

MMUSIC WG
Internet-Draft
Intended status: Informational
Expires: December 30, 2013

R. Even
Huawei Technologies
J. Lennox
Vidyo
Q. Wu
Huawei Technologies
June 28, 2013

The Session Description Protocol (SDP) Application Token Attribute
draft-even-mmusic-application-token-00.txt

Abstract

The RTP fixed header includes the payload type number and the SSRC values of the RTP stream. RTP defines how you de-multiplex streams within an RTP session, but in some use cases applications need further identifiers in order to identify the application semantics associated with particular streams within the session.

This document defines a mechanism to provide the mapping between the SSRCs of RTP streams and the application semantics by defining extensions to RTP and RTCP messages.

Status of This Memo

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

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

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

This Internet-Draft will expire on December 30, 2013.

Copyright Notice

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

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents

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

Table of Contents

1. Introduction	2
2. Terminology	3
3. Proposal for an Application ID token	4
3.1. RTCP SDES message	6
3.2. RTP Header Extension	6
4. Using Application ID token	7
5. Acknowledgements	8
6. IANA Considerations	9
7. Security Considerations	9
8. References	9
8.1. Normative References	9
8.2. Informative References	9
Authors' Addresses	10

1. Introduction

The RTP [RFC3550] header includes the payload type number and the SSRC values of the RTP stream. RTP defines how you de-multiplex streams within an RTP session, but in some use cases, applications need further identifiers in order to identify semantics associated with particular streams within the session.

There is ongoing work to define how to support, using SDP [RFC4566], multiple RTP media streams in one or more m-lines that define a single RTP session (as specified in [RFC3550]). The work is addressing the WebRTC architecture [I-D.ietf-rtcweb-overview], and some work will be needed when looking for a general solution in MMUSIC that can be used for non-WebRTC systems.

RTCWEB Plan A [I-D.roach-rtcweb-plan-a] that an m-line in SDP represents a single RTP stream. De-multiplexing is done by payload type (PT) number (which MUST be unique), and if unique PTs are not feasible, use SSRC information in the SDP to identify the RTP stream.

RTCWEB Plan B [I-D.uberti-rtcweb-plan] takes a different approach, and creates a hierarchy within SDP; an m= line defines an "envelope", specifying codec and transport parameters, and [RFC5576] a=ssrc lines are used to describe individual media sources within that envelope.

Each m-line defines multiple RTP streams. This requires that the SSRCs of all RTP streams in the session are declared before they appear as RTP streams.

No plan [I-D.ivov-rtcweb-noplan] proposes using a single m-line for each media type but does not require that all SSRCs will be declared in the SDP. The de-multiplexing is done based on the unique PT numbers and the mapping of SSRC to the application usage may be done by application protocol. In web application, for example, the application specific signaling may use something like { "leftSSRC": "1234", "rightSSRC": "5678" }.

Some applications may require more information about the usage of the RTP streams. For example, RTP streams from different cameras that need to be identified by the application in order to render them correctly, or a source that can send multiple versions of the same stream in different resolutions (Simulcast [I-D.westerlund-avtcore-rtp-simulcast]).

SDP provides in [RFC4574] a "label" attribute that contains a token defined by an application and is used in its context. "Label" can be attached to m-lines in multiple SDP documents allowing the application to logically identify the media streams across SDP sessions when necessary. The "label" attribute is a token and does not provide any information about the content of the stream. [RFC4796] defines the "content" attribute providing information about the content of the stream, currently there is a small set of values for the content attribute.

Both "label" and "content" attribute are SDP media-level attributes, so when an SDP m-line supports multiple RTP streams, this value is applicable to all sources described by the SDP m-line.

There is a need to have a token that will allow the mapping between a single source (identified by an SSRC) in an m-line to the application logic (Source may be a single RTP stream identified by a unique SSRC). For example, SSRC1 is the RTP stream from the left camera and SSRC2 is the RTP stream from the right camera both can be specified in a single SDP m-line and may have the same PT number.

2. Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC2119[RFC2119] and indicate requirement levels for compliant RTP implementations.

3. Proposal for an Application ID token

As we saw in the previous section, there are tokens defined that could be used for the mapping, but they have existing usages and semantics, and tend to apply at media-level rather than source-level. In order to avoid overload of existing attributes, it is better to have a new token attribute that can identify a specific source corresponding to the application. This document defines such a new token, called "AppID".

AppID is a general-purpose token associated with an RTP stream, allowing the semantics of the stream with a token to be defined by the application. This token may be mapped, for example, to a CLUE media capture using CLUE protocol [I-D.ietf-clue-framework], or to a specific resolution in a simulcast application described in the SDP .

The token is chosen by the sender, and represents the RTP stream that will be sent to the receiver.

The proposed token can be sent using SDP, RTCP SDES messages [RFC3550], or an RTP header extension [RFC5285]

The SSRC mapping may be available to the receiver when receiving the RTP stream through the RTP header extension, but may also be available ahead of time via an RTCP SDES message conveyed before the source started sending, even if the receiver has not seen any RTP packets from this source like in a multipoint conference or in the SDP description.

The receiver can receive new sources that may be of two kinds.

- o A new RTP stream replacing an existing RTP stream, in which case the AppID of the replaced RTP stream will be assigned to the new SSRC.
- o A new RTP stream requiring a different AppID, for example, when adding a presentation stream to an existing call with two video cameras from a room.

The solution should support a RTP session as described using SDP. The RTP session may be specified using a single SDP m-line, or using Bundle [I-D.ietf-mmusic-sdp-bundle-negotiation], using more than one m-line. In the latter case, if the SSRCs of all RTP streams are not known in advance, the AppIDs associated with each m-line need to be available to the receiver in order to map each SSRC to a specific m-line configuration.

To support these cases the document defines a new SDP media level attribute `a=appID` that can be used to list all the appIDs that an application may use.

The appID syntax provides a token identifier and optional SDP attributes that describe the application usage if exists in SDP. Application usage in SDP may be, for example, an image attribute describing a simulcast application usage [`I-D.westerlund-avtcore-rtp-simulcast`].

Each value of the AppID maps to one SSRC at a time. When a new SSRC is mapped to an existing AppID using an RTP header extension or SDES message, it replaces the previous RTP stream for this application usage.

The formal representation of the appID token is:

```
appid-attribute = "appID:" tokenlist [SP attribute]

tokenlist = token *("," token)

; The base definition of "attribute" is in [RFC4566].

; (It is the content of "a=" lines.)
```

Examples:

The SSRCs of the streams are not known when the SDP offer is sent, two appID are specified and can be used for mapping to specific SSRCs in the application.

```
m=video 49200 RTP/AVP 99

a=rtpmap:99 H264/90000

a=appID:2,3
```

The second example is when the application usage of the RTP steam is specified using SDP to provide different image resolutions.

```
m=video 49200 RTP/AVP 98, 99

a=rtpmap:98 H264/90000

a=rtpmap:99 H264/90000

a=appID:2 imageattr:98 send [x=480,y=320] recv *
```

```
a=appID:3 imageattr:99 send [x=800,y=640] recv *
```

3.1. RTCP SDES message

The document specify a new RTCP SDES message

```

0          1          2          3
  0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
|   AppID = XXX   |   length   |AppID token|
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
|   ....         |

```

This AppID is the same token as defined in the new SDP attribute and will also be used in the RTP header extension.

This SDES message MAY be sent in a compound RTCP packet based on the application need.

3.2. RTP Header Extension

The Application ID could be carried within the RTP header extension field, using [RFC5285] two bytes header extension.

This is negotiated within the SDP i.e.

```
a=extmap:1 urn:ietf:params:rtp-hdrext:App-ID
```

Packets tagged by the sender with the AppID will then contain a header extension as shown below

```

0          1          2          3
  0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
| ID=1 |   Len=1   |   AppID   |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
| AppID ..... |
+---+---+---+---+---+

```

To add or modify the AppID by an intermediary can be an expensive operation, particularly if SRTP is used to authenticate the packet. Modification to the contents of the RTP header requires a re-authentication of the complete packet, and this could prove to be a limiting factor in the throughput of a multipoint device.

There is no need to send the AppID header extension with all RTP packets. Senders MAY choose to send it only when a new SSRC is sent, or when an SSRC changes its association to an AppID. If such a mode is being used, the header extension SHOULD be sent in the first few RTP packets to reduce the risk of losing it due to packet loss. For codecs with decoder refresh points (such as I-Frames in video codecs), senders also SHOULD send the AppID header extension along with the packets carrying the decoder refresh.

4. Using Application ID token

The usage of mapping may depend on the de-multiplexing of the RTP streams in the SDP m-lines. Currently we have three options discussed based on input from the RTCweb WG.

For plan A [I-D.roach-rtcweb-plan-a], since each RTP stream is described by a specific m-line it will be enough to have a media level token for mapping the sent stream.

Only need for example:

```
m=video 49200 RTP/AVP 99
```

```
a=rtpmap:99 H264/90000
```

```
a=appID 2
```

For plan B [I-D.uberti-rtcweb-plan] which adds another level of RTP stream description, the mapping of SSRC to the application will need to be at the SSRC level base on [RFC5576] since all SSRCs are specified in the m-line. The document addresses the mapping of SSRCs using the SSRC attribute but uses the msid [I-D.ietf-mmusic-msid] that defines a specific semantics for each SSRC. The following offer example is using RFC5576 to provide source specific attribute identifier.

```
m=video 49200 RTP/AVP
```

```
a=rtpmap:99 H264/90000
```

```
a=max-send-ssrc:{*:3}
```

```
a=max-recv-ssrc:{*:3}
```

```
a=ssrc:11111 AppID:1
```

```
a=ssrc:22222 AppID:2
```

```
a=ssrc:33333 AppID:3
```

When using noplan [I-D.ivov-rtcweb-noplan] in MMUSIC, not all SSRCs will be known ahead of time. For example, in the following SDP the offer offers either two streams with the same resolution (for example two cameras) or two streams with different resolutions.

```
m=video 5002 RTP/SAVPF 98
```

```
a=rtpmap:98 H264/90000
```

```
a= appID 1,2 imageattr:98 send [x=800,y=640,sar=1.1,q=0.6] recv *
```

```
a= appID 3,4 imageattr:98 send [x=480,y=320] recv *
```

```
a=max-send-ssrc:{*:2}
```

In the CLUE WG case the mapping is from an RTP stream to a CLUE media capture specified in the CLUE framework [I-D.ietf-clue-framework]. The SSRCs of all streams may be known like in PLAN B but there are cases where the SDP may not be available so a pre-announce is recommended like in the following example.

```
m=video 49200 RTP/AVP
```

```
a=rtpmap:99 H264/90000
```

```
a=max-send-ssrc:{*:5}
```

```
a=max-recv-ssrc:{*:3}
```

```
a=ssrc:11111 AppID:1
```

```
a=ssrc:22222 AppID:2
```

```
a=ssrc:33333 AppID:3
```

```
a=appID 4, 5
```

The pre-announce is needed since the new RTCP SDES message includes only the SSRC and the appID but not the PT. A receiver of the SDES message will be able to map the SSRC to a codec configuration based on the SDP pre-announced tokens.

5. Acknowledgements

Place Holder

6. IANA Considerations

TBD

7. Security Considerations

TBD.

8. References

8.1. Normative References

- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.
- [RFC3550] Schulzrinne, H., Casner, S., Frederick, R., and V. Jacobson, "RTP: A Transport Protocol for Real-Time Applications", STD 64, RFC 3550, July 2003.
- [RFC5285] Singer, D. and H. Desineni, "A General Mechanism for RTP Header Extensions", RFC 5285, July 2008.

8.2. Informative References

- [I-D.ietf-clue-framework]
Duckworth, M., Pepperell, A., and S. Wenger, "Framework for Telepresence Multi-Streams", draft-ietf-clue-framework-10 (work in progress), May 2013.
- [I-D.ietf-mmusic-msid]
Alvestrand, H., "Cross Session Stream Identification in the Session Description Protocol", draft-ietf-mmusic-msid-00 (work in progress), February 2013.
- [I-D.ietf-mmusic-sdp-bundle-negotiation]
Holmberg, C., Alvestrand, H., and C. Jennings, "Multiplexing Negotiation Using Session Description Protocol (SDP) Port Numbers", draft-ietf-mmusic-sdp-bundle-negotiation-04 (work in progress), June 2013.
- [I-D.ietf-rtcweb-overview]
Alvestrand, H., "Overview: Real Time Protocols for Browser-based Applications", draft-ietf-rtcweb-overview-06 (work in progress), February 2013.
- [I-D.iovov-rtcweb-noplan]
Ivov, E., Marocco, E., and P. Thatcher, "No Plan: Economical Use of the Offer/Answer Model in WebRTC"

Sessions with Multiple Media Sources", draft-ivov-rtcweb-noplan-01 (work in progress), June 2013.

[I-D.roach-rtcweb-plan-a]

Roach, A. and M. Thomson, "Using SDP with Large Numbers of Media Flows", draft-roach-rtcweb-plan-a-00 (work in progress), May 2013.

[I-D.uberti-rtcweb-plan]

Uberti, J., "Plan B: a proposal for signaling multiple media sources in WebRTC.", draft-uberti-rtcweb-plan-00 (work in progress), May 2013.

[I-D.westerlund-avtcore-rtp-simulcast]

Westerlund, M., Lindqvist, M., and F. Jansson, "Using Simulcast in RTP Sessions", draft-westerlund-avtcore-rtp-simulcast-02 (work in progress), February 2013.

[RFC4566] Handley, M., Jacobson, V., and C. Perkins, "SDP: Session Description Protocol", RFC 4566, July 2006.

[RFC4574] Levin, O. and G. Camarillo, "The Session Description Protocol (SDP) Label Attribute", RFC 4574, August 2006.

[RFC4575] Rosenberg, J., Schulzrinne, H., and O. Levin, "A Session Initiation Protocol (SIP) Event Package for Conference State", RFC 4575, August 2006.

[RFC4796] Hautakorpi, J. and G. Camarillo, "The Session Description Protocol (SDP) Content Attribute", RFC 4796, February 2007.

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

Authors' Addresses

Roni Even
Huawei Technologies
Tel Aviv
Israel

Email: roni.even@mail01.huawei.com

Jonathan Lennox
Vidyo, Inc.
433 Hackensack Avenue
Seventh Floor
Hackensack, NJ 07601
US

Email: jonathan@vidyo.com

Qin Wu
Huawei Technologies

Email: bill.wu@huawei.com

AVTEXT
Internet Draft
Intended status: Standards Track
Expires: January 8, 2014

A. Fineberg
vLine, Inc.
July 8, 2013

A Real-Time Transport Protocol (RTP) Header Extension for VP8 Temporal Layer
Information

draft-fineberg-avtext-temporal-layer-ext-00

Status of this Memo

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

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF), its areas, and its working groups. Note that other groups may also distribute working documents as Internet-Drafts.

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

The list of current Internet-Drafts can be accessed at <http://www.ietf.org/ietf/lid-abstracts.txt>

The list of Internet-Draft Shadow Directories can be accessed at <http://www.ietf.org/shadow.html>

This Internet-Draft will expire on December 9, 2013.

Copyright Notice

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

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

Abstract

This document defines a mechanism by which packets of Real-Time Transport Protocol (RTP) video streams encoded with the VP8 codec can indicate, in an RTP header extension, the temporal layer information about the frame encoded in the RTP packet. This information can be used in a middlebox performing bandwidth management of streams without requiring it to decrypt the streams.

Table of Contents

1. Introduction.....	2
2. Conventions used in this document.....	3
3. Temporal Layer Information.....	3
4. Signaling (Setup) Information.....	3
5. Considerations on Use.....	4
6. Security Considerations.....	4
7. IANA Considerations.....	4
8. References.....	5
8.1. Normative References.....	5
8.2. Informative References.....	5

1. Introduction

In a centralized Real-Time Transport Protocol (RTP) [RFC3550] video conference, a video mixer or forwarder receives video streams from many or all of the conference participants. It then selectively forwards some or all of the video streams to other participants in the conference.

It is often necessary to manage the bandwidth usage when forwarding these streams. One tool provided to manage the bandwidth usage when handling video streams that are encoded using the VP8 codec is the use of temporal scaling by selectively forwarding only the desired temporal layers. These layers are defined by their temporal layer index. The temporal layer information is encoded in the RTP payload format for VP8 video [draft-ietf-payload-vp8-08].

It is desirable however in the case of Secure Real-Time Transport Protocol (SRTP) [RFC3711] to be able to manage the temporal scaling without having to decrypt the SRTP packets.

As an alternative, this document defines an RTP header extension [RFC5285] through which senders of video packets can indicate the temporal layer information of the packets' payload, reducing the processing load for a server.

2. Conventions used in this document

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC-2119 [RFC2119].

3. Temporal Layer Information

The VP8 temporal layer information header extension carries the 2-bit temporal layer index (TID), layer sync bit (Y), and 15-bit running index of the frames (PictureID), as defined in [draft-ietf-payload-vp8-08], of the VP8 encoded video frame in the RTP [RFC3550] payload of the packet it is associated with. This information is carried in an RTP header extension element as defined by the "General Mechanism for RTP Header Extensions" [RFC5285].

The payload of the VP8 temporal layer information header extension element can be encoded using the one-byte header defined in [RFC5285]. Figure 1 shows a representative VP8 temporal layer information encoding.

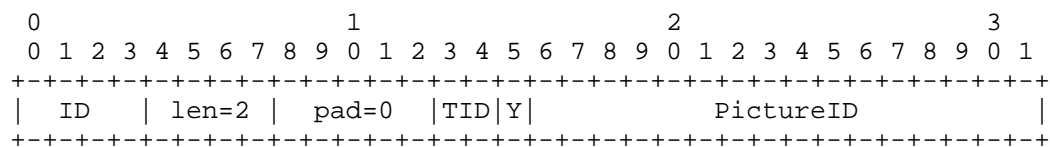


Figure 1 Representative VP8 temporal layer information encoding using the one-byte header format

Note that, as indicated in [RFC5285] length field in the one-byte header format takes the value 2 to indicate that 3 byte follows.

4. Signaling (Setup) Information

The URI for declaring this header extension in an extmap attribute is "urn:ietf:params:rtp-hdext:vp8tlinfo". It does not contain any extension attributes.

An example attribute line in the SDP, might look like:

```
a=extmap:3 urn:ietf:params:rtp-hdext:vp8tlinfo
```

If this extension is signaled when the VP8 codec is not in use, it SHOULD be ignored and this header extension element SHOULD NOT be included in the RTP header. If the header extension element is included in the RTP header when VP8 is not in use or when VP8 is in

use but temporal layering is disabled, the value for the temporal layer index (TID) MUST be 0.

5. Considerations on Use

This header extension makes no attempt to change the values already contained in the RTP Payload Format for VP8 [draft-ietf-payload-vp8]. Therefore when this header extension is present, the values in the header extension MUST be faithful representations of the values contained in the RTP payload. Any discrepancy between the values MUST be treated as an error.

6. Security Considerations

A malicious endpoint could choose to set the values in this header extension falsely so as to claim a different encoding state. It is not clear what could be gained by falsifying the encoding state as it would only impact the decoding of the senders video. While this might not impact a receiving endpoint that chooses to use the values contained in the RTP payload instead of those in the RTP header extension, it could impact endpoints that rely solely on the values from the RTP header extension as well as adversely impact the work done on a middlebox. As typically a middlebox will only use this information for bandwidth management (which includes in this context handling endpoints without the capability to decode a stream at full frame rate), this falsified data could convince the middlebox that every frame is a required frame and thereby implement a denial of service attack on an endpoint. Thus if a middlebox device relies on data from an untrusted endpoint, it SHOULD periodically audit the data to ensure the RTP payload values match those in the RTP header extension.

In the Secure Real-Time Transport Protocol (SRTP) [RFC3711], RTP header extensions are authenticated but not encrypted. When this header extension is used, VP8 temporal layer information is therefore visible on a packet-by-packet basis to an attacker passively observing the video stream. As this information provides no insight into the contents of the video stream, it is not considered a high risk exposure.

7. IANA Considerations

This document defines a new extension URI to the RTP Compact HeaderExtensions subregistry of the Real-Time Transport Protocol (RTP) Parameters registry, according to the following data:

Extension URI: urn:ietf:params:rtp-hdext:vp8tlinfo
Description: VP8 Temporal Layer Information
Contact: fineberg@vline.com
Reference: RFC XXXX

Note to RFC Editor: please replace RFC XXXX with the number of this RFC.

8. References

8.1. Normative References

- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.
- [RFC3550] Schulzrinne, H., Casner, S., Frederick, R., and V. Jacobson, "RTP: A Transport Protocol for Real-Time Applications", STD 64, RFC3550, July 2003.
- [RFC4301] Kent, S. and K. Seo, "Security Architecture for the Internet Protocol", RFC 4301, December 2005.
- [RFC5285] Singer, D. and H. Desineni, "A General Mechanism for RTP Header Extensions", RFC 5285, July 2008.

8.2. Informative References

- [I-D.ietf-payload-vp8-08] Westin, P., Lundin, H., Glover, M., Uberti, J., and F. Galligan, "RTP Payload Format for VP8 Video", (work in progress) January, 2013.
- [I-D.ietf-avtcore-srtp-encrypted-header-ext] Lennox, J., "Encryption of Header Extensions in the Secure Real-Time Transport Protocol (SRTP)" (work in progress) October, 2011.
- [RFC6386] Bankoski, J., Koleszar, J., Quillio, L., Salonen, J., Wilkins, P., and Y. Xu, "VP8 Data Format and Decoding Guide", RFC 6386, November 2011.
- [RFC3711] Baugher, M., McGrew, D., Naslund, M., Carrara, E., and K. Norrman, "The Secure Real-Time Transport Protocol (SRTP)", RFC3711, March 2004.

Authors' Addresses

Adam Fineberg
vLine, Inc.
116 Hamilton Ave
Palo Alto, CA 94301

Email: fineberg@vline.com

Network Working Group
Internet-Draft
Updates: 3550 (if approved)
Intended status: Standards Track
Expires: October 24, 2013

M.P.H. Petit-Huguenin
Impedance Mismatch
G. Zorn, Ed.
Network Zen
April 22, 2013

Support for Multiple Clock Rates in an RTP Session
draft-ietf-avtext-multiple-clock-rates-09

Abstract

This document clarifies the RTP specification when different clock rates are used in an RTP session. It also provides guidance on how to interoperate with legacy RTP implementations that use multiple clock rates.

Status of This Memo

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

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

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

This Internet-Draft will expire on October 24, 2013.

Copyright Notice

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

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

Table of Contents

1. Introduction	2
2. Terminology	4
3. Legacy RTP	4
3.1. Different SSRC	4
3.2. Same SSRC	4
3.2.1. Monotonic timestamps	5
3.2.2. Non-monotonic timestamps	5
4. Recommendations	6
4.1. RTP Sender (with RTCP)	6
4.2. RTP Sender (without RTCP)	6
4.3. RTP Receiver	7
5. Security Considerations	7
6. IANA Considerations	7
7. Acknowledgements	7
8. References	8
8.1. Normative References	8
8.2. Informative References	8
Appendix A. Example Values	9
Appendix B. Using a Fixed Clock Rate	11
Appendix C. Behavior of Legacy Implementations	11
C.1. libccrtp 2.0.2	11
C.2. libmediastreamer0 2.6.0	11
C.3. libpjmedia 1.0	12
C.4. Android RTP stack 4.0.3	12
Authors' Addresses	12

1. Introduction

The clock rate is a parameter of the payload format as identified in RTP and RTCP by the payload type value. It is often defined as being the same as the sampling rate but that is not always the case (see e.g. the G722 and MPA audio codecs [RFC3551]).

An RTP sender can switch between different payloads during the lifetime of an RTP session and because clock rates are defined by payload format, it is possible that the clock rate will also vary during an RTP session. Schulzrinne, et al. [RFC3550] lists using multiple clock rates as one of the reasons to not use different payloads on the same SSRC but unfortunately this advice has not always been followed and some RTP implementations change the payload in the same SSRC even if the different payloads use different clock rates.

This creates three problems:

- o The method used to calculate the RTP timestamp field in an RTP packet is underspecified.
- o When the same SSRC is used for different clock rates, it is difficult to know what clock rate was used for the RTP timestamp field in an RTCP SR packet.
- o When the same SSRC is used for different clock rates, it is difficult to know what clock rate was used for the interarrival jitter field in an RTCP RR packet.

Table 1 contains a non-exhaustive list of fields in RTCP packets that uses a clock rate as unit:

Field name	RTCP packet type	Reference
RTP timestamp	SR	[RFC3550]
Interarrival jitter	RR	[RFC3550]
min_jitter	XR Summary Block	[RFC3611]
max_jitter	XR Summary Block	[RFC3611]
mean_jitter	XR Summary Block	[RFC3611]
dev_jitter	XR Summary Block	[RFC3611]
Interarrival jitter	IJ	[RFC5450]
RTP timestamp	SMPTE TC	[RFC5484]
Jitter	RSI Jitter Block	[RFC5760]
Median jitter	RSI Stats Block	[RFC5760]

Table 1

This document first tries to list in Section 3 and subsections all of the algorithms known to be used in existing RTP implementations at the time of writing. These sections are not normative.

Section 4 and subsections then recommend a unique algorithm that modifies RFC 3550. These sections are normative.

2. Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [RFC2119]. In addition, this document uses the following terms:

Clock rate	The multiplier used to convert from a wallclock value in seconds to an equivalent RTP timestamp value (without the fixed random offset). Note that RFC 3550 uses various terms like "clock frequency", "media clock rate", "timestamp unit", "timestamp frequency", and "RTP timestamp clock rate" as synonymous to clock rate.
RTP Sender	A logical network element that sends RTP packets, sends RTCP SR packets, and receives RTCP RR packets.
RTP Receiver	A logical network element that receives RTP packets, receives RTCP SR packets, and sends RTCP RR packets.

3. Legacy RTP

The following sections describe the various ways legacy RTP implementations behave when multiple clock rates are used. Legacy RTP refers to RFC 3550 without the modifications introduced by this document.

3.1. Different SSRC

One way of managing multiple clock rates is to use a different SSRC for each different clock rate, as in this case there is no ambiguity on the clock rate used by fields in the RTCP packets. This method also seems to be the original intent of RTP as can be deduced from points 2 and 3 of section 5.2 of RFC 3550.

On the other hand changing the SSRC can be a problem for some implementations designed to work only with unicast IP addresses, where having multiple SSRCs is considered a corner case. Lip synchronization can also be a problem in the interval between the beginning of the new stream and the first RTCP SR packet. This is not different than what happen at the beginning of the RTP session but it can be more annoying for the end-user.

3.2. Same SSRC

The simplest way of managing multiple clock rates is to use the same SSRC for all the payload types regardless of the clock rates.

Unfortunately there is no clear definition on how the RTP timestamp should be calculated in this case. The following subsections present the algorithms used in the field.

3.2.1. Monotonic timestamps

This method of calculating the RTP timestamp ensures that the value increases monotonically. The formula used by this method is as follows:

```
timestamp = previous_timestamp
           + (current_capture_time - previous_capture_time)
           * current_clock_rate
```

The problem with this method is that the jitter calculation on the receiving side gives an invalid result during the transition between two clock rates, as shown in Table 2. The capture and arrival time are in seconds, starting at the beginning of the capture of the first packet; clock rate is in Hz; the RTP timestamp does not include the random offset; the transit, jitter, and average jitter use the clock rate as unit.

Calculating the correct transit time on the receiving side can be done by using the following formulas:

1. $\text{current_capture_time} = (\text{current_timestamp} - \text{previous_timestamp}) / \text{current_clock_rate} + \text{previous_capture_time}$
2. $\text{transit} = \text{current_clock_rate} * (\text{arrival_time} - \text{current_capture_time})$
3. $\text{previous_capture_time} = \text{current_capture_time}$

The main problem with this method, in addition to the fact that the jitter calculation described in RFC 3550 cannot be used, is that is it dependent on the previous RTP packets, packets that can be reordered or lost in the network.

3.2.2. Non-monotonic timestamps

An alternate way of generating the RTP timestamps is to use the following formula:

```
timestamp = capture_time * clock_rate
```

With this formula, the jitter calculation is correct but the RTP timestamp values are no longer increasing monotonically as shown in Table 3. RFC 3550 states that "[t]he sampling instant MUST be derived from a clock that increments monotonically[...]" but nowhere says that the RTP timestamp must increment monotonically.

The advantage with this method is that it works with the jitter calculation described in RFC 3550, as long as the correct clock rates are used. It seems that this is what most implementations are using.

4. Recommendations

The following subsections describe behavioral recommendations for RTP senders (with and without RTCP) and RTP receivers.

4.1. RTP Sender (with RTCP)

An RTP Sender with RTCP turned on MUST use a different SSRC for each different clock rate. An RTCP BYE MUST be sent and a new SSRC MUST be used if the clock rate switches back to a value already seen in the RTP stream.

To accelerate lip synchronization, the next compound RTCP packet sent by the RTP sender MUST contain multiple SR packets, the first one containing the mapping for the current clock rate and the next SR packets containing the mapping for the other clock rates seen during the last period.

The RTP extension defined in Perkins & Schierl [RFC6051] MAY be used to accelerate the synchronization.

4.2. RTP Sender (without RTCP)

An RTP Sender with RTCP turned off (i.e. by setting the RS and RR bandwidth modifiers [RFC3556] to 0) SHOULD use a different SSRC for each different clock rate but MAY use different clock rates on the same SSRC as long as the RTP timestamp is calculated as explained below:

Each time the clock rate changes, the `start_offset` and `capture_start` values are calculated with the following formulas:

```
start_offset += (capture_time - capture_start) * previous_clock_rate
capture_start = capture_time
```

For the first RTP packet, the values are initialized with the following values:


```
start_offset = random_initial_offset
capture_start = capture_time
```

After eventually updating these values, the RTP timestamp is calculated with the following formula:

```
timestamp = (capture_time - capture_start) * clock_rate
            + start_offset
```

Note that in all the formulas, capture_start is the first instant that the new timestamp rate is used. The output of the above method is exemplified in Table 4.

4.3. RTP Receiver

An RTP Receiver MUST calculate the jitter using the following formula:

```
D(i,j) = (arrival_time_j * clock_rate_i - timestamp_j)
          - (arrival_time_i * clock_rate_i - timestamp_i)
```

An RTP Receiver MUST be able to handle a compound RTCP packet with multiple SR packets.

5. Security Considerations

This document is not believed to effect the security of the RTP sessions described here in any way.

6. IANA Considerations

This document requires no IANA actions.

7. Acknowledgements

Thanks to Colin Perkins, Ali C. Begen, Harald Alvestrand, Qin Wu, and Magnus Westerlund for their comments, suggestions and questions that helped to improve this document.

Thanks to Bo Burman (who provided the values in Table 4).

Thanks to Robert Sparks and the attendees of SIPit 26 for the survey on multiple clock rates interoperability.

This document was written with the xml2rfc tool described in Rose [RFC2629].

8. References

8.1. Normative References

- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.
- [RFC3550] Schulzrinne, H., Casner, S., Frederick, R., and V. Jacobson, "RTP: A Transport Protocol for Real-Time Applications", STD 64, RFC 3550, July 2003.

8.2. Informative References

- [I-D.ietf-avt-variable-rate-audio] Wenger, S. and C. Perkins, "RTP Timestamp Frequency for Variable Rate Audio Codecs", draft-ietf-avt-variable-rate-audio-00 (work in progress), October 2004.
- [RFC2629] Rose, M.T., "Writing I-Ds and RFCs using XML", RFC 2629, June 1999.
- [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.
- [RFC3611] Friedman, T., Caceres, R., and A. Clark, "RTP Control Protocol Extended Reports (RTCP XR)", RFC 3611, November 2003.
- [RFC5450] Singer, D. and H. Desineni, "Transmission Time Offsets in RTP Streams", RFC 5450, March 2009.
- [RFC5484] Singer, D., "Associating Time-Codes with RTP Streams", RFC 5484, March 2009.
- [RFC5760] Ott, J., Chesterfield, J., and E. Schooler, "RTP Control Protocol (RTCP) Extensions for Single-Source Multicast Sessions with Unicast Feedback", RFC 5760, February 2010.
- [RFC6051] Perkins, C. and T. Schierl, "Rapid Synchronisation of RTP Flows", RFC 6051, November 2010.

Appendix A. Example Values

The following tables illustrate the timestamp and jitter values produced when the various methods discussed in the text are used.

The values shown are purely exemplary, illustrative and non-normative.

Capt. time	Clock rate	RTP timestamp	Arrival time	Transit	Jitter	Average jitter
0	8000	0	0.1	800		
0.02	8000	160	0.12	800	0	0
0.04	8000	320	0.14	800	0	0
0.06	8000	480	0.16	800	0	0
0.08	16000	800	0.18	2080	480	30
0.1	16000	1120	0.2	2080	0	28
0.12	16000	1440	0.22	2080	0	26
0.14	8000	1600	0.24	320	720	70
0.16	8000	1760	0.26	320	0	65

Table 2: Monotonic Timestamps

Capt. time	Clock rate	RTP timestamp	Arrival time	Transit	Jitter	Average jitter
0	8000	0	0.1	800		
0.02	8000	160	0.12	800	0	0
0.04	8000	320	0.14	800	0	0
0.06	8000	480	0.16	800	0	0
0.08	16000	1280	0.18	1600	0	0
0.1	16000	1600	0.2	1600	0	0
0.12	16000	1920	0.22	1600	0	0
0.14	8000	1120	0.24	800	0	0
0.16	8000	1280	0.26	800	0	0

Table 3: Non-monotonic Timestamps

Capt. time	Clock rate	RTP timestamp	Arrival time	Transit	Jitter	Average jitter
0	8000	0	0.1	800		
0.02	8000	160	0.12	800	0	0
0.04	8000	320	0.14	800	0	0
0.06	8000	480	0.16	800	0	0
0.08	16000	640	0.18	1600	0	0
0.1	16000	960	0.2	1600	0	0
0.12	16000	1280	0.22	1600	0	0
0.14	8000	1600	0.24	320	0	0
0.16	8000	1760	0.26	320	0	0

Table 4: Recommended Method for RTP Sender (without RTCP)

Appendix B. Using a Fixed Clock Rate

An alternate way of fixing the multiple clock rates issue was proposed by Wenger & Perkins [I-D.ietf-avt-variable-rate-audio]. This document proposed to define a unified clock rate, but the proposal was rejected at IETF 61.

Appendix C. Behavior of Legacy Implementations

C.1. libccrtp 2.0.2

This library uses the formula described in Section 3.2.2.

Note that this library uses `gettimeofday(2)` which is not guaranteed to increment monotonically, like when the clock is adjusted by NTP.

C.2. libmediastreamer0 2.6.0

This library (which uses the `oRTP` library) uses the formula described in Section 3.2.2.

Note that in some environments this library uses `gettimeofday(2)` which is not guaranteed to increment monotonically.

C.3. libpjmedia 1.0

This library uses the formula described in Section 3.2.2.

C.4. Android RTP stack 4.0.3

This library changes the SSRC each time the format changes, as described in Section 3.1.

Authors' Addresses

Marc Petit-Huguenin
Impedance Mismatch

Email: petithug@acm.org

Glen Zorn (editor)
Network Zen
227/358 Thanon Sanphawut
Bang Na, Bangkok 10260
Thailand

Phone: +66 (0) 8-1000-4155
Email: glenzorn@gmail.com

AVTEXT
Internet-Draft
Intended status: Standards Track
Expires: September 22, 2013

A. Begen
Cisco
C. Perkins
University of Glasgow
March 21, 2013

Duplicating RTP Streams
draft-ietf-avtext-rtp-duplication-02

Abstract

Packet loss is undesirable for real-time multimedia sessions, but can occur due to congestion, or other unplanned network outages. This is especially true for IP multicast networks, where packet loss patterns can vary greatly between receivers. One technique that can be used to recover from packet loss without incurring unbounded delay for all the receivers is to duplicate the packets and send them in separate redundant streams. This document explains how Real-time Transport Protocol (RTP) streams can be duplicated without breaking RTP or RTP Control Protocol (RTCP) rules.

Status of This Memo

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

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

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

This Internet-Draft will expire on September 22, 2013.

Copyright Notice

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

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

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

Table of Contents

1. Introduction	2
2. Terminology and Requirements Notation	3
3. Dual Streaming Use Cases	3
3.1. Temporal Redundancy	3
3.2. Spatial Redundancy	4
3.3. Dual Streaming over a Single Path or Multiple Paths	4
4. Use of RTP and RTCP with Temporal Redundancy	5
4.1. RTCP Considerations	6
4.2. Signaling Considerations	6
5. Use of RTP and RTCP with Spatial Redundancy	7
5.1. RTCP Considerations	7
5.2. Signaling Considerations	8
6. Use of RTP and RTCP with Temporal and Spatial Redundancy . .	8
7. Security Considerations	8
8. IANA Considerations	9
9. Acknowledgments	9
10. References	9
10.1. Normative References	9
10.2. Informative References	9
Authors' Addresses	10

1. Introduction

The Real-time Transport Protocol (RTP) [RFC3550] is widely used today for delivering IPTV traffic, and other real-time multimedia sessions. Many of these applications support very large numbers of receivers, and rely on intra-domain UDP/IP multicast for efficient distribution of traffic within the network.

While this combination has proved successful, there does exist a weakness. As [RFC2354] noted, packet loss is not avoidable, even in a carefully managed network. This loss might be due to congestion, it might also be a result of an unplanned outage caused by a flapping link, link or interface failure, a software bug, or a maintenance person accidentally cutting the wrong fiber. Since UDP/IP flows do not provide any means for detecting loss and retransmitting packets, it leaves up to the RTP layer and the applications to detect, and recover from, packet loss.

One technique to recover from packet loss without incurring unbounded delay for all the receivers is to duplicate the packets and send them in separate redundant streams. Variations on this idea have been implemented and deployed today [IC2011]. However, duplication of RTP streams without breaking the RTP and RTCP functionality has not been documented properly. This document explains how duplication can be achieved for RTP streams.

Stream duplication offers a simple way to protect media flows from packet loss. It has a comparatively high bandwidth overhead, since everything is sent twice, but with a low processing overhead. It is also very predictable in its overheads. Alternative approaches may be suitable in some cases, for example retransmission-based recovery [RFC4588] or Forward Error Correction [RFC6363].

2. Terminology and Requirements Notation

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119].

3. Dual Streaming Use Cases

Dual streaming refers to a technique that involves transmitting two redundant RTP streams (the original plus its duplicate) of the same content, with each stream capable of supporting the playback when there is no packet loss. Therefore, adding an additional RTP stream provides a protection against packet loss. The level of protection depends on how the packets are sent and transmitted inside the network.

It is important to note that dual streaming can easily be extended to support cases when more than two streams are desired. However, using three or more streams is rare in practice, due to the high overhead that it incurs.

3.1. Temporal Redundancy

From a routing perspective, two streams are considered identical if the following two IP header fields are the same, since they will be both routed over the same path:

- o IP Source Address
- o IP Destination Address

Two routing-plane identical RTP streams might carry the same payload, but can use different Synchronization Sources (SSRC) to differentiate the RTP packets belonging to each stream. In the context of dual RTP streaming, we assume that the sender duplicates the RTP packets and sends them in separate RTP streams, each with a unique SSRC. All the redundant streams are transmitted in the same RTP session.

For example, one main stream and its duplicate stream can be sent to the same IP destination address and UDP destination port with a certain delay between them [I-D.ietf-mmusic-delayed-duplication]. The streams carry the same payload in their respective RTP packets with identical sequence numbers. This allows receivers (or other nodes responsible for gap filling and duplicate suppression) to identify and suppress the duplicate packets, and subsequently produce a hopefully loss-free and duplication-free output stream. This process is commonly called stream merging or de-duplication.

3.2. Spatial Redundancy

An RTP source might be associated with multiple network interfaces, allowing it to send two redundant streams from two separate source addresses. Such streams can be routed over diverse or identical paths depending on the routing algorithm used inside the network. At the receiving end, the node responsible for duplicate suppression can look into various RTP header fields, for example SSRC and sequence number, to identify and suppress the duplicate packets.

If source-specific multicast (SSM) transport is used to carry such redundant streams, there will be a separate SSM session for each redundant stream since the streams are sourced from different interfaces (i.e., IP addresses). Thus, the receiving host has to join each SSM session separately.

Alternatively, an RTP source might send the redundant streams to separate IP destination addresses.

3.3. Dual Streaming over a Single Path or Multiple Paths

Having described the characteristics of the streams, one can reach the following conclusions:

1. When two routing-plane identical streams are used, the two streams will have identical IP headers. This makes it impractical to forward the packets onto different paths. In order to minimize packet loss, the packets belonging to one stream are often interleaved with packets belonging to its duplicate stream, and with a delay, so that if there is a packet loss, such a delay would allow the same packet from the duplicate

stream to reach the receiver because the chances that the same packet is lost in transit again is often small. This is what is also known as Time-shifted Redundancy, Temporal Redundancy or simply Delayed Duplication [I-D.ietf-mmusic-delayed-duplication] [IC2011]. This approach can be used with both types of dual streaming, described in Section 3.1 and Section 3.2.

2. If the two streams have different IP headers, an additional opportunity arises in that one is able to build a network, with physically diverse paths, to deliver the two streams concurrently to the intended receivers. This reduces the delay when packet loss occurs and needs to be recovered. Additionally, it also further reduces chances for packet loss. An unrecoverable loss happens only when two network failures happen in such a way that the same packet is affected on both paths. This is referred to as Spatial Diversity or Spatial Redundancy [IC2011]. The techniques used to build diverse paths are beyond the scope of this document.

Note that spatial redundancy often offers less delay in recovering from packet loss provided that the forwarding delay of the network paths are more or less the same (This is often made sure through careful network design). For both temporal and spatial redundancy approaches, packet misordering might still happen and needs to be handled using the sequence numbers of some sort (e.g., RTP sequence numbers).

To summarize, dual streaming allows an application and a network to work together to provide a near zero-loss transport with a bounded or minimum delay. The additional advantage includes a predictable bandwidth overhead that is proportional to the minimum bandwidth needed for the multimedia session, but independent of the number of receivers experiencing a packet loss and requesting a retransmission. For a survey and comparison of similar approaches, refer to [IC2011].

4. Use of RTP and RTCP with Temporal Redundancy

To achieve temporal redundancy, the main and duplicate RTP streams SHOULD be sent using the same 5-tuple of transport protocol, source and destination IP addresses, and source and destination transport ports. Due to the possible presence of network address and port translation (NAPT) devices, load balancers, or other middleboxes, use of anything other than an identical 5-tuple might also cause spatial redundancy (which might introduce an additional delay due to the delta between the path delays), and so is NOT RECOMMENDED unless the path is known to be free of such middleboxes.

Since the main and duplicate RTP streams follow an identical path, they are part of the same RTP session. Accordingly, the sender MUST choose a different SSRC for the duplicate RTP stream than it chose for the main RTP stream, following the rules in [RFC3550] Section 8.

4.1. RTCP Considerations

If RTCP is being sent for the main RTP stream, then the sender MUST also generate RTCP for the duplicate RTP stream. The RTCP for the duplicate RTP stream is generated exactly as-if the duplicate RTP stream were a regular media stream. The sender MUST NOT duplicate the RTCP packets sent for the main RTP stream when sending the duplicate stream, instead it MUST generate new RTCP reports for the duplicate stream. The sender MUST use the same RTCP CNAME in the RTCP reports it sends for both streams, so that the receiver can synchronize them.

The main and duplicate streams are conceptually synchronized using the standard RTCP Sender Report-based mechanism, deriving a mapping between their timelines. However, the RTP timestamps and sequence numbers MUST be identical in the main and duplicate streams, making the mapping quite trivial.

Both the main and duplicate RTP streams, and their corresponding RTCP reports, will be received. If RTCP is used, receivers MUST generate RTCP reports for both the main and duplicate streams in the usual way, treating them as entirely separate media streams.

4.2. Signaling Considerations

Signaling is needed to allow the receiver to determine that an RTP stream is a duplicate of another, rather than a separate stream that needs to be rendered in parallel. There are two parts to this: an SDP extension is needed in the offer/answer exchange to negotiate support for temporal redundancy; and signaling is needed to indicate which stream is the duplicate (the latter can be done in-band using an RTCP extension, or out-of-band in the SDP description).

We require out-of-band signaling for both features. The required SDP attribute to signal duplication in the SDP offer/answer exchange ('duplication-delay') is defined in [I-D.ietf-mmusic-delayed-duplication]. The required SDP grouping semantics are defined in [I-D.ietf-mmusic-duplication-grouping].

In the following SDP example, a video stream is duplicated, and the main and duplicate streams are transmitted in two separate SSRCs (1000 and 1010):

```
v=0
o=ali 1122334455 1122334466 IN IP4 dup.example.com
s=Delayed Duplication
t=0 0
m=video 30000 RTP/AVP 100
c=IN IP4 233.252.0.1/127
a=source-filter:incl IN IP4 233.252.0.1 198.51.100.1
a=rtpmap:100 MP2T/90000
a=ssrc:1000 cname:ch1a@example.com
a=ssrc:1010 cname:ch1a@example.com
a=ssrc-group:DUP 1000 1010
a=duplication-delay:50
a=mid:Ch1
```

As specified in Section 3.2 of [I-D.ietf-mmusic-duplication-grouping], it is advisable that the SSRC listed first in the "a=ssrc-group:" line (i.e., SSRC of 1000) is sent first, with the other SSRC (i.e., SSRC of 1010) being the time-delayed duplicate. This is not critical, however, and a receiving host should size its playout buffer based on the 'duplication-delay' attribute, and play the stream that arrives first in preference, with the other stream acting as a repair stream, irrespective of the order in which they are signaled.

5. Use of RTP and RTCP with Spatial Redundancy

When using spatial redundancy, the duplicate RTP stream is sent using a different source and/or destination address/port pair. This will be a separate RTP session to the session conveying the main RTP stream. Thus, the SSRCs used for the main and duplicate streams MUST be chosen randomly, following the rules in Section 8 of [RFC3550]. Accordingly, they will almost certainly not match each other. The sender MUST, however, use the same RTCP CNAME for both the main and duplicate streams. An "a=group:DUP" line or "a=ssrc-group:DUP" line is used to indicate duplication.

5.1. RTCP Considerations

If RTCP is being sent for the main RTP stream, then the sender MUST also generate RTCP for the duplicate RTP stream. The RTCP for the duplicate RTP stream is generated exactly as-if the duplicate RTP stream were a regular media stream. The sender MUST NOT duplicate the RTCP packets sent for the main RTP stream when sending the duplicate stream, instead it MUST generate new RTCP reports for the duplicate stream. The sender MUST use the same RTCP CNAME in the RTCP reports it sends for both streams, so that the receiver can synchronize them.

The main and duplicate streams are conceptually synchronized using the standard RTCP Sender Report-based mechanism, deriving a mapping between their timelines. However, the RTP timestamps and sequence numbers **MUST** be identical in the main and duplicate streams, making the mapping quite trivial.

Both the main and duplicate RTP streams, and their corresponding RTCP reports, will be received. If RTCP is used, receivers **MUST** generate RTCP reports for both the main and duplicate streams in the usual way, treating them as entirely separate media streams.

5.2. Signaling Considerations

The required SDP grouping semantics have been defined in [I-D.ietf-mmusic-duplication-grouping]. In the following example, the redundant streams have different IP destination addresses. The example shows the same UDP port number and IP source address for each stream, but either or both could have been different for the two streams.

```
v=0
o=ali 1122334455 1122334466 IN IP4 dup.example.com
s=DUP Grouping Semantics
t=0 0
a=group:DUP Sla Slb
m=video 30000 RTP/AVP 100
c=IN IP4 233.252.0.1/127
a=source-filter:incl IN IP4 233.252.0.1 198.51.100.1
a=rtpmap:100 MP2T/90000
a=mid:Sla
m=video 30000 RTP/AVP 101
c=IN IP4 233.252.0.2/127
a=source-filter:incl IN IP4 233.252.0.2 198.51.100.1
a=rtpmap:101 MP2T/90000
a=mid:Slb
```

6. Use of RTP and RTCP with Temporal and Spatial Redundancy

This uses the same RTP/RTCP mechanisms from Sections Section 4 and Section 5, plus a combination of both sets of signaling.

7. Security Considerations

The security considerations of [RFC3550], [I-D.ietf-mmusic-delayed-duplication], and [I-D.ietf-mmusic-duplication-grouping] apply.

Stream de-duplication can be done by an in-network middlebox, rewriting the SSRC as appropriate. If the Secure RTP (SRTP) profile [RFC3711] is used to authenticate RTP packets, such rewriting is not possible without breaking the authentication, unless the de-duplication middlebox is trusted to re-authenticate the packets. This would require additional signaling that is not specified here. The use of the encryption features of SRTP does not affect stream de-duplication middleboxes, since the RTP headers are sent in the clear.

8. IANA Considerations

No IANA actions are required.

9. Acknowledgments

Thanks to Magnus Westerlund for his suggestions.

10. References

10.1. Normative References

- [RFC3550] Schulzrinne, H., Casner, S., Frederick, R., and V. Jacobson, "RTP: A Transport Protocol for Real-Time Applications", STD 64, RFC 3550, July 2003.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.
- [I-D.ietf-mmusic-delayed-duplication]
Begen, A., Cai, Y., and H. Ou, "Delayed Duplication Attribute in the Session Description Protocol", draft-ietf-mmusic-delayed-duplication-01 (work in progress), March 2013.
- [I-D.ietf-mmusic-duplication-grouping]
Begen, A., Cai, Y., and H. Ou, "Duplication Grouping Semantics in the Session Description Protocol", draft-ietf-mmusic-duplication-grouping-01 (work in progress), March 2013.
- [RFC3711] Baugher, M., McGrew, D., Naslund, M., Carrara, E., and K. Norrman, "The Secure Real-time Transport Protocol (SRTP)", RFC 3711, March 2004.

10.2. Informative References

- [RFC2354] Perkins, C. and O. Hodson, "Options for Repair of Streaming Media", RFC 2354, June 1998.

- [RFC4588] Rey, J., Leon, D., Miyazaki, A., Varsa, V., and R. Hakenberg, "RTP Retransmission Payload Format", RFC 4588, July 2006.
- [RFC6363] Watson, M., Begen, A., and V. Roca, "Forward Error Correction (FEC) Framework", RFC 6363, October 2011.
- [IC2011] Evans, J., Begen, A., Greengrass, J., and C. Filsfils, "Toward Lossless Video Transport (to appear in IEEE Internet Computing)", November 2011.

Authors' Addresses

Ali Begen
Cisco
181 Bay Street
Toronto, ON M5J 2T3
CANADA

Email: abegen@cisco.com

Colin Perkins
University of Glasgow
School of Computing Science
Glasgow G12 8QQ
UK

Email: csp@csp Perkins.org

Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: December 19, 2013

E. Iovov
Jitsi
E. Marocco
Telecom Italia
P. Thatcher
Google
June 17, 2013

No Plan: Economical Use of the Offer/Answer Model in WebRTC Sessions
with Multiple Media Sources
draft-iovov-rtcweb-noplan-01

Abstract

This document describes a model for the lightweight use of SDP Offer/Answer in WebRTC. The goal is to minimize reliance on Offer/Answer exchanges in a WebRTC session and provide applications with the tools necessary to implement the signalling that they may need in a way that best fits their custom requirements and topologies. This simplifies signalling of multiple media sources or providing RTP Synchronisation source (SSRC) identification in multi-party sessions. Another important goal of this model is to remove from clients topological constraints such as the requirement to know in advance all SSRC identifiers that they could potentially introduce in a particular session.

The model described here is similar to the one employed by the data channel JavaScript APIs in WebRTC, where methods are supported on PeerConnection without being reflected in SDP.

This document does not question the use of SDP and the Offer/Answer model or the value they have in terms of interoperability with legacy or other non-WebRTC devices.

Status of This Memo

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

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

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

This Internet-Draft will expire on December 19, 2013.

Copyright Notice

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

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

Table of Contents

1. Background	2
2. Introduction	4
3. Reliance on Offer/Answer	5
3.1. Interoperability with Legacy	6
4. Additional Session Control and Signalling	8
5. Demultiplexing and Identifying Streams (Use of Bundle)	9
6. Simulcasting, FEC, Layering and RTX (Open Issue)	10
7. WebRTC API Requirements	11
7.1. Suggested WebRTC API Using TrackSendParams	12
7.1.1. Example 2	15
8. IANA Considerations	18
9. Informative References	18
Appendix A. Acknowledgements	19
Authors' Addresses	20

1. Background

In its early stages the RTCWEB working group chose to use the Session Description Protocol (SDP) and the Offer/Answer model [RFC3264] when establishing and negotiating sessions. This choice was also accompanied by the decision not to mandate a specific signalling protocol so that, once interoperability has been achieved, web applications can choose the semantics that best fit their

requirements. In some scenarios however, such as those involving the use of multiple media sources, these choices have left open the issue of exactly which operations should be handled by SDP Offer/Answer and which of them should be left to application-specific signalling.

At the time of writing of this document, the RTCWEB working group is considering two approaches to addressing the issue, that are often referred to as Plan A [PlanA] and Plan B [PlanB]. Both of them describe semantics that require Offer/Answer exchanges in a number of situations where this could be avoided, particularly when adding or removing media sources to a session. This requirement applies equally to cases where a client adds the stream of a newly activated web cam, a simulcast flow or upon the arrival or departure of a conference participant.

Plan A handles such notifications with the addition or removal of independent m= lines [PlanA], while Plan B relies on the use of multiplexed m= lines but still depends on the Offer/Answer exchanges for the addition or removal of media stream identifiers [MSID].

By taking the Offer/Answer approach, both Plan A and Plan B take away from the application the opportunity to handle such events in a way that is most fitting for the use case, which, among other things, also goes against the working group's decision to not to define a specific signalling protocol. (It could be argued that it is therefore only natural how proponents of each plan, having different use cases in mind, are remarkably far from reaching consensus).

Reliance on preliminary announcement of SSRC identifiers is another issue. While this could be perceived as relatively straightforward in one-to-one sessions or even conference calls within controlled environments, it can be a problem in the following cases:

- o interoperability with legacy/non-WebRTC endpoints
- o use within non-controlled and potentially federated conference environments where new RTP streams may appear relatively often. In such cases the signalling required to describe all of them through Offer/Answer may represent substantial overhead while none or only a part of it (e.g. the description of a main, active speaker stream) may be required by the application.

By increasing the number of Offer/Answer exchanges Both Plan A and Plan B also increase the risk of encountering glare situations (i.e. cases where both parties attempt to modify a session at the same time). While glare is also possible with basic Offer/Answer and resolution of such situations must be implemented anyway, the need to frequently resort to such code may either negatively impact user

experience (e.g. when "back off" resolution is used) or require substantial modifications in the Offer/Answer model and/or further venturing into the land of signalling protocols [ROACH-GLARELESS-ADD].

2. Introduction

The goal of this document is to provide directions for use of the SDP Offer/Answer model in a way that satisfies the following requirements:

- o the addition and removal of media sources (e.g. conference participants, multiple web cams or "slides") must be possible without the need of Offer/Answer exchanges;
- o the addition or removal of simulcast or layered streams must be possible without the need for Offer/Answer exchanges beyond the initial declaration of such capabilities for either direction.
- o call establishment must not require preliminary announcement or even knowledge of all potentially participating media sources;
- o application specific signalling should be used to cover most semantics following call establishment, such as adding, removing or identifying SSRCs;
- o straightforward interoperability with widely deployed legacy endpoints with rudimentary support for Offer/Answer. This includes devices that allow for one audio and potentially one video m= line and that expect to only ever be required to render a single RTP stream at a time for any of them. (Note that this does NOT include devices that expect to see multiple "m=video" lines for different SSRCs as they can hardly be viewed as "widely deployed legacy").

To achieve the above requirements this specification expects that browsers and WebRTC endpoints in general will only use SDP Offer/Answer to establish transport channels and initialize an RTP stack and codec/processing chains. This also includes any renegotiation that requires the re-initialisation of these chains. For example, adding VP8 to a session that was setup with only H.264, would obviously still require an Offer/Answer exchange.

All other session control and signalling are to be left to applications.

The actual Offer/Answer semantics presented here do not differ fundamentally from those proposed by Plan A and Plan B. The main

differentiation point of this approach is the fact that the exact protocol mechanism is left to WebRTC applications. Such applications or lightweight signalling gateways can then implement either Plan A, or Plan B, or an entirely different signalling protocol, depending on what best matches their use cases and topology.

3. Reliance on Offer/Answer

The model presented in this specification relies on use of SDP and Offer/Answer in quite the same way as many of the pre-WebRTC (and most of the legacy) endpoints do: negotiating formats, establishing transport channels and exchanging, in a declarative way, media and transport parameters that are then used for the initialization of the corresponding stacks.

The following is an example presenting what this specification views as a typical offer sent by a WebRTC endpoint:

```
v=0
o=- 0 0 IN IP4 198.51.100.33
s=
t=0 0

a=group:BUNDLE audio video           // declaring BUNDLE Support
c=IN IP4 198.51.100.33
a=ice-ufrag:Qq8o/jZwnkmXpIh         // initializing ICE
a=ice-pwd:gTMACiJcZvlxdPrjfbTHL5qo
a=ice-options:trickle
a=fingerprint:sha-1                 // DTLS-SRTP keying
    a4:bl:97:ab:c7:12:9b:02:12:b8:47:45:df:d8:3a:97:54:08:3f:16

m=audio 5000 RTP/SAVPF 96 0 8
a=mid:audio
a=rtcp-mux

a=rtpmap:96 opus/48000/2             // PT mappings
a=rtpmap:0 PCMU/8000
a=rtpmap:8 PCMA/8000

a=extmap:1 urn:ietf:params:rtp-hdext:csrc-audio-level //5825 header
a=extmap:2 urn:ietf:params:rtp-hdext:ssrc-audio-level //extensions

[ICE Candidates]

m=video 5002 RTP/SAVPF 97 98
a=mid:video
a=rtcp-mux
```

```

a=rtpmap:97 VP8/90000          // PT mappings and resolutions capabilities
a=imageattr:97 \
  send [x=[480:16:800],y=[320:16:640],par=[1.2-1.3],q=0.6] \
    [x=[176:8:208],y=[144:8:176],par=[1.2-1.3]] \
  recv *
a=rtpmap:98 H264/90000
a=imageattr:98 send [x=800,y=640,sar=1.1,q=0.6] [x=480,y=320] \
  recv [x=330,y=250]

a=extmap:3 urn:ietf:params:rtp-hdrext:fec-source-ssrc //5825 header
a=extmap:4 urn:ietf:params:rtp-hdrext:rtx-source-ssrc //extensions

a=max-send-ssrc:{*:1}          // declaring maximum
a=max-recv-ssrc:{*:4}          // number of SSRCs

[ICE Candidates]

```

The answer to the offer above would have roughly the same structure and content. The most important aspects here are:

- o Preserves interoperability with most kinds of legacy or non-WebRTC endpoints.
- o Allows the negotiation of most parameters that concern the media/RTP stack (typically the browser).
- o Only a single Offer/Answer exchange is required for session establishment and, in most cases, for the entire duration of a session.
- o Leaves complete freedom to applications as to the way that they are going to signal any other information such as SSRC identification information or the addition or removal of RTP streams.

3.1. Interoperability with Legacy

Interoperating with the "widely deployed legacy endpoints" is one of the main reasons for the RTCWEB working group to choose the SDP Offer/Answer model as basis for media negotiation. It is hence important to clarify the compatibility claims that this specification makes.

A "widely deployed legacy endpoint" is considered to have the following characteristics:

- o Likely to use the SIP protocol.
- o Capability to gracefully handle one audio and potentially one video m= line in an SDP Offer.
- o Capability to render one SSRC per m=line at any given moment but multiple, consecutive SSRCs over a period of time. This would be the case with transferred session replacements for example. While the capability to handle multiple SSRCs simultaneously is not uncommon it cannot be relied upon and should first be confirmed by signalling.
- o Possibly have features such as ICE, BUNDLE, RTCP-MUX, etc. Just as likely not to.
- o Very unlikely to announce in SDP the SSRCs that they intend to use for a given session.
- o Exact set of features and capabilities: Guaranteed to be wildly and widely diverse.

While it is relatively simple for RTCWEB to accommodate some of the above, it is obviously impossible to design a model that could simply be labeled as "compatible with legacy". It is reasonable to assume that use cases involving use of such endpoints will be designed for a relatively specific set of devices and applications. The role of the WebRTC framework is to hence provide a least-common-denominator model that can then be extended by applications.

It is just as important not to make choices or assumptions that will render interoperability for some applications or topologies difficult or even impossible.

This is exactly what the use of Offer/Answer discussed here strives to achieve. Audio/Video offers originating from WebRTC endpoints will always have a maximum of one audio and one video m= line. It will be up to applications to determine exactly how many streams they can afford to send once such a session has been established. The exact mechanism to do this is outside the scope of this document (or WebRTC in general).

Note that it is still possible for WebRTC endpoints to indicate support for a maximum number of incoming or outgoing streams for reasons such as processing constraints. Use of the "max-send-ssrc" and "max-recv-ssrc" attributes [MAX-SSRC] could be one way of doing this, although that mechanism would need to be extended to provide ways of distinguishing between independent flows and complementary ones such as layered FEC and RTX. Even with this in mind it is still

important, not to rely on the presence of that indication in incoming descriptions as well as to provide applications with a way of retrieving such capabilities from the WebRTC stack (e.g. the browser).

Determining whether a peer has the ability to seamlessly switch from one SSRC to another is also left to application specific signalling. It is worth noting that protocols such as SIP for example, often accompany SSRC replacements with extra signalling (re-INVITEs with a "replaces" header) that can easily be reused by applications or mapped to something that they deem more convenient.

For the sake of interoperability this specification strongly advises against the use of multiple m= lines for a single media type. Not only would such use be meaningless to a large number of legacy endpoints but it is also likely to be mishandled by many of them and to cause unexpected behaviour.

Finally, it is also worth pointing out that there is a significant number of feature rich non-WebRTC applications and devices that have relatively advanced, modern sets of capabilities. Such endpoints hardly fit the "legacy" qualification. Yet, as is often the case with novel and/or proprietary applications, they too have adopted diverse signalling mechanisms and the requirements described in this section fully apply when it comes to interoperating with them.

4. Additional Session Control and Signalling

- o Adding and removing RTP streams to an existing session.
- o Accepting and refusing some of them.
- o Identifying SSRCs and obtaining additional metadata for them (e.g. the user corresponding to a specific SSRC).

All of the above semantics are best handled and hence should be left to applications. There are numerous existing or emerging solutions, some of them developed by the IETF, that already cover this. This includes CLUE channels [CLUE], the SIP Event Package For Conference State [RFC4575] and its XMPP variant [COIN] as well as the protocols defined within the Centralised Conferencing IETF working group [XCON]. Additional mechanisms, undoubtedly many based on JSON, are very likely to emerge in the future as WebRTC applications address varying use cases, scenarios and topologies.

The most important part of this specification is hence to prevent certain assumptions or topologies from being imposed on applications. One example of this is the need to know and include in the Offer/

Answer exchange, all the SSRCs that can show up in a session. This can be particularly problematic for scenarios that involve non-WebRTC endpoints.

Large scale conference calls, potentially federated through RTP translator-like bridges, would be another problematic scenario. Being able to always pre-announce SSRCs in such situations could of course be made to work but it would come at a price. It would either require a very high number of Offer/Answer updates that propagate the information through the entire topology, or use of tricks such as pre-allocating a range of "fake" SSRCs, announcing them to participants and then overwriting the actual SSRCs with them. Depending on the scenario both options could prove inappropriate or inefficient while some applications may not even need such information. Others could be retrieving it through simplistic means such as access to a centralized resource (e.g. an URL pointing to a JSON description of the conference).

5. Demultiplexing and Identifying Streams (Use of Bundle)

This document assumes use of BUNDLE in WebRTC endpoints. This implies that all RTP streams are likely to end up being received on the same port. A demuxing mechanism is therefore necessary in order for these packets to then be fed into the appropriate processing chain (i.e. matched to an m= line).

Note: it is important to distinguish between the demultiplexing and the identification of incoming flows. Throughout this specification the former is used to refer to the process of choosing selecting a depacketizing/decoding/processing chain to feed incoming packets to. Such decisions depend solely on the format that is used to encode the content of incoming packets.

The above is not to be confused with the process of making rendering decision about a processed flow. Such decisions include showing a "current speaker" flow at a specific location, window or video tag, while choosing a different one for a second, "slides" flow. Another example would be the possibility to attach "Alice", "Bob" and "Carol" labels on top of the appropriate UI components. This specification leaves such rendering choices entirely to application-specific signalling as described in Section 4.

This specification uses demuxing based on RTP payload types. When creating offers and answers WebRTC applications MUST therefore allocate RTP payload types only once per bundle group. In cases where rtcp-mux is in use this would mean a maximum of 96 payload types per bundle [RFC5761]. It has been pointed out that some legacy devices may have unpredictable behaviour with payload types that are

outside the 96-127 range reserved by [RFC3551] for dynamic use. Some applications or implementations may therefore choose not to use values outside this range. Whatever the reason, offerers that find they need more than the available payload type numbers, will simply need to either use a second bundle group or not use BUNDLE at all (which in the case of a single audio and a single video m= line amounts to roughly the same thing). This would also imply building a dynamic table, mapping SSRCs to PTs and m= lines, in order to then also allow for RTCP demuxing.

While not desirable, the implications of such a decision would be relatively limited. Use of trickle ICE [TRICKLE-ICE] is going to lessen the impact on call establishment latency. Also, the fact that this would only occur in a limited number of cases makes it unlikely to have a significant effect on port consumption.

An additional requirement that has been expressed toward demuxing is the ability to assign incoming packets with the same payload type to different processing chains depending on their SSRCs. A possible example for this is a scenario where two video streams are being rendered on different video screens that each have their own decoding hardware.

While the above may appear as a demuxing and a decoding related problem it is really mostly a rendering policy specific to an application. As such it should be handled by app. specific signalling that could involve custom-formatted, per-SSRC information that accompanies SDP offers and answers.

6. Simulcasting, FEC, Layering and RTX (Open Issue)

From a WebRTC perspective, repair flows such as layering, FEC, RTX and to some extent simulcasting, present an interesting challenge, which is why they are considered an open issue by this specification.

On the one hand they are transport utilities that need to be understood, supported and used by browsers in a way that is mostly transparent to applications. On the other, some applications may need to be made aware of them and given the option to control their use. This could be necessary in cases where their use needs to be signalled to non-WebRTC endpoints in an application specific way. Another example is the possibility for an application to choose to disable some or all repair flows because it has been made aware by application-specific signalling that they are temporarily not being used/rendered by the remote end (e.g. because it is only displaying a thumbnail or because a corresponding video tag is not currently visible).

One way of handling such flows would be to advertise them in the way suggested by [RFC5956] and to then control them through application specific signalling. This options has the merit of already existing but it also implies the pre-announcement and propagation of SSRCs and the bloated signalling that this incurs. Also, relying solely on Offer/Answer here would expose an offerer to the typical race condition of repair SSRCs arriving before the answer and the processing ambiguity that this would imply.

Another approach could be a combination of RTCP and RTP header extensions [RFC5285] in a way similar to the one employed by the Rapid Synchronisation of RTP Flows [RFC6051]. While such a mechanism is not currently defined by the IETF, specifying it could be relatively straightforward:

Every packet belonging to a repair flow could carry an RTP header extension [RFC5285] that points to the source stream (or source layer in case of layered mechanisms).

Again, these are just some possibilities. Different mechanisms may and probably will require different extensions or signalling ([SRCNAME] will likely be an option for some). In some cases, where layering information is provided by the codec, an extensions is not going to be necessary at all.

In cases where FEC or simulcast relations are not immediately needed by the recipient, this information could also be delayed until the reception of the first RTCP packet.

7. WebRTC API Requirements

One of the main characteristics of this specification is the use of SDP for transport channel setup and media stack initialisation only. In order for applications to be able to cover everything else it is important that WebRTC APIs actually allow for it. Given the initial directions taken by early implementations and specification work, this is currently almost but not entirely possible.

The following is a list of requirements that the WebRTC APIs would need to satisfy in order for this specification to be usable. (Note: some of the items are already possible and are only included for the sake of completeness.)

1. Expose the SSRCs of all local `MediaStreamTrack`-s that the application attaches to a `PeerConnection`.
2. Expose the SSRCs of all remote `MediaStreamTrack`-s that are received on a `PeerConnection`

3. Expose to applications all locally generated repair flows that exist for a source (e.g. FEC and RTX flows that will be generated for a webcam) their types relations and SSRCS.
4. Expose information about the maximum number of incoming streams that can be decoded and rendered.
5. Applications should be able to pause and resume (disable and enable) any `MediaStreamTrack`. This should also include the possibility to do so for specific repair flows.
6. Information about how certain `MediaStreamTrack`-s relate to each other (e.g. a given audio flow is related to a specific video flow) may be exchanged by applications after media has started arriving. At that point the corresponding `MediaStreamTrack`-s may have been announced to the application within independent `MediaStream`-s. It should therefore be possible for applications to join such tracks within a single `MediaStream`.

The following section Section 7.1 provides suggestions for addressing the above requirements.

7.1. Suggested WebRTC API Using `TrackSendParams`

This document proposes that the following methods and dictionaries be added to the WebRTC API. The changes follow the model of `createDataChannel`, which has a JS method on `PeerConnection` that makes it possible to add data channels without going through SDP. Furthermore, just like `createDataChannel` allows 2 ways to handle negotiation (the "I know what I'm doing; Here's what I want to send; Let me signal everything" mode and the "please take care of it for me; send an OPEN message" mode), this also has 2 ways to handle negotiation (the "I know what I'm doing; Here's what I want to send; Let me signal everything" mode and the "please take care of it for me; send SDP back and forth" mode).

Following the success of `createDataChannel`, this allows simple applications to Just Work and more advanced applications to easily control what they need to. In particular, it's possible to use this API to implement either Plan A or Plan B.

```
// The following two method are added to RTCPeerConnection
partial interface RTCPeerConnection {
  // Create a stream that is used to send a source stream.
  // The MediaSendStream.description can be used for signalling.
  // No media is sent until addStream(MediaSendStream) is called.
  LocalMediaStream createLocalStream(MediaStream sourceStream);
```

```
// Create a stream that is used to receive media from the remote side,  
// given the parameters signalled from MediaStreamDescription.  
MediaStream createRemoteStream(MediaStreamDescription description);  
}  
  
interface LocalMediaStream implements MediaStream {  
    // This can be changed at any time, but especially before calling  
    // PeerConnection.addStream  
    attribute MediaStreamDescription description;  
}  
  
// Represents the parameters used to either send or receive a stream  
// over a PeerConnection.  
dictionary MediaStreamDescription {  
    MediaStreamTrackDescription[] tracks;  
}  
  
// Represents the parameters used to either send or receive a track over  
// a PeerConnection. A track has many "flows", which can be grouped  
// together.  
dictionary MediaStreamTrackDescription {  
    // Same as the MediaStreamTrack.id  
    DOMString id;  
  
    // Same as the MediaStreamTrack.kind  
    DOMString kind;  
  
    // A track can have many "flows", such as for Simulcast, FEC, etc.  
    // And they can be grouped in arbitrary ways.  
    MediaFlowDescription[] flows;  
    MediaFlowGroup[] flowGroups;  
}  
  
// Represents the parameters used to either send or receive a "flow"  
// over a PeerConnection. A "flow" is a media that arrives with a  
// single, unique SSRC. One to many flows together make up the media  
// for a track. For example, there may be Simulcast, FEC, and RTX  
// flows.  
dictionary MediaFlowDescription {  
    // The "flow id" must be unique to the track, but need not be unique  
    // outside of the track (two tracks could both have a flow with the  
    // same flow ID).  
    DOMString id;  
  
    // Each flow can go over its own transport. If the JS sets this to a
```

```
// transportId that doesn't have a transport setup already, the
// browser will use SDP negotiation to setup a transport to back that
// transportId. If This is set to an MID in the SDP, then that MID's
// transport is used.
DOMString transportId;

// The SSRC used to send the flow.
unsigned int ssrc;

// When used as receive parameters, this indicates the possible list
// of codecs that might come in for this flow. For example, a given
// receive flow could be setup to receive any of OPUS, ISