Stream: Internet Engineering Task Force (IETF)
RFC: 8834
Category: Standards Track
Published: October 2020
ISSN: 2070-1721
Authors: C. Perkins, *University of Glasgow*; M. Westerlund, *Ericsson*; J. Ott, *Aalto University*

RFC 8834
Media Transport and Use of RTP in WebRTC

Abstract

The framework for Web Real-Time Communication (WebRTC) 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 is an Internet Standards Track document.

This document is a product of the Internet Engineering Task Force (IETF). It represents the consensus of the IETF community. It has received public review and has been approved for publication by the Internet Engineering Steering Group (IESG). Further information on Internet Standards is available in Section 2 of RFC 7841.

Information about the current status of this document, any errata, and how to provide feedback on it may be obtained at https://www.rfc-editor.org/info/rfc8834.

Copyright Notice

Table of Contents

1. Introduction

The Real-time Transport Protocol (RTP) [RFC3550] provides a framework for delivery of audio and video teleconferencing data and other real-time media applications. Previous work has defined the RTP protocol, along with numerous profiles, payload formats, and other extensions. When combined with appropriate signaling, 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 [RFC8825].

The structure of this memo is as follows. Section 2 outlines our rationale for 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 signaling 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 support many new media encodings, but it 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 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**", "**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. 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-MEDIA-CAPTURE]. A MediaStream consists of zero or more MediaStreamTracks.

MediaStreamTrack: Part of the MediaStream concept defined by the W3C in the WebRTC API [W3C-MEDIA-CAPTURE]. A MediaStreamTrack is an individual stream of media from any type of media source such as a microphone or a camera, but conceptual sources such as an audio mix or a video composition are also possible.

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

Bidirectional transport-layer flow: A bidirectional 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 [RFC7656] and [RFC8825]. Other terms are used according to their definitions from the RTP specification [RFC3550]. In particular, 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 they are **REQUIRED** in all WebRTC endpoints:

- Support for use of multiple simultaneous synchronization source (SSRC) values in a single RTP session, including support for RTP endpoints that send many SSRC values simultaneously, following [RFC3550] and [RFC8108]. The RTCP optimizations for multi-SSRC sessions defined in [RFC8861] **MAY** be supported; if supported, the usage **MUST** be signaled.
- Random choice of SSRC on joining a session; collision detection and resolution for SSRC values (see also Section 4.8).
- Support for reception of RTP data packets containing contributing source (CSRC) lists, as generated by RTP mixers, and RTCP packets relating to CSRCs.
- Sending correct synchronization information in the RTCP Sender Reports, to allow receivers to implement lip synchronization; see Section 5.2.1 regarding support for the rapid RTP synchronization extensions.
- Support for multiple synchronization contexts. Participants that send multiple simultaneous RTP packet streams **SHOULD** do so as part of a single synchronization context, using a single RTCP CNAME for all streams and allowing receivers to play the streams out in a synchronized manner. For compatibility with potential future versions of this specification, or for interoperability with non-WebRTC devices through a gateway, receivers **MUST** support multiple synchronization 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.
- Support for sending and receiving RTCP SR, RR, Source Description (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 Session Description Protocol (SDP) bundle negotiation extension will use the SDP Grouping Framework "mid" attribute to identify media streams. Such endpoints **MUST** implement the RTCP SDES media identification (MID) item described in [RFC8843].
- 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 randomized RTCP transmission intervals to avoid synchronization of RTCP reports; support for RTCP timer reconsideration (Section 6.3.6 of [RFC3550]) and reverse reconsideration (Section 6.3.4 of [RFC3550]).
- 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].

- 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 (nonreduced) 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 [RFC8108]).
- 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 it is not audio specific and can be used with any RTP payload format.
- Ignore unknown RTCP packet types and RTP header extensions. This is to ensure robust handling of future extensions, middlebox behaviors, etc., that can result in receiving RTP header extensions or RTCP packet types that were not signaled. If a compound RTCP packet that contains a mixture of known and unknown RTCP packet types is received, the known packet 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 systems with only Voice over IP (VoIP), 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 the basic RTP/AVP profile [RFC3551], the RTP profile for RTCP-based feedback (RTP/AVPF) [RFC4585], and the secure RTP profile (RTP/SAVP) [RFC3711].

The RTCP-based feedback extensions [RFC4585] are needed for the improved RTCP timer model. 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 7.1.3 of [RFC8108].

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. In other words, 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 [RFC8827].

4.3. Choice of RTP Payload Formats

Mandatory-to-implement audio codecs and RTP payload formats for WebRTC endpoints are defined in [RFC7874]. Mandatory-to-implement video codecs and RTP payload formats for WebRTC endpoints are defined in [RFC7742]. WebRTC endpoints **MAY** additionally implement any other codec for which an RTP payload format and associated signaling 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 signaling 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 signaled 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 signaling contexts based on the RTP payload type, then the assignment of RTP payload type numbers **MUST** be unique across signaling 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 signaling 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 [RFC8843] form a single RTP session).

An endpoint that has signaled 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 signaled 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 signaled for the type of media that is being sent by that source; see Section 4.4. To support rapid rate adaptation by changing codecs, RTP does not require advance signaling for changes between RTP payload formats used by a single SSRC that were signaled during session setup.

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 [RFC8860] (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, the mechanisms described in [RFC8843] 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 [MULTIPLEX].

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 for RTP and one 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 setup 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 signaling channel before it is used. If SDP is used for signaling, 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 [RFC8858].

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 a 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 they 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 behavior. Implementations **MUST** support sending and receiving noncompound RTCP feedback packets [RFC5506]. Use of noncompound RTCP packets **MUST** be negotiated using the signaling channel. If SDP is used for signaling, 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 noncompound RTCP in the signaling 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 bidirectional 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, because the NAT mappings or firewall state is not unnecessarily bloated. The amount of per-flow QoS state kept in the network is also reduced.

4.8. Choice of RTP Synchronization Source (SSRC)

Implementations are **REQUIRED** to support signaled RTP synchronization source (SSRC) identifiers. If SDP is used, this **MUST** be done using the "a=ssrc:" SDP attribute defined in Sections 4.1 and 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 signaled 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 signaled ahead of time. Implementations **MUST** support random SSRC assignment and **MUST** support SSRC collision detection and resolution, according to [RFC3550]. When using signaled 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 is 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 signaled, 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 signaled, the RTP payload type numbers used in an RTP packet stream are often sufficient to associate that packet stream with a signaling context. For example, 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 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 an RTCPeerConnection [W3C.WebRTC], 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 synchronization 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 unsynchronized reference clocks, and hence different synchronization contexts, it will need to use multiple RTCP CNAMEs, one for each synchronization 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 a single RTCPeerConnection (that is, an RTCPeerConnection forms a synchronization 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 that are part of a multiparty 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.

A 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 given in [RFC7164] regarding handling of leap seconds to limit their impact on RTP media play-out and synchronization **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 [RFC7667].

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 mesh-based topologies, such as a point-to-multipoint mesh configured as a single RTP session ("Topo-Mesh" in the terminology of [RFC7667]).

However, a point-to-multipoint mesh constructed using several RTP sessions, implemented in WebRTC using independent RTCPeerConnections [W3C.WebRTC], 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 [RFC7667] 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:

- Video-switching Multipoint Control Units (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 [RFC7667]).
- 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 standardized.

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

- 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 [RFC7667]).

The RTP extensions described in Sections 5.1.1 to 5.1.6 are designed to be used with centralized 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; 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 "Codec Control Messages in the RTP Audio-Visual Profile with Feedback (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 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 centralized 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 fulfill 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 learned that encoder-decoder synchronization 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 signaled. Receivers that detect that encoder-decoder synchronization has been lost **SHOULD** generate an RPSI feedback message if the codec being used supports reference-picture selection. An 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 Temporary Maximum Media Stream Bit Rate Request (TMMBR) feedback message is defined in Sections 3.5.4 and 4.2.1 of Codec Control Messages [RFC5104]. This request and its corresponding Temporary Maximum Media Stream Bit Rate Notification (TMMBN) message [RFC5104] 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 messages 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 signaled following the general mechanism for RTP header extensions defined in [RFC8285], since this gives well-defined semantics to RTP header extensions.

As noted in [RFC8285], 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 signaled (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 Synchronization

Many RTP sessions require synchronization between audio, video, and other content. This synchronization 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 synchronization extensions described in [RFC6051] be implemented in addition to RTCP SR-based synchronization.

