 nack	[RFC5104]
a=rtcp-fb:120 nack pli	[RFC5104]
a=rtcp-fb:120 ccm fir	[RFC5104]
a=ssrc:44444 cname:EocUG1f0fcg/yvY7	[RFC5576]
a=rtcp-rsize	[RFC5506]
m=application 49203 UDP/DTLS/SCTP 5000	[I-D.ietf-mmusic-sctp-sdp]
c=IN IP4 98.248.92.771	[RFC4566]
a=mid:data	[RFC5888]
a=sctpmap:5000 webrtc-DataChannel streams=16;label="channel 1"; subprotocol="chat";	[I-D.ietf-mmusic-sctp-sdp]
a=setup:active	[RFC4145]
a=sendrecv	[RFC3264]
a=ice-ufrag:c300d85b	[RFC5245]
a=ice-pwd:de4e99bd291c325921d5d4 7efbabd9a2	[RFC5245]
a=fingerprint:sha-1 99:41:49:83: 4a:97:0e:1f:ef:6d:f7:c9:c7:70: 9d:1f:66:79:a8:07	[RFC5245]
a=candidate:0 1 UDP 2122194687 192.168.1.7 49203 typ host	[RFC5245]
a=candidate:1 1 UDP 1685987071 98.248.92.77 49203 typ srflx raddr 192.168.1.7 rport 49203	[RFC5245]

Table 17: 5.2.8 SDP Answer

5.2.9. Audio, Video Session with BUNDLE Unsupported

This use-case illustrates SDP Offer/Answer exchange where the far-end (Bob) either doesn’t support media bundling or doesn’t want to group m=lines over a single 5-tuple.

On successful Offer/Answer exchange, Alice and Bob each end up using unique 5-tuple for audio and video media streams respectively.

Two-Way Secure Audio,Video with BUNDLE Unsupported

```
Alice                                                   Bob
|                                                        |
|                                                        |
|     Alice offers BUNDLE support with unique address    |
|           for the audio and video m-line               |
|                                                        |
|                                                        |
|           Offer(Audio:Opus Video:VP8)                  |
|------------------------------------------------------>|
|                                                        |Bob
|                                                        |doesn’t
|                                                        |support
|                                                        |BUNDLE
|           Answer(Audio:Opus Video:VP8)                 |
|<------------------------------------------------------|
|                                                        |Bob uses
|                                                        |unique
|                                                        |addresses
|                                                        |across the
|                                                        |m=lines
|                                                        |
|2Way Call with Audio and Video on different 5-tuples    |
|.......................................................|
|                                                        |
|                                                        |
```

SDP Contents	RFC#/Notes
v=0	[RFC4566]
o=- 20518 0 IN IP4 0.0.0.0	[RFC4566]
s=-	[RFC4566]
t=0 0	[RFC4566]
a=msid-semantic:WMS ma	[I-D.ietf-mmusic-msid]

a=group:BUNDLE audio video	[I-D.ietf-mmusic-sdp-bundle-negotiation] Alice supports grouping of m=lines under BUNDLE semantics
a=ice-options:trickle	[I-D.ietf-mmusic-trickle-ice]
m=audio 55232 UDP/TLS/RTP/SAVPF 109	[RFC4566]
c=IN IP4 24.23.204.141	[RFC4566]
a=msid:ma ta	Identifies RTCMediaStream ID (ma) and RTCMediaStreamTrack ID (ta)
a=mid:audio	[RFC5888] Audio m=line part of BUNDLE group with a unique port number
a=rtcp:55232 IN IP4 24.23.204.141	[RFC3605]
a=rtpmap:109 opus/48000/2	[I-D.ietf-payload-rtp-opus]
a=extmap:1 urn:ietf:params:rtp-hdrext:ssrc-audio-level	[RFC6464]
a=ptime:20	[I-D.ietf-payload-rtp-opus]
a=setup:actpass	[RFC4145] - Alice can perform DTLS before Answer arrives
a=sendrecv	[RFC3264]
a=rtcp-mux	[RFC5761]
a=rtcp-fb:109 nack	[RFC5104]
a=ssrc:11111 cname:EocUG1f0fcg/yvY7	[RFC5576]
a=ice-ufrag:074c6550	[RFC5245]
a=ice-pwd:a28a397a4c3f31747d1ee3474af08a068	[RFC5245]
a=fingerprint:sha-1 99:41:49:83:4a:97:0e:1f:ef:6d:f7:c9:c7:70:9d:1f:66:79:a8:07	[RFC5245]
a=candidate:0 1 UDP 2122194687 192.168.1.4 55232 typ host	[RFC5245]
a=candidate:0 2 UDP 2122194687 192.168.1.4 55232 typ host	[RFC5245]
a=candidate:1 1 UDP 1685987071 24.23.204.141 55232 typ srflx raddr 192.168.1.4 rport 55232	[RFC5245]
a=candidate:1 2 UDP 1685987071 24.23.204.141 55232 typ srflx raddr 192.168.1.4 rport 55232	[RFC5245]
a=rtcp-rsize	[RFC5506]
m=video 54332 UDP/TLS/RTP/SAVPF 120	[RFC4566]
c=IN IP4 24.23.204.141	[RFC4566]
a=msid:ma tb	Identifies RTCMediaStream ID

	(ma) and RTCMediaStreamTrack ID (tb)
a=mid:video	[RFC5888] Video m=line part of the BUNDLE group with a unique port number
a=rtcp:54332 IN IP4 24.23.204.141	[RFC3605]
a=rtpmap:120 VP8/90000	[I-D.ietf-payload-vp8]
a=sendrecv	[RFC3264]
a=setup:actpass	[RFC4145] - Alice can perform DTLS before Answer arrives
a=rtcp-mux	[RFC5761]
a=ssrc:22222	[RFC5576]
cname:yvY7/EocUG1f0fcg	
a=ice-ufrag:7872093	[RFC5245]
a=ice-pwd:ee3474af08a068a28a397a 4c3f31747d1	[RFC5245]
a=fingerprint:sha-1 6d:f7:c9:c7: 70:9d:1f:66:79:a8:07:99:41: 49:83:4a:97:0e:1f:ef	[RFC5245]
a=candidate:0 1 UDP 2122194687 192.168.1.4 54332 typ host	[RFC5245]
a=candidate:0 2 2122194687 192.168.1.4 54332 typ host	[RFC5245]
a=candidate:1 1 UDP 1685987071 24.23.204.141 54332 typ srflx raddr 192.168.1.4 rport 54332	[RFC5245]
a=candidate:1 2 UDP 1685987071 24.23.204.141 54332 typ srflx raddr 192.168.1.4 rport 54332	[RFC5245]
a=rtcp-fb:120 nack	[RFC5104]
a=rtcp-fb:120 nack pli	[RFC5104]
a=rtcp-fb:120 ccm fir	[RFC5104]
a=rtcp-rsize	[RFC5506]

Table 18: 5.2.9 SDP Offer w/BUNDLE

SDP Contents	RFC#/Notes
v=0	[RFC4566]
o=- 16833 0 IN IP4 0.0.0.0	[RFC4566]
s=-	[RFC4566]
t=0 0	[RFC4566]
a=msid-semantic:WMS ma	[I-D.ietf-mmusic-msid]
a=ice-options:trickle	[I-D.ietf-mmusic-trickle-ice]

```
| m=audio 53214 UDP/TLS/RTP/SAVPF 109   | [RFC4566]                 |
| c=IN IP4 98.248.92.77                 | [RFC4566]                 |
| a=msid:ma ta                          | Identifies RTCMediaStream |
|                                       | ID (ma) and               |
|                                       | RTCMediaStreamTrack ID    |
|                                       | (ta)                      |
| a=rtcp:60065 IN IP4 98.248.92.77      | [RFC3605]                 |
| a=rtpmap:109 opus/48000/2             | [I-D.ietf-payload-rtp-opu |
|                                       | s]                        |
| a=extmap:1 urn:ietf:params:rtp-hdrext | [RFC6464]                 |
| :ssrc-audio-level                     |                           |
| a=ptime:20                            | [I-D.ietf-payload-rtp-opu |
|                                       | s]                        |
| a=setup:active                        | [RFC4145] - Bob carries   |
|                                       | out DTLS Handshake in     |
|                                       | parallel                  |
| a=sendrecv                            | [RFC3264]                 |
| a=rtcp-fb:109 nack                    | [RFC5104]                 |
| a=ice-ufrag:c300d85b                  | [RFC5245]                 |
| a=ice-                                | [RFC5245]                 |
| pwd:de4e99bd291c325921d5d47efbabd9a2  |                           |
| a=fingerprint:sha-1 99:41:49:83:4a:97 | [RFC5245]                 |
| :0e:1f:ef:6d:f7:c9:c7:70:             |                           |
| 9d:1f:66:79:a8:07                     |                           |
| a=candidate:0 1 UDP 2122194687        | [RFC5245]                 |
| 192.168.1.7 53214 typ host            |                           |
| a=candidate:1 1 UDP 1685987071        | [RFC5245]                 |
| 98.248.92.77 53214 typ srflx raddr    |                           |
| 192.168.1.7 rport 53214               |                           |
| a=candidate:0 2 UDP 2122194687        | [RFC5245]                 |
| 192.168.1.7 60065 typ host            |                           |
| a=candidate:1 2 UDP 1685987071        | [RFC5245]                 |
| 98.248.92.77 60065 typ srflx raddr    |                           |
| 192.168.1.7 rport 60065               |                           |
| a=rtcp-rsize                          | [RFC5506]                 |
| m=video 58679 UDP/TLS/RTP/SAVPF 120   | [RFC4566]                 |
| c=IN IP4 98.248.92.77                 | [RFC4566]                 |
| a=msid:ma tb                          | Identifies RTCMediaStream |
|                                       | ID (ma) and               |
|                                       | RTCMediaStreamTrack ID    |
|                                       | (tb)                      |
| a=rtcp:56507 IN IP4 98.248.92.77      | [RFC3605]                 |
| a=rtpmap:120 VP8/90000                | [I-D.ietf-payload-vp8]    |
| a=setup:active                        | [RFC4145] - Bob carries   |
|                                       | out DTLS Handshake in     |
|                                       | parallel                  |
| a=sendrecv                            | [RFC3264]                 |
| a=ice-ufrag:85bC300                   | [RFC5245]                 |
```

a=ice-pwd:325921d5d47efbabd9a2de4e99bd291c	[RFC5245]
a=fingerprint:sha-1 9d:1f:66:79:a8:07 :99:41:49:83:4a:97:0e:1f: ef:6d:f7:c9:c7:70	[RFC5245]
a=candidate:0 1 UDP 2122194687 192.168.1.7 58679 typ host	[RFC5245]
a=candidate:1 1 UDP 1685987071 98.248.92.77 58679 typ srflx raddr 192.168.1.7 rport 58679	[RFC5245]
a=candidate:0 1 UDP 2122194687 192.168.1.7 56507 typ host	[RFC5245]
a=candidate:1 1 UDP 1685987071 98.248.92.77 56507 typ srflx raddr 192.168.1.7 rport 58679	[RFC5245]
a=rtcp-fb:120 nack	[RFC5104]
a=rtcp-fb:120 nack pli	[RFC5104]
a=rtcp-fb:120 ccm fir	[RFC5104]
a=rtcp-rsize	[RFC5506]

Table 19: 5.2.9 SDP Answer without BUNDLE

5.2.10. Audio, Video BUNDLED, but Data (Not BUNDLED)

This example show-cases SDP for negotiating a session with Audio, Video and data streams between Alice and Bob with data stream not being part of the BUNDLE group. This is shown by assigning unique port for data media sections.

Audio, Video, with Data (Not in BUNDLE)

```
   Alice                                                  Bob
   |                                                       |
   |                                                       |
   |Alice wants to multiplex audio, video but not data     |
   |                                                       |
   |                                                       |
   |  Offer(Audio:Opus Video:VP8, Data(not in BUNDLE))     |
   |------------------------------------------------------>|
   |                                                       |
   |                                                       |
   |       Answer(Audio:Opus Video:VP8, Data)              |
   |<------------------------------------------------------|
   |                                                       |
   |                                                       |
   |2 Way Call with Audio, Video Multiplexed except data   |
   |.......................................................|
   |                                                       |


   +----------------------------------+--------------------------------+
   | SDP Contents                     | RFC#/Notes                     |
   +----------------------------------+--------------------------------+
   | v=0                              | [RFC4566]                      |
   | o=- 20518 0 IN IP4 0.0.0.0       | [RFC4566]                      |
   | s=-                              | [RFC4566]                      |
   | t=0 0                            | [RFC4566]                      |
   | a=msid-semantic:WMS ma           | [I-D.ietf-mmusic-msid]         |
   | a=group:BUNDLE audio video       | [I-D.ietf-mmusic-sdp-bundle-ne |
   |                                  | gotiation] Alice wants to      |
   |                                  | BUNDLE only audio and video    |
   |                                  | media.                         |
   | a=ice-options:trickle            | [I-D.ietf-mmusic-trickle-ice]  |
   | m=audio 54609 UDP/TLS/RTP/SAVPF  | [RFC4566]                      |
   | 109                              |                                |
   | c=IN IP4 24.23.204.141           | [RFC4566]                      |
   | a=msid:ma ta                     | Identifies RTCMediaStream ID   |
   |                                  | (ma) and RTCMediaStreamTrack   |
   |                                  | ID (ta)                        |
   | a=rtcp:54609 IN IP4              | [RFC3605]                      |
   | 24.23.204.141                    |                                |
   | a=mid:audio                      | [RFC5888]                      |
   | a=rtpmap:109 opus/48000/2        | [I-D.ietf-payload-rtp-opus]    |
   | a=extmap:1 urn:ietf:params:rtp-  | [RFC6464]                      |
   | hdrext:ssrc-audio-level          |                                |
   | a=ptime:20                       | [I-D.ietf-payload-rtp-opus]    |
```

```
| a=sendrecv                        | [RFC3264]                        |
| a=setup:actpass                   | [RFC4145]                        |
| a=rtcp-mux                        | [RFC5761]                        |
| a=ice-ufrag:074c6550              | [RFC5245]                        |
| a=ice-pwd:a28a397a4c3f31747d1ee3 | [RFC5245]                        |
| 474af08a068                       |                                  |
| a=fingerprint:sha-1 99:41:49:83:  | [RFC5245]                        |
| 4a:97:0e:1f:ef:6d:f7:c9:c7:70:    |                                  |
| 9d:1f:66:79:a8:07                 |                                  |
| a=candidate:0 1 UDP  2122194687   | [RFC5245]                        |
| 192.168.1.4 54609 typ host        |                                  |
| a=candidate:0 2 UDP 2122194687    | [RFC5245]                        |
| 192.168.1.4 54609 typ host        |                                  |
| a=candidate:1 1 UDP  1685987071   | [RFC5245]                        |
| 24.23.204.141 54609 typ srflx     |                                  |
| raddr 192.168.1.4 rport 54609     |                                  |
| a=candidate:1 2 UDP  1685987071   | [RFC5245]                        |
| 24.23.204.141 54609 typ srflx     |                                  |
| raddr 192.168.1.4 rport 54609     |                                  |
| a=rtcp-fb:109 nack                | [RFC5104]                        |
| a=ssrc:11111                      | [RFC5576]                        |
| cname:Q/NWs1ao1HmN4Xa5            |                                  |
| a=rtcp-rsize                      | [RFC5506]                        |
| m=video 54609 UDP/TLS/RTP/SAVPF   | [RFC4566]                        |
| 120                               |                                  |
| c=IN IP4 24.23.204.141            | [RFC4566]                        |
| a=msid:ma tb                      | Identifies RTCMediaStream ID     |
|                                   | (ma) and RTCMediaStreamTrack     |
|                                   | ID (tb)                          |
| a=rtcp:54609 IN IP4               | [RFC3605]                        |
| 24.23.204.141                     |                                  |
| a=mid:video                       | [RFC5888]                        |
| a=rtpmap:120 VP8/90000            | [I-D.ietf-payload-vp8]           |
| a=sendrecv                        | [RFC3264]                        |
| a=setup:actpass                   | [RFC4145]                        |
| a=rtcp-mux                        | [RFC5761]                        |
| a=ice-ufrag:074c6550              | [RFC5245]                        |
| a=ice-pwd:a28a397a4c3f31747d1ee3 | [RFC5245]                        |
| 474af08a068                       |                                  |
| a=fingerprint:sha-1 99:41:49:83:  | [RFC5245]                        |
| 4a:97:0e:1f:ef:6d:f7:c9:c7:70:    |                                  |
| 9d:1f:66:79:a8:07                 |                                  |
| a=candidate:0 1 UDP  2122194687   | [RFC5245]                        |
| 192.168.1.4 54609 typ host        |                                  |
| a=candidate:0 2 UDP 2122194687    | [RFC5245]                        |
| 192.168.1.4 54609 typ host        |                                  |
| a=candidate:1 1 UDP  1685987071   | [RFC5245]                        |
| 24.23.204.141 54609 typ srflx     |                                  |
```

raddr 192.168.1.4 rport 54609	
a=candidate:1 2 UDP 1685987071	[RFC5245]
24.23.204.141 54609 typ srflx	
raddr 192.168.1.4 rport 54609	
a=rtcp-fb:120 nack	[RFC5104]
a=rtcp-fb:120 nack pli	[RFC5104]
a=rtcp-fb:120 ccm fir	[RFC5104]
a=ssrc:22222	[RFC5576]
cname:Q/aoNWs11HmN4Xa5	
a=rtcp-rsize	[RFC5506]
m=application 10000	[I-D.ietf-rtcweb-data-channel]
UDP/DTLS/SCTP 5000	
c=IN IP4 24.23.204.141	[RFC4566]
a=mid:data	[RFC5888]
a=sctpmap:5000 webrtc-	[I-D.ietf-mmusic-sctp-sdp]
DataChannel	
streams=16;label="channel 1";	
subprotocol="chat";	
a=sendrecv	[RFC3264]
a=setup:actpass	[RFC4145]
a=ice-ufrag:89819013	[RFC5245]
a=ice-pwd:1747d1ee3474af08a068a2	[RFC5245]
8a397a4c3f3	
a=fingerprint:sha-1 0e:1f:ef:6d:	[RFC5245]
f7:c9:c7:70:99:41:49:83:4a:97:	
9d:1f:66:79:a8:07	
a=candidate:0 1 UDP 2113667327	[RFC5245]
192.168.1.4 10000 typ host	
a=candidate:1 1 UDP 1685987071	[RFC5245]
24.23.204.141 10000 typ srflx	
raddr 192.168.1.4 rport 10000	

Table 20: 5.2.10 SDP Offer

SDP Contents	RFC#/Notes
v=0	[RFC4566]
o=- 16833 0 IN IP4 0.0.0.0	[RFC4566] - Session Origin Information
s=-	[RFC4566]
t=0 0	[RFC4566]
a=msid-semantic:WMS ma	[I-D.ietf-mmusic-msid]
a=group:BUNDLE audio video	[I-D.ietf-mmusic-sdp-bundle-negotiation]
a=ice-options:trickle	[I-D.ietf-mmusic-trickle-ice]
m=audio 49203 UDP/TLS/RTP/SAVPF	[RFC4566]

109	
c=IN IP4 98.248.92.77	[RFC4566]
a=msid:ma ta	Identifies RTCMediaStream ID (ma) and RTCMediaStreamTrack ID (ta)
a=mid:audio	[RFC5888]
a=rtpmap:109 opus/48000/2	[I-D.ietf-payload-rtp-opus]
a=extmap:1 urn:ietf:params:rtp-hdrext:ssrc-audio-level	[RFC6464]
a=ptime:20	[I-D.ietf-payload-rtp-opus]
a=sendrecv	[RFC3264]
a=setup:active	[RFC4145]
a=rtcp-mux	[RFC5761]
a=rtcp-fb:109 nack	[RFC5104]
a=ice-ufrag:c300d85b	[RFC5245]
a=ice-pwd:de4e99bd291c325921d5d47efbabd9a2	[RFC5245]
a=fingerprint:sha-1 99:41:49:83:4a:97:0e:1f:ef:6d:f7:c9:c7:70:9d:1f:66:79:a8:07	[RFC5245]
a=candidate:0 1 UDP 2113667327 192.168.1.7 49203 typ host	[RFC5245]
a=candidate:1 1 UDP 1685987071 24.23.204.141 49203 typ srflx raddr 192.168.1.7 rport 49203	[RFC5245]
a=ssrc:33333 cname:L/aoNWs11HmN4Xa5	[RFC5576]
a=rtcp-rsize	[RFC5506]
m=video 49203 UDP/TLS/RTP/SAVPF 120	[RFC4566]
c=IN IP4 98.248.92.771	[RFC4566]
a=msid:ma tb	Identifies RTCMediaStream ID (ma) and RTCMediaStreamTrack ID (tb)
a=mid:video	[RFC5888]
a=rtpmap:120 VP8/90000	[I-D.ietf-payload-vp8]
a=sendrecv	[RFC3264]
a=setup:active	[RFC4145]
a=rtcp-mux	[RFC5761]
a=ice-ufrag:c300d85b	[RFC5245]
a=ice-pwd:de4e99bd291c325921d5d47efbabd9a2	[RFC5245]
a=fingerprint:sha-1 99:41:49:83:4a:97:0e:1f:ef:6d:f7:c9:c7:70:9d:1f:66:79:a8:07	[RFC5245]
a=candidate:0 1 UDP 2113667327 192.168.1.7 49203 typ host	[RFC5245]
a=candidate:1 1 UDP 1685987071	[RFC5245]

```
| 24.23.204.141 49203 typ srflx    |                                |
| raddr 192.168.1.7 rport 49203    |                                |
| a=rtcp-fb:120 nack               | [RFC5104]                      |
| a=rtcp-fb:120 nack pli           | [RFC5104]                      |
| a=rtcp-fb:120 ccm fir            | [RFC5104]                      |
| a=ssrc:44444                     | [RFC5576]                      |
| cname:EocUG1f0fcg/yvY7           |                                |
| a=rtcp-rsize                     | [RFC5506]                      |
| m=application 20000              | [I-D.ietf-mmusic-sctp-sdp]     |
| UDP/DTLS/SCTP 5000               |                                |
| c=IN IP4 98.248.92.77            | [RFC4566]                      |
| a=mid:data                       | [RFC5888]                      |
| a=sctpmap:5000 webrtc-           | [I-D.ietf-mmusic-sctp-sdp]     |
| DataChannel                      |                                |
| streams=1;label="channel 1";     |                                |
| subprotocol="chat";              |                                |
| a=setup:active                   | [RFC4145]                      |
| a=sendrecv                       | [RFC3264]                      |
| a=ice-ufrag:991Ca2a5e            | [RFC5245]                      |
| a=ice-pwd:921d5d47efbabd9a2de4e9 | [RFC5245]                      |
| 9bd291c325                       |                                |
| a=fingerprint:sha-1 6d:f7:c9:c7: | [RFC5245]                      |
| 70:9d:1f:66:79:a8:07:99:41:49:   |                                |
| 83:4a:97:0e:1f:ef                |                                |
| a=candidate:0 1 UDP 2113667327   | [RFC5245]                      |
| 192.168.1.7 20000 typ host       |                                |
| a=candidate:1 1 UDP  1685987071  | [RFC5245]                      |
| 24.23.204.141 20000 typ srflx    |                                |
| raddr 192.168.1.7 rport 20000    |                                |
+----------------------------------+--------------------------------+
```

Table 21: 5.2.10 SDP Answer

5.2.11. Audio Only, Add Video to BUNDLE

This example involves 2 Offer/Answer exchanges. First one setting up Audio-only session followed by an updated Offer/Answer exchange to add video stream to the ongoing session. Also the newly added video stream is BUNDLED with the audio stream.

Audio Only , Add Video and BUNDLE

```
   Alice                                                   Bob
   |                                                        |
   |                                                        |
   |        Alice indicates support for BUNDLE              |
   |                                                        |
   |                Offer(Audio:Opus)                       |
   |------------------------------------------------------>|
   |                                                        |Bob
   |                                                        |supports
   |                                                        |BUNDLE
   |                Answer(Audio:Opus)                      |
   |<------------------------------------------------------|Alice
   |                                                        |adds
   |                                                        |video
   |        Updated Offer(Audio:Opus, Video:VP8)            |to BUNDLE
   |------------------------------------------------------>|
   |                                                        |
   |                                                        |Bob
   |        Updated Answer(Audio:Opus, Video:VP8)           |accepts
   |<------------------------------------------------------|
   |                                                        |
   |   2Way Call with Audio and Video Multiplexed           |
   |.......................................................|
   |                                                        |
   |                                                        |
```

SDP Contents	RFC#/Notes
v=0	[RFC4566]
o=- 20518 0 IN IP4 0.0.0.0	[RFC4566]
s=-	[RFC4566]
t=0 0	[RFC4566]
a=msid-semantic:WMS ma	[I-D.ietf-mmusic-msid]
a=group:BUNDLE audio	[I-D.ietf-mmusic-sdp-bundle-negotiation]
a=ice-options:trickle	[I-D.ietf-mmusic-trickle-ice]
m=audio 54609 UDP/TLS/RTP/SAVPF 109	[RFC4566]
c=IN IP4 24.23.204.141	[RFC4566]
a=msid:ma ta	Identifies RTCMediaStream ID (ma) and RTCMediaStreamTrack ID (ta)
a=rtcp-mux	[RFC5761]
a=rtcp:54609 IN IP4	[RFC3605]

24.23.204.141	
a=mid:audio	[RFC5888]
a=rtpmap:109 opus/48000/2	[I-D.ietf-payload-rtp-opus]
a=extmap:1 urn:ietf:params:rtp-hdrext:ssrc-audio-level	[RFC6464]
a=ptime:20	[I-D.ietf-payload-rtp-opus]
a=sendrecv	[RFC3264]
a=setup:actpass	[RFC4145]
a=ice-ufrag:074c6550	[RFC5245]
a=ice-pwd:a28a397a4c3f31747d1ee3474af08a068	[RFC5245]
a=fingerprint:sha-1 99:41:49:83:4a:97:0e:1f:ef:6d:f7:c9:c7:70:9d:1f:66:79:a8:07	[RFC5245]
a=candidate:0 1 UDP 2122194687 192.168.1.4 54609 typ host	[RFC5245]
a=candidate:0 2 UDP 2122194687 192.168.1.4 54609 typ host	[RFC5245]
a=candidate:1 1 UDP 1685987071 24.23.204.141 54609 typ srflx raddr 192.168.1.4 rport 54609	[RFC5245]
a=candidate:1 2 UDP 1685987071 24.23.204.141 54609 typ srflx raddr 192.168.1.4 rport 54609	[RFC5245]
a=rtcp-fb:109 nack	[RFC5104]
a=ssrc:11111 cname:Q/NWs1ao1HmN4Xa5	[RFC5576]
a=rtcp-rsize	[RFC5506]

Table 22: 5.2.11 SDP Offer

SDP Contents	RFC#/Notes
v=0	[RFC4566]
o=- 16833 0 IN IP4 0.0.0.0	[RFC4566] - Session Origin Information
s=-	[RFC4566]
t=0 0	[RFC4566]
a=msid-semantic:WMS ma	[I-D.ietf-mmusic-msid]
a=group:BUNDLE audio	[I-D.ietf-mmusic-sdp-bundle-negotiation]
a=ice-options:trickle	[I-D.ietf-mmusic-trickle-ice]
m=audio 49203 UDP/TLS/RTP/SAVPF 109	[RFC4566]
c=IN IP4 98.248.92.77	[RFC4566]
a=extmap:1 urn:ietf:params:rtp-hdrext:ssrc-audio-level	[RFC6464]
a=msid:ma ta	Identifies RTCMediaStream ID (ma) and RTCMediaStreamTrack ID (ta)
a=mid:audio	[RFC5888]
a=rtpmap:109 opus/48000/2	[I-D.ietf-payload-rtp-opus]
a=ptime:20	[I-D.ietf-payload-rtp-opus]
a=sendrecv	[RFC3264]
a=setup:active	[RFC4145]
a=rtcp-mux	[RFC5761]
a=rtcp-fb:109 nack	[RFC5104]
a=ice-ufrag:c300d85b	[RFC5245]
a=ice-pwd:de4e99bd291c325921d5d47efbabd9a2	[RFC5245]
a=fingerprint:sha-1 99:41:49:83:4a:97:0e:1f:ef:6d:f7:c9:c7:70:9d:1f:66:79:a8:07	[RFC5245]
a=candidate:0 1 UDP 2122194687 192.168.1.7 49203 typ host	[RFC5245]
a=candidate:1 1 UDP 1685987071 98.248.92.77 49203 typ srflx raddr 192.168.1.7 rport 49203	[RFC5245]
a=ssrc:33333 cname:L/aoNWs11HmN4Xa5	[RFC5576]
a=rtcp-rsize	[RFC5506]

Table 23: 5.2.10 SDP Answer

SDP Contents	RFC#/Notes

```
| v=1                                | Version number incremented     |
|                                    | [RFC4566]                      |
| o=- 20518 0 IN IP4 0.0.0.0         | [RFC4566]                      |
| s=-                                | [RFC4566]                      |
| t=0 0                              | [RFC4566]                      |
| a=msid-semantic:WMS ma             | [I-D.ietf-mmusic-msid]         |
| a=group:BUNDLE audio video         | [I-D.ietf-mmusic-sdp-bundle-ne |
|                                    | gotiation] Alice adds video    |
|                                    | stream to BUNDLE.              |
| a=ice-options:trickle              | [I-D.ietf-mmusic-trickle-ice]  |
| m=audio 54609 UDP/TLS/RTP/SAVPF    | [RFC4566]                      |
| 109                                |                                |
| c=IN IP4 24.23.204.141             | [RFC4566]                      |
| a=extmap:1 urn:ietf:params:rtp-    | [RFC6464]                      |
| hdrext:ssrc-audio-level            |                                |
| a=msid:ma ta                       | Identifies RTCMediaStream ID   |
|                                    | (ma) and RTCMediaStreamTrack   |
|                                    | ID (ta)                        |
| a=rtcp-mux                         | [RFC5761]                      |
| a=rtcp:54609 IN IP4                | [RFC3605]                      |
| 24.23.204.141                      |                                |
| a=mid:audio                        | [RFC5888]                      |
| a=rtpmap:109 opus/48000/2          | [I-D.ietf-payload-rtp-opus]    |
| a=ptime:20                         | [I-D.ietf-payload-rtp-opus]    |
| a=sendrecv                         | [RFC3264]                      |
| a=setup:actpass                    | [RFC4145]                      |
| a=ice-ufrag:074c6550               | [RFC5245]                      |
| a=ice-pwd:a28a397a4c3f31747d1ee3   | [RFC5245]                      |
| 474af08a068                        |                                |
| a=fingerprint:sha-1 99:41:49:83:   | [RFC5245]                      |
| 4a:97:0e:1f:ef:6d:f7:c9:c7:70:9d   |                                |
| : 1f:66:79:a8:07                   |                                |
| a=candidate:0 1 UDP  2122194687    | [RFC5245]                      |
| 192.168.1.4 54609 typ host         |                                |
| a=candidate:0 2 UDP 2122194687     | [RFC5245]                      |
| 192.168.1.4 54609 typ host         |                                |
| a=candidate:1 1 UDP  1685987071    | [RFC5245]                      |
| 24.23.204.141 54609 typ srflx      |                                |
| raddr 192.168.1.4 rport 54609      |                                |
| a=candidate:1 2 UDP  1685987071    | [RFC5245]                      |
| 24.23.204.141 54609 typ srflx      |                                |
| raddr 192.168.1.4 rport 54609      |                                |
| a=rtcp-fb:109 nack                 | [RFC5104]                      |
| a=ssrc:11111                       | [RFC5576]                      |
| cname:Q/NWs1ao1HmN4Xa5             |                                |
| a=rtcp-rsize                       | [RFC5506]                      |
| m=video 54609 UDP/TLS/RTP/SAVPF    | [RFC4566]                      |
| 120                                |                                |
```

c=IN IP4 24.23.204.141	[RFC4566]
a=msid:ma tb	Identifies RTCMediaStream ID (ma) and RTCMediaStreamTrack ID (tb)
a=rtcp:54609 IN IP4 24.23.204.141	[RFC3605]
a=mid:video	[RFC5888]
a=rtpmap:120 VP8/90000	[I-D.ietf-payload-vp8]
a=sendrecv	[RFC3264]
a=setup:actpass	[RFC4145]
a=rtcp-mux	[RFC5761]
a=ice-ufrag:074c6550	[RFC5245]
a=ice-pwd:a28a397a4c3f31747d1ee3 474af08a068	[RFC5245]
a=fingerprint:sha-1 99:41:49:83: 4a:97:0e:1f:ef:6d:f7:c9:c7:70: 9d:1f:66:79:a8:07	[RFC5245]
a=candidate:0 1 UDP 2122194687 192.168.1.4 54609 typ host	[RFC5245]
a=candidate:0 2 UDP 2122194687 192.168.1.4 54609 typ host	[RFC5245]
a=candidate:1 1 UDP 1685987071 24.23.204.141 54609 typ srflx raddr 192.168.1.4 rport 54609	[RFC5245]
a=candidate:1 2 UDP 1685987071 24.23.204.141 54609 typ srflx raddr 192.168.1.4 rport 54609	[RFC5245]
a=rtcp-fb:120 nack	[RFC5104]
a=rtcp-fb:120 nack pli	[RFC5104]
a=rtcp-fb:120 ccm fir	[RFC5104]
a=ssrc:22222 cname:Q/aoNWs11HmN4Xa5	[RFC5576]
a=rtcp-rsize	[RFC5506]

Table 24: 5.2.11 SDP Updated Offer

SDP Contents	RFC#/Notes
v=1	[RFC4566] Version number incremented
o=- 16833 0 IN IP4 0.0.0.0	[RFC4566] - Session Origin Information
s=-	[RFC4566]
t=0 0	[RFC4566]
a=msid-semantic:WMS ma	[I-D.ietf-mmusic-msid]
a=group:BUNDLE audio video	[I-D.ietf-mmusic-sdp-bundle-ne

```
   |                                   | gotiation]                      |
   | a=ice-options:trickle             | [I-D.ietf-mmusic-trickle-ice]   |
   | m=audio 49203 UDP/TLS/RTP/SAVPF   | [RFC4566]                       |
   | 109                               |                                 |
   | c=IN IP4 98.248.92.77             | [RFC4566]                       |
   | a=msid:ma ta                      | Identifies RTCMediaStream ID    |
   |                                   | (ma) and RTCMediaStreamTrack    |
   |                                   | ID (ta)                         |
   | a=mid:audio                       | [RFC5888]                       |
   | a=rtpmap:109 opus/48000/2         | [I-D.ietf-payload-rtp-opus]     |
   | a=extmap:1 urn:ietf:params:rtp-   | [RFC6464]                       |
   | hdrext:ssrc-audio-level           |                                 |
   | a=ptime:20                        | [I-D.ietf-payload-rtp-opus]     |
   | a=sendrecv                        | [RFC3264]                       |
   | a=setup:active                    | [RFC4145]                       |
   | a=rtcp-mux                        | [RFC5761]                       |
   | a=rtcp-fb:109 nack                | [RFC5104]                       |
   | a=ice-ufrag:c300d85b              | [RFC5245]                       |
   | a=ice-pwd:de4e99bd291c325921d5d4  | [RFC5245]                       |
   | 7efbabd9a2                        |                                 |
   | a=fingerprint:sha-1 99:41:49:83:  | [RFC5245]                       |
   | 4a:97:0e:1f:ef:6d:f7:c9:c7:70:    |                                 |
   | 9d:1f:66:79:a8:07                 |                                 |
   | a=candidate:0 1 UDP 2122194687    | [RFC5245]                       |
   | 192.168.1.7 49203 typ host        |                                 |
   | a=candidate:1 1 UDP 1685987071    | [RFC5245]                       |
   | 98.248.92.77 49203 typ srflx      |                                 |
   | raddr 192.168.1.7 rport 49203     |                                 |
   | a=ssrc:33333                      | [RFC5576]                       |
   | cname:L/aoNWs11HmN4Xa5            |                                 |
   | a=rtcp-rsize                      | [RFC5506]                       |
   | m=video 49203 UDP/TLS/RTP/SAVPF   | [RFC4566]                       |
   | 120                               |                                 |
   | c=IN IP4 98.248.92.77             | [RFC4566]                       |
   | a=msid:ma tb                      | Identifies RTCMediaStream ID    |
   |                                   | (ma) and RTCMediaStreamTrack    |
   |                                   | ID (tb)                         |
   | a=mid:video                       | [RFC5888]                       |
   | a=rtpmap:120 VP8/90000            | [I-D.ietf-payload-vp8]          |
   | a=sendrecv                        | [RFC3264]                       |
   | a=setup:active                    | [RFC4145]                       |
   | a=rtcp-mux                        | [RFC5761]                       |
   | a=ice-ufrag:c300d85b              | [RFC5245]                       |
   | a=ice-pwd:de4e99bd291c325921d5d4  | [RFC5245]                       |
   | 7efbabd9a2                        |                                 |
   | a=fingerprint:sha-1 99:41:49:83:  | [RFC5245]                       |
   | 4a:97:0e:1f:ef:6d:f7:c9:c7:70:    |                                 |
   | 9d:1f:66:79:a8:07                 |                                 |
```

a=candidate:0 1 UDP 2122194687 192.168.1.7 49203 typ host	[RFC5245]
a=candidate:1 1 UDP 1685987071 98.248.92.77 49203 typ srflx raddr 192.168.1.7 rport 49203	[RFC5245]
a=rtcp-fb:120 nack	[RFC5104]
a=rtcp-fb:120 nack pli	[RFC5104]
a=rtcp-fb:120 ccm fir	[RFC5104]
a=ssrc:44444 cname:EocUG1f0fcg/yvY7	[RFC5576]
a=rtcp-rsize	[RFC5506]

Table 25: 5.2.11 SDP Updated Answer

5.3. MultiResolution, RTX, FEC Examples

This section deals with scenarios related to multi-source, multi-stream negotiation such as layered coding, simulcast, along with techniques that deal with providing robustness against transmission errors such as FEC and RTX. Also to note, mechanisms such as FEC and RTX could be envisioned in the above basic scenarios as well.

