

Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: January 17, 2013

C. Perkins
University of Glasgow
M. Westerlund
Ericsson
J. Ott
Aalto University
July 16, 2012

Web Real-Time Communication (WebRTC): Media Transport and Use of RTP
draft-ietf-rtcweb-rtp-usage-04

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 17, 2013.

Copyright Notice

Copyright (c) 2012 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	4
2. Rationale	4
3. Terminology	5
4. WebRTC Use of RTP: Core Protocols	6
4.1. RTP and RTCP	6
4.2. Choice of the RTP Profile	7
4.3. Choice of RTP Payload Formats	8
4.4. RTP Session Multiplexing	9
4.5. RTP and RTCP Multiplexing	10
4.6. Reduced Size RTCP	10
4.7. Symmetric RTP/RTCP	11
4.8. Choice of RTP Synchronisation Source (SSRC)	11
4.9. Generation of the RTCP Canonical Name (CNAME)	11
5. WebRTC Use of RTP: Extensions	12
5.1. Conferencing Extensions	12
5.1.1. Full Intra Request (FIR)	13
5.1.2. Picture Loss Indication (PLI)	13
5.1.3. Slice Loss Indication (SLI)	13
5.1.4. Reference Picture Selection Indication (RPSI)	14
5.1.5. Temporal-Spatial Trade-off Request (TSTR)	14
5.1.6. Temporary Maximum Media Stream Bit Rate Request	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. Congestion Control Requirements	18
7.2. Rate Control Boundary Conditions	19
7.3. RTCP Limitations for Congestion Control	19
7.4. Congestion Control Interoperability With Legacy Systems	20
8. WebRTC Use of RTP: Performance Monitoring	20
9. WebRTC Use of RTP: Future Extensions	21
10. Signalling Considerations	21
11. WebRTC API Considerations	22
11.1. API MediaStream to RTP Mapping	22
12. RTP Implementation Considerations	23

12.1. RTP Sessions and PeerConnection	23
12.2. Multiple Sources	25
12.3. Multiparty	25
12.4. SSRC Collision Detection	26
12.5. Contributing Sources	27
12.6. Media Synchronization	28
12.7. Multiple RTP End-points	28
12.8. Simulcast	29
12.9. Differentiated Treatment of Flows	29
13. IANA Considerations	31
14. Security Considerations	31
15. Acknowledgements	31
16. References	32
16.1. Normative References	32
16.2. Informative References	34
Appendix A. Supported RTP Topologies	36
A.1. Point to Point	36
A.2. Multi-Unicast (Mesh)	39
A.3. Mixer Based	42
A.3.1. Media Mixing	42
A.3.2. Media Switching	45
A.3.3. Media Projecting	48
A.4. Translator Based	51
A.4.1. Transcoder	51
A.4.2. Gateway / Protocol Translator	52
A.4.3. Relay	54
A.5. End-point Forwarding	58
A.6. Simulcast	59
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 endpoints, 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 requirement terminology. Requirements for core RTP protocols are described in Section 4 and recommended RTP extensions are described in Section 5. Section 6 outlines mechanisms that can increase robustness to network problems, while Section 7 describes the required 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 15 years development of RTP to mandate the use of extensions that have shown widespread utility, while still remaining compatible with the wide installed base of RTP implementations where possible.

RTP and RTCP extensions not discussed in this document can still be implemented by a WebRTC end-point, but they are considered optional, are not required for interoperability, and do not provide features needed to address the WebRTC use cases and requirements [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

This memo specifies various requirements levels for implementation or use of RTP features and extensions. When we describe the importance of RTP extensions, or the need for implementation support, we use the following requirement levels to specify the importance of the feature in the WebRTC framework:

MUST: This word, or the terms "REQUIRED" or "SHALL", mean that the definition is an absolute requirement of the specification.

SHOULD: This word, or the adjective "RECOMMENDED", mean that there may exist valid reasons in particular circumstances to ignore a particular item, but the full implications must be understood and carefully weighed before choosing a different course.

MAY: This word, or the adjective "OPTIONAL", mean that an item is truly optional. One vendor may choose to include the item because a particular marketplace requires it or because the vendor feels that it enhances the product while another vendor may omit the same item. An implementation which does not include a particular option **MUST** be prepared to interoperate with another implementation which does include the option, though perhaps with reduced functionality. In the same vein an implementation which

does include a particular option MUST be prepared to interoperate with another implementation which does not include the option (except, of course, for the feature the option provides.)

These key words are used in a manner consistent with their definition in [RFC2119]. The above 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:

- o Support for use of multiple simultaneous SSRC values in a single RTP session, including support for RTP end-points that send many SSRC values simultaneously.
- o Random choice of SSRC on joining a session; collision detection and resolution for SSRC values (but see also Section 4.8).
- o Support for reception of RTP data packets containing CSRC lists, as generated by RTP mixers, and RTCP packets relating to CSRCs.
- o Support for sending correct synchronization information in the RTCP Sender Reports, to allow a receiver to implement lip-sync, with RECOMMENDED support for the rapid RTP synchronisation extensions (see Section 5.2.1).
- o Support for sending and receiving RTCP SR, RR, SDES, and BYE packet types, with OPTIONAL support for other RTCP packet types; implementations MUST ignore unknown RTCP packet types.
- o Support for multiple end-points in a single RTP session, and for scaling the RTCP transmission interval according to the number of participants in the session; support for randomised RTCP transmission intervals to avoid synchronisation of RTCP reports; support for RTCP timer reconsideration.
- o Support for configuring the RTCP bandwidth as a fraction of the media bandwidth, and for configuring the fraction of the RTCP bandwidth allocated to senders, e.g., using the SDP "b=" line.

It is known that a significant number of legacy RTP implementations, especially those targeted at VoIP-only systems, do not support all of the above features, and in some cases do not support RTCP at all. Implementers are advised to consider the requirements for graceful degradation when interoperating with legacy implementations.

Other implementation considerations are discussed in Section 12.

4.2. Choice of the RTP Profile

The complete specification of RTP for a particular application domain requires the choice of an RTP Profile. For WebRTC use, the "Extended

Secure RTP Profile for Real-time Transport Control Protocol (RTCP)-Based Feedback (RTP/SAVPF) [RFC5124] is REQUIRED to 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 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 is needed to provide SRTP 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 requirement from Section 6 of [RFC3551] that "Audio applications operating under this profile SHOULD, at a minimum, be able to send and/or receive payload types 0 (PCMU) and 5 (DVI4)" applies, since Section 4.2 of this memo mandates the use of the RTP/SAVPF profile, which inherits this restriction from the RTP/AVP profile.

(tbd: there is ongoing discussion on whether support for other audio and video codecs is to be mandated)

Endpoints MAY signal support for multiple media formats, or multiple

configurations of a single format, provided each uses a different RTP payload type number. An endpoint that has signalled its support for multiple formats is REQUIRED to accept data in any of those formats at any time, unless it has previously signalled limitations on its decoding capability.

This requirement is constrained if several media types 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 media type that is being sent by that source; see Section 4.4. To support rapid rate adaptation, RTP does not require signalling in advance for changes between payload formats that were signalled during session setup.

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. RTP Session Multiplexing

An association amongst a set of participants communicating with RTP is known as an RTP session. A participant can be involved in multiple RTP sessions at the same time. In a multimedia session, each medium has typically been carried in a separate RTP session with its own RTCP packets (i.e., one RTP session for the audio, with a separate RTP session using a different transport address for the video; if SDP is used, this corresponds to one RTP session for each "m=" line in the SDP). WebRTC implementations of RTP are REQUIRED to implement support for multimedia sessions in this way, for compatibility with legacy systems.

In today's networks, however, with the widespread use of Network Address/Port Translators (NAT/NAPT) and Firewalls (FW), it is desirable to reduce the number of transport addresses used by real-time media applications using RTP by combining multimedia traffic in a single RTP session. (Details of how this is to be done are tbd, but see [I-D.lennox-rtcweb-rtp-media-type-mux], [I-D.holmberg-mmusic-sdp-bundle-negotiation] and [I-D.westerlund-avtcore-multiplex-architecture].) Using a single RTP session also effects the possibility for differentiated treatment of media flows. This is further discussed in Section 12.9.

WebRTC implementations of RTP are REQUIRED to support multiplexing of

a multimedia session onto a single RTP session according to (tbd). If such RTP session multiplexing is to be used, this MUST be negotiated during the signalling phase. Support for multiple RTP sessions over a single UDP flow as defined by [I-D.westerlund-avtcore-transport-multiplexing] is RECOMMENDED/OPTIONAL.

(tbd: No consensus on the level of including support of Multiple RTP sessions over a single UDP flow.)

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 setup 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 sending of RTP and RTCP to separate destination ports.

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, these general 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 sending RTCP feedback packets as [RFC5506] non-compound packets is REQUIRED when signalled. For backwards compatibility, implementations are

also REQUIRED to support the use of compound RTCP feedback packets.

4.7. Symmetric RTP/RTCP

To ease traversal of NAT and firewall devices, implementations are REQUIRED to implement and use Symmetric RTP [RFC4961]. This requires that the IP address and port used for sending and receiving RTP and RTCP packets are identical. 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 SSRC values, using the "a=ssrc:" SDP attribute defined in Sections 4.1 and 5 of [RFC5576], and MUST also support the "previous-ssrc" source attribute defined in Section 6.2 of [RFC5576]. Other attributes defined in [RFC5576] MAY be supported.

Use of the "a=ssrc:" attribute is OPTIONAL. Implementations MUST support random SSRC assignment, and MUST support SSRC collision detection and resolution, both according to [RFC3550].

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 method (b) specified in Section 4.2 of "Guidelines for Choosing RTP Control Protocol (RTCP) Canonical Names (CNAMEs)" [RFC6222] is RECOMMENDED. Note, however, that this does not resolve the privacy concern as there is not sufficient randomness to avoid tracking of an end-point.

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 according to the recommended form above.

(tbd: there seems to be a growing consensus that the working group wants randomly-chosen CNAME values; need to reference a draft that describes how this is to be done)

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 was popular in early deployments, in today's practice, overlay-based conferencing dominates, typically using one or more central servers to connect endpoints 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 [RFC5117].

As discussed in Section 3.5 of [RFC5117], the use of a video switching MCU makes the use of RTCP for congestion control, or any type of quality reports, very problematic. Also, as discussed in section 3.6 of [RFC5117], the use of a content modifying MCU with RTCP termination breaks RTP loop detection and removes the ability for receivers to identify active senders. 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 below to be used with centralised

conferencing -- where one RTP Mixer (e.g., a conference bridge) receives a participant's RTP media streams and distributes them to the other participants -- are not necessary for interoperability; an RTP endpoint that does not implement these extensions will work correctly, but may 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 packets assisting in such operation are defined in the 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. It is REQUIRED that this feedback message is supported by RTP senders in WebRTC, since it greatly improves the user experience when using centralised mixers-based conferencing.

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 there may be multiple methods to fulfill the request. It is REQUIRED that senders 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 macroblocks, and would like to have these repaired somehow. The use of 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

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 may have various reasons; for example, an RTP mixer may 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. It is REQUIRED that this feedback message is supported. A RTP media stream sender receiving a TMMBR for its SSRC MUST follow the limitations set by the message; 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. 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 some tools that can make RTP flows robust against Packet loss and reduce the impact on media quality. However, they all add extra bits compared to a non-robust stream. These extra bits need to be considered, and the aggregate bit-rate must be rate-controlled. Thus, 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 should 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). 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 may be 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 should be noted that using redundancy encoding or FEC will lead to increased playout delay, which should 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 very varied network environment with a heterogeneous set of link technologies, including wired and wireless, interconnecting peers at different topological locations resulting in network paths with widely varying one way delays, bit-rate capacity, load levels and traffic mixes. In addition, individual end-points will open one or more WebRTC sessions between one or more peers. Each of these session may contain different mixes of media and data flows. Asymmetric usage of media bit-rates and number of RTP media streams is also to be expected. A single end-point may receive zero to many simultaneous RTP media streams while itself transmitting one or more streams.

The WebRTC application is very dependent from a quality perspective on the media adaptation working well so that an end-point doesn't transmit significantly more than the path is capable of handling. If it would, the result would be high levels of packet loss or delay spikes causing media quality degradation.

WebRTC applications using more than a single RTP media stream of any media type or data flows have an additional concern. In this case, the different flows should try to avoid affecting each other negatively. In addition, in case there is a resource limitation, the available resources need to be shared. How to share them is something the application should prioritize so that the limitations in quality or capabilities are those that have the least impact on the application.

Overall, the diversity of operating environments lead to the need for functionality that adapts to the available capacity and that competes fairly with other network flows. If it would not compete fairly enough WebRTC could be used as an attack method for starving out other traffic on specific links as long as the attacker is able to create traffic across the links in question. A possible attack scenario is to use a web-service capable of attracting large numbers of end-points, combined with BGP routing state to have the server pick client pairs to drive traffic to specific paths.

The above clearly motivates the need for a well working media adaptation mechanism. This mechanism also have a number of requirements on what services it should provide and what performance it needs to provide.

The biggest issue is that there are no standardised and ready to use mechanism that can simply be included in WebRTC. Thus, there will be a need for the IETF to produce such a specification. Therefore, the suggested way forward is to specify requirements on any solution for the media adaptation. For now, we propose that these requirements be documented in this specification. In addition, a proposed detailed solution will be developed, but is expected to take longer time to finalize than this document.

7.1. Congestion Control Requirements

Requirements for congestion control of WebRTC sessions are discussed in [I-D.jesup-rtp-congestion-reqs].

Implementations are REQUIRED to implement the RTP circuit breakers described in [I-D.perkins-avtcore-rtp-circuit-breakers].

(tbd: Should add the RTP/RTCP Mechanisms that an WebRTC

implementation is required to support. Potential candidates include Transmission Timestamps (RFC 5450).)

7.2. Rate Control Boundary Conditions

The session establishment signalling will establish certain boundary that the media bit-rate adaptation can act within. First of all the set of media codecs provide practical limitations in the supported bit-rate span where it can provide useful quality, which packetization choices that exist. Next the signalling can establish maximum media bit-rate boundaries using SDP b=AS or b=CT.

(tbd: This section needs expanding on how to use these limits)

7.3. 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 may 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 required 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 may not be able to afford waiting for packet losses to occur to indicate congestion, because an increase in playout delay due to queuing (most prominent in wireless networks) may easily lead to packets being dropped due to late arrival at the receiver. Therefore, more sophisticated cues may 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.

7.4. Congestion Control Interoperability With 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 must 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 are required to interwork with such end-points MUST ensure that they keep within the RTP circuit breaker [I-D.perkins-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 are required to interwork with such end-points MUST ensure that they stay within the RTP circuit breaker [I-D.perkins-avtcore-rtp-circuit-breakers] constraints to limit the congestion they can cause, but may find that they can achieve better congestion response depending on the amount of feedback that is available.

8. WebRTC Use of RTP: Performance Monitoring

RTCP does contains a basic set of RTP flow monitoring metrics like packet loss and jitter. There are a number of extensions that could be included in the set to be supported. However, in most cases which RTP monitoring that is needed depends on the application, which makes it difficult to select which to include when the set of applications is very large.

Exposing some metrics in the WebRTC API should be considered allowing the application to gather the measurements of interest. However, security implications for the different data sets exposed will need to be considered in this.

(tbd: If any RTCP XR metrics should be added is still an open question, but possible to extend at a later stage)

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 should adopt RTP extensions that are of general utility, to enable easy gatewaying 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 of an external signalling channel that can be used to configure the 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 in addition to any cryptographic transformation of the packet content. As WebRTC requires the usage of the RTP/SAVPF profile this can be inferred as there is only a single profile, but in SDP this is still required information to be signalled. Interworking functions may transform this into RTP/SAVP for a legacy use case by indicating to the WebRTC end-point a RTP/SAVPF end-point 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 media 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).

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 may 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 following sections describe how the WebRTC API features map onto the RTP mechanisms described in this memo.

11.1. API MediaStream to RTP Mapping

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.

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 must be improved and achieved consensus on. Interim meeting in June 2012 shows large differences in opinions.)

12. RTP Implementation Considerations

The following provide some guidance on the implementation of the RTP features described in this memo.

This section discusses RTP functionality that is part of the RTP standard, required by decisions made, or to enable use cases raised and their motivations. This discussion is from an WebRTC end-point perspective. It will occasionally talk about central nodes, but as this specification is for an end-point, this is where the focus lies. For more discussion on the central nodes and details about RTP topologies please see Appendix A.

The section will touch on the relation with certain RTP/RTCP extensions, but will focus on the RTP core functionality. The definition of what functionalities and the level of requirement on implementing it is defined in Section 2.

12.1. RTP Sessions and PeerConnection

An RTP session is an association among RTP nodes, which have one common SSRC space. An RTP session can include any number of end-points and nodes sourcing, sinking, manipulating or reporting on the

RTP media streams being sent within the RTP session. A PeerConnection being a point-to-point association between an end-point and another node. That peer node may be both an end-point or centralized processing node of some type; thus, the RTP session may terminate immediately on the far end of the PeerConnection, but it may also continue as further discussed below in Multiparty (Section 12.3) and Multiple RTP End-points (Section 12.7).

A PeerConnection can contain one or more RTP session depending on how it is setup 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 different UDP flows. However, the default usage in WebRTC will be to use one RTP session for all media types. This usage then uses only one UDP flow, as also RTP and RTCP multiplexing is mandated (Section 4.5). However, for legacy interworking and network prioritization (Section 12.9) based on flows, a WebRTC end-point needs to support a mode of operation where one RTP session per media type is used. Currently, each RTP session must use its own UDP flow. Discussions are ongoing if a solution enabling multiple 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.

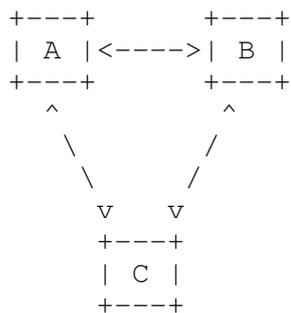


Figure 1: Multi-unicast

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 may 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 will be required 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 playout multiple RTP media streams of the same type concurrently.

tbd: Are any mechanism needed to signal limitations in the number of 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 may incur a heavy tax on the end-points' outgoing paths. It may 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 may both be dependent on transport path and the different end-points preferences related to playout of the media.

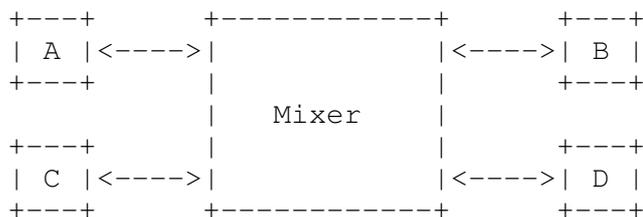


Figure 2: RTP Mixer with Only Unicast Paths

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 may 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 must 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 may 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 unsignalled SSRCs. 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.

12.5. Contributing Sources

Contributing Sources (CSRC) is a functionality in the RTP header that allows an RTP node to combine media packets from multiple sources into one and to identify which sources yielded the result. For WebRTC end-points, supporting contributing sources is trivial. The set of CSRCs is provided in a given RTP packet. This information can then be exposed to the applications using some form of API, possibly a mapping back into WebRTC MediaStream identities to avoid having to expose two namespaces and the handling of SSRC collision handling to the JavaScript.

(tbd: should the API provide the ability to add a CSRC list to an outgoing packet? this is only useful if the sender is mixing content)

There are also at least one extension that depends on the CSRC list being used: the Mixer-to-client audio level [RFC6465], which enhances the information provided by the CSRC to actual energy levels for audio for each contributing source.

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 is required 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 may 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 setup 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 fulfill 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 required. It should be noted that it might be a central node, rather than an WebRTC end-point that would benefit from receiving simulcasted 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.

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 which should 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 shall 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.

With the exception of diffserv both flow based and DPI have issues with running multiple media types and flows on a single UDP flow, especially when combined with data transport (SCTP/DTLS). DPI has issues because multiple types of flows are aggregated and thus it becomes more difficult to analyse them. The flow-based differentiation will provide the same treatment to all packets within the flow, i.e., relative prioritization is not possible. Moreover, if the resources are limited it may 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.

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.

tbd: The model for providing differentiated treatment needs to be

evolved. This includes:

1. How the application can prioritize `MediaStreamTracks` differently in the API
2. How the browser or application determine availability of transport differentiation
3. How to learn about any configuration information for transport differentiation, such as DSCPs.

13. IANA Considerations

This memo makes no request of IANA.

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

14. Security Considerations

RTP and its various extensions each have their own security considerations. These should be taken into account when considering the security properties of the complete suite. We currently don't think this suite creates any additional security issues or properties. The use of SRTP [RFC3711] will provide protection or mitigation against most of the fundamental issues by offering confidentiality, integrity and partial source authentication. A mandatory to implement media security solution will be required to be picked. We currently don't discuss the key-management aspect of SRTP in this memo, that needs to be done taking the WebRTC communication model into account.

Privacy concerns are under discussion and the generation of non-trackable CNAMEs are under discussion.

The guidelines in [RFC6562] apply when using variable bit rate (VBR) audio codecs, for example Opus or the Mixer audio level header extensions.

Security considerations for the WebRTC work are discussed in [I-D.ietf-rtcweb-security].

15. Acknowledgements

The authors would like to thank Harald Alvestrand, Cary Bran, Charles

Eckel and Cullen Jennings for valuable feedback.

16. References

16.1. Normative References

- [I-D.holmberg-mmusic-sdp-bundle-negotiation]
Holmberg, C. and H. Alvestrand, "Multiplexing Negotiation Using Session Description Protocol (SDP) Port Numbers", draft-holmberg-mmusic-sdp-bundle-negotiation-00 (work in progress), October 2011.
- [I-D.ietf-avtc core-srtp-encrypted-header-ext]
Lennox, J., "Encryption of Header Extensions in the Secure Real-Time Transport Protocol (SRTP)", draft-ietf-avtc core-srtp-encrypted-header-ext-01 (work in progress), October 2011.
- [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-05 (work in progress), May 2012.
- [I-D.ietf-rtcweb-overview]
Alvestrand, H., "Overview: Real Time Protocols for Browser-based Applications", draft-ietf-rtcweb-overview-04 (work in progress), June 2012.
- [I-D.ietf-rtcweb-security]
Rescorla, E., "Security Considerations for RTC-Web", draft-ietf-rtcweb-security-03 (work in progress), June 2012.
- [I-D.lennox-rtcweb-rtp-media-type-mux]
Rosenberg, J. and J. Lennox, "Multiplexing Multiple Media Types In a Single Real-Time Transport Protocol (RTP) Session", draft-lennox-rtcweb-rtp-media-type-mux-00 (work in progress), October 2011.
- [I-D.perkins-avtc core-rtp-circuit-breakers]
Perkins, C. and V. Singh, "RTP Congestion Control: Circuit Breakers for Unicast Sessions", draft-perkins-avtc core-rtp-circuit-breakers-00 (work in progress), March 2012.
- [I-D.westerlund-avtc core-transport-multiplexing]

Westerlund, M. and C. Perkins, "Multiple RTP Sessions on a Single Lower-Layer Transport", draft-westerlund-avtcore-transport-multiplexing-02 (work in progress), March 2012.

- [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.
- [RFC6222] Begen, A., Perkins, C., and D. Wing, "Guidelines for Choosing RTP Control Protocol (RTCP) Canonical Names (CNAMEs)", RFC 6222, April 2011.
- [RFC6464] Lennox, J., Iovov, E., and E. Marocco, "A Real-time Transport Protocol (RTP) Header Extension for Client-to-Mixer Audio Level Indication", RFC 6464, December 2011.
- [RFC6465] Iovov, E., Marocco, E., and J. Lennox, "A Real-time Transport Protocol (RTP) Header Extension for Mixer-to-Client Audio Level Indication", RFC 6465, December 2011.
- [RFC6562] Perkins, C. and JM. Valin, "Guidelines for the Use of Variable Bit Rate Audio with Secure RTP", RFC 6562, March 2012.

16.2. 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-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-09 (work in progress), June 2012.

[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., Burman, B., and C. Perkins, "RTP Multiplexing Architecture", draft-westerlund-avtcore-multiplex-architecture-01 (work in progress), March 2012.

[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.

[RFC5117] Westerlund, M. and S. Wenger, "RTP Topologies", RFC 5117, January 2008.

[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 [RFC5117], 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 differencies and the in many case small differences in implementation to support a larger set of possible topologies.

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.

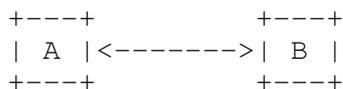


Figure 3: Point to Point

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 must be opened to allow RTP traffic. To reduce these costs and

session setup 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 may 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 are required.

Thus we can introduce a couple of different notiations in the below two alternate figures of a single peer connection in a a point to point setup. 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.

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 differnt peers, thus not forcing B to endure the same quality reductions if there are limiations in the transport from A to C as C will.

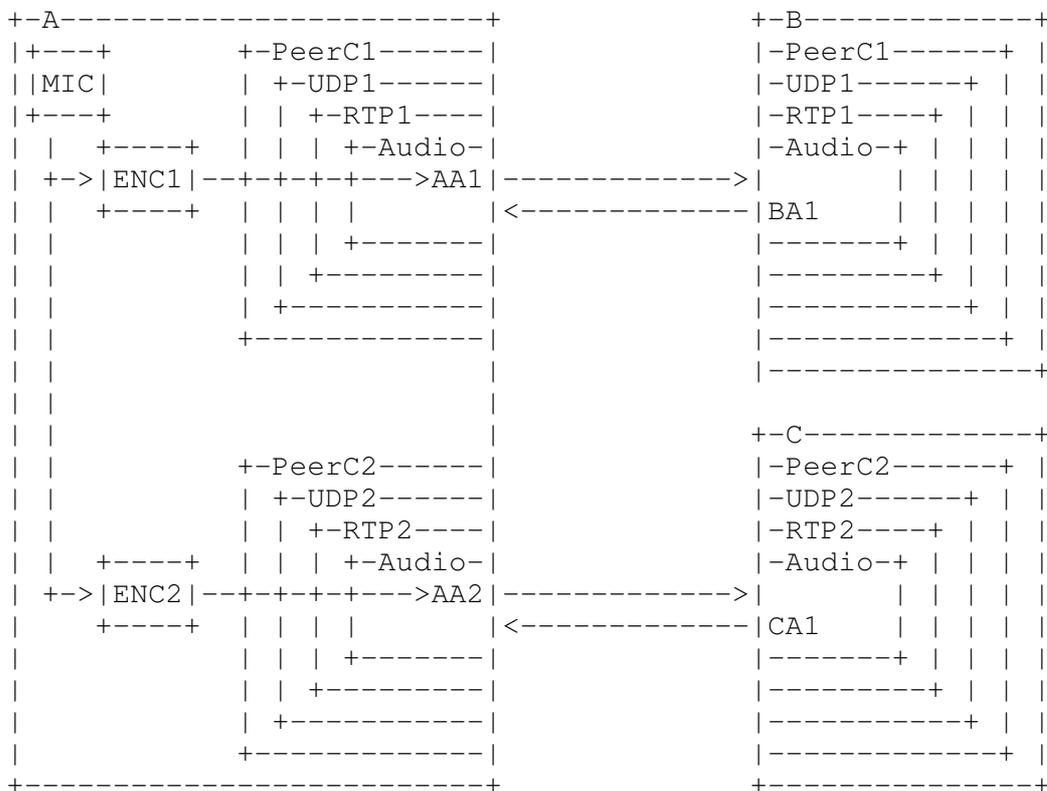


Figure 7: Session structure for Multi-Unicast Setup

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 setup 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 costrained will use a different implementation strategy where the encoder is shared between the different PeerConnections as shown below Figure 8.

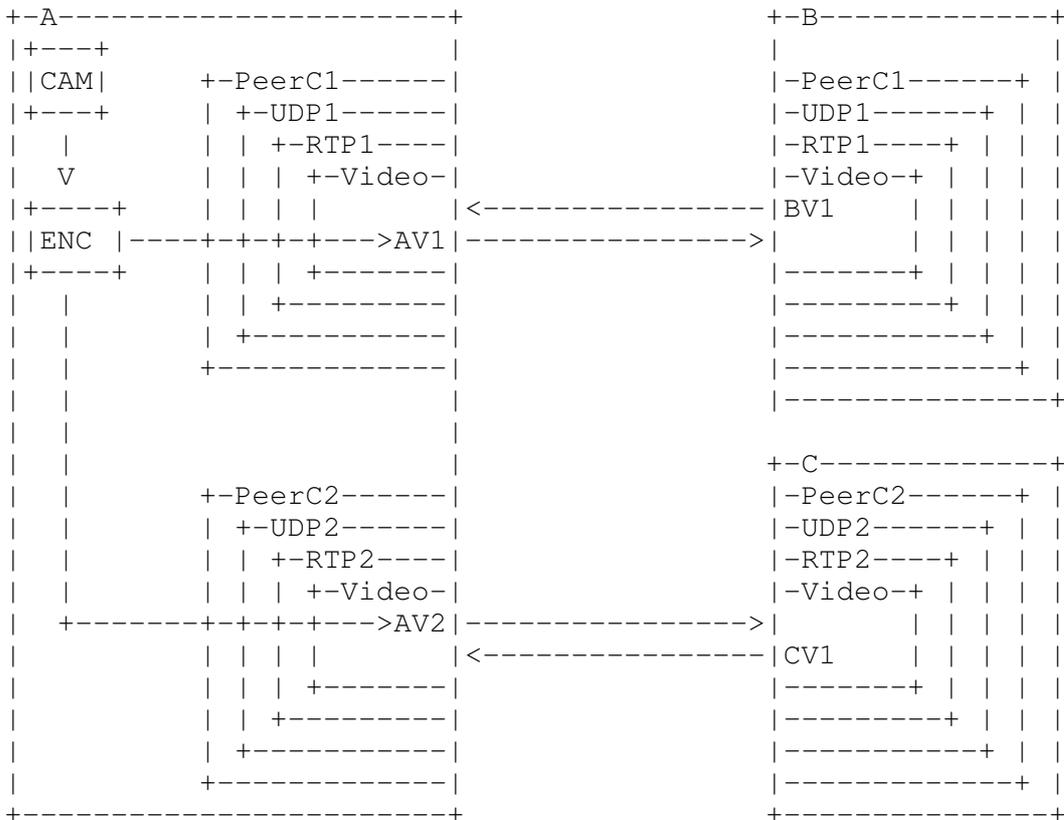


Figure 8: Single Encoder Multi-Unicast Setup

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 display area available is another aspect requiring consideration. The need for this type of descion 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 required in stead of 3 in the depicted scenario (Figure 9).



Figure 9: RTP Mixer with Only Unicast Paths

Mixers has two downsides, the first is that the mixer must be a trusted node as they either performs media operations or at least repacketize 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 integegrity 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 pattern 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. Let's assume that you want to mix N number of streams from different participants. Then the mixer needs to perform N decodings. 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 doesn't contain themselves, as this would result in an echo. When N is lower than the number of all participants one may produce a Mix of all N streams for the group that are currently 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 particularly useful for the users, however creating a 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 needs 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 packetizes it and sends it out.

The problem with media mixing is that it both consumes large amounts 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.

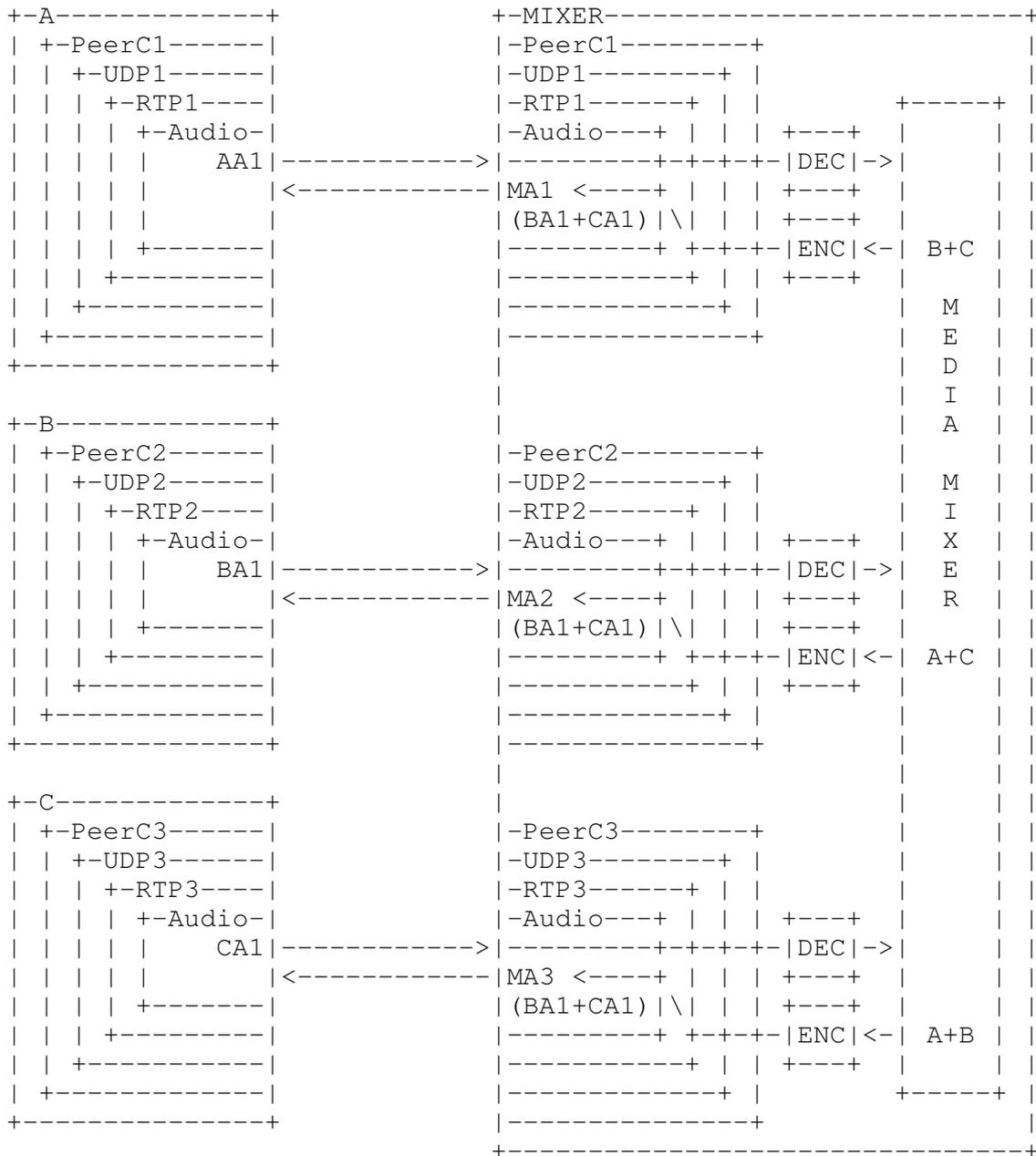


Figure 10: Session and SSRC details for Media Mixer

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 (SDS) 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 may 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 misreferencing. Also SSRC collision handling will still be needed as it may occur between the mixer and the end-point.

Session termination may 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 achieved 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 required 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 must be set to the value of the Mixer's SSRC. Secondly, the sequence number must be 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 may 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 required.

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.

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 transmitt 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 consequitvely 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 indepdentently also thus working around any SSRC collisions by having remapping tables between the RTP sessions. However the related WebRTC MediaStream signalling must 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 comming 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 should at all be 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 must 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 should 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.

The legacy gateway may 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 RTP 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 may need the capabilities 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 required 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 flavors, 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 provide a group communication feature as depicted below in Figure 16.

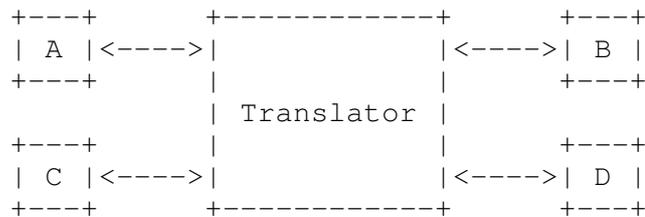


Figure 16: RTP Translator (Relay) with Only Unicast Paths

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 keymanagement 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.

different network paths between A, and B or C. This results in that the client A must 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 shall 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 must 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 must be agreed such so that the relay doesn't get it, e.g. no DTLS-SRTP handshake with the relay, instead some other method must be used. Secondly, the keying structure must 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 protect all RTP/RTCP packets it sends. Each participant's 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.

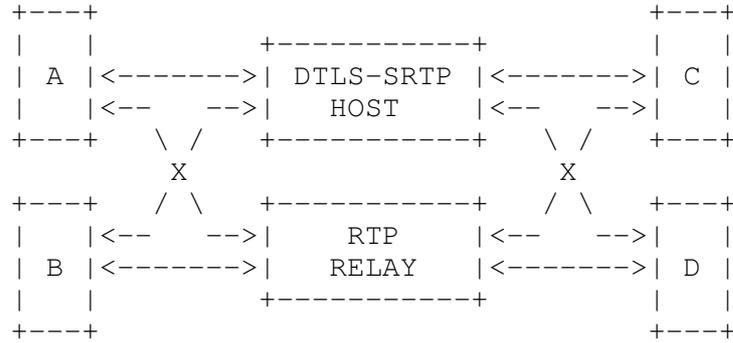


Figure 18: DTLS-SRTP host and RTP Relay Separated

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.

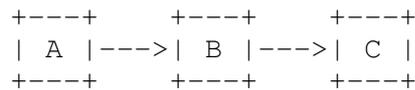


Figure 19: MediaStream Forwarding

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 may 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 must 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 should 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@csp Perkins.org

Magnus Westerlund
Ericsson
Farogatan 6
SE-164 80 Kista
Sweden

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

Joerg Ott
Aalto University
School of Electrical Engineering
Espoo 02150
Finland

Email: jorg.ott@aalto.fi