This header extension uses the generic header extension framework described in [RFC8285] 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 header extension [RFC6464] **MUST** be implemented. It is **REQUIRED** that implementations be 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 signaling.

This header extension uses the generic header extension framework described in [RFC8285] 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 an 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 optimization of non-critical functions and is hence **OPTIONAL** to implement. If this header extension is implemented, it is **REQUIRED** that implementations be 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 be used, unless the encryption has been explicitly disabled through API or signaling.

This header extension uses the generic header extension framework described in [RFC8285] 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 [RFC8843].

This header extension uses the generic header extension framework described in [RFC8285] 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 [RFC7742].

This header extension uses the generic header extension framework described in [RFC8285] 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 bitrate **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 subsections 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 optimize 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 they **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 it 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 cause 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 need 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 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, and others operate across RTP packets and can be used with any payload format. Note 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 [RFC8854]. 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 bitrates, 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, bitrate, 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 [RFC8836]. 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 [RFC8083]. The RTP circuit breaker is designed to enable applications to recognize 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 signaling will also necessarily establish boundaries to which the media bitrate will conform. The choice of media codecs provides upper and lower bounds on the supported bitrates that the application can utilize to provide useful quality, and the packetization choices that exist. In addition, the signaling channel can establish maximum media bitrate boundaries using, for example, the SDP "b=AS:" or "b=CT:" lines and the RTP/AVPF TMMBR

messages (see Section 5.1.6 of this memo). Signaled 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 [RFC8083] 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 purposes such as TMMBR [RFC5104]. Implementations that have to interwork with such endpoints **MUST** ensure that they stay within the RTP circuit breaker [RFC8083] constraints to limit the congestion they can cause, but they 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 a situation where one direction is dual controlled when the other direction is not controlled. This memo cannot mandate behavior 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 standardize 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] and [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 signaled; 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 "Guidelines for Writers of RTP Payload Format Specifications" [RFC2736], "How to Write an RTP Payload Format" [RFC8088], and "Guidelines for Extending the RTP Control Protocol (RTCP)" [RFC5968]. They also **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. Signaling Considerations

RTP is built with the assumption that an external signaling 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 the session. The RTP/AVP [RFC3551] and RTP/AVPF [RFC4585] profiles can interoperate on a 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 nonsecure 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 signaled. 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(es) and ports for RTP and RTCP **MUST** be signaled for each RTP session. In WebRTC, these transport addresses will be provided by Interactive Connectivity Establishment (ICE) [RFC8445] 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 signaled (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 signaled. Each media type **MAY** also have a number of media type parameters that **MUST** also be signaled 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 signaled. This signaling ensures that a WebRTC endpoint's behavior, especially when sending, is predictable and consistent. For robustness and 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 with 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 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 signaling model and API how these parameters need to be configured, but they will need to either be set in the API or explicitly signaled between the peers.

11. WebRTC API Considerations

The WebRTC API [W3C.WebRTC] and the Media Capture and Streams API [W3C-MEDIA-CAPTURE] define and use 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, such as a microphone or a camera, but conceptual sources, like an audio mix or a video composition, are also possible. The MediaStreamTracks within a MediaStream might need to be synchronized during playback.

A MediaStreamTrack's realization in RTP, in the context of an RTCPeerConnection, consists of a source packet stream, identified by an SSRC, sent within an RTP session that is 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 an 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 optimizations would need to take into account that the constraints for one of the MediaStreamTracks can change at any moment, 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 synchronization base. To ensure that this works in all cases and does not force an endpoint to disrupt the media by changing synchronization 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 to use 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 [RFC8826] 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 outgoing one on any RTCPeerConnection to perform resynchronization 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 the timebase for the synchronization. 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 synchronization. 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, followed by encoding of the media again into the outgoing packet stream. Optimizations of this method are clearly possible and implementation specific.

A WebRTC endpoint **MUST** support receiving multiple MediaStreamTracks, where each of the different MediaStreamTracks (and its sets of associated packet streams) uses different CNAMEs. However, MediaStreamTracks that are received with different CNAMEs have no defined synchronization.

> 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 multistream middleboxes or endpoints that are not WebRTC.

"JavaScript Session Establishment Protocol (JSEP)" [RFC8829] specifies that the binding between the WebRTC MediaStreams, MediaStreamTracks, and the SSRC is done as specified in "WebRTC MediaStream Identification in the Session Description Protocol" [RFC8830]. Section 4.1 of the MediaStream Identification (MSID) document [RFC8830] also defines how to map source packet streams with unknown 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); 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 behavior 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 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 bidirectional 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 it 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 an 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:

  Within a single RTP session, 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.

- 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, which 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 use of these mechanisms to provide differentiated quality of service for some RTP packet streams is desired, 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 centralized 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 bitrate, and it 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. However, 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 multiparty 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 is part of the same session beyond the sender itself.

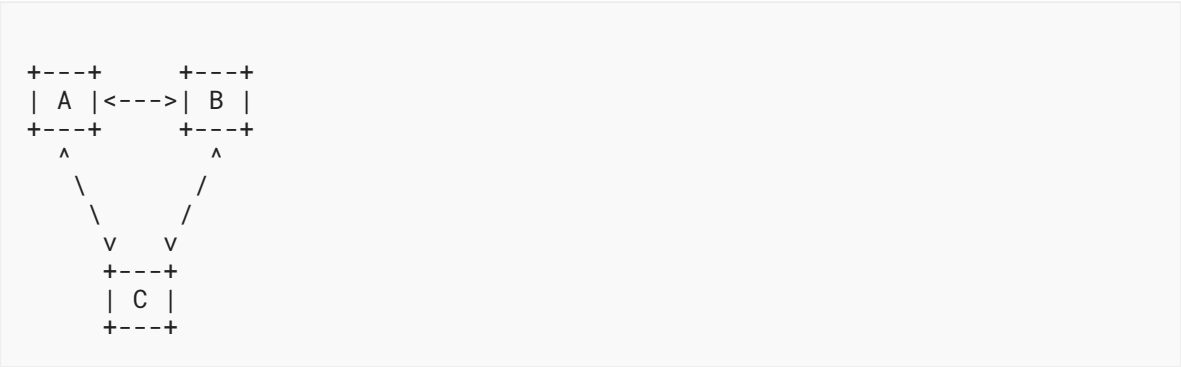

Figure 1: Multi-unicast Using Several RTP Sessions

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 be 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 shown 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 bitrates and RTP session configurations between the different peers, thus not forcing B to endure the same quality reductions as C will if there are limitations in the transport from A to C. It is believed that these advantages outweigh the limitations in debugging power.

- To indirectly connect with multiple peers:

  A common scenario in multiparty 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 centralized conference. The middlebox acts to optimize 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.

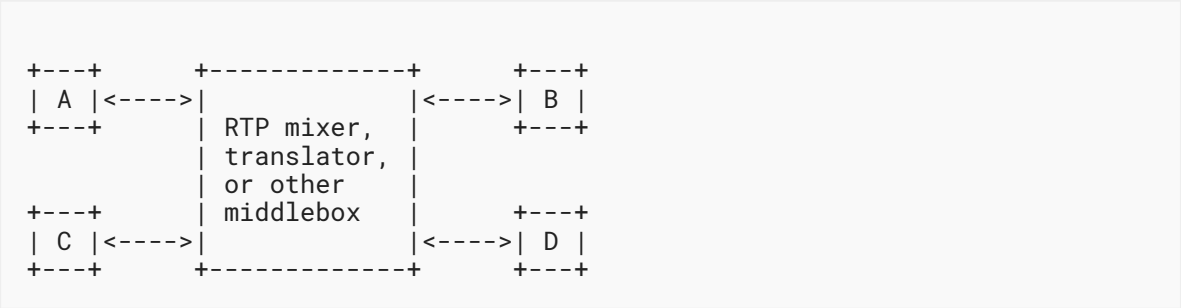

Figure 2: RTP Mixer with Only Unicast Paths

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 multiparty session. Other types of middleboxes 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 having the encoder produce a so-called Intra frame. Another common aspect is limiting the bitrate of a stream to better match the mixed output. Other aspects are controlling the most suitable frame rate, picture resolution, and the trade-off between frame rate and spatial quality. The middlebox has the responsibility to correctly perform congestion control, identify sources, and manage synchronization while providing the application with suitable media optimizations. 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 them 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.

Mixers are expected to not forward RTCP reports regarding RTP packet streams across themselves. This is due to the difference between the RTP packet streams provided to the different endpoints. The original media source lacks information about a mixer's manipulations prior to being sent to the different receivers. This scenario also results in an endpoint's feedback or requests going to the mixer. When the mixer can't act on this by itself, it is forced to go to the original media source to fulfill the receiver's 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 multiparty 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 claiming to come from another endpoint in the session. Any node performing relay can perform noncryptographic 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, Timed Efficient Stream Loss-Tolerant Authentication (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. The are some obvious security and privacy implications in supporting this, but also potential uses. This is supported in the W3C API by taking the received and decoded media and using it as a media source that is re-encoded 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 from the RTP session on which the media was received. This will require media transcoding for congestion control purposes to produce a suitable bitrate 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 optimize 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 will be sent and their allocation of bitrate out of the current available aggregate, as determined by the congestion control.

It is expected that the WebRTC API [W3C.WebRTC] 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 determining how to divide the available bandwidth between the RTP packet streams for the sake of congestion control. 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, with regard to how robust that codec is against packet loss. However, a default policy might be to use the same priority for a redundant RTP packet stream as for the source RTP packet stream.

Secondly, the network can prioritize transport-layer flows and subflows, 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 for prioritizing 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 Differentiated Services Code Point (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 DSCPs to use and know 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 "Differentiated Services Code Point (DSCP) Packet Markings for WebRTC QoS" [RFC8837].

Despite the SRTP media encryption, deep packet inspectors will still be fairly capable of 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 prioritize any payloads carrying only B frames less, as these are less damaging to lose. However, depending on the QoS mechanism and what markings are applied, this can result in not only different packet-drop probabilities but also packet reordering; see [RFC8837] and [RFC7657] for further discussion. As a default policy, all RTP packets related to an 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 (SRs) 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 an 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 synchronization source (SSRC) identifier. The SSRC identifier is carried in each of the RTP packets comprising an 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 need 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 namespace 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., be able to 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:

- In a point-to-point session where each SSRC is associated with either of the two endpoints and the main media-carrying SSRC identifier will be announced in the signaling 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 signaled it to the peer and received acknowledgement on the signaling message. "Source-Specific Media Attributes in the Session Description Protocol (SDP)" [RFC5576] contains a mechanism to signal how the endpoint resolved the SSRC collision.
- SSRC values that have not been signaled 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 from 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 signaling channel to function correctly on both RTP and RTCPeerConnection level.
- 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.
- 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 be able to recognize and handle the case when it occurs. This case becomes even more problematic when media mixers and such 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 the RTP level when the same RTP session is extended across multiple RTCPeerConnections by an RTP middlebox. To resolve the more generic case where multiple RTCPeerConnections are interconnected, identification of the media source or sources that are part of a MediaStreamTrack being propagated across multiple interconnected RTCPeerConnection needs to be preserved across these interconnections.

12.2.3. Media Synchronization 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, synchronization is provided by having a set of RTP packet streams be indicated as coming from the same synchronization 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; whether or not the information is used is up to the receiver.

13. Security Considerations

The overall security architecture for WebRTC is described in [RFC8827], and security considerations for the WebRTC framework are described in [RFC8826]. 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 believed that there are no new security considerations resulting from the combination of these various protocol extensions.

"Extended Secure RTP Profile for Real-time Transport Control Protocol (RTCP)-Based Feedback (RTP/SAVPF)" [RFC5124] provides handling of fundamental issues by offering confidentiality, integrity, and partial source authentication. A media-security solution that is mandatory to implement and use is created by combining this secured RTP profile and DTLS-SRTP keying [RFC5764], as defined by Section 5.5 of [RFC8827].

RTCP packets convey a Canonical Name (CNAME) identifier that is used to associate RTP packet streams that need to be synchronized 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 RFC 7022, 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 RTCP traffic not being congestion controlled; 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 that are based on the reporting interval. This is a particular concern if endpoints use a small but nonzero value for the RTP/AVPF minimal receiver report interval (trr-int) [RFC4585], as discussed in Section 6.1 of [RFC8108].

Premature participant timeout can be avoided by using the fixed (nonreduced) minimum interval when calculating the participant timeout (see Section 4.1 of this memo and Section 7.1.2 of [RFC8108]). 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 bitrate (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 signaling. If further evidence is produced to show that information leakage is significant from audio-level indications, then use of encryption needs to be mandated at that time.

In multiparty communication scenarios using RTP middleboxes, a lot of trust is placed on these middleboxes to preserve the session's security. The middlebox needs to maintain confidentiality and integrity and perform source authentication. As discussed in Section 12.1.1, the middlebox can perform checks that prevent any endpoint participating in a conference from impersonating another. Some additional security considerations regarding multiparty topologies can be found in [RFC7667].

14. IANA Considerations

This document has no IANA actions.

15. References

15.1. 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>.

[RFC2736] Handley, M. and C. Perkins, "Guidelines for Writers of RTP Payload Format Specifications", BCP 36, RFC 2736, DOI 10.17487/RFC2736, December 1999, <https://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, <https://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, <https://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, <https://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, <https://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, <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>.

[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>.

[RFC4961] Wing, D., "Symmetric RTP / RTP Control Protocol (RTCP)", BCP 131, RFC 4961, DOI 10.17487/RFC4961, July 2007, <https://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, <https://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, <https://www.rfc-editor.org/info/rfc5124>.

[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>.

[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>.

[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>.

[RFC6051] Perkins, C. and T. Schierl, "Rapid Synchronisation of RTP Flows", RFC 6051, DOI 10.17487/RFC6051, November 2010, <https://www.rfc-editor.org/info/rfc6051>.

[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>.

[RFC6465] Ivov, E., Ed., Marocco, E., Ed., and J. Lennox, "A Real-time Transport Protocol (RTP) Header Extension for Mixer-to-Client Audio Level Indication", RFC 6465, DOI 10.17487/RFC6465, December 2011, <https://www.rfc-editor.org/info/rfc6465>.

[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, <https://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, <https://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, <https://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, <https://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, <https://www.rfc-editor.org/info/rfc7160>.

[RFC7164] Gross, K. and R. Brandenburg, "RTP and Leap Seconds", RFC 7164, DOI 10.17487/RFC7164, March 2014, <https://www.rfc-editor.org/info/rfc7164>.

[RFC7742] Roach, A.B., "WebRTC Video Processing and Codec Requirements", RFC 7742, DOI 10.17487/RFC7742, March 2016, <https://www.rfc-editor.org/info/rfc7742>.

[RFC7874] Valin, JM. and C. Bran, "WebRTC Audio Codec and Processing Requirements", RFC 7874, DOI 10.17487/RFC7874, May 2016, <https://www.rfc-editor.org/info/rfc7874>.

[RFC8083] Perkins, C. and V. Singh, "Multimedia Congestion Control: Circuit Breakers for Unicast RTP Sessions", RFC 8083, DOI 10.17487/RFC8083, March 2017, <https://www.rfc-editor.org/info/rfc8083>.

[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>.

[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>.

[RFC8285] Singer, D., Desineni, H., and R. Even, Ed., "A General Mechanism for RTP Header Extensions", RFC 8285, DOI 10.17487/RFC8285, October 2017, <https://www.rfc-editor.org/info/rfc8285>.

[RFC8825] Alvestrand, H., "Overview: Real-Time Protocols for Browser-Based Applications", RFC 8825, DOI 10.17487/RFC8825, October 2020, <https://www.rfc-editor.org/info/rfc8825>.

[RFC8826] Rescorla, E., "Security Considerations for WebRTC", RFC 8826, DOI 10.17487/RFC8826, October 2020, <https://www.rfc-editor.org/info/rfc8826>.

[RFC8827] Rescorla, E., "WebRTC Security Architecture", RFC 8827, DOI 10.17487/RFC8827, October 2020, <https://www.rfc-editor.org/info/rfc8827>.

[RFC8843] Holmberg, C., Alvestrand, H., and C. Jennings, "Negotiating Media Multiplexing Using the Session Description Protocol (SDP)", RFC 8843, DOI 10.17487/RFC8843, October 2020, <https://www.rfc-editor.org/info/rfc8843>.

[RFC8854] Uberti, J., "WebRTC Forward Error Correction Requirements", RFC 8854, DOI 10.17487/RFC8854, October 2020, <https://www.rfc-editor.org/info/rfc8854>.

[RFC8858] Holmberg, C., "Indicating Exclusive Support of RTP and RTP Control Protocol (RTCP) Multiplexing Using the Session Description Protocol (SDP)", RFC 8858, DOI 10.17487/RFC8858, October 2020, <https://www.rfc-editor.org/info/rfc8858>.

[RFC8860] Westerlund, M., Perkins, C., and J. Lennox, "Sending Multiple Types of Media in a Single RTP Session", RFC 8860, DOI 10.17487/RFC8860, October 2020, <https://www.rfc-editor.org/info/rfc8860>.

[RFC8861] Lennox, J., Westerlund, M., W, Q., and C. Perkins, "Sending Multiple RTP Streams in a Single RTP Session: Grouping RTP Control Protocol (RTCP) Reception Statistics and Other Feedback", RFC 8861, DOI 10.17487/RFC8861, October 2020, <https://www.rfc-editor.org/info/rfc8861>.

[W3C-MEDIA-CAPTURE] Burnett, D., Bergkvist, A., Jennings, C., Narayanan, A., Aboba, B., Bruaroey, J-I., and H. Boström, "Media Capture and Streams", W3C Candidate Recommendation, 2 July 2019, <https://www.w3.org/TR/2019/CR-mediacapture-streams-20190702/>.

[W3C.WebRTC] Jennings, C., Boström, H., and J-I. Bruaroey, "WebRTC 1.0: Real-time Communication Between Browsers", W3C Candidate Recommendation, 13 December 2019, <https://www.w3.org/TR/2019/CR-webrtc-20191213/>.

15.2. Informative References

[MULTIPLEX] Westerlund, M., Burman, B., Perkins, C., Alvestrand, H., and R. Even, "Guidelines for using the Multiplexing Features of RTP to Support Multiple Media Streams", Work in Progress, Internet-Draft, draft-ietf-avtcore-multiplex-guidelines-12, 16 June 2020, <https://tools.ietf.org/html/draft-ietf-avtcore-multiplex-guidelines-12>.

[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, <https://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, <https://www.rfc-editor.org/info/rfc4383>.

[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>.

[RFC5968] Ott, J. and C. Perkins, "Guidelines for Extending the RTP Control Protocol (RTCP)", RFC 5968, DOI 10.17487/RFC5968, September 2010, <https://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, <https://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, <https://www.rfc-editor.org/info/rfc6792>.

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

[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, <https://www.rfc-editor.org/info/rfc7656>.

[RFC7657] Black, D., Ed. and P. Jones, "Differentiated Services (Diffserv) and Real-Time Communication", RFC 7657, DOI 10.17487/RFC7657, November 2015, <https://www.rfc-editor.org/info/rfc7657>.

[RFC7667] Westerlund, M. and S. Wenger, "RTP Topologies", RFC 7667, DOI 10.17487/RFC7667, November 2015, <https://www.rfc-editor.org/info/rfc7667>.

[RFC8088] Westerlund, M., "How to Write an RTP Payload Format", RFC 8088, DOI 10.17487/RFC8088, May 2017, <https://www.rfc-editor.org/info/rfc8088>.

[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>.

[RFC8829] Uberti, J., Jennings, C., and E. Rescorla, Ed., "JavaScript Session Establishment Protocol (JSEP)", RFC 8829, DOI 10.17487/RFC8829, October 2020, <https://www.rfc-editor.org/info/rfc8829>.

[RFC8830] Alvestrand, H., "WebRTC MediaStream Identification in the Session Description Protocol", RFC 8830, DOI 10.17487/RFC8830, October 2020, <https://www.rfc-editor.org/info/rfc8830>.

[RFC8836] Jesup, R. and Z. Sarker, Ed., "Congestion Control Requirements for Interactive Real-Time Media", RFC 8836, DOI 10.17487/RFC8836, October 2020, <https://www.rfc-editor.org/info/rfc8836>.

[RFC8837] Jones, P., Dhesikan, S., Jennings, C., and D. Druta, "Differentiated Services Code Point (DSCP) Packet Markings for WebRTC QoS", RFC 8837, DOI 10.17487/RFC8837, October 2020, <https://www.rfc-editor.org/info/rfc8837>.

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.

Authors' Addresses

Colin Perkins
University of Glasgow
School of Computing Science
Glasgow
G12 8QQ
United Kingdom
Email: csp@csperkins.org
URI: https://csperkins.org/

Magnus Westerlund
Ericsson
Farogatan 6
SE-164 80 Kista
Sweden
Email: magnus.westerlund@ericsson.com

Jörg Ott
Aalto University
School of Electrical Engineering
FI-02150 Espoo
Finland
Email: jorg.ott@aalto.fi