RTCWEB Working Group
Internet-Draft
Intended status: Standards Track
Expires: January 16, 2014

C. S. Perkins
University of Glasgow
M. Westerlund
Ericsson
J. Ott
Aalto University
July 15, 2013

Web Real-Time Communication (WebRTC): Media Transport and Use of RTP
draft-ietf-rtcweb-rtp-usage-07

Abstract

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

Status of This Memo

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

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

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

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

Copyright Notice

Copyright (c) 2013 IETF Trust and the persons identified as the document authors. All rights reserved.

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

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

Table of Contents

1. Introduction 3
2. Rationale 4
3. Terminology 5
4. WebRTC Use of RTP: Core Protocols 5
4.1. RTP and RTCP 6
4.2. Choice of the RTP Profile 7
4.3. Choice of RTP Payload Formats 7
4.4. Use of RTP Sessions 9
4.5. RTP and RTCP Multiplexing 9
4.6. Reduced Size RTCP 10
4.7. Symmetric RTP/RTCP 10
4.8. Choice of RTP Synchronisation Source (SSRC) 10
4.9. Generation of the RTCP Canonical Name (CNAME) 11
5. WebRTC Use of RTP: Extensions 12
5.1. Conferencing Extensions 12
5.1.1. Full Intra Request (FIR) 13
5.1.2. Picture Loss Indication (PLI) 13
5.1.3. Slice Loss Indication (SLI) 13
5.1.4. Reference Picture Selection Indication (RPSI) 13
5.1.5. Temporal-Spatial Trade-off Request (TSTR) 14
5.1.6. Temporary Maximum Media Stream Bit Rate Request (TMMBR) 14
5.2. Header Extensions 14
5.2.1. Rapid Synchronisation 15
5.2.2. Client-to-Mixer Audio Level 15
5.2.3. Mixer-to-Client Audio Level 15
6. WebRTC Use of RTP: Improving Transport Robustness 16
6.1. Negative Acknowledgements and RTP Retransmission 16
6.2. Forward Error Correction (FEC) 17
7. WebRTC Use of RTP: Rate Control and Media Adaptation 17
7.1. Boundary Conditions and Circuit Breakers 18
7.2. RTCP Limitations for Congestion Control 19
7.3. Congestion Control Interoperability and Legacy Systems . 19
8. WebRTC Use of RTP: Performance Monitoring 20
9. WebRTC Use of RTP: Future Extensions 21
10. Signalling Considerations 21
11. WebRTC API Considerations 23
12. RTP Implementation Considerations 23
12.1. RTP Sessions and PeerConnections 24
12.2. Multiple Sources 25

12.3. Multiparty 25
12.4. SSRC Collision Detection 27
12.5. Contributing Sources and the CSRC List 28
12.6. Media Synchronization 28
12.7. Multiple RTP End-points 29
12.8. Simulcast 30
12.9. Differentiated Treatment of Flows 30
13. Security Considerations 32
14. IANA Considerations 33
15. Open Issues 33
16. Acknowledgements 33
17. References 33
17.1. Normative References 34
17.2. Informative References 37
Appendix A. Supported RTP Topologies 38
A.1. Point to Point 39
A.2. Multi-Unicast (Mesh) 41
A.3. Mixer Based 44
A.3.1. Media Mixing 45
A.3.2. Media Switching 47
A.3.3. Media Projecting 50
A.4. Translator Based 52
A.4.1. Transcoder 52
A.4.2. Gateway / Protocol Translator 53
A.4.3. Relay 55
A.5. End-point Forwarding 58
A.6. Simulcast 60
Authors' Addresses 60

1. Introduction

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

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

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

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

2. Rationale

The RTP framework comprises the RTP data transfer protocol, the RTP control protocol, and numerous RTP payload formats, profiles, and extensions. This range of add-ons has allowed RTP to meet various needs that were not envisaged by the original protocol designers, and to support many new media encodings, but raises the question of what extensions are to be supported by new implementations. The development of the WebRTC framework provides an opportunity for us to review the available RTP features and extensions, and to define a common baseline feature set for all WebRTC implementations of RTP. This builds on the past 20 years development of RTP to mandate the use of extensions that have shown widespread utility, while still remaining compatible with the wide installed base of RTP implementations where possible.

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

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

3. Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119]. The RFC 2119 interpretation of these key words applies only when written in ALL CAPS. Lower- or mixed-case uses of these key words are not to be interpreted as carrying special significance in this memo.

We define the following terms:

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

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

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

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

4. WebRTC Use of RTP: Core Protocols

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

4.1. RTP and RTCP

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

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

- Support for use of multiple simultaneous SSRC values in a single RTP session, including support for RTP end-points that send many SSRC values simultaneously, following [RFC3550] and [I-D.ietf-avtcore-rtp-multi-stream]. Support for the RTCP optimisations for multi-SSRC sessions defined in [I-D.ietf-avtcore-rtp-multi-stream-optimisation] is RECOMMENDED.

  * (tbd: is draft-westerlund-mmusic-max-ssrc-01 needed?)

- 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 CSRC lists, as generated by RTP mixers, and RTCP packets relating to CSRCs.

- Support for sending correct synchronization information in the RTCP Sender Reports, to allow a receiver to implement lip-sync, with RECOMMENDED support for the rapid RTP synchronisation extensions (see Section 5.2.1).

- Support for sending and receiving RTCP SR, RR, SDES, and BYE packet types, with OPTIONAL support for other RTCP packet types; implementations MUST ignore unknown RTCP packet types.

- Support for multiple end-points in a single RTP session, and for scaling the RTCP transmission interval according to the number of participants in the session; support for randomised RTCP transmission intervals to avoid synchronisation of RTCP reports; support for RTCP timer reconsideration.

- Support for configuring the RTCP bandwidth as a fraction of the media bandwidth, and for configuring the fraction of the RTCP bandwidth allocated to senders, e.g., using the SDP "b=" line.

It is known that a significant number of legacy RTP implementations, especially those targeted at VoIP-only systems, do not support all of

the above features, and in some cases do not support RTCP at all. Implementers are advised to consider the requirements for graceful degradation when interoperating with legacy implementations.

Other implementation considerations are discussed in Section 12.

4.2. Choice of the RTP Profile

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

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

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

The secure RTP profile [RFC3711] is needed to provide media encryption, integrity protection, replay protection and a limited form of source authentication. WebRTC implementations MUST NOT send packets using the basic RTP/AVP profile or the RTP/AVPF profile; they MUST employ the full RTP/SAVPF profile to protect all RTP and RTCP packets that are generated. The default and mandatory to implement transforms listed in Section 5 of [RFC3711] SHALL apply.

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

4.3. Choice of RTP Payload Formats

The set of mandatory to implement codecs and RTP payload formats for WebRTC is not specified in this memo. Implementations can support

any codec for which an RTP payload format and associated signalling is defined. Implementation cannot assume that the other participants in an RTP session understand any RTP payload format, no matter how common; support for all RTP payload formats MUST be negotiated before they are used.

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

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

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

4.4. Use of RTP Sessions

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

In modern day networks, however, with the widespread use of network address/port translators (NAT/NAPT) and firewalls, it is desirable to reduce the number of transport-layer flows used by RTP applications. This can be done by sending all the RTP media streams in a single RTP session, which will comprise a single transport-layer flow (this will prevent the use of some quality-of-service mechanisms, as discussed in Section 12.9). Implementations are REQUIRED to support transport of all RTP media streams, independent of media type, in a single RTP session according to [I-D.ietf-avtcore-multi-media-rtp-session]. If such RTP session set-up is to be used, this MUST be negotiated during the signalling phase [I-D.ietf-mmusic-sdp-bundle-negotiation].

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

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

4.5. RTP and RTCP Multiplexing

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

a single port for each RTP session is REQUIRED, as specified in [RFC5761]. For backwards compatibility, implementations are also REQUIRED to support RTP and RTCP sent on separate transport-layer addresses.

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

4.6. Reduced Size RTCP

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

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

4.7. Symmetric RTP/RTCP

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

4.8. Choice of RTP Synchronisation Source (SSRC)

Implementations are REQUIRED to support signalled RTP synchronisation source (SSRC) identifiers, using the "a=ssrc:" SDP attribute defined

in Section 4.1 and Section 5 of [RFC5576]. Implementations MUST also support the "previous-ssrc" source attribute defined in Section 6.2 of [RFC5576]. Other per-SSRC attributes defined in [RFC5576] MAY be supported.

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

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

4.9. Generation of the RTCP Canonical Name (CNAME)

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

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

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

5. WebRTC Use of RTP: Extensions

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

5.1. Conferencing Extensions

RTP is inherently a group communication protocol. Groups can be implemented using a centralised server, multi-unicast, or using IP multicast. While IP multicast is popular in IPTV systems, overlay-based topologies dominate in interactive conferencing environments. Such overlay-based topologies typically use one or more central servers to connect end-points in a star or flat tree topology. These central servers can be implemented in a number of ways as discussed in Appendix A, and in the memo on RTP Topologies [I-D.westerlund-avtcore-rtp-topologies-update].

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

- The use of video switching MCUs makes the use of RTCP for congestion control and quality of service reports problematic (see Section 3.7 of [I-D.westerlund-avtcore-rtp-topologies-update]).

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

- RTP Transport Translators (Topo-Translator) are not of immediate interest to WebRTC, although the main difference compared to point to point is the possibility of seeing multiple different transport paths in any RTCP feedback.

Accordingly, only Point to Point (Topo-Point-to-Point), Multiple concurrent Point to Point (Mesh) and RTP Mixers (Topo-Mixer) topologies are needed to achieve the use-cases to be supported in WebRTC initially. These RECOMMENDED topologies are expected to be supported by all WebRTC end-points (these topologies require no special RTP-layer support in the end-point if the RTP features mandated in this memo are implemented).

The RTP extensions described in Section 5.1.1 to Section 5.1.6 are designed to be used with centralised conferencing, where an RTP

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

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

5.1.1. Full Intra Request (FIR)

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

5.1.2. Picture Loss Indication (PLI)

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

5.1.3. Slice Loss Indication (SLI)

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

5.1.4. Reference Picture Selection Indication (RPSI)

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

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

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

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

This feedback message is defined in Sections 3.5.4 and 4.2.1 of the Codec Control Messages [RFC5104]. This message and its notification message are used by a media receiver to inform the sending party that there is a current limitation on the amount of bandwidth available to this receiver. This can be various reasons for this: for example, an RTP mixer can use this message to limit the media rate of the sender being forwarded by the mixer (without doing media transcoding) to fit the bottlenecks existing towards the other session participants. WebRTC senders are REQUIRED to implement support for TMMBR messages, and MUST follow bandwidth limitations set by a TMMBR message received for their SSRC. The sending of TMMBR requests is OPTIONAL.

5.2. Header Extensions

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

As noted in [RFC5285], the requirement from the RTP specification that header extensions are "designed so that the header extension may be ignored" [RFC3550] stands. To be specific, header extensions MUST only be used for data that can safely be ignored by the recipient without affecting interoperability, and MUST NOT be used when the presence of the extension has changed the form or nature of the rest

of the packet in a way that is not compatible with the way the stream is signalled (e.g., as defined by the payload type). Valid examples might include metadata that is additional to the usual RTP information.

5.2.1. Rapid Synchronisation

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

5.2.2. Client-to-Mixer Audio Level

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

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

5.2.3. Mixer-to-Client Audio Level

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

6. WebRTC Use of RTP: Improving Transport Robustness

There are tools that can make RTP media streams robust against packet loss and reduce the impact of loss on media quality. However, they all add extra bits compared to a non-robust stream. The overhead of these extra bits needs to be considered, and the aggregate bit-rate MUST be rate controlled to avoid causing network congestion (see Section 7). As a result, improving robustness might require a lower base encoding quality, but has the potential to deliver that quality with fewer errors. The mechanisms described in the following sub-sections can be used to improve tolerance to packet loss.

6.1. Negative Acknowledgements and RTP Retransmission

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

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

The RTP Retransmission Payload Format [RFC4588] offers the ability to retransmit lost packets based on NACK feedback. Retransmission needs to be used with care in interactive real-time applications to ensure that the retransmitted packet arrives in time to be useful, but can be effective in environments with relatively low network RTT (an RTP sender can estimate the RTT to the receivers using the information in RTCP SR and RR packets, as described at the end of Section 6.4.1 of [RFC3550]). The use of retransmissions can also increase the forward RTP bandwidth, and can potentially worsen the problem if the packet loss was caused by network congestion. We note, however, that retransmission of an important lost packet to repair decoder state can have lower cost than sending a full intra frame. It is not appropriate to blindly retransmit RTP packets in response to a NACK. The importance of lost packets and the likelihood of them arriving in time to be useful needs to be considered before RTP retransmission is used.

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

6.2. Forward Error Correction (FEC)

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

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

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

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

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

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

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

7.1. Boundary Conditions and Circuit Breakers

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

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

7.2. RTCP Limitations for Congestion Control

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

The total RTCP bandwidth is limited in its transmission rate to a fraction of the RTP traffic (by default 5%). RTCP packets are larger than, e.g., TCP ACKs (even when non-compound RTCP packets are used). The RTP media stream bit rate thus limits the maximum feedback rate as a function of the mean RTCP packet size.

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

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

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

7.3. Congestion Control Interoperability and Legacy Systems

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

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

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

With proprietary congestion control algorithms issues can arise when different algorithms and implementations interact in a communication session. If the different implementations have made different choices in regards to the type of adaptation, for example one sender based, and one receiver based, then one could end up in situation where one direction is dual controlled, when the other direction is not controlled. This memo cannot mandate behaviour for proprietary congestion control algorithms, but implementations that use such algorithms ought to be aware of this issue, and try to ensure that both effective congestion control is negotiated for media flowing in both directions. If the IETF were to standardise both sender- and receiver-based congestion control algorithms for WebRTC traffic in the future, the issues of interoperability, control, and ensuring that both directions of media flow are congestion controlled would also need to be considered.

8. WebRTC Use of RTP: Performance Monitoring

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

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

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

9. WebRTC Use of RTP: Future Extensions

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

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

10. Signalling Considerations

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

RTP Profile: The name of the RTP profile to be used in session. The RTP/AVP [RFC3551] and RTP/AVPF [RFC4585] profiles can interoperate on basic level, as can their secure variants RTP/SAVP [RFC3711] and RTP/SAVPF [RFC5124]. The secure variants of the profiles do not directly interoperate with the non-secure variants, due to the presence of additional header fields for authentication in SRTP packets and cryptographic transformation of the payload. WebRTC requires the use of the RTP/SAVPF profile, and this MUST be signalled if SDP is used. Interworking functions might transform

this into the RTP/SAVP profile for a legacy use case, by indicating to the WebRTC end-point that the RTP/SAVPF is used, and limiting the usage of the "a=rtcp:" attribute to indicate a trr-int value of 4 seconds.

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

RTP Payload Types, media formats, and format parameters: The mapping between media type names (and hence the RTP payload formats to be used), and the RTP payload type numbers MUST be signalled. Each media type MAY also have a number of media type parameters that MUST also be signalled to configure the codec and RTP payload format (the "a=fmtp:" line from SDP). Section 4.3 of this memo discusses requirements for uniqueness of payload types.

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

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

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

11. WebRTC API Considerations

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

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

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

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

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

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

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

12. RTP Implementation Considerations

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

12.1. RTP Sessions and PeerConnections

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

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

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

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

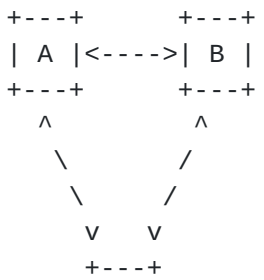




Figure 1: Multi-unicast

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

12.2. Multiple Sources

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

tbd: there needs to be a way of indicating how RTP stream relate when there are multiple sources, possibly with simulcast or layered coding, and different types of mixer or other middlebox. It is possible that the various BUNDLE/Plan-X proposals will solve this, but it might also need RTP-level stream identification. To be resolved once the outcome of the BUNDLE and plan-X discussions is known.

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

12.3. Multiparty

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

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

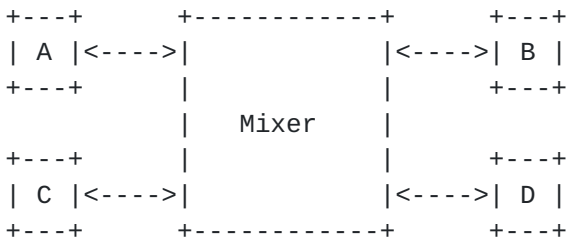


Figure 2: RTP Mixer with Only Unicast Paths

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

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

Mixers also need to be trusted nodes when it comes to security as it manipulates either RTP or the media itself before sending it on

towards the end-point(s), thus they need to be able to decrypt and then encrypt it before sending it out.

12.4. SSRC Collision Detection

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

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

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

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

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

These SSRC/CSRC collisions can only be handled on RTP level as long as the same RTP session is extended across multiple PeerConnections by a RTP middlebox. To resolve the more generic case where multiple

PeerConnections are interconnected, then identification of the media source(s) part of a MediaStreamTrack being propagated across multiple interconnected PeerConnection needs to be preserved across these interconnections.

12.5. Contributing Sources and the CSRC List

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

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

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

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

12.6. Media Synchronization

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

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

12.7. Multiple RTP End-points

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

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

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

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

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

12.8. Simulcast

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

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

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

12.9. Differentiated Treatment of Flows

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

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

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

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

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

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

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

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

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

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

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

13. Security Considerations

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

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

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

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

The guidelines in [RFC6562] apply when using variable bit rate (VBR) audio codecs such as Opus (see Section 4.3 for discussion of mandated

audio codecs). These guidelines in [RFC6562] also apply, but are of lesser importance, when using the client-to-mixer audio level header extensions (Section 5.2.2) or the mixer-to-client audio level header extensions (Section 5.2.3).

14. IANA Considerations

This memo makes no request of IANA.

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

15. Open Issues

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

1. tbd: The API mapping to RTP level concepts has to be agreed and documented in Section 11.

2. tbd: An open question if any requirements are needed to agree and limit the number of simultaneously used media sources (SSRCs) within an RTP session. See Section 12.2 and Section 4.1.

3. tbd: The method for achieving simulcast of a media source has to be decided as discussed in Section 12.8.

4. tbd: Possible documentation of what support for differentiated treatment that are needed on RTP level as the API and the network level specification matures as discussed in Section 12.9.

5. tbd: Editing of Appendix A to remove redundancy between this and the update of RTP Topologies [I-D.westerlund-avtcore-rtp-topologies-update].

16. Acknowledgements

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

17. References

17.1. Normative References

[I-D.ietf-avtcore-6222bis]
Begen, A., Perkins, C., Wing, D., and E. Rescorla, "Guidelines for Choosing RTP Control Protocol (RTCP) Canonical Names (CNAMEs)", draft-ietf-avtcore-6222bis-06 (work in progress), July 2013.

[I-D.ietf-avtcore-avp-codecs]
Terriberry, T., "Update to Remove DVI4 from the Recommended Codecs for the RTP Profile for Audio and Video Conferences with Minimal Control (RTP/AVP)", draft-ietf-avtcore-avp-codecs-03 (work in progress), July 2013.

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

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

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

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

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

[I-D.ietf-avtext-multiple-clock-rates]

Petit-Huguenin, M. and G. Zorn, "Support for Multiple Clock Rates in an RTP Session", draft-ietf-avtext-multiple-clock-rates-09 (work in progress), April 2013.

[I-D.ietf-mmusic-sdp-bundle-negotiation]
Holmberg, C., Alvestrand, H., and C. Jennings, "Multiplexing Negotiation Using Session Description Protocol (SDP) Port Numbers", draft-ietf-mmusic-sdp-bundle-negotiation-04 (work in progress), June 2013.

[I-D.ietf-rtcweb-overview]
Alvestrand, H., "Overview: Real Time Protocols for Brower-based Applications", draft-ietf-rtcweb-overview-06 (work in progress), February 2013.

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

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

[I-D.westerlund-avtcore-transport-multiplexing]
Westerlund, M. and C. Perkins, "Multiple RTP Sessions on a Single Lower-Layer Transport", draft-westerlund-avtcore-transport-multiplexing-05 (work in progress), February 2013.

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

[RFC2736] Handley, M. and C. Perkins, "Guidelines for Writers of RTP Payload Format Specifications", BCP 36, RFC 2736, December 1999.

[RFC3550] Schulzrinne, H., Casner, S., Frederick, R., and V. Jacobson, "RTP: A Transport Protocol for Real-Time Applications", STD 64, RFC 3550, July 2003.

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

[RFC3556] Casner, S., "Session Description Protocol (SDP) Bandwidth Modifiers for RTP Control Protocol (RTCP) Bandwidth", RFC 3556, July 2003.

[RFC3711] Baugher, M., McGrew, D., Naslund, M., Carrara, E., and K. Norrman, "The Secure Real-time Transport Protocol (SRTP)", RFC 3711, March 2004.

[RFC4585] Ott, J., Wenger, S., Sato, N., Burmeister, C., and J. Rey, "Extended RTP Profile for Real-time Transport Control Protocol (RTCP)-Based Feedback (RTP/AVPF)", RFC 4585, July 2006.

[RFC4588] Rey, J., Leon, D., Miyazaki, A., Varsa, V., and R. Hakenberg, "RTP Retransmission Payload Format", RFC 4588, July 2006.

[RFC4961] Wing, D., "Symmetric RTP / RTP Control Protocol (RTCP)", BCP 131, RFC 4961, July 2007.

[RFC5104] Wenger, S., Chandra, U., Westerlund, M., and B. Burman, "Codec Control Messages in the RTP Audio-Visual Profile with Feedback (AVPF)", RFC 5104, February 2008.

[RFC5124] Ott, J. and E. Carrara, "Extended Secure RTP Profile for Real-time Transport Control Protocol (RTCP)-Based Feedback (RTP/SAVPF)", RFC 5124, February 2008.

[RFC5285] Singer, D. and H. Desineni, "A General Mechanism for RTP Header Extensions", RFC 5285, July 2008.

[RFC5506] Johansson, I. and M. Westerlund, "Support for Reduced-Size Real-Time Transport Control Protocol (RTCP): Opportunities and Consequences", RFC 5506, April 2009.

[RFC5761] Perkins, C. and M. Westerlund, "Multiplexing RTP Data and Control Packets on a Single Port", RFC 5761, April 2010.

[RFC5764] McGrew, D. and E. Rescorla, "Datagram Transport Layer Security (DTLS) Extension to Establish Keys for the Secure Real-time Transport Protocol (SRTP)", RFC 5764, May 2010.

[RFC6051] Perkins, C. and T. Schierl, "Rapid Synchronisation of RTP Flows", RFC 6051, November 2010.

[RFC6464] Lennox, J., Ivov, E., and E. Marocco, "A Real-time Transport Protocol (RTP) Header Extension for Client-to-Mixer Audio Level Indication", RFC 6464, December 2011.

[RFC6465] Ivov, E., Marocco, E., and J. Lennox, "A Real-time Transport Protocol (RTP) Header Extension for Mixer-to-Client Audio Level Indication", RFC 6465, December 2011.

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

17.2. Informative References

[I-D.alvestrand-rtcweb-msid]
Alvestrand, H., "Cross Session Stream Identification in the Session Description Protocol", draft-alvestrand-rtcweb-msid-02 (work in progress), May 2012.

[I-D.ietf-avt-srtp-ekt]
Wing, D., McGrew, D., and K. Fischer, "Encrypted Key Transport for Secure RTP", draft-ietf-avt-srtp-ekt-03 (work in progress), October 2011.

[I-D.ietf-rtcweb-qos]
Dhesikan, S., Druta, D., Jones, P., and J. Polk, "DSCP and other packet markings for RTCWeb QoS", draft-ietf-rtcweb-qos-00 (work in progress), October 2012.

[I-D.ietf-rtcweb-use-cases-and-requirements]
Holmberg, C., Hakansson, S., and G. Eriksson, "Web Real-Time Communication Use-cases and Requirements", draft-ietf-rtcweb-use-cases-and-requirements-11 (work in progress), June 2013.

[I-D.jesup-rtp-congestion-reqs]
Jesup, R. and H. Alvestrand, "Congestion Control Requirements For Real Time Media", draft-jesup-rtp-congestion-reqs-00 (work in progress), March 2012.

[I-D.westerlund-avtcore-multiplex-architecture]
Westerlund, M., Perkins, C., and H. Alvestrand, "Guidelines for using the Multiplexing Features of RTP", draft-westerlund-avtcore-multiplex-architecture-03 (work in progress), February 2013.

[I-D.westerlund-avtcore-rtp-topologies-update]
Westerlund, M. and S. Wenger, "RTP Topologies", draft-westerlund-avtcore-rtp-topologies-update-02 (work in progress), February 2013.

[RFC3611] Friedman, T., Caceres, R., and A. Clark, "RTP Control Protocol Extended Reports (RTCP XR)", RFC 3611, November 2003.

[RFC4341] Floyd, S. and E. Kohler, "Profile for Datagram Congestion Control Protocol (DCCP) Congestion Control ID 2: TCP-like Congestion Control", RFC 4341, March 2006.

[RFC4342] Floyd, S., Kohler, E., and J. Padhye, "Profile for Datagram Congestion Control Protocol (DCCP) Congestion Control ID 3: TCP-Friendly Rate Control (TFRC)", RFC 4342, March 2006.

[RFC4383] Baugher, M. and E. Carrara, "The Use of Timed Efficient Stream Loss-Tolerant Authentication (TESLA) in the Secure Real-time Transport Protocol (SRTP)", RFC 4383, February 2006.

[RFC4828] Floyd, S. and E. Kohler, "TCP Friendly Rate Control (TFRC): The Small-Packet (SP) Variant", RFC 4828, April 2007.

[RFC5348] Floyd, S., Handley, M., Padhye, J., and J. Widmer, "TCP Friendly Rate Control (TFRC): Protocol Specification", RFC 5348, September 2008.

[RFC5576] Lennox, J., Ott, J., and T. Schierl, "Source-Specific Media Attributes in the Session Description Protocol (SDP)", RFC 5576, June 2009.

[RFC5681] Allman, M., Paxson, V., and E. Blanton, "TCP Congestion Control", RFC 5681, September 2009.

[RFC5968] Ott, J. and C. Perkins, "Guidelines for Extending the RTP Control Protocol (RTCP)", RFC 5968, September 2010.

[RFC6263] Marjou, X. and A. Sollaud, "Application Mechanism for Keeping Alive the NAT Mappings Associated with RTP / RTP Control Protocol (RTCP) Flows", RFC 6263, June 2011.

Appendix A. Supported RTP Topologies

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

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

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

A.1. Point to Point

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

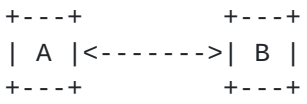


Figure 3: Point to Point

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

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

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

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

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

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

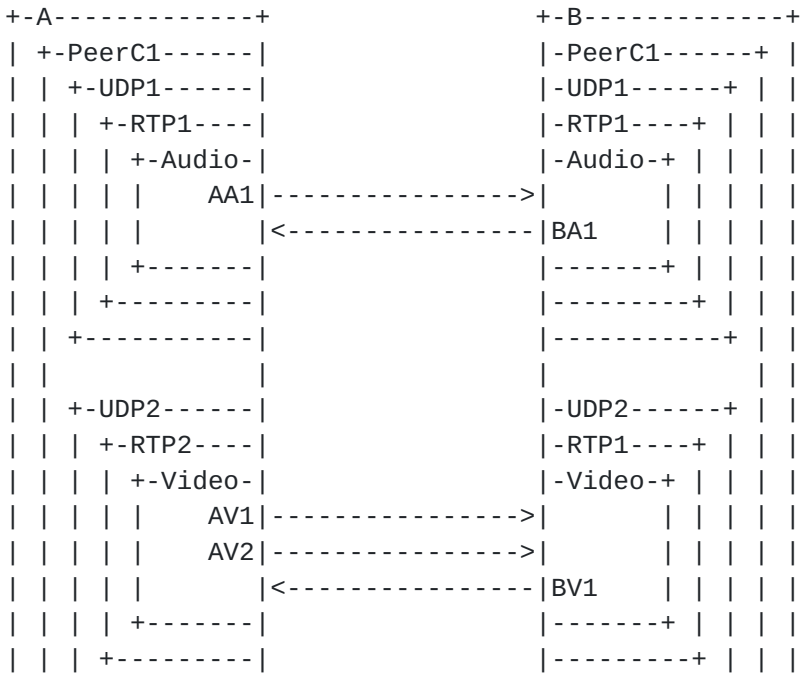




Figure 4: Point to Point: Multiple RTP sessions

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

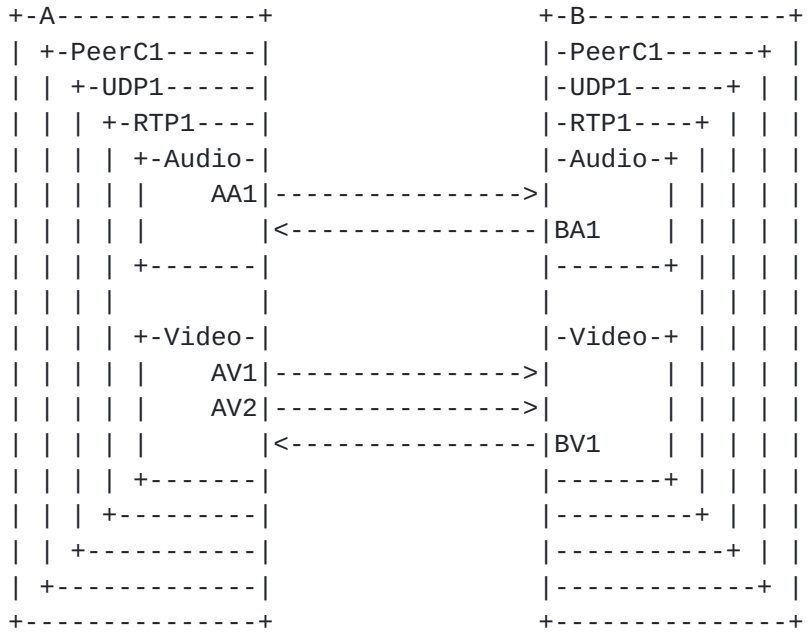


Figure 5: Point to Point: Single RTP session.

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

A.2. Multi-Unicast (Mesh)

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

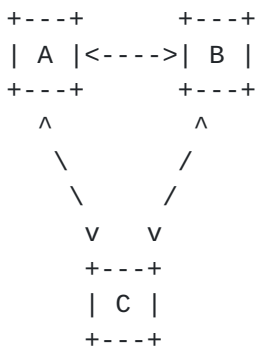
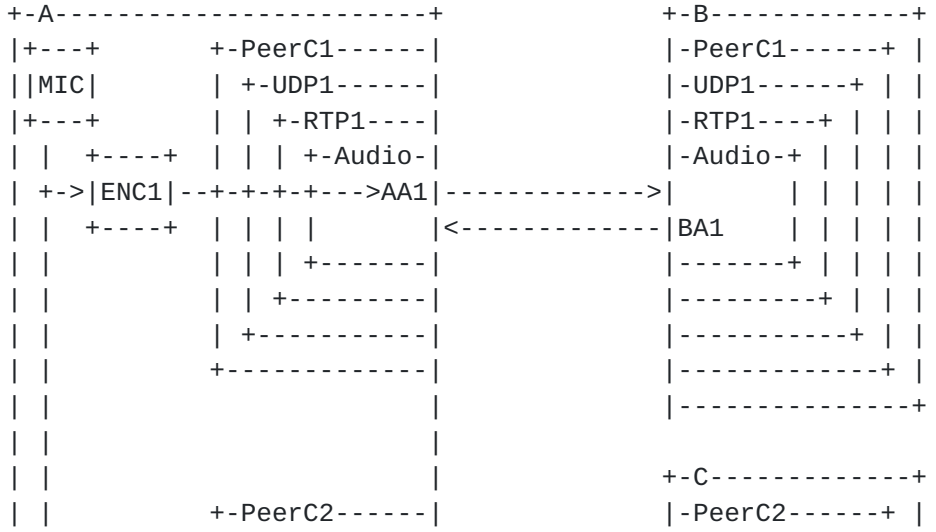


Figure 6: Multi-unicast

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



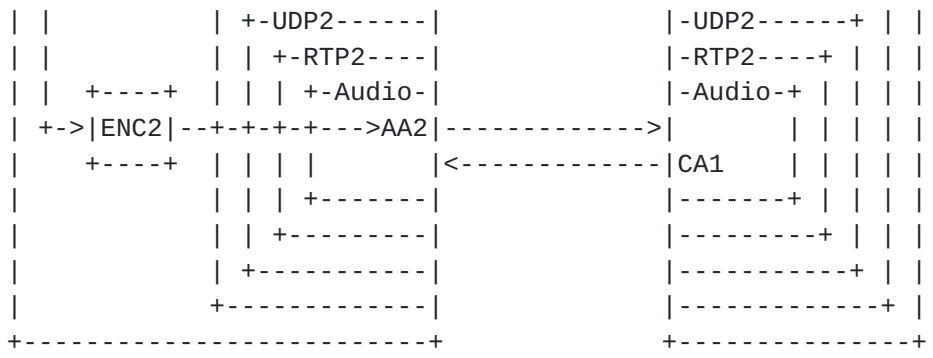
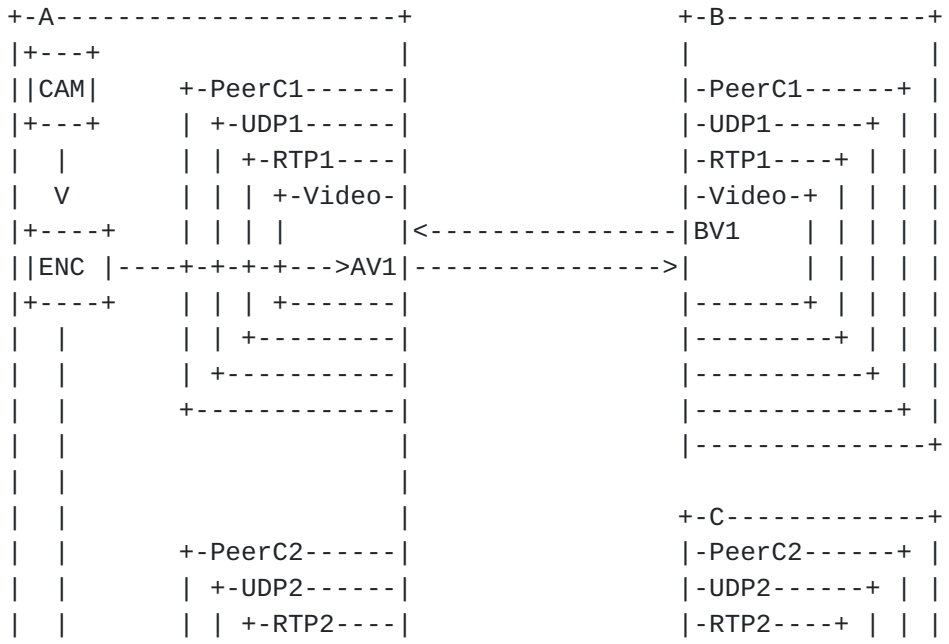