5.3.1. Sendonly Simulcast Session with 2 cameras and 2 encodings per camera

The SDP below shows Offer/Answer exchange with one audio and two video sources. Each of the video source can be sent at two different resolutions.

One video source corresponds to VP8 encoding, while the other corresponds to H.264 encoding.

bundle-only framework is used along with BUNDLE grouping framework to enable multiplexing of all the 5 streams (1 audio stream + 4 video streams) over a single RTP Session.

One Way Successful Simulcast w/BUNDLE

```
Alice                                                Bob
|                                                     |
|     Alice offers 2 sendonly video sources           |
|     with 2 simulcast encodings per source           |
|           and bundle-only for video                 |
|                                                     |
|                                                     |
|   Offer(Audio:Opus,Video1:VP8,Video2:H.264)         |
|-------------------------------------------------->|
|                                                     |
|                                                     |
|    Answer(Audio:Opus Video1:VP8,Video2:H.264)       |
|<--------------------------------------------------|
|                                                     |
|One-Way 1 Opus, 2 H.264 and 2 VP8 video streams,     |
| all multiplexed                                     |
|.....................................................|
|                                                     |
|                                                     |
```

```
+-------------------------------------+-----------------------------+
| SDP Contents                        | RFC#/Notes                  |
+-------------------------------------+-----------------------------+
| v=0                                 | [RFC4566]                   |
| o=- 20519 0 IN IP4 0.0.0.0          | [RFC4566]                   |
| s=-                                 | [RFC4566]                   |
| t=0 0                               | [RFC4566]                   |
| a=msid-semantic:WMS ma              | [I-D.ietf-mmusic-msid]      |
| a=group:BUNDLE m0 m1 m2             | [I-D.ietf-mmusic-sdp-bundle |
|                                     | -negotiation] Alice         |
|                                     | supports grouping of        |
|                                     | m=lines under BUNDLE        |
|                                     | semantics                   |
| a=ice-options:trickle               | [I-D.ietf-mmusic-trickle-ic |
|                                     | e]                          |
| m=audio 54609 UDP/TLS/RTP/SAVPF 109 | [RFC4566]                   |
| c=IN IP4 24.23.204.141              | [RFC4566]                   |
| a=msid:ma ta                        | Identifies RTCMediaStream   |
|                                     | ID (ma) and                 |
|                                     | RTCMediaStreamTrack ID (ta) |
| a=rtcp-mux                          | [RFC5761]                   |
| a=rtcp:54609 IN IP4 24.23.204.141   | [RFC3605]                   |
| a=mid:m0                            | [RFC5888]                   |
| a=rtpmap:109 opus/48000/2           | [I-D.ietf-payload-rtp-opus] |
```

a=extmap:1 urn:ietf:params:rtp-hdrext:ssrc-audio-level	[RFC6464]
a=ptime:20	[I-D.ietf-payload-rtp-opus]
a=sendonly	[RFC3264]
a=setup:actpass	[RFC4145]
a=rtcp-fb:109 nack	[RFC5104]
a=ssrc:11111 C90alEocUG1f0fcg	[RFC5576]
a=ice-ufrag:074c6550	[RFC5245]
a=ice-pwd:a28a397a4c3f31747d1ee3474af08a068	[RFC5245]
a=fingerprint:sha-1 99:41:49:83:4a:97:0e:1f:ef:6d:f7:c9:c7:70:9d:1f:66:79:a8:07	[RFC5245]
a=candidate:0 1 UDP 2122194687 192.168.1.4 54609 typ host	[RFC5245]
a=candidate:0 2 UDP 2122194687 192.168.1.4 54609 typ host	[RFC5245]
a=candidate:1 1 UDP 1685987071 24.23.204.141 54609 typ srflx raddr 192.168.1.4 rport 54609	[RFC5245]
a=candidate:1 2 UDP 1685987071 24.23.204.141 54609 typ srflx raddr 192.168.1.4 rport 54609	[RFC5245]
a=rtcp-rsize	[RFC5506]
m=video 0 UDP/TLS/RTP/SAVPF 98 100	bundle-only video line with port number set to zero
c=IN IP4 24.23.204.141	[RFC4566]
a=msid:ma tb	Identifies RTCMediaStream ID (ma) and RTCMediaStreamTrack ID (tb)
a=rtcp-mux	[RFC5761]
a=rtcp:0 IN IP4 24.23.204.141	[RFC3605]
a=mid:m1	[RFC5888] Video m=line part of BUNDLE group
a=rtpmap:98 VP8/90000	[I-D.ietf-payload-vp8]
a=rtpmap:100 VP8/90000	[I-D.ietf-payload-vp8]
a=imageattr:98 [x=1280,y=720]	[RFC6236]Camera-1,Encoding-1 Resolution
a=fmtp:98 max-fr=30	[RFC4566]
a=imageattr:100 [x=640,y=480]	[RFC6236] Camera-1,Encoding-2 Resolution
a=fmtp:100 max-fr=15	[RFC4566]
a=simulcast: send 98;100	[I-D.ietf-mmusic-sdp-simulcast] Alice can send 2 resolutions
a=ssrc:12345 cname:axzo1278npDlAzM73	[RFC5576] [RFC7022] Camera-1,Encoding-1 SSRC

	with Session CNAME
a=ssrc:45678	[RFC5576] [RFC7022]
cname:axzo1278npDlAzM73	Camera-1,Encoding-2 SSRC with Session CNAME
a=sendonly	[RFC3264] - Send only video stream
a=bundle-only	[UNIFIED-PLAN]
a=rtcp-fb:98 nack	[RFC5104]
a=rtcp-fb:98 nack pli	[RFC5104]
a=rtcp-fb:98 ccm fir	[RFC5104]
a=rtcp-fb:100 nack	[RFC5104]
a=rtcp-fb:100 nack pli	[RFC5104]
a=rtcp-fb:100 ccm fir	[RFC5104]
a=rtcp-rsize	[RFC5506]
m=video 0 UDP/TLS/RTP/SAVPF 101 102	bundle-only video line with port number set to zero
c=IN IP4 24.23.204.141	[RFC4566]
a=msid:ma tc	Identifies RTCMediaStream ID (ma) and RTCMediaStreamTrack ID (tc)
a=rtcp:0 IN IP4 24.23.204.141	[RFC3605]
a=mid:m2	[RFC5888] Video m=line part of BUNDLE group
a=rtcp-mux	[RFC5761]
a=rtcp:0 IN IP4 24.23.204.141	[RFC3605]
a=rtpmap:101 H264/90000	[RFC3984]
a=rtpmap:102 H264/90000	[RFC3984]
a=fmtp:101 profile-level-id=4d0028 ;packetization-mode=1;max-fr=30	[RFC3984]Camera-2,Encoding-1 Resolution
a=fmtp:102 profile-level-id=4d0028 ;packetization-mode=1;max-fr=15	[RFC3984]Camera-2,Encoding-2 Resolution
a=simulcast: send 101;102	[I-D.ietf-mmusic-sdp-simulcast]
a=ssrc:67890	[RFC5576] [RFC7022]
cname:axzo1278npDlAzM73	Camera-2,Encoding-1 SSRC with Session CNAME
a=ssrc:56789	[RFC5576] [RFC7022]
cname:axzo1278npDlAzM73	Camera-2,Encoding-2 SSRC with Session CNAME
a=sendonly	[RFC3264] - Send only video stream
a=bundle-only	[UNIFIED-PLAN]
a=rtcp-fb:101 nack	[RFC5104]
a=rtcp-fb:101 nack pli	[RFC5104]
a=rtcp-fb:101 ccm fir	[RFC5104]
a=rtcp-fb:102 nack	[RFC5104]
a=rtcp-fb:102 nack pli	[RFC5104]
a=rtcp-fb:102 ccm fir	[RFC5104]

a=rtcp-rsize	[RFC5506]

Table 26: 5.3.1 SDP Offer

SDP Contents	RFC#/Notes
v=0	[RFC4566]
o=- 20519 0 IN IP4 0.0.0.0	[RFC4566]
s=-	[RFC4566]
t=0 0	[RFC4566]
a=msid-semantic:WMS ma	[I-D.ietf-mmusic-msid]
a=group:BUNDLE m0 m1 m2	[I-D.ietf-mmusic-sdp-bundle -negotiation] Alice supports grouping of m=lines under BUNDLE semantics
a=ice-options:trickle	[I-D.ietf-mmusic-trickle-ice]
m=audio 49203 UDP/TLS/RTP/SAVPF 109	[RFC4566]
c=IN IP4 98.248.92.77	[RFC4566]
a=mid:m0	[RFC5888]
a=msid:ma ta	Identifies RTCMediaStream ID (ma) and RTCMediaStreamTrack ID (ta)
a=rtpmap:109 opus/48000/2	[I-D.ietf-payload-rtp-opus]
a=extmap:1 urn:ietf:params:rtp-hdrext:ssrc-audio-level	[RFC6464]
a=rtcp-fb:109 nack	[RFC5104]
a=ptime:20	[I-D.ietf-payload-rtp-opus]
a=recvonly	[RFC3264]
a=setup:active	[RFC4145]
a=rtcp-mux	[RFC5761]
a=ssrc:22222 cname:y8/C90alEocUG1f0fcg	[RFC5576]
a=ice-ufrag:c300d85b	[RFC5245]
a=ice-pwd:de4e99bd291c325921d5d47efbabd9a2	[RFC5245]
a=fingerprint:sha-1 99:41:49:83:4a:97:0e:1f:ef:6d:f7:c9:c7:70:9d:1f:66:79:a8:07	[RFC5245]
a=candidate:0 2 UDP 2122194687 192.168.1.7 49203 typ host	[RFC5245]
a=candidate:1 2 UDP 1685987071 98.248.92.77 49203 typ srflx raddr 192.168.1.7 rport 49203	[RFC5245]
a=rtcp-rsize	[RFC5506]

```
| m=video 49203 UDP/TLS/RTP/SAVPF 98  | BUNDLE accepted with port   |
| 100                                 | repeated from the audio     |
|                                     | port                        |
| c=IN IP4 98.248.92.77               | [RFC4566]                   |
| a=msid:ma tb                        | Identifies RTCMediaStream   |
|                                     | ID (ma) and                 |
|                                     | RTCMediaStreamTrack ID (tb) |
| a=mid:m1                            | [RFC5888] Video m=line part |
|                                     | of BUNDLE group             |
| a=rtpmap:98 VP8/90000               | [I-D.ietf-payload-vp8]      |
| a=rtpmap:100 VP8/90000              | [I-D.ietf-payload-vp8]      |
| a=imageattr:98 [x=1280,y=720]       | [RFC6236]Camera-1,Encoding- |
|                                     | 1 Resolution                |
| a=fmtp:98 max-fr=30                 | [RFC4566]                   |
| a=imageattr:100 [x=640,y=480]       | [RFC6236]                   |
|                                     | Camera-1,Encoding-2         |
|                                     | Resolution                  |
| a=fmtp:100 max-fr=15                | [RFC4566]                   |
| a=recvonly                          | [RFC3264] - receive only    |
|                                     | video stream                |
| a=simulcast: recv 98;100            | [I-D.ietf-mmusic-sdp-simulc |
|                                     | ast]                        |
| a=ssrc:54321                        | [RFC5576]                   |
| cname:y8/C90alEocUG1f0fcg           |                             |
| a=ice-ufrag:c300d85b                | [RFC5245]                   |
| a=ice-pwd:de4e99bd291c325921d5d47ef | [RFC5245]                   |
| babd9a2                             |                             |
| a=fingerprint:sha-1 99:41:49:83:4a: | [RFC5245]                   |
| 97:0e:1f:ef:6d:f7:c9:c7:70:9d:      |                             |
| 1f:66:79:a8:07                      |                             |
| a=candidate:0 2 UDP 2113667326      | [RFC5245]                   |
| 192.168.1.7 49203 typ host          |                             |
| a=candidate:1 2 UDP  1694302206     | [RFC5245]                   |
| 98.248.92.77 49203 typ srflx raddr  |                             |
| 192.168.1.7 rport 49203             |                             |
| a=setup:active                      | [RFC4145]                   |
| a=rtcp-mux                          | [RFC5576]                   |
| a=bundle-only                       | [UNIFIED-PLAN]              |
| a=rtcp-rsize                        | [RFC5506]                   |
| m=video 49203 UDP/TLS/RTP/SAVPF 101 | BUNDLE accepted with port   |
| 102                                 | repeated from the audio     |
|                                     | port                        |
| c=IN IP4 98.248.92.77               | [RFC4566]                   |
| a=msid:ma tc                        | Identifies RTCMediaStream   |
|                                     | ID (ma) and                 |
|                                     | RTCMediaStreamTrack ID (tc) |
| a=mid:m2                            | [RFC5888] Video m=line part |
|                                     | of BUNDLE group             |
```

a=rtpmap:101 H264/90000	[RFC3984]
a=rtpmap:102 H264/90000	[RFC3984]
a=recvonly	[RFC3264]
a=fmtp:101 profile-level-id=4d0028 ;packetization-mode=1;max-fr=30	[RFC3984]
a=fmtp:102 profile-level-id=4d0028 ;packetization-mode=1;max-fr=15	[RFC3984]
a=simulcast: recv 101;102	[I-D.ietf-mmusic-sdp-simulcast] Bob accepts to receieve the offered simulcast streams
a=ssrc:90876 cname:axzo1278npDlAzM73	[RFC5576]
a=ice-ufrag:ufrag:c300d85b	[RFC5245]
a=ice-pwd:de4e99bd291c325921d5d47ef babd9a2	[RFC5245]
a=fingerprint:sha-1 99:41:49:83:4a: 97:0e:1f:ef:6d:f7:c9:c7:70: 9d:1f:66:79:a8:07	[RFC5245]
a=candidate:0 2 UDP 1685987071 192.168.1.7 49203 typ host	[RFC5245]
a=candidate:1 2 UDP 1685987071 98.248.92.77 49203 typ srflx raddr 192.168.1.7 rport 49203	[RFC5245]
a=setup:active	[RFC4145]
a=rtcp-mux	[RFC5576]
a=bundle-only	[UNIFIED-PLAN]
a=rtcp-rsize	[RFC5506]

Table 27: 5.3.1 SDP Anwer

5.3.2. Successful SVC Video Session

This section shows an SDP Offer/Answer for a session with an audio and a single video source. The video source is encoded as layered coding at 3 different resolutions based on [RFC5583]. The video m=line shows 3 streams with the last stream (payload 100) dependent on streams with payload 96 and 97 for decoding.

SVC Session - 3 Layers w/BUNDLE

```
Alice                                                Bob
|                                                     |
|                                                     |
|      Alice offers 3 sendonly video streams          |
|      as 3 layers of SVC and bundle-only             |
|               for video streams.                    |
|                                                     |
|            Offer(Video:H.264 SVC)                   |
|---------------------------------------------------->|
|                                                     |Bob accepts
|                                                     |Alice's
|                                                     |offered
|                                                     |codec
|                                                     |operation
|             Answer(Video:H.264)                     |points
|<----------------------------------------------------|
|                                                     |
|       One-Way  H.264 SVC video streams              |
|.....................................................|
|                                                     |
|                                                     |
```

SDP Contents	RFC#/Notes
v=0	[RFC4566]
o=- 20519 0 IN IP4 0.0.0.0	[RFC4566]
s=-	[RFC4566]
t=0 0	[RFC4566]
a=msid-semantic:WMS ma	[I-D.ietf-mmusic-msid]
a=group:BUNDLE m0 m1	[I-D.ietf-mmusic-sdp-bundle-negotiation] Alice supports grouping of m=lines under BUNDLE semantics
a=ice-options:trickle	[I-D.ietf-mmusic-trickle-ice]
m=audio 54609 UDP/TLS/RTP/SAVPF 109	[RFC4566]
c=IN IP4 24.23.204.141	[RFC4566]
a=msid:ma ta	Identifies RTCMediaStream ID (ma) and RTCMediaStreamTrack ID (ta)
a=rtcp-mux	[RFC5761]
a=rtcp:54609 IN IP4 24.23.204.141	[RFC3605]

a=mid:m0	[RFC5888] Audio m=line part of BUNDLE group with a unique port number
a=rtpmap:109 opus/48000/2	[I-D.ietf-payload-rtp-opus]
a=extmap:1 urn:ietf:params:rtp-hdrext:ssrc-audio-level	[RFC6464]
a=ptime:20	[I-D.ietf-payload-rtp-opus]
a=sendonly	[RFC3264]
a=rtcp-fb:109 nack	[RFC5104]
a=setup:actpass	[RFC4145]
a=ice-ufrag:074c6550	[RFC5245]
a=ice-pwd:a28a397a4c3f31747d1ee3474af08a068	[RFC5245]
a=fingerprint:sha-1 99:41:49:83:4a:97:0e:1f:ef:6d:f7:c9:c7:70:9d:1f:66:79:a8:07	[RFC5245]
a=candidate:0 1 UDP 2122194687 192.168.1.4 54609 typ host	[RFC5245]
a=candidate:0 2 UDP 2122194687 192.168.1.4 54609 typ host	[RFC5245]
a=candidate:1 1 UDP 1685987071 24.23.204.141 54609 typ srflx raddr 192.168.1.4 rport 54609	[RFC5245]
a=candidate:1 2 UDP 1694302206 24.23.204.141 54609 typ srflx raddr 192.168.1.4 rport 54609	[RFC5245]
a=ssrc:67890 cname:axzo1278npDlAzM73	[RFC5576]
a=rtcp-rsize	[RFC5506]
m=video 0 UDP/TLS/RTP/SAVPF 96 97 100	bundle-only video line with port number set to zero
c=IN IP4 24.23.204.141	[RFC4566]
a=msid:ma tb	Identifies RTCMediaStream ID (ma) and RTCMediaStreamTrack ID (tc)
a=rtcp-mux	[RFC5761]
a=rtcp:0 IN IP4 24.23.204.141	[RFC3605]
a=mid:m1	[RFC5888] Audio m=line part of BUNDLE group
a=msid:ma tb	
a=rtpmap:96 H264/90000	[RFC3984]
a=fmtp:96 profile-level-id=4d0028; packetization-mode=1;max-fr=30;max-fs=8040	[RFC3984]H.264 Layer 1
a=rtpmap:97 H264/90000	[RFC3984]
a=fmtp:97 profile-level-id=4d0028;packetization-mode=1; max-fr=15;max-fs=1200	[RFC3984] H.264 Layer 2

a=rtpmap:100 H264-SVC/90000	[RFC3984]
a=fmtp:100 profile-level-id=4d0028;packetization-mode=1;max-fr=30;max-fs=8040	[RFC3984]
a=depend:100 lay m1:96,97;	[RFC5583]Layer 3 dependent on layers 1 and 2
a=sendonly	[RFC3264] - Send only video stream
a=bundle-only	[UNIFIED-PLAN]
a=ssrc:1732846380 cname:axzo1278npDlAzM73	[RFC5576]
a=ssrc:1732846381 cname:axzo1278npDlAzM73	[RFC5576]
a=ssrc:1732846382 cname:axzo1278npDlAzM73	[RFC5576]
a=rtcp-fb:* nack	[RFC5104]
a=rtcp-fb:* nack pli	[RFC5104]
a=rtcp-fb:* ccm fir	[RFC5104]
a=rtcp-rsize	[RFC5506]

