    Web Real-Time Communication (WebRTC): Media Transport and Use of RTP
                    draft-ietf-rtcweb-rtp-usage-11

Abstract

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

Status of This Memo

Copyright Notice

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

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

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

   This memo specifies a protocol intended for use within the WebRTC
   framework, but is not restricted to that context. An overview of the
   WebRTC framework is given in [I-D.ietf-rtcweb-overview].

   The structure of this memo is as follows. Section 2 outlines our
   rationale in preparing this memo and choosing these RTP features.
   Section 3 defines terminology. Requirements for core RTP protocols
   are described in Section 4 and suggested RTP extensions are described
   in Section 5. Section 6 outlines mechanisms that can increase
   robustness to network problems, while Section 7 describes congestion
   control and rate adaptation mechanisms. The discussion of mandated
   RTP mechanisms concludes in Section 8 with a review of performance
   monitoring and network management tools 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. The memo concludes with security considerations
(Section 13) and IANA considerations (Section 14).

2. Rationale

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

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

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

3. Terminology

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

We define the following terms:

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

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

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

Other terms are used according to their definitions from the RTP
Specification [RFC3550].

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, following [RFC3550] and
   [I-D.ietf-avtcore-rtp-multi-stream]. Support for the RTCP
   optimisations for multi-SSRC sessions defined in
   [I-D.ietf-avtcore-rtp-multi-stream-optimisation] is RECOMMENDED.

   o Random choice of SSRC on joining a session; collision detection
      and resolution for SSRC values (see also Section 4.8).

   o Support for reception of RTP data packets containing CSRC lists,
      as generated by RTP mixers, and RTCP packets relating to CSRCs.

   o Sending correct synchronisation information in the RTCP Sender
      Reports, to allow receivers to implement lip-sync, with support
      for the rapid RTP synchronisation extensions (see Section 5.2.1)
      being RECOMMENDED.

   o Support for multiple synchronisation contexts. Participants that
      send multiple simultaneous RTP media streams MAY do so as part of
      a single synchronisation context, using a single RTCP CNAME for
      all streams and allowing receivers to play the streams out in a
      synchronised manner, or they MAY use different synchronisation
      contexts, and hence different RTCP CNAMEs, for some or all of the
      streams. Receivers MUST support reception of multiple RTCP CNAMEs
      from each participant in an RTP session. See also Section 4.9.

   o Support for sending and receiving RTCP SR, RR, SDES, and BYE
      packet types, with OPTIONAL support for other RTCP packet types;
      implementations MUST ignore unknown RTCP packet types. Note that
      additional RTCP Packet types are needed by the RTP/SAVPF Profile
      (Section 4.2) and the other RTCP extensions (Section 5).

   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 RTCP-Based Feedback (RTP/SAVPF) [RFC5124], as
extended by [RFC7007], MUST be implemented. This builds on the basic
RTP/AVP profile [RFC3551], the RTP profile for RTCP-based feedback
(RTP/AVPF) [RFC4585], and the secure RTP profile (RTP/SAVP)
[RFC3711].

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

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

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

The keying mechanism(s) to be used with the RTP/SAVPF profile are
defined in Section 5.5 of [I-D.ietf-rtcweb-security-arch] or its
replacement.

4.3. Choice of RTP Payload Formats

The set of mandatory to implement codecs and RTP payload formats for
WebRTC is not specified in this memo. Implementations can support
any codec for which an RTP payload format and associated signalling
is defined. Implementation cannot assume that the other participants
in an RTP session understand any RTP payload format, no matter how
common; the mapping between RTP payload type numbers and specific
configurations of particular RTP payload formats MUST be agreed
before those payload types/formats can be used. In an SDP context,
this can be done using the "a=rtpmap:" and "a=fmtp:" attributes
associated with an "m=" line.

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

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

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

4.4. Use of RTP Sessions

An association amongst a set of participants communicating using RTP
is known as an RTP session. A participant can be involved in several
RTP sessions at the same time. In a multimedia session, each type of
media has typically been carried in a separate RTP session (e.g.,
using one RTP session for the audio, and a separate RTP session using

different transport addresses for the video). WebRTC implementations
of RTP are REQUIRED to implement support for multimedia sessions in
this way, separating each session using different transport-layer
addresses (e.g., different UDP ports) for compatibility with legacy
systems.

In modern day networks, however, with the widespread use of network
address/port translators (NAT/NAPT) and firewalls, it is desirable to
reduce the number of transport-layer flows used by RTP applications.
This can be done by sending all the RTP media streams in a single RTP
session, which will comprise a single transport-layer flow (this will
prevent the use of some quality-of-service mechanisms, as discussed
in Section 12.1.3). Implementations are REQUIRED to support
transport of all RTP media streams, independent of media type, in a
single RTP session according to
[I-D.ietf-avtcore-multi-media-rtp-session]. If multiple types of
media are to be used in a single RTP session, all participants in
that session MUST agree to this usage. In an SDP context,
[I-D.ietf-mmusic-sdp-bundle-negotiation] can be used to signal this.

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

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

4.5. RTP and RTCP Multiplexing

Historically, RTP and RTCP have been run on separate transport layer
addresses (e.g., two UDP ports for each RTP session, one port for RTP
and one port for RTCP). With the increased use of Network Address/
Port Translation (NAPT) this has become problematic, since
maintaining multiple NAT bindings can be costly. It also complicates
firewall administration, since multiple ports need to be opened to
allow RTP traffic. To reduce these costs and session set-up times,
support for multiplexing RTP data packets and RTCP control packets on
a single port for each RTP session is REQUIRED, as specified in
[RFC5761]. For backwards compatibility, implementations are also
REQUIRED to support RTP and RTCP sent on separate transport-layer
addresses.

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

4.6. Reduced Size RTCP

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

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

4.7. Symmetric RTP/RTCP

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

4.8. Choice of RTP Synchronisation Source (SSRC)

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

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

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

4.9. Generation of the RTCP Canonical Name (CNAME)

The RTCP Canonical Name (CNAME) provides a persistent transport-level
identifier for an RTP endpoint. While the Synchronisation Source
(SSRC) identifier for an RTP endpoint can change if a collision is
detected, or when the RTP application is restarted, its RTCP CNAME is
meant to stay unchanged, so that RTP endpoints can be uniquely
identified and associated with their RTP media streams within a set
of related RTP sessions. For proper functionality, each RTP endpoint
needs to have at least one unique RTCP CNAME value. An endpoint MAY
have multiple CNAMEs, as the CNAME also identifies a particular
synchronisation context, i.e. all SSRC associated with a CNAME share
a common reference clock, and if an endpoint have SSRCs associated
with different reference clocks it will need to use multiple CNAMEs.
This ought not be common, and if possible reference clocks ought to
be mapped to each other and one chosen to be used with RTP and RTCP.

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 [RFC7022] is
RECOMMENDED.

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

5. WebRTC Use of RTP: Extensions

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

5.1. Conferencing Extensions

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

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

o The use of video switching MCUs makes the use of RTCP for
   congestion control and quality of service reports problematic (see
   Section 3.6.2 of [I-D.ietf-avtcore-rtp-topologies-update]).

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

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

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

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

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

5.1.1. Full Intra Request (FIR)

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

5.1.2. Picture Loss Indication (PLI)

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

5.1.3. Slice Loss Indication (SLI)

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

5.1.4. Reference Picture Selection Indication (RPSI)

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

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

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

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

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

5.2. Header Extensions

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

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

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

5.2.1. Rapid Synchronisation

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

5.2.2. Client-to-Mixer Audio Level

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

The Client-to-Mixer Audio Level [RFC6464] extension is RECOMMENDED to
be implemented. If it is implemented, it is REQUIRED that the header
extensions are encrypted according to [RFC6904] 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 [RFC6904] since the information
contained in these header extensions can be considered sensitive.

5.2.4. Associating RTP Media Streams and Signalling Contexts

(tbd: it seems likely that we need a mechanism to associate RTP media
streams with signalling contexts. The mechanism by which this is
done will likely be some combination of an RTP header extension,
periodic transmission of a new RTCP SDES item, and some signalling
extension. The semantics of those items are not yet settled; see
draft-westerlund-avtext-rtcp-sdes-srcname, draft-ietf-mmusic-msid,
and draft-even-mmusic-application-token for discussion).

6. WebRTC Use of RTP: Improving Transport Robustness

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

6.1. Negative Acknowledgements and RTP Retransmission

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

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

   The RTP Retransmission Payload Format [RFC4588] offers the ability to
   retransmit lost packets based on NACK feedback. Retransmission needs
   to be used with care in interactive real-time applications to ensure
   that the retransmitted packet arrives in time to be useful, but can
   be effective in environments with relatively low network RTT (an RTP
   sender can estimate the RTT to the receivers using the information in
   RTCP SR and RR packets, as described at the end of Section 6.4.1 of
   [RFC3550]). The use of retransmissions can also increase the forward

RTP bandwidth, and can potentially worsen the problem if the packet
loss was caused by network congestion. We note, however, that
retransmission of an important lost packet to repair decoder state
can have lower cost than sending a full intra frame. It is not
appropriate to blindly retransmit RTP packets in response to a NACK.
The importance of lost packets and the likelihood of them arriving in
time to be useful needs to be considered before RTP retransmission is
used.

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

6.2. Forward Error Correction (FEC)

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

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

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

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

WebRTC will be used in heterogeneous network environments using a
variety set of link technologies, including both wired and wireless
links, to interconnect potentially large groups of users around the
world. As a result, the network paths between users can have widely
varying one-way delays, available bit-rates, load levels, and traffic
mixtures. Individual end-points can send one or more RTP media

streams to each participant in a WebRTC conference, and there can be
several participants. Each of these RTP media streams can contain
different types of media, and the type of media, bit rate, and number
of flows can be highly asymmetric. Non-RTP traffic can share the
network paths with RTP flows. Since the network environment is not
predictable or stable, WebRTC endpoints MUST ensure that the RTP
traffic they generate can adapt to match changes in the available
network capacity.

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

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

7.1. Boundary Conditions and Circuit Breakers

   In the absence of a concrete congestion control algorithm, all WebRTC
   implementations MUST implement the RTP circuit breaker algorithm that
   is in described [I-D.ietf-avtcore-rtp-circuit-breakers]. The RTP
   circuit breaker is designed to enable applications to recognise and
   react to situations of extreme network congestion. However, since
   the RTP circuit breaker might not be triggered until congestion
   becomes extreme, it cannot be considered a substitute for congestion
   control, and applications MUST also implement congestion control to
   allow them to adapt to changes in network capacity. Any future RTP
   congestion control algorithms are expected to operate within the
   envelope allowed by the circuit breaker.

   The session establishment signalling will also necessarily establish
   boundaries to which the media bit-rate will conform. The choice of
   media codecs provides upper- and lower-bounds on the supported bit-
   rates that the application can utilise to provide useful quality, and
   the packetization choices that exist. In addition, the signalling

channel can establish maximum media bit-rate boundaries using the SDP
"b=AS:" or "b=CT:" lines, and the RTP/AVPF Temporary Maximum Media
Stream Bit Rate (TMMBR) Requests (see Section 5.1.6 of this memo).
The combination of media codec choice and signalled bandwidth limits
SHOULD be used to limit traffic based on known bandwidth limitations,
for example the capacity of the edge links, to the extent possible.

7.2. RTCP Limitations for Congestion Control

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

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

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

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

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

7.3. Congestion Control Interoperability and Legacy Systems

There are legacy implementations that do not implement RTCP, and
hence do not provide any congestion feedback. Congestion control

cannot be performed with these end-points. WebRTC implementations
that need to interwork with such end-points MUST limit their
transmission to a low rate, equivalent to a VoIP call using a low
bandwidth codec, that is unlikely to cause any significant
congestion.

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

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

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

8. WebRTC Use of RTP: Performance Monitoring

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

A large number of additional performance metrics are supported by the
RTCP Extended Reports (XR) framework [RFC3611][RFC6792]. It is not
yet clear what extended metrics are appropriate for use in the WebRTC
context, so there is no requirement that implementations generate
RTCP XR packets. However, implementations that can use detailed
performance monitoring data MAY generate RTCP XR packets as
appropriate; the use of such packets SHOULD be signalled in advance.

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

9. WebRTC Use of RTP: Future Extensions

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

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

10. Signalling Considerations

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

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

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

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

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

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

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

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

11. WebRTC API Considerations

   The WebRTC API [W3C.WD-webrtc-20130910] and the Media Capture and
   Streams API [W3C.WD-mediacapture-streams-20130903] defines and uses
   the concept of a MediaStream that consists of zero or more
   MediaStreamTracks. A MediaStreamTrack is an individual stream of
   media from any type of media source like a microphone or a camera,
   but also conceptual sources, like a audio mix or a video composition,
   are possible. The MediaStreamTracks within a MediaStream need to be
   possible to play out synchronised.

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

      Note: It is quite likely that a simulcast specification will
      result in multiple source packet streams, and thus SSRCs, based on
      the same source stream associated with the MediaStreamTrack being
      simulcasted. Each such source packet stream can have dependent
      and redundant packet streams associated with them. However, the
      final conclusion on this awaits the specification of simulcast.
      Simulcast will also require signalling to correctly separate and
      associate the source packet streams with their sets of dependent
      and/or redundant streams.

   It is important to note that the same media source can be feeding
   multiple MediaStreamTracks. As different sets of constraints or
   other parameters can be applied to the MediaStreamTrack, each
   MediaStreamTrack instance added to a RTCPeerConnection SHALL result
   in an independent source packet stream, with its own set of
   associated packet streams, and thus different SSRC(s). It will
   depend on applied constraints and parameters if the source stream and
   the encoding configuration will be identical between different
   MediaStreamTracks sharing the same media source. Thus it is possible
   for multiple source packet streams to share encoded streams (but not
   packet streams), but this is an implementation choice to try to
   utilise such optimisations. Note that such optimizations would need
   to take into account that the constraints for one of the
   MediaStreamTracks can at any moment change, meaning that the encoding
   configurations should no longer be identical.

The same MediaStreamTrack can also be included in multiple
MediaStreams, thus multiple sets of MediaStreams can implicitly need
to use the same synchronisation base. To ensure that this works in
all cases, and don't forces a endpoint to change synchronisation base
and CNAME in the middle of a ongoing delivery of any packet streams,
which would cause media disruption; all MediaStreamTracks and their
associated SSRCs originating from the same endpoint MUST be sent
using the same CNAME within one RTCPeerConnection as well as across
all RTCPeerConnections part of the same communication session
context, which for a browser are a single origin.

   Note: It is important that the same CNAME is not used in different
   communication session contexts or origins, as that could enable
   tracking of a user and its device usage of different services.
   See Section 4.4.1 of Security Considerations for WebRTC
   [I-D.ietf-rtcweb-security] for further discussion.

   The reasons to require the same CNAME across multiple
   RTCPeerConnections is to enable synchronisation of different
   MediaStreamTracks originating from one endpoint despite them being
   transported over different RTCPeerConnections.

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

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

   Note: The motivation for supporting reception of multiple CNAMEs
   are to allow for forward compatibility with any future changes

that enables more efficient stream handling when endpoints relay/
forward streams. It also ensures that endpoints can interoperate
with certain types of multi-stream middleboxes or endpoints that
are not WebRTC.

The binding between the WebRTC MediaStreams, MediaStreamTracks and
the SSRC is done as specified in "Cross Session Stream Identification
in the Session Description Protocol" [I-D.ietf-mmusic-msid]. This
document [I-D.ietf-mmusic-msid] also defines, in section 4.1, how to
map unknown source packet stream SSRCs to MediaStreamTracks and
MediaStreams. Commonly the RTP Payload Type of any incoming packets
will reveal if the packet stream is a source stream or a redundancy
or dependent packet stream. The association to the correct source
packet stream depends on the payload format in use for the packet
stream.

12. RTP Implementation Considerations

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

12.1. Configuration and Use of RTP Sessions

   A WebRTC end-point will be a simultaneous participant in one or more
   RTP sessions. Each RTP session can convey multiple media flows, and
   can include media data from multiple end-points. In the following,
   we outline some ways in which WebRTC end-points can configure and use
   RTP sessions.

12.1.1. Use of Multiple Media Flows Within an RTP Session

   RTP is a group communication protocol, and in a WebRTC context every
   RTP session can potentially contain multiple media flows. There are
   several reasons why this might be desirable:

Multiple media types: Outside of WebRTC, it is common to use one RTP
    session for each type of media (e.g., one RTP session for audio
    and one for video, each sent on a different UDP port). However,
    to reduce the number of UDP ports used, the default in WebRTC is
    to send all types of media in a single RTP session, as described
    in Section 4.4, using RTP and RTCP multiplexing (Section 4.5) to
    further reduce the number of UDP ports needed. This RTP session
    then uses only one UDP flow, but will contain multiple RTP media
    streams, each containing a different type of media. A common
    example might be an end-point with a camera and microphone that
    sends two RTP streams, one video and one audio, into a single RTP
    session.

Multiple Capture Devices: A WebRTC end-point might have multiple
    cameras, microphones, or other media capture devices, and so might
    want to generate several RTP media streams of the same media type.
    Alternatively, it might want to send media from a single capture
    device in several different formats or quality settings at once.
    Both can result in a single end-point sending multiple RTP media
    streams of the same media type into a single RTP session at the
    same time.

Associated Repair Data: An end-point might send a media stream that
    is somehow associated with another stream. For example, it might
    send an RTP stream that contains FEC or retransmission data
    relating to another stream. Some RTP payload formats send this
    sort of associated repair data as part of the original media
    stream, while others send it as a separate stream.

Layered or Multiple Description Coding: An end-point can use a
    layered media codec, for example H.264 SVC, or a multiple
    description codec, that generates multiple media flows, each with
    a distinct RTP SSRC, within a single RTP session.

RTP Mixers, Translators, and Other Middleboxes: An RTP session, in
    the WebRTC context, is a point-to-point association between an
    end-point and some other peer device, where those devices share a
    common SSRC space. The peer device might be another WebRTC end-
    point, or it might be an RTP mixer, translator, or some other form
    of media processing middlebox. In the latter cases, the middlebox
    might send mixed or relayed RTP streams from several participants,
    that the WebRTC end-point will need to render. Thus, even though
    a WebRTC end-point might only be a member of a single RTP session,
    the peer device might be extending that RTP session to incorporate
    other end-points. WebRTC is a group communication environment and
    end-points need to be capable of receiving, decoding, and playing
    out multiple RTP media streams at once, even in a single RTP
    session.

