

Internet Engineering Task Force
Internet-Draft
Intended status: Informational
Expires: July 24, 2014

T. Stach
A. Hutton
Unify
J. Uberti
Google
January 20, 2014

RTCWEB Considerations for NATs, Firewalls and HTTP proxies
draft-hutton-rtcweb-nat-firewall-considerations-03

Abstract

This document describes mechanism to enable media stream establishment for Real-Time Communication in WEB-browsers (WebRTC) in the presence of network address translators, firewalls and HTTP proxies. HTTP proxy and firewall deployed in many private network domains introduce obstacles to the successful establishment of media stream via WebRTC. This document examines some of these deployment scenarios and specifies requirements on WebRTC enabled web browsers designed to provide the best possible chance of media connectivity between WebRTC peers.

Status of This Memo

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

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

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

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

Copyright Notice

Copyright (c) 2014 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of

publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction	2
1.1. Requirements Language	4
2. Considerations for NATs/Firewalls independent of HTTP proxies	4
2.1. NAT/Firewall open for outgoing UDP and TCP traffic	4
2.2. NAT/Firewall open only for TCP traffic	4
2.3. NAT/Firewall open only for TCP on restricted ports	5
3. Considerations for NATs/Firewalls in presence of HTTP proxies	6
3.1. HTTP proxy with NAT/Firewall open for outgoing UDP and TCP traffic	6
3.2. HTTP proxy with NAT/Firewall open only for TCP traffic . .	6
3.3. HTTP proxy with NAT/Firewall open only to proxy routed traffic	6
4. Solutions for Further Study	7
4.1. HTTP CONNECT based mechanism	7
4.2. ALPN - Use of Application Layer Protocol Negotiation . .	8
4.3. TURN server connection via WebSocket	9
4.4. HTTP Fallback for RTP Media Streams	9
4.5. Port Control Protocol	9
4.6. Network Specific TURN Server	9
5. Requirements for RTCWEB-enabled browsers	10
6. Acknowledgements	11
7. IANA Considerations	11
8. Security Considerations	11
9. References	11
9.1. Normative References	11
9.2. Informative References	12
Authors' Addresses	13

1. Introduction

WebRTC is a web-based technique for direct interactive rich communication using audio, video, and data between two peer browsers.

Many organizations, e.g. an enterprise, a public service agency or a university, deploy Network Address Translators (NAT) and firewalls (FW) at the border to the public internet. WebRTC relies on ICE [RFC5245] in order to establish a media path between two WebRTC peers in the presence of such NATs/FWs.

When WebRTC is deployed by the corporate IT department one can assume that the corporate IT configures the corporate NATs, Firewalls, DPI units, TURN servers accordingly. If so desired by the organization WebRTC media streams can then be established to WebRTC peers outside of the organization subject to the applied policies. In order to cater for NAT/FWs with address and port dependent mapping characteristics [RFC4787], the peers will introduce a TURN server [RFC5766] in the public internet as a media relay. Such a TURN server could be deployed by the organization wanting to assert policy on WebRTC traffic.

However, there are also environments that are not prepared for WebRTC and have NAT/FW deployed that prevent media stream establishment although such blocking is not intentional. These environments include e.g. internet cafes or hotels offering their customers access to the web and have opened the well-known HTTP(S) ports but nothing else. In such an environment ICE will fail to establish connectivity. Re-configuration of the NAT/FW is also often impracticable or not possible.

In such an environment a WebRTC user may easily reach its WebRTC server possibly via an HTTP proxy and start establishing a WebRTC session, but will become frustrated when a media connection cannot be established. A corresponding use case and its requirements relating to WebRTC NAT/FW traversal can be found in [draft-ietf-rtcweb-use-cases-and-requirements].

The TURN server in the public internet is not sufficient to establish connectivity for RTP-based media [RFC3550] and the WebRTC data channel [draft-ietf-rtcweb-data-channel] towards external WebRTC peers since the FW policies include blocking of all UDP based traffic and allowing only traffic to the TCP ports 80/443 with the intent to support HTTP(S) [RFC2616].

We explicitly don't address even more restricted environments, that deploy HTTP traffic validation. This could e.g. be done by means of DPI validation or traffic pattern analysis to determine the contents of the packets that the traffic is, in fact, HTTP or HTTPS-looking or by an HTTP proxy that breaks into the TLS exchange and looks for HTTP in the traffic. However we want to address the case when access to the World Wide Web from inside an organization is only possible via a transparent HTTP Proxy that just tunnels traffic after e.g. enforcing an acceptable use policy.

This document examines impact of NAT/FW policies in Section 2. Additional impacts due to the presence of a HTTP proxy are examined in Section 3.

1.1. 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].

2. Considerations for NATs/Firewalls independent of HTTP proxies

This section covers aspects of how NAT/FW characteristic influence the establishment of a media stream. Additional aspects introduced by the presence of a HTTP proxy are covered in Section 3.

If the NATs serving caller and callee both show port and address dependent mapping behavior the need for a TURN server arises in order to establish connectivity for media streams. The TURN server will relay the RTP packet to the WebRTC peer using UDP. How the RTP packets can be transported from the WebRTC client within the private network to the TURN server depends on what the firewall will let pass through.

Other types of NATs do not require using the TURN relay. Nevertheless, the FW rules and policies still affect how media streams can be established.

2.1. NAT/Firewall open for outgoing UDP and TCP traffic

This scenario assumes that the NAT/FW is transparent for all outgoing traffic independent of using UDP or TCP as the transport protocol. This case is used as starting point for introduction of more restrictive firewall policies. It presents the least critical example with respect to the establishment of the media streams.

The TURN server can be reached directly from within the private network via the NAT/FW and the ICE procedures will reveal that media can be sent via the TURN server. The TURN client will send its media to the allocated resources at the TURN server via UDP.

Dependent on the port range that is used for WebRTC media streams, the same statement would be true if the NAT/Firewall would allow UDP traffic for a restricted UDP port range only.

2.2. NAT/Firewall open only for TCP traffic

This scenario assumes that the NAT/FW is transparent for outgoing traffic only using TCP as transport protocol. Theoretically, this gives two options for media stream establishment dependent on the NAT's mapping characteristics. Either transporting RTP over TCP

directly to the peer or contacting a TURN server via TCP that then relays RTP.

In the first case the browser does not use any TURN server to get through its NAT/FW. However, the browser needs to use ICE-TCP [RFC6544] and provide active, passive and/or simultaneous-open TCP candidates. Assuming the peer also provides TCP candidates, a connectivity check for a TCP connection between the two peers should be successful.

In the second case the browser contacts the TURN server via TCP for allocation of an UDP-based relay address at the TURN server. The ICE procedures will reveal that RTP media can be sent via the TURN relay using the TCP connection between TURN client and TURN server. The TURN server would then relay the RTP packets using UDP, as well as other UDP-based protocols. ICE-TCP is not needed in this context.

Note that the second case is not to be confused with using TURN to request a "TCP Allocation" as described in [RFC6062], which deals with how to establish a TCP connection from a TURN server to the peer. For this document we assume that the TURN server can reach the peer always via UDP, possibly via a second TURN server, in case the WebRTC peer is located in a similar environment as described in this section.

We don't see a need to request TCP allocations at the TURN server since it is preferable that WebRTC media is transported over UDP as far as possible. For the same reason we also prefer using TCP just as transport to the TURN server over using the ICE-TCP with an end-to-end TCP connection

2.3. NAT/Firewall open only for TCP on restricted ports

In this case the firewall blocks all outgoing traffic except for TCP traffic to specific ports, for example port 80 (HTTP) for HTTP or 443 for HTTPS(HTTPS). A TURN server listening to its default ports (3478 for TCP/UDP, 5349 for TLS) would not be reachable in this case. However, the TURN server can still be reached when it is configured to listen to e.g. the HTTP(S) ports.

In addition the browser needs to be configured to contact the TURN server over the HTTP(S) ports and/or the WebRTC client has to provide this information to browser.

3. Considerations for NATs/Firewalls in presence of HTTP proxies

This section considers a scenario where all HTTP(S) traffic is routed via an HTTP proxy. We assume that the HTTP proxy is transparent and just tunnels traffic after e.g. enforcing an acceptable use policy with respect to domains that are allowed to be reached. We don't consider cases where the HTTP proxy is used to deploy HTTP traffic validation. This includes DPI validation that the traffic is, in fact, HTTP or HTTPS-looking or a HTTP proxy that breaks into the TLS exchange and looks for HTTP in the traffic.

Note: If both WebRTC clients are located behind the same HTTP proxy, we, of course, assume that ICE would give us a direct media connection within the private network. We don't consider this case in detail within this document.

3.1. HTTP proxy with NAT/Firewall open for outgoing UDP and TCP traffic

As in Section 2.1 we assume that the NAT/FW is transparent for all outgoing traffic independent of using UDP or TCP as transport protocol. The HTTP proxy has no impact on the transport of media streams in this case. Consequently, the same considerations as in Section 2.1 apply with respect to the traversal of the NAT/FW.

3.2. HTTP proxy with NAT/Firewall open only for TCP traffic

As in Section 2.2 we assume that the NAT/FW is transparent only for outgoing TCP traffic. The HTTP proxy has no impact on the transport of media streams in this case. Consequently, the same considerations as in Section 2.2 apply with respect to the traversal of the NAT/FW.

3.3. HTTP proxy with NAT/Firewall open only to proxy routed traffic

Different from the previous scenarios, we assume that the NAT/FW accepts outgoing traffic only via a TCP connection that is initiated from the HTTP proxy. Currently only the case of an explicit proxy is considered here.

This scenario is the most complex and controversial as it requires the WebRTC media to be tunneled through the proxy. However such techniques are already specified in RFC's and deployed an example of this is websockets [RFC6455] which uses the HTTP CONNECT mechanism in the presense of HTTP Proxies.

This document discusses some alternative approaches to achieving connectivity for WebRTC media in this environment but does not currently make any firm recommendations as the alternatives are

mostly work in progress in other areas of the IETF. Therefore it is not possible to make such a recommendation at this time.

4. Solutions for Further Study

The following sections outline and provide some analysis of various solutions to the issues raised regarding WebRTC media traversing firewalls and proxies. All of these potential solutions require further analysis by the IETF RTCWEB working group and in some cases may require work in other IETF working groups.

It is possible that due to different network environments that WebRTC browsers may need to implement more than one solution.

NOTE - THIS ANALYSIS IS NOT COMPLETE.

4.1. HTTP CONNECT based mechanism

A WebRTC browser could make use of the HTTP CONNECT method [RFC2817] and request that the HTTP proxy establishes a tunnel connection on its behalf in order to get access to the TURN server. The HTTP CONNECT request needs to convey the TURN Server URI or transport address. As a result the HTTP Proxy will establish a TCP connection to the TURN server and when successful the HTTP Proxy will answer the HTTP CONNECT request with a 200OK response. In case of a transparent proxy, the HTTP proxy will now switch into tunneling mode and will transparently relay the traffic to the TURN server.

By using the HTTP CONNECT method, the TURN server only has to handle a standard TCP connection. An update to the TURN protocol or the TURN software is not needed.

Afterwards, the browser could upgrade the connection to use TLS, forward STUN/TURN traffic via the HTTP proxy and use the TURN server as media relay. Note that upgrading in this case is not to be misunderstood as usage of the HTTP UPGRADE method as specified in [RFC2817] as this would require the TURN server to support HTTP. The following is a possible sequence of events:

- o the browser opens a TCP connection to the HTTP proxy,
- o the browser issues a HTTP CONNECT request to the HTTP proxy with the TURN server address in the Request URI, for example
 - * CONNECT turn_server.example.com:5349 HTTP/1.1 Host:
turn_server.example.com:5349

- o the HTTP proxy opens a TCP connection to the TURN server and "bridges" the incoming and outgoing TCP connections together, forming a virtual end-to-end TCP connection,
- o the browser can do a TLS handshake over the virtual end-to-end TCP connection with the TURN server.

Strictly speaking the TLS upgrade is not necessary, but using TLS would also prevent the HTTP proxy from sniffing into the data stream and provides the same flow as HTTPS and might improve interoperability with proxy servers. The WebRTC application has the ability to control whether TLS is used by the parameters it supplies to the TURN URI (e.g. turns: vs. turn:), so the decision to access the TURN server via TCP versus TLS could be left up to the application or possibly the browser configuration script.

In contrast to using UDP or TCP for transporting the STUN messages, the browser would now need to first establish a HTTP over TCP connection to the HTTP proxy, upgrade to using TLS and then switch to using this TLS connection for transport of STUN messages.

Further considerations apply to the default connection timeout of the HTTP proxy connection to the TURN server and the timeout of the TURN server allocation. Whereas [RFC5766] specifies a 10 minutes default lifetime of the TURN allocation, typical proxy connection lifetimes are in the range of 60 seconds if no activity is detected. Thus, if the WebRTC client wants to pre-allocate TURN resources it needs to refresh TURN allocations more frequently in order to keep the TCP connection to its TURN server alive.

4.2. ALPN - Use of Application Layer Protocol Negotiation

The application layer protocol negotiation (ALPN) [draft-ietf-tls-appplayerprotoneg] specifies a TLS extension which permits the application layer to negotiate protocol selection within the TLS handshake. This provides an explicit and visible indication of the application layer protocol associated with the TLS connection allowing the application protocol to be visible without relying on the port number to identify the protocol.

[draft-ietf-tls-appplayerprotoneg] could therefore be used to identify that it is WebRTC media that is contained within the TLS connection.

ALPN is effectively an extension to the HTTP CONNECT mechanism described in Section 4.1 since the establishment of the TLS connection would require the use of this mechanism in the presence of a proxy as described in [draft-ietf-httpbis-http2].

4.3. TURN server connection via WebSocket

The WebRTC client could connect to a TURN server via WebSocket [RFC6455] as described in [draft-chenxin-behave-turn-WebSocket]. This might have benefits in very restrictive environments where HTTPS is not permitted through the proxy. However, such environments are also likely to deploy DPI boxes which would eventually complain against usage of WebSocket or block WebRTC traffic based on other heuristic means. It is also to be expected that an environment that does not allow HTTPS will also forbid usage of WebSocket over TLS.

In addition, usage of TURN over WebSocket puts an additional burden on existing TURN server implementation to support HTTP and WebSocket.

This is again effectively an extension to the HTTP CONNECT mechanism described in Section 4.1 since the establishment of the websockets connection would require the use of this mechanism in the presence of a proxy as described in [draft-ietf-httpbis-http2]. Like the ALPN approach the websockets approach also includes that the purpose of the websockets connection is to transport WebRTC media.

4.4. HTTP Fallback for RTP Media Streams

As an alternative to using a TURN server [draft-miniero-rtcweb-http-fallback] proposed to send RTP directly over HTTP. This approach bears some similarities with TURN as it also uses a RTP relay. However, it uses HTTP GET and POST requests to receive and send RTP packets.

Despite a number of open issues, the proposal addresses some corner cases. However, the expected benefit in form of an increased success rate for establishment of a media stream seems rather small.

4.5. Port Control Protocol

As a further alternative, the Port Control Protocol (PCP) [RFC6887] allows the client to communicate with the NAT/FW and negotiate how incoming IPv6 or IPv4 packets are translated and forwarded. However, to be successful such a solution would require the widespread deployment and use of PCP enabled firewalls so this does not appear to be a workable solution at least for early deployments of WebRTC.

4.6. Network Specific TURN Server

If a network specific TURN server under administrative control of the organization is deployed it is desirable to reach this TURN server via UDP. The TURN server could be specified in the proxy configuration script, giving the browser the possibility to learn how

to access it. Then, when gathering candidates, this TURN server would always be used such that the WebRTC client application could get UDP traffic out to the internet.

Since the TURN server is under the same administrative control as the NAT/FW then it can be assumed that the NAT/FW allows WebRTC media that traverses the TURN server to traverse the NAT/FW.

The implementation of this solution in WebRTC is actually a requirement specified in [draft-ietf-rtcweb-use-cases-and-requirements].

The implementation of this solution in WebRTC does not remove the need for other solutions for the case when there is no such network specific TURN server.

5. Requirements for RTCWEB-enabled browsers

THIS SECTION IS EVEN MORE WORK IN PROGRESS THAN PREVIOUS SECTIONS.

For the purpose of relaying WebRTC media streams or data channels a browser needs to be able to

- o connect to a TURN server via UDP, TCP and TLS,
- o support a mechanism for connecting to a TURN server in the presence of a firewall that only permits connections that originate from a HTTP Proxy. The mechanism is for further study.
- o connect to the TURN server via application specified ports other than the default STUN ports including the HTTP(s) ports,
- o use the same proxy selection procedure for TURN as currently done for HTTP (e.g. Web Proxy Autodiscovery Protocol (WPAD) and .pac-files for Proxy-Auto-Config),
- o use a preconfigured or standardized port range for UDP-based media streams or data channels,
- o learn from the proxy configuration script about the presence of a local TURN server and use it for sending UDP traffic to the internet,
- o as an option and if needed, support ICE-TCP for TCP-based direct media connection to the WebRTC peer.

6. Acknowledgements

The authors want to thank Heinrich Haager for all his input during many valuable discussions.

Furthermore, the authors want to thank for comments and suggestions received from Bernard Aboba, Xavier Marjou, Dan Wing, ...

7. IANA Considerations

This memo includes no request to IANA.

8. Security Considerations

In case of using HTTP CONNECT to a TURN server the security consideration of [[draft-ietf-httpbis-p2-semantics], Section-4.3.6] apply. It states that there "are significant risks in establishing a tunnel to arbitrary servers, particularly when the destination is a well-known or reserved TCP port that is not intended for Web traffic. ... Proxies that support CONNECT SHOULD restrict its use to a limited set of known ports or a configurable whitelist of safe request targets."

Consequently when HTTP CONNECT is used to reach a TURN server, the proxy administrator SHOULD configure a whitelist of trusted TURN servers and/or a blacklist of TURN server known to be subject to fraud or other undesired behavior.

With respect to the other discussed alternatives the security considerations of the corresponding RFCs and Internet Drafts apply.

9. References

9.1. Normative References

- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.
- [RFC2616] Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., and T. Berners-Lee, "Hypertext Transfer Protocol -- HTTP/1.1", RFC 2616, June 1999.
- [RFC2817] Khare, R. and S. Lawrence, "Upgrading to TLS Within HTTP/1.1", RFC 2817, May 2000.
- [RFC3550] Schulzrinne, H., Casner, S., Frederick, R., and V. Jacobson, "RTP: A Transport Protocol for Real-Time Applications", STD 64, RFC 3550, July 2003.

- [RFC4787] Audet, F. and C. Jennings, "Network Address Translation (NAT) Behavioral Requirements for Unicast UDP", BCP 127, RFC 4787, January 2007.
- [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.

9.2. Informative References

- [RFC6062] Perreault, S. and J. Rosenberg, "Traversal Using Relays around NAT (TURN) Extensions for TCP Allocations", RFC 6062, November 2010.
- [RFC6455] Fette, I. and A. Melnikov, "The WebSocket Protocol", RFC 6455, December 2011.
- [RFC6544] Rosenberg, J., Keranen, A., Lowekamp, B., and A. Roach, "TCP Candidates with Interactive Connectivity Establishment (ICE)", RFC 6544, March 2012.
- [RFC6887] Wing, D., Cheshire, S., Boucadair, M., Penno, R., and P. Selkirk, "Port Control Protocol (PCP)", RFC 6887, April 2013.
- [draft-chenxin-behave-turn-WebSocket]
Xin. Chen, "Traversal Using Relays around NAT (TURN) Extensions for WebSocket Allocations", 2013,
<<http://tools.ietf.org/html/draft-chenxin-behave-turn-WebSocket>>.
- [draft-ietf-httpbis-http2]
M. Belshe, R. Peon, M. Thomson, A. Melnikov, "Hypertext Transfer Protocol version 2.0", 2013,
<<http://tools.ietf.org/html/draft-ietf-httpbis-http2-09#section-8.3>>.
- [draft-ietf-httpbis-p2-semantics]
R. Fielding, J. Reschke, "Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content", 2013,
<<http://tools.ietf.org/html/draft-ietf-httpbis-p2-semantics-25#section-4.3.6>>.

[draft-ietf-rtcweb-data-channel]

R. Jesup, S. Loreto, M. Tuexen, "RTCWeb Data Channels",
2013, <[http://tools.ietf.org/html/
draft-ietf-rtcweb-data-channel](http://tools.ietf.org/html/draft-ietf-rtcweb-data-channel)>.

[draft-ietf-rtcweb-use-cases-and-requirements]

C. Holmberg, S. Hakansson, G. Eriksson, "Web Real-Time
Communication Use-cases and Requirements", 2013,
<[http://tools.ietf.org/html/
draft-ietf-WebRTC-use-cases-and-requirements](http://tools.ietf.org/html/draft-ietf-WebRTC-use-cases-and-requirements)>.

[draft-ietf-tls-applayerprotoneg]

S. Friedl, A. Popov, A. Langley, E. Stephan, "Transport
Layer Security (TLS) Application Layer Protocol
Negotiation Extension", 2013, <[http://tools.ietf.org/html/
draft-ietf-tls-applayerprotoneg](http://tools.ietf.org/html/draft-ietf-tls-applayerprotoneg)>.

[draft-miniero-rtcweb-http-fallback]

L. Miniero, "HTTP Fallback for RTP Media Streams", 2012,
<[http://tools.ietf.org/html/
draft-miniero-rtcweb-http-fallback](http://tools.ietf.org/html/draft-miniero-rtcweb-http-fallback)>.

Authors' Addresses

Thomas Stach
Unify
Dietrichgasse 27-29
Vienna 1030
AT

Email: thomas.stach@unify.com

Andrew Hutton
Unify
Technology Drive
Nottingham NG9 1LA
UK

Email: andrew.hutton@unify.com

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

Email: justin@uberti.name

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

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

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

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

This Internet-Draft will expire on July 8, 2015.

Copyright Notice

Copyright (c) 2015 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of

publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction	2
2. Conventions	3
3. Use Cases	3
3.1. Use Cases for Unreliable Data Channels	4
3.2. Use Cases for Reliable Data Channels	4
4. Requirements	4
5. SCTP over DTLS over UDP Considerations	6
6. The Usage of SCTP for Data Channels	8
6.1. SCTP Protocol Considerations	8
6.2. SCTP Association Management	9
6.3. SCTP Streams	9
6.4. Data Channel Definition	10
6.5. Opening a Data Channel	10
6.6. Transferring User Data on a Data Channel	11
6.7. Closing a Data Channel	12
7. Security Considerations	13
8. IANA Considerations	13
9. Acknowledgments	14
10. References	14
10.1. Normative References	14
10.2. Informative References	15
Authors' Addresses	16

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].

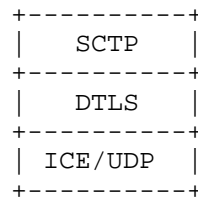


Figure 1: Basic stack diagram

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 user data 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.

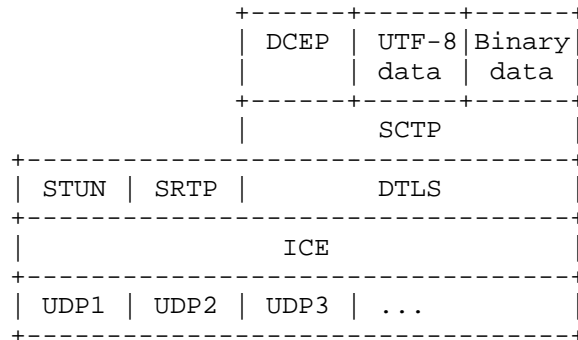


Figure 2: WebRTC protocol layers

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 is 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-
tsvlg-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

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

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

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

This Internet-Draft will expire on July 8, 2015.

Copyright Notice

Copyright (c) 2015 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect

to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction	2
2. Conventions	2
3. Terminology	3
4. Protocol Overview	3
5. Message Formats	4
5.1. DATA_CHANNEL_OPEN Message	4
5.2. DATA_CHANNEL_ACK Message	7
6. Procedures	7
7. Security Considerations	8
8. IANA Considerations	9
8.1. SCTP Payload Protocol Identifier	9
8.2. New Standalone Registry for the DCEP	9
8.2.1. New Message Type Registry	9
8.2.2. New Channel Type Registry	10
9. Acknowledgments	11
10. References	11
10.1. Normative References	11
10.2. Informational References	12
Authors' Addresses	12

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:

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

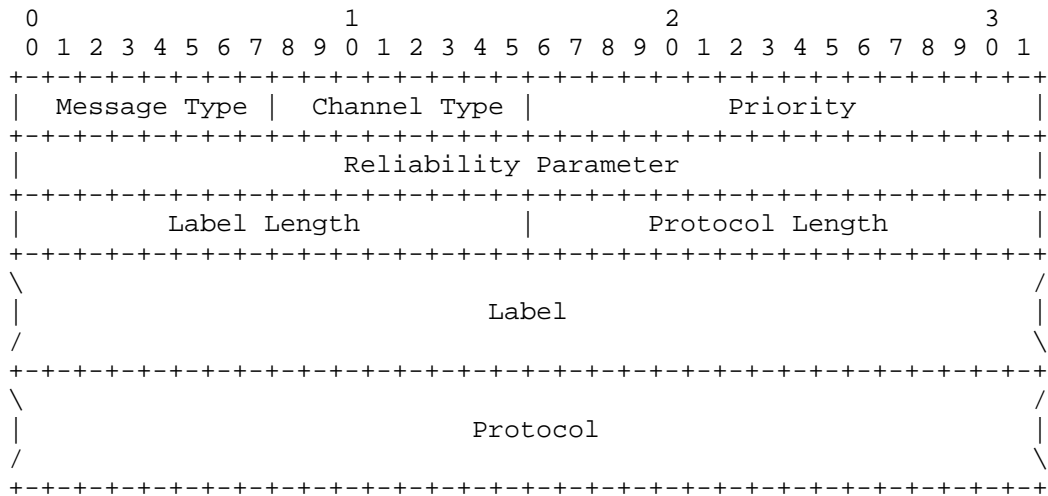
The protocol field is to ease cross-application interoperation ("federation") by identifying the user data being passed with an IANA-registered string ('WebSocket Subprotocol Name Registry' defined in [RFC6455]), and may be useful for homogeneous applications which may create more than one type of Data 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.



Message Type: 1 byte (unsigned integer)

This field holds the IANA defined messagetype 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 milliseconds 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 endpoint 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 31, 2019

J. Uberti
Google
C. Jennings
Cisco
E. Rescorla, Ed.
Mozilla
February 27, 2019

JavaScript Session Establishment Protocol
draft-ietf-rtcweb-jsep-26

Abstract

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

Status of This Memo

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

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

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

This Internet-Draft will expire on August 31, 2019.

Copyright Notice

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

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<https://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of

the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction	4
1.1. General Design of JSEP	4
1.2. Other Approaches Considered	6
2. Terminology	6
3. Semantics and Syntax	7
3.1. Signaling Model	7
3.2. Session Descriptions and State Machine	7
3.3. Session Description Format	11
3.4. Session Description Control	11
3.4.1. RtpTransceivers	11
3.4.2. RtpSenders	12
3.4.3. RtpReceivers	12
3.5. ICE	12
3.5.1. ICE Gathering Overview	12
3.5.2. ICE Candidate Trickling	13
3.5.2.1. ICE Candidate Format	13
3.5.3. ICE Candidate Policy	14
3.5.4. ICE Candidate Pool	15
3.5.5. ICE Versions	16
3.6. Video Size Negotiation	16
3.6.1. Creating an imageattr Attribute	16
3.6.2. Interpreting imageattr Attributes	17
3.7. Simulcast	19
3.8. Interactions With Forking	20
3.8.1. Sequential Forking	20
3.8.2. Parallel Forking	21
4. Interface	22
4.1. PeerConnection	22
4.1.1. Constructor	22
4.1.2. addTrack	24
4.1.3. removeTrack	24
4.1.4. addTransceiver	25
4.1.5. createDataChannel	25
4.1.6. createOffer	25
4.1.7. createAnswer	26
4.1.8. SessionDescriptionType	27
4.1.8.1. Use of Provisional Answers	28
4.1.8.2. Rollback	28
4.1.9. setLocalDescription	29
4.1.10. setRemoteDescription	30
4.1.11. currentLocalDescription	30
4.1.12. pendingLocalDescription	30
4.1.13. currentRemoteDescription	30

4.1.14.	pendingRemoteDescription	31
4.1.15.	canTrickleIceCandidates	31
4.1.16.	setConfiguration	31
4.1.17.	addIceCandidate	32
4.2.	RtpTransceiver	33
4.2.1.	stop	33
4.2.2.	stopped	33
4.2.3.	setDirection	33
4.2.4.	direction	34
4.2.5.	currentDirection	34
4.2.6.	setCodecPreferences	34
5.	SDP Interaction Procedures	35
5.1.	Requirements Overview	35
5.1.1.	Usage Requirements	35
5.1.2.	Profile Names and Interoperability	35
5.2.	Constructing an Offer	37
5.2.1.	Initial Offers	37
5.2.2.	Subsequent Offers	43
5.2.3.	Options Handling	47
5.2.3.1.	IceRestart	47
5.2.3.2.	VoiceActivityDetection	47
5.3.	Generating an Answer	48
5.3.1.	Initial Answers	48
5.3.2.	Subsequent Answers	55
5.3.3.	Options Handling	56
5.3.3.1.	VoiceActivityDetection	56
5.4.	Modifying an Offer or Answer	57
5.5.	Processing a Local Description	57
5.6.	Processing a Remote Description	58
5.7.	Processing a Rollback	58
5.8.	Parsing a Session Description	59
5.8.1.	Session-Level Parsing	60
5.8.2.	Media Section Parsing	61
5.8.3.	Semantics Verification	64
5.9.	Applying a Local Description	65
5.10.	Applying a Remote Description	67
5.11.	Applying an Answer	71
6.	Processing RTP/RTCP	74
7.	Examples	74
7.1.	Simple Example	74
7.2.	Detailed Example	78
7.3.	Early Transport Warmup Example	88
8.	Security Considerations	95
9.	IANA Considerations	96
10.	Acknowledgements	96
11.	References	96
11.1.	Normative References	96
11.2.	Informative References	100

Appendix A. Appendix A	103
Appendix B. Change log	105
Authors' Addresses	115

1. Introduction

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

1.1. General Design of JSEP

WebRTC call setup has been designed to focus on controlling the media plane, leaving signaling plane behavior 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 call signaling protocol, 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. As described above, JSEP assumes a model in which a JavaScript application executes inside a runtime containing WebRTC APIs (the "JSEP implementation"). The JSEP implementation is almost entirely divorced 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. The combination of the JSEP implementation and the JavaScript application is referred to throughout this document as a "JSEP endpoint".

In this document, the use of JSEP is described as if it always occurs between two JSEP endpoints. Note though in many cases it will actually be between a JSEP endpoint and some kind of server, such as a gateway or MCU. This distinction is invisible to the JSEP endpoint; 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 then uses that offer to set up its local config via the `setLocalDescription()` API. The offer is finally 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 [RFC8445], JSEP decouples the ICE state machine from the overall signaling state machine, as the ICE state machine must remain in the JSEP implementation, because only the implementation has the necessary knowledge of candidates and other transport information. Performing this separation provides additional flexibility in protocols that decouple session descriptions from transport. For instance, in traditional SIP, each offer or answer is self-contained, including both the session descriptions and the transport information. However, [I-D.ietf-mmusic-trickle-ice-sip] allows SIP to be used with trickle ICE [I-D.ietf-ice-trickle], in which the session description can be sent immediately and the transport information can be sent when available. 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. JSEP's decoupling of the ICE and signaling state machines allows it to accommodate either model.

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 make life easier for the application 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 JSEP implementation, forcing it to have to understand and handle concepts like signaling glare (see [RFC3264], Section 4).

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 the argument 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 JSEP implementation. Instead of providing createOffer/createAnswer methods within the implementation, 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.

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 session 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 JSEP implementation 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 implementation, and when.

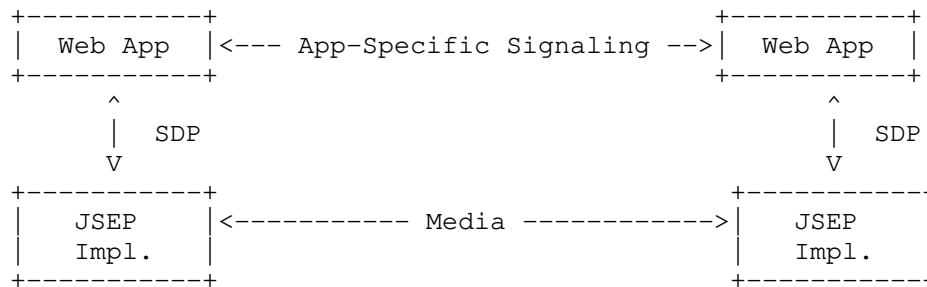


Figure 1: JSEP Signaling Model

3.2. Session Descriptions and State Machine

In order to establish the media plane, the JSEP implementation 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; some parameters are declarative and the remote side MUST either accept them or reject

them altogether. An example of such a parameter is the DTLS fingerprints [RFC8122], which are calculated based on the local certificate(s) offered, and are not subject to negotiation.

In addition, various RFCs put different conditions on the format of offers versus answers. For example, an offer may propose an arbitrary number of m= sections (i.e., media descriptions as described in [RFC4566], Section 5.14), 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, the offerer may receive ICE checks, and possibly media (e.g., in the case of a re-offer after a connection has been established) before it receives 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 JSEP implementation 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])`.

During the offer/answer exchange, the outstanding offer is considered to be "pending" at the offerer and the answerer, as it may either be accepted or rejected. If this is a re-offer, each side will also have "current" local and remote descriptions, which reflect the result of the last offer/answer exchange. Sections Section 4.1.12, Section 4.1.14, Section 4.1.11, and Section 4.1.13, provide more detail on pending and current descriptions.

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; as a result, multiple dissimilar provisional answers, with their own codec choices, transport parameters, etc., can be received and applied during call setup. Note that the final answer itself may be different than any received provisional answers.

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:

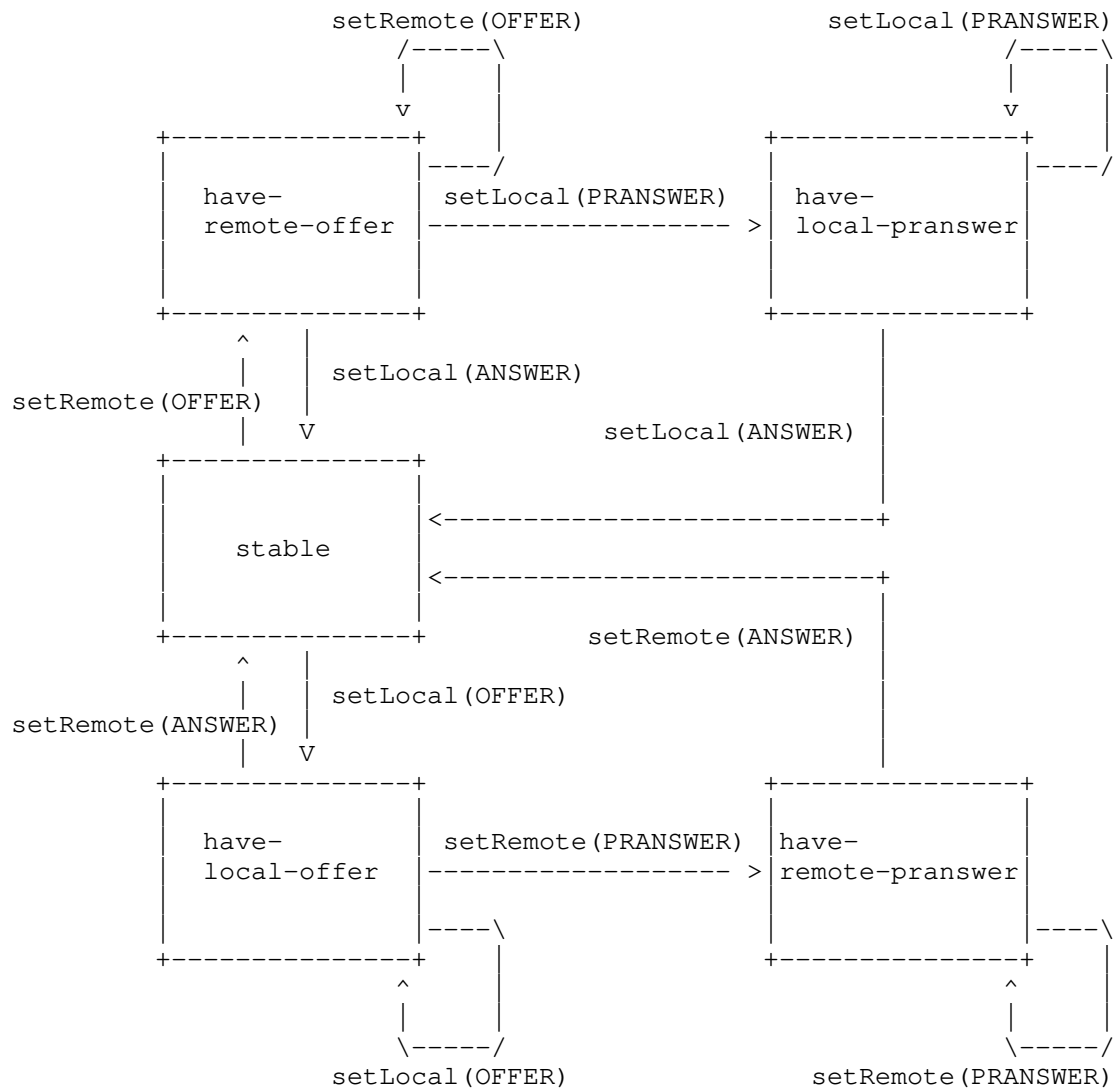


Figure 2: JSEP State Machine

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

JSEP's session descriptions use SDP syntax for their internal representation. 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.

However, to 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.

As detailed below, 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.

3.4. Session Description Control

In order to give the application control over various common session parameters, JSEP provides control surfaces which tell the JSEP implementation how to generate session descriptions. This avoids the need for JavaScript to modify session descriptions in most cases.

Changes to these objects result in changes to the session descriptions generated by subsequent createOffer/Answer calls.

3.4.1. RtpTransceivers

RtpTransceivers allow the application to control the RTP media associated with one m= section. Each RtpTransceiver has an RtpSender and an RtpReceiver, which an application can use to control the sending and receiving of RTP media. The application may also modify the RtpTransceiver directly, for instance, by stopping it.

RtpTransceivers generally have a 1:1 mapping with m= sections, although there may be more RtpTransceivers than m= sections when RtpTransceivers are created but not yet associated with a m= section, or if RtpTransceivers have been stopped and disassociated from m= sections. An RtpTransceiver is said to be associated with an m= section if its mid property is non-null; otherwise it is said to be disassociated. The associated m= section is determined using a mapping between transceivers and m= section indices, formed when creating an offer or applying a remote offer.

An RtpTransceiver is never associated with more than one m= section, and once a session description is applied, a m= section is always associated with exactly one RtpTransceiver. However, in certain cases where a m= section has been rejected, as discussed in Section 5.2.2 below, that m= section will be "recycled" and associated with a new RtpTransceiver with a new mid value.

RtpTransceivers can be created explicitly by the application or implicitly by calling setRemoteDescription with an offer that adds new m= sections.

3.4.2. RtpSenders

RtpSenders allow the application to control how RTP media is sent. An RtpSender is conceptually responsible for the outgoing RTP stream(s) described by an m= section. This includes encoding the attached MediaStreamTrack, sending RTP media packets, and generating/processing RTCP for the outgoing RTP streams(s).

3.4.3. RtpReceivers

RtpReceivers allow the application to inspect how RTP media is received. An RtpReceiver is conceptually responsible for the incoming RTP stream(s) described by an m= section. This includes processing received RTP media packets, decoding the incoming stream(s) to produce a remote MediaStreamTrack, and generating/processing RTCP for the incoming RTP stream(s).

3.5. ICE

3.5.1. 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= section 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 JSEP implementation in response to changes in the ICE configuration.

When the ICE configuration changes in a way that requires a new gathering phase, a 'needs-ice-restart' bit is set. When this bit is set, calls to the createOffer API will generate new ICE credentials. This bit is cleared by a call to the setLocalDescription API with new ICE credentials from either an offer or an answer, i.e., from either a local- or remote-initiated ICE restart.

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 current and/or pending 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= sections; other m= sections continue to use their existing candidates. Also, if an m= section is bundled (either by a successful bundle negotiation or by being marked as bundle-only), then candidates will be gathered and exchanged for that m= section if and only if its MID is a BUNDLE-tag, as described in [I-D.ietf-mmusic-sdp-bundle-negotiation].

3.5.2. 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-ice-trickle]. 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.

3.5.2.1. ICE Candidate Format

In JSEP, ICE candidates are abstracted by an IceCandidate object, and as with session descriptions, SDP syntax is used for the internal representation.

The candidate details are specified in an IceCandidate field, using the same SDP syntax as the "candidate-attribute" field defined in [I-D.ietf-mmusic-ice-sip-sdp], Section 4.1. Note that this field does not contain an "a=" prefix, as indicated in the following example:

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

The IceCandidate object contains a field to indicate which ICE ufrag it is associated with, as defined in [I-D.ietf-mmusic-ice-sip-sdp], Section 4.4. This value is used to determine which session description (and thereby which gathering phase) this IceCandidate belongs to, which helps resolve ambiguities during ICE restarts. If this field is absent in a received IceCandidate (perhaps when communicating with a non-JSEP endpoint), the most recently received session description is assumed.

The IceCandidate object also contains fields to indicate which m= section it is associated with, which can be identified in one of two ways, either by a m= section index, or a MID. The m= section index is a zero-based index, with index N referring to the N+1th m= section in the session description referenced by this IceCandidate. The MID is a "media stream identification" value, as defined in [RFC5888], Section 4, which provides a more robust way to identify the m= section in the session description, using the MID of the associated RtpTransceiver object (which may have been locally generated by the answerer when interacting with a non-JSEP endpoint that does not support the MID attribute, as discussed in Section 5.10 below). If the MID field is present in a received IceCandidate, it MUST be used for identification; otherwise, the m= section index is used instead.

When creating an IceCandidate object, JSEP implementations MUST populate each of the candidate, ufrag, m= section index, and MID fields. Implementations MUST also be prepared to receive objects with some fields missing, as mentioned above.

3.5.3. ICE Candidate Policy

Typically, when gathering ICE candidates, the JSEP implementation 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 only use relay candidates, to leak as little location information as possible (keeping in mind that this choice comes with corresponding operational costs). To accomplish this, JSEP allows the application

to restrict which ICE candidates are used in a session. Note that this filtering is applied on top of any restrictions the implementation chooses to enforce regarding which IP addresses are permitted for the application, as discussed in [I-D.ietf-rtcweb-ip-handling].

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 JSEP implementation **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 JSEP implementation will determine the current setting at the start of each gathering phase. Then, during the gathering phase, the implementation **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.

3.5.4. ICE Candidate Pool

JSEP applications typically inform the JSEP implementation to begin ICE gathering via the information supplied to `setLocalDescription`, as the local description indicates the number of ICE components which will be needed and for which candidates must be gathered. However, to accelerate cases where the application knows the number of ICE components to use ahead of time, it may ask the implementation to gather a pool of potential ICE candidates to help ensure rapid media setup.

When `setLocalDescription` is eventually called, and the JSEP implementation 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. This only occurs for the first offer/answer exchange, after which the candidate pool is emptied and no longer used.

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.

3.5.5. ICE Versions

While this specification formally relies on [RFC8445], at the time of its publication, the majority of WebRTC implementations support the version of ICE described in [RFC5245]. The use of the "ice2" attribute defined in [RFC8445] can be used to detect the version in use by a remote endpoint and to provide a smooth transition from the older specification to the newer one. Implementations **MUST** be able to accept remote descriptions that do not have the "ice2" attribute.

3.6. Video Size Negotiation

Video size negotiation is the process through which a receiver can use the "a=imageattr" SDP attribute [RFC6236] to indicate what video frame sizes it is capable of receiving. A receiver may have hard limits on what its video decoder can process, or it may have some maximum set by policy. By specifying these limits in an "a=imageattr" attribute, JSEP endpoints can attempt to ensure that the remote sender transmits video at an acceptable resolution. However, when communicating with a non-JSEP endpoint that does not understand this attribute, any signaled limits may be exceeded, and the JSEP implementation **MUST** handle this gracefully, e.g., by discarding the video.

Note that certain codecs support transmission of samples with aspect ratios other than 1.0 (i.e., non-square pixels). JSEP implementations will not transmit non-square pixels, but **SHOULD** receive and render such video with the correct aspect ratio. However, sample aspect ratio has no impact on the size negotiation described below; all dimensions are measured in pixels, whether square or not.

3.6.1. Creating an imageattr Attribute

The receiver will first intersect any known local limits (e.g., hardware decoder capabilities, local policy) to determine the absolute minimum and maximum sizes it can receive. If there are no known local limits, the "a=imageattr" attribute **SHOULD** be omitted.

If these local limits preclude receiving any video, i.e., the degenerate case of no permitted resolutions, the "a=imageattr" attribute MUST be omitted, and the m= section MUST be marked as sendonly/inactive, as appropriate.

Otherwise, an "a=imageattr" attribute is created with "recv" direction, and the resulting resolution space formed from the aforementioned intersection is used to specify its minimum and maximum x= and y= values.

The rules here express a single set of preferences, and therefore, the "a=imageattr" q= value is not important. It SHOULD be set to 1.0.

The "a=imageattr" field is payload type specific. When all video codecs supported have the same capabilities, use of a single attribute, with the wildcard payload type (*), is RECOMMENDED. However, when the supported video codecs have different limitations, specific "a=imageattr" attributes MUST be inserted for each payload type.

As an example, consider a system with a multiformat video decoder, which is capable of decoding any resolution from 48x48 to 720p. In this case, the implementation would generate this attribute:

```
a=imageattr:* recv [x=[48:1280],y=[48:720],q=1.0]
```

This declaration indicates that the receiver is capable of decoding any image resolution from 48x48 up to 1280x720 pixels.

3.6.2. Interpreting imageattr Attributes

[RFC6236] defines "a=imageattr" to be an advisory field. This means that it does not absolutely constrain the video formats that the sender can use, but gives an indication of the preferred values.

This specification prescribes more specific behavior. When a `MediaStreamTrack`, which is producing video of a certain resolution (the "track resolution"), is attached to a `RtpSender`, which is encoding the track video at the same or lower resolution(s) (the "encoder resolutions"), and a remote description is applied that references the sender and contains valid "a=imageattr recv" attributes, it MUST follow the rules below to ensure the sender does not transmit a resolution that would exceed the size criteria specified in the attributes. These rules MUST be followed as long as the attributes remain present in the remote description, including cases in which the track changes its resolution, or is replaced with a different track.

Depending on how the RtpSender is configured, it may be producing a single encoding at a certain resolution, or, if simulcast Section 3.7 has been negotiated, multiple encodings, each at their own specific resolution. In addition, depending on the configuration, each encoding may have the flexibility to reduce resolution when needed, or may be locked to a specific output resolution.

For each encoding being produced by the RtpSender, the set of "a=imageattr recv" attributes in the corresponding m= section of the remote description is processed to determine what should be transmitted. Only attributes that reference the media format selected for the encoding are considered; each such attribute is evaluated individually, starting with the attribute with the highest "q=" value. If multiple attributes have the same "q=" value, they are evaluated in the order they appear in their containing m= section. Note that while JSEP endpoints will include at most one "a=imageattr recv" attribute per media format, JSEP endpoints may receive session descriptions from non-JSEP endpoints with m= sections that contain multiple such attributes.

For each "a=imageattr recv" attribute, the following rules are applied. If this processing is successful, the encoding is transmitted accordingly, and no further attributes are considered for that encoding. Otherwise, the next attribute is evaluated, in the aforementioned order. If none of the supplied attributes can be processed successfully, the encoding MUST NOT be transmitted, and an error SHOULD be raised to the application.

- o The limits from the attribute are compared to the encoder resolution. Only the specific limits mentioned below are considered; any other values, such as picture aspect ratio, MUST be ignored. When considering a MediaStreamTrack that is producing rotated video, the unrotated resolution MUST be used for the checks. This is required regardless of whether the receiver supports performing receive-side rotation (e.g., through CVO [TS26.114]), as it significantly simplifies the matching logic.
- o If the attribute includes a "sar=" (sample aspect ratio) value set to something other than "1.0", indicating the receiver wants to receive non-square pixels, this cannot be satisfied and the attribute MUST NOT be used.
- o If the encoder resolution exceeds the maximum size permitted by the attribute, and the encoder is allowed to adjust its resolution, the encoder SHOULD apply downscaling in order to satisfy the limits. Downscaling MUST NOT change the picture aspect ratio of the encoding, ignoring any trivial differences due to rounding. For example, if the encoder resolution is 1280x720,

and the attribute specified a maximum of 640x480, the expected output resolution would be 640x360. If downscaling cannot be applied, the attribute MUST NOT be used.

- o If the encoder resolution is less than the minimum size permitted by the attribute, the attribute MUST NOT be used; the encoder MUST NOT apply upscaling. JSEP implementations SHOULD avoid this situation by allowing receipt of arbitrarily small resolutions, perhaps via fallback to a software decoder.
- o If the encoder resolution is within the maximum and minimum sizes, no action is needed.

3.7. Simulcast

JSEP supports simulcast transmission of a `MediaStreamTrack`, where multiple encodings of the source media can be transmitted within the context of a single `m=` section. The current JSEP API is designed to allow applications to send simulcasted media but only to receive a single encoding. This allows for multi-user scenarios where each sending client sends multiple encodings to a server, which then, for each receiving client, chooses the appropriate encoding to forward.

Applications request support for simulcast by configuring multiple encodings on an `RtpSender`. Upon generation of an offer or answer, these encodings are indicated via SDP markings on the corresponding `m=` section, as described below. Receivers that understand simulcast and are willing to receive it will also include SDP markings to indicate their support, and JSEP endpoints will use these markings to determine whether simulcast is permitted for a given `RtpSender`. If simulcast support is not negotiated, the `RtpSender` will only use the first configured encoding.

Note that the exact simulcast parameters are up to the sending application. While the aforementioned SDP markings are provided to ensure the remote side can receive and demux multiple simulcast encodings, the specific resolutions and bitrates to be used for each encoding are purely a send-side decision in JSEP.

JSEP currently does not provide a mechanism to configure receipt of simulcast. This means that if simulcast is offered by the remote endpoint, the answer generated by a JSEP endpoint will not indicate support for receipt of simulcast, and as such the remote endpoint will only send a single encoding per `m=` section.

In addition, JSEP does not provide a mechanism to handle an incoming offer requesting simulcast from the JSEP endpoint. This means that setting up simulcast in the case where the JSEP endpoint receives the

initial offer requires out-of-band signaling or SDP inspection. However, in the case where the JSEP endpoint sets up simulcast in its initial offer, any established simulcast streams will continue to work upon receipt of an incoming re-offer. Future versions of this specification may add additional APIs to handle the incoming initial offer scenario.

When using JSEP to transmit multiple encodings from a RtpSender, the techniques from [I-D.ietf-mmusic-sdp-simulcast] and [I-D.ietf-mmusic-rid] are used. Specifically, when multiple encodings have been configured for a RtpSender, the m= section for the RtpSender will include an "a=simulcast" attribute, as defined in [I-D.ietf-mmusic-sdp-simulcast], Section 6.2, with a "send" simulcast stream description that lists each desired encoding, and no "recv" simulcast stream description. The m= section will also include an "a=rid" attribute for each encoding, as specified in [I-D.ietf-mmusic-rid], Section 4; the use of RID identifiers allows the individual encodings to be disambiguated even though they are all part of the same m= section.

3.8. 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.

3.8.1. 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 pending remote description as a final answer.

3.8.2. 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 [RFC3960], Section 3.1. 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 (see Section 4.1) 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 using `setDirection` (see Section 4.2.3), 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. PeerConnection

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 3.5.3, causing the JSEP implementation to only surface the permitted candidates (including any implementation-internal filtering) to the application, and only use those candidates for connectivity checks. The set of available policies is as follows:

all: All candidates permitted by implementation policy will be gathered and used.

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 and chooses relay servers, this could obfuscate location to a metro or possibly even global level.

The default ICE candidate policy **MUST** be set to "all" as this is generally the desired policy, and also typically reduces use of application TURN server resources significantly.

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]. Regardless of policy, the application will always try to negotiate bundle onto a single transport, and will offer a single bundle group across all m= sections; use of this single transport is contingent upon the answerer accepting bundle. However, by specifying a policy from the list below, the application can control exactly how aggressively it will try to bundle media streams together, which affects how it will interoperate with a non-bundle-aware endpoint. When negotiating with a non-bundle-aware endpoint, only the streams not marked as bundle-only streams will be established.

The set of available policies is as follows:

balanced: The first m= section of each type (audio, video, or application) will contain transport parameters, which will allow an answerer to unbundle that section. The second and any subsequent m= section of each type will be marked bundle-only. The result is that if there are N distinct media types, then candidates will be gathered for for N media streams. This policy balances desire to multiplex with the need to ensure basic audio and video can still be negotiated in legacy cases. When acting as answerer, if there is no bundle group in the offer, the implementation will reject all but the first m= section of each type.

max-compat: All m= sections will contain transport parameters; none will be marked 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: Only the first m= section will contain transport parameters; 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. When acting as answerer, the implementation will reject any m= sections other than the first m= section, unless they are in the same bundle group as that m= section.

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 JSEP implementation 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 JSEP implementation will only gather RTP candidates and will insert an "a=rtcp-mux-only" indication into any new m= sections in offers it generates. This halves the number of candidates that the offerer needs to gather. Applying a description with an m= section that does not contain an "a=rtcp-mux" attribute will cause an error to be returned.

The default multiplexing policy MUST be set to "require". Implementations MAY choose to reject attempts by the application to set the multiplexing policy to "negotiate".

4.1.1.2. addTrack

The addTrack method adds a MediaStreamTrack to the PeerConnection, using the MediaStream argument to associate the track with other tracks in the same MediaStream, so that they can be added to the same "LS" group when creating an offer or answer. Adding tracks to the same "LS" group indicates that the playback of these tracks should be synchronized for proper lip sync, as described in [RFC5888], Section 7. addTrack attempts to minimize the number of transceivers as follows: If the PeerConnection is in the "have-remote-offer" state, the track will be attached to the first compatible transceiver that was created by the most recent call to setRemoteDescription() and does not have a local track. Otherwise, a new transceiver will be created, as described in Section 4.1.4.

4.1.1.3. removeTrack

The removeTrack method removes a MediaStreamTrack from the PeerConnection, using the RtpSender argument to indicate which sender should have its track removed. The sender's track is cleared, and the sender stops sending. Future calls to createOffer will mark the m= section associated with the sender as recvonly (if transceiver.direction is sendrecv) or as inactive (if transceiver.direction is sendonly).

4.1.4. addTransceiver

The `addTransceiver` method adds a new `RtpTransceiver` to the `PeerConnection`. If a `MediaStreamTrack` argument is provided, then the transceiver will be configured with that media type and the track will be attached to the transceiver. Otherwise, the application **MUST** explicitly specify the type; this mode is useful for creating recvonly transceivers as well as for creating transceivers to which a track can be attached at some later point.

At the time of creation, the application can also specify a transceiver direction attribute, a set of `MediaStreams` which the transceiver is associated with (allowing LS group assignments), and a set of encodings for the media (used for simulcast as described in Section 3.7).

4.1.5. createDataChannel

The `createDataChannel` method creates a new data channel and attaches it to the `PeerConnection`. If no data channel currently exists for this `PeerConnection`, then a new offer/answer exchange is required. All data channels on a given `PeerConnection` share the same SCTP/DTLS association and therefore the same `m=` section, so subsequent creation of data channels does not have any impact on the JSEP state.

The `createDataChannel` method also includes a number of arguments which are used by the `PeerConnection` (e.g., `maxPacketLifetime`) but are not reflected in the SDP and do not affect the JSEP state.

4.1.6. 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 media added 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 allows an application 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 stopping `RtpTransceivers`, 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.

Calling this method may do things such as generating new ICE credentials, but does not change the `PeerConnection` state, trigger candidate gathering, or cause media to start or stop flowing. Specifically, the offer is not applied, and does not become the pending local description, until `setLocalDescription` is called.

4.1.1.7. `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 media added 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.

Calling this method may do things such as generating new ICE credentials, but does not change the `PeerConnection` state, trigger candidate gathering, or or cause a media state change. Specifically, the answer is not applied, and does not become the current local description, until `setLocalDescription` is called.

4.1.8. `SessionDescriptionType`

Session description objects (`RTCSessionDescription`) may be of type "offer", "pranswer", "answer" or "rollback". 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 an "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 stable state, as described in Section 4.1.8.2. The contents **MUST** be empty.

4.1.8.1. Use of Provisional Answers

Most 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 JSEP application is to create and send a "sendonly" final answer with a null `MediaStreamTrack` immediately after receiving the offer, which will prevent media from being sent by the caller, and allow media to be sent immediately upon answer by the callee. Later, when the callee actually accepts the call, the application can plug in the real `MediaStreamTrack` and create a new "sendrecv" offer to update the previous offer/answer pair and start bidirectional media flow. While this could also be done with a "sendonly" pranswer, followed by a "sendrecv" answer, the initial pranswer leaves the offer-answer exchange open, which means that the caller cannot send an updated offer during this time.

As an example, consider a typical JSEP application that wants to set up audio and video as quickly as possible. When the callee receives an offer with audio and video `MediaStreamTracks`, it will send an immediate answer accepting these tracks as sendonly (meaning that the caller will not send the callee any media yet, and because the callee has not yet added its own `MediaStreamTracks`, the callee will not send any media either). It will then ask the user to accept the call and acquire the needed local tracks. Upon acceptance by the user, the application will plug in the tracks it has acquired, which, because ICE and DTLS handshaking have likely completed by this point, can start transmitting immediately. The application will also send a new offer to the remote side indicating call acceptance and moving the audio and video to be two-way media. A detailed example flow along these lines is shown in Section 7.3.

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.

4.1.8.2. 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", which discards any proposed changes to the session, returning the state machine to the stable state. A rollback is performed by supplying a session description of type "rollback" with empty contents to either `setLocalDescription` or `setRemoteDescription`.

4.1.9. `setLocalDescription`

The `setLocalDescription` method instructs the `PeerConnection` to apply the supplied session description as its local configuration. The type field indicates whether the description should be processed as an offer, provisional answer, final answer, or rollback; 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 current and pending local descriptions (e.g., support the codecs that exist in either description). This dual processing begins when the `PeerConnection` enters the "have-local-offer" state, and continues until `setRemoteDescription` is called with either a final answer, at which point the `PeerConnection` can fully adopt the pending local description, or a rollback, which results in a revert to the current local description.

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 each transport, using ones from the candidate pool if available.

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

4.1.10. setRemoteDescription

The setRemoteDescription method instructs the PeerConnection to apply the supplied session description as the desired remote configuration. As in setLocalDescription, the type field of the description indicates how it 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 is available to send, this will result in the starting of media transmission.

4.1.11. currentLocalDescription

The currentLocalDescription method returns the current negotiated local description - i.e., the local description from the last successful offer/answer exchange - in addition to any local candidates that have been generated by the ICE agent since the local description was set.

A null object will be returned if an offer/answer exchange has not yet been completed.

4.1.12. pendingLocalDescription

The pendingLocalDescription method returns a copy of the local description currently in negotiation - i.e., a local offer set without any corresponding remote answer - in addition to any local candidates that have been generated by the ICE agent since the local description was set.

A null object will be returned if the state of the PeerConnection is "stable" or "have-remote-offer".

4.1.13. currentRemoteDescription

The currentRemoteDescription method returns a copy of the current negotiated remote description - i.e., the remote description from the last successful offer/answer exchange - in addition to any remote candidates that have been supplied via processIceMessage since the remote description was set.

A null object will be returned if an offer/answer exchange has not yet been completed.

4.1.14. pendingRemoteDescription

The pendingRemoteDescription method returns a copy of the remote description currently in negotiation - i.e., a remote offer set without any corresponding local answer - in addition to any remote candidates that have been supplied via processIceMessage since the remote description was set.

A null object will be returned if the state of the PeerConnection is "stable" or "have-local-offer".

4.1.15. 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 3.5.2, 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.

4.1.16. 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:

- o Any changes to the STUN/TURN servers to use affect the next gathering phase. If an ICE gathering phase has already started or completed, the 'needs-ice-restart' bit mentioned in Section 3.5.1 will be set. 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, and a local description has not yet been applied, any existing candidates will be discarded, and new candidates will be gathered from the new servers.
- o Any change to the ICE candidate policy affects the next gathering phase. If an ICE gathering phase has already started or completed, the 'needs-ice-restart' bit will be set. Either way, changes to the policy have no effect on the candidate pool, because pooled candidates are not made available to the application until a gathering phase occurs, and so any necessary filtering can still be done on any pooled candidates.
- o The ICE candidate pool size MUST NOT be changed after applying a local description. If a local description has not yet been applied, 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.

4.1.17. addIceCandidate

The addIceCandidate method provides an update to the ICE agent via an IceCandidate object Section 3.5.2.1. If the IceCandidate's candidate field is filled in, the IceCandidate is treated as a new remote ICE candidate, which will be added to the current and/or pending remote description according to the rules defined for Trickle ICE. Otherwise, the IceCandidate is treated as an end-of-candidates indication, as defined in [I-D.ietf-ice-trickle].

In either case, the m= section index, MID, and ufrag fields from the supplied IceCandidate are used to determine which m= section and ICE candidate generation the IceCandidate belongs to, as described in Section 3.5.2.1 above. In the case of an end-of-candidates indication, the absence of both the m= section index and MID fields is interpreted to mean that the indication applies to all m= sections in the specified ICE candidate generation. However, if both fields

are absent for a new remote candidate, this MUST be treated as an invalid condition, as specified below.

If any IceCandidate fields contain invalid values, or an error occurs during the processing of the IceCandidate object, the supplied IceCandidate MUST be ignored and an error MUST be returned.

Otherwise, the new remote candidate or end-of-candidates indication is supplied to the ICE agent. In the case of a new remote candidate, connectivity checks will be sent to the new candidate.

4.2. RtpTransceiver

4.2.1. stop

The stop method stops an RtpTransceiver. This will cause future calls to createOffer to generate a zero port for the associated m= section. See below for more details.

4.2.2. stopped

The stopped property indicates whether the transceiver has been stopped, either by a call to stopTransceiver or by applying an answer that rejects the associated m= section. In either of these cases, it is set to "true", and otherwise will be set to "false".

A stopped RtpTransceiver does not send any outgoing RTP or RTCP or process any incoming RTP or RTCP. It cannot be restarted.

4.2.3. setDirection

The setDirection method sets the direction of a transceiver, which affects the direction property of the associated m= section on future calls to createOffer and createAnswer. The permitted values for direction are "recvonly", "sendrecv", "sendonly", and "inactive", mirroring the identically-named directional attributes defined in [RFC4566], Section 6.

When creating offers, the transceiver direction is directly reflected in the output, even for re-offers. When creating answers, the transceiver direction is intersected with the offered direction, as explained in Section 5.3 below.

Note that while setDirection sets the direction property of the transceiver immediately (Section 4.2.4), this property does not immediately affect whether the transceiver's RtpSender will send or its RtpReceiver will receive. The direction in effect is represented

by the `currentDirection` property, which is only updated when an answer is applied.

4.2.4. `direction`

The `direction` property indicates the last value passed into `setDirection`. If `setDirection` has never been called, it is set to the direction the transceiver was initialized with.

4.2.5. `currentDirection`

The `currentDirection` property indicates the last negotiated direction for the transceiver's associated `m=` section. More specifically, it indicates the [RFC3264] directional attribute of the associated `m=` section in the last applied answer (including provisional answers), with "send" and "recv" directions reversed if it was a remote answer. For example, if the directional attribute for the associated `m=` section in a remote answer is "recvonly", `currentDirection` is set to "sendonly".

If an answer that references this transceiver has not yet been applied, or if the transceiver is stopped, `currentDirection` is set to null.

4.2.6. `setCodecPreferences`

The `setCodecPreferences` method sets the codec preferences of a transceiver, which in turn affect the presence and order of codecs of the associated `m=` section on future calls to `createOffer` and `createAnswer`. Note that `setCodecPreferences` does not directly affect which codec the implementation decides to send. It only affects which codecs the implementation indicates that it prefers to receive, via the offer or answer. Even when a codec is excluded by `setCodecPreferences`, it still may be used to send until the next offer/answer exchange discards it.

The codec preferences of an `RtpTransceiver` can cause codecs to be excluded by subsequent calls to `createOffer` and `createAnswer`, in which case the corresponding media formats in the associated `m=` section will be excluded. The codec preferences cannot add media formats that would otherwise not be present.

The codec preferences of an `RtpTransceiver` can also determine the order of codecs in subsequent calls to `createOffer` and `createAnswer`, in which case the order of the media formats in the associated `m=` section will follow the specified preferences.

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.

5.1.1. Usage Requirements

All session descriptions handled by JSEP implementations, 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.

- o ICE, as specified in [RFC8445], MUST be used. Note that the remote endpoint may use a Lite implementation; implementations MUST properly handle remote endpoints which do ICE-Lite.
- o DTLS [RFC6347] or DTLS-SRTP [RFC5763], MUST be used, as appropriate for the media type, as specified in [I-D.ietf-rtcweb-security-arch]

The SDES SRTP keying mechanism from [RFC4568] MUST NOT be used, as discussed in [I-D.ietf-rtcweb-security-arch].

5.1.2. Profile Names and Interoperability

For media m= sections, JSEP implementations MUST support the "UDP/TLS/RTP/SAVPF" profile specified in [RFC5764] as well as the "TCP/DTLS/RTP/SAVPF" profile specified in [RFC7850], and MUST indicate one of these profiles for each media m= line they produce in an offer. For data m= sections, implementations MUST support the "UDP/DTLS/SCTP" profile as well as the "TCP/DTLS/SCTP" profile, and MUST indicate one of these profiles for each data m= line they produce in an offer. The exact profile to use is determined by the protocol associated with the current default or selected ICE candidate, as described in [I-D.ietf-mmusic-ice-sip-sdp], Section 3.2.1.2.

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/TLS/RTP/SAVPF" or "TCP/DTLS/RTP/SAVPF". In order to simplify compatibility with such endpoints, JSEP

implementations MUST follow the following rules when processing the media m= sections in a received offer:

- o Any profile in the offer matching one of the following MUST be accepted:
 - * "RTP/AVP" (Defined in [RFC4566], Section 8.2.2)
 - * "RTP/AVPF" (Defined in [RFC4585], Section 9)
 - * "RTP/SAVP" (Defined in [RFC3711], Section 12)
 - * "RTP/SAVPF" (Defined in [RFC5124], Section 6)
 - * "TCP/DTLS/RTP/SAVP" (Defined in [RFC7850], Section 3.4)
 - * "TCP/DTLS/RTP/SAVPF" (Defined in [RFC7850], Section 3.5)
 - * "UDP/TLS/RTP/SAVP" (Defined in [RFC5764], Section 9)
 - * "UDP/TLS/RTP/SAVPF" (Defined in [RFC5764], Section 9)
- o The profile in any "m=" line in any generated answer MUST exactly match the profile provided in the offer.
- o Because DTLS-SRTP is REQUIRED, the choice of SAVP or AVP has no effect; support for DTLS-SRTP is determined by the presence of one or more "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 a value of "trr-int=4000".
- o For data m= sections, implementations MUST support receiving the "UDP/DTLS/SCTP", "TCP/DTLS/SCTP", or "DTLS/SCTP" (for backwards compatibility) profiles.

Note that re-offers by JSEP implementations MUST use the correct profile strings even if the initial offer/answer exchange used an (incorrect) older profile string. This simplifies JSEP behavior, with minimal downside, as any remote endpoint that fails to handle such a re-offer will also fail to handle a JSEP endpoint's initial offer.

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 sess-id MUST be representable by a 64-bit signed integer, and the value MUST be less than $(2^{63})-1$. It is RECOMMENDED that the sess-id be constructed by generating a 64-bit quantity with the highest bit set to zero and the remaining 63 bits being cryptographically random. 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 a local IP address in this field; this problem is discussed in [I-D.ietf-rtcweb-ip-handling]. 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 in JSEP, having the same meaningless value seems clearer.
- o Session Information ("i="), URI ("u="), Email Address ("e="), Phone Number ("p="), 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=ice-options" line with the "trickle" and "ice2" options MUST be added, as specified in [I-D.ietf-ice-trickle], Section 3 and [RFC8445], Section 10.
- o If WebRTC identity is being used, an "a=identity" line as described in [I-D.ietf-rtcweb-security-arch], Section 5.

The next step is to generate m= sections, as specified in [RFC4566], Section 5.14. An m= section is generated for each RtpTransceiver that has been added to the PeerConnection, excluding any stopped RtpTransceivers; this is done in the order the RtpTransceivers were added to the PeerConnection. If there are no such RtpTransceivers, no m= sections are generated; more can be added later, as discussed in [RFC3264], Section 5.

For each m= section generated for an RtpTransceiver, establish a mapping between the transceiver and the index of the generated m= section.

Each m= section, provided it is not marked as bundle-only, MUST generate a unique set of ICE credentials and gather its own unique set of ICE candidates. Bundle-only m= sections MUST NOT contain any ICE credentials and MUST NOT gather any candidates.

For DTLS, all m= sections MUST use all the certificate(s) that have been specified for the PeerConnection; as a result, they MUST all have the same [RFC8122] fingerprint value(s), or these value(s) MUST be session-level attributes.

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 If the m= section is marked as bundle-only, then the port value MUST be set to 0. Otherwise, the port value is set to the port of the default ICE candidate for this m= section, but given that no candidates are available yet, the "dummy" port value of 9 (Discard) MUST be used, as indicated in [I-D.ietf-ice-trickle], 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.

- o If codec preferences have been set for the associated transceiver, media formats MUST be generated in the corresponding order, and MUST exclude any codecs not present in the codec preferences.
- o Unless excluded by the above restrictions, the media formats MUST include the mandatory audio/video codecs as specified in [RFC7874], Section 3, and [RFC7742], Section 5.

The m= line MUST be followed immediately by a "c=" line, as specified in [RFC4566], Section 5.7. Again, as no candidates are available yet, the "c=" line must contain the "dummy" value "IN IP4 0.0.0.0", as defined in [I-D.ietf-ice-trickle], Section 5.1.

[I-D.ietf-mmusic-sdp-mux-attributes] groups SDP attributes into different categories. To avoid unnecessary duplication when bundling, attributes of category IDENTICAL or TRANSPORT MUST NOT be repeated in bundled m= sections, repeating the guidance from [I-D.ietf-mmusic-sdp-bundle-negotiation], Section 8.1. This includes m= sections for which bundling has been negotiated and is still desired, as well as m= sections marked as bundle-only.

The following attributes, which are of a category other than IDENTICAL or TRANSPORT, MUST be included in each m= section:

- o An "a=mid" line, as specified in [RFC5888], Section 4. All MID values MUST be generated in a fashion that does not leak user information, e.g., randomly or using a per-PeerConnection counter, and SHOULD 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 14. Note that this does not set the RtpTransceiver mid property, as that only occurs when the description is applied. The generated MID value can be considered a "proposed" MID at this point.
- o A direction attribute which is the same as that of the associated transceiver.
- o For each media format on the m= line, "a=rtpmap" and "a=fmtp" lines, as specified in [RFC4566], Section 6, and [RFC3264], Section 5.1.
- o For each primary codec where RTP retransmission should be used, a corresponding "a=rtpmap" line indicating "rtx" with the clock rate of the primary codec and an "a=fmtp" line that references the payload type 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 If this m= section is for media with configurable durations of media per packet, e.g., audio, an "a=maxptime" line, indicating the maximum amount of media, specified in milliseconds, that can be encapsulated in each packet, as specified in [RFC4566], Section 6. This value is set to the smallest of the maximum duration values across all the codecs included in the m= section.
- o If this m= section is for video media, and there are known limitations on the size of images which can be decoded, an "a=imageattr" line, as specified in Section 3.6.
- o For each supported RTP header extension, an "a=extmap" line, as specified in [RFC5285], Section 5. The list of header extensions that SHOULD/MUST be supported is specified in [I-D.ietf-rtcweb-rtp-usage], Section 5.2. Any header extensions that require encryption MUST be specified as indicated in [RFC6904], Section 4.
- o For each supported RTCP feedback mechanism, an "a=rtcp-fb" line, as specified in [RFC4585], Section 4.2. The list of RTCP feedback mechanisms that SHOULD/MUST be supported is specified in [I-D.ietf-rtcweb-rtp-usage], Section 5.1.
- o If the RtpTransceiver has a sendrecv or sendonly direction:
 - * For each MediaStream that was associated with the transceiver when it was created via addTrack or addTransceiver, an "a=msid" line, as specified in [I-D.ietf-mmusic-msid], Section 2, but omitting the "appdata" field.
- o If the RtpTransceiver has a sendrecv or sendonly direction, and the application has specified RID values or has specified more than one encoding in the RtpSenders's parameters, an "a=rid" line for each encoding specified. The "a=rid" line is specified in [I-D.ietf-mmusic-rid], and its direction MUST be "send". If the application has chosen a RID value, it MUST be used as the rid-identifier; otherwise a RID value MUST be generated by the implementation. RID values MUST be generated in a fashion that does not leak user information, e.g., randomly or using a per-PeerConnection counter, and SHOULD be 3 bytes or less, to allow them to efficiently fit into the RTP header extension defined in [I-D.ietf-avtext-rid], Section 3. If no encodings have been

specified, or only one encoding is specified but without a RID value, then no "a=rid" lines are generated.

- o If the RtpTransceiver has a sendrecv or sendonly direction and more than one "a=rid" line has been generated, an "a=simulcast" line, with direction "send", as defined in [I-D.ietf-mmusic-sdp-simulcast], Section 6.2. The list of RIDs MUST include all of the RID identifiers used in the "a=rid" lines for this m= section.
- 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.

The following attributes, which are of category IDENTICAL or TRANSPORT, MUST appear only in "m=" sections which either have a unique address or which are associated with the bundle-tag. (In initial offers, this means those "m=" sections which do not contain an "a=bundle-only" attribute.)

- o "a=ice-ufrag" and "a=ice-pwd" lines, as specified in [I-D.ietf-mmusic-ice-sip-sdp], Section 4.4.
- o For each desired digest algorithm, one or more "a=fingerprint" lines for each of the endpoint's certificates, as specified in [RFC8122], Section 5.
- o An "a=setup" line, as specified in [RFC4145], Section 4, and clarified for use in DTLS-SRTP scenarios in [RFC5763], Section 5. The role value in the offer MUST be "actpass".
- o An "a=tls-id" line, as specified in [I-D.ietf-mmusic-dtls-sdp], Section 5.2.
- o An "a=rtcp" line, as specified in [RFC3605], Section 2.1, containing the dummy value "9 IN IP4 0.0.0.0", because no candidates have yet been gathered.
- o An "a=rtcp-mux" line, as specified in [RFC5761], Section 5.1.3.
- o If the RTP/RTCP multiplexing policy is "require", an "a=rtcp-mux-only" line, as specified in [I-D.ietf-mmusic-mux-exclusive], Section 4.
- o An "a=rtcp-rsize" line, as specified in [RFC5506], Section 5.

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" [I-D.ietf-mmusic-sctp-sdp]. The "fmt" value MUST be set to "webrtc-datachannel" as specified in [I-D.ietf-mmusic-sctp-sdp], Section 4.1.

Within the data m= section, an "a=mid" line MUST be generated and included as described above, along with an "a=sctp-port" line referencing the SCTP port number, as defined in [I-D.ietf-mmusic-sctp-sdp], Section 5.1, and, if appropriate, an "a=max-message-size" line, as defined in [I-D.ietf-mmusic-sctp-sdp], Section 6.1.

As discussed above, the following attributes of category IDENTICAL or TRANSPORT are included only if the data m= section either has a unique address or is associated with the bundle-tag (e.g., if it is the only m= section):

- o "a=ice-ufrag"
- o "a=ice-pwd"
- o "a=fingerprint"
- o "a=setup"
- o "a=tls-id"

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 JSEP implementation offers all m= sections as one bundle group. However, whether the m= sections are bundle-only or not depends on the bundle policy.

The next step is to generate session-level lip sync groups as defined in [RFC5888], Section 7. For each MediaStream referenced by more than one RtpTransceiver (by passing those MediaStreams as arguments to the addTrack and addTransceiver methods), a group of type "LS" MUST be added that contains the mid values for each RtpTransceiver.

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 assists development and debugging by making it easier to understand individual media sections, especially if one of a set of initially identical attributes is subsequently changed. However, implementations MAY choose to aggregate attributes at the

session level and JSEP implementations MUST be prepared to receive attributes in either location.

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

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

Note that when bundle is used, any additional attributes that are added MUST follow the advice in [I-D.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 previous offer was not applied using `setLocalDescription`, meaning the `PeerConnection` is still in the "stable" state, the steps for generating an initial offer should be followed, subject to the following restriction:

- o The fields of the "o=" line MUST stay the same except for the `<session-version>` field, which MUST increment by one on each call to `createOffer` if the offer might differ from the output of the previous call to `createOffer`; implementations MAY opt to increment `<session-version>` on every call. The value of the generated `<session-version>` is independent of the `<session-version>` of the current local description; in particular, in the case where the current version is N, an offer is created and applied with version N+1, and then that offer is rolled back so that the current version is again N, the next generated offer will still have version N+2.

Note that if the application creates an offer by reading `currentLocalDescription` instead of calling `createOffer`, the returned

SDP may be different than when `setLocalDescription` was originally called, due to the addition of gathered ICE candidates, but the `<session-version>` will not have changed. There are no known scenarios in which this causes problems, but if this is a concern, the solution is simply to use `createOffer` to ensure a unique `<session-version>`.

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

- o The "s=" and "t=" lines MUST stay the same.
- o If any `RtpTransceiver` has been added, and there exists an m= section with a zero port in the current local description or the current remote description, that m= section MUST be recycled by generating an m= section for the added `RtpTransceiver` as if the m= section were being added to the session description (including a new MID value), and placing it at the same index as the m= section with a zero port.
- o If an `RtpTransceiver` is stopped and is not associated with an m= section, an m= section MUST NOT be generated for it. This prevents adding back `RtpTransceivers` whose m= sections were recycled and used for a new `RtpTransceiver` in a previous offer/answer exchange, as described above.
- o If an `RtpTransceiver` has been stopped and is associated with an m= section, and the m= section is not being recycled as described above, an m= section MUST be generated for it with the port set to zero and all "a=msid" lines removed.
- o For `RtpTransceivers` that are not stopped, the "a=msid" line(s) MUST stay the same if they are present in the current description, regardless of changes to the transceiver's direction or track. If no "a=msid" line is present in the current description, "a=msid" line(s) MUST be generated according to the same rules as for an initial offer.
- o Each "m=" and "c=" line MUST be filled in with the port, relevant RTP profile, and address of the default candidate for the m= section, as described in [I-D.ietf-mmusic-ice-sip-sdp], Section 3.2.1.2, and clarified in Section 5.1.2. If no RTP candidates have yet been gathered, dummy values MUST still 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 5.2.3.1 was specified. If the m= section is bundled into another m= section, it still MUST NOT contain any ICE credentials.
- o If the m= section is not bundled into another m= section, its "a=rtcp" attribute line MUST be filled in with the port and address of the default RTCP candidate, as indicated in [RFC5761], Section 5.1.3. If no RTCP candidates have yet been gathered, dummy values MUST be used, as described in the initial offer section above.
- o If the m= section is not bundled into another m= section, for each candidate that has been gathered during the most recent gathering phase (see Section 3.5.1), an "a=candidate" line MUST be added, as defined in [I-D.ietf-mmusic-ice-sip-sdp], Section 4.1. If candidate gathering for the section has completed, an "a=end-of-candidates" attribute MUST be added, as described in [I-D.ietf-ice-trickle], Section 9.3. If the m= section is bundled into another m= section, both "a=candidate" and "a=end-of-candidates" MUST be omitted.
- o For RtpTransceivers that are still present, the "a=rid" lines MUST stay the same.
- o For RtpTransceivers that are still present, any "a=simulcast" line MUST stay the same.

If the previous offer was applied using setLocalDescription, and a corresponding answer from the remote side has been applied using setRemoteDescription, meaning the PeerConnection is in the "have-remote-pranswer" or "stable" states, an offer is generated based on the negotiated session descriptions by following the steps mentioned for the "have-local-offer" state above.

In addition, for each existing, 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 local or remote description, as appropriate:

- o The m= line and corresponding "a=rtpmap" and "a=fmtp" lines MUST only include media formats which have not been excluded by the codec preferences of the associated transceiver, and MUST include all currently available formats. Media formats that were

previously offered but are no longer available (e.g., a shared hardware codec) MAY be excluded.

- o Unless codec preferences have been set for the associated transceiver, the media formats on the m= line MUST be generated in the same order as in the most recent answer. Any media formats that were not present in the most recent answer MUST be added after all existing formats.
- o The RTP header extensions MUST only include those that are present in the most recent answer.
- o The RTCP feedback mechanisms MUST only include those that are present in the most recent answer, except for the case of format-specific mechanisms that are referencing a newly-added media format.
- o The "a=rtcp" line MUST NOT be added if the most recent answer included an "a=rtcp-mux" line.
- o The "a=rtcp-mux" line MUST be the same as that in the most recent answer.
- o The "a=rtcp-mux-only" line MUST NOT be added.
- o The "a=rtcp-rsize" line MUST NOT be added unless present in the most recent answer.
- o An "a=bundle-only" line MUST NOT be added, as indicated in [I-D.ietf-mmusic-sdp-bundle-negotiation], Section 6. Instead, JSEP implementations MUST simply omit parameters in the IDENTICAL and TRANSPORT categories for bundled m= sections, as described in [I-D.ietf-mmusic-sdp-bundle-negotiation], Section 8.1.
- o Note that if media m= sections are bundled into a data m= section, then certain TRANSPORT and IDENTICAL attributes may appear in the data m= section even if they would otherwise only be appropriate for a media m= section (e.g., "a=rtcp-mux"). This cannot happen in initial offers because in the initial offer JSEP implementations always list media m= sections (if any) before the data m= section (if any), and at least one of those media m= sections will not have the "a=bundle-only" attribute. Therefore, in initial offers, any "a=bundle-only" m= sections will be bundled into a preceding non-bundle-only media m= section.

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.

"a=group:LS" attributes are generated in the same way as for initial offers, with the additional stipulation that any lip sync groups that were present in the most recent answer MUST continue to exist and MUST contain any previously existing MID identifiers, as long as the identified m= sections still exist and are not rejected, and the group still contains at least two MID identifiers. This ensures that any synchronized "recvonly" m= sections continue to be synchronized in the new offer.

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. 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 [I-D.ietf-mmusic-ice-sip-sdp], Section 3.4.1.1.1. 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.

5.2.3.2. VoiceActivityDetection

Silence suppression, also known as discontinuous transmission ("DTX"), can reduce the bandwidth used for audio by switching to a special encoding when voice activity is not detected, at the cost of some fidelity.

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 fmp parameters for that codec MUST be specified to indicate that silence suppression for received audio is desired. For example, when using the Opus codec [RFC6716], the

"usedtx=1" parameter, specified in [RFC7587], would be used in the offer.

If the "VoiceActivityDetection" option is specified, with a value of "false", the JSEP implementation MUST NOT emit "CN" codecs. For codecs that have their own internal silence suppression support, the appropriate fmp parameters for that codec MUST be specified to indicate that silence suppression for received audio is not desired. For example, when using the Opus codec, the "usedtx=0" parameter would be specified in the offer. In addition, the implementation MUST NOT use silence suppression for media it generates, regardless of whether the "CN" codecs or related fmp parameters appear in the peer's description. The impact of these rules is that silence suppression in JSEP depends on mutual agreement of both sides, which ensures consistent handling regardless of which codec is used.

The "VoiceActivityDetection" option does not have any impact on the setting of the "vad" value in the signaling of the client to mixer audio level header extension described in [RFC6464], Section 4.

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, except that the "a=ice-options" line, with the "trickle" option as specified in [I-D.ietf-ice-trickle], Section 3, and the "ice2" option as specified

in [RFC8445], Section 10, is only included if such an option was present in the offer.

The next step is to generate session-level lip sync groups, as defined in [RFC5888], Section 7. For each group of type "LS" present in the offer, select the local RtpTransceivers that are referenced by the MID values in the specified group, and determine which of them either reference a common local MediaStream (specified in the calls to addTrack/addTransceiver used to create them), or have no MediaStream to reference because they were not created by addTrack/addTransceiver. If at least two such RtpTransceivers exist, a group of type "LS" with the mid values of these RtpTransceivers MUST be added. Otherwise the offered "LS" group MUST be ignored and no corresponding group generated in the answer.

As a simple example, consider the following offer of a single audio and single video track contained in the same MediaStream. SDP lines not relevant to this example have been removed for clarity. As explained in Section 5.2, a group of type "LS" has been added that references each track's RtpTransceiver.

```
a=group:LS a1 v1
m=audio 10000 UDP/TLS/RTP/SAVPF 0
a=mid:a1
a=msid:ms1
m=video 10001 UDP/TLS/RTP/SAVPF 96
a=mid:v1
a=msid:ms1
```

If the answerer uses a single MediaStream when it adds its tracks, both of its transceivers will reference this stream, and so the subsequent answer will contain a "LS" group identical to that in the offer, as shown below:

```
a=group:LS a1 v1
m=audio 20000 UDP/TLS/RTP/SAVPF 0
a=mid:a1
a=msid:ms2
m=video 20001 UDP/TLS/RTP/SAVPF 96
a=mid:v1
a=msid:ms2
```

However, if the answerer groups its tracks into separate MediaStreams, its transceivers will reference different streams, and so the subsequent answer will not contain a "LS" group.

```
m=audio 20000 UDP/TLS/RTP/SAVPF 0
a=mid:a1
a=msid:ms2a
m=video 20001 UDP/TLS/RTP/SAVPF 96
a=mid:v1
a=msid:ms2b
```

Finally, if the answerer does not add any tracks, its transceivers will not reference any MediaStreams, causing the preferences of the offerer to be maintained, and so the subsequent answer will contain an identical "LS" group.

```
a=group:LS a1 v1
m=audio 20000 UDP/TLS/RTP/SAVPF 0
a=mid:a1
a=recvonly
m=video 20001 UDP/TLS/RTP/SAVPF 96
a=mid:v1
a=recvonly
```

The Section 7.2 example later in this document shows a more involved case of "LS" group generation.

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 are considered to be present in each m= section. Each offered m= section will have an associated RtpTransceiver, as described in Section 5.10. If there are more RtpTransceivers than there are m= sections, the unmatched RtpTransceivers will need to be associated in a subsequent offer.

For each offered m= section, if any of the following conditions are true, 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:

- o The associated RtpTransceiver has been stopped.

- o None of the offered media formats are supported and, if applicable, allowed by codec preferences.
- o The bundle policy is "max-bundle", and this is not the first m= section or in the same bundle group as the first m= section.
- o The bundle policy is "balanced", and this is not the first m= section for this media type or in the same bundle group as the first m= section for this media type.
- o This m= section is in a bundle group, and the group's offerer tagged m= section is being rejected due to one of the above reasons. This requires all m= sections in the bundle group to be rejected, as specified in [I-D.ietf-mmusic-sdp-bundle-negotiation], Section 7.3.3.

Otherwise, 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 are available yet, the "dummy" port value of 9 (Discard) MUST be used, as indicated in [I-D.ietf-ice-trickle], Section 5.1.
- o The <proto> field MUST be set to exactly match the <proto> field for the corresponding m= line in the offer.
- o If codec preferences have been set for the associated transceiver, media formats MUST be generated in the corresponding order, regardless of what was offered, and MUST exclude any codecs not present in the codec preferences.
- o Otherwise, the media formats on the m= line MUST be generated in the same order as those offered in the current remote description, excluding any currently unsupported formats. Any currently available media formats that are not present in the current remote description MUST be added after all existing formats.
- o In either case, the media formats in the answer MUST include at least one format that is present in the offer, but MAY include formats that are locally supported but not present in the offer, as mentioned in [RFC3264], Section 6.1. If no common format exists, the m= section is rejected as described above.

The m= line MUST be followed immediately by a "c=" line, as specified in [RFC4566], Section 5.7. Again, as no candidates are available

yet, the "c=" line must contain the "dummy" value "IN IP4 0.0.0.0", as defined in [I-D.ietf-ice-trickle], 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 contain the following attributes (which are of attribute types other than IDENTICAL and TRANSPORT):

- o If and only 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 A direction attribute, determined by applying the rules regarding the offered direction specified in [RFC3264], Section 6.1, and then intersecting with the direction of the associated RtpTransceiver. For example, in the case where an m= section is offered as "sendonly", and the local transceiver is set to "sendrecv", the result in the answer is a "recvonly" direction.
- o For each media format on the m= line, "a=rtpmap" and "a=fmtp" lines, as specified in [RFC4566], Section 6, and [RFC3264], Section 6.1.
- o If "rtx" is present in the offer, for each primary codec where RTP retransmission should be used, a corresponding "a=rtpmap" line indicating "rtx" with the clock rate of the primary codec and an "a=fmtp" line that references the payload type 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 If this m= section is for media with configurable durations of media per packet, e.g., audio, an "a=maxptime" line, as described in Section 5.2.
- o If this m= section is for video media, and there are known limitations on the size of images which can be decoded, an "a=imageattr" line, as specified in Section 3.6.
- o For each supported RTP header extension that is present in the offer, an "a=extmap" line, as specified in [RFC5285], Section 5. The list of header extensions that SHOULD/MUST be supported is specified in [I-D.ietf-rtcweb-rtp-usage], Section 5.2. Any header

extensions that require encryption MUST be specified as indicated in [RFC6904], Section 4.

- o For each supported RTCP feedback mechanism that is present in the offer, an "a=rtcp-fb" line, as specified in [RFC4585], Section 4.2. The list of RTCP feedback mechanisms that SHOULD/MUST be supported is specified in [I-D.ietf-rtcweb-rtp-usage], Section 5.1.
- o If the RtpTransceiver has a sendrecv or sendonly direction:
 - * For each MediaStream that was associated with the transceiver when it was created via addTrack or addTransceiver, an "a=msid" line, as specified in [I-D.ietf-mmusic-msid], Section 2, but omitting the "appdata" field.

Each m= section which is not bundled into another m= section, MUST contain the following attributes (which are of category IDENTICAL or TRANSPORT):

- o "a=ice-ufrag" and "a=ice-pwd" lines, as specified in [I-D.ietf-mmusic-ice-sip-sdp], Section 4.4.
- o For each desired digest algorithm, one or more "a=fingerprint" lines for each of the endpoint's certificates, as specified in [RFC8122], Section 5.
- 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". When the offer contains the "actpass" value, as will always be the case with JSEP endpoints, the answerer SHOULD use the "active" role. Offers from non-JSEP endpoints MAY send other values for "a=setup", in which case the answer MUST use a value consistent with the value in the offer.
- o An "a=tls-id" line, as specified in [I-D.ietf-mmusic-dtls-sdp], Section 5.3.
- o If present in the offer, an "a=rtcp-mux" line, as specified in [RFC5761], Section 5.1.3. Otherwise, an "a=rtcp" line, as specified in [RFC3605], Section 2.1, containing the dummy value "9 IN IP4 0.0.0.0" (because no candidates have yet been gathered).
- o If present in the offer, an "a=rtcp-rsize" line, as specified in [RFC5506], Section 5.

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> and <fmt> fields MUST be set to exactly match the fields in the offer.

Within the data m= section, an "a=mid" line MUST be generated and included as described above, along with an "a=sctp-port" line referencing the SCTP port number, as defined in [I-D.ietf-mmusic-sctp-sdp], Section 5.1, and, if appropriate, an "a=max-message-size" line, as defined in [I-D.ietf-mmusic-sctp-sdp], Section 6.1.

As discussed above, the following attributes of category IDENTICAL or TRANSPORT are included only if the data m= section is not bundled into another m= section:

- o "a=ice-ufrag"
- o "a=ice-pwd"
- o "a=fingerprint"
- o "a=setup"
- o "a=tls-id"

Note that if media m= sections are bundled into a data m= section, then certain TRANSPORT and IDENTICAL attributes may also appear in the data m= section even if they would otherwise only be appropriate for a media m= section (e.g., "a=rtcp-mux").

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 previous 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 [I-D.ietf-mmusic-ice-sip-sdp], Section 3.2.1.2. Note that in certain cases, the m= line protocol may not match that of the default candidate, because the m= line protocol value MUST match what was supplied in the offer, as described above.
- o Each "a=ice-ufrag" and "a=ice-pwd" line MUST stay the same, unless the m= section is restarting, in which case new ICE credentials must be created as specified in [I-D.ietf-mmusic-ice-sip-sdp], Section 3.4.1.1.1. If the m= section is bundled into another m= section, it still MUST NOT contain any ICE credentials.
- o Each "a=tls-id" line MUST stay the same unless the offerer's "a=tls-id" line changed, in which case a new "a=tls-id" value MUST be created, as described in [I-D.ietf-mmusic-dtls-sdp], Section 5.2.
- o Each "a=setup" line MUST use an "active" or "passive" role value consistent with the existing DTLS association, if the association is being continued by the offerer.
- o RTCP multiplexing must be used, and an "a=rtcp-mux" line inserted if and only if the m= section previously used RTCP multiplexing.

- o If the m= section is not bundled into another m= section and RTCP multiplexing is not active, an "a=rtcp" attribute line MUST be filled in with the port and address of the default RTCP candidate. If no RTCP candidates have yet been gathered, dummy values MUST be used, as described in the initial answer section above.
- o If the m= section is not bundled into another m= section, for each candidate that has been gathered during the most recent gathering phase (see Section 3.5.1), an "a=candidate" line MUST be added, as defined in [I-D.ietf-mmusic-ice-sip-sdp], Section 4.1. If candidate gathering for the section has completed, an "a=end-of-candidates" attribute MUST be added, as described in [I-D.ietf-ice-trickle], Section 9.3. If the m= section is bundled into another m= section, both "a=candidate" and "a=end-of-candidates" MUST be omitted.
- o For RtpTransceivers that are not stopped, the "a=msid" line(s) MUST stay the same, regardless of changes to the transceiver's direction or track. If no "a=msid" line is present in the current description, "a=msid" line(s) MUST be generated according to the same rules as for an initial answer.

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 IceRestart option is unnecessary, as ICE credentials will automatically be changed for all m= sections 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 5.2.3.2, with one exception: if support for silence suppression was not indicated in the offer, the VoiceActivityDetection parameter has no effect, and the answer should be generated as if VoiceActivityDetection was set to false. This is done on a per-codec basis (e.g., if the offerer somehow offered support for CN but set "usedtx=0" for Opus, setting VoiceActivityDetection to true would result in an answer with CN codecs and "usedtx=0"). The impact of this rule is that an answerer will not try to use silence suppression with any endpoint that does not offer it, making silence suppression support bilateral even with non-JSEP endpoints.

5.4. Modifying an Offer or Answer

The SDP returned from `createOffer` or `createAnswer` MUST NOT be changed before passing it to `setLocalDescription`. If precise control over the SDP is needed, the aforementioned `createOffer/createAnswer` options or `RtpTransceiver` APIs MUST be used.

After calling `setLocalDescription` with an offer or answer, the application MAY modify the SDP to reduce its capabilities before sending it to the far side, as long as it follows the rules above that define a valid JSEP offer or answer. Likewise, an application that has received an offer or answer from a peer MAY modify the received SDP, subject to the same constraints, before calling `setRemoteDescription`.

As always, the application is solely responsible for what it sends to the other party, and all incoming SDP will be processed by the JSEP implementation 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.

5.5. Processing a Local Description

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

- o If the description is of type "rollback", follow the processing defined in Section 5.7 and skip the processing described in the rest of this section.
- o Otherwise, 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".
- o If the type is not correct for the current state, processing MUST stop and an error MUST be returned.
- o The `SessionDescription` is then checked to ensure that its contents are identical to those generated in the last call to `createOffer/createAnswer`, and thus have not been altered, as discussed in

Section 5.4; otherwise, processing MUST stop and an error MUST be returned.

- o Next, the SessionDescription is parsed into a data structure, as described in Section 5.8 below.
- o Finally, the parsed SessionDescription is applied as described in Section 5.9 below.

5.6. Processing a Remote Description

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

- o If the description is of type "rollback", follow the processing defined in Section 5.7 and skip the processing described in the rest of this section.
- o Otherwise, 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 Section 5.8 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 Section 5.10 below.

5.7. Processing a Rollback

A rollback may be performed if the PeerConnection is in any state except for "stable". This means that both offers and provisional answers can be rolled back. 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. If a rollback is attempted in the "stable" state, processing MUST stop and an error MUST be returned. Note that this implies that once the answerer has performed setLocalDescription with his answer, this cannot be rolled back.

The effect of rollback MUST be the same regardless of whether `setLocalDescription` or `setRemoteDescription` is called.

In order to process rollback, a JSEP implementation abandons the current offer/answer transaction, sets the signaling state to "stable", and sets the pending local and/or remote description (see Section 4.1.12 and Section 4.1.14) to null. Any resources or candidates that were allocated by the abandoned local description are discarded; any media that is received is processed according to the previous local and remote descriptions.

A rollback disassociates any `RtpTransceivers` that were associated with `m=` sections by the application of the rolled-back session description (see Section 5.10 and Section 5.9). This means that some `RtpTransceivers` that were previously associated will no longer be associated with any `m=` section; in such cases, the value of the `RtpTransceiver`'s `mid` property MUST be set to null, and the mapping between the transceiver and its `m=` section index MUST be discarded. `RtpTransceivers` that were created by applying a remote offer that was subsequently rolled back MUST be stopped and removed from the `PeerConnection`. However, a `RtpTransceiver` MUST NOT be removed if a track was attached to the `RtpTransceiver` via the `addTrack` method. This is so that an application may call `addTrack`, then call `setRemoteDescription` with an offer, then roll back that offer, then call `createOffer` and have a `m=` section for the added track appear in the generated offer.

5.8. Parsing a Session Description

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 to its defining ABNF, check that it conforms to [RFC4566] and [RFC3264] semantics, and then either parse and store or discard the provided value, as described below.

If any line is not well-formed, or cannot be parsed as described, the parser MUST stop with an error and reject the session description, even if the value is to be discarded. This ensures that implementations do not accidentally misinterpret ambiguous SDP.

5.8.1. 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.

The following non-attribute lines are not meaningful in the JSEP context and MAY be discarded once they have been checked.

The "c=" line MUST be checked for syntax but its value is only used for ICE mismatch detection, as defined in [RFC8445], Section 5.4. Note that JSEP implementations should never encounter this condition because ICE is required for WebRTC.

The "i=", "u=", "e=", "p=", "t=", "r=", "z=", and "k=" lines are not used by this specification; they MUST be checked for syntax but their values are not used.

The remaining non-attribute lines are processed as follows:

The "v=" line MUST have a version of 0, as specified in [RFC4566], Section 5.1.

The "o=" line MUST be parsed as specified in [RFC4566], Section 5.2.

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

Finally, the attribute lines are processed. Specific processing MUST be applied for the following session-level attribute ("a=") lines:

- o Any "a=group" lines are parsed as specified in [RFC5888], Section 5, and the group's semantics and mids are stored.
- o If present, a single "a=ice-lite" line is parsed as specified in [I-D.ietf-mmusic-ice-sip-sdp], Section 4.3, 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 [I-D.ietf-mmusic-ice-sip-sdp], Section 4.4, and the ufrag value is stored.

- o If present, a single "a=ice-pwd" line is parsed as specified in [I-D.ietf-mmusic-ice-sip-sdp], Section 4.4, and the password value is stored.
- o If present, a single "a=ice-options" line is parsed as specified in [I-D.ietf-mmusic-ice-sip-sdp], Section 4.6, and the set of specified options is stored.
- o Any "a=fingerprint" lines are parsed as specified in [RFC8122], 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 If present, a single "a=tls-id" line is parsed as specified in [I-D.ietf-mmusic-dtls-sdp] Section 5, and the tls-id value is stored.
- o Any "a=identity" lines are parsed and the identity values stored for subsequent verification, as specified [I-D.ietf-rtcweb-security-arch], Section 5.
- o Any "a=extmap" lines are parsed as specified in [RFC5285], Section 5, and their values are stored.

Other attributes that are not relevant to JSEP may also be present, and implementations SHOULD process any that they recognize. As required by [RFC4566], Section 5.13, unknown attribute lines MUST be ignored.

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

5.8.2. Media Section Parsing

Like the session-level lines, the media section 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 As with the "c=" line at the session level, the "c=" line MUST be parsed according to [RFC4566], Section 5.7, but its value is not used.
- 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-ufrag" line is parsed as specified in [I-D.ietf-mmusic-ice-sip-sdp], Section 4.4, and the ufrag value is stored.
- o If present, a single "a=ice-pwd" line is parsed as specified in [I-D.ietf-mmusic-ice-sip-sdp], Section 4.4, and the password value is stored.
- o If present, a single "a=ice-options" line is parsed as specified in [I-D.ietf-mmusic-ice-sip-sdp], Section 4.6, and the set of specified options is stored.
- o Any "a=candidate" attributes MUST be parsed as specified in [I-D.ietf-mmusic-ice-sip-sdp], Section 4.1, and their values stored.
- o Any "a=remote-candidates" attributes MUST be parsed as specified in [I-D.ietf-mmusic-ice-sip-sdp], Section 4.2, but their values are ignored.
- o If present, a single "a=end-of-candidates" attribute MUST be parsed as specified in [I-D.ietf-ice-trickle], Section 8.2, and its presence or absence flagged and stored.
- o Any "a=fingerprint" lines are parsed as specified in [RFC8122], Section 5, and the set of fingerprint and algorithm values is stored.

If the "m=" proto value indicates use of RTP, as described in Section 5.1.2 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 "a=maxptime" 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" attributes MUST be parsed as specified in [RFC5576], Section 4.1, 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" attribute MUST be parsed as specified in [RFC5761], Section 5.1.3, and its presence or absence flagged and stored.
- o If present, a single "a=rtcp-mux-only" attribute MUST be parsed as specified in [I-D.ietf-mmusic-mux-exclusive], Section 3, and its presence or absence flagged and stored.
- o If present, a single "a=rtcp-rsize" attribute MUST be parsed as specified in [RFC5506], Section 5, and its presence or absence flagged and stored.
- o If present, a single "a=rtcp" attribute MUST be parsed as specified in [RFC3605], Section 2.1, but its value is ignored, as this information is superfluous when using ICE.
- o If present, "a=msid" attributes MUST be parsed as specified in [I-D.ietf-mmusic-msid], Section 3.2, and their values stored, ignoring any "appdata" field. If no "a=msid" attributes are present, a random msid-id value is generated for a "default" MediaStream for the session, if not already present, and this value is stored.
- o Any "a=imageattr" attributes MUST be parsed as specified in [RFC6236], Section 3, and their values stored.
- o Any "a=rid" lines MUST be parsed as specified in [I-D.ietf-mmusic-rid], Section 10, and their values stored.

- o If present, a single "a=simulcast" line MUST be parsed as specified in [I-D.ietf-mmusic-sdp-simulcast], and its values stored.

Otherwise, if the "m=" proto value indicates 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 If present, a single "a=max-message-size" attribute MUST be parsed as specified in [I-D.ietf-mmusic-sctp-sdp], Section 6, and the value stored. Otherwise, use the specified default.

Other attributes that are not relevant to JSEP may also be present, and implementations SHOULD process any that they recognize. As required by [RFC4566], Section 5.13, unknown attribute lines MUST be ignored.

5.8.3. 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 5.1.1 MUST be present. These values MAY either be present at the media level, or inherited from the session level.
- * ICE ufrag and password values, which MUST comply with the size limits specified in [I-D.ietf-mmusic-ice-sip-sdp], Section 4.4.
- * tls-id value, which MUST be set according to [I-D.ietf-mmusic-dtls-sdp], Section 5. If this is a re-offer or a response to a re-offer and the tls-id value is different from that presently in use, the DTLS connection is not being continued and the remote description MUST be part of an ICE restart, together with new ufrag and password values.
- * DTLS setup value, which MUST be set according to the rules specified in [RFC5763], Section 5 and MUST be consistent with

the selected role of the current DTLS connection, if one exists and is being continued.

- * DTLS fingerprint values, where at least one fingerprint MUST be present.
- o All RID values referenced in an "a=simulcast" line MUST exist as "a=rid" lines.
- o Each m= section is also checked to ensure prohibited features are not used.
- o If the RTP/RTCP multiplexing policy is "require", each m= section MUST contain an "a=rtcp-mux" attribute. If an m= section contains an "a=rtcp-mux-only" attribute, that section MUST also contain an "a=rtcp-mux" attribute.
- o If an m= section was present in the previous answer, the state of RTP/RTCP multiplexing MUST match what was previously negotiated.

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, including the requirement that the number of m= sections MUST exactly match the number of m= sections in the associated offer.
- o For each m= section, the media type and protocol values MUST exactly match the media type and protocol values in the corresponding m= section in the associated offer.

If any of the preceding checks failed, processing MUST stop and an error MUST be returned.

5.9. Applying a Local Description

The following steps are performed at the media engine level to apply a local description. If an error is returned, the session MUST be restored to the state it was in before performing these steps.

First, m= sections are processed. For each m= 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 If this m= section is new, begin gathering candidates for it, as defined in [RFC8445], Section 5.1.1, unless it is definitively

being bundled (either this is an offer and the m= section is marked bundle-only, or it is an answer and the m= section is bundled into into another m= section.)

- o Or, if the ICE ufrag and password values have changed, trigger the ICE agent to start an ICE restart as described in [RFC8445], Section 9, and begin gathering new candidates for the m= section. If this description is an answer, also start checks on that media section.
- o If the m= section proto value indicates use of RTP:
 - * If there is no RtpTransceiver associated with this m= section, find one and associate it with this m= section according to the following steps. Note that this situation will only occur when applying an offer.
 - + Find the RtpTransceiver that corresponds to this m= section, using the mapping between transceivers and m= section indices established when creating the offer.
 - + Set the value of this RtpTransceiver's mid property to the MID of the m= section.
 - * If RTCP mux is indicated, prepare to demux RTP and RTCP from the RTP ICE component, as specified in [RFC5761], Section 5.1.3.
 - * For each specified RTP header extension, establish a mapping between the extension ID and URI, as described in [RFC5285], Section 6.
 - * If the MID header extension is supported, prepare to demux RTP streams intended for this m= section based on the MID header extension, as described in [I-D.ietf-mmusic-sdp-bundle-negotiation], Section 15.
 - * For each specified media format, establish a mapping between the payload type and the actual media format, as described in [RFC3264], Section 6.1. In addition, prepare to demux RTP streams intended for this m= section based on the media formats supported by this m= section, as described in [I-D.ietf-mmusic-sdp-bundle-negotiation], Section 10.2.
 - * For each specified "rtx" media format, establish a mapping between the RTX payload type and its associated primary payload type, as described in [RFC4588], Sections 8.6 and 8.7.

- * If the directional attribute is of type "sendrecv" or "recvonly", enable receipt and decoding of media.

Finally, if this description is of type "pranswer" or "answer", follow the processing defined in Section 5.11 below.

5.10. Applying a Remote Description

The following steps are performed to apply a remote description. If an error is returned, the session MUST be restored to the state it was in before performing these steps.

If the answer contains any "a=ice-options" attributes where "trickle" is listed as an attribute, update the PeerConnection canTrickle property to be true. Otherwise, set this property to false.

The following steps MUST be performed for attributes at the session level; if any parameters are out of bounds, or cannot be applied, processing MUST stop and an error MUST be returned.

- o For any specified "CT" bandwidth value, set this as the limit for the maximum total bitrate for all m= sections, as specified in [RFC4566], Section 5.8. Within this overall limit, the implementation can dynamically decide how to best allocate the available bandwidth between m= sections, respecting any specific limits that have been specified for individual m= sections.
- o For any specified "RR" or "RS" bandwidth values, handle as specified in [RFC3556], Section 2.
- o Any "AS" bandwidth value MUST be ignored, as the meaning of this construct at the session level is not well defined.

For each m= 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 If the ICE ufrag or password changed from the previous remote description:
 - * If the description is of type "offer", the implementation MUST note that an ICE restart is needed, as described in [I-D.ietf-mmusic-ice-sip-sdp], Section 3.4.1.1.1
 - * If the description is of type "answer" or "pranswer", then check to see if the current local description is an ICE restart, and if not, generate an error. If the PeerConnection state is "have-remote-pranswer", and the ICE ufrag or password

changed from the previous provisional answer, then signal the ICE agent to discard any previous ICE check list state for the m= section. Finally, signal the ICE agent to begin checks.

- o If the current local description indicates an ICE restart, and either the ICE ufrag or password has not changed from the previous remote description, as prescribed by [RFC8445], Section 9, generate an error.
- o Configure the ICE components associated with this media section to use the supplied ICE remote ufrag and password for their connectivity checks.
- o Pair any supplied ICE candidates with any gathered local candidates, as described in [RFC8445], Section 6.1.2, and start connectivity checks with the appropriate credentials.
- o If an "a=end-of-candidates" attribute is present, process the end-of-candidates indication as described in [I-D.ietf-ice-trickle], Section 11.
- o If the m= section proto value indicates use of RTP:
 - * If the m= section is being recycled (see Section 5.2.2), dissociate the currently associated RtpTransceiver by setting its mid property to null, and discard the mapping between the transceiver and its m= section index.
 - * If the m= section is not associated with any RtpTransceiver (possibly because it was dissociated in the previous step), either find an RtpTransceiver or create one according to the following steps:
 - + If the m= section is sendrecv or recvonly, and there are RtpTransceivers of the same type that were added to the PeerConnection by addTrack and are not associated with any m= section and are not stopped, find the first (according to the canonical order described in Section 5.2.1) such RtpTransceiver.
 - + If no RtpTransceiver was found in the previous step, create one with a recvonly direction.
 - + Associate the found or created RtpTransceiver with the m= section by setting the value of the RtpTransceiver's mid property to the MID of the m= section, and establish a mapping between the transceiver and the index of the m= section. If the m= section does not include a MID (i.e.,

the remote endpoint does not support the MID extension), generate a value for the RtpTransceiver mid property, following the guidance for "a=mid" mentioned in Section 5.2.1.

- * For each specified media format that is also supported by the local implementation, establish a mapping between the specified payload type and the media format, as described in [RFC3264], Section 6.1. Specifically, this means that the implementation records the payload type to be used in outgoing RTP packets when sending each specified media format, as well as the relative preference for each format that is indicated in their ordering. If any indicated media format is not supported by the local implementation, it MUST be ignored.
- * For each specified "rtx" media format, establish a mapping between the RTX payload type and its associated primary payload type, as described in [RFC4588], Section 4. If any referenced primary payload types are not present, this MUST result in an error. Note that RTX payload types may refer to primary payload types which are not supported by the local media implementation, in which case, the RTX payload type MUST also be ignored.
- * For each specified fmtp parameter that is supported by the local implementation, enable them on the associated media formats.
- * For each specified SSRC that is signaled in the m= section, prepare to demux RTP streams intended for this m= section using that SSRC, as described in [I-D.ietf-mmusic-sdp-bundle-negotiation], Section 10.2.
- * For each specified RTP header extension that is also supported by the local implementation, establish a mapping between the extension ID and URI, as described in [RFC5285], Section 5. Specifically, this means that the implementation records the extension ID to be used in outgoing RTP packets when sending each specified header extension. If any indicated RTP header extension is not supported by the local implementation, it MUST be ignored.
- * For each specified RTCP feedback mechanism that is supported by the local implementation, enable them on the associated media formats.
- * For any specified "TIAS" bandwidth value, set this value as a constraint on the maximum RTP bitrate to be used when sending

media, as specified in [RFC3890]. If a "TIAS" value is not present, but an "AS" value is specified, generate a "TIAS" value using this formula:

$$\text{TIAS} = \text{AS} * 1000 * 0.95 - (50 * 40 * 8)$$

The 50 is based on 50 packets per second, the 40 is based on an estimate of total header size, the 1000 changes the unit from kbps to bps (as required by TIAS), and the 0.95 is to allocate 5% to RTCP. "TIAS" is used in preference to "AS" because it provides more accurate control of bandwidth.

- * For any "RR" or "RS" bandwidth values, handle as specified in [RFC3556], Section 2.
- * Any specified "CT" bandwidth value MUST be ignored, as the meaning of this construct at the media level is not well defined.
- * If the m= section is of type audio:
 - + For each specified "CN" media format, configure silence suppression for all supported media formats with the same clockrate, as described in [RFC3389], Section 5, except for formats that have their own internal silence suppression mechanisms. Silence suppression for such formats (e.g., Opus) is controlled via fmltp parameters, as discussed in Section 5.2.3.2.
 - + For each specified "telephone-event" media format, enable DTMF transmission for all supported media formats with the same clockrate, as described in [RFC4733], Section 2.5.1.2. If there are any supported media formats that do not have a corresponding telephone-event format, disable DTMF transmission for those formats.
 - + For any specified "ptime" value, configure the available media formats to use the specified packet size when sending. If the specified size is not supported for a media format, use the next closest value instead.

Finally, if this description is of type "pranswer" or "answer", follow the processing defined in Section 5.11 below.

5.11. Applying an Answer

In addition to the steps mentioned above for processing a local or remote description, the following steps are performed when processing a description of type "pranswer" or "answer".

For each m= section, the following steps MUST be performed:

- o If the m= section has been rejected (i.e. port is set to zero in the answer), stop any reception or transmission of media for this section, and, unless a non-rejected m= section is bundled with this m= section, discard any associated ICE components, as described in [I-D.ietf-mmusic-ice-sip-sdp], Section 3.4.3.1.
- o If the remote DTLS fingerprint has been changed or the tls-id has changed, tear down the DTLS connection. This includes the case when the PeerConnection state is "have-remote-pranswer". If a DTLS connection needs to be torn down but the answer does not indicate an ICE restart or, in the case of "have-remote-pranswer", new ICE credentials, an error MUST be generated. If an ICE restart is performed without a change in tls-id or fingerprint, then the same DTLS connection is continued over the new ICE channel. Note that although JSEP requires that answerers change the tls-id value if and only if the offerer does, non-JSEP answerers are permitted to change the tls-id as long as the offer contained an ICE restart. Thus, JSEP implementations which process DTLS data prior to receiving an answer MUST be prepared to receive either a ClientHello or data from the previous DTLS connection.
- o If no valid DTLS connection exists, prepare to start a DTLS connection, using the specified roles and fingerprints, on any underlying ICE components, once they are active.
- o If the m= section proto value indicates use of RTP:
 - * If the m= section references RTCP feedback mechanisms that were not present in the corresponding m= section in the offer, this indicates a negotiation problem and MUST result in an error. However, new media formats and new RTP header extension values are permitted in the answer, as described in [RFC3264], Section 7, and [RFC5285], Section 6.
 - * If the m= section has RTCP mux enabled, discard the RTCP ICE component, if one exists, and begin or continue muxing RTCP over the RTP ICE component, as specified in [RFC5761], Section 5.1.3. Otherwise, prepare to transmit RTCP over the

RTCP ICE component; if no RTCP ICE component exists, because RTCP mux was previously enabled, this MUST result in an error.

- * If the m= section has reduced-size RTCP enabled, configure the RTCP transmission for this m= section to use reduced-size RTCP, as specified in [RFC5506].
- * If the directional attribute in the answer indicates that the JSEP implementation should be sending media ("sendonly" for local answers, "recvonly" for remote answers, or "sendrecv" for either type of answer), choose the media format to send as the most preferred media format from the remote description that is also locally supported, as discussed in [RFC3264], Sections 6.1 and 7, and start transmitting RTP media using that format once the underlying transport layers have been established. If an SSRC has not already been chosen for this outgoing RTP stream, choose a random one. If media is already being transmitted, the same SSRC SHOULD be used unless the clockrate of the new codec is different, in which case a new SSRC MUST be chosen, as specified in [RFC7160], Section 3.1.
- * The payload type mapping from the remote description is used to determine payload types for the outgoing RTP streams, including the payload type for the send media format chosen above. Any RTP header extensions that were negotiated should be included in the outgoing RTP streams, using the extension mapping from the remote description; if the RID header extension has been negotiated, and RID values are specified, include the RID header extension in the outgoing RTP streams, as indicated in [I-D.ietf-mmusic-rid], Section 4.
- * If the m= section is of type audio, and silence suppression was configured for the send media format as a result of processing the remote description, and is also enabled for that format in the local description, use silence suppression for outgoing media, in accordance with the guidance in Section 5.2.3.2. If these conditions are not met, silence suppression MUST NOT be used for outgoing media.
- * If simulcast has been negotiated, send the number of Source RTP Streams as specified in [I-D.ietf-mmusic-sdp-simulcast], Section 6.2.2.
- * If the send media format chosen above has a corresponding "rtx" media format, or a FEC mechanism has been negotiated, establish a Redundancy RTP Stream with a random SSRC for each Source RTP Stream, and start or continue transmitting RTX/FEC packets as needed.

- * If the send media format chosen above has a corresponding "red" media format of the same clockrate, allow redundant encoding using the specified format for resiliency purposes, as discussed in [I-D.ietf-rtcweb-fec], Section 3.2. Note that unlike RTX or FEC media formats, the "red" format is transmitted on the Source RTP Stream, not the Redundancy RTP Stream.
 - * Enable the RTCP feedback mechanisms referenced in the media section for all Source RTP Streams using the specified media formats. Specifically, begin or continue sending the requested feedback types and reacting to received feedback, as specified in [RFC4585], Section 4.2. When sending RTCP feedback, follow the rules and recommendations from [RFC8108] Section 5.4.1, to select which SSRC to use.
 - * If the directional attribute in the answer indicates that the JSEP implementation should not be sending media ("recvonly" for local answers, "sendonly" for remote answers, or "inactive" for either type of answer) stop transmitting all RTP media, but continue sending RTCP, as described in [RFC3264], Section 5.1.
- o If the m= section proto value indicates use of SCTP:
 - * If an SCTP association exists, and the remote SCTP port has changed, discard the existing SCTP association. This includes the case when the PeerConnection state is "have-remote-pranswer".
 - * If no valid SCTP association exists, prepare to initiate a SCTP association over the associated ICE component and DTLS connection, using the local SCTP port value from the local description, and the remote SCTP port value from the remote description, as described in [I-D.ietf-mmusic-sctp-sdp], Section 10.2.

If the answer contains valid bundle groups, discard any ICE components for the m= sections that will be bundled onto the primary ICE components in each bundle, and begin muxing these m= sections accordingly, as described in [I-D.ietf-mmusic-sdp-bundle-negotiation], Section 8.2.

If the description is of type "answer", and there are still remaining candidates in the ICE candidate pool, discard them.

6. Processing RTP/RTCP

When bundling, associating incoming RTP/RTCP with the proper m= section is defined in [I-D.ietf-mmusic-sdp-bundle-negotiation], Section 10.2. When not bundling, the proper m= section is clear from the ICE component over which the RTP/RTCP is received.

Once the proper m= section(s) are known, RTP/RTCP is delivered to the RtpTransceiver(s) associated with the m= section(s) and further processing of the RTP/RTCP is done at the RtpTransceiver level. This includes using RID [I-D.ietf-mmusic-rid] to distinguish between multiple Encoded Streams, as well as determine which Source RTP stream should be repaired by a given Redundancy RTP stream.

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, including examples with IPv6 addresses, can be found in [I-D.ietf-rtcweb-sdp].

7.1. Simple Example

This section shows a very simple example that sets up a minimal audio / video call between two JSEP endpoints without using trickle ICE. The example in the following section provides a more detailed example of what could happen in a JSEP session.

The code flow below shows Alice's endpoint initiating the session to Bob's endpoint. The messages from the JavaScript application in Alice's browser to the JavaScript in Bob's browser, abbreviated as AliceJS and BobJS respectively, are assumed to flow over some signaling protocol via a web server. The JavaScript 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. The user agents (JSEP implementations) in Alice and Bob's browsers, abbreviated as AliceUA and BobUA respectively, are using the default bundle policy of "balanced", and the default RTCP mux policy of "require".

```
//          set up local media state
AliceJS->AliceUA: create new PeerConnection
AliceJS->AliceUA: addTrack with two tracks: 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 pendingLocalDescription

//          |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:      ontrack events for audio and video tracks

//          Bob accepts call
BobJS->BobUA:      addTrack with local tracks
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 currentLocalDescription

//          |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:  ontrack events for audio and video tracks

//          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=ice-options:trickle ice2
a=group:BUNDLE a1 v1
a=group:LS a1 v1

m=audio 10100 UDP/TLS/RTP/SAVPF 96 0 8 97 98
c=IN IP4 203.0.113.100
a=mid:a1
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=fmtp:97 0-15
a=fmtp:98 0-15
a=maxptime:120
a=extmap:1 urn:ietf:params:rtp-hdext:sdes:mid
a=extmap:2 urn:ietf:params:rtp-hdext:ssrc-audio-level
a=msid:47017fee-b6c1-4162-929c-a25110252400
a=ice-ufrag:ETEn
a=ice-pwd:OtSK0WpNtpUjkY4+86js7ZQl
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=tls-id:91bbf309c0990a6bec11e38ba2933cee
a=rtcp:10101 IN IP4 203.0.113.100
a=rtcp-mux
a=rtcp-rsize
a=candidate:1 1 udp 2113929471 203.0.113.100 10100 typ host
a=candidate:1 2 udp 2113929470 203.0.113.100 10101 typ host
a=end-of-candidates

m=video 10102 UDP/TLS/RTP/SAVPF 100 101 102 103
c=IN IP4 203.0.113.100
a=mid:v1
a=sendrecv
a=rtpmap:100 VP8/90000
a=rtpmap:101 H264/90000
a=fmtp:101 packetization-mode=1;profile-level-id=42e01f
a=rtpmap:102 rtx/90000
a=fmtp:102 apt=100
a=rtpmap:103 rtx/90000
a=fmtp:103 apt=101
a=extmap:1 urn:ietf:params:rtp-hdext:sdes:mid
a=extmap:3 urn:ietf:params:rtp-hdext:sdes:rtp-stream-id
```

```
a=rtcp-fb:100 ccm fir
a=rtcp-fb:100 nack
a=rtcp-fb:100 nack pli
a=msid:47017fee-b6c1-4162-929c-a25110252400
a=ice-ufrag:BGKk
a=ice-pwd:mqyWsAjvtKwTGnvhPztQ9mIf
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=tls-id:91bbf309c0990a6bec11e38ba2933cee
a=rtcp:10103 IN IP4 203.0.113.100
a=rtcp-mux
a=rtcp-rsize
a=candidate:1 1 udp 2113929471 203.0.113.100 10102 typ host
a=candidate:1 2 udp 2113929470 203.0.113.100 10103 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=ice-options:trickle ice2
a=group:BUNDLE a1 v1
a=group:LS a1 v1

m=audio 10200 UDP/TLS/RTP/SAVPF 96 0 8 97 98
c=IN IP4 203.0.113.200
a=mid:a1
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=fmtp:97 0-15
a=fmtp:98 0-15
a=maxptime:120
a=extmap:1 urn:ietf:params:rtp-hdrext:sdes:mid
a=extmap:2 urn:ietf:params:rtp-hdrext:ssrc-audio-level
a=msid:61317484-2ed4-49d7-9eb7-1414322a7aae
a=ice-ufrag:6sFv
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=tls-id:eec3392ab83e11ceb6a0990c903fbb19
a=rtcp-mux
a=rtcp-rsize
a=candidate:1 1 udp 2113929471 203.0.113.200 10200 typ host
a=end-of-candidates

m=video 10200 UDP/TLS/RTP/SAVPF 100 101 102 103
c=IN IP4 203.0.113.200
a=mid:v1
a=sendrecv
a=rtpmap:100 VP8/90000
a=rtpmap:101 H264/90000
a=fmtp:101 packetization-mode=1;profile-level-id=42e01f
a=rtpmap:102 rtx/90000
a=fmtp:102 apt=100
a=rtpmap:103 rtx/90000
a=fmtp:103 apt=101
a=extmap:1 urn:ietf:params:rtp-hdrext:sdes:mid
a=extmap:3 urn:ietf:params:rtp-hdrext:sdes:rtp-stream-id
a=rtcp-fb:100 ccm fir
a=rtcp-fb:100 nack
a=rtcp-fb:100 nack pli
a=msid:61317484-2ed4-49d7-9eb7-1414322a7aae

```

7.2. Detailed Example

This section shows a more involved example of a session between two JSEP endpoints. 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. Initially, both Alice and Bob establish an audio channel and a data channel. Later, Bob adds two video flows, one for his video feed, and one for screensharing, both supporting FEC, and with the video feed configured for simulcast. Alice accepts these video flows, but does not add video flows of her own, so they are handled as recvonly. Alice also specifies a maximum video decoder resolution.

```

//      set up local media state
AliceJS->AliceUA: create new PeerConnection
AliceJS->AliceUA: addTrack with an 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:      ontrack with audio track from Alice

//          candidates are sent to Bob
AliceUA->AliceJS:  onicecandidate (host) |offer-B1-candidate-1|
AliceJS->WebServer: signaling with |offer-B1-candidate-1|
AliceUA->AliceJS:  onicecandidate (srflx) |offer-B1-candidate-2|
AliceJS->WebServer: signaling with |offer-B1-candidate-2|
AliceUA->AliceJS:  onicecandidate (relay) |offer-B1-candidate-3|
AliceJS->WebServer: signaling with |offer-B1-candidate-3|

WebServer->BobJS:  signaling with |offer-B1-candidate-1|
BobJS->BobUA:      addIceCandidate with |offer-B1-candidate-1|
WebServer->BobJS:  signaling with |offer-B1-candidate-2|
BobJS->BobUA:      addIceCandidate with |offer-B1-candidate-2|
WebServer->BobJS:  signaling with |offer-B1-candidate-3|
BobJS->BobUA:      addIceCandidate with |offer-B1-candidate-3|

//          Bob accepts call
BobJS->BobUA:      addTrack with local audio
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:  ontrack event with audio track from Bob

//          candidates are sent to Alice
BobUA->BobJS:      onicecandidate (host) |answer-B1-candidate-1|
BobJS->WebServer:  signaling with |answer-B1-candidate-1|
BobUA->BobJS:      onicecandidate (srflx) |answer-B1-candidate-2|
BobJS->WebServer:  signaling with |answer-B1-candidate-2|
BobUA->BobJS:      onicecandidate (relay) |answer-B1-candidate-3|
BobJS->WebServer:  signaling with |answer-B1-candidate-3|

WebServer->AliceJS: signaling with |answer-B1-candidate-1|
AliceJS->AliceUA:  addIceCandidate with |answer-B1-candidate-1|
WebServer->AliceJS: signaling with |answer-B1-candidate-2|
AliceJS->AliceUA:  addIceCandidate with |answer-B1-candidate-2|
```

```
WebServer->AliceJS: signaling with |answer-B1-candidate-3|
AliceJS->AliceUA:   addIceCandidate with |answer-B1-candidate-3|

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

//                  media is flowing between endpoints
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:       addTrack with first video stream
BobJS->BobUA:       addTrack 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:   ontrack event with first video track
AliceUA->AliceJS:   ontrack event with second video track
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 endpoints
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=ice-options:trickle ice2
a=group:BUNDLE a1 d1

m=audio 9 UDP/TLS/RTP/SAVPF 96 0 8 97 98
c=IN IP4 0.0.0.0
a=mid:a1
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=fmtp:97 0-15
a=fmtp:98 0-15
a=maxptime:120
a=extmap:1 urn:ietf:params:rtp-hdrext:sdes:mid
a=extmap:2 urn:ietf:params:rtp-hdrext:ssrc-audio-level
a=msid:57017fee-b6c1-4162-929c-a25110252400
a=ice-ufrag:ATEn
a=ice-pwd:AtSK0WpNtpUjkY4+86js7ZQl
a=fingerprint:sha-256
                29: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=tls-id:17f0f4ba8a5f1213faca591b58ba52a7
a=rtcp-mux
a=rtcp-mux-only
a=rtcp-rsize

m=application 0 UDP/DTLS/SCTP webrtc-datachannel
c=IN IP4 0.0.0.0
a=mid:d1
a=sctp-port:5000
a=max-message-size:65536
a=bundle-only
```

|offer-B1-candidate-1| looks like:

```
ufrag ATEn
index 0
mid a1
attr candidate:1 1 udp 2113929471 203.0.113.100 10100 typ host
```

|offer-B1-candidate-2| looks like:

```
ufrag ATEn
index 0
mid a1
attr candidate:1 1 udp 1845494015 198.51.100.100 11100 typ srflx
      raddr 203.0.113.100 rport 10100
```

|offer-B1-candidate-3| looks like:

```
ufrag ATEn
index 0
mid a1
attr candidate:1 1 udp 255 192.0.2.100 12100 typ relay
      raddr 198.51.100.100 rport 11100
```

The SDP for |answer-B1| looks like:

```
v=0
o=- 7729291447651054566 1 IN IP4 0.0.0.0
s=-
t=0 0
a=ice-options:trickle ice2
a=group:BUNDLE a1 d1

m=audio 9 UDP/TLS/RTP/SAVPF 96 0 8 97 98
c=IN IP4 0.0.0.0
a=mid:a1
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=fmtp:97 0-15
a=fmtp:98 0-15
a=maxptime:120
a=extmap:1 urn:ietf:params:rtp-hdrext:sdes:mid
a=extmap:2 urn:ietf:params:rtp-hdrext:ssrc-audio-level
a=msid:71317484-2ed4-49d7-9eb7-1414322a7aae
a=ice-ufrag:7sFv
a=ice-pwd:DOTZKZNVlO9RSGsEGM63JXT2
a=fingerprint:sha-256
          7B: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=tls-id:7a25ab85b195acaf3121f5a8ab4f0f71
a=rtcp-mux
a=rtcp-mux-only
a=rtcp-rsize

m=application 9 UDP/DTLS/SCTP webrtc-datachannel
c=IN IP4 0.0.0.0
a=mid:d1
a=sctp-port:5000
a=max-message-size:65536
```

|answer-B1-candidate-1| looks like:

```
ufrag 7sFv
index 0
mid a1
attr candidate:1 1 udp 2113929471 203.0.113.200 10200 typ host
```

|answer-B1-candidate-2| looks like:

```
ufrag 7sFv
index 0
mid a1
attr candidate:1 1 udp 1845494015 198.51.100.200 11200 typ srflx
      raddr 203.0.113.200 rport 10200
```

|answer-B1-candidate-3| looks like:

```
ufrag 7sFv
index 0
mid a1
attr candidate:1 1 udp 255 192.0.2.200 12200 typ relay
      raddr 198.51.100.200 rport 11200
```

The SDP for |offer-B2| is shown below. In addition to the new m= sections for video, both of which are offering FEC, and one of which is offering simulcast, note the increment of the version number in the o= line, changes to the c= line, indicating the local candidate that was selected, and the inclusion of gathered candidates as a=candidate lines.

```
v=0
o=- 7729291447651054566 2 IN IP4 0.0.0.0
s=-
t=0 0
a=ice-options:trickle ice2
a=group:BUNDLE a1 d1 v1 v2
a=group:LS a1 v1

m=audio 12200 UDP/TLS/RTP/SAVPF 96 0 8 97 98
c=IN IP4 192.0.2.200
a=mid:a1
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=fmtp:97 0-15
a=fmtp:98 0-15
a=maxptime:120
```

```
a=extmap:1 urn:ietf:params:rtp-hdrext:sdes:mid
a=extmap:2 urn:ietf:params:rtp-hdrext:ssrc-audio-level
a=msid:71317484-2ed4-49d7-9eb7-1414322a7aae
a=ice-ufrag:7sFv
a=ice-pwd:DOTZKZNVlO9RSGsEGM63JXT2
a=fingerprint:sha-256
      7B: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=tls-id:7a25ab85b195acaf3121f5a8ab4f0f71
a=rtcp-mux
a=rtcp-mux-only
a=rtcp-rsize
a=candidate:1 1 udp 2113929471 203.0.113.200 10200 typ host
a=candidate:1 1 udp 1845494015 198.51.100.200 11200 typ srflx
      raddr 203.0.113.200 rport 10200
a=candidate:1 1 udp 255 192.0.2.200 12200 typ relay
      raddr 198.51.100.200 rport 11200
a=end-of-candidates

m=application 12200 UDP/DTLS/SCTP webrtc-datachannel
c=IN IP4 192.0.2.200
a=mid:d1
a=sctp-port:5000
a=max-message-size:65536

m=video 12200 UDP/TLS/RTP/SAVPF 100 101 102 103 104
c=IN IP4 192.0.2.200
a=mid:v1
a=sendrecv
a=rtpmap:100 VP8/90000
a=rtpmap:101 H264/90000
a=fmtp:101 packetization-mode=1;profile-level-id=42e01f
a=rtpmap:102 rtx/90000
a=fmtp:102 apt=100
a=rtpmap:103 rtx/90000
a=fmtp:103 apt=101
a=rtpmap:104 flexfec/90000
a=extmap:1 urn:ietf:params:rtp-hdrext:sdes:mid
a=extmap:3 urn:ietf:params:rtp-hdrext:sdes:rtp-stream-id
a=rtcp-fb:100 ccm fir
a=rtcp-fb:100 nack
a=rtcp-fb:100 nack pli
a=msid:71317484-2ed4-49d7-9eb7-1414322a7aae
a=rid:1 send
a=rid:2 send
a=rid:3 send
a=simulcast:send 1;2;3
```



```
m=video 12200 UDP/TLS/RTP/SAVPF 100 101 102 103 104
c=IN IP4 192.0.2.200
a=mid:v2
a=sendrecv
a=rtpmap:100 VP8/90000
a=rtpmap:101 H264/90000
a=fmtp:101 packetization-mode=1;profile-level-id=42e01f
a=rtpmap:102 rtx/90000
a=fmtp:102 apt=100
a=rtpmap:103 rtx/90000
a=fmtp:103 apt=101
a=rtpmap:104 flexfec/90000
a=extmap:1 urn:ietf:params:rtp-hdrext:sdes:mid
a=extmap:3 urn:ietf:params:rtp-hdrext:sdes:rtp-stream-id
a=rtcp-fb:100 ccm fir
a=rtcp-fb:100 nack
a=rtcp-fb:100 nack pli
a=msid:81317484-2ed4-49d7-9eb7-1414322a7aae
```

The SDP for |answer-B2| is shown below. In addition to the acceptance of the video m= sections, the use of a=recvonly to indicate one-way video, and the use of a=imageattr to limit the received resolution, note the use of setup:passive to maintain the existing DTLS roles.

```
v=0
o=- 4962303333179871723 2 IN IP4 0.0.0.0
s=-
t=0 0
a=ice-options:trickle ice2
a=group:BUNDLE a1 d1 v1 v2
a=group:LS a1 v1

m=audio 12100 UDP/TLS/RTP/SAVPF 96 0 8 97 98
c=IN IP4 192.0.2.100
a=mid:a1
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=fmtp:97 0-15
a=fmtp:98 0-15
a=maxptime:120
a=extmap:1 urn:ietf:params:rtp-hdrext:sdes:mid
```

```
a=extmap:2 urn:ietf:params:rtp-hdrext:ssrc-audio-level
a=msid:57017fee-b6c1-4162-929c-a25110252400
a=ice-ufrag:ATEn
a=ice-pwd:AtSK0WpNtpUjky4+86js7ZQl
a=fingerprint:sha-256
    29: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=tls-id:17f0f4ba8a5f1213faca591b58ba52a7
a=rtcp-mux
a=rtcp-mux-only
a=rtcp-rsize
a=candidate:1 1 udp 2113929471 203.0.113.100 10100 typ host
a=candidate:1 1 udp 1845494015 198.51.100.100 11100 typ srflx
    raddr 203.0.113.100 rport 10100
a=candidate:1 1 udp 255 192.0.2.100 12100 typ relay
    raddr 198.51.100.100 rport 11100
a=end-of-candidates

m=application 12100 UDP/DTLS/SCTP webrtc-datachannel
c=IN IP4 192.0.2.100
a=mid:d1
a=sctp-port:5000
a=max-message-size:65536

m=video 12100 UDP/TLS/RTP/SAVPF 100 101 102 103
c=IN IP4 192.0.2.100
a=mid:v1
a=recvonly
a=rtpmap:100 VP8/90000
a=rtpmap:101 H264/90000
a=fmtp:101 packetization-mode=1;profile-level-id=42e01f
a=rtpmap:102 rtx/90000
a=fmtp:102 apt=100
a=rtpmap:103 rtx/90000
a=fmtp:103 apt=101
a=imageattr:100 recv [x=[48:1920],y=[48:1080],q=1.0]
a=extmap:1 urn:ietf:params:rtp-hdrext:sdes:mid
a=extmap:3 urn:ietf:params:rtp-hdrext:sdes:rtp-stream-id
a=rtcp-fb:100 ccm fir
a=rtcp-fb:100 nack
a=rtcp-fb:100 nack pli

m=video 12100 UDP/TLS/RTP/SAVPF 100 101 102 103
c=IN IP4 192.0.2.100
a=mid:v2
a=recvonly
a=rtpmap:100 VP8/90000
```

```

a=rtpmap:101 H264/90000
a=fmtp:101 packetization-mode=1;profile-level-id=42e01f
a=rtpmap:102 rtx/90000
a=fmtp:102 apt=100
a=rtpmap:103 rtx/90000
a=fmtp:103 apt=101
a=imageattr:100 recv [x=[48:1920],y=[48:1080],q=1.0]
a=extmap:1 urn:ietf:params:rtp-hdrext:sdes:mid
a=extmap:3 urn:ietf:params:rtp-hdrext:sdes:rtp-stream-id
a=rtcp-fb:100 ccm fir
a=rtcp-fb:100 nack
a=rtcp-fb:100 nack pli

```

7.3. Early Transport Warmup Example

This example demonstrates the early warmup technique described in Section 4.1.8.1. Here, Alice's endpoint sends an offer to Bob's endpoint to start an audio/video call. Bob immediately responds with an answer that accepts the audio/video m= sections, but marks them as sendonly (from his perspective), meaning that Alice will not yet send media. This allows the JSEP implementation to start negotiating ICE and DTLS immediately. Bob's endpoint then prompts him to answer the call, and when he does, his endpoint sends a second offer which enables the audio and video m= sections, and thereby bidirectional media transmission. The advantage of such a flow is that as soon as the first answer is received, the implementation can proceed with ICE and DTLS negotiation and establish the session transport. If the transport setup completes before the second offer is sent, then media can be transmitted immediately by the callee immediately upon answering the call, minimizing perceived post-dial-delay. The second offer/answer exchange can also change the preferred codecs or other session parameters.

This example also makes use of the "relay" ICE candidate policy described in Section 3.5.3 to minimize the ICE gathering and checking needed.

```

//                               set up local media state
AliceJS->AliceUA:               create new PeerConnection with "relay" ICE policy
AliceJS->AliceUA:               addTrack with two tracks: audio and video
AliceJS->AliceUA:               createOffer to get |offer-C1|
AliceJS->AliceUA:               setLocalDescription with |offer-C1|

//                               |offer-C1| is sent over signaling protocol to Bob
AliceJS->WebServer:              signaling with |offer-C1|
WebServer->BobJS:               signaling with |offer-C1|

```

```
//          |offer-C1| arrives at Bob
BobJS->BobUA: create new PeerConnection with "relay" ICE policy
BobJS->BobUA: setRemoteDescription with |offer-C1|
BobUA->BobJS: ontrack events for audio and video

//          a relay candidate is sent to Bob
AliceUA->AliceJS: onicecandidate (relay) |offer-C1-candidate-1|
AliceJS->WebServer: signaling with |offer-C1-candidate-1|

WebServer->BobJS: signaling with |offer-C1-candidate-1|
BobJS->BobUA: addIceCandidate with |offer-C1-candidate-1|

//          Bob prepares an early answer to warmup the transport
BobJS->BobUA: addTransceiver with null audio and video tracks
BobJS->BobUA: transceiver.setDirection(sendonly) for both
BobJS->BobUA: createAnswer
BobJS->BobUA: setLocalDescription with answer

//          |answer-C1| is sent over signaling protocol to Alice
BobJS->WebServer: signaling with |answer-C1|
WebServer->AliceJS: signaling with |answer-C1|

//          |answer-C1| (sendonly) arrives at Alice
AliceJS->AliceUA: setRemoteDescription with |answer-C1|
AliceUA->AliceJS: ontrack events for audio and video

//          a relay candidate is sent to Alice
BobUA->BobJS: onicecandidate (relay) |answer-B1-candidate-1|
BobJS->WebServer: signaling with |answer-B1-candidate-1|

WebServer->AliceJS: signaling with |answer-B1-candidate-1|
AliceJS->AliceUA: addIceCandidate with |answer-B1-candidate-1|

//          ICE and DTLS establish while call is ringing

//          Bob accepts call, starts media, and sends new offer
BobJS->BobUA: transceiver.setTrack with audio and video tracks
BobUA->AliceUA: media sent from Bob to Alice
BobJS->BobUA: transceiver.setDirection(sendrecv) for both
transceivers
BobJS->BobUA: createOffer
BobJS->BobUA: setLocalDescription with offer

//          |offer-C2| is sent over signaling protocol to Alice
BobJS->WebServer: signaling with |offer-C2|
WebServer->AliceJS: signaling with |offer-C2|

//          |offer-C2| (sendrecv) arrives at Alice
```

```
AliceJS->AliceUA: setRemoteDescription with |offer-C2|
AliceJS->AliceUA: createAnswer
AliceJS->AliceUA: setLocalDescription with |answer-C2|
AliceUA->BobUA:   media sent from Alice to Bob

//           |answer-C2| is sent over signaling protocol to Bob
AliceJS->WebServer: signaling with |answer-C2|
WebServer->BobJS:   signaling with |answer-C2|
BobJS->BobUA:       setRemoteDescription with |answer-C2|
```

The SDP for |offer-C1| looks like:

```
v=0
o=- 1070771854436052752 1 IN IP4 0.0.0.0
s=-
t=0 0
a=ice-options:trickle ice2
a=group:BUNDLE a1 v1
a=group:LS a1 v1

m=audio 9 UDP/TLS/RTP/SAVPF 96 0 8 97 98
c=IN IP4 0.0.0.0
a=mid:a1
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=fmtp:97 0-15
a=fmtp:98 0-15
a=maxptime:120
a=extmap:1 urn:ietf:params:rtp-hdrext:sdes:mid
a=extmap:2 urn:ietf:params:rtp-hdrext:ssrc-audio-level
a=msid:bbce3ba6-abfc-ac63-d00a-e15b286f8fce
a=ice-ufrag:4ZcD
a=ice-pwd:ZaaG6OG7tCn4J/lehAGz+HHD
a=fingerprint:sha-256
                C4:68:F8:77:6A:44:F1:98:6D:7C:9F:47:EB:E3:34:A4:
                0A:AA:2D:49:08:28:70:2E:1F:AE:18:7D:4E:3E:66:BF
a=setup:actpass
a=tls-id:9e5b948ade9c3d41de6617b68f769e55
a=rtcp-mux
a=rtcp-mux-only
a=rtcp-rsize
```

```
m=video 0 UDP/TLS/RTP/SAVPF 100 101 102 103
c=IN IP4 0.0.0.0
a=mid:v1
a=sendrecv
a=rtpmap:100 VP8/90000
a=rtpmap:101 H264/90000
a=fmtp:101 packetization-mode=1;profile-level-id=42e01f
a=rtpmap:102 rtx/90000
a=fmtp:102 apt=100
a=rtpmap:103 rtx/90000
a=fmtp:103 apt=101
a=extmap:1 urn:ietf:params:rtp-hdrext:sdes:mid
a=extmap:3 urn:ietf:params:rtp-hdrext:sdes:rtp-stream-id
a=rtcp-fb:100 ccm fir
a=rtcp-fb:100 nack
a=rtcp-fb:100 nack pli
a=msid:bbce3ba6-abfc-ac63-d00a-e15b286f8fce
a=bundle-only
```

|offer-C1-candidate-1| looks like:

```
ufrag 4ZcD
index 0
mid a1
attr candidate:1 1 udp 255 192.0.2.100 12100 typ relay
      raddr 0.0.0.0 rport 0
```

The SDP for |answer-C1| looks like:

```
v=0
o=- 6386516489780559513 1 IN IP4 0.0.0.0
s=-
t=0 0
a=ice-options:trickle ice2
a=group:BUNDLE a1 v1
a=group:LS a1 v1

m=audio 9 UDP/TLS/RTP/SAVPF 96 0 8 97 98
c=IN IP4 0.0.0.0
a=mid:a1
a=sendonly
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=fmtp:97 0-15
a=fmtp:98 0-15
a=maxptime:120
a=extmap:1 urn:ietf:params:rtp-hdrext:sdes:mid
a=extmap:2 urn:ietf:params:rtp-hdrext:ssrc-audio-level
a=msid:751f239e-4ae0-c549-aa3d-890de772998b
a=ice-ufrag:TpaA
a=ice-pwd:t2Ouhc67y8JcCaYZxUUTgKw/
a=fingerprint:sha-256
                A2:F3:A5:6D:4C:8C:1E:B2:62:10:4A:F6:70:61:C4:FC:
                3C:E0:01:D6:F3:24:80:74:DA:7C:3E:50:18:7B:CE:4D
a=setup:active
a=tls-id:55e967f86b7166ed14d3c9eda849b5e9
a=rtcp-mux
a=rtcp-mux-only
a=rtcp-rsize

m=video 9 UDP/TLS/RTP/SAVPF 100 101 102 103
c=IN IP4 0.0.0.0
a=mid:v1
a=sendonly
a=rtpmap:100 VP8/90000
a=rtpmap:101 H264/90000
a=fmtp:101 packetization-mode=1;profile-level-id=42e01f
a=rtpmap:102 rtx/90000
a=fmtp:102 apt=100
=rtpmap:103 rtx/90000
a=fmtp:103 apt=101
a=extmap:1 urn:ietf:params:rtp-hdrext:sdes:mid
a=extmap:3 urn:ietf:params:rtp-hdrext:sdes:rtp-stream-id
a=rtcp-fb:100 ccm fir
a=rtcp-fb:100 nack
a=rtcp-fb:100 nack pli
a=msid:751f239e-4ae0-c549-aa3d-890de772998b
```

|answer-C1-candidate-1| looks like:

```
ufrag TpaA
index 0
mid a1
attr candidate:1 1 udp 255 192.0.2.200 12200 typ relay
                raddr 0.0.0.0 rport 0
```

The SDP for |offer-C2| looks like:

```
v=0
o=- 6386516489780559513 2 IN IP4 0.0.0.0
s=-
t=0 0
a=ice-options:trickle ice2
a=group:BUNDLE a1 v1
a=group:LS a1 v1

m=audio 12200 UDP/TLS/RTP/SAVPF 96 0 8 97 98
c=IN IP4 192.0.2.200
a=mid:a1
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=fmtp:97 0-15
a=fmtp:98 0-15
a=maxptime:120
a=extmap:1 urn:ietf:params:rtp-hdext:sdes:mid
a=extmap:2 urn:ietf:params:rtp-hdext:ssrc-audio-level
a=msid:751f239e-4ae0-c549-aa3d-890de772998b
a=ice-ufrag:TpaA
a=ice-pwd:t2Ouhc67y8JcCaYZxUUTgKw/
a=fingerprint:sha-256
          A2:F3:A5:6D:4C:8C:1E:B2:62:10:4A:F6:70:61:C4:FC:
          3C:E0:01:D6:F3:24:80:74:DA:7C:3E:50:18:7B:CE:4D
a=setup:actpass
a=tls-id:55e967f86b7166ed14d3c9eda849b5e9
a=rtcp-mux
a=rtcp-mux-only
a=rtcp-rsize
a=candidate:1 1 udp 255 192.0.2.200 12200 typ relay
          raddr 0.0.0.0 rport 0
a=end-of-candidates

m=video 12200 UDP/TLS/RTP/SAVPF 100 101 102 103
c=IN IP4 192.0.2.200
a=mid:v1
a=sendrecv
a=rtpmap:100 VP8/90000
a=rtpmap:101 H264/90000
a=fmtp:101 packetization-mode=1;profile-level-id=42e01f
a=rtpmap:102 rtx/90000
```



```
a=fmtp:102 apt=100
=rtpmap:103 rtx/90000
a=fmtp:103 apt=101
a=extmap:1 urn:ietf:params:rtp-hdrext:sdes:mid
a=extmap:3 urn:ietf:params:rtp-hdrext:sdes:rtp-stream-id
a=rtcp-fb:100 ccm fir
a=rtcp-fb:100 nack
a=rtcp-fb:100 nack pli
a=msid:751f239e-4ae0-c549-aa3d-890de772998b
```

The SDP for |answer-C2| looks like:

```
v=0
o=- 1070771854436052752 2 IN IP4 0.0.0.0
s=-
t=0 0
a=ice-options:trickle ice2
a=group:BUNDLE a1 v1
a=group:LS a1 v1

m=audio 12100 UDP/TLS/RTP/SAVPF 96 0 8 97 98
c=IN IP4 192.0.2.100
a=mid:a1
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=fmtp:97 0-15
a=fmtp:98 0-15
a=maxptime:120
a=extmap:1 urn:ietf:params:rtp-hdrext:sdes:mid
a=extmap:2 urn:ietf:params:rtp-hdrext:ssrc-audio-level
a=msid:bbce3ba6-abfc-ac63-d00a-e15b286f8fce
a=ice-ufrag:4ZcD
a=ice-pwd:ZaaG6OG7tCn4J/lehAGz+HHD
a=fingerprint:sha-256
          C4:68:F8:77:6A:44:F1:98:6D:7C:9F:47:EB:E3:34:A4:
          0A:AA:2D:49:08:28:70:2E:1F:AE:18:7D:4E:3E:66:BF
a=setup:passive
a=tls-id:9e5b948ade9c3d41de6617b68f769e55
a=rtcp-mux
a=rtcp-mux-only
a=rtcp-rsize
a=candidate:1 1 udp 255 192.0.2.100 12100 typ relay
```

```
        raddr 0.0.0.0 rport 0
a=end-of-candidates

m=video 12100 UDP/TLS/RTP/SAVPF 100 101 102 103
c=IN IP4 192.0.2.100
a=mid:v1
a=sendrecv
a=rtpmap:100 VP8/90000
a=rtpmap:101 H264/90000
a=fmtp:101 packetization-mode=1;profile-level-id=42e01f
a=rtpmap:102 rtx/90000
a=fmtp:102 apt=100
a=rtpmap:103 rtx/90000
a=fmtp:103 apt=101
a=extmap:1 urn:ietf:params:rtp-hdrext:sdes:mid
a=extmap:3 urn:ietf:params:rtp-hdrext:sdes:rtp-stream-id
a=rtcp-fb:100 ccm fir
a=rtcp-fb:100 nack
a=rtcp-fb:100 nack pli
a=msid:bbce3ba6-abfc-ac63-d00a-e15b286f8fce
```

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 as an Internet protocol, with the application JavaScript being untrustworthy from the perspective of the JSEP implementation. Thus, the threat model of [RFC3552] applies. In particular, JavaScript 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()`, there is no guarantee that applications do so. The JSEP implementation MUST be prepared for the JavaScript to pass in bogus data instead.

Conversely, the application programmer needs to be aware that the JavaScript does not have complete control of endpoint 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

Harald Alvestrand, Taylor Brandstetter, Suhas Nandakumar, and Peter Thatcher provided significant text for this draft. Bernard Aboba, Adam Bergkvist, Dan Burnett, Ben Campbell, Alissa Cooper, Richard Ejzak, Stefan Hakansson, Ted Hardie, Christer Holmberg Andrew Hutton, Randell Jesup, Matthew Kaufman, Anant Narayanan, Adam Roach, Robert Sparks, Neil Stratford, Martin Thomson, Sean Turner, and Magnus Westerlund all provided valuable feedback on this proposal.

11. References

11.1. Normative References

[I-D.ietf-avtext-rid]

Roach, A., Nandakumar, S., and P. Thatcher, "RTP Stream Identifier Source Description (SDS)", draft-ietf-avtext-rid-09 (work in progress), October 2016.

[I-D.ietf-ice-trickle]

Ivov, E., Rescorla, E., Uberti, J., and P. Saint-Andre, "Trickle ICE: Incremental Provisioning of Candidates for the Interactive Connectivity Establishment (ICE) Protocol", draft-ietf-ice-trickle-21 (work in progress), April 2018.

[I-D.ietf-mmusic-dtls-sdp]

Holmberg, C. and R. Shpount, "Session Description Protocol (SDP) Offer/Answer Considerations for Datagram Transport Layer Security (DTLS) and Transport Layer Security (TLS)", draft-ietf-mmusic-dtls-sdp-32 (work in progress), October 2017.

[I-D.ietf-mmusic-ice-sip-sdp]

Petit-Huguenin, M., Nandakumar, S., and A. Keranen, "Session Description Protocol (SDP) Offer/Answer procedures for Interactive Connectivity Establishment (ICE)", draft-ietf-mmusic-ice-sip-sdp-24 (work in progress), November 2018.

- [I-D.ietf-mmusic-msid]
Alvestrand, H., "WebRTC MediaStream Identification in the Session Description Protocol", draft-ietf-mmusic-msid-17 (work in progress), December 2018.
- [I-D.ietf-mmusic-mux-exclusive]
Holmberg, C., "Indicating Exclusive Support of RTP/RTCP Multiplexing using SDP", draft-ietf-mmusic-mux-exclusive-12 (work in progress), May 2017.
- [I-D.ietf-mmusic-rid]
Roach, A., "RTP Payload Format Restrictions", draft-ietf-mmusic-rid-15 (work in progress), May 2018.
- [I-D.ietf-mmusic-sctp-sdp]
Holmberg, C., Shpount, R., Loreto, S., and G. Camarillo, "Session Description Protocol (SDP) Offer/Answer Procedures For Stream Control Transmission Protocol (SCTP) over Datagram Transport Layer Security (DTLS) Transport.", draft-ietf-mmusic-sctp-sdp-26 (work in progress), April 2017.
- [I-D.ietf-mmusic-sdp-bundle-negotiation]
Holmberg, C., Alvestrand, H., and C. Jennings, "Negotiating Media Multiplexing Using the Session Description Protocol (SDP)", draft-ietf-mmusic-sdp-bundle-negotiation-54 (work in progress), December 2018.
- [I-D.ietf-mmusic-sdp-mux-attributes]
Nandakumar, S., "A Framework for SDP Attributes when Multiplexing", draft-ietf-mmusic-sdp-mux-attributes-17 (work in progress), February 2018.
- [I-D.ietf-mmusic-sdp-simulcast]
Burman, B., Westerlund, M., Nandakumar, S., and M. Zanaty, "Using Simulcast in SDP and RTP Sessions", draft-ietf-mmusic-sdp-simulcast-13 (work in progress), June 2018.
- [I-D.ietf-rtcweb-fec]
Uberti, J., "WebRTC Forward Error Correction Requirements", draft-ietf-rtcweb-fec-08 (work in progress), March 2018.
- [I-D.ietf-rtcweb-rtp-usage]
Perkins, C., Westerlund, M., and J. Ott, "Web Real-Time Communication (WebRTC): Media Transport and Use of RTP", draft-ietf-rtcweb-rtp-usage-26 (work in progress), March 2016.

- [I-D.ietf-rtcweb-security]
Rescorla, E., "Security Considerations for WebRTC", draft-ietf-rtcweb-security-11 (work in progress), February 2019.
- [I-D.ietf-rtcweb-security-arch]
Rescorla, E., "WebRTC Security Architecture", draft-ietf-rtcweb-security-arch-18 (work in progress), February 2019.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<https://www.rfc-editor.org/info/rfc2119>>.
- [RFC3261] Rosenberg, J., Schulzrinne, H., Camarillo, G., Johnston, A., Peterson, J., Sparks, R., Handley, M., and E. Schooler, "SIP: Session Initiation Protocol", RFC 3261, DOI 10.17487/RFC3261, June 2002, <<https://www.rfc-editor.org/info/rfc3261>>.
- [RFC3264] Rosenberg, J. and H. Schulzrinne, "An Offer/Answer Model with Session Description Protocol (SDP)", RFC 3264, DOI 10.17487/RFC3264, June 2002, <<https://www.rfc-editor.org/info/rfc3264>>.
- [RFC3552] Rescorla, E. and B. Korver, "Guidelines for Writing RFC Text on Security Considerations", BCP 72, RFC 3552, DOI 10.17487/RFC3552, July 2003, <<https://www.rfc-editor.org/info/rfc3552>>.
- [RFC3605] Huitema, C., "Real Time Control Protocol (RTCP) attribute in Session Description Protocol (SDP)", RFC 3605, DOI 10.17487/RFC3605, October 2003, <<https://www.rfc-editor.org/info/rfc3605>>.
- [RFC3711] Baugher, M., McGrew, D., Naslund, M., Carrara, E., and K. Norrman, "The Secure Real-time Transport Protocol (SRTP)", RFC 3711, DOI 10.17487/RFC3711, March 2004, <<https://www.rfc-editor.org/info/rfc3711>>.
- [RFC3890] Westerlund, M., "A Transport Independent Bandwidth Modifier for the Session Description Protocol (SDP)", RFC 3890, DOI 10.17487/RFC3890, September 2004, <<https://www.rfc-editor.org/info/rfc3890>>.
- [RFC4145] Yon, D. and G. Camarillo, "TCP-Based Media Transport in the Session Description Protocol (SDP)", RFC 4145, DOI 10.17487/RFC4145, September 2005, <<https://www.rfc-editor.org/info/rfc4145>>.

- [RFC4566] Handley, M., Jacobson, V., and C. Perkins, "SDP: Session Description Protocol", RFC 4566, DOI 10.17487/RFC4566, July 2006, <<https://www.rfc-editor.org/info/rfc4566>>.
- [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, DOI 10.17487/RFC4585, July 2006, <<https://www.rfc-editor.org/info/rfc4585>>.
- [RFC5124] Ott, J. and E. Carrara, "Extended Secure RTP Profile for Real-time Transport Control Protocol (RTCP)-Based Feedback (RTP/SAVPF)", RFC 5124, DOI 10.17487/RFC5124, February 2008, <<https://www.rfc-editor.org/info/rfc5124>>.
- [RFC5285] Singer, D. and H. Desineni, "A General Mechanism for RTP Header Extensions", RFC 5285, DOI 10.17487/RFC5285, July 2008, <<https://www.rfc-editor.org/info/rfc5285>>.
- [RFC5761] Perkins, C. and M. Westerlund, "Multiplexing RTP Data and Control Packets on a Single Port", RFC 5761, DOI 10.17487/RFC5761, April 2010, <<https://www.rfc-editor.org/info/rfc5761>>.
- [RFC5888] Camarillo, G. and H. Schulzrinne, "The Session Description Protocol (SDP) Grouping Framework", RFC 5888, DOI 10.17487/RFC5888, June 2010, <<https://www.rfc-editor.org/info/rfc5888>>.
- [RFC6236] Johansson, I. and K. Jung, "Negotiation of Generic Image Attributes in the Session Description Protocol (SDP)", RFC 6236, DOI 10.17487/RFC6236, May 2011, <<https://www.rfc-editor.org/info/rfc6236>>.
- [RFC6347] Rescorla, E. and N. Modadugu, "Datagram Transport Layer Security Version 1.2", RFC 6347, DOI 10.17487/RFC6347, January 2012, <<https://www.rfc-editor.org/info/rfc6347>>.
- [RFC6716] Valin, JM., Vos, K., and T. Terriberry, "Definition of the Opus Audio Codec", RFC 6716, DOI 10.17487/RFC6716, September 2012, <<https://www.rfc-editor.org/info/rfc6716>>.
- [RFC6904] Lennox, J., "Encryption of Header Extensions in the Secure Real-time Transport Protocol (SRTP)", RFC 6904, DOI 10.17487/RFC6904, April 2013, <<https://www.rfc-editor.org/info/rfc6904>>.

- [RFC7160] Petit-Huguenin, M. and G. Zorn, Ed., "Support for Multiple Clock Rates in an RTP Session", RFC 7160, DOI 10.17487/RFC7160, April 2014, <<https://www.rfc-editor.org/info/rfc7160>>.
- [RFC7587] Spittka, J., Vos, K., and JM. Valin, "RTP Payload Format for the Opus Speech and Audio Codec", RFC 7587, DOI 10.17487/RFC7587, June 2015, <<https://www.rfc-editor.org/info/rfc7587>>.
- [RFC7742] Roach, A., "WebRTC Video Processing and Codec Requirements", RFC 7742, DOI 10.17487/RFC7742, March 2016, <<https://www.rfc-editor.org/info/rfc7742>>.
- [RFC7850] Nandakumar, S., "Registering Values of the SDP 'proto' Field for Transporting RTP Media over TCP under Various RTP Profiles", RFC 7850, DOI 10.17487/RFC7850, April 2016, <<https://www.rfc-editor.org/info/rfc7850>>.
- [RFC7874] Valin, JM. and C. Bran, "WebRTC Audio Codec and Processing Requirements", RFC 7874, DOI 10.17487/RFC7874, May 2016, <<https://www.rfc-editor.org/info/rfc7874>>.
- [RFC8108] Lennox, J., Westerlund, M., Wu, Q., and C. Perkins, "Sending Multiple RTP Streams in a Single RTP Session", RFC 8108, DOI 10.17487/RFC8108, March 2017, <<https://www.rfc-editor.org/info/rfc8108>>.
- [RFC8122] Lennox, J. and C. Holmberg, "Connection-Oriented Media Transport over the Transport Layer Security (TLS) Protocol in the Session Description Protocol (SDP)", RFC 8122, DOI 10.17487/RFC8122, March 2017, <<https://www.rfc-editor.org/info/rfc8122>>.
- [RFC8445] Keranen, A., Holmberg, C., and J. Rosenberg, "Interactive Connectivity Establishment (ICE): A Protocol for Network Address Translator (NAT) Traversal", RFC 8445, DOI 10.17487/RFC8445, July 2018, <<https://www.rfc-editor.org/info/rfc8445>>.

11.2. Informative References

- [I-D.ietf-mmusic-trickle-ice-sip]
Ivov, E., Stach, T., Marocco, E., and C. Holmberg, "A Session Initiation Protocol (SIP) Usage for Incremental Provisioning of Candidates for the Interactive Connectivity Establishment (Trickle ICE)", draft-ietf-mmusic-trickle-ice-sip-18 (work in progress), June 2018.

- [I-D.ietf-rtcweb-ip-handling]
Uberti, J., "WebRTC IP Address Handling Requirements",
draft-ietf-rtcweb-ip-handling-11 (work in progress),
November 2018.
- [I-D.ietf-rtcweb-sdp]
Nandakumar, S. and C. Jennings, "Annotated Example SDP for
WebRTC", draft-ietf-rtcweb-sdp-11 (work in progress),
October 2018.
- [RFC3389] Zopf, R., "Real-time Transport Protocol (RTP) Payload for
Comfort Noise (CN)", RFC 3389, DOI 10.17487/RFC3389,
September 2002, <<https://www.rfc-editor.org/info/rfc3389>>.
- [RFC3556] Casner, S., "Session Description Protocol (SDP) Bandwidth
Modifiers for RTP Control Protocol (RTCP) Bandwidth",
RFC 3556, DOI 10.17487/RFC3556, July 2003,
<<https://www.rfc-editor.org/info/rfc3556>>.
- [RFC3960] Camarillo, G. and H. Schulzrinne, "Early Media and Ringing
Tone Generation in the Session Initiation Protocol (SIP)",
RFC 3960, DOI 10.17487/RFC3960, December 2004,
<<https://www.rfc-editor.org/info/rfc3960>>.
- [RFC4568] Andreasen, F., Baugher, M., and D. Wing, "Session
Description Protocol (SDP) Security Descriptions for Media
Streams", RFC 4568, DOI 10.17487/RFC4568, July 2006,
<<https://www.rfc-editor.org/info/rfc4568>>.
- [RFC4588] Rey, J., Leon, D., Miyazaki, A., Varsa, V., and R.
Hakenberg, "RTP Retransmission Payload Format", RFC 4588,
DOI 10.17487/RFC4588, July 2006,
<<https://www.rfc-editor.org/info/rfc4588>>.
- [RFC4733] Schulzrinne, H. and T. Taylor, "RTP Payload for DTMF
Digits, Telephony Tones, and Telephony Signals", RFC 4733,
DOI 10.17487/RFC4733, December 2006,
<<https://www.rfc-editor.org/info/rfc4733>>.
- [RFC5245] Rosenberg, J., "Interactive Connectivity Establishment
(ICE): A Protocol for Network Address Translator (NAT)
Traversal for Offer/Answer Protocols", RFC 5245,
DOI 10.17487/RFC5245, April 2010,
<<https://www.rfc-editor.org/info/rfc5245>>.

- [RFC5506] Johansson, I. and M. Westerlund, "Support for Reduced-Size Real-Time Transport Control Protocol (RTCP): Opportunities and Consequences", RFC 5506, DOI 10.17487/RFC5506, April 2009, <<https://www.rfc-editor.org/info/rfc5506>>.
- [RFC5576] Lennox, J., Ott, J., and T. Schierl, "Source-Specific Media Attributes in the Session Description Protocol (SDP)", RFC 5576, DOI 10.17487/RFC5576, June 2009, <<https://www.rfc-editor.org/info/rfc5576>>.
- [RFC5763] Fischl, J., Tschofenig, H., and E. Rescorla, "Framework for Establishing a Secure Real-time Transport Protocol (SRTP) Security Context Using Datagram Transport Layer Security (DTLS)", RFC 5763, DOI 10.17487/RFC5763, May 2010, <<https://www.rfc-editor.org/info/rfc5763>>.
- [RFC5764] McGrew, D. and E. Rescorla, "Datagram Transport Layer Security (DTLS) Extension to Establish Keys for the Secure Real-time Transport Protocol (SRTP)", RFC 5764, DOI 10.17487/RFC5764, May 2010, <<https://www.rfc-editor.org/info/rfc5764>>.
- [RFC6464] Lennox, J., Ed., Ivov, E., and E. Marocco, "A Real-time Transport Protocol (RTP) Header Extension for Client-to-Mixer Audio Level Indication", RFC 6464, DOI 10.17487/RFC6464, December 2011, <<https://www.rfc-editor.org/info/rfc6464>>.
- [RFC6544] Rosenberg, J., Keranen, A., Lowekamp, B., and A. Roach, "TCP Candidates with Interactive Connectivity Establishment (ICE)", RFC 6544, DOI 10.17487/RFC6544, March 2012, <<https://www.rfc-editor.org/info/rfc6544>>.
- [TS26.114] 3GPP TS 26.114 V12.8.0, "3rd Generation Partnership Project; Technical Specification Group Services and System Aspects; IP Multimedia Subsystem (IMS); Multimedia Telephony; Media handling and interaction (Release 12)", December 2014, <<http://www.3gpp.org/DynaReport/26114.htm>>.
- [W3C.webrtc] Bergkvist, A., Burnett, D., Jennings, C., Narayanan, A., Aboba, B., and T. Brandstetter, "WebRTC 1.0: Real-time Communication Between Browsers", World Wide Web Consortium WD WD-webrtc-20170515, May 2017, <<https://www.w3.org/TR/2017/WD-webrtc-20170515/>>.

Appendix A. Appendix A

For the syntax validation performed in Section 5.8, the following list of ABNF definitions is used:

Attribute	Reference
ptime	[RFC4566] Section 9
maxptime	[RFC4566] Section 9
rtpmap	[RFC4566] Section 9
recvonly	[RFC4566] Section 9
sendrecv	[RFC4566] Section 9
sendonly	[RFC4566] Section 9
inactive	[RFC4566] Section 9
framerate	[RFC4566] Section 9
fntp	[RFC4566] Section 9
quality	[RFC4566] Section 9
rtcp	[RFC3605] Section 2.1
setup	[RFC4145] Sections 3, 4, and 5
connection	[RFC4145] Sections 3, 4, and 5
fingerprint	[RFC8122] Section 5
rtcp-fb	[RFC4585] Section 4.2
extmap	[RFC5285] Section 7
mid	[RFC5888] Sections 4 and 5
group	[RFC5888] Sections 4 and 5
imageattr	[RFC6236] Section 3.1
extmap (encrypt option)	[RFC6904] Section 4
candidate	[I-D.ietf-mmusic-ice-sip-sdp] Section 4.1
remote-candidates	[I-D.ietf-mmusic-ice-sip-sdp] Section 4.2
ice-lite	[I-D.ietf-mmusic-ice-sip-sdp] Section 4.3
ice-ufrag	[I-D.ietf-mmusic-ice-sip-sdp] Section 4.4
ice-pwd	[I-D.ietf-mmusic-ice-sip-sdp] Section 4.4
ice-options	[I-D.ietf-mmusic-ice-sip-sdp] Section 4.6
msid	[I-D.ietf-mmusic-msid] Section 2
rid	[I-D.ietf-mmusic-rid] Section 10
simulcast	[I-D.ietf-mmusic-sdp-simulcast] Section 6.1
tls-id	[I-D.ietf-mmusic-dtls-sdp] Section 4

Table 1: SDP ABNF References

Appendix B. Change log

Note to RFC Editor: Please remove this section before publication.

Changes in draft-26:

- o Update guidance on generation of the m= proto value to be consistent with ice-sip-sdp.

Changes in draft-25:

- o Remove MSID track ID from offers and answers.
- o Add note about rejecting all m= sections in a BUNDLE group.
- o Update ICE references to RFC 8445 and mention ice2.

Changes in draft-24:

- o Clarify that rounding is permitted when trying to maintain aspect ratio.
- o Update tls-id handling to match what is specified in dtls-sdp.

Changes in draft-23:

- o Clarify rollback handling, and treat it similarly to other setLocal/setRemote usages.
- o Adopt a first-fit policy for handling multiple remote a=imageattr attributes.
- o Clarify that a session description with zero m= sections is legal.

Changes in draft-22:

- o Clarify currentDirection versus direction.
- o Correct session-id text so that it aligns with RFC 3264.
- o Clarify that generated ICE candidate objects must have all four fields.
- o Make rollback work from any state besides stable and regardless of whether setLocalDescription or setRemoteDescription is used.
- o Allow modifying SDP before sending or after receiving either offers or answers (previously this was forbidden for answers).

- o Provide rationale for several design choices.

Changes in draft-21:

- o Change dtls-id to tls-id to match MMUSIC draft.
- o Replace regular expression for proto field with a list and clarify that the answer must exactly match the offer.
- o Remove text about how to error check on setLocal because local descriptions cannot be changed.
- o Rework silence suppression support to always require that both sides agree to silence suppression or none is used.
- o Remove instructions to parse "a=ssrc-group".
- o Allow the addition of new codecs in answers and in subsequent offers.
- o Clarify imageattr processing. Replace use of [x=0,y=0] with direction indicators.
- o Document when early media can occur.
- o Fix ICE default port handling when bundle-only is used.
- o Forbid duplicating IDENTICAL/TRANSPORT attributes when you are bundling.
- o Clarify the number of components to gather when bundle is involved.
- o Explicitly state that PTs and SSRCS are to be used for demuxing.
- o Update guidance on "a=setup" line. This should now match the MMUSIC draft.
- o Update guidance on certificate/digest matching to conform to RFC8122.
- o Update examples.

Changes in draft-20:

- o Remove Appendix-B.

Changes in draft-19:

- o Examples are now machine-generated for correctness, and use IETF-approved example IP addresses.
- o Add early transport warmup example, and add missing attributes to existing examples.
- o Only send "a=rtcp-mux-only" and "a=bundle-only" on new m= sections.
- o Update references.
- o Add coverage of a=identity.
- o Explain the lipsync group algorithm more thoroughly.
- o Remove unnecessary list of MTI specs.
- o Allow codecs which weren't offered to appear in answers and which weren't selected to appear in subsequent offers.
- o Codec preferences now are applied on both initial and subsequent offers and answers.
- o Clarify a=msid handling for recvonly m= sections.
- o Clarify behavior of attributes for bundle-only data channels.
- o Allow media attributes to appear in data m= sections when all the media m= sections are bundle-only.
- o Use consistent terminology for JSEP implementations.
- o Describe how to handle failed API calls.
- o Some cleanup on routing rules.

Changes in draft-18:

- o Update demux algorithm and move it to an appendix in preparation for merging it into BUNDLE.
- o Clarify why we can't handle an incoming offer to send simulcast.
- o Expand IceCandidate object text.
- o Further document use of ICE candidate pool.
- o Document removeTrack.

- o Update requirements to only accept the last generated offer/answer as an argument to setLocalDescription.
- o Allow round pixels.
- o Fix code around default timing when AVPF is not specified.
- o Clean up terminology around m= line and m=section.
- o Provide a more realistic example for minimum decoder capabilities.
- o Document behavior when rtcp-mux policy is require but rtcp-mux attribute not provided.
- o Expanded discussion of RtpSender and RtpReceiver.
- o Add RtpTransceiver.currentDirection and document setDirection.
- o Require imageattr x=0, y=0 to indicate that there are no valid resolutions.
- o Require a privacy-preserving MID/RID construction.
- o Require support for RFC 3556 bandwidth modifiers.
- o Update maxptime description.
- o Note that endpoints may encounter extra codecs in answers and subsequent offers from non-JSEP peers.
- o Update references.

Changes in draft-17:

- o Split createOffer and createAnswer sections to clearly indicate attributes which always appear and which only appear when not bundled into another m= section.
- o Add descriptions of RtpTransceiver methods.
- o Describe how to process RTCP feedback attributes.
- o Clarify transceiver directions and their interaction with 3264.
- o Describe setCodecPreferences.
- o Update RTP demux algorithm. Include RTCP.

- o Update requirements for when a=rtcp is included, limiting to cases where it is needed for backward compatibility.
- o Clarify SAR handling.
- o Updated addTrack matching algorithm.
- o Remove a=ssrc requirements.
- o Handle a=setup in reoffers.
- o Discuss how RTX/FEC should be handled.
- o Discuss how telephone-event should be handled.
- o Discuss how CN/DTX should be handled.
- o Add missing references to ABNF table.

Changes in draft-16:

- o Update addIceCandidate to indicate ICE generation and allow per-m= section end-of-candidates.
- o Update fingerprint handling to use draft-ietf-mmusic-4572-update.
- o Update text around SDP processing of RTP header extensions and payload formats.
- o Add sections on simulcast, addTransceiver, and createDataChannel.
- o Clarify text to ensure that the session ID is a positive 63 bit integer.
- o Clarify SDP processing for direction indication.
- o Describe SDP processing for rtcp-mux-only.
- o Specify how SDP session version in o= line.
- o Require that when doing an re-offer, the capabilities of the new session are mostly required to be a subset of the previously negotiated session.
- o Clarified ICE restart interaction with bundle-only.
- o Remove support for changing SDP before calling setLocalDescription.

- o Specify algorithm for demuxing RTP based on MID, PT, and SSRC.
- o Clarify rules for rejecting m= lines when bundle policy is balanced or max-bundle.

Changes in draft-15:

- o Clarify text around codecs offered in subsequent transactions to refer to what's been negotiated.
- o Rewrite LS handling text to indicate edge cases and that we're living with them.
- o Require that answerer reject m= lines when there are no codecs in common.
- o Enforce max-bundle on offer processing.
- o Fix TIAS formula to handle bits vs. kilobits.
- o Describe addTrack algorithm.
- o Clean up references.

Changes in draft-14:

- o Added discussion of RtpTransceivers + RtpSenders + RtpReceivers, and how they interact with createOffer/createAnswer.
- o Removed obsolete OfferToReceiveX options.
- o Explained how addIceCandidate can be used for end-of-candidates.

Changes in draft-13:

- o Clarified which SDP lines can be ignored.
- o Clarified how to handle various received attributes.
- o Revised how attributes should be generated for bundled m= lines.
- o Remove unused references.
- o Remove text advocating use of unilateral PTs.
- o Trigger an ICE restart even if the ICE candidate policy is being made more strict.

- o Remove the 'public' ICE candidate policy.
- o Move open issues into GitHub issues.
- o Split local/remote description accessors into current/pending.
- o Clarify a=imageattr handling.
- o Add more detail on VoiceActivityDetection handling.
- o Reference draft-shieh-rtcweb-ip-handling.
- o Make it clear when an ICE restart should occur.
- o Resolve changes needed in references.
- o Remove MSID semantics.
- o ice-options are now at session level.
- o Default RTCP mux policy is now 'require'.

Changes in draft-12:

- o Filled in sections on applying local and remote descriptions.
- o Discussed downscaling and upscaling to fulfill imageattr requirements.
- o Updated what SDP can be modified by the application.
- o Updated to latest datachannel SDP.
- o Allowed multiple fingerprint lines.
- o Switched back to IPv4 for dummy candidates.
- o Added additional clarity on ICE default candidates.

Changes in draft-11:

- o Clarified handling of RTP CNAMEs.
- o Updated what SDP lines should be processed or ignored.
- o Specified how a=imageattr should be used.

Changes in draft-10:

- o Described video size negotiation with imageattr.
- o Clarified rejection of sections that do not have mux-only.
- o Add handling of LS groups

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 St S
Kirkland, WA 98033
USA

Email: justin@uberti.name

Cullen Jennings
Cisco
400 3rd Avenue SW
Calgary, AB T2P 4H2
Canada

Email: fluffy@iii.ca

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

Email: ekr@rtfm.com

RTCWEB Working Group
Internet-Draft
Intended status: Standards Track
Expires: September 18, 2016

C. Perkins
University of Glasgow
M. Westerlund
Ericsson
J. Ott
Aalto University
March 17, 2016

Web Real-Time Communication (WebRTC): Media Transport and Use of RTP
draft-ietf-rtcweb-rtp-usage-26

Abstract

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

Status of This Memo

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

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

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

This Internet-Draft will expire on September 18, 2016.

Copyright Notice

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

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents

carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction	3
2. Rationale	4
3. Terminology	4
4. WebRTC Use of RTP: Core Protocols	5
4.1. RTP and RTCP	5
4.2. Choice of the RTP Profile	7
4.3. Choice of RTP Payload Formats	8
4.4. Use of RTP Sessions	10
4.5. RTP and RTCP Multiplexing	10
4.6. Reduced Size RTCP	11
4.7. Symmetric RTP/RTCP	11
4.8. Choice of RTP Synchronisation Source (SSRC)	12
4.9. Generation of the RTCP Canonical Name (CNAME)	12
4.10. Handling of Leap Seconds	13
5. WebRTC Use of RTP: Extensions	13
5.1. Conferencing Extensions and Topologies	14
5.1.1. Full Intra Request (FIR)	15
5.1.2. Picture Loss Indication (PLI)	15
5.1.3. Slice Loss Indication (SLI)	16
5.1.4. Reference Picture Selection Indication (RPSI)	16
5.1.5. Temporal-Spatial Trade-off Request (TSTR)	16
5.1.6. Temporary Maximum Media Stream Bit Rate Request (TMMBR)	16
5.2. Header Extensions	17
5.2.1. Rapid Synchronisation	17
5.2.2. Client-to-Mixer Audio Level	17
5.2.3. Mixer-to-Client Audio Level	18
5.2.4. Media Stream Identification	18
5.2.5. Coordination of Video Orientation	18
6. WebRTC Use of RTP: Improving Transport Robustness	19
6.1. Negative Acknowledgements and RTP Retransmission	19
6.2. Forward Error Correction (FEC)	20
7. WebRTC Use of RTP: Rate Control and Media Adaptation	20
7.1. Boundary Conditions and Circuit Breakers	21
7.2. Congestion Control Interoperability and Legacy Systems	22
8. WebRTC Use of RTP: Performance Monitoring	22
9. WebRTC Use of RTP: Future Extensions	23
10. Signalling Considerations	23
11. WebRTC API Considerations	25
12. RTP Implementation Considerations	27

12.1.	Configuration and Use of RTP Sessions	27
12.1.1.	Use of Multiple Media Sources Within an RTP Session	27
12.1.2.	Use of Multiple RTP Sessions	28
12.1.3.	Differentiated Treatment of RTP Streams	33
12.2.	Media Source, RTP Streams, and Participant Identification	35
12.2.1.	Media Source Identification	35
12.2.2.	SSRC Collision Detection	36
12.2.3.	Media Synchronisation Context	37
13.	Security Considerations	37
14.	IANA Considerations	39
15.	Acknowledgements	39
16.	References	39
16.1.	Normative References	39
16.2.	Informative References	44
	Authors' Addresses	46

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 of RTP development 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 [RFC7478].

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]. A MediaStream consists of zero or more MediaStreamTracks.

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

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 Stream, RTP Session, and Endpoint.

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 endpoints 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 mandates the usage of a single CNAME when sending RTP Streams in some circumstances, see Section 4.9.
- o Support for sending and receiving RTCP SR, RR, SDES, and BYE packet types. Note that support for other RTCP packet types is OPTIONAL, 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 endpoints 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]. 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 is 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 endpoint 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 endpoints 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.

Endpoints can signal support for multiple RTP payload formats, or multiple configurations of a single RTP payload format, as long as each unique RTP payload format configuration uses a different RTP payload type number. As outlined in Section 4.8, the RTP payload type number is sometimes used to associate an RTP 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 endpoint 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 endpoints communicating using RTP is known as an RTP session [RFC3550]. An endpoint 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 mechanism defined in [RFC5761]. Implementations can also support sending RTP and RTCP on separate transport-layer flows, but this is OPTIONAL to implement. If an implementation does not support RTP and RTCP sent on separate transport-layer flows, it MUST indicate that using the mechanism defined in [I-D.ietf-mmusic-mux-exclusive].

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 endpoint does not agree to the use of non-compound RTCP in the signalling exchange.

4.7. Symmetric RTP/RTCP

To ease traversal of NAT and firewall devices, implementations are REQUIRED to implement and use Symmetric RTP [RFC4961]. The 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 endpoints, 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 endpoint. While the Synchronisation Source (SSRC) identifier for an RTP endpoint can change if a collision is detected, or when the RTP application is restarted, its RTCP CNAME is meant to stay unchanged for the duration of a RTCPeerConnection [W3C.WD-webrtc-20130910], so that RTP endpoints can be uniquely identified and associated with their RTP packet streams within a set of related RTP sessions.

Each RTP endpoint 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 endpoint 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 endpoints 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 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 endpoint that does not implement these extensions will work correctly, but might offer poor performance. Support for the listed extensions will greatly improve the quality of experience and, to provide a reasonable baseline quality, some 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. There 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 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 endpoints 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. Endpoints 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]. If a standardized congestion control algorithm that satisfies these requirements is developed in the future, this memo will need to be updated to mandate its use.

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. The congestion control algorithm will have to be proprietary until a standardized congestion control algorithm is available. 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

All endpoints that wish to interwork with WebRTC MUST implement RTCP and provide congestion feedback via the defined RTCP reporting mechanisms.

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 endpoints 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 endpoint supports RTP/AVPF, this enables negotiation of important parameters for frequent reporting, such as the "trr-int" parameter, and the possibility that the endpoint supports some useful feedback format for congestion control purpose such as TMMBR [RFC5104]. Implementations that have to interwork with such endpoints 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, see [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 RTCP XR packets SHOULD be signalled; implementations MUST ignore RTCP XR packets that are unexpected or not understood.

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 have to 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 endpoints 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 endpoints have a common view of the RTCP bandwidth. A common view of the RTCP bandwidth among different endpoints is important, to prevent differences in RTCP packet timing and timeout intervals causing interoperability problems.

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` might need to be synchronized during play back.

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 endpoint 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 endpoint need to be sent using the same CNAME within one `RTCPeerConnection`. This is motivating the use of a single CNAME in Section 4.9.

The requirement on using the same CNAME for all SSRCs that originate from the same endpoint, does not require a middlebox that forwards traffic from multiple endpoints 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-origin 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 within a single origin.

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. Since the sending party needs to change the CNAME to the one it uses, this implies it 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 be 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 are clearly possible and implementation specific.

A WebRTC Endpoint MUST support receiving multiple `MediaStreamTracks`, where each of the 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 enable more efficient stream handling when endpoints relay/forward streams. It also ensures that endpoints can interoperate with certain types of multi-stream middleboxes or endpoints that are not WebRTC.

Javascript Session Establishment Protocol [I-D.ietf-rtcweb-jsep] specifies that 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]. The MSID 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 interoperability. 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 endpoints. In the following, some ways in which WebRTC Endpoints can configure and use RTP sessions are 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 source (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 endpoint 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 endpoint sending multiple RTP packet streams of the same media type into a single RTP session at the same time.

Associated Repair Data: An endpoint 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 endpoint 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 endpoint 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 endpoints. WebRTC is a group communication environment and endpoints 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 endpoint 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 endpoints 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.

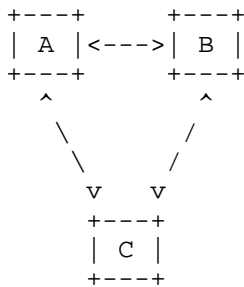


Figure 1: Multi-unicast using several RTP sessions

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 of the relationship between RTP sessions and `RTCPeerConnection` objects it is recommended that this is implemented as several individual RTP sessions. The only downside is that endpoint A will not learn of the quality of any transmission happening between B and C, since it will not see RTCP reports for the RTP session between B and C, whereas it would if all three participants were part of a single RTP session. Experience with the Mbone tools (experimental RTP-based multicast conferencing tools from the late 1990s) has showed that RTCP reception quality reports for third parties can 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.

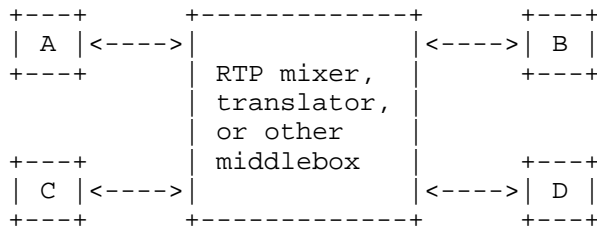


Figure 2: RTP mixer with only unicast paths

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 endpoints in one multi-party session. Other types of middlebox might use separate RTP sessions between each endpoint 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 endpoint, before sending it on towards the endpoint(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 endpoint experiences a situation in-between a session with only two endpoints 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 endpoints. 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 endpoint's

feedback or requests go 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 to 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, endpoints 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 endpoint. In RTP sessions where multiple endpoints are directly visible to an endpoint, all endpoints will have knowledge about each others' master keys, and can thus inject packets claimed to come from another endpoint 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 endpoints 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 endpoint that receives an RTP packet stream to be able to forward that RTP packet stream to a third party. There 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 endpoint 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

endpoint 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 endpoint 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 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 endpoint 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. Three common methods for classifying IP packets are:

DiffServ: The endpoint 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. The use of flow-based differentiation needs to be coordinated between the WebRTC system and the network(s). The WebRTC endpoint needs to know that flow-based differentiation might be used to provide the separation of the RTP packet streams onto different UDP flows to enable a more granular usage of flow based differentiation. The used flows, their 5-tuples and prioritization will need to be communicated to the network so that it can identify the flows correctly to enable prioritization. No specific protocol support for this is specified.

DiffServ assumes that either the endpoint or a classifier can mark the packets with an appropriate DSCP so that the packets are treated according to that marking. If the endpoint 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].

Deep Packet Inspectors will, despite the SRTP media encryption, still be fairly capable at classifying the RTP streams. The reason is that SRTP leaves the first 12 bytes of the RTP header unencrypted. This enables easy RTP stream identification using the SSRC and provides the classifier with useful information that can be correlated to determine for example the stream's media type. Using packet sizes, reception times, packet inter-spacing, RTP timestamp increments and sequence numbers, fairly reliable classifications are achieved.

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 lose. 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] and [I-D.ietf-dart-dscp-rtp] 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 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 endpoints 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:

- o In a point-to-point session where each SSRC is associated with either of the two endpoints 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 endpoints 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 endpoint resolved the SSRC collision.
- o 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 endpoint 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.
- o Multiple endpoints 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 endpoints from the RTP middlebox collisions can occur.
- o An RTP middlebox could connect an endpoint's RTCPeerConnection to another RTCPeerConnection from the same endpoint, thus forming a loop where the endpoint will receive its own traffic. While it is clearly considered a bug, it is important that the endpoint 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 endpoint 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 endpoint 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 and use media security solution is created by combining this secured RTP profile and DTLS-SRTP keying [RFC5764] as defined by Section 5.5 of [I-D.ietf-rtcweb-security-arch].

RTCP packets convey a Canonical Name (CNAME) identifier that is used to associate 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 mandates generation of short-term persistent RTCP CNAMEs, as specified in RFC7022, resulting in 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 performing voice activity based source selection 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.

In multi-party communication scenarios using RTP Middleboxes, a lot of trust is placed on these middleboxes to preserve the sessions security. The middlebox needs to maintain the confidentiality, integrity and perform source authentication. As discussed in Section 12.1.1 the middlebox can perform checks that prevents any endpoint participating in a conference to impersonate another. Some additional security considerations regarding multi-party topologies can be found in [I-D.ietf-avtcore-rtp-topologies-update].

14. IANA Considerations

This memo makes no request of IANA.

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

15. Acknowledgements

The authors would like to thank Bernard Aboba, Harald Alvestrand, Cary Bran, Ben Campbell, Alissa Cooper, Spencer Dawkins, Charles Eckel, Alex Eleftheriadis, Christian Groves, Chris Inacio, 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-13 (work in progress), December 2015.

- [I-D.ietf-avtc core-rtp-circuit-breakers]
Perkins, C. and V. Varun, "Multimedia Congestion Control: Circuit Breakers for Unicast RTP Sessions", draft-ietf-avtc core-rtp-circuit-breakers-13 (work in progress), February 2016.
- [I-D.ietf-avtc core-rtp-multi-stream]
Lennox, J., Westerlund, M., Wu, Q., and C. Perkins, "Sending Multiple RTP Streams in a Single RTP Session", draft-ietf-avtc core-rtp-multi-stream-11 (work in progress), December 2015.
- [I-D.ietf-avtc core-rtp-multi-stream-optimisation]
Lennox, J., Westerlund, M., Wu, Q., and C. Perkins, "Sending Multiple RTP Streams in a Single RTP Session: Grouping RTCP Reception Statistics and Other Feedback", draft-ietf-avtc core-rtp-multi-stream-optimisation-12 (work in progress), March 2016.
- [I-D.ietf-avtc core-rtp-topologies-update]
Westerlund, M. and S. Wenger, "RTP Topologies", draft-ietf-avtc core-rtp-topologies-update-10 (work in progress), July 2015.
- [I-D.ietf-mmusic-mux-exclusive]
Holmberg, C., "Indicating Exclusive Support of RTP/RTCP Multiplexing using SDP", draft-ietf-mmusic-mux-exclusive-03 (work in progress), February 2016.
- [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-27 (work in progress), February 2016.
- [I-D.ietf-rtcweb-audio]
Valin, J. and C. Bran, "WebRTC Audio Codec and Processing Requirements", draft-ietf-rtcweb-audio-10 (work in progress), February 2016.
- [I-D.ietf-rtcweb-fec]
Uberti, J., "WebRTC Forward Error Correction Requirements", draft-ietf-rtcweb-fec-02 (work in progress), October 2015.

- [I-D.ietf-rtcweb-overview]
Alvestrand, H., "Overview: Real Time Protocols for Browser-based Applications", draft-ietf-rtcweb-overview-15 (work in progress), January 2016.
- [I-D.ietf-rtcweb-security]
Rescorla, E., "Security Considerations for WebRTC", draft-ietf-rtcweb-security-08 (work in progress), February 2015.
- [I-D.ietf-rtcweb-security-arch]
Rescorla, E., "WebRTC Security Architecture", draft-ietf-rtcweb-security-arch-11 (work in progress), March 2015.
- [I-D.ietf-rtcweb-video]
Roach, A., "WebRTC Video Processing and Codec Requirements", draft-ietf-rtcweb-video-06 (work in progress), June 2015.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<http://www.rfc-editor.org/info/rfc2119>>.
- [RFC2736] Handley, M. and C. Perkins, "Guidelines for Writers of RTP Payload Format Specifications", BCP 36, RFC 2736, DOI 10.17487/RFC2736, December 1999, <<http://www.rfc-editor.org/info/rfc2736>>.
- [RFC3550] Schulzrinne, H., Casner, S., Frederick, R., and V. Jacobson, "RTP: A Transport Protocol for Real-Time Applications", STD 64, RFC 3550, DOI 10.17487/RFC3550, July 2003, <<http://www.rfc-editor.org/info/rfc3550>>.
- [RFC3551] Schulzrinne, H. and S. Casner, "RTP Profile for Audio and Video Conferences with Minimal Control", STD 65, RFC 3551, DOI 10.17487/RFC3551, July 2003, <<http://www.rfc-editor.org/info/rfc3551>>.
- [RFC3556] Casner, S., "Session Description Protocol (SDP) Bandwidth Modifiers for RTP Control Protocol (RTCP) Bandwidth", RFC 3556, DOI 10.17487/RFC3556, July 2003, <<http://www.rfc-editor.org/info/rfc3556>>.
- [RFC3711] Baugher, M., McGrew, D., Naslund, M., Carrara, E., and K. Norrman, "The Secure Real-time Transport Protocol (SRTP)", RFC 3711, DOI 10.17487/RFC3711, March 2004, <<http://www.rfc-editor.org/info/rfc3711>>.

- [RFC4566] Handley, M., Jacobson, V., and C. Perkins, "SDP: Session Description Protocol", RFC 4566, DOI 10.17487/RFC4566, July 2006, <<http://www.rfc-editor.org/info/rfc4566>>.
- [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, DOI 10.17487/RFC4585, July 2006, <<http://www.rfc-editor.org/info/rfc4585>>.
- [RFC4588] Rey, J., Leon, D., Miyazaki, A., Varsa, V., and R. Hakenberg, "RTP Retransmission Payload Format", RFC 4588, DOI 10.17487/RFC4588, July 2006, <<http://www.rfc-editor.org/info/rfc4588>>.
- [RFC4961] Wing, D., "Symmetric RTP / RTP Control Protocol (RTCP)", BCP 131, RFC 4961, DOI 10.17487/RFC4961, July 2007, <<http://www.rfc-editor.org/info/rfc4961>>.
- [RFC5104] Wenger, S., Chandra, U., Westerlund, M., and B. Burman, "Codec Control Messages in the RTP Audio-Visual Profile with Feedback (AVPF)", RFC 5104, DOI 10.17487/RFC5104, February 2008, <<http://www.rfc-editor.org/info/rfc5104>>.
- [RFC5124] Ott, J. and E. Carrara, "Extended Secure RTP Profile for Real-time Transport Control Protocol (RTCP)-Based Feedback (RTP/SAVPF)", RFC 5124, DOI 10.17487/RFC5124, February 2008, <<http://www.rfc-editor.org/info/rfc5124>>.
- [RFC5285] Singer, D. and H. Desineni, "A General Mechanism for RTP Header Extensions", RFC 5285, DOI 10.17487/RFC5285, July 2008, <<http://www.rfc-editor.org/info/rfc5285>>.
- [RFC5506] Johansson, I. and M. Westerlund, "Support for Reduced-Size Real-Time Transport Control Protocol (RTCP): Opportunities and Consequences", RFC 5506, DOI 10.17487/RFC5506, April 2009, <<http://www.rfc-editor.org/info/rfc5506>>.
- [RFC5761] Perkins, C. and M. Westerlund, "Multiplexing RTP Data and Control Packets on a Single Port", RFC 5761, DOI 10.17487/RFC5761, April 2010, <<http://www.rfc-editor.org/info/rfc5761>>.
- [RFC5764] McGrew, D. and E. Rescorla, "Datagram Transport Layer Security (DTLS) Extension to Establish Keys for the Secure Real-time Transport Protocol (SRTP)", RFC 5764, DOI 10.17487/RFC5764, May 2010, <<http://www.rfc-editor.org/info/rfc5764>>.

- [RFC6051] Perkins, C. and T. Schierl, "Rapid Synchronisation of RTP Flows", RFC 6051, DOI 10.17487/RFC6051, November 2010, <<http://www.rfc-editor.org/info/rfc6051>>.
- [RFC6464] Lennox, J., Ed., Iovov, E., and E. Marocco, "A Real-time Transport Protocol (RTP) Header Extension for Client-to-Mixer Audio Level Indication", RFC 6464, DOI 10.17487/RFC6464, December 2011, <<http://www.rfc-editor.org/info/rfc6464>>.
- [RFC6465] Iovov, E., Ed., Marocco, E., Ed., and J. Lennox, "A Real-time Transport Protocol (RTP) Header Extension for Mixer-to-Client Audio Level Indication", RFC 6465, DOI 10.17487/RFC6465, December 2011, <<http://www.rfc-editor.org/info/rfc6465>>.
- [RFC6562] Perkins, C. and JM. Valin, "Guidelines for the Use of Variable Bit Rate Audio with Secure RTP", RFC 6562, DOI 10.17487/RFC6562, March 2012, <<http://www.rfc-editor.org/info/rfc6562>>.
- [RFC6904] Lennox, J., "Encryption of Header Extensions in the Secure Real-time Transport Protocol (SRTP)", RFC 6904, DOI 10.17487/RFC6904, April 2013, <<http://www.rfc-editor.org/info/rfc6904>>.
- [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, DOI 10.17487/RFC7007, August 2013, <<http://www.rfc-editor.org/info/rfc7007>>.
- [RFC7022] Begen, A., Perkins, C., Wing, D., and E. Rescorla, "Guidelines for Choosing RTP Control Protocol (RTCP) Canonical Names (CNAMEs)", RFC 7022, DOI 10.17487/RFC7022, September 2013, <<http://www.rfc-editor.org/info/rfc7022>>.
- [RFC7160] Petit-Huguenin, M. and G. Zorn, Ed., "Support for Multiple Clock Rates in an RTP Session", RFC 7160, DOI 10.17487/RFC7160, April 2014, <<http://www.rfc-editor.org/info/rfc7160>>.
- [RFC7164] Gross, K. and R. Brandenburg, "RTP and Leap Seconds", RFC 7164, DOI 10.17487/RFC7164, March 2014, <<http://www.rfc-editor.org/info/rfc7164>>.

[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>>.

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-avtext-rtp-grouping-taxonomy]

Lennox, J., Gross, K., Nandakumar, S., Salgueiro, G., and B. Burman, "A Taxonomy of Semantics and Mechanisms for Real-Time Transport Protocol (RTP) Sources", draft-ietf-avtext-rtp-grouping-taxonomy-08 (work in progress), July 2015.

[I-D.ietf-dart-dscp-rtp]

Black, D. and P. Jones, "Differentiated Services (DiffServ) and Real-time Communication", draft-ietf-dart-dscp-rtp-10 (work in progress), November 2014.

[I-D.ietf-mmusic-msid]

Alvestrand, H., "WebRTC MediaStream Identification in the Session Description Protocol", draft-ietf-mmusic-msid-11 (work in progress), October 2015.

[I-D.ietf-payload-rtp-howto]

Westerlund, M., "How to Write an RTP Payload Format", draft-ietf-payload-rtp-howto-14 (work in progress), May 2015.

[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-jsep]
Uberti, J., Jennings, C., and E. Rescorla, "Javascript Session Establishment Protocol", draft-ietf-rtcweb-jsep-13 (work in progress), March 2016.
- [I-D.ietf-tsvwg-rtcweb-qos]
Jones, P., Dhesikan, S., Jennings, C., and D. Druta, "DSCP and other packet markings for WebRTC QoS", draft-ietf-tsvwg-rtcweb-qos-14 (work in progress), March 2016.
- [RFC3611] Friedman, T., Ed., Caceres, R., Ed., and A. Clark, Ed., "RTP Control Protocol Extended Reports (RTCP XR)", RFC 3611, DOI 10.17487/RFC3611, November 2003, <<http://www.rfc-editor.org/info/rfc3611>>.
- [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, DOI 10.17487/RFC4383, February 2006, <<http://www.rfc-editor.org/info/rfc4383>>.
- [RFC5245] Rosenberg, J., "Interactive Connectivity Establishment (ICE): A Protocol for Network Address Translator (NAT) Traversal for Offer/Answer Protocols", RFC 5245, DOI 10.17487/RFC5245, April 2010, <<http://www.rfc-editor.org/info/rfc5245>>.
- [RFC5576] Lennox, J., Ott, J., and T. Schierl, "Source-Specific Media Attributes in the Session Description Protocol (SDP)", RFC 5576, DOI 10.17487/RFC5576, June 2009, <<http://www.rfc-editor.org/info/rfc5576>>.
- [RFC5968] Ott, J. and C. Perkins, "Guidelines for Extending the RTP Control Protocol (RTCP)", RFC 5968, DOI 10.17487/RFC5968, September 2010, <<http://www.rfc-editor.org/info/rfc5968>>.
- [RFC6263] Marjou, X. and A. Sollaud, "Application Mechanism for Keeping Alive the NAT Mappings Associated with RTP / RTP Control Protocol (RTCP) Flows", RFC 6263, DOI 10.17487/RFC6263, June 2011, <<http://www.rfc-editor.org/info/rfc6263>>.
- [RFC6792] Wu, Q., Ed., Hunt, G., and P. Arden, "Guidelines for Use of the RTP Monitoring Framework", RFC 6792, DOI 10.17487/RFC6792, November 2012, <<http://www.rfc-editor.org/info/rfc6792>>.

[RFC7478] Holmberg, C., Hakanesson, S., and G. Eriksson, "Web Real-Time Communication Use Cases and Requirements", RFC 7478, DOI 10.17487/RFC7478, March 2015, <<http://www.rfc-editor.org/info/rfc7478>>.

Authors' Addresses

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

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

Magnus Westerlund
Ericsson
Farogatan 6
SE-164 80 Kista
Sweden

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

Joerg Ott
Aalto University
School of Electrical Engineering
Espoo 02150
Finland

Email: jorg.ott@aalto.fi

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

E. Rescorla
RTFM, Inc.
July 5, 2019

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

Abstract

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

Status of This Memo

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

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

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

This Internet-Draft will expire on January 6, 2020.

Copyright Notice

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

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

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

Table of Contents

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

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

1. Introduction

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

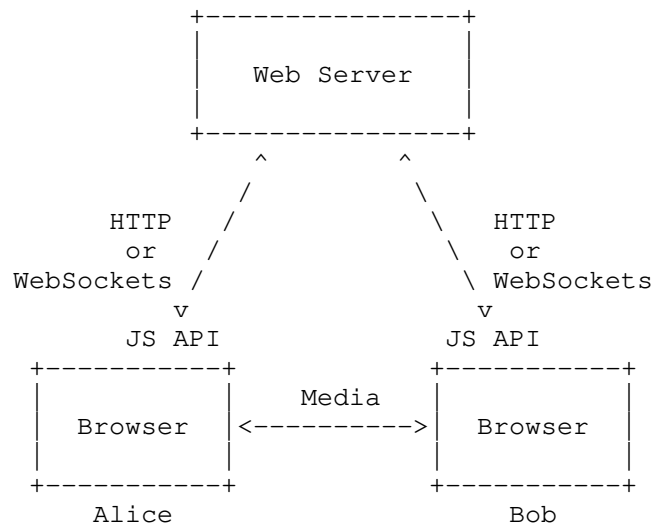


Figure 1: A simple WebRTC system

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

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

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

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

2. Terminology

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

3. The Browser Threat Model

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

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

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

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

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

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

3.1. Access to Local Resources

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

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

3.2. Same-Origin Policy

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

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

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

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

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

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

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

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

4. Security for WebRTC Applications

4.1. Access to Local Devices

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

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

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

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

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

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

4.1.1. Threats from Screen Sharing

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

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

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

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

4.1.2. Calling Scenarios and User Expectations

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

4.1.2.1. Dedicated Calling Services

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

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

4.1.2.2. Calling the Site You're On

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

4.1.3. Origin-Based Security

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

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

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

Individual Consent

Ask the user for permission for each call.

Callee-oriented Consent

Only allow calls to a given user.

Cryptographic Consent

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

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

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

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

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

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

4.1.4. Security Properties of the Calling Page

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

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

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

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

4.2. Communications Consent Verification

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

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

4.2.1. ICE

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

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

4.2.2. Masking

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

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

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

4.2.3. Backward Compatibility

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

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

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

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

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

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

4.2.4. IP Location Privacy

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

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

4.3. Communications Security

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

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

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

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

Retrospective compromise of calling service.

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

During-call attack by calling service.

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

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

4.3.1. Protecting Against Retrospective Compromise

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

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

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

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

4.3.2. Protecting Against During-Call Attack

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

4.3.2.1. Key Continuity

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

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

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

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

4.3.2.2. Short Authentication Strings

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

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

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

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

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

4.3.2.3. Third Party Identity

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

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

4.3.2.4. Page Access to Media

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

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

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

4.3.3. Malicious Peers

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

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

4.4. Privacy Considerations

4.4.1. Correlation of Anonymous Calls

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

4.4.2. Browser Fingerprinting

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

5. Security Considerations

This entire document is about security.

6. Acknowledgements

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

7. IANA Considerations

There are no IANA considerations.

8. Changes Since -04

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

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

9. References

9.1. Normative References

- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<https://www.rfc-editor.org/info/rfc2119>>.
- [RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <<https://www.rfc-editor.org/info/rfc8174>>.

9.2. Informative References

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

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

- [RFC3552] Rescorla, E. and B. Korver, "Guidelines for Writing RFC Text on Security Considerations", BCP 72, RFC 3552, DOI 10.17487/RFC3552, July 2003, <<https://www.rfc-editor.org/info/rfc3552>>.
- [RFC3711] Baugher, M., McGrew, D., Naslund, M., Carrara, E., and K. Norrman, "The Secure Real-time Transport Protocol (SRTP)", RFC 3711, DOI 10.17487/RFC3711, March 2004, <<https://www.rfc-editor.org/info/rfc3711>>.
- [RFC3760] Gustafson, D., Just, M., and M. Nystrom, "Securely Available Credentials (SACRED) - Credential Server Framework", RFC 3760, DOI 10.17487/RFC3760, April 2004, <<https://www.rfc-editor.org/info/rfc3760>>.
- [RFC4251] Ylonen, T. and C. Lonvick, Ed., "The Secure Shell (SSH) Protocol Architecture", RFC 4251, DOI 10.17487/RFC4251, January 2006, <<https://www.rfc-editor.org/info/rfc4251>>.
- [RFC4568] Andreassen, F., Baugher, M., and D. Wing, "Session Description Protocol (SDP) Security Descriptions for Media Streams", RFC 4568, DOI 10.17487/RFC4568, July 2006, <<https://www.rfc-editor.org/info/rfc4568>>.
- [RFC5479] Wing, D., Ed., Fries, S., Tschofenig, H., and F. Audet, "Requirements and Analysis of Media Security Management Protocols", RFC 5479, DOI 10.17487/RFC5479, April 2009, <<https://www.rfc-editor.org/info/rfc5479>>.
- [RFC5763] Fischl, J., Tschofenig, H., and E. Rescorla, "Framework for Establishing a Secure Real-time Transport Protocol (SRTP) Security Context Using Datagram Transport Layer Security (DTLS)", RFC 5763, DOI 10.17487/RFC5763, May 2010, <<https://www.rfc-editor.org/info/rfc5763>>.
- [RFC6189] Zimmermann, P., Johnston, A., Ed., and J. Callas, "ZRTP: Media Path Key Agreement for Unicast Secure RTP", RFC 6189, DOI 10.17487/RFC6189, April 2011, <<https://www.rfc-editor.org/info/rfc6189>>.
- [RFC6222] Begen, A., Perkins, C., and D. Wing, "Guidelines for Choosing RTP Control Protocol (RTCP) Canonical Names (CNAMEs)", RFC 6222, DOI 10.17487/RFC6222, April 2011, <<https://www.rfc-editor.org/info/rfc6222>>.
- [RFC6347] Rescorla, E. and N. Modadugu, "Datagram Transport Layer Security Version 1.2", RFC 6347, DOI 10.17487/RFC6347, January 2012, <<https://www.rfc-editor.org/info/rfc6347>>.

- [RFC6454] Barth, A., "The Web Origin Concept", RFC 6454, DOI 10.17487/RFC6454, December 2011, <<https://www.rfc-editor.org/info/rfc6454>>.
- [RFC6455] Fette, I. and A. Melnikov, "The WebSocket Protocol", RFC 6455, DOI 10.17487/RFC6455, December 2011, <<https://www.rfc-editor.org/info/rfc6455>>.
- [RFC6749] Hardt, D., Ed., "The OAuth 2.0 Authorization Framework", RFC 6749, DOI 10.17487/RFC6749, October 2012, <<https://www.rfc-editor.org/info/rfc6749>>.
- [RFC7022] Begen, A., Perkins, C., Wing, D., and E. Rescorla, "Guidelines for Choosing RTP Control Protocol (RTCP) Canonical Names (CNAMEs)", RFC 7022, DOI 10.17487/RFC7022, September 2013, <<https://www.rfc-editor.org/info/rfc7022>>.
- [RFC7033] Jones, P., Salgueiro, G., Jones, M., and J. Smarr, "WebFinger", RFC 7033, DOI 10.17487/RFC7033, September 2013, <<https://www.rfc-editor.org/info/rfc7033>>.
- [RFC7675] Perumal, M., Wing, D., Ravindranath, R., Reddy, T., and M. Thomson, "Session Traversal Utilities for NAT (STUN) Usage for Consent Freshness", RFC 7675, DOI 10.17487/RFC7675, October 2015, <<https://www.rfc-editor.org/info/rfc7675>>.
- [RFC8445] Keranen, A., Holmberg, C., and J. Rosenberg, "Interactive Connectivity Establishment (ICE): A Protocol for Network Address Translator (NAT) Traversal", RFC 8445, DOI 10.17487/RFC8445, July 2018, <<https://www.rfc-editor.org/info/rfc8445>>.
- [SWF] "SWF File Format Specification Version 19", April 2013.
- [whitten-johnny] Whitten, A. and J. Tygar, "Why Johnny Can't Encrypt: A Usability Evaluation of PGP 5.0", Proceedings of the 8th USENIX Security Symposium, 1999, August 1999.
- [XmlHttpRequest] van Kesteren, A., "XMLHttpRequest Level 2", January 2012.

Author's Address

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

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

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

E. Rescorla
RTFM, Inc.
July 21, 2019

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

Abstract

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

Status of This Memo

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

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

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

This Internet-Draft will expire on January 22, 2020.

Copyright Notice

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

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

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

Table of Contents

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

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

1. Introduction

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

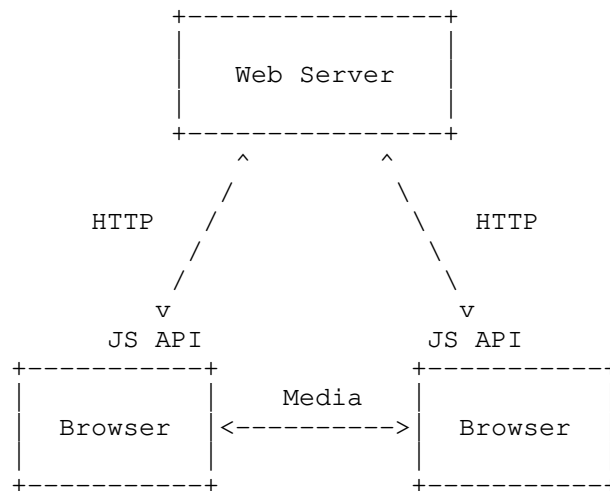


Figure 1: A simple WebRTC system

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

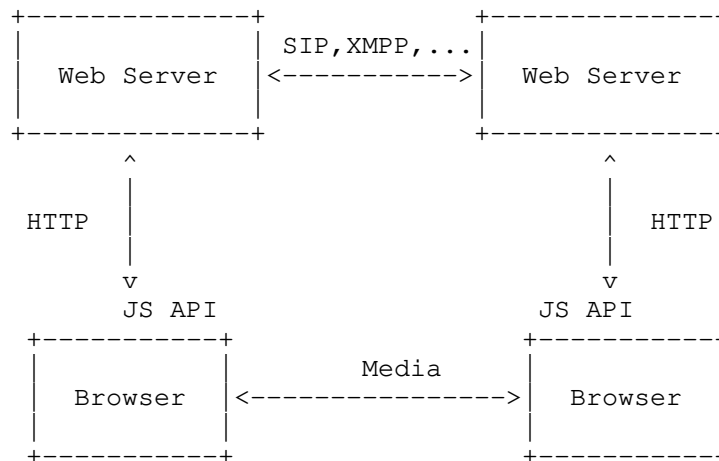


Figure 2: A multidomain WebRTC system

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

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

2. Terminology

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

3. Trust Model

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

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

3.1. Authenticated Entities

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

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

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

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

3.2. Unauthenticated Entities

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

4. Overview

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

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

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

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

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

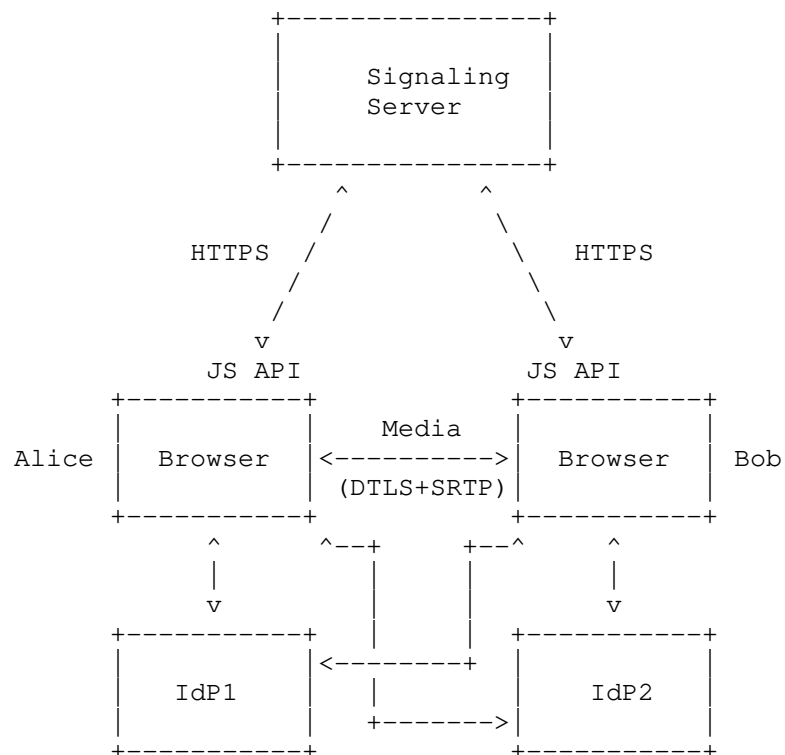


Figure 3: A call with IdP-based identity

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

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

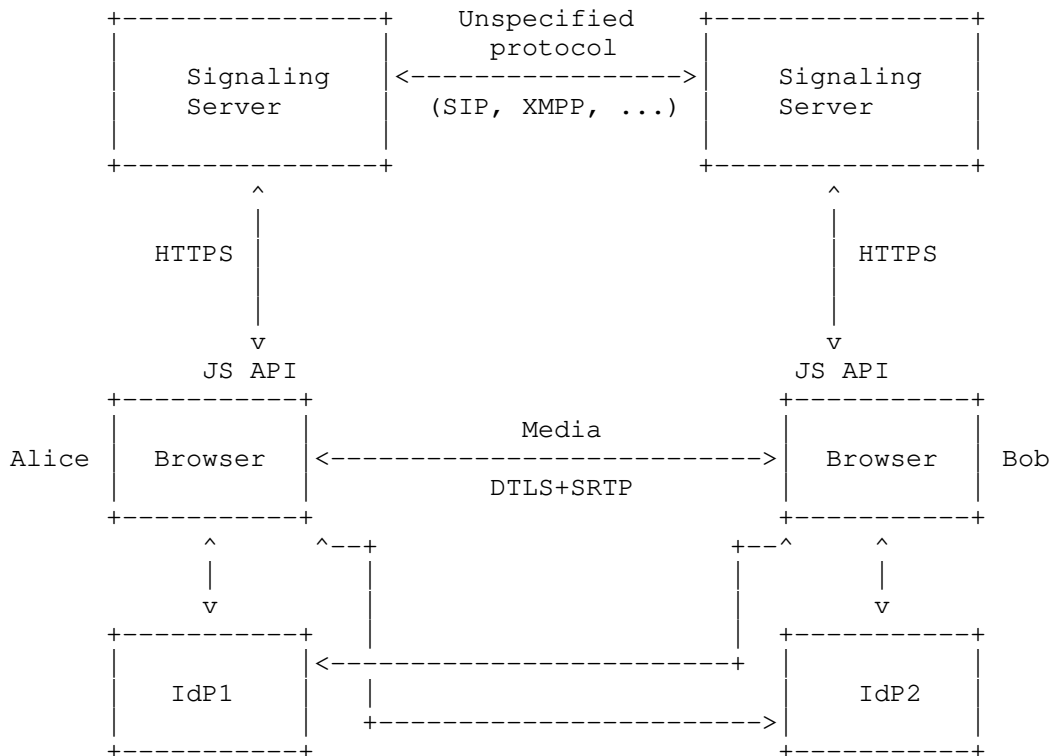


Figure 4: A federated call with IdP-based identity

4.1. Initial Signaling

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

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

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

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

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

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

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

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

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

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

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

4.2. Media Consent Verification

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

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

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

4.3. DTLS Handshake

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

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

4.4. Communications and Consent Freshness

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

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

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

5. SDP Identity Attribute

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

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

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

Name: identity

Value: identity-assertion

Usage Level: session

Charset Dependent: no

Default Value: N/A

Name: identity

Syntax:

```

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

```

<ALPHA and DIGIT as defined in [RFC4566]>

<base64 as defined in [RFC4566]>

Example:

```

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

```

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

This specification does not define any extensions for the attribute.

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

5.1. Offer/Answer Considerations

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

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

5.1.1. Generating the Initial SDP Offer

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

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

5.1.2. Generating of SDP Answer

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

5.1.3. Processing an SDP Offer or Answer

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

5.1.4. Modifying the Session

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

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

6. Detailed Technical Description

6.1. Origin and Web Security Issues

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

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

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

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

6.2. Device Permissions Model

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

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

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

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

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

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

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

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

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

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

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

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

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

6.3. Communications Consent

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

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

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

6.4. IP Location Privacy

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

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

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

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

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

6.5. Communications Security

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

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

All data channels MUST be secured via DTLS.

All Implementations MUST support DTLS 1.2 with the TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256 cipher suite and the P-256 curve [FIPS186]. Earlier drafts of this specification required DTLS 1.0 with the cipher suite TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA, and at the time of this writing some implementations do not support DTLS 1.2; endpoints which support only DTLS 1.2 might encounter interoperability issues. The DTLS-SRTP protection profile SRTP_AES128_CM_HMAC_SHA1_80 MUST be supported for SRTP. Implementations MUST favor cipher suites which support (Perfect Forward Secrecy) PFS over non-PFS cipher suites and SHOULD favor AEAD over non-AEAD cipher suites.

Implementations MUST NOT implement DTLS renegotiation and MUST reject it with a "no_renegotiation" alert if offered.

Endpoints MUST NOT implement TLS False Start [RFC7918].

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

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

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

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

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

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

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

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

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

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

7. Web-Based Peer Authentication

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

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

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

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

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

7.1. Trust Relationships: IdPs, APs, and RPs

Any federated identity protocol has three major participants:

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

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

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

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

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

At a high level there are two kinds of IdPs:

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

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

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

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

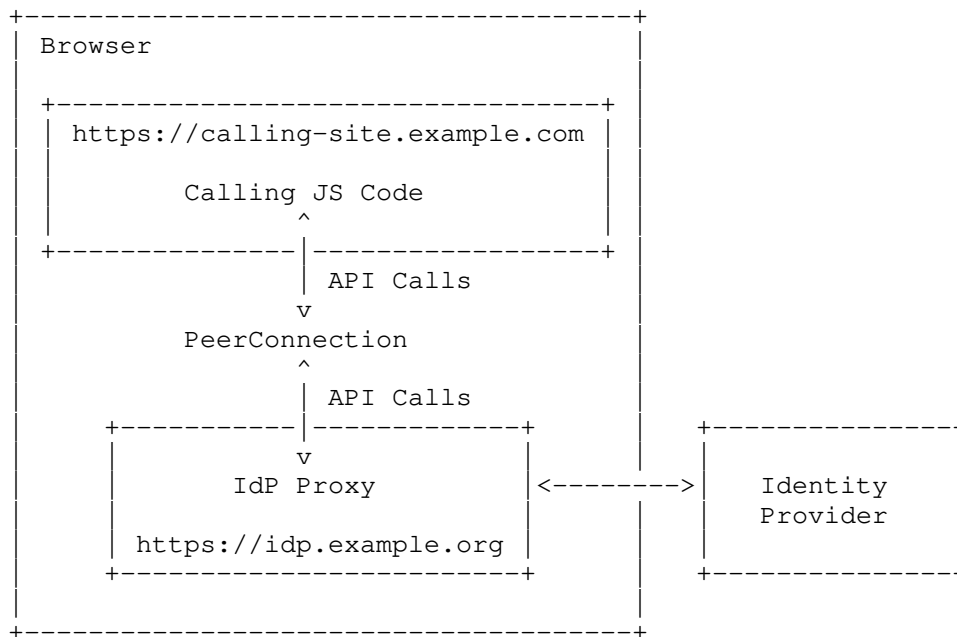
7.2. Overview of Operation

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

Note that there are two logically separate functions here:

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

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



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

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

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

7.3. Items for Standardization

There are two parts to this work:

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

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

7.4. Binding Identity Assertions to JSEP Offer/Answer Transactions

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

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

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

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

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

7.4.1. Carrying Identity Assertions

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

```

v=0
o=- 1181923068 1181923196 IN IP4 ual.example.com
s=example1
c=IN IP4 ual.example.com
a=fingerprint:sha-1 \
  4A:AD:B9:B1:3F:82:18:3B:54:02:12:DF:3E:5D:49:6B:19:E5:7C:AB
a=identity:\
  eyJpZHAiOnsizG9tYWluIjoizXhhbXBsZS5vcmdiLCJwcm90b2NvbCI6ImJvZ3Vz\
  In0sImFzc2VydgIvbiI6IntcImlkZW50aXR5XCI6XCJib2JAZXhhbXBsZS5vcmdc\
  IixcImNvbmlbnRzXCI6XCJhYmNkZWZnaGlqa2xtbm9wcXJzdHV2d3l6XCIsXCJz\
  aWduYXRlcmVcIjpcIjAxMDIwMzA0MDUwNlwiJ9
a=...
t=0 0
m=audio 6056 RTP/SAVP 0
a=sendrecv
...

```

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

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

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

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

7.5. Determining the IdP URI

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

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

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

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

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

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

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

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

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

7.5.1. Authenticating Party

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

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

7.5.2. Relying Party

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

7.6. Requesting Assertions

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

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

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

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

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

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

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

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

Figure 5: Example assertion

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

7.7. Managing User Login

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

The IdP proxy is able to access cookies, HTTP authentication or other persistent session data because it operates in the security context of the IdP origin. Therefore, if a user is logged in, the IdP could have all the information needed to generate an assertion.

An IdP proxy is unable to generate an assertion if the user is not logged in, or the IdP wants to interact with the user to acquire more information before generating the assertion. If the IdP wants to interact with the user before generating an assertion, the IdP proxy can fail to generate an assertion and instead indicate a URL where login should proceed.

The application can then load the provided URL to enable the user to enter credentials. The communication between the application and the IdP is described in [webrtc-api].

8. Verifying Assertions

The input to identity validation is the assertion string taken from a decoded 'identity' attribute.

The IdP proxy verifies the assertion. Depending on the identity protocol, the proxy might contact the IdP server or other servers. For instance, an OAuth-based protocol will likely require using the IdP as an oracle, whereas with a signature-based scheme might be able to verify the assertion without contacting the IdP, provided that it has cached the relevant public key.

Regardless of the mechanism, if verification succeeds, a successful response from the IdP proxy consists of the following information:

identity: The identity of the AP from the IdP's perspective.

Details of this are provided in Section 8.1.

contents: The original unmodified string provided by the AP as input to the assertion generation process.

Figure 6 shows an example response, which is JSON-formatted.

```
{
  "identity": "bob@example.org",
  "contents": "{\"fingerprint\":[ ... ]}"
}
```

Figure 6: Example verification result

8.1. Identity Formats

The identity provided from the IdP to the RP browser MUST consist of a string representing the user's identity. This string is in the form "<user>@<domain>", where "user" consists of any character, and domain is an internationalized domain name [RFC5890] encoded as a sequence of U-labels.

The PeerConnection API MUST check this string as follows:

1. If the "domain" portion of the string is equal to the domain name of the IdP proxy, then the assertion is valid, as the IdP is authoritative for this domain. Comparison of domain names is done using the label equivalence rule defined in Section 2.3.2.4 of [RFC5890].
2. If the "domain" portion of the string is not equal to the domain name of the IdP proxy, then the PeerConnection object MUST reject the assertion unless both:
 1. the IdP domain is trusted as an acceptable third-party IdP;
and

2. local policy is configured to trust this IdP domain for the domain portion of the identity string.

Any "@" or "%" characters in the "user" portion of the identity MUST be escaped according to the "Percent-Encoding" rules defined in Section 2.1 of [RFC3986]. Characters other than "@" and "%" MUST NOT be percent-encoded. For example, with a "user" of "user@133" and a "domain" of "identity.example.com", the resulting string will be encoded as "user%40133@identity.example.com".

Implementations are cautioned to take care when displaying user identities containing escaped "@" characters. If such characters are unescaped prior to display, implementations MUST distinguish between the domain of the IdP proxy and any domain that might be implied by the portion of the "<user>" portion that appears after the escaped "@" sign.

9. Security Considerations

Much of the security analysis of this problem is contained in [I-D.ietf-rtcweb-security] or in the discussion of the particular issues above. In order to avoid repetition, this section focuses on (a) residual threats that are not addressed by this document and (b) threats produced by failure/misbehavior of one of the components in the system.

9.1. Communications Security

IF HTTPS is not used to secure communications to the signaling server, and the identity mechanism used in Section 7 is not used, then any on-path attacker can replace the DTLS-SRTP fingerprints in the handshake and thus substitute its own identity for that of either endpoint.

Even if HTTPS is used, the signaling server can potentially mount a man-in-the-middle attack unless implementations have some mechanism for independently verifying keys. The UI requirements in Section 6.5 are designed to provide such a mechanism for motivated/security conscious users, but are not suitable for general use. The identity service mechanisms in Section 7 are more suitable for general use. Note, however, that a malicious signaling service can strip off any such identity assertions, though it cannot forge new ones. Note that all of the third-party security mechanisms available (whether X.509 certificates or a third-party IdP) rely on the security of the third party--this is of course also true of the user's connection to the Web site itself. Users who wish to assure themselves of security against a malicious identity provider can only do so by verifying

peer credentials directly, e.g., by checking the peer's fingerprint against a value delivered out of band.

In order to protect against malicious content JavaScript, that JavaScript MUST NOT be allowed to have direct access to---or perform computations with---DTLS keys. For instance, if content JS were able to compute digital signatures, then it would be possible for content JS to get an identity assertion for a browser's generated key and then use that assertion plus a signature by the key to authenticate a call protected under an ephemeral Diffie-Hellman (DH) key controlled by the content JS, thus violating the security guarantees otherwise provided by the IdP mechanism. Note that it is not sufficient merely to deny the content JS direct access to the keys, as some have suggested doing with the WebCrypto API [webcrypto]. The JS must also not be allowed to perform operations that would be valid for a DTLS endpoint. By far the safest approach is simply to deny the ability to perform any operations that depend on secret information associated with the key. Operations that depend on public information, such as exporting the public key are of course safe.

9.2. Privacy

The requirements in this document are intended to allow:

- o Users to participate in calls without revealing their location.
- o Potential callees to avoid revealing their location and even presence status prior to agreeing to answer a call.

However, these privacy protections come at a performance cost in terms of using TURN relays and, in the latter case, delaying ICE. Sites SHOULD make users aware of these tradeoffs.

Note that the protections provided here assume a non-malicious calling service. As the calling service always knows the users status and (absent the use of a technology like Tor) their IP address, they can violate the users privacy at will. Users who wish privacy against the calling sites they are using must use separate privacy enhancing technologies such as Tor. Combined WebRTC/Tor implementations SHOULD arrange to route the media as well as the signaling through Tor. Currently this will produce very suboptimal performance.

Additionally, any identifier which persists across multiple calls is potentially a problem for privacy, especially for anonymous calling services. Such services SHOULD instruct the browser to use separate DTLS keys for each call and also to use TURN throughout the call. Otherwise, the other side will learn linkable information that would

allow them to correlate the browser across multiple calls. Additionally, browsers SHOULD implement the privacy-preserving CNAME generation mode of [RFC7022].

9.3. Denial of Service

The consent mechanisms described in this document are intended to mitigate denial of service attacks in which an attacker uses clients to send large amounts of traffic to a victim without the consent of the victim. While these mechanisms are sufficient to protect victims who have not implemented WebRTC at all, WebRTC implementations need to be more careful.

Consider the case of a call center which accepts calls via WebRTC. An attacker proxies the call center's front-end and arranges for multiple clients to initiate calls to the call center. Note that this requires user consent in many cases but because the data channel does not need consent, he can use that directly. Since ICE will complete, browsers can then be induced to send large amounts of data to the victim call center if it supports the data channel at all. Preventing this attack requires that automated WebRTC implementations implement sensible flow control and have the ability to triage out (i.e., stop responding to ICE probes on) calls which are behaving badly, and especially to be prepared to remotely throttle the data channel in the absence of plausible audio and video (which the attacker cannot control).

Another related attack is for the signaling service to swap the ICE candidates for the audio and video streams, thus forcing a browser to send video to the sink that the other victim expects will contain audio (perhaps it is only expecting audio!) potentially causing overload. Muxing multiple media flows over a single transport makes it harder to individually suppress a single flow by denying ICE keepalives. Either media-level (RTCP) mechanisms must be used or the implementation must deny responses entirely, thus terminating the call.

Yet another attack, suggested by Magnus Westerlund, is for the attacker to cross-connect offers and answers as follows. It induces the victim to make a call and then uses its control of other users' browsers to get them to attempt a call to someone. It then translates their offers into apparent answers to the victim, which looks like large-scale parallel forking. The victim still responds to ICE responses and now the browsers all try to send media to the victim. Implementations can defend themselves from this attack by only responding to ICE Binding Requests for a limited number of remote ufrags (this is the reason for the requirement that the JS not be able to control the ufrag and password).

[I-D.ietf-rtcweb-rtp-usage] Section 13 documents a number of potential RTCP-based DoS attacks and countermeasures.

Note that attacks based on confusing one end or the other about consent are possible even in the face of the third-party identity mechanism as long as major parts of the signaling messages are not signed. On the other hand, signing the entire message severely restricts the capabilities of the calling application, so there are difficult tradeoffs here.

9.4. IdP Authentication Mechanism

This mechanism relies for its security on the IdP and on the PeerConnection correctly enforcing the security invariants described above. At a high level, the IdP is attesting that the user identified in the assertion wishes to be associated with the assertion. Thus, it must not be possible for arbitrary third parties to get assertions tied to a user or to produce assertions that RPs will accept.

9.4.1. PeerConnection Origin Check

Fundamentally, the IdP proxy is just a piece of HTML and JS loaded by the browser, so nothing stops a Web attacker from creating their own IFRAME, loading the IdP proxy HTML/JS, and requesting a signature over his own keys rather than those generated in the browser. However, that proxy would be in the attacker's origin, not the IdP's origin. Only the browser itself can instantiate a context that (a) is in the IdP's origin and (b) exposes the correct API surface. Thus, the IdP proxy on the sender's side MUST ensure that it is running in the IdP's origin prior to issuing assertions.

Note that this check only asserts that the browser (or some other entity with access to the user's authentication data) attests to the request and hence to the fingerprint. It does not demonstrate that the browser has access to the associated private key, and therefore an attacker can attach their own identity to another party's keying material, thus making a call which comes from Alice appear to come from the attacker. See [I-D.ietf-mmusic-sdp-uks] for defenses against this form of attack.

9.4.2. IdP Well-known URI

As described in Section 7.5 the IdP proxy HTML/JS landing page is located at a well-known URI based on the IdP's domain name. This requirement prevents an attacker who can write some resources at the IdP (e.g., on one's Facebook wall) from being able to impersonate the IdP.

9.4.3. Privacy of IdP-generated identities and the hosting site

Depending on the structure of the IdP's assertions, the calling site may learn the user's identity from the perspective of the IdP. In many cases this is not an issue because the user is authenticating to the site via the IdP in any case, for instance when the user has logged in with Facebook Connect and is then authenticating their call with a Facebook identity. However, in other case, the user may not have already revealed their identity to the site. In general, IdPs SHOULD either verify that the user is willing to have their identity revealed to the site (e.g., through the usual IdP permissions dialog) or arrange that the identity information is only available to known RPs (e.g., social graph adjacencies) but not to the calling site. The "domain" field of the assertion request can be used to check that the user has agreed to disclose their identity to the calling site; because it is supplied by the PeerConnection it can be trusted to be correct.

9.4.4. Security of Third-Party IdPs

As discussed above, each third-party IdP represents a new universal trust point and therefore the number of these IdPs needs to be quite limited. Most IdPs, even those which issue unqualified identities such as Facebook, can be recast as authoritative IdPs (e.g., 123456@facebook.com). However, in such cases, the user interface implications are not entirely desirable. One intermediate approach is to have special (potentially user configurable) UI for large authoritative IdPs, thus allowing the user to instantly grasp that the call is being authenticated by Facebook, Google, etc.

9.4.4.1. Confusable Characters

Because a broad range of characters are permitted in identity strings, it may be possible for attackers to craft identities which are confusable with other identities (see [RFC6943] for more on this topic). This is a problem with any identifier space of this type (e.g., e-mail addresses). Those minting identifiers should avoid mixed scripts and similar confusable characters. Those presenting these identifiers to a user should consider highlighting cases of mixed script usage (see [RFC5890], section 4.4). Other best practices are still in development.

9.4.5. Web Security Feature Interactions

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

9.4.5.1. Popup Blocking

When popup blocking is in use, the IdP proxy is unable to generate popup windows, dialogs or any other form of user interactions. This prevents the IdP proxy from being used to circumvent user interaction. The "LOGINNEEDED" message allows the IdP proxy to inform the calling site of a need for user login, providing the information necessary to satisfy this requirement without resorting to direct user interaction from the IdP proxy itself.

9.4.5.2. Third Party Cookies

Some browsers allow users to block third party cookies (cookies associated with origins other than the top level page) for privacy reasons. Any IdP which uses cookies to persist logins will be broken by third-party cookie blocking. One option is to accept this as a limitation; another is to have the PeerConnection object disable third-party cookie blocking for the IdP proxy.

10. IANA Considerations

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

Contact Name: IESG (iesg@ietf.org)

Attribute Name: identity

Long Form: identity

Type of Attribute: session-level

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

Purpose: This attribute carries an identity assertion, binding an identity to the transport-level security session.

Appropriate Values: See Section 5 of RFCXXXX [[Editor Note: This document.]]

Mux Category: NORMAL.

This section registers the "idp-proxy" well-known URI from [RFC5785].

URI suffix: idp-proxy

Change controller: IETF

11. Acknowledgements

Bernard Aboba, Harald Alvestrand, Richard Barnes, Dan Druta, Cullen Jennings, Hadriel Kaplan, Matthew Kaufman, Jim McEachern, Martin Thomson, Magnus Westerland. Matthew Kaufman provided the UI material in Section 6.5. Christer Holmberg provided the initial version of Section 5.1.

12. Changes

[RFC Editor: Please remove this section prior to publication.]

12.1. Changes since -15

Rewrite the Identity section in more conventional offer/answer format.

Clarify rules on changing identities.

12.2. Changes since -11

Update discussion of IdP security model

Replace "domain name" with RFC 3986 Authority

Clean up discussion of how to generate IdP URI.

Remove obsolete text about null cipher suites.

Remove obsolete appendixes about older IdP systems

Require support for ECDSA, PFS, and AEAD

12.3. Changes since -10

Update cipher suite profiles.

Rework IdP interaction based on implementation experience in Firefox.

12.4. Changes since -06

Replaced RTCWEB and RTC-Web with WebRTC, except when referring to the IETF WG

Forbade use in mixed content as discussed in Orlando.

Added a requirement to surface NULL ciphers to the top-level.

Tried to clarify SRTP versus DTLS-SRTP.

Added a section on screen sharing permissions.

Assorted editorial work.

12.5. Changes since -05

The following changes have been made since the -05 draft.

- o Response to comments from Richard Barnes
- o More explanation of the IdP security properties and the federation use case.
- o Editorial cleanup.

12.6. Changes since -03

Version -04 was a version control mistake. Please ignore.

The following changes have been made since the -04 draft.

- o Move origin check from IdP to RP per discussion in YVR.
- o Clarified treatment of X.509-level identities.
- o Editorial cleanup.

12.7. Changes since -03

12.8. Changes since -02

The following changes have been made since the -02 draft.

- o Forbid persistent HTTP permissions.
- o Clarified the text in S 5.4 to clearly refer to requirements on the API to provide functionality to the site.
- o Fold in the IETF portion of draft-rescorla-rtcweb-generic-idp
- o Retarget the continuing consent section to assume Binding Requests
- o Added some more privacy and linkage text in various places.

- o Editorial improvements

13. References

13.1. Normative References

- [FIPS186] National Institute of Standards and Technology (NIST), "Digital Signature Standard (DSS)", NIST PUB 186-4 , July 2013.
- [I-D.ietf-mmusic-sdp-uks]
Thomson, M. and E. Rescorla, "Unknown Key Share Attacks on uses of TLS with the Session Description Protocol (SDP)", draft-ietf-mmusic-sdp-uks-06 (work in progress), July 2019.
- [I-D.ietf-rtcweb-jsep]
Uberti, J., Jennings, C., and E. Rescorla, "JavaScript Session Establishment Protocol", draft-ietf-rtcweb-jsep-26 (work in progress), February 2019.
- [I-D.ietf-rtcweb-overview]
Alvestrand, H., "Overview: Real Time Protocols for Browser-based Applications", draft-ietf-rtcweb-overview-19 (work in progress), November 2017.
- [I-D.ietf-rtcweb-rtp-usage]
Perkins, C., Westerlund, M., and J. Ott, "Web Real-Time Communication (WebRTC): Media Transport and Use of RTP", draft-ietf-rtcweb-rtp-usage-26 (work in progress), March 2016.
- [I-D.ietf-rtcweb-security]
Rescorla, E., "Security Considerations for WebRTC", draft-ietf-rtcweb-security-12 (work in progress), July 2019.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<https://www.rfc-editor.org/info/rfc2119>>.
- [RFC2818] Rescorla, E., "HTTP Over TLS", RFC 2818, DOI 10.17487/RFC2818, May 2000, <<https://www.rfc-editor.org/info/rfc2818>>.

- [RFC3264] Rosenberg, J. and H. Schulzrinne, "An Offer/Answer Model with Session Description Protocol (SDP)", RFC 3264, DOI 10.17487/RFC3264, June 2002, <<https://www.rfc-editor.org/info/rfc3264>>.
- [RFC3711] Baugher, M., McGrew, D., Naslund, M., Carrara, E., and K. Norrman, "The Secure Real-time Transport Protocol (SRTP)", RFC 3711, DOI 10.17487/RFC3711, March 2004, <<https://www.rfc-editor.org/info/rfc3711>>.
- [RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, DOI 10.17487/RFC3986, January 2005, <<https://www.rfc-editor.org/info/rfc3986>>.
- [RFC4566] Handley, M., Jacobson, V., and C. Perkins, "SDP: Session Description Protocol", RFC 4566, DOI 10.17487/RFC4566, July 2006, <<https://www.rfc-editor.org/info/rfc4566>>.
- [RFC4568] Andreassen, F., Baugher, M., and D. Wing, "Session Description Protocol (SDP) Security Descriptions for Media Streams", RFC 4568, DOI 10.17487/RFC4568, July 2006, <<https://www.rfc-editor.org/info/rfc4568>>.
- [RFC4648] Josefsson, S., "The Base16, Base32, and Base64 Data Encodings", RFC 4648, DOI 10.17487/RFC4648, October 2006, <<https://www.rfc-editor.org/info/rfc4648>>.
- [RFC5246] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", RFC 5246, DOI 10.17487/RFC5246, August 2008, <<https://www.rfc-editor.org/info/rfc5246>>.
- [RFC5763] Fischl, J., Tschofenig, H., and E. Rescorla, "Framework for Establishing a Secure Real-time Transport Protocol (SRTP) Security Context Using Datagram Transport Layer Security (DTLS)", RFC 5763, DOI 10.17487/RFC5763, May 2010, <<https://www.rfc-editor.org/info/rfc5763>>.
- [RFC5764] McGrew, D. and E. Rescorla, "Datagram Transport Layer Security (DTLS) Extension to Establish Keys for the Secure Real-time Transport Protocol (SRTP)", RFC 5764, DOI 10.17487/RFC5764, May 2010, <<https://www.rfc-editor.org/info/rfc5764>>.

- [RFC5785] Nottingham, M. and E. Hammer-Lahav, "Defining Well-Known Uniform Resource Identifiers (URIs)", RFC 5785, DOI 10.17487/RFC5785, April 2010, <<https://www.rfc-editor.org/info/rfc5785>>.
- [RFC5890] Klensin, J., "Internationalized Domain Names for Applications (IDNA): Definitions and Document Framework", RFC 5890, DOI 10.17487/RFC5890, August 2010, <<https://www.rfc-editor.org/info/rfc5890>>.
- [RFC6347] Rescorla, E. and N. Modadugu, "Datagram Transport Layer Security Version 1.2", RFC 6347, DOI 10.17487/RFC6347, January 2012, <<https://www.rfc-editor.org/info/rfc6347>>.
- [RFC6454] Barth, A., "The Web Origin Concept", RFC 6454, DOI 10.17487/RFC6454, December 2011, <<https://www.rfc-editor.org/info/rfc6454>>.
- [RFC7022] Begen, A., Perkins, C., Wing, D., and E. Rescorla, "Guidelines for Choosing RTP Control Protocol (RTCP) Canonical Names (CNAMEs)", RFC 7022, DOI 10.17487/RFC7022, September 2013, <<https://www.rfc-editor.org/info/rfc7022>>.
- [RFC7675] Perumal, M., Wing, D., Ravindranath, R., Reddy, T., and M. Thomson, "Session Traversal Utilities for NAT (STUN) Usage for Consent Freshness", RFC 7675, DOI 10.17487/RFC7675, October 2015, <<https://www.rfc-editor.org/info/rfc7675>>.
- [RFC7918] Langley, A., Modadugu, N., and B. Moeller, "Transport Layer Security (TLS) False Start", RFC 7918, DOI 10.17487/RFC7918, August 2016, <<https://www.rfc-editor.org/info/rfc7918>>.
- [RFC8122] Lennox, J. and C. Holmberg, "Connection-Oriented Media Transport over the Transport Layer Security (TLS) Protocol in the Session Description Protocol (SDP)", RFC 8122, DOI 10.17487/RFC8122, March 2017, <<https://www.rfc-editor.org/info/rfc8122>>.
- [RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <<https://www.rfc-editor.org/info/rfc8174>>.
- [RFC8259] Bray, T., Ed., "The JavaScript Object Notation (JSON) Data Interchange Format", STD 90, RFC 8259, DOI 10.17487/RFC8259, December 2017, <<https://www.rfc-editor.org/info/rfc8259>>.

- [RFC8261] Tuexen, M., Stewart, R., Jesup, R., and S. Loreto, "Datagram Transport Layer Security (DTLS) Encapsulation of SCTP Packets", RFC 8261, DOI 10.17487/RFC8261, November 2017, <<https://www.rfc-editor.org/info/rfc8261>>.
- [RFC8445] Keranen, A., Holmberg, C., and J. Rosenberg, "Interactive Connectivity Establishment (ICE): A Protocol for Network Address Translator (NAT) Traversal", RFC 8445, DOI 10.17487/RFC8445, July 2018, <<https://www.rfc-editor.org/info/rfc8445>>.
- [webcrypto] editors, W., "Web Cryptography API", June 2013.
Available at <http://www.w3.org/TR/WebCryptoAPI/>
- [webrtc-api] editors, W., "WebRTC 1.0: Real-time Communication Between Browsers", October 2011.
Available at <http://dev.w3.org/2011/webrtc/editor/webrtc.html>

13.2. Informative References

- [RFC3261] Rosenberg, J., Schulzrinne, H., Camarillo, G., Johnston, A., Peterson, J., Sparks, R., Handley, M., and E. Schooler, "SIP: Session Initiation Protocol", RFC 3261, DOI 10.17487/RFC3261, June 2002, <<https://www.rfc-editor.org/info/rfc3261>>.
- [RFC5705] Rescorla, E., "Keying Material Exporters for Transport Layer Security (TLS)", RFC 5705, DOI 10.17487/RFC5705, March 2010, <<https://www.rfc-editor.org/info/rfc5705>>.
- [RFC6120] Saint-Andre, P., "Extensible Messaging and Presence Protocol (XMPP): Core", RFC 6120, DOI 10.17487/RFC6120, March 2011, <<https://www.rfc-editor.org/info/rfc6120>>.
- [RFC6265] Barth, A., "HTTP State Management Mechanism", RFC 6265, DOI 10.17487/RFC6265, April 2011, <<https://www.rfc-editor.org/info/rfc6265>>.
- [RFC6455] Fette, I. and A. Melnikov, "The WebSocket Protocol", RFC 6455, DOI 10.17487/RFC6455, December 2011, <<https://www.rfc-editor.org/info/rfc6455>>.

[RFC6943] Thaler, D., Ed., "Issues in Identifier Comparison for Security Purposes", RFC 6943, DOI 10.17487/RFC6943, May 2013, <<https://www.rfc-editor.org/info/rfc6943>>.

[RFC7617] Reschke, J., "The 'Basic' HTTP Authentication Scheme", RFC 7617, DOI 10.17487/RFC7617, September 2015, <<https://www.rfc-editor.org/info/rfc7617>>.

[XmlHttpRequest]
van Kesteren, A., "XMLHttpRequest Level 2", January 2012.

Author's Address

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

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

RTCWEB
Internet-Draft
Intended status: Standards Track
Expires: February 14, 2016

M. Perumal
Ericsson
D. Wing
R. Ravindranath
T. Reddy
Cisco Systems
M. Thomson
Mozilla
August 13, 2015

STUN Usage for Consent Freshness
draft-ietf-rtcweb-stun-consent-freshness-16

Abstract

To prevent WebRTC applications, such as browsers, from launching attacks by sending traffic to unwilling victims, periodic consent to send needs to be obtained from remote endpoints.

This document describes a consent mechanism using a new Session Traversal Utilities for NAT (STUN) usage.

Status of This Memo

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

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

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

This Internet-Draft will expire on February 14, 2016.

Copyright Notice

Copyright (c) 2015 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of

publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction	2
2. Applicability	3
3. Terminology	3
4. Design Considerations	3
5. Solution	4
5.1. Expiration of Consent	4
5.2. Immediate Revocation of Consent	6
6. DiffServ Treatment for Consent	7
7. DTLS applicability	7
8. Security Considerations	7
9. IANA Considerations	7
10. Acknowledgement	7
11. References	8
11.1. Normative References	8
11.2. Informative References	8
Authors' Addresses	9

1. Introduction

To prevent attacks on peers, endpoints have to ensure the remote peer is willing to receive traffic. Verification of peer consent before sending traffic is necessary in deployments like WebRTC to ensure that a malicious JavaScript cannot use the browser as a platform for launching attacks. 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. The consent mechanism does not update the ICE procedures defined in [RFC5245].

Consent is obtained only by full ICE implementations. An ICE-lite agent (as defined in Section 2.7 of [RFC5245]) does not generate connectivity checks or run the ICE state machine. Hence, an ICE-lite agent does not generate consent checks and will only respond to any checks that 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. Applicability

This document defines what it takes to obtain, maintain, and lose consent to send using ICE. Section 4.4 and Section 5.3 of [I-D.ietf-rtcweb-security-arch] further explains the value of obtaining and maintaining consent.

Other Applications that have similar security requirements to verify peer consent before sending non-ICE packets can use the consent mechanism described in this document. The mechanism of how applications are made aware of consent expiration is outside the scope of the document.

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].

Consent: The mechanism of obtaining permission from the remote endpoint to send non-ICE traffic to a remote transport address. Consent is obtained using ICE. Note that this is an application-level consent; no human intervention is involved.

Consent Freshness: Maintaining and renewing consent over time.

Transport Address: The remote peer's IP address and UDP or TCP port number.

4. 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]. STUN binding requests sent for consent freshness also serve the keepalive purpose (i.e. to keep NAT bindings alive). Because of that, dedicated keepalives (e.g. STUN Binding Indications) are not sent on candidate pairs where consent requests are sent, in accordance with Section 20.2.3 of [RFC5245].

When Secure Real-time Transport Protocol (SRTP) is used, the following considerations are applicable. 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 other's 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.

The mechanism proposed in the document is an optional extension to the ICE protocol, it can be deployed at one end of the two-party communication session without impact on the other party.

5. Solution

Initial consent to send traffic is obtained using ICE [RFC5245]. An endpoint gains consent to send on a candidate pair when the pair enters the Succeeded ICE state. This document establishes a 30 second expiry time on consent. 30 seconds was chosen to balance the need to minimize the time taken to respond to a loss of consent with the desire to reduce the occurrence of spurious failures.

ICE does not identify when consent to send traffic ends. This document describes two ways in which consent to send ends: expiration of consent and immediate revocation of consent, which are discussed in the following sections.

5.1. Expiration of Consent

A full ICE implementation obtains consent to send using ICE. After ICE concludes on a particular candidate pair and whenever the endpoint sends application data on that pair consent is maintained following the procedure described in this document.

An endpoint **MUST NOT** send data other than the messages used to establish consent unless the receiving endpoint has consented to receive data. Connectivity checks that are paced as described in Section 16 of [RFC5245] and responses to connectivity checks are permitted. That is, no application data (e.g., RTP or Datagram Transport Layer Security (DTLS)) can be sent until consent is obtained.

Explicit consent to send is obtained and maintained by sending a 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]).

Consent expires after 30 seconds. That is, if a valid STUN binding response has not been received from the remote peer's transport address in 30 seconds, 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. Implementations **MUST NOT** set the period between checks to less than 4 seconds. This timer is independent of the consent expiry timeout.

Each STUN binding request for consent **MUST** use a new STUN transaction identifier, as described in Section 6 of [RFC5389]. Each STUN binding request for consent is transmitted once only. A sender therefore cannot assume that it will receive a response for every consent request, and a response might be for a previous request (rather than for the most recently sent request).

An endpoint **SHOULD** await a binding response for each request it sends for a time based on the estimated round-trip time (RTT) (see Section 7.2.1 of [RFC5389]) with an allowance for variation in network delay. The RTT value can be updated as described in [RFC5389]. All outstanding STUN consent transactions for a candidate pair **MUST** be discarded when consent expires.

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 identifier, 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 cannot cause a TCP sender to send once the consent timer expires (30 seconds).

An endpoint does not need to maintain consent if it does not send application data. However, an endpoint MUST regain consent before it resumes sending application data. In the absence of any packets, any bindings in middleboxes for the flow might expire. Furthermore, having one peer unable to send is detrimental to many protocols. Absent better information about the network, if an endpoint needs to ensure its NAT or firewall mappings do not expire, it can be done using keepalive or other techniques (see Section 10 of [RFC5245] and see [RFC6263]).

After consent is lost, 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 on the affected candidate pair.

5.2. 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.

6. 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.

7. DTLS applicability

The DTLS applicability is identical to what is described in Section 4.2 of [RFC7350].

8. Security Considerations

This document describes a security mechanism, details of which are mentioned in Section 4.1 and Section 4.2. Consent requires 96 bits transaction ID defined in section 6 of [RFC5389] to be uniformly and randomly chosen from the interval $0 \dots 2^{96}-1$, and be cryptographically strong. This is good enough security against an off-path attacker replaying old STUN consent responses. Consent Verification to avoid attacks using a browser as an attack platform against machines is discussed in Sections 3.3 and 4.2 of [I-D.ietf-rtcweb-security].

The security considerations discussed in [RFC5245] should also be taken into account.

9. IANA Considerations

This document does not require any action from IANA.

10. Acknowledgement

Thanks to Eric Rescorla, Harald Alvestrand, Bernard Aboba, Magnus Westerlund, Cullen Jennings, Christer Holmberg, Simon Perreault, Paul Kyzivat, Emil Ivov, Jonathan Lennox, Inaki Baz Castillo, Rajmohan

Banavi, Christian Groves, Meral Shirazipour, David Black, Barry Leiba, Ben Campbell and Stephen Farrell for their valuable inputs and comments. Thanks to Christer Holmberg for doing a thorough review.

11. References

11.1. Normative References

- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<http://www.rfc-editor.org/info/rfc2119>>.
- [RFC5245] Rosenberg, J., "Interactive Connectivity Establishment (ICE): A Protocol for Network Address Translator (NAT) Traversal for Offer/Answer Protocols", RFC 5245, DOI 10.17487/RFC5245, April 2010, <<http://www.rfc-editor.org/info/rfc5245>>.
- [RFC5389] Rosenberg, J., Mahy, R., Matthews, P., and D. Wing, "Session Traversal Utilities for NAT (STUN)", RFC 5389, DOI 10.17487/RFC5389, October 2008, <<http://www.rfc-editor.org/info/rfc5389>>.

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-security]
Rescorla, E., "Security Considerations for WebRTC", draft-ietf-rtcweb-security-08 (work in progress), February 2015.
- [I-D.ietf-rtcweb-security-arch]
Rescorla, E., "WebRTC Security Architecture", draft-ietf-rtcweb-security-arch-11 (work in progress), March 2015.
- [I-D.ietf-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-04 (work in progress), July 2015.

- [RFC3830] Arkko, J., Carrara, E., Lindholm, F., Naslund, M., and K. Norrman, "MIKEY: Multimedia Internet KEYing", RFC 3830, DOI 10.17487/RFC3830, August 2004, <<http://www.rfc-editor.org/info/rfc3830>>.
- [RFC4568] Andreassen, F., Baugher, M., and D. Wing, "Session Description Protocol (SDP) Security Descriptions for Media Streams", RFC 4568, DOI 10.17487/RFC4568, July 2006, <<http://www.rfc-editor.org/info/rfc4568>>.
- [RFC4953] Touch, J., "Defending TCP Against Spoofing Attacks", RFC 4953, DOI 10.17487/RFC4953, July 2007, <<http://www.rfc-editor.org/info/rfc4953>>.
- [RFC5961] Ramaiah, A., Stewart, R., and M. Dalal, "Improving TCP's Robustness to Blind In-Window Attacks", RFC 5961, DOI 10.17487/RFC5961, August 2010, <<http://www.rfc-editor.org/info/rfc5961>>.
- [RFC6062] Perreault, S., Ed. and J. Rosenberg, "Traversal Using Relays around NAT (TURN) Extensions for TCP Allocations", RFC 6062, DOI 10.17487/RFC6062, November 2010, <<http://www.rfc-editor.org/info/rfc6062>>.
- [RFC6263] Marjou, X. and A. Sollaud, "Application Mechanism for Keeping Alive the NAT Mappings Associated with RTP / RTP Control Protocol (RTCP) Flows", RFC 6263, DOI 10.17487/RFC6263, June 2011, <<http://www.rfc-editor.org/info/rfc6263>>.
- [RFC7350] Petit-Huguenin, M. and G. Salgueiro, "Datagram Transport Layer Security (DTLS) as Transport for Session Traversal Utilities for NAT (STUN)", RFC 7350, DOI 10.17487/RFC7350, August 2014, <<http://www.rfc-editor.org/info/rfc7350>>.

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: tiredy@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 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

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

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

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

This Internet-Draft will expire on April 29, 2017.

Copyright Notice

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

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

Table of Contents

1. Introduction	2
2. Requirements language	3
3. Transport and Middlebox specification	3
3.1. System-provided interfaces	3
3.2. Ability to use IPv4 and IPv6	4
3.3. Usage of temporary IPv6 addresses	4
3.4. Middle box related functions	5
3.5. Transport protocols implemented	6
4. Media Prioritization	7
4.1. Local prioritization	8
4.2. Usage of Quality of Service - DSCP and Multiplexing	9
5. IANA Considerations	11
6. Security Considerations	11
7. Acknowledgements	11
8. References	11
8.1. Normative References	11
8.2. Informative References	15
Appendix A. Change log	16
A.1. Changes from -00 to -01	16
A.2. Changes from -01 to -02	16
A.3. Changes from -02 to -03	17
A.4. Changes from -03 to -04	17
A.5. Changes from -04 to -05	17
A.6. Changes from -05 to -06	17
A.7. Changes from -06 to -07	18
A.8. Changes from -07 to -08	18
A.9. Changes from -08 to -09	18
A.10. Changes from -09 to -10	18
A.11. Changes from -10 to -11	18
A.12. Changes from -11 to -12	19
A.13. Changes from -12 to -13	19
A.14. Changes from -13 to -14	19
A.15. Changes from -14 to -15	19
A.16. Changes from -15 to -16	19
A.17. Changes from -16 to -17	20
Author's Address	20

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 server 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 congestion 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:

- 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.

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:

- 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 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

- 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

- o 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.
- o 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

- o Reworded introduction
- o Removed all references to "WebRTC". It now uses only the term RTCWEB.
- o Addressed a number of clarity / language comments
- o Rewrote the prioritization to cover data channels and to describe multiple ways of prioritizing flows
- o Made explicit reference to "MUST do DTLS-SRTP", and referred to security-arch for details

A.6. Changes from -05 to -06

- o 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 circuit 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