Table 28: 5.3.2 SDP Offer with SVC

SDP Contents	RFC#/Notes
v=0	[RFC4566]
o=- 20519 0 IN IP4 0.0.0.0	[RFC4566]
s=-	[RFC4566]
t=0 0	[RFC4566]
a=msid-semantic:WMS ma	[I-D.ietf-mmusic-msid]
a=group:BUNDLE m0 m1	[I-D.ietf-mmusic-sdp-bundle-negotiation]
a=ice-options:trickle	[I-D.ietf-mmusic-trickle-ice]
m=audio 49203 UDP/TLS/RTP/SAVPF 109	[RFC4566]
c=IN IP4 98.248.92.77	[RFC4566]
a=msid:ma ta	Identifies RTCMediaStream ID (ma) and RTCMediaStreamTrack ID (ta)
a=mid:m0	[RFC5888]
a=rtpmap:109 opus/48000/2	[I-D.ietf-payload-rtp-opus]
a=extmap:1 urn:ietf:params:rtp-hdrext:ssrc-audio-level	[RFC6464]
a=ptime:20	[I-D.ietf-payload-rtp-opus]
a=rtcp-fb:109 nack	[RFC5104]
a=recvonly	[RFC3264]
a=setup:active	[RFC4145]

```
| a=rtcp-mux                        | [RFC5761]                       |
| a=ice-ufrag:074c6550              | [RFC5245]                       |
| a=ice-pwd:a28a397a4c3f31747d1ee3 | [RFC5245]                       |
| 474af08a068                       |                                 |
| a=fingerprint:sha-1 99:41:49:83:  | [RFC5245]                       |
| 4a:97:0e:1f:ef:6d:f7:c9:c7:70:    |                                 |
| 9d:1f:66:79:a8:07                 |                                 |
| a=candidate:0 2 UDP 2122194687    | [RFC5245]                       |
| 192.168.1.7 49203 typ host        |                                 |
| a=candidate:1 2 UDP  1685987071   | [RFC5245]                       |
| 98.248.92.77 49203 typ srflx      |                                 |
| raddr 192.168.1.5 rport 49203     |                                 |
| a=rtcp-rsize                      | [RFC5506]                       |
| m=video 49203 UDP/TLS/RTP/SAVPF   | BUNDLE accepted Bundle address  |
| 96 100                            | same as audio m=line.           |
| c=IN IP4 98.248.92.77             | [RFC4566]                       |
| a=msid:ma tb                      | Identifies RTCMediaStream ID    |
|                                   | (ma) and RTCMediaStreamTrack    |
|                                   | ID (tb)                         |
| a=mid:m1                          | [RFC5888] Video m=line part of  |
|                                   | BUNDLE group                    |
| a=rtpmap:96 H264/90000            | [RFC3984]                       |
| a=fmtp:96 profile-level-          | [RFC3984]H.264 Layer 1          |
| id=4d0028;packetization-mode=1;   |                                 |
| max-fr=30;max-fs=8040             |                                 |
| a=rtpmap:100 H264-SVC/90000       | [RFC3984]                       |
| a=fmtp:100 profile-level-         | [RFC3984]                       |
| id=4d0028;packetization-mode=1;   |                                 |
| max-fr=30;max-fs=8040             |                                 |
| a=depend:100 lay m1:96;           | [RFC5583] Bob chooses 2 Codec   |
|                                   | Operation points                |
| a=ice-ufrag:074c6550              | [RFC5245]                       |
| a=ice-pwd:a28a397a4c3f31747d1ee3 | [RFC5245]                       |
| 474af08a068                       |                                 |
| a=fingerprint:sha-1 99:41:49:83:  | [RFC5245]                       |
| 4a:97:0e:1f:ef:6d:f7:c9:c7:70:    |                                 |
| 9d:1f:66:79:a8:07                 |                                 |
| a=candidate:0 2 UDP 2122194687    | [RFC5245]                       |
| 192.168.1.5 49203 typ host        |                                 |
| a=candidate:1 2 UDP  1685987071   | [RFC5245]                       |
| 24.23.204.142 49203 typ srflx     |                                 |
| raddr 192.168.1.5 rport 49203     |                                 |
| a=recvonly                        | [RFC3264] - Receive only video  |
|                                   | stream                          |
| a=setup:active                    | [RFC4145]                       |
| a=rtcp-mux                        | [RFC5761]                       |
| a=bundle-only                     | [UNIFIED-PLAN]                  |
| a=ssrc:4638117328                 | [RFC5576]                       |
```

```
| cname:axzo1278npDlAzM73          |                                |
| a=rtcp-rsize                     | [RFC5506]                      |
+----------------------------------+--------------------------------+
```

Table 29: 5.3.2 SDP Answer with SVC

5.3.3. Successful Simulcast Video Session with Retransmission

This section shows an SDP Offer/Answer exchange for a simulcast scenario with 2 two resolutions and has [RFC4588] style re-transmission flows.

Simulcast Streams with Retransmission

```
Alice                                                      Bob
|                                                           |
|                                                           |
|Alice offers single audio and simulcasted video streams    |
|                                                           |
|                                                           |
|    Offer(Audio:Opus Video:VP8 with 2 resolutions)         |
|    & RTX stream                                           |
|---------------------------------------------------------->|
|                                                           |
|                                                           |
|          Answer (Bob accepts Alice's offer)               |
|<----------------------------------------------------------|
|                                                           |
|                                                           |
|One-Way 1 Opus, 2 VP8 and RTX video streams,all muxed      |
|...........................................................|
|                                                           |
|                                                           |
```

```
+----------------------------------+--------------------------------+
| SDP Contents                     | RFC#/Notes                     |
+----------------------------------+--------------------------------+
| v=0                              | [RFC4566]                      |
| o=- 20519 0 IN IP4 0.0.0.0       | [RFC4566]                      |
| s=-                              | [RFC4566]                      |
| t=0 0                            | [RFC4566]                      |
| a=msid-semantic:WMS ma           | [I-D.ietf-mmusic-msid]         |
| a=group:BUNDLE m0 m1             | [I-D.ietf-mmusic-sdp-bundle-ne |
|                                  | gotiation] Alice supports      |
|                                  | grouping of m=lines under      |
```

	BUNDLE semantics
a=ice-options:trickle	[I-D.ietf-mmusic-trickle-ice]
m=audio 54609 UDP/TLS/RTP/SAVPF 109	[RFC4566]
c=IN IP4 24.23.204.141	[RFC4566]
a=msid:ma ta	Identifies RTCMediaStream ID (ma) and RTCMediaStreamTrack ID (ta)
a=rtcp:54609 IN IP4 24.23.204.141	[RFC3605]
a=mid:m0	[RFC5888] Audio m=line part of BUNDLE group with a unique port number
a=rtpmap:109 opus/48000/2	[I-D.ietf-payload-rtp-opus]
a=extmap:1 urn:ietf:params:rtp-hdrext:ssrc-audio-level	[RFC6464]
a=ptime:20	[I-D.ietf-payload-rtp-opus]
a=rtcp-fb:109 nack	[RFC5104]
a=sendonly	[RFC3264]
a=setup:actpass	[RFC4145]
a=rtcp-mux	[RFC5761]
a=ssrc:11111 cname:EocUG1f0fcg/yvY7	[RFC5576]
a=ice-ufrag:074c6550	[RFC5245]
a=ice-pwd:a28a397a4c3f31747d1ee3474af08a068	[RFC5245]
a=fingerprint:sha-1 99:41:49:83:4a:97:0e:1f:ef:6d:f7:c9:c7:70:9d:1f:66:79:a8:07	[RFC5245]
a=candidate:0 1 UDP 2122194687 192.168.1.4 54609 typ host	[RFC5245]
a=candidate:0 2 UDP 2122194687 192.168.1.4 54609 typ host	[RFC5245]
a=candidate:1 1 UDP 1685987071 24.23.204.141 54609 typ srflx raddr 192.168.1.4 rport 54609	[RFC5245]
a=candidate:1 2 UDP 1685987071 24.23.204.141 54609 typ srflx raddr 192.168.1.4 rport 54609	[RFC5245]
a=rtcp-rsize	[RFC5506]
m=video 0 UDP/TLS/RTP/SAVPF 98 100 101 103	bundle-only video line with port number set to zero
c=IN IP4 24.23.204.141	[RFC4566]
a=msid:ma tb	Identifies RTCMediaStream ID (ma) and RTCMediaStreamTrack ID (tb)
a=rtcp:0 IN IP4 24.23.204.141	[RFC3605]
a=mid:m1	[RFC5888]

a=rtpmap:98 VP8/90000	[I-D.ietf-payload-vp8]
a=rtpmap:100 VP8/90000	[I-D.ietf-payload-vp8]
a=rtpmap:101 VP8/90000	[I-D.ietf-payload-vp8]
a=rtpmap:103 VP8/90000	[I-D.ietf-payload-vp8]
a=fmtp:98 max-fr=30;max-fs=8040	[RFC4566]
a=fmtp:100 max-fr=15;max-fs=1200	[RFC4566]
a=fmtp:101 apt=98;rtx-time=3000	[RFC4588]
a=fmtp:103 apt=100;rtx-time=3000	[RFC4588]
a=simulast: send 98;100	[I-D.ietf-mmusic-sdp-simulcast]
a=ssrc-group:FID 12345 34567	[RFC5888]
a=ssrc-group:FID 78990 90887	[RFC5888]
a=ssrc:12345 cname:Q/NWs1ao1HmN4Xa5	[RFC5576]
a=ssrc:78990 cname:Q/NWs1ao1HmN4Xa5	[RFC5576]
a=ssrc:34567 cname:Q/NWs1ao1HmN4Xa5	[RFC5576]
a=ssrc:90887 cname:Q/NWs1ao1HmN4Xa5	[RFC5576]
a=sendonly	[RFC3264]
a=rtcp-mux	[RFC5761]
a=bundle-only	[UNIFIED-PLAN]
a=rtcp-fb:* nack	[RFC5104]
a=rtcp-fb:* nack pli	[RFC5104]
a=rtcp-fb:* ccm fir	[RFC5104]
a=rtcp-rsize	[RFC5506]

Table 30: 5.3.3 SDP Offer w/Simulcast, RTX

SDP Contents	RFC#/Notes
v=0	[RFC4566]
o=- 20519 0 IN IP4 0.0.0.0	[RFC4566]
s=-	[RFC4566]
t=0 0	[RFC4566]
a=msid-semantic:WMS ma	[I-D.ietf-mmusic-msid]
a=group:BUNDLE m0 m1	[I-D.ietf-mmusic-sdp-bundle-negotiation] Alice supports grouping of m=lines under BUNDLE semantics
a=ice-options:trickle	[I-D.ietf-mmusic-trickle-ice]
m=audio 49203 UDP/TLS/RTP/SAVPF 109	[RFC4566]
c=IN IP4 98.248.92.77	[RFC4566]
a=msid:ma ta	Identifies RTCMediaStream ID

	(ma) and RTCMediaStreamTrack ID (ta)
a=mid:m0	[RFC5888]
a=rtpmap:109 opus/48000/2	[I-D.ietf-payload-rtp-opus]
a=extmap:1 urn:ietf:params:rtp-hdrext:ssrc-audio-level	[RFC6464]
a=ptime:20	[I-D.ietf-payload-rtp-opus]
a=rtcp-fb:109 nack	[RFC5104]
a=recvonly	[RFC3264]
a=setup:active	[RFC4145]
a=rtcp-mux	[RFC5761]
a=ssrc:33333 cname:L/HmN4Xa5NWs1ao1	[RFC5576]
a=ice-ufrag:074c6550	[RFC5245]
a=ice-pwd:a28a397a4c3f31747d1ee3474af08a068	[RFC5245]
a=fingerprint:sha-1 99:41:49:83:4a:97:0e:1f:ef:6d:f7:c9:c7:70:9d:1f:66:79:a8:07	[RFC5245]
a=candidate:0 2 UDP 2122194687 192.168.1.7 49203 typ host	[RFC5245]
a=candidate:1 2 UDP 1685987071 98.248.92.77 49203 typ srflx raddr 192.168.1.7 rport 49203	[RFC5245]
a=rtcp-rsize	[RFC5506]
m=video 49203 UDP/TLS/RTP/SAVPF 98 100 101 103	BUNDLE accepted with Bundle address identical to audio m-line
c=IN IP4 98.248.92.77	[RFC4566]
a=msid:ma tb	Identifies RTCMediaStream ID (ma) and RTCMediaStreamTrack ID (tb)
a=mid:m1	[RFC5888] Video m=line part of BUNDLE group
a=rtpmap:98 VP8/90000	[I-D.ietf-payload-vp8]
a=rtpmap:100 VP8/90000	[I-D.ietf-payload-vp8]
a=rtpmap:101 VP8/90000	[I-D.ietf-payload-vp8]
a=rtpmap:103 VP8/90000	[I-D.ietf-payload-vp8]
a=fmtp:98 max-fr=30;max-fs=8040	[RFC4566]
a=fmtp:100 max-fr=15;max-fs=1200	[RFC4566]
a=fmtp:101 apt=98;rtx-time=3000	[RFC4588]
a=fmtp:103 apt=100;rtx-time=3000	[RFC4588]
a=ice-ufrag:074c6550	[RFC5245]
a=ice-pwd:a28a397a4c3f31747d1ee3474af08a068	[RFC5245]
a=fingerprint:sha-1 99:41:49:83:4a:97:0e:1f:ef:6d:f7:c9:c7:70:9d:1f:66:79:a8:07	[RFC5245]

a=candidate:0 2 UDP 2122194687 192.168.1.7 49203 typ host	[RFC5245]
a=candidate:1 2 UDP 1685987071 98.248.92.772 49203 typ srflx raddr 192.168.1.7 rport 49203	[RFC5245]
a=simulcast: recv 98;100	[I-D.ietf-mmusic-sdp-simulcast]
a=recvonly	[RFC3264]
a=setup:active	[RFC4145]
a=rtcp-mux	[RFC5761]
a=bundle-only	[UNIFIED-PLAN]
a=rtcp-fb:* nack	[RFC5104]
a=rtcp-fb:* nack pli	[RFC5104]
a=rtcp-fb:* ccm fir	[RFC5104]
a=rtcp-rsize	[RFC5506]

Table 31: 5.3.3 SDP Answer w/Simulcast, RTX

5.3.4. Successful 1-way Simulcast Sessio with 2 resolutions and RTX - One resolution rejected

This section shows an SDP Offer/Answer exchange for a simulcast scenario with 2 two resolutions.

It also showcases when Bob rejects one of the Simulcast Video Stream which results in the rejection of the associated repair stream implicitly.

Simulcast Streams with Retransmission Rejected

```
 Alice                                                     Bob
 |                                                          |
 |                                                          |
 |Alice offers single audio and simulcasted video streams   |
 | with bundle-only for video                               |
 |                                                          |
 |                                                          |
 |Offer(Audio:Opus Video:VP8 with 2 resolutions,RTX Stream)|
 |---------------------------------------------------------->|
 |                                                          |Bob accepts
 |                                                          |1 simulcast
 |                                                          |rtx, rejects
 |                                                          |the other
 |                                                          |
 |   Answer(Audio:Opus Video:VP8 with 1 res & RTX Stream)   |
 |<----------------------------------------------------------|
 |                                                          |
 |                                                          |
 |1-way audio,video session and its associated RTX stream,  |
 | all multiplexed                                          |
 |..........................................................|
 |                                                          |
 |                                                          |
```

```
  +----------------------------------+-------------------------------+
  | SDP Contents                     | RFC#/Notes                    |
  +----------------------------------+-------------------------------+
  | v=0                              | [RFC4566]                     |
  | o=- 20519 0 IN IP4 0.0.0.0       | [RFC4566]                     |
  | s=-                              | [RFC4566]                     |
  | t=0 0                            | [RFC4566]                     |
  | a=msid-semantic:WMS ma           | [I-D.ietf-mmusic-msid]        |
  | a=group:BUNDLE m0 m1             | [I-D.ietf-mmusic-sdp-bundle-ne |
  |                                  | gotiation] Alice supports     |
  |                                  | grouping of m=lines under     |
  |                                  | BUNDLE semantics              |
  | a=ice-options:trickle            | [I-D.ietf-mmusic-trickle-ice] |
  | m=audio 54609 UDP/TLS/RTP/SAVPF  | [RFC4566]                     |
  | 109                              |                               |
  | c=IN IP4 24.23.204.141           | [RFC4566]                     |
  | a=msid:ma ta                     | Identifies RTCMediaStream ID  |
  |                                  | (ma) and RTCMediaStreamTrack  |
  |                                  | ID (ta)                       |
  | a=rtcp:54609 IN IP4              | [RFC3605]                     |
```

```
| 24.23.204.141                      |                                |
| a=mid:m0                           | [RFC5888]                      |
| a=rtpmap:109 opus/48000/2          | [I-D.ietf-payload-rtp-opus]    |
| a=extmap:1 urn:ietf:params:rtp-    | [RFC6464]                      |
| hdrext:ssrc-audio-level            |                                |
| a=ptime:20                         | [I-D.ietf-payload-rtp-opus]    |
| a=rtcp-fb:109 nack                 | [RFC5104]                      |
| a=sendonly                         | [RFC3264]                      |
| a=setup:actpass                    | [RFC4145]                      |
| a=rtcp-mux                         | [RFC5761]                      |
| a=ssrc:11111                       | [RFC5576]                      |
| cname:LP/NWs1ao1HmN4Xa5            |                                |
| a=ice-ufrag:074c6550               | [RFC5245]                      |
| a=ice-pwd:a28a397a4c3f31747d1ee3 | [RFC5245]                      |
| 474af08a068                        |                                |
| a=fingerprint:sha-1 99:41:49:83:   | [RFC5245]                      |
| 4a:97:0e:1f:ef:6d:f7:c9:c7:70:     |                                |
| 9d:1f:66:79:a8:07                  |                                |
| a=candidate:0 1 UDP  2122194687    | [RFC5245]                      |
| 192.168.1.4 54609 typ host         |                                |
| a=candidate:0 2 UDP 2122194687     | [RFC5245]                      |
| 192.168.1.4 54609 typ host         |                                |
| a=candidate:1 1 UDP  1685987071    | [RFC5245]                      |
| 24.23.204.141 54609 typ srflx      |                                |
| raddr 192.168.1.4 rport 54609      |                                |
| a=candidate:1 2 UDP  1685987071    | [RFC5245]                      |
| 24.23.204.141 54609 typ srflx      |                                |
| raddr 192.168.1.4 rport 54609      |                                |
| a=rtcp-rsize                       | [RFC5506]                      |
| m=video 0 UDP/TLS/RTP/SAVPF 98     | bundle-only video line with    |
| 100 101 103                        | port number set to zero        |
| c=IN IP4 24.23.204.141             | [RFC4566]                      |
| a=msid:ma tb                       | Identifies RTCMediaStream ID   |
|                                    | (ma) and RTCMediaStreamTrack   |
|                                    | ID (tb)                        |
| a=rtcp:0 IN IP4 24.23.204.141      | [RFC3605]                      |
| a=mid:m1                           | [RFC5888]                      |
| a=rtpmap:98 VP8/90000              | [I-D.ietf-payload-vp8]         |
| a=rtpmap:100 VP8/90000             | [I-D.ietf-payload-vp8]         |
| a=rtpmap:101 VP8/90000             | [I-D.ietf-payload-vp8]         |
| a=rtpmap:103 VP8/90000             | [I-D.ietf-payload-vp8]         |
| a=fmtp:98 max-fr=30;max-fs=8040    | [RFC4566]                      |
| a=fmtp:100 max-fr=15;max-fs=1200   | [RFC4566]                      |
| a=fmtp:101 apt=98;rtx-time=3000    | [RFC4588]                      |
| a=fmtp:103 apt=100;rtx-time=3000   | [RFC4588]                      |
| a=simulcast: send 98;100           | [I-D.ietf-mmusic-sdp-simulcast |
|                                    | ]                              |
| a=ssrc-group:FID 12345 34567       | [RFC5888]                      |
```

a=ssrc-group:FID 78990 90887	[RFC5888]
a=ssrc:12345	[RFC5576]
cname:Q/NWs1ao1HmN4Xa5	
a=ssrc:78990	[RFC5576]
cname:Q/NWs1ao1HmN4Xa5	
a=ssrc:34567	[RFC5576]
cname:Q/NWs1ao1HmN4Xa5	
a=ssrc:90887	[RFC5576]
cname:Q/NWs1ao1HmN4Xa5	
a=sendonly	[RFC3264]
a=rtcp-mux	[RFC5761]
a=bundle-only	[UNIFIED-PLAN]
a=rtcp-fb:* nack	[RFC5104]
a=rtcp-fb:* nack pli	[RFC5104]
a=rtcp-fb:* ccm fir	[RFC5104]
a=rtcp-rsize	[RFC5506]