12.1.2. Use of Multiple RTP Sessions

   In addition to sending and receiving multiple media streams within a
   single RTP session, a WebRTC end-point might participate in multiple
   RTP sessions. There are several reasons why a WebRTC end-point might
   choose to do this:

   To interoperate with legacy devices: The common practice in the non-
      WebRTC world is to send different types of media in separate RTP
      sessions, for example using one RTP session for audio and another
      RTP session, on a different UDP port, for video. All WebRTC end-
      points need to support the option of sending different types of
      media on different RTP sessions, so they can interwork with such
      legacy devices. This is discussed further in Section 4.4.

   To provide enhanced quality of service: Some network-based quality
      of service mechanisms operate on the granularity of UDP 5-tuples.
      If it is desired to use these mechanisms to provide differentiated
      quality of service for some RTP flows, then those RTP flows need
      to be sent in a separate RTP session using a different UDP port
      number, and with appropriate quality of service marking. This is
      discussed further in Section 12.1.3.

   To separate media with different purposes: An end-point might want
      to send media streams that have different purposes on different
      RTP sessions, to make it easy for the peer device to distinguish
      them. For example, some centralised multiparty conferencing
      systems display the active speaker in high resolution, but show
      low resolution "thumbnails" of other participants. Such systems
      might configure the end-points to send simulcast high- and low-
      resolution versions of their video using separate RTP sessions, to
      simplify the operation of the central mixer. In the WebRTC
      context this appears to be most easily accomplished by
      establishing multiple RTCPeerConnection all being feed the same
      set of WebRTC MediaStreams. Each RTCPeerConnection is then
      configured to deliver a particular media quality and thus media
      bit-rate, and will produce an independently encoded version with
      the codec parameters agreed specifically in the context of that
      RTCPeerConnection. The central mixer can always distinguish
      packets corresponding to the low- and high-resolution streams by
      inspecting their SSRC, RTP payload type, or some other information
      contained in RTP header extensions or RTCP packets, but it can be
      easier to distinguish the flows if they arrive on separate RTP
      sessions on separate UDP ports.

   To directly connect with multiple peers: A multi-party conference
      does not need to use a central mixer. Rather, a multi-unicast
      mesh can be created, comprising several distinct RTP sessions,

with each participant sending RTP traffic over a separate RTP
session (that is, using an independent RTCPeerConnection object)
to every other participant, as shown in Figure 1. This topology
has the benefit of not requiring a central mixer node that is
trusted to access and manipulate the media data. The downside is
that it increases the used bandwidth at each sender by requiring
one copy of the RTP media streams for each participant that are
part of the same session beyond the sender itself.


```
    +---+       +---+
    | A |<--->| B |
    +---+       +---+
      ^           ^
       \         /
        \       /
         v     v
        +---+
        | C |
        +---+
```
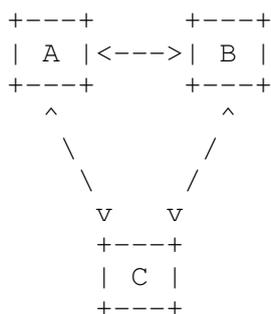

              Figure 1: Multi-unicast using several RTP sessions

   The multi-unicast topology could also be implemented as a single
   RTP session, spanning multiple peer-to-peer transport layer
   connections, or as several pairwise RTP sessions, one between each
   pair of peers. To maintain a coherent mapping between the
   relation between RTP sessions and RTCPeerConnection objects we
   recommend that this is implemented as several individual RTP
   sessions. The only downside is that end-point A will not learn of
   the quality of any transmission happening between B and C, since
   it will not see RTCP reports for the RTP session between B and C,
   whereas it would it all three participants were part of a single
   RTP session. Experience with the Mbone tools (experimental RTP-
   based multicast conferencing tools from the late 1990s) has showed
   that RTCP reception quality reports for third parties can usefully
   be presented to the users in a way that helps them understand
   asymmetric network problems, and the approach of using separate
   RTP sessions prevents this. However, an advantage of using
   separate RTP sessions is that it enables using different media
   bit-rates and RTP session configurations between the different
   peers, thus not forcing B to endure the same quality reductions if
   there are limitations in the transport from A to C as C will. It
   it believed that these advantages outweigh the limitations in
   debugging power.

To indirectly connect with multiple peers: A common scenario in
   multi-party conferencing is to create indirect connections to
   multiple peers, using an RTP mixer, translator, or some other type
   of RTP middlebox. Figure 2 outlines a simple topology that might
   be used in a four-person centralised conference. The middlebox
   acts to optimise the transmission of RTP 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.

```
+---+        +-------------+       +---+
| A |<---->|               |<---->| B |
+---+        | RTP mixer,  |       +---+
             | translator, |
             | or other    |
+---+        | middlebox   |       +---+
| C |<---->|               |<---->| D |
+---+        +-------------+       +---+
```
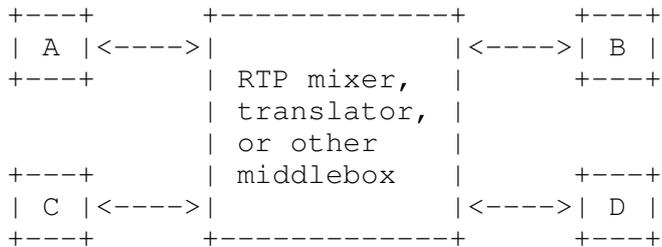
Figure 2: RTP mixer with only unicast paths

There are various methods of implementation for the middlebox. If
   implemented as a standard RTP mixer or translator, a single RTP
   session will extend across the middlebox and encompass all the
   end-points in one multi-party session. Other types of middlebox
   might use separate RTP sessions between each end-point and the
   middlebox. A common aspect is that these central nodes can use a
   number of tools to control the media encoding provided by a WebRTC
   end-point. This includes functions like requesting breaking the
   encoding chain and have the encoder produce a so called Intra
   frame. Another is limiting the bit-rate of a given stream to
   better suit the mixer view of the multiple down-streams. Others
   are controlling the most suitable frame-rate, picture resolution,
   the trade-off between frame-rate and spatial quality. The
   middlebox gets the significant responsibility to correctly perform
   congestion control, source identification, manage synchronisation
   while providing the application with suitable media optimizations.
   The middlebox is also has to be a trusted node when it comes to
   security, since it manipulates either the RTP header or the media
   itself (or both) received from one end-point, before sending it on
   towards the end-point(s), thus they need to be able to decrypt and
   then encrypt it before sending it out.

RTP Mixers can create a situation where an end-point experiences a
   situation in-between a session with only two end-points and
   multiple RTP sessions. Mixers are expected to not forward RTCP

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

Providing source authentication in multi-party scenarios is a
challenge. In the mixer-based topologies, end-points source
authentication is based on, firstly, verifying that media comes
from the mixer by cryptographic verification and, secondly, trust
in the mixer to correctly identify any source towards the end-
point. In RTP sessions where multiple end-points are directly
visible to an end-point, all end-points will have knowledge about
each others' master keys, and can thus inject packets claimed to
come from another end-point in the session. Any node performing
relay can perform non-cryptographic mitigation by preventing
forwarding of packets that have SSRC fields that came from other
end-points before. For cryptographic verification of the source
SRTP would require additional security mechanisms, for example
TESLA for SRTP [RFC4383], that are not part of the base WebRTC
standards.

To forward media between multiple peers: It is sometimes desirable
   for an end-point that receives an RTP media stream to be able to
   forward that media stream to a third party. The are some obvious
   security and privacy implications in supporting this, but also
   potential uses. This is supported in the W3C API by taking the
   received and decoded media and using it as media source that is
   re-encoding and transmitted as a new stream.

   At the RTP layer, media forwarding acts as a back-to-back RTP
   receiver and RTP sender. The receiving side terminates the RTP
   session and decodes the media, while the sender side re-encodes
   and transmits the media using an entirely separate RTP session.
   The original sender will only see a single receiver of the media,
   and will not be able to tell that forwarding is happening based on
   RTP-layer information since the RTP session that is used to send
   the forwarded media is not connected to the RTP session on which
   the media was received by the node doing the forwarding.

   The end-point that is performing the forwarding is responsible for
   producing an RTP media stream suitable for onwards transmission.
   The outgoing RTP session that is used to send the forwarded media

   is entirely separate to the RTP session on which the media was
   received. This will require media transcoding for congestion
   control purpose to produce a suitable bit-rate for the outgoing
   RTP session, reducing media quality and forcing the forwarding
   end-point to spend the resource on the transcoding. The media
   transcoding does result in a separation of the two different legs
   removing almost all dependencies, and allowing the forwarding end-
   point to optimize its media transcoding operation. The cost is
   greatly increased computational complexity on the forwarding node.
   Receivers of the forwarded stream will see the forwarding device
   as the sender of the stream, and will not be able to tell from the
   RTP layer that they are receiving a forwarded stream rather than
   an entirely new media stream generated by the forwarding device.

12.1.3. Differentiated Treatment of Flows

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

   It is expected that the WebRTC API will allow the application to
   indicate relative priorities for different MediaStreamTracks. These
   priorities can then be used to influence the local RTP processing,
   especially when it comes to congestion control response in how to
   divide the available bandwidth between the RTP flows. Any changes in
   relative priority will also need to be considered for RTP flows that
   are associated with the main RTP flows, such as RTP retransmission
   streams and FEC. The importance of such associated RTP traffic flows
   is dependent on the media type and codec used, in regards to how
   robust that codec is to packet loss. However, a default policy might
   to be to use the same priority for associated RTP flows as for the
   primary RTP flow.

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

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

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

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

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

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

For packet based marking schemes it might be possible to mark
individual RTP packets differently based on the relative priority of
the RTP payload. For example video codecs that have I, P, and B
pictures could prioritise any payloads carrying only B frames less,
as these are less damaging to loose. As default policy all RTP
packets related to a media stream ought to be provided with the same
prioritization; per-packet prioritization is outside the scope of
this memo, but might be specified elsewhere in future.

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

12.2. Source, Flow, and Participant Identification

12.2.1. Media Streams

   Each RTP media stream is identified by a unique synchronisation
   source (SSRC) identifier. The SSRC identifier is carried in the RTP
   data packets comprising a media stream, and is also used to identify
   that stream in the corresponding RTCP reports. The SSRC is chosen as
   discussed in Section 4.8. The first stage in demultiplexing RTP and
   RTCP packets received at a WebRTC end-point is to separate the media
   streams based on their SSRC value; once that is done, additional
   demultiplexing steps can determine how and where to render the media.

   RTP allows a mixer, or other RTP-layer middlebox, to combine media
   flows from multiple sources to form a new media flow. The RTP data
   packets in that new flow can include a Contributing Source (CSRC)
   list, indicating which original SSRCs contributed to the combined
   packet. As described in Section 4.1, implementations need to support
   reception of RTP data packets containing a CSRC list and RTCP packets
   that relate to sources present in the CSRC list. The CSRC list can
   change on a packet-by-packet basis, depending on the mixing operation
   being performed. Knowledge of what sources contributed to a
   particular RTP packet can be important if the user interface
   indicates which participants are active in the session. Changes in
   the CSRC list included in packets needs to be exposed to the WebRTC
   application using some API, if the application is to be able to track
   changes in session participation. It is desirable to map CSRC values
   back into WebRTC MediaStream identities as they cross this API, to
   avoid exposing the SSRC/CSRC name space to JavaScript applications.

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

12.2.2. Media Streams: SSRC Collision Detection

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

   In a point-to-point session where each SSRC is associated with either
   of the two end-points and where the main media carrying SSRC
   identifier will be announced in the signalling channel, a collision
   is less likely to occur due to the information about used SSRCs
   provided by Source-Specific SDP Attributes [RFC5576]. Still if both
   end-points start uses an new SSRC identifier prior to having
   signalled it to the peer and received acknowledgement on the

signalling message, there can be collisions. The Source-Specific SDP
Attributes [RFC5576] contains no mechanism to resolve SSRC collisions
or reject a end-points usage of an SSRC.

There could also appear SSRC values that are not signalled. This is
more likely than it appears as certain RTP functions need extra SSRCs
to provide functionality related to another (the "main") SSRC, for
example, SSRC multiplexed RTP retransmission [RFC4588]. In those
cases, an end-point can create a new SSRC that strictly doesn't need
to be announced over the signalling channel to function correctly on
both RTP and RTCPeerConnection 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 RTCPeerConnection to another RTCPeerConnection the end-point
already has, thus forming a loop where the end-point will receive its
own traffic. While is is clearly considered a bug, it is important
that the end-point is able to recognise and handle the case when it
occurs. This case becomes even more problematic when media mixers,
and so on, are involved, where the stream received is a different
stream but still contains this client's input.

These SSRC/CSRC collisions can only be handled on RTP level as long
as the same RTP session is extended across multiple
RTCPeerConnections by a RTP middlebox. To resolve the more generic
case where multiple RTCPeerConnections are interconnected, then
identification of the media source(s) part of a MediaStreamTrack
being propagated across multiple interconnected RTCPeerConnection
needs to be preserved across these interconnections.

12.2.3. Media Synchronisation Context

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

The next provision is that the internal clocks of all media sources,
i.e., what drives the RTP timestamp, can be correlated to a system
clock that is provided in RTCP Sender Reports encoded in an NTP
format. By correlating all RTP timestamps to a common system clock

for all sources, the timing relation of the different RTP 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.

13. Security Considerations

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

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

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

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

   The guidelines in [RFC6562] apply when using variable bit rate (VBR)
   audio codecs such as Opus (see Section 4.3 for discussion of mandated
   audio codecs). These guidelines in [RFC6562] also apply, but are of
   lesser importance, when using the client-to-mixer audio level header
   extensions (Section 5.2.2) or the mixer-to-client audio level header
   extensions (Section 5.2.3).

14. IANA Considerations

   This memo makes no request of IANA.

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

15. Open Issues

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

   1. tbd: The discussion at IETF 88 confirmed that there is broad
       agreement to support simulcast, however the method for achieving
       simulcast of a media source has to be decided.

16. Acknowledgements

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

17. References

17.1. Normative References

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

   [I-D.ietf-avtcore-rtp-circuit-breakers]
              Perkins, C. and V. Singh, "Multimedia Congestion Control:
              Circuit Breakers for Unicast RTP Sessions", draft-ietf-
              avtcore-rtp-circuit-breakers-03 (work in progress), July
              2013.

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

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

   [I-D.ietf-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-11 (work in progress), November
                2013.

   [I-D.ietf-mmusic-sdp-bundle-negotiation]
                Holmberg, C., Alvestrand, H., and C. Jennings,
                "Multiplexing Negotiation Using Session Description
                Protocol (SDP) Port Numbers", draft-ietf-mmusic-sdp-
                bundle-negotiation-05 (work in progress), October 2013.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

   [RFC6904]   Lennox, J., "Encryption of Header Extensions in the Secure
               Real-time Transport Protocol (SRTP)", RFC 6904, April
               2013.

   [RFC7007]   Terriberry, T., "Update to Remove DVI4 from the
               Recommended Codecs for the RTP Profile for Audio and Video
               Conferences with Minimal Control (RTP/AVP)", RFC 7007,
               August 2013.

   [RFC7022] Begen, A., Perkins, C., Wing, D., and E. Rescorla,
              "Guidelines for Choosing RTP Control Protocol (RTCP)
              Canonical Names (CNAMEs)", RFC 7022, September 2013.

   [W3C.WD-mediacapture-streams-20130903]
              Burnett, D., Bergkvist, A., Jennings, C., and A.
              Narayanan, "Media Capture and Streams", World Wide Web
              Consortium WD WD-mediacapture-streams-20130903, September
              2013, <http://www.w3.org/TR/2013/
              WD-mediacapture-streams-20130903>.

   [W3C.WD-webrtc-20130910]
              Bergkvist, A., Burnett, D., Jennings, C., and A.
              Narayanan, "WebRTC 1.0: Real-time Communication Between
              Browsers", World Wide Web Consortium WD WD-
              webrtc-20130910, September 2013,
              <http://www.w3.org/TR/2013/WD-webrtc-20130910>.

17.2. Informative References

   [I-D.dhesikan-tsvwg-rtcweb-qos]
              Dhesikan, S., Druta, D., Jones, P., and J. Polk, "DSCP and
              other packet markings for RTCWeb QoS", draft-dhesikan-
              tsvwg-rtcweb-qos-03 (work in progress), December 2013.

   [I-D.ietf-avtcore-multiplex-guidelines]
              Westerlund, M., Perkins, C., and H. Alvestrand,
              "Guidelines for using the Multiplexing Features of RTP to
              Support Multiple Media Streams", draft-ietf-avtcore-
              multiplex-guidelines-01 (work in progress), July 2013.

   [I-D.ietf-avtcore-rtp-topologies-update]
              Westerlund, M. and S. Wenger, "RTP Topologies", draft-
              ietf-avtcore-rtp-topologies-update-01 (work in progress),
              October 2013.

   [I-D.ietf-mmusic-msid]
              Alvestrand, H., "Cross Session Stream Identification in
              the Session Description Protocol", draft-ietf-mmusic-
              msid-02 (work in progress), November 2013.

   [I-D.ietf-rtcweb-overview]
              Alvestrand, H., "Overview: Real Time Protocols for Brower-
              based Applications", draft-ietf-rtcweb-overview-08 (work
              in progress), September 2013.

   [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-12 (work in
progress), October 2013.

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

[I-D.westerlund-avtcore-transport-multiplexing]
          Westerlund, M. and C. Perkins, "Multiplexing Multiple RTP
          Sessions onto a Single Lower-Layer Transport", draft-
          westerlund-avtcore-transport-multiplexing-07 (work in
          progress), October 2013.

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

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

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

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

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

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

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

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

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

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

   [RFC6792] Wu, Q., Hunt, G., and P. Arden, "Guidelines for Use of the
              RTP Monitoring Framework", RFC 6792, November 2012.

Authors' Addresses

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

   Email: csp@csperkins.org
   URI: http://csperkins.org/

   Magnus Westerlund
   Ericsson
   Farogatan 6
   SE-164 80 Kista
   Sweden

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

   Joerg Ott
   Aalto University
   School of Electrical Engineering
   Espoo 02150
   Finland

   Email: jorg.ott@aalto.fi