Figure 7: Session structure for Multi-Unicast Setup

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

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



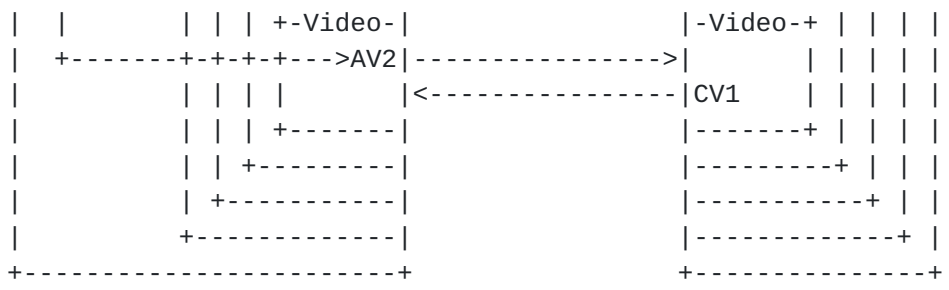


Figure 8: Single Encoder Multi-Unicast Setup

This will clearly save resources consumed by encoding but does introduce the need for the end-point A to make decisions on how it encodes the media so it suites delivery to both B and C. This is not limited to congestion control, also preferred resolution to receive based on dispaly area available is another aspect requiring consideration. The need for this type of decision logic does arise in several different topologies and implementation.

A.3. Mixer Based

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

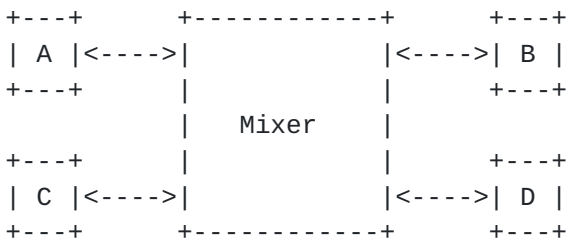


Figure 9: RTP Mixer with Only Unicast Paths

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

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

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

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

A.3.1. Media Mixing

This type of mixer is one which clearly can be called RTP mixer is likely the one that most thinks of when they hear the term mixer. Its basic patter of operation is that it will receive the different participants RTP media stream. Select which that are to be included in a media domain mix of the incoming RTP media streams. Then create a single outgoing stream from this mix.

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

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

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

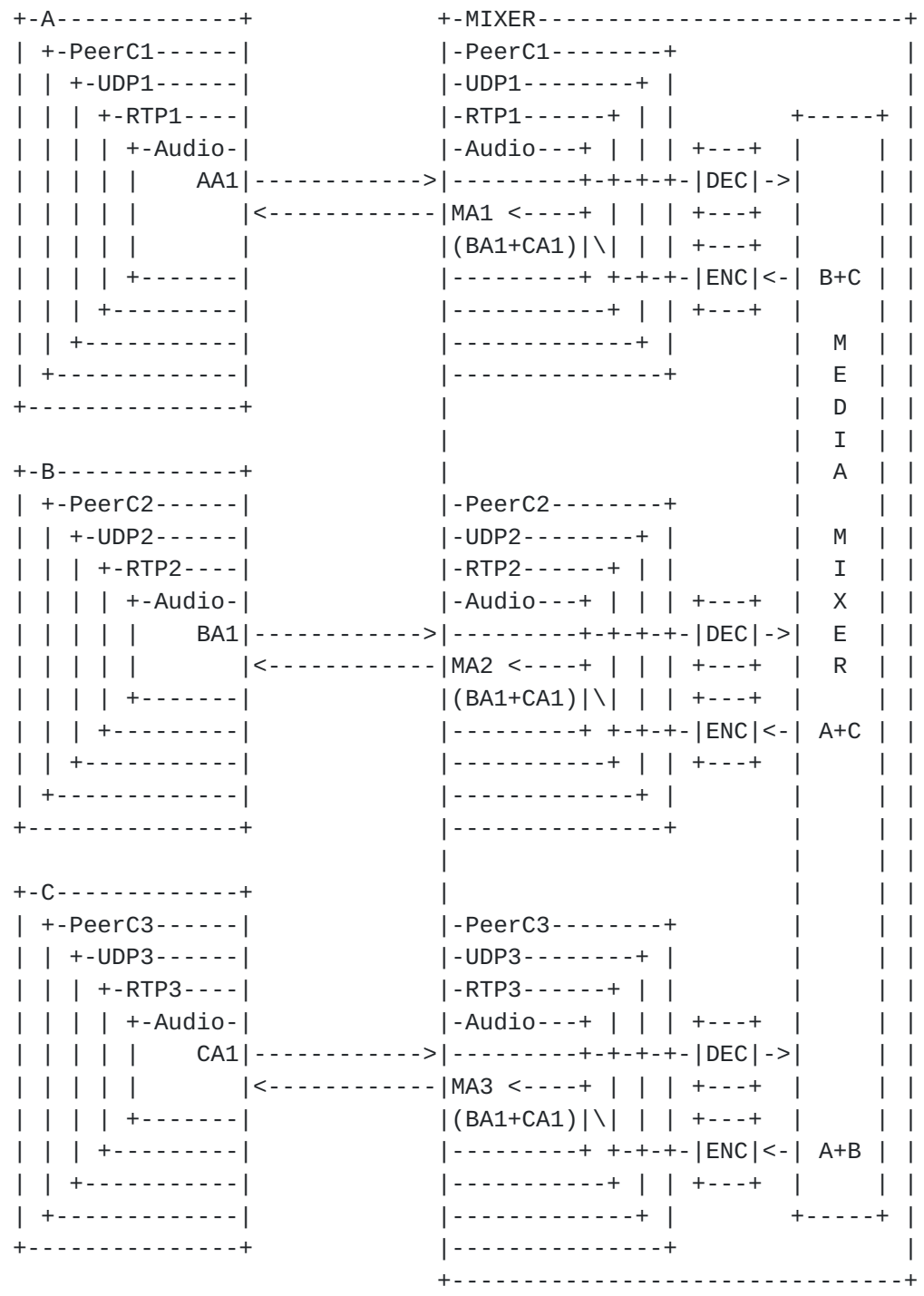


Figure 10: Session and SSRC details for Media Mixer

From an RTP perspective media mixing can be very straight forward as can be seen in Figure 10. The mixer present one SSRC towards the peer client, e.g. MA1 to Peer A, which is the media mix of the other participants. As each peer receives a different version produced by the mixer there are no actual relation between the different RTP

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

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

A.3.1.1. RTP Session Termination

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

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

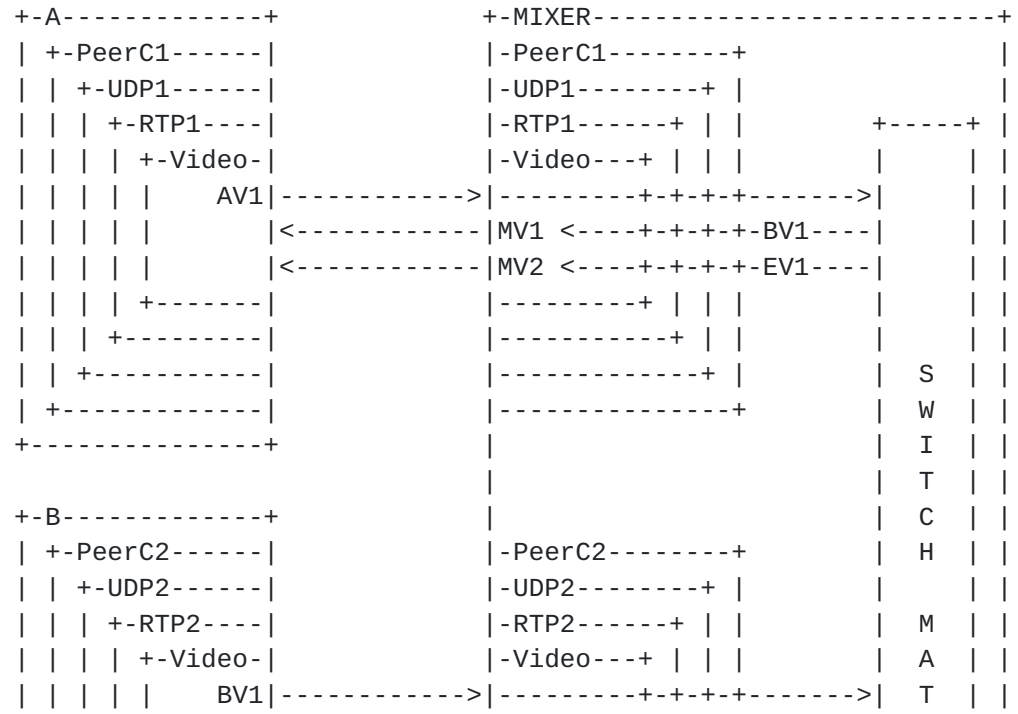
A.3.2. Media Switching

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

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

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

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



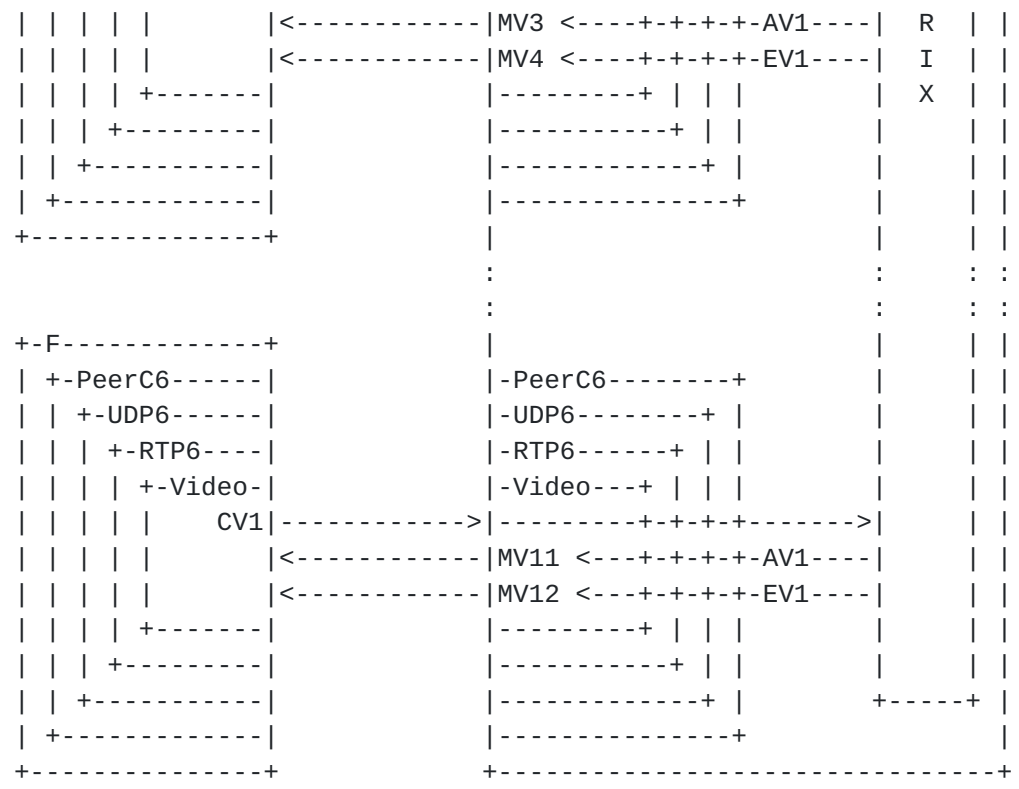


Figure 11: Media Switching RTP Mixer

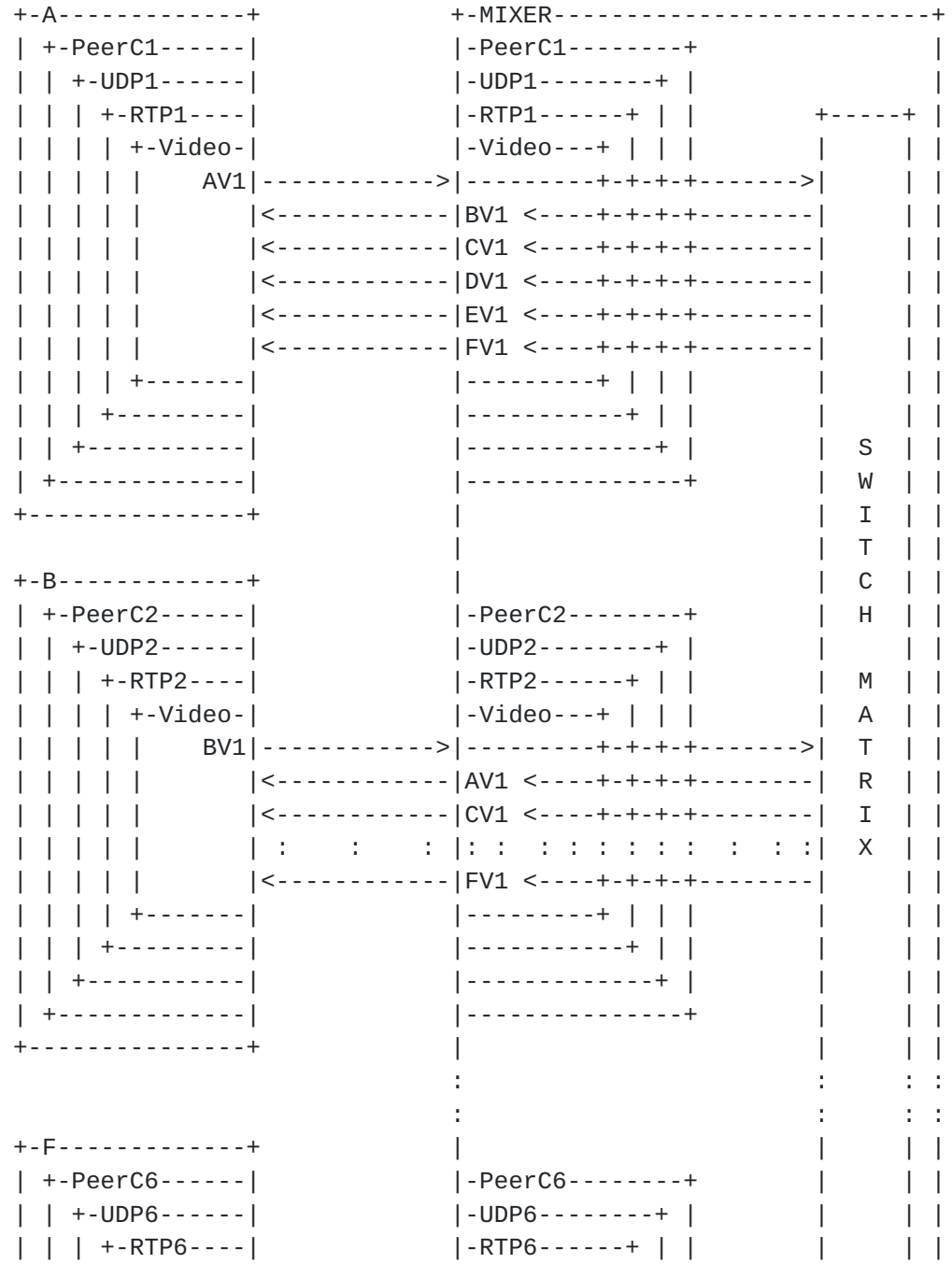
The Media Switching RTP mixer can similar to the Media Mixing one reduce the bit-rate needed towards the different peers by selecting and switching in a sub-set of RTP media streams out of the ones it receives from the conference participations.

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

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

A.3.3. Media Projecting

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



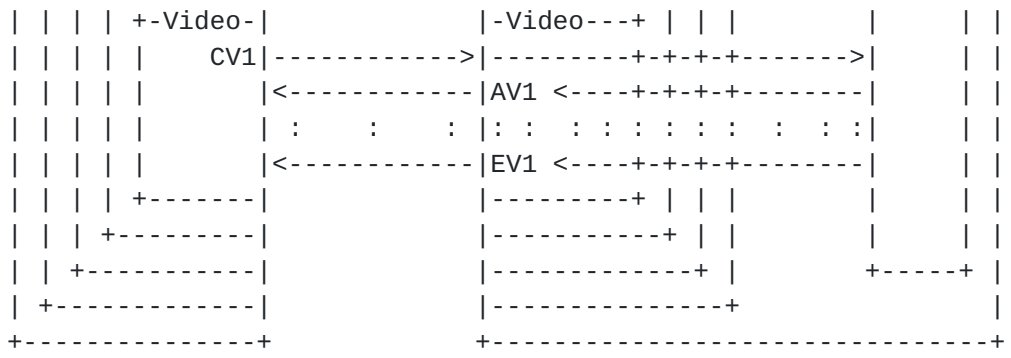


Figure 12: Media Projecting Mixer

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

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

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

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

A.4. Translator Based

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

A.4.1. Transcoder

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

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

transcoding as the 2.5 Mbps source stream. At the same time increasing media bit-rate further than what is needed to represent the incoming quality accurate is also wasted resources.

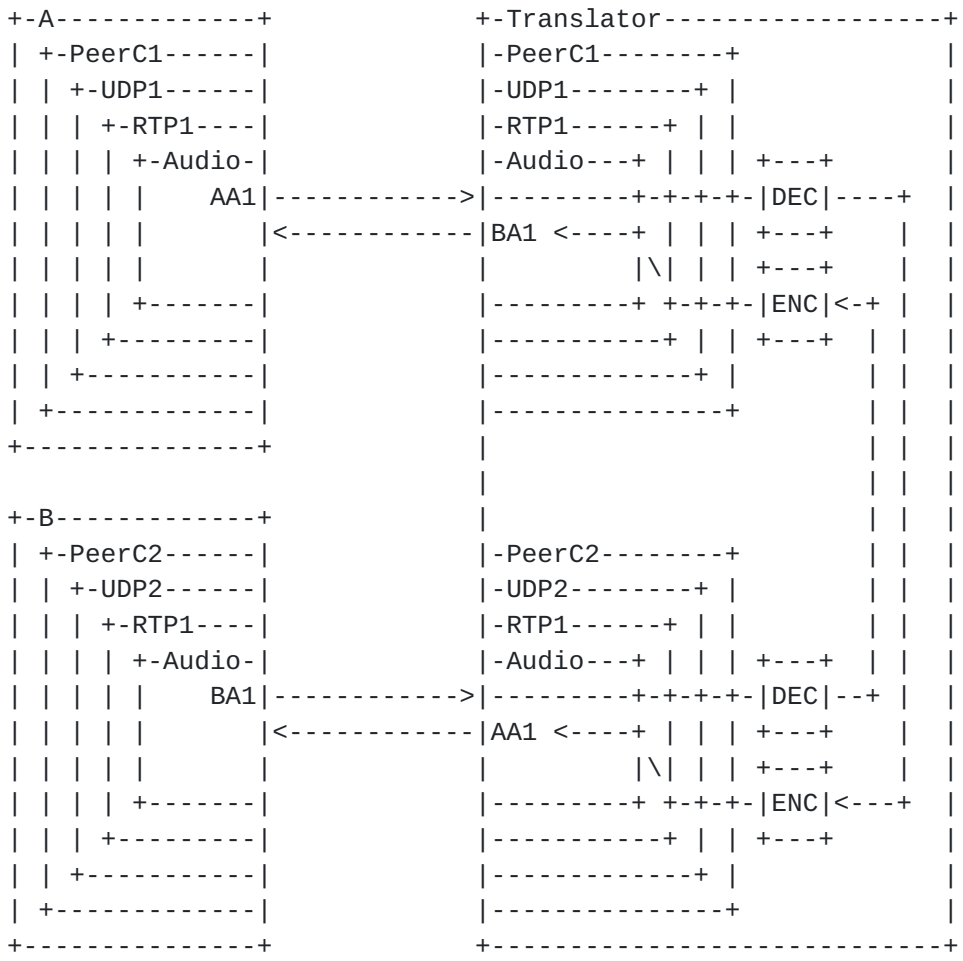


Figure 13: Media Transcoder

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

A.4.2. Gateway / Protocol Translator

Gateways are used when some protocol feature that are needed are not supported by an end-point wants to participate in session. This RTP

translator in Figure 14 takes on the role of ensuring that from the perspective of participant A, participant B appears as a fully compliant WebRTC end-point (that is, it is the combination of the Translator and participant B that looks like a WebRTC end point).

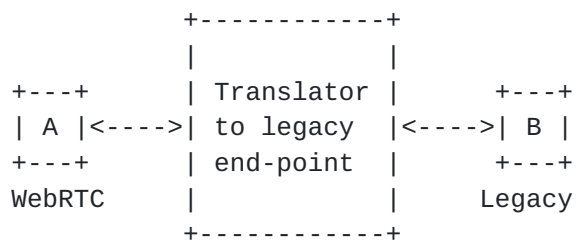
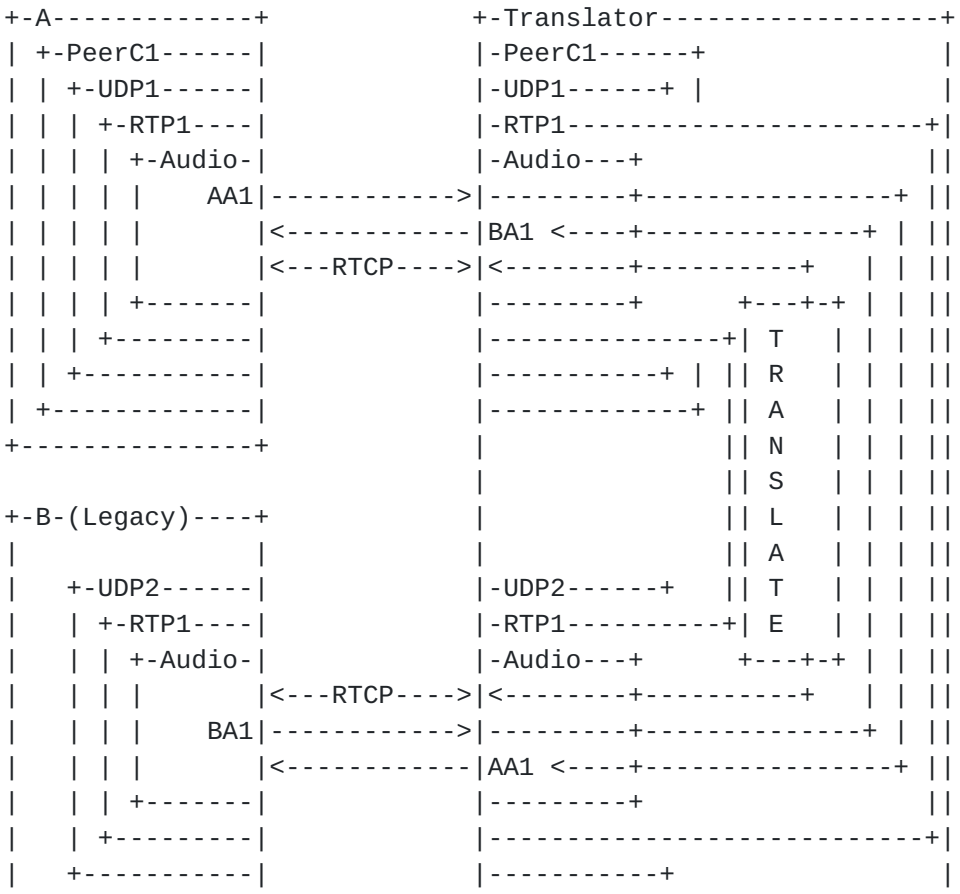