Table 32: 5.3.4 SDP Offer w/Simulcast, RTX

SDP Contents	RFC#/Notes
v=0	[RFC4566]
o=- 20519 0 IN IP4 0.0.0.0	[RFC4566]
s=-	[RFC4566]
t=0 0	[RFC4566]
a=msid-semantic:WMS ma	[I-D.ietf-mmusic-msid]
a=group:BUNDLE m0 m1	[I-D.ietf-mmusic-sdp-bundle-negotiation] Alice supports grouping of m=lines under BUNDLE semantics
a=ice-options:trickle	[I-D.ietf-mmusic-trickle-ice]
m=audio 49203 UDP/TLS/RTP/SAVPF 109	[RFC4566]
c=IN IP4 98.248.92.77	[RFC4566]
a=msid:ma ta	Identifies RTCMediaStream ID (ma) and RTCMediaStreamTrack ID (ta)
a=mid:m0	[RFC5888]
a=rtpmap:109 opus/48000/2	[I-D.ietf-payload-rtp-opus]
a=extmap:1 urn:ietf:params:rtp-hdrext:ssrc-audio-level	[RFC6464]
a=ptime:20	[I-D.ietf-payload-rtp-opus]
a=recvonly	[RFC3264]
a=setup:active	[RFC4145]
a=rtcp-mux	[RFC5761]
a=ice-ufrag:074c6550	[RFC5245]

a=ice-pwd:a28a397a4c3f31747d1ee3 474af08a068	[RFC5245]
a=fingerprint:sha-1 99:41:49:83: 4a:97:0e:1f:ef:6d:f7:c9:c7:70: 9d:1f:66:79:a8:07	[RFC5245]
a=candidate:0 2 UDP 2122194687 192.168.1.7 49203 typ host	[RFC5245]
a=candidate:1 2 UDP 1685987071 98.248.92.77 49203 typ srflx raddr 192.168.1.7 rport 49203	[RFC5245]
a=rtcp-rsize	[RFC5506]
m=video 49203 UDP/TLS/RTP/SAVPF 98 101	BUNDLE accepted with Bundle address identical to audio m-line
c=IN IP4 98.248.92.77	[RFC4566]
a=msid:ma tb	Identifies RTCMediaStream ID (ma) and RTCMediaStreamTrack ID (tb)
a=mid:m1	[RFC5888]
a=rtpmap:98 VP8/90000	[I-D.ietf-payload-vp8]
a=rtpmap:101 VP8/90000	[I-D.ietf-payload-vp8]
a=fmtp:98 max-fr=30;max-fs=8040	[RFC4566]
a=fmtp:101 apt=98;rtx-time=3000	[RFC4588]
a=ice-ufrag:074c6550	[RFC5245]
a=ice-pwd:a28a397a4c3f31747d1ee3 474af08a068	[RFC5245]
a=fingerprint:sha-1 99:41:49:83: 4a:97:0e:1f:ef:6d:f7:c9:c7:70: 9d:1f:66:79:a8:07	[RFC5245]
a=candidate:0 2 UDP 2122194687 192.168.1.7 49203 typ host	[RFC5245]
a=candidate:1 2 UDP 1685987071 98.248.92.77 49203 typ srflx raddr 192.168.1.5 rport 49203	[RFC5245]
a=simulcast: recv 98	[I-D.ietf-mmusic-sdp-simulcast] Bob accepts only one simulcast resolution
a=ssrc:54321 cname:NWs1ao1HmN4Xa5	[RFC5576]
a=recvonly	[RFC3264]
a=setup:active	[RFC4145]
a=rtcp-mux	[RFC5761]
a=bundle-only	[UNIFIED-PLAN]
a=rtcp-rsize	[RFC5506]

Table 33: 5.3.4 SDP Answer no Simulcast

5.3.5. Simulcast Video Session with Forward Error Correction

This section shows an SDP Offer/Answer exchange for Simulcast video stream at two resolutions and and has [RFC5956] style FEC flows.

On completion of the Offer/Answer exchange mechanism we end up one audio stream, 2 simulcast video streams and 2 associated FEC streams are sent over a single 5-tuple.

Simulcast Streams with Forward Error Correction

```
Alice                                                            Bob
|                                                               |
|                                                               |
|                                                               |
|Alice offers single audio and simulcasted video streams        |
|with bundle-only                                               |
|                                                               |
|                                                               |
|Offer(Audio:Opus Video:VP8 with 2 resolutions with FEC Streams)|
|-------------------------------------------------------------->|
|                                                               |
|                                                               |Bob
|                                                               |accepts
|                                                               |Alice's
|                                                               |offer
|Answer(Audio:Opus Video:VP8 with 2 resolutions w/FEC Streams)  |
|<--------------------------------------------------------------|
|                                                               |
|One-Way Audio,Video session with 4 video streams(Simulcast     |
| and FEC) all multiplexed                                      |
|...............................................................|
|                                                               |
|                                                               |
|                                                               |
```

SDP Contents	RFC#/Notes
v=0	[RFC4566]
o=- 20519 0 IN IP4 0.0.0.0	[RFC4566]
s=-	[RFC4566]
t=0 0	[RFC4566]
a=msid-semantic:WMS ma	[I-D.ietf-mmusic-msid]
a=group:BUNDLE m0 m1	[I-D.ietf-mmusic-sdp-bundle-negotiation] Alice supports

```
   |                                  | grouping of m=lines under      |
   |                                  | BUNDLE semantics               |
   | a=ice-options:trickle            | [I-D.ietf-mmusic-trickle-ice]  |
   | m=audio 54609 UDP/TLS/RTP/SAVPF  | [RFC4566]                      |
   | 109                              |                                |
   | c=IN IP4 24.23.204.141           | [RFC4566]                      |
   | a=msid:ma ta                     | Identifies RTCMediaStream ID   |
   |                                  | (ma) and RTCMediaStreamTrack   |
   |                                  | ID (ta)                        |
   | a=rtcp:54609 IN IP4              | [RFC3605]                      |
   | 24.23.204.141                    |                                |
   | a=mid:m0                         | [RFC5888]                      |
   | a=rtpmap:109 opus/48000/2        | [I-D.ietf-payload-rtp-opus]    |
   | a=extmap:1 urn:ietf:params:rtp-  | [RFC6464]                      |
   | hdrext:ssrc-audio-level          |                                |
   | a=ptime:20                       | [I-D.ietf-payload-rtp-opus]    |
   | a=rtcp-fb:109 nack               | [RFC5104]                      |
   | a=sendonly                       | [RFC3264]                      |
   | a=setup:actpass                  | [RFC4145]                      |
   | a=rtcp-mux                       | [RFC5761]                      |
   | a=ssrc:11111                     | [RFC5576]                      |
   | cname:Q/NWs1ao1HmN4Xa5           |                                |
   | a=ice-ufrag:074c6550             | [RFC5245]                      |
   | a=ice-pwd:a28a397a4c3f31747d1ee3 | [RFC5245]                      |
   | 474af08a068                      |                                |
   | a=fingerprint:sha-1 99:41:49:83: | [RFC5245]                      |
   | 4a:97:0e:1f:ef:6d:f7:c9:c7:70:   |                                |
   | 9d:1f:66:79:a8:07                |                                |
   | a=candidate:0 1 UDP  2122194687  | [RFC5245]                      |
   | 192.168.1.4 54609 typ host       |                                |
   | a=candidate:0 2 UDP 2122194687   | [RFC5245]                      |
   | 192.168.1.4 54609 typ host       |                                |
   | a=candidate:1 1 UDP  1685987071  | [RFC5245]                      |
   | 24.23.204.141 54609 typ srflx    |                                |
   | raddr 192.168.1.4 rport 54609    |                                |
   | a=candidate:1 2 UDP  1685987071  | [RFC5245]                      |
   | 24.23.204.141 54609 typ srflx    |                                |
   | raddr 192.168.1.4 rport 54609    |                                |
   | a=rtcp-rsize                     | [RFC5506]                      |
   | m=video 0 UDP/TLS/RTP/SAVPF 98   | bundle-only video line with    |
   | 100 101 103                      | port number set to zero        |
   | c=IN IP4 24.23.204.141           | [RFC4566]                      |
   | a=msid:ma tb                     | Identifies RTCMediaStream ID   |
   |                                  | (ma) and RTCMediaStreamTrack   |
   |                                  | ID (tb)                        |
   | a=rtcp:0 IN IP4 24.23.204.141    | [RFC3605]                      |
   | a=mid:m1                         | [RFC5888] Video m=line part of |
   |                                  | BUNDLE group                   |
```

a=rtpmap:98 VP8/90000	[I-D.ietf-payload-vp8]
a=rtpmap:100 VP8/90000	[I-D.ietf-payload-vp8]
a=rtpmap:101 1d-interleaved-parityfec/90000	[RFC5956]
a=rtpmap:103 1d-interleaved-parityfec/90000	[RFC5956]
a=fmtp:98 max-fr=30;max-fs=8040	[RFC4566]
a=fmtp:100 max-fr=15;max-fs=1200	[RFC4566]
a=fmtp:101 L=5; D=10; repair-window=200000	[RFC5956]
a=fmtp:103 L=5; D=10; repair-window=200000	[RFC5956]
a=simulcast: send 98;100	[I-D.ietf-mmusic-sdp-simulcast]
a=depend:98 fec m1:101	TBD
a=depend:100 fec m1:103	TBD
a=ssrc-group:FEC-FR 12345 34567	[RFC5888]
a=ssrc-group:FEC-FR 78990 90887	[RFC5888]
a=ssrc:12345 cname:Q/NWs1ao1HmN4Xa5	[RFC5576]
a=ssrc:78990 cname:Q/NWs1ao1HmN4Xa5	[RFC5576]
a=ssrc:34567 cname:Q/NWs1ao1HmN4Xa5	[RFC5576]
a=ssrc:90887 cname:Q/NWs1ao1HmN4Xa5	[RFC5576]
a=sendonly	[RFC3264]
a=rtcp-mux	[RFC5761]
a=bundle-only	[UNIFIED-PLAN]
a=rtcp-fb:* nack	[RFC5104]
a=rtcp-fb:* nack pli	[RFC5104]
a=rtcp-fb:* ccm fir	[RFC5104]
a=rtcp-rsize	[RFC5506]

Table 34: 5.3.5 SDP Offer

SDP Contents	RFC#/Notes
v=0	[RFC4566]
o=- 20519 0 IN IP4 0.0.0.0	[RFC4566]
s=-	[RFC4566]
t=0 0	[RFC4566]
a=msid-semantic:WMS m0	[I-D.ietf-mmusic-msid]
a=group:BUNDLE m0 m1	[I-D.ietf-mmusic-sdp-bundle-negotiation]
a=ice-options:trickle	[I-D.ietf-mmusic-trickle-ice]

m=audio 49203 UDP/TLS/RTP/SAVPF 109	[RFC4566]
c=IN IP4 98.248.92.77	[RFC4566]
a=msid:ma ta	Identifies RTCMediaStream ID (ma) and RTCMediaStreamTrack ID (ta)
a=mid:m0	[RFC5888] Audio m=line part of BUNDLE group with a unique port number
a=rtpmap:109 opus/48000/2	[I-D.ietf-payload-rtp-opus]
a=extmap:1 urn:ietf:params:rtp-hdrext:ssrc-audio-level	[RFC6464]
a=ptime:20	[I-D.ietf-payload-rtp-opus]
a=rtcp-fb:109 nack	[RFC5104]
a=recvonly	[RFC3264]
a=setup:active	[RFC4145]
a=rtcp-mux	[RFC5761]
a=ssrc:33333 cname:Y9/cZke09JAtpl98	[RFC5576]
a=ice-ufrag:074c6550	[RFC5245]
a=ice-pwd:a28a397a4c3f31747d1ee3474af08a068	[RFC5245]
a=fingerprint:sha-1 99:41:49:83:4a:97:0e:1f:ef:6d:f7:c9:c7:70:9d:1f:66:79:a8:07	[RFC5245]
a=candidate:0 2 UDP 2122194687 192.168.1.7 49203 typ host	[RFC5245]
a=candidate:1 2 UDP 1685987071 98.248.92.77 49203 typ srflx raddr 192.168.1.7 rport 49203	[RFC5245]
a=rtcp-rsize	[RFC5506]
m=video 49203 UDP/TLS/RTP/SAVPF 98 100 101 103	BUNDLE accepted with Bundle Address identical to audio m=line.
c=IN IP4 98.248.92.77	[RFC4566]
a=msid:ma tb	Identifies RTCMediaStream ID (ma) and RTCMediaStreamTrack ID (tb)
a=mid:m1	[RFC5888] Video m=line part of BUNDLE group
a=rtpmap:98 VP8/90000	[I-D.ietf-payload-vp8]
a=rtpmap:100 VP8/90000	[I-D.ietf-payload-vp8]
a=rtpmap:101 1d-interleaved-parityfec/90000	[RFC5956]
a=rtpmap:103 1d-interleaved-parityfec/90000	[RFC5956]
a=fmtp:98 max-fr=30;max-fs=8040	[RFC4566]
a=fmtp:100 max-fr=15;max-fs=1200	[RFC4566]

a=fmtp:101 L=5; D=10; repair-window=200000	[RFC5956]
a=fmtp:103 L=5; D=10; repair-window=200000	[RFC5956]
a=simulcast: recv 98;100	[I-D.ietf-mmusic-sdp-simulcast]
a=depend:98 fec m1:101	TBD
a=depend:100 fec m1:103	TBD
a=recvonly	[RFC3264]
a=setup:active	[RFC4145]
a=rtcp-mux	[RFC5761]
a=bundle-only	[UNIFIED-PLAN]
a=ice-ufrag:074c6550	[RFC5245]
a=ice-pwd:a28a397a4c3f31747d1ee3474af08a068	[RFC5245]
a=fingerprint:sha-1 99:41:49:83:4a:97:0e:1f:ef:6d:f7:c9:c7:70:9d:1f:66:79:a8:07	[RFC5245]
a=candidate:0 2 UDP 2122194687 192.168.1.7 49203 typ host	[RFC5245]
a=candidate:1 2 UDP 1685987071 98.248.92.77 49203 typ srflx raddr 192.168.1.7 rport 49203	[RFC5245]
a=rtcp-fb:* nack	[RFC5104]
a=rtcp-fb:* nack pli	[RFC5104]
a=rtcp-fb:* ccm fir	[RFC5104]
a=rtcp-rsize	[RFC5506]

Table 35: 5.3.5 SDP Answer

5.4. Others

The examples in the section provide SDP for a variety of scenarios related to RTP Header extension, Legacy Interop scenarios and more.

5.4.1. Audio Session - Voice Activity Detection

This example shows Alice indicating the support of the RTP header extension to include the audio-level of the audio sample carried in the RTP packet.

2-Way Audio with VAD

```
Alice                                     Bob
|                                          |
|                                          |
|Alice indicates support for including     |
|audio level in RTP header                 |
|                                          |
|     Offer(Audio:Opus,PCMU,PCMA)          |
|----------------------------------------->|
|                                          |
|                                          |
|     Answer(Audio:Opus,PCMU,PCMA)         |
|<-----------------------------------------|
|                                          |Bob
|                                          |accepts &
|                                          |indicates
|                                          |his support
|                                          |support
|     Two way Opus Audio                   |
|.........................................|
|                                          |
|Per packet audio-level is included in the|
|RTP header                                |
|                                          |
```

SDP Contents	RFC#/Notes
v=0	[RFC4566]
o=- 20518 0 IN IP4 0.0.0.0	[RFC4566]
s=-	[RFC4566]
t=0 0	[RFC4566]
a=msid-semantic:WMS ma	[I-D.ietf-mmusic-msid]
a=group:BUNDLE audio	[I-D.ietf-mmusic-sdp-bundle-negotiation]
a=ice-options:trickle	[I-D.ietf-mmusic-trickle-ice]
m=audio 54609 UDP/TLS/RTP/SAVPF 109 0 8	[RFC4566]
c=IN IP4 24.23.204.141	[RFC4566]
a=mid:audio	[RFC5888]
a=msid:ma ta	Identifies RTCMediaStream ID (ma) and RTCMediaStreamTrack ID (ta)
a=rtcp:54609 IN IP4 24.23.204.141	[RFC3605]

a=extmap:1 urn:ietf:params:rtp-hdrext:ssrc-audio-level	[RFC6464]
a=rtpmap:109 opus/48000/2	[I-D.ietf-payload-rtp-opus]
a=ptime:20	[I-D.ietf-payload-rtp-opus]
a=rtpmap:0 PCMU/8000	[RFC3551]
a=rtpmap:0 PCMA/8000	[RFC3551]
a=sendrecv	[RFC3264]
a=setup:actpass	[RFC4145]
a=rtcp-mux	[RFC5761]
a=ice-ufrag:074c6550	[RFC5245]
a=ice-pwd:a28a397a4c3f31747d1ee3474af08a068	[RFC5245]
a=fingerprint:sha-1 99:41:49:83:4a:97:0e:1f:ef:6d:f7:c9:c7:70:9d:1f:66:79:a8:07	[RFC5245]
a=candidate:0 1 UDP 2122194687 192.168.1.4 54609 typ host	[RFC5245]
a=candidate:0 2 UDP 2122194687 192.168.1.4 54609 typ host	[RFC5245]
a=candidate:1 1 UDP 1685987071 24.23.204.141 54609 typ srflx raddr 192.168.1.4 rport 54609	[RFC5245]
a=candidate:1 2 UDP 1685987071 24.23.204.141 54609 typ srflx raddr 192.168.1.4 rport 54609	[RFC5245]
a=rtcp-fb:* nack	[RFC5104]
a=ssrc:11111 cname:QCL/1HmN4Xa5CClapa	[RFC5576]
a=rtcp-rsize	[RFC5506]

Table 36: 5.4.1 SDP Offer

SDP Contents	RFC#/Notes
v=0	[RFC4566]
o=- 16833 0 IN IP4 0.0.0.0	[RFC4566] - Session Origin Information
s=-	[RFC4566]
t=0 0	[RFC4566]
a=msid-semantic:WMS ma	[I-D.ietf-mmusic-msid]
a=group:BUNDLE audio	[I-D.ietf-mmusic-sdp-bundle-negotiation]
a=ice-options:trickle	[I-D.ietf-mmusic-trickle-ice]
m=audio 49203 UDP/TLS/RTP/SAVPF 109 0 98	[RFC4566]
c=IN IP4 98.248.92.77	[RFC4566]

a=mid:audio	[RFC5888]
a=msid:ma ta	Identifies RTCMediaStream ID (ma) and RTCMediaStreamTrack ID (ta)
a=extmap:1 urn:ietf:params:rtp-hdrext:ssrc-audio-level	[RFC6464]
a=rtpmap:109 opus/48000/2	[I-D.ietf-payload-rtp-opus] - Bob accepts only Opus Codec
a=ptime:20	[I-D.ietf-payload-rtp-opus]
a=rtpmap:0 PCMU/8000	[RFC3551] PCMU Audio Codec
a=rtpmap:0 PCMA/8000	[RFC3551] PCMA Audio Codec
a=rtcp-fb:* nack	[RFC5104]
a=sendrecv	[RFC3264] - Bob can send and recv audio
a=setup:active	[RFC4145]
a=rtcp-mux	[RFC5761] - Bob can perform RTP/RTCP Muxing on port 49203
a=ice-ufrag:c300d85b	[RFC5245]
a=ice-pwd:de4e99bd291c325921d5d47efbabd9a2	[RFC5245]
a=fingerprint:sha-1 99:41:49:83:4a:97:0e:1f:ef:6d:f7:c9:c7:70:9d:1f:66:79:a8:07	[RFC5245]
a=candidate:0 1 UDP 2122194687 192.168.1.7 49203 typ host	[RFC5245]
a=candidate:1 1 UDP 1685987071 98.248.92.77 49203 typ srflx raddr 192.168.1.7 rport 49203	[RFC5245]
a=ssrc:1732846380 cname:EocUG1f0fcg/yvY7	[RFC5576]
a=rtcp-rsize	[RFC5506]

Table 37: 5.4.1 SDP Answer

5.4.2. Audio Conference - Voice Activity Detection

This example shows SDP for RTP header extension that allows RTP-level mixers in audio conferences to deliver information about the audio level of individual participants.

Audio Conference with VAD Support

```
   Alice                                      Mixer
   |                                           |
   |Alice indicates her interest to audio      |
   |levels for the contributing sources        |
   |                                           |
   |Offer(Audio:Opus,PCMU,PCMA)                |
   |----------------------------------------->|
   |                                           |
   |                                           |
   |Answer(Audio:Opus,PCMU,PCMA)               |
   |<-----------------------------------------|
   |                                           |Mixer
   |                                           |indicates
   |                                           |it can
   |                                           |support
   |                                           |audio-levels
   |           Two way Opus Audio              |
   |.........................................|
   |                                           |
   |Audio-levels per CSRCS is included in the|
   |RTP header                                 |
   |                                           |
```

SDP Contents	RFC#/Notes
v=0	[RFC4566]
o=- 20518 0 IN IP4 0.0.0.0	[RFC4566] - Session Origin Information
s=-	[RFC4566]
t=0 0	[RFC4566]
a=msid-semantic:WMS ma	[I-D.ietf-mmusic-msid]
a=group:BUNDLE audio	[I-D.ietf-mmusic-sdp-bundle-negotiation]
a=ice-options:trickle	[I-D.ietf-mmusic-trickle-ice]
m=audio 54609 UDP/TLS/RTP/SAVPF 109 0 8	[RFC4566]
c=IN IP4 24.23.204.141	[RFC4566]
a=mid:audio	[RFC5888]
a=msid:ma ta	Identifies RTCMediaStream ID (ma) and RTCMediaStreamTrack ID (ta)
a=rtcp:54609 IN IP4 24.23.204.141	[RFC3605]

a=extmap:1/recvonly	[RFC6465]
urn:ietf:params:rtp-hdrext:csrc-audio-level	
a=extmap:1 urn:ietf:params:rtp-hdrext:ssrc-audio-level	[RFC6464]
a=rtpmap:109 opus/48000/2	[I-D.ietf-payload-rtp-opus]
a=ptime:20	[I-D.ietf-payload-rtp-opus]
a=rtpmap:0 PCMU/8000	[RFC3551] PCMU Audio Codec
a=rtpmap:0 PCMA/8000	[RFC3551] PCMA Audio Codec
a=rtcp-fb:* nack	[RFC5104]
a=sendrecv	[RFC3264] - Alice can send and recv audio
a=setup:actpass	[RFC4145]
a=rtcp-mux	[RFC5761]
a=ice-ufrag:074c6550	[RFC5245]
a=ice-pwd:a28a397a4c3f31747d1ee3474af08a068	[RFC5245]
a=fingerprint:sha-1 99:41:49:83:4a:97:0e:1f:ef:6d:f7:c9:c7:70:9d:1f:66:79:a8:07	[RFC5245]
a=candidate:0 1 UDP 2122194687 192.168.1.4 54609 typ host	[RFC5245]
a=candidate:0 2 UDP 2122194687 192.168.1.4 54609 typ host	[RFC5245]
a=candidate:1 1 UDP 1685987071 24.23.204.141 54609 typ srflx raddr 192.168.1.4 rport 54609	[RFC5245]
a=candidate:1 2 UDP 1685987071 24.23.204.141 54609 typ srflx raddr 192.168.1.4 rport 54609	[RFC5245]
a=ssrc:11111 cname:QCL/1HmN4Xa5CClapa	[RFC5576]
a=rtcp-rsize	[RFC5506]

Table 38: 5.4.2 SDP Offer

SDP Contents	RFC#/Notes
v=0	[RFC4566]
o=- 16833 0 IN IP4 0.0.0.0	[RFC4566] - Session Origin Information
s=-	[RFC4566]
t=0 0	[RFC4566]
a=msid-semantic:WMS ma	[I-D.ietf-mmusic-msid]
a=group:BUNDLE audio	[I-D.ietf-mmusic-sdp-bundle-negotiation]
a=ice-options:trickle	[I-D.ietf-mmusic-trickle-ice]
m=audio 49203 UDP/TLS/RTP/SAVPF 109 0 98	[RFC4566]
c=IN IP4 98.248.92.77	[RFC4566]
a=mid:audio	[RFC5888]
a=msid:ma ta	Identifies RTCMediaStream ID (ma) and RTCMediaStreamTrack ID (ta)
a=extmap:1/sendonly urn:ietf:params:rtp-hdrext:csrc-audio-level	[RFC6465]
a=rtpmap:109 opus/48000/2	[I-D.ietf-payload-rtp-opus]
a=ptime:20	[I-D.ietf-payload-rtp-opus]
a=rtpmap:0 PCMU/8000	[RFC3551] PCMU Audio Codec
a=rtpmap:0 PCMA/8000	[RFC3551] PCMA Audio Codec
a=rtcp-fb:* nack	[RFC5104]
a=sendrecv	[RFC3264]
a=setup:active	[RFC4145]
a=rtcp-mux	[RFC5761]
a=ice-ufrag:c300d85b	[RFC5245]
a=ice-pwd:de4e99bd291c325921d5d47efbabd9a2	[RFC5245]
a=fingerprint:sha-1 99:41:49:83:4a:97:0e:1f:ef:6d:f7:c9:c7:70:9d:1f:66:79:a8:07	[RFC5245]
a=candidate:0 1 UDP 2122194687 192.168.1.7 49203 typ host	[RFC5245]
a=candidate:1 1 UDP 1685987071 98.248.92.77 49203 typ srflx raddr 192.168.1.7 rport 49203	[RFC5245]
a=ssrc:2222 cname:HmN4Xa5CC/lapa	[RFC5576]
a=rtcp-rsize	[RFC5506]

Table 39: 5.4.2 SDP Answer

5.4.3. Successful legacy Interop Fallaback with bundle-only

In the scenario described below, Alice is a multi-stream capable WebRTC endpoint while Bob is a legacy VOIP end-point. The SDP Offer/Answer exchange demonstrates successful session setup with fallback to audio only stream negotiated via bundle-only framework between the end-points. Specifically,

- o Offer from Alice describes 2 cameras via 2 video m=lines with both marked as bundle-only.

- o Since Bob doesnot recognize either the BUNDLE mechanism or the bundle-only attribute, he accepts only the audio stream from Alice.

```
         Successful 2-Way WebRTC <-> VOIP Interop

Alice                                                        Bob
|                                                            |
|                                                            |
|       Alice is a multistream capable WebRTC end-point      |
|          & Bob is behind a legacy VOIP system              |
|                                                            |
|Offer(Audio:Opus Video:2 VP8,2 H2.64 Streams) with          |
|          bundle-only                                       |
|------------------------------------------------------------>|
|     Alice marks both the video streams as bundle-only      |
|                                                            |
|                                                            |
|                  Answer(Audio:Opus)                        |
|<------------------------------------------------------------|
|                                                            |Bob
|                                                            |accepts
|                                                            |audio
|                                                            |stream,
|                                                            |since he
|                                                            |doesn't
|                                                            |recognize
|                                                            |bundle-only
|                                                            |
|                   Two way Opus Audio                       |
|............................................................|
|                                                            |
|                                                            |


   +-------------------------------------+----------------------------+
```

```
| SDP Contents                          | RFC#/Notes                  |
+---------------------------------------+-----------------------------+
| v=0                                   | [RFC4566]                   |
| o=- 20519 0 IN IP4 0.0.0.0            | [RFC4566]                   |
| s=-                                   | [RFC4566]                   |
| t=0 0                                 | [RFC4566]                   |
| a=msid-semantic:WMS ma                | [I-D.ietf-mmusic-msid]      |
| a=group:BUNDLE m0 m1 m2               | [I-D.ietf-mmusic-sdp-bundle |
|                                       | -negotiation] Alice         |
|                                       | supports grouping of        |
|                                       | m=lines under BUNDLE        |
|                                       | semantics                   |
| a=ice-options:trickle                 | [I-D.ietf-mmusic-trickle-ic |
|                                       | e]                          |
| m=audio 54609 UDP/TLS/RTP/SAVPF 109   | [RFC4566]                   |
| c=IN IP4 24.23.204.141                | [RFC4566]                   |
| a=rtcp:54609 IN IP4 24.23.204.141     | [RFC3605]                   |
| a=mid:m0                              | [RFC5888] Audio m=line part |
|                                       | of BUNDLE group with a      |
|                                       | unique port number          |
| a=msid:ma ta                          | Identifies RTCMediaStream   |
|                                       | ID (ma) and                 |
|                                       | RTCMediaStreamTrack ID (ta) |
| a=rtpmap:109 opus/48000/2             | [I-D.ietf-payload-rtp-opus] |
| a=extmap:1 urn:ietf:params:rtp-       | [RFC6464]                   |
| hdrext:ssrc-audio-level               |                             |
| a=ptime:20                            | [I-D.ietf-payload-rtp-opus] |
| a=rtcp-fb:109 nack                    | [RFC5104]                   |
| a=sendrecv                            | [RFC3264]                   |
| a=setup:actpass                       | [RFC4145]                   |
| a=rtcp-mux                            | [RFC5761]                   |
| a=ice-ufrag:074c6550                  | [RFC5245]                   |
| a=ice-pwd:a28a397a4c3f31747d1ee3474   | [RFC5245]                   |
| af08a068                              |                             |
| a=fingerprint:sha-1 99:41:49:83:4a:   | [RFC5245]                   |
| 97:0e:1f:ef:6d:f7:c9:c7:70:           |                             |
| 9d:1f:66:79:a8:07                     |                             |
| a=candidate:0 1 UDP  2122194687       | [RFC5245]                   |
| 192.168.1.4 54609 typ host            |                             |
| a=candidate:0 2 UDP 2122194687        | [RFC5245]                   |
| 192.168.1.4 54609 typ host            |                             |
| a=candidate:1 1 UDP  1685987071       | [RFC5245]                   |
| 24.23.204.141 54609 typ srflx raddr   |                             |
| 192.168.1.4 rport 54609               |                             |
| a=candidate:1 2 UDP  1685987071       | [RFC5245]                   |
| 24.23.204.141 54609 typ srflx raddr   |                             |
| 192.168.1.4 rport 54609               |                             |
| a=ssrc:11111                          | [RFC5576]E                  |
```

cname:axzo1278npDlAzM73	
a=rtcp-rsize	[RFC5506]
m=video 0 UDP/TLS/RTP/SAVPF 98 100	bundle-only video line with port number set to zero
c=IN IP4 24.23.204.141	[RFC4566]
a=rtcp-mux	[RFC5761]
a=rtcp:0 IN IP4 24.23.204.141	[RFC3605]
a=mid:m1	[RFC5888] Video m=line part of BUNDLE group
a=msid:ma tb	Identifies RTCMediaStream ID (ma) and RTCMediaStreamTrack ID (tb)
a=rtpmap:98 VP8/90000	[I-D.ietf-payload-vp8]
a=imageattr:98 [x=1280,y=720]	[RFC6236]
a=fmtp:98 max-fr=30	[RFC4566]
a=ssrc:12345	[RFC5576]
cname:axzo1278npDlAzM73	
a=bundle-only	[UNIFIED-PLAN]
a=sendrecv	[RFC3264]
a=rtcp-rsize	[RFC5506]
m=video 0 UDP/TLS/RTP/SAVPF 101 103	bundle-only video line with port number set to zero
c=IN IP4 24.23.204.141	[RFC4566]
a=rtcp-mux	[RFC5761]
a=rtcp:0 IN IP4 24.23.204.141	[RFC3605]
a=mid:m2	[RFC5888] Video m=line part of BUNDLE group
a=msid:ma tc	Identifies RTCMediaStream ID (ma) and RTCMediaStreamTrack ID (tc)
a=rtpmap:101 H264/90000	[RFC3984]
a=rtpmap:103 H264/90000	[RFC3984]
a=fmtp:101 profile-level-id=4d0028 ;packetization-mode=1;max-fr=30	[RFC3984]Camera-2,Encoding-1 Resolution
a=ssrc:67890	[RFC5576]
cname:axzo1278npDlAzM73	
a=bundle-only	[UNIFIED-PLAN]
a=sendrecv	[RFC3264]
a=rtcp-rsize	[RFC5506]

Table 40: 5.4.3 SDP Simulcast bundle-only

SDP Contents	RFC#/Notes
v=0	[RFC4566]
o=- 20519 0 IN IP4 0.0.0.0	[RFC4566]

```
| s=-                                        | [RFC4566]                |
| t=0 0                                      | [RFC4566]                |
| m=audio 49203 UDP/TLS/RTP/SAVPF 109        | [RFC4566]                |
| c=IN IP4 24.23.204.141                     | [RFC4566]                |
| a=rtcp:60065 IN IP4 24.23.204.141          | [RFC3605]                |
| a=rtpmap:109 opus/48000/2                  | [I-D.ietf-payload-rtp-   |
|                                            | opus]                    |
| a=extmap:1 urn:ietf:params:rtp-hdrext      | [RFC6464]                |
| :ssrc-audio-level                          |                          |
| a=ptime:20                                 | [I-D.ietf-payload-rtp-   |
|                                            | opus]                    |
| a=rtcp-fb:109 nack                         | [RFC5104]                |
| a=sendrecv                                 | [RFC3264]                |
| a=setup:active                             | [RFC4145]                |
| a=ice-ufrag:ufrag:c300d85b                 | [RFC5245]                |
| a=ice-                                     | [RFC5245]                |
| pwd:de4e99bd291c325921d5d47efbabd9a2       |                          |
| a=fingerprint:sha-1                        | [RFC5245]                |
| 99:41:49:83:4a:97:0e:1f:ef:6d:f7:c9:c7:    |                          |
| 70:9d:1f:66:79:a8:07                       |                          |
| a=candidate:0 1 UDP  2122194687            | [RFC5245]                |
| 192.168.1.7 49203 typ host                 |                          |
| a=candidate:1 1 UDP  1685987071            | [RFC5245]                |
| 98.248.92.77 49203 typ srflx raddr         |                          |
| 192.168.1.7 rport 49203                    |                          |
| a=candidate:0 2 UDP 2122194687             | [RFC5245]                |
| 192.168.1.7 60065 typ host                 |                          |
| a=candidate:1 2 UDP  1685987071            | [RFC5245]                |
| 98.248.92.77 60065 typ srflx raddr         |                          |
| 192.168.1.7 rport 60065                    |                          |
| a=rtcp-rsize                               | [RFC5506]                |
| m=video 0 UDP/TLS/RTP/SAVPF 98 100         | Bob doesn’t recognize    |
|                                            | bundle-only and hence    |
|                                            | rejects the video        |
|                                            | stream                   |
| c=IN IP4 98.248.92.77                      | [RFC4566]                |
| a=rtpmap:98 VP8/90000                      | [I-D.ietf-payload-vp8]   |
| a=rtpmap:100 VP8/90000                     | [I-D.ietf-payload-vp8]   |
| a=imageattr:98 [x=1280,y=720]              | [RFC6236]                |
| a=fmtp:98 max-fr=30                        | [RFC4566]                |
| m=video 0 UDP/TLS/RTP/SAVPF 98 100         | Bob doesn’t recognize    |
|                                            | bundle-only and hence    |
|                                            | rejects the video        |
|                                            | stream                   |
| c=IN IP4 98.248.92.77                      | [RFC4566]                |
| a=rtpmap:101 H264/90000                    | [RFC3984]                |
| a=fmtp:101 profile-level-id=4d0028         | [RFC3984]Camera-2,Enco   |
| ;packetization-mode=1;max-fr=30            | ding-1 Resolution        |
```

```
+-----------------------------------------+------------------------+
```

Table 41: 5.4.3 SDP Anwer

5.4.4. Legacy Interop with RTP/AVP profile

In this section, we attempt to provide session descriptions showcasing inter-operability between a WebRTC end-point and a Legacy VOIP end-point. The ideas included in here are not fully baked into the standards and might be controversial in nature. The hope here is to demonstrate a plausible SDP composition to enchance seamless inter-operability between the aforementioned communication systems.

In the scenario desribed below, Alice is a legacy end-point which sends [RFC3264] Offer with two sets of media descriptions per media type.

One set that correponds to [WebRTC] compliant UDP/TLS/RTP/SAVPF based audio and video descriptions.

Another set with RTP/AVP based audio and video descriptions for the legacy Interop purposes.

Also to note, Alice includes session level DTLS information and media level RTCP feedback information as applicable to both the sets of media descriptions

On the other hand, Bob being a WebRTC end-point, recognizes accepts the media descriptions with RTP/AVP profile. The security and feedback requirements for the session are either handled by a intermediate gateway or with some combination of Alice’s capabilities and the intermediate gateway.

Successful 2-Way WebRTC <-> VOIP Interop

```
Alice                                                            Bob
|                                                                 |
|                                                                 |
|  Alice is a legacy VOIP End-point & Bob is a WebRTC End-Point |
|                                                                 |
|                                                                 |
|                                                                 |
|              Offer(Audio:Opus Video:H.264)                      |
|----------------------------------------------------------------->|
|                                                                 |
|                                                                 |
|Alice includes 2 copies of media descriptions                    |
|1. WebRTC compliant media description (UDP/TLS/RTP/SAVPF)        |
|2. Legacy compliant media description (RTP/AVP)                  |
|                                                                 |
|                                                                 |
|             Answer(Audio:Opus, Video:H.264)                     |
|<-----------------------------------------------------------------|
|                                                                 |Bob
|                                                                 |accepts
|                                                                 |"legacy
|                                                                 |compliant"
|                                                                 |m=line
|                                                                 |
|                                                                 |
|             Two way Opus Audio, H.264 Video                     |
|.................................................................|
|       Session also suports RTP/RTCP Mux, RTCP Feedback          |
|                                                                 |
```

SDP Contents	RFC#/Notes
v=0	[RFC4566]
o=- 20518 0 IN IP4 0.0.0.0	[RFC4566]
s=-	[RFC4566]
t=0 0	[RFC4566]
a=ice-ufrag:074c6550	[RFC5245]
a=ice-pwd:a28a397a4c3f31747d1ee3474af08a068	[RFC5245]
a=fingerprint:sha-1 99:41:49:83:4a:97:0e:1f:e f:6d:f7:c9:c7:70:9d:1f:66:79:a8:07	[RFC5245]
a=rtcp-rsize	[RFC5506]
m=audio 54609 UDP/TLS/RTP/SAVPF 109	[RFC4566]
c=IN IP4 24.23.204.141	[RFC4566]
a=rtpmap:109 opus/48000	

```
   | a=ptime:20                                      |                   |
   | a=sendrecv                                      | [RFC3264]         |
   | a=rtcp-mux                                      | [RFC5761]         |
   | a=candidate:0 1 UDP  2113667327 192.168.1.4     | [RFC5245]         |
   | 54609 typ host                                  |                   |
   | a=candidate:1 1 UDP  694302207 24.23.204.141    | [RFC5245]         |
   | 54609 typ srflx raddr 192.168.1.4 rport 54609   |                   |
   | a=candidate:0 2 UDP 2113667326 192.168.1.4      | [RFC5245]         |
   | 64678 typ host                                  |                   |
   | a=candidate:1 2 UDP  1694302206 24.23.204.141   | [RFC5245]         |
   | 64678 typ srflx raddr 192.168.1.4 rport 64678   |                   |
   | a=rtcp-fb:109 nack                              | [RFC5104]         |
   | m=video  62537 UDP/TLS/RTP/SAVPF 120            | [RFC4566]         |
   | c=IN IP4 24.23.204.141                          | [RFC4566]         |
   | a=rtpmap:120 VP8/90000                          | [I-D.ietf-payload |
   |                                                 | -vp8]             |
   | a=sendrecv                                      | [RFC3264]         |
   | a=rtcp-mux                                      | [RFC5761]         |
   | a=candidate:0 1 UDP  2113667327 192.168.1.4     | [RFC5245]         |
   | 62537 typ host                                  |                   |
   | a=candidate:1 1 UDP  1694302207 24.23.204.141   | [RFC5245]         |
   | 62537 typ srflx raddr 192.168.1.4 rport 62537   |                   |
   | a=candidate:0 2 2113667326 192.168.1.4 54721    | [RFC5245]         |
   | typ host                                        |                   |
   | a=candidate:1 2 UDP 1694302206 24.23.204.141    | [RFC5245]         |
   | 54721 typ srflx raddr 192.168.1.4 rport 54721   |                   |
   | a=rtcp-fb:120 nack pli                          | [RFC5104]         |
   | a=rtcp-fb:120 ccm fir                           | [RFC5104]         |
   | ---------------                                 | These set of      |
   |                                                 | media             |
   |                                                 | descriptions are  |
   |                                                 | for Legacy Inter- |
   |                                                 | op purposes       |
   | m=audio 54732 RTP/AVP 109                       | [RFC4566]Alice    |
   |                                                 | includes RTP/AVP  |
   |                                                 | audio stream      |
   |                                                 | description       |
   | c=IN IP4 24.23.204.141                          | [RFC4566]         |
   | a=fingerprint:sha-1 99:41:49:83:4a:97:0e:1f:7   | [RFC5245]         |
   | f:7d:f9:c9:c7:70:9d:1f:66:79:a8:07              |                   |
   | a=rtpmap:109 opus/48000                         |                   |
   | a=ptime:20                                      |                   |
   | a=sendrecv                                      | [RFC3264]         |
   | a=rtcp-mux                                      | [RFC5761]Alice    |
   |                                                 | still includes    |
   |                                                 | RTP/RTCP Mux      |
   |                                                 | support           |
   | a=candidate:0 1 UDP  2113667327 192.168.1.4     | [RFC5245]         |
```

```
   | 54732 typ host                                  |                   |
   | a=candidate:1 1 UDP  694302207 24.23.204.141  | [RFC5245]         |
   | 54732 typ srflx raddr 192.168.1.4 rport 54732 |                   |
   | a=candidate:0 2 UDP 2113667326 192.168.1.4    | [RFC5245]         |
   | 64678 typ host                                  |                   |
   | a=candidate:1 2 UDP  1694302206 24.23.204.141 | [RFC5245]         |
   | 64678 typ srflx raddr 192.168.1.4 rport 64678 |                   |
   | a=rtcp-fb:109 nack                              | [RFC5104]She adds |
   |                                                 | her intent for    |
   |                                                 | NACK RTCP         |
   |                                                 | feedback support  |
   | m=video  62445 RTP/AVP 120                      | [RFC4566]Alice    |
   |                                                 | includes RTP/AVP  |
   |                                                 | video stream      |
   |                                                 | description       |
   | c=IN IP4 24.23.204.141                          | [RFC4566]         |
   | a=fingerprint:sha-1 99:41:49:83:4a:97:0e:1f:e | [RFC5245]         |
   | f:7d:f7:c9:c7:70:9d:1f:66:79:a8:07              |                   |
   | a=rtpmap:120 VP8/90000                          | [I-D.ietf-payload |
   |                                                 | -vp8]             |
   | a=sendrecv                                      | [RFC3264]         |
   | a=rtcp-mux                                      | [RFC5761]Alice    |
   |                                                 | intends to        |
   |                                                 | perform RTP/RTCP  |
   |                                                 | Mux               |
   | a=candidate:0 1 UDP  2113667327 192.168.1.4   | [RFC5245]         |
   | 62445 typ host                                  |                   |
   | a=candidate:1 1 UDP  1694302207 24.23.204.141 | [RFC5245]         |
   | 62537 typ srflx raddr 192.168.1.4 rport 62445 |                   |
   | a=candidate:0 2 2113667326 192.168.1.4 54721  | [RFC5245]         |
   | typ host                                        |                   |
   | a=candidate:1 2 UDP 1694302206 24.23.204.141  | [RFC5245]         |
   | 54721 typ srflx raddr 192.168.1.4 rport 54721 |                   |
   | a=rtcp-fb:120 nack pli                          | [RFC5104] Alice   |
   |                                                 | indicates support |
   |                                                 | for Picture loss  |
   |                                                 | Indication and    |
   |                                                 | NACK RTCP         |
   |                                                 | feedback          |
   | a=rtcp-fb:120 ccm fir                           | [RFC5104]         |
   +-------------------------------------------------+-------------------+
```

Table 42: 5.4.5 SDP Offer

```
+-----------------------------------------------+-------------------+
| SDP Contents                                  | RFC#/Notes        |
+-----------------------------------------------+-------------------+
| v=0                                           | [RFC4566]         |
| o=-  16833 0 IN IP4 0.0.0.0                   | [RFC4566]         |
| s=-                                           | [RFC4566]         |
| t=0 0                                         | [RFC4566]         |
| a=ice-ufrag:c300d85b                          | [RFC5245]         |
| a=ice-pwd:de4e99bd291c325921d5d47efbabd9a2    | [RFC5245]         |
| a=fingerprint:sha-1 99:41:49:83:4a:97:0e:1f:e | [RFC5245]         |
| f:6d:f7:c9:c7:70:9d:1f:66:79:a8:07            |                   |
| m=audio 49203 RTP/AVP 109                     | [RFC4566] Bob     |
|                                               | accepts RTP/AVP   |
|                                               | based audio       |
|                                               | stream            |
| c=IN IP4 98.248.92.77                         | [RFC4566]         |
| a=rtpmap:109 opus/48000                       |                   |
| a=ptime:20                                    |                   |
| a=sendrecv                                    | [RFC3264]         |
| a=candidate:0 1 UDP 2113667327 192.168.1.7    | [RFC5245]         |
| 49203 typ host                                |                   |
| a=candidate:1 1 UDP 1694302207 98.248.92.77   | [RFC5245]         |
| 49203 typ srflx raddr 192.168.1.7 rport 49203 |                   |
| a=candidate:0 2 UDP 2113667326 192.168.1.7    | [RFC5245]         |
| 60065 typ host                                |                   |
| a=candidate:1 2 UDP 1694302206 98.248.92.77   | [RFC5245]         |
| 60065 typ srflx raddr 192.168.1.7 rport 60065 |                   |
| m=video  63130 RTP/SAVP 120                   | [RFC4566] Bob     |
|                                               | accepts RTP/AVP   |
|                                               | based video stram |
| c=IN IP4 98.248.92.771                        | [RFC4566]         |
| a=rtpmap:120 VP8/90000                        | [I-D.ietf-payload |
|                                               | -vp8]             |
| a=sendrecv                                    | [RFC3264]         |
| a=candidate:0 1 UDP 2113667327 192.168.1.7    | [RFC5245]         |
| 63130 typ host                                |                   |
| a=candidate:1 1 UDP 1694302207 98.248.92.77   | [RFC5245]         |
| 63130 typ srflx raddr 192.168.1.7 rport 63130 |                   |
| a=candidate:0 2 UDP 2113667326 192.168.1.7    | [RFC5245]         |
| 56607 typ host                                |                   |
| a=candidate:1 2 UDP 1694302206 98.248.92.77   | [RFC5245]         |
| 56607 typ srflx raddr 192.168.1.7 rport 56607 |                   |
+-----------------------------------------------+-------------------+
```

Table 43: 5.4.5 SDP Answer

6. IANA Considerations

This document requires no actions from IANA.

7. Acknowledgments

We would like to thanks Justin Uberti, Chris Flo for their detailed review and inputs.

8. Change Log

[RFC EDITOR NOTE: Please remove this section when publishing]

Changes from draft-nandakumar-rtcweb-sdp-06

- o Align with latest BUNDLE draft
- o More clean ups on the content

Changes from draft-nandakumar-rtcweb-sdp-05

- o Added Ascii chart for all the SDP Eaxamples
- o Improved text and updated SDP Examples for Simulcast and FEC
- o Fixed MediaStream ID Semantics SDP Errors

Changes from draft-nandakumar-rtcweb-sdp-04

- o Interim version of the draft to avert expiry
- o Corrected placement of c= line as per RFC4566
- o Updated simulcast SDP to reflect draft-westerlund-avtcore-rtp-simulcast-04

Changes from draft-nandakumar-rtcweb-sdp-03

- o Aligned more closely with JSEP version -05
- o Added Conventions to help readability
- o Add more examples to clarify BUNDLE use-cases

Changes from draft-nandakumar-rtcweb-sdp-02

- o Major refactoring was done to group the examples in to categories

- o SDP was updated through out to reflect JSEP-04 style of defining attributes per m=line than at the session level.

- o Added 8 new examples.

- o Updated references for Trickle, Unified Plan

- o Add section to explain the syntax conventions followed in the examples.

Changes from draft-nandakumar-rtcweb-sdp-01

- o Updated references to OPUS RTP Payload Specification.

- o Updated BUNDLE examples based on the latest draft-ietf-mmusic-sdp-bundle-negotiation.

- o Added examples for multiple audio and video flows based on Unified Plan.

- o Added new examples for RTX and FEC streams

- o Updated Simulcast and SVC examples

Changes from draft-nandakumar-rtcweb-sdp-00

- o Fixed editorial comments on the mailing list.

- o Updated Data-channel SDP information based on draft-ietf-mmusic-sctp-sdp.

- o Updated BUNDLE examples based on draft-ietf-mmusic-sdp-bundle-negotiation.

- o Added examples for few more BUNDLE variants

- o Added new examples for Simulcast and SVC

9. Informative References

[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.

[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.

[RFC5506] Johansson, I. and M. Westerlund, "Support for Reduced-Size Real-Time Transport Control Protocol (RTCP): Opportunities and Consequences", RFC 5506, April 2009.

[RFC3551] Schulzrinne, H. and S. Casner, "RTP Profile for Audio and Video Conferences with Minimal Control", STD 65, RFC 3551, July 2003.

[RFC3952] Duric, A. and S. Andersen, "Real-time Transport Protocol (RTP) Payload Format for internet Low Bit Rate Codec (iLBC) Speech", RFC 3952, December 2004.

[RFC4796] Hautakorpi, J. and G. Camarillo, "The Session Description Protocol (SDP) Content Attribute", RFC 4796, February 2007.

[RFC5761] Perkins, C. and M. Westerlund, "Multiplexing RTP Data and Control Packets on a Single Port", RFC 5761, April 2010.

[RFC3556] Casner, S., "Session Description Protocol (SDP) Bandwidth Modifiers for RTP Control Protocol (RTCP) Bandwidth", RFC 3556, July 2003.

[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.

[RFC4588] Rey, J., Leon, D., Miyazaki, A., Varsa, V., and R. Hakenberg, "RTP Retransmission Payload Format", RFC 4588, July 2006.

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

[RFC5888] Camarillo, G. and H. Schulzrinne, "The Session Description Protocol (SDP) Grouping Framework", RFC 5888, June 2010.

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

[RFC3984] Wenger, S., Hannuksela, M., Stockhammer, T., Westerlund, M., and D. Singer, "RTP Payload Format for H.264 Video", RFC 3984, February 2005.

[RFC5583] Schierl, T. and S. Wenger, "Signaling Media Decoding Dependency in the Session Description Protocol (SDP)", RFC 5583, July 2009.

[RFC5576] Lennox, J., Ott, J., and T. Schierl, "Source-Specific Media Attributes in the Session Description Protocol (SDP)", RFC 5576, June 2009.

[RFC3550] Schulzrinne, H., Casner, S., Frederick, R., and V. Jacobson, "RTP: A Transport Protocol for Real-Time Applications", STD 64, RFC 3550, July 2003.

[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.

[RFC2326] Schulzrinne, H., Rao, A., and R. Lanphier, "Real Time Streaming Protocol (RTSP)", RFC 2326, April 1998.

[RFC3605] Huitema, C., "Real Time Control Protocol (RTCP) attribute in Session Description Protocol (SDP)", RFC 3605, October 2003.

[RFC2833] Schulzrinne, H. and S. Petrack, "RTP Payload for DTMF Digits, Telephony Tones and Telephony Signals", RFC 2833, May 2000.

[RFC6464] Lennox, J., Ivov, E., and E. Marocco, "A Real-time Transport Protocol (RTP) Header Extension for Client-to-Mixer Audio Level Indication", RFC 6464, December 2011.

[RFC6465] Ivov, E., Marocco, E., and J. Lennox, "A Real-time Transport Protocol (RTP) Header Extension for Mixer-to-Client Audio Level Indication", RFC 6465, December 2011.

[RFC7022] Begen, A., Perkins, C., Wing, D., and E. Rescorla, "Guidelines for Choosing RTP Control Protocol (RTCP) Canonical Names (CNAMEs)", RFC 7022, September 2013.

[I-D.ietf-mmusic-sdp-bundle-negotiation]
Holmberg, C., Alvestrand, H., and C. Jennings, "Negotiating Media Multiplexing Using the Session Description Protocol (SDP)", draft-ietf-mmusic-sdp-bundle-negotiation-12 (work in progress), October 2014.

[I-D.ietf-mmusic-sdp-simulcast]
Westerlund, M., Nandakumar, S., and M. Zanaty, "Using Simulcast in SDP and RTP Sessions", draft-ietf-mmusic-sdp-simulcast-00 (work in progress), January 2015.

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

[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-vp8-12 (work in progress), August 2014.

[I-D.ietf-rtcweb-jsep]
Uberti, J., Jennings, C., and E. Rescorla, "Javascript Session Establishment Protocol", draft-ietf-rtcweb-jsep-07 (work in progress), July 2014.

[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-01 (work in progress), February 2014.

[I-D.ietf-mmusic-msid]
Alvestrand, H., "WebRTC MediaStream Identification in the Session Description Protocol", draft-ietf-mmusic-msid-06 (work in progress), June 2014.

[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-07 (work in progress), July 2014.

[I-D.ietf-rtcweb-data-channel]
Jesup, R., Loreto, S., and M. Tuexen, "WebRTC Data Channels", draft-ietf-rtcweb-data-channel-12 (work in progress), September 2014.

[WebRTC] W3C, "WebRTC 1.0: Real-time Communication Between Browsers", <http://dev.w3.org/2011/webrtc/editor/webrtc.html> , .

[UNIFIED-PLAN] Roach, A., Uberti, J., and M. Thomson, "A Unified Plan for Using SDP with Large Numbers of Media Flows", draft-roach-mmusic-unified-plan (work in progress), July 2013.

Authors' Addresses

Suhas Nandakumar
Cisco
170 West Tasman Drive
San Jose, CA 95134
USA

Email: snandaku@cisco.com

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

Phone: +1 408 421-9990
Email: fluffy@cisco.com

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

B. Schwartz
J. Uberti
Google

November 10, 2014

Recursively Encapsulated TURN (RETURN) for Connectivity and Privacy in WebRTC

draft-schwartz-rtcweb-return-04

Abstract

In the context of WebRTC, the concept of a local TURN proxy has been suggested, but not reviewed in detail. WebRTC applications are already using TURN to enhance connectivity and privacy. This document explains how local TURN proxies and WebRTC applications can work together.

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

TURN [RFC5766] is a protocol for communication between a client and a TURN server, in order to route UDP traffic to and from one or more peers. As noted in [RFC5766], the TURN relay server "typically sits in the public Internet". In a WebRTC context, if a TURN server is to be used, it is typically provided by the application, either to provide connectivity between users whose NATs would otherwise prevent it, or to obscure the identity of the participants by concealing their IP addresses from one another.

In many enterprises, direct UDP transmissions are not permitted between clients on the internal networks and external IP addresses, so media must flow over TCP. To enable WebRTC services in such a situation, clients must use TURN-TCP, or TURN-TLS. These configurations are not ideal: they send all traffic over TCP, which leads to higher latency than would otherwise be necessary, and they force the application provider to operate a TURN server because WebRTC endpoints behind NAT cannot typically act as TCP servers. These configurations may result in especially bad behaviors when operating through TCP or HTTP proxies that were not designed to carry real-time media streams.

To avoid forcing WebRTC media streams through a TCP stage, enterprise network operators may operate a TURN server for their network, which can be discovered by clients using TURN Auto-Discovery [I-D.ietf-tram-turn-server-discovery], or through a proprietary mechanism. This TURN server may be placed inside the network, with a firewall configuration allowing it to communicate with the public internet, or it may be operated by the a third party outside the network, with a firewall configuration that allows hosts inside the network. to communicate with it. Use of the specified TURN server may be the only way for clients on the network to achieve a high quality WebRTC experience. This scenario is required to be supported by the WebRTC requirements document [I-D.ietf-rtcweb-use-cases-and-requirements] Section 3.3.5.1.

When the application intends to use a TURN server for identity cloaking, and the enterprise network administrator intends to use a TURN server for connectivity, there is a conflict. In current WebRTC implementations, TURN can only be used on a single-hop basis in each candidate, but using only the enterprise’s TURN server reveals information about the user (e.g. organizational affiliation), and using only the application’s TURN server may be blocked by the network administrator, or may require using TURN-TCP or TURN-TLS, resulting in a significant sacrifice in latency.

To resolve this conflict, we introduce Recursively Encapsulated TURN, a procedure that allows a WebRTC endpoint to route traffic through multiple TURN servers, and get improved connectivity and privacy in return.

2. Goals

These goals are requirements on this document (not on implementations of the specification).

2.1. Connectivity

As noted in [I-D.ietf-rtcweb-use-cases-and-requirements] Section 3.3.5.1 and requirement F20, a WebRTC browser endpoint MUST be able to direct UDP connections through a designated TURN server configured by enterprise policy (a "proxy").

It MUST be possible to configure a WebRTC endpoint that supports proxies to achieve connectivity no worse than if the endpoint were operating at the proxy’s address.

For efficiency, network administrators SHOULD be able to prevent browsers from attempting to send traffic through routes that are already known to be blocked.

2.2. Privacy

To prevent WebRTC peers from determining each others’ IP addresses, applications MUST have the ability to direct all traffic through an application-specified TURN server.

A compatible WebRTC browser MAY attempt to prevent a hostile web page from determining the endpoint’s public IP address. (This requirement is documented in [I-D.ietf-rtcweb-security] Section 4.2.4. Note that the measures proposed here are not sufficient by themselves to achieve this goal. Implementing this specification in current browsers would still leave many other ways for a malicious website to determine the endpoint’s IP address. Operating-system-wide VPN configurations are therefore currently preferred for this purpose.)

A compatible WebRTC browser MAY allow the user to prevent non-malicious web pages from accidentally revealing the IP address of remote peers to a local passive network adversary. This ability SHOULD NOT reduce performance when it is not in use. (Due to the difficulty of distinguishing between stupidity and malice, this goal is principally aspirational.)

3. Concepts

To achieve our goals, we introduce the following new concepts:

3.1. Proxy

In this document a "proxy" is any TURN server that was provided by any mechanism other than through the standard WebRTC-application ICE candidate provisioning API [I-D.ietf-rtcweb-jsep]. If a proxy is to be used, it will be the destination of traffic generated by the client. There is no analogue to the transparent/intercepting HTTP

proxy configuration, which modifies traffic at the network layer. Mechanisms to configure a proxy include Auto-Discovery [I-D.ietf-tram-turn-server-discovery] and local policy ([I-D.ietf-rtcweb-jsep], "ICE candidate policy").

In an application context, a proxy may be "active" (producing candidates) or "inactive" (not in use, having no effect on the context).

3.2. Virtual interface

A typical WebRTC browser endpoint may have multiple network interfaces available, such as wired ethernet, wireless ethernet, and WAN. In this document, a "virtual interface" is a procedure for generating ICE candidates that are not simply generated by a particular physical interface. A virtual interface can produce "host", "server-reflexive", and "relay" candidates, but may be restricted to only some type of candidate (e.g. UDP-only).

3.3. Proxy configuration leakiness

"Leakiness" is an attribute of a proxy configuration. This document defines two values for the "leakiness" of a proxy configuration: "leaky" and "sealed". Proxy configuration, including leakiness, may be set by local policy ([I-D.ietf-rtcweb-jsep], "ICE candidate policy") or other mechanisms.

A leaky configuration adds a proxy and also allows the browser to use routes that transit directly via the endpoint’s physical interfaces (not through the proxy). In a leaky configuration, setting a proxy augments the available set of ICE candidates. Multiple leaky-configuration proxies may therefore be active simultaneously.

A sealed proxy configuration requires the browser to route all WebRTC traffic through the proxy, eliminating all ICE candidates that do not go through the proxy. Only one sealed proxy may be active at a time.

3.4. Sealed proxy rank

In some configurations, an endpoint may be subject to multiple sealed proxy settings at the same time. In that case, one of those settings will have highest rank, and it will be the active proxy. In a given application context (e.g. a webpage), there is at most one active sealed proxy. This document does not specify a representation for rank.

4. Diagrams

This figure shows the connections that provide the ICE candidates for WebRTC in the basic configuration (no proxy). This figure is provided in order to serve as a baseline against which to compare the candidate routes that make use of a proxy.

```
+-------------+       *     *
|UDP generator|       *     *     +----+
|         host+----+--O-----O.....+STUN|
|relay   srflx|    |  *     *     +----+
+--+-------+--+    |  *     *
   |       |       |  * LAN *
   |       \-------/  *     *
   |                  *     *                *
   |      +------+    *     *   +------+     *
   \------+ TURN +==============+ TURN +-----O
          |client|    *     *   |server|     *
          +------+    *     *   +------+     *

.. STUN packets               *** Network interface
-- Bare UDP content link      *O* Candidate port
== TURN encapsulated link
```

Figure 1: Basic WebRTC ICE candidates (no proxy)

This figure shows the connections that provide the ICE candidates for WebRTC on the virtual interface that represents a proxy.

```
+-------------+    *  +------+  *                     +----+
|UDP generator|    *  |Proxy |  *                    .+STUN|
|         host+-------+ TURN |  *               *   . +----+
|relay   srflx|    *  |Client|  *               *  .
+--+-------+--+    *  |      |  *   +------+    * .   +------+     *
   |       |       *  |      |  *   |Proxy |    *.    | App  |     *
   |       \----------+      +######+ TURN +????O=====+TURN  +-----O
   |               *  |      |  *   |Server|    *     |Server|     *
   |      +------+ *  |      |  *   +------+    *     +------+     *
   |      | App  | *  |      |  *               *
   \------+ TURN +====+      |  *               *
          |client| *  |      |  *
          +------+ *  +------+  *

.. STUN packets                          *** Network interface
-- Bare UDP content link                 *O* Candidate port
== TURN encapsulated UDP content link
## RETURN double-encapsulated link
?? Mixed content link
```

Figure 2: WebRTC ICE candidates using a proxy

5. Requirements

5.1. ICE candidates produced in the presence of a proxy

When a proxy is configured, by Auto-Discovery or a proprietary means, the browser MUST NOT report a "relay" candidate representing the proxy. Instead, the browser MUST connect to the proxy and then, if the connection is successful, treat the TURN tunnel as a UDP-only virtual interface.

For a virtual interface representing a TURN proxy, this means that the browser MUST report the public-facing IP address and port acquired through TURN as a "host" candidate, the browser MUST perform STUN through the TURN proxy (if STUN is configured), and it MUST perform TURN by recursive encapsulation through the TURN proxy, resulting in TURN candidates whose "raddr" and "rport" attributes match the acquired public-facing IP address and port on the proxy.

Because the virtual interface has some additional overhead due to indirection, it SHOULD have lower priority than the physical interfaces if physical interfaces are also active. Specifically, even host candidates generated by a virtual interface SHOULD have priority 0 when physical interfaces are active (similar to [RFC5245]

Section 4.1.2.2, "the local preference for host candidates from a VPN interface SHOULD have a priority of 0").

5.2. Leaky proxy configuration

If the active proxy for an application is leaky, the browser should undertake the standard ICE candidate discovery mechanism [RFC5245] on the available physical and virtual interfaces.

5.3. Sealed proxy configuration

If the active proxy for an application is sealed, the browser MUST NOT gather or produce any candidates on physical interfaces. The WebRTC implementation MUST direct its traffic from those interfaces only to the proxy, and perform ICE candidate discovery only on the single virtual interface representing the active proxy.

5.4. Proxy rank

Any browser mechanism for specifying a proxy SHOULD allow the caller to indicate a higher rank than the proxy provided by Auto-Discovery [I-D.ietf-tram-turn-server-discovery].

5.5. Multiple physical interfaces

Some operating systems allow the browser to use multiple interfaces to contact a single remote IP address. To avoid producing an excessive number of candidates, WebRTC endpoints MUST NOT use multiple physical interfaces to connect to a single proxy simultaneously. (If this were violated, it could produce a number of virtual interfaces equal to the product of the number of physical interfaces and the number of active proxies.)

For strategies to choose the best interface for communication with a proxy, see [I-D.reddy-mmusic-ice-best-interface-pcp]. Similar considerations apply when connecting to an application-specified TURN server in the presence of physical and virtual interfaces.

5.6. IPv4 and IPv6

A proxy MAY have both an IPv4 and an IPv6 address (e.g. if the proxy is specified by DNS and has both A and AAAA records). The client MAY try both of these addresses, but MUST select one, preferring IPv6, before allocating any remote addresses. This corresponds to the the Happy Eyeballs [RFC6555] procedure for dual-stack clients.

A proxy MAY provide both IPv4 and IPv6 remote addresses to clients [RFC6156]. A client SHOULD request both address families. If both

requests are granted, the client SHOULD treat the two addresses as host candidates on a dual-stack virtual interface.

5.7. Unspecified leakiness

If a proxy configuration mechanism does not specify leakiness, browsers SHOULD treat the proxy as leaky. This is similar to current WebRTC implementations' behavior in the presence of SOCKS and HTTP proxies: the candidate allocation code continues to generate UDP candidates that do not transit through the proxy.

5.8. Interaction with SOCKS5-UDP

The SOCKS5 proxy standard [RFC1928] permits compliant SOCKS proxies to support UDP traffic. However, most implementations of SOCKS5 today do not support UDP. Accordingly, WebRTC browsers MUST by default (i.e. unless deliberately configured otherwise) treat SOCKS5 proxies as leaky and having lower rank than any configured TURN proxies.

5.9. Encapsulation overhead, fragmentation, and Path MTU

Encapsulating a link in TURN adds overhead on the path between the client and the TURN server, because each packet must be wrapped in a TURN message. This overhead is sometimes doubled in RETURN proxying. To avoid excessive overhead, client implementations SHOULD use ChannelBind and ChannelData messages to connect and send data through proxies and application TURN servers when possible. Clients MAY buffer messages to be sent until the ChannelBind command completes (requiring one round trip to the proxy), or they MAY use CreatePermission and Send messages for the first few packets to reduce startup latency at the cost of higher overhead.

Adding overhead to packets on a link decreases the effective Maximum Transmissible Unit on that link. Accordingly, clients that support proxying MUST NOT rely on the effective MTU complying with the Internet Protocol's minimum MTU requirement.

ChannelData messages have constant overheard, enabling consistent effective PMTU, but Send messages do not necessarily have constant overhead. TURN messages may be fragmented and reassembled if they are not marked with the Don't Fragment (DF) IP bit or the DONT-FRAGMENT TURN attribute. Client implementors should keep this in mind, especially if they choose to implement PMTU discovery through the proxy.

5.10. Interaction with alternate TURN server fallback

As per [RFC5766], a TURN server MAY respond to an Allocate request with an error code of 300 and an ALTERNATE-SERVER indication. When connecting to proxies or application TURN servers, clients SHOULD attempt to connect to the specified alternate server in accordance with [RFC5766]. The client MUST route a connection to the alternate server through the proxy if and only if the original connection attempt was routed through the proxy.

6. Examples

6.1. Firewalled enterprise network with a basic application

In this example, an enterprise network is configured with a firewall that blocks all UDP traffic, and a TURN server is advertised for Auto-Discovery in accordance with [I-D.ietf-tram-turn-server-discovery]. The proxy leakiness of the TURN server is unspecified, so the browser treats it as leaky.

The application specifies a STUN and TURN server on the public net. In accordance with the ICE candidate gathering algorithm RFC 5245 [RFC5245], it receives a set of candidates like:

1. A host candidate acquired from one interface.

   * e.g. candidate:1610808681 1 udp 2122194687 [internal ip addr for interface 0] 63555 typ host generation 0

2. A host candidate acquired from a different interface.

   * e.g. candidate:1610808681 1 udp 2122194687 [internal ip addr for interface 1] 54253 typ host generation 0

3. The proxy, as a host candidate.

   * e.g. candidate:3458234523 1 udp 24584191 [public ip addr for the proxy] 54606 typ host generation 0

4. The virtual interface also generates a STUN candidate, but it is eliminated because it is redundant with the host candidate, as noted in [RFC5245] Sec 4.1.2..

5. The application-provided TURN server as seen through the virtual interface. (Traffic through this candidate is recursively encapsulated.)

* e.g. candidate:702786350 1 udp 24583935 [public ip addr of the application TURN server] 52631 typ relay raddr [public ip addr for the proxy] rport 54606 generation 0

There are no STUN or TURN candidates on the physical interfaces, because the application-specified STUN and TURN servers are not reachable through the firewall.

If the remote peer is within the same network, it may be possible to establish a direct connection using both peers' host candidates. If the network prevents this kind of direct connection, the path will instead take a "hairpin" route through the enterprise's proxy, using one peer's physical "host" candidate and the other's virtual "host" candidate, or (if that is also disallowed by the network configuration) a "double hairpin" using both endpoints' virtual "host" candidates.

6.2. Conflicting proxies configured by Auto-Discovery and local policy

Consider an enterprise network with TURN and HTTP proxies advertised for Auto-Discovery with unspecified leakiness (thus defaulting to leaky). The browser endpoint configures an additional TURN proxy by a proprietary local mechanism.

If the locally configured proxy is leaky, then the browser MUST produce candidates representing any physical interfaces (including SSLTCP routes through the HTTP proxy), plus candidates for both UDP-only virtual interfaces created by the two TURN servers.

There MUST NOT be any candidate that uses both proxies. Multiple configured proxies are not chained recursively.

If the locally configured proxy is "sealed", then the browser MUST produce only candidates from the virtual interface associated with that proxy.

If both proxies are configured for "sealed" use, then the browser MUST produce only candidates from the virtual interface associated with the proxy with higher rank.

7. Security Considerations

This document describes web browser behaviors that, if implemented correctly, allow users to achieve greater identity-confidentiality during WebRTC calls under certain configurations.

If a site administrator offers the site's users a TURN proxy, websites running in the users' browsers will be able to initiate a

UDP-based WebRTC connection to any UDP transport address via the proxy. Websites' connections will quickly terminate if the remote endpoint does not reply with a positive indication of ICE consent, but no such restriction applies to other applications that access the TURN server. Administrators should take care to provide TURN access credentials only to the users who are authorized to have global UDP network access.

TURN proxies and application TURN servers can provide some privacy protection by obscuring the identity of one peer from the other. However, unencrypted TURN provides no additional privacy from an observer who can monitor the link between the TURN client and server, and even encrypted TURN ([I-D.ietf-tram-stun-dtls] Section 4.6) does not provide significant privacy from an observer who sniff traffic on both legs of the TURN connection, due to packet timing correlations.

8. IANA Considerations

This document requires no actions from IANA.

9. Acknowledgements

Significant review, including the virtual-interface formulation, was provided by Justin Uberti. Thanks to Harald Alvestrand, Philipp Hancke, and Tirumaleswar Reddy for suggestions to improve the content and presentation.

10. References

10.1. Normative References

[I-D.ietf-rtcweb-jsep]
Uberti, J. and C. Jennings, "Javascript Session Establishment Protocol", draft-ietf-rtcweb-jsep-06 (work in progress), February 2014.

[RFC1928] Leech, M., Ganis, M., Lee, Y., Kuris, R., Koblas, D., and L. Jones, "SOCKS Protocol Version 5", RFC 5766, March 1996.

[RFC5245] Rosenberg, J., "Interactive Connectivity Establishment (ICE): A Protocol for Network Address Translator (NAT) Traversal for Offer/Answer Protocols", RFC 5245, April 2010.

[RFC5766] Mahy, R., Matthews, P., and J. Rosenberg, "Traversal Using Relays around NAT (TURN): Relay Extensions to Session Traversal Utilities for NAT (STUN)", RFC 5766, April 2010.

[RFC6156] Camarillo, G., Novo, O., and S. Perreault, "Traversal Using Relays around NAT (TURN) Extension for IPv6", RFC 6156, April 2011.

[RFC6555] Wing, D. and A. Yourtchenko, "Happy Eyeballs: Success with Dual-Stack Hosts", RFC 6555, April 2012.

10.2. Informative References

[I-D.ietf-rtcweb-security]
Rescorla, E., "Security Considerations for WebRTC", ietf-rtcweb-security-07 (work in progress), July 2014.

[I-D.ietf-rtcweb-use-cases-and-requirements]
Holmberg, C., Hakansson, S., and G. Eriksson, "Web Real-Time Communication Use-cases and Requirements", ietf-rtcweb-use-cases-and-requirements-14 (work in progress), February 2014.

[I-D.ietf-tram-stun-dtls]
Petit-Huguenin, M. and G. Salgueiro, "Datagram Transport Layer Security (DTLS) as Transport for Session Traversal Utilities for NAT (STUN)", ietf-rtcweb-use-cases-and-requirements-14 (work in progress), June 2014.

[I-D.ietf-tram-turn-server-discovery]
Patil, P., Reddy, T., and D. Wing, "TURN Server Auto Discovery", draft-ietf-tram-turn-server-discovery-00 (work in progress), July 2014.

[I-D.reddy-mmusic-ice-best-interface-pcp]
Reddy, T., Wing, D., VerSteeg, B., Penno, R., and V. Singh, "Improving ICE Interface Selection Using Port Control Protocol (PCP) Flow Extension", draft-ietf-tram-turn-server-discovery-00 (work in progress), October 2013.

Authors' Addresses

Benjamin M. Schwartz
Google, Inc.
111 8th Ave
New York, NY 10011
USA

Email: bemasc@webrtc.org

Justin Uberti
Google, Inc.
747 6th Street South
Kirkland, WA 98033
USA

Email: justin@uberti.name