Figure 14: Gateway (RTP translator) towards legacy end-point

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



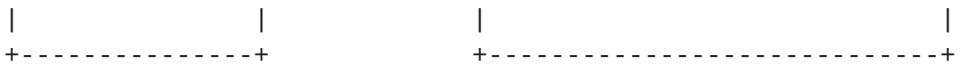


Figure 15: RTP/RTCP Protocol Translator

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

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

A.4.3. Relay

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

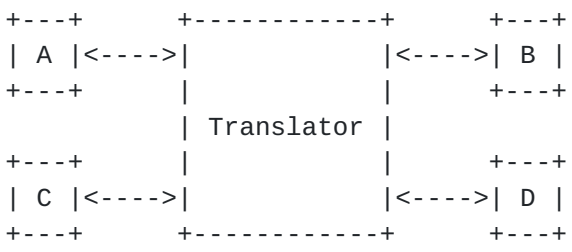
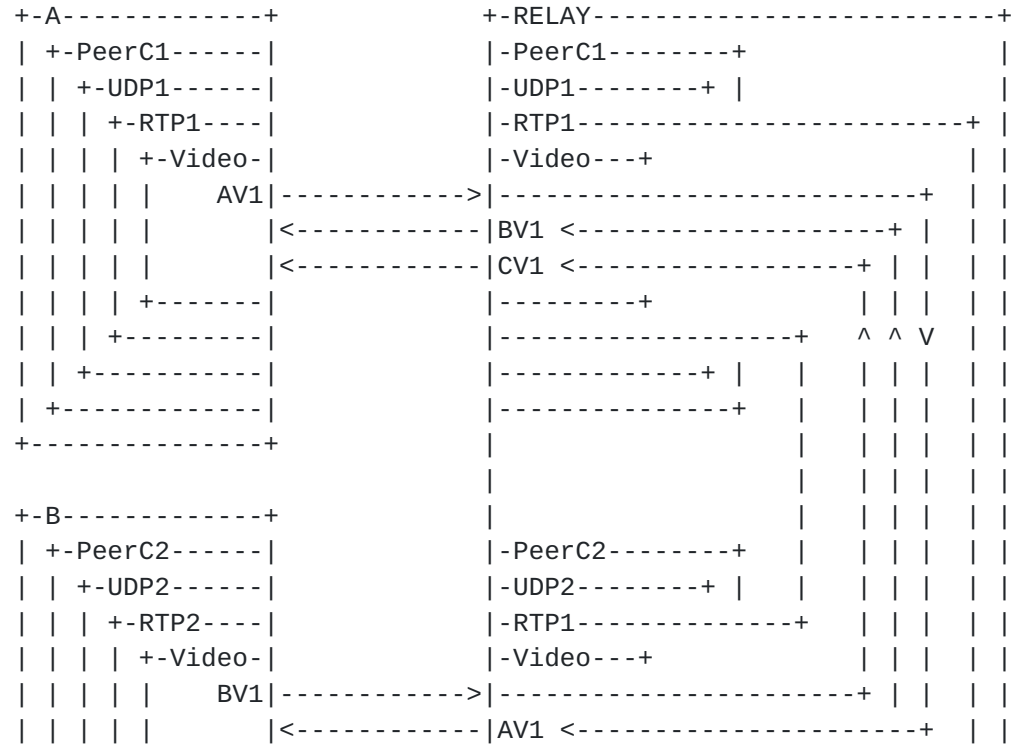


Figure 16: RTP Translator (Relay) with Only Unicast Paths

The first kind is straight forward and is likely to exist in WebRTC context when an legacy end-point is compatible with the exception for ICE, and thus needs a gateway that terminates the ICE and then forwards all the RTP/RTCP traffic and key management to the end-point only rewriting the IP/UDP to forward the packet to the legacy node.

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

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



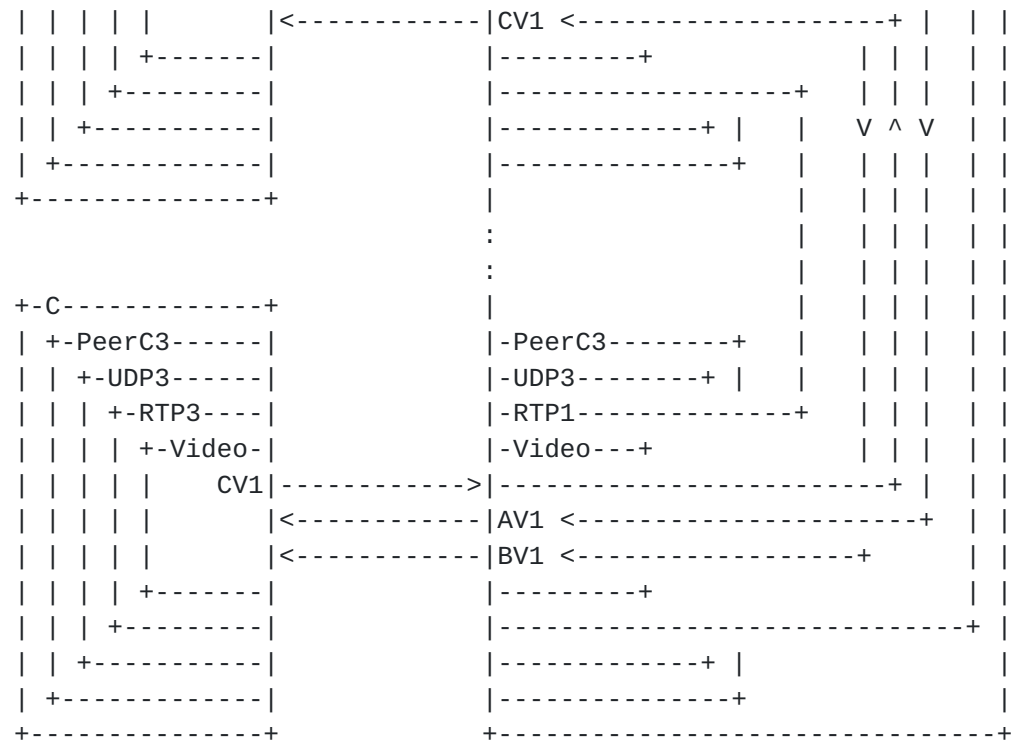


Figure 17: Transport Multi-party Relay

As the Relay RTP and RTCP packets between the UDP flows as indicated by the arrows for the media flow a given WebRTC end-point, like A will see the remote sources BV1 and CV1. There will be also two different network paths between A, and B or C. This results in that the client A has to be capable of handling that when determining congestion state that there might exist multiple destinations on the far side of a PeerConnection and that these paths have to be treated differently. It also results in a requirement to combine the different congestion states into a decision to transmit a particular RTP media stream suitable to all participants.

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

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

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

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

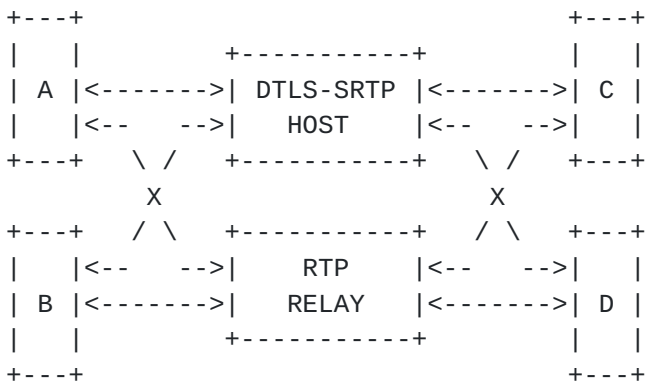


Figure 18: DTLS-SRTP host and RTP Relay Separated

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

A.5. End-point Forwarding

An WebRTC end-point (B in Figure 19) will receive a WebRTC MediaStream (set of SSRCs) over a PeerConnection (from A). For the

moment is not decided if the end-point is allowed or not to in its turn send that WebRTC MediaStream over another PeerConnection to C. This section discusses the RTP and end-point implications of allowing such functionality, which on the API level is extremely simplistic to perform.

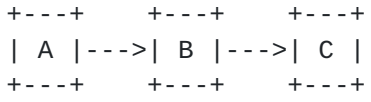


Figure 19: MediaStream Forwarding

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

A relay approach will result in that the WebRTC end-points will have to have the same capabilities as being discussed in Relay (Appendix A.4.3). Thus A will see an RTP session that is extended beyond the PeerConnection and see two different receiving end-points with different path characteristics (B and C). Thus A's congestion control needs to be capable of handling this. The security solution can either support mechanism that allows A to inform C about the key A is using despite B and C having agreed on another set of keys. Alternatively B will decrypt and then re-encrypt using a new key. The relay based approach has the advantage that B does not need to transcode the media thus both maintaining the quality of the encoding and reducing B's complexity requirements. If the right security solutions are supported then also C will be able to verify the authenticity of the media coming from A. As downside A are forced to take both B and C into consideration when delivering content.

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

A.6. Simulcast

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

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

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

Authors' Addresses

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

Email: csp@csperkins.org

Magnus Westerlund
Ericsson
Farogatan 6
SE-164 80 Kista
Sweden

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

Joerg Ott
Aalto University
School of Electrical Engineering
Espoo 02150
Finland

Email: jorg.ott@aalto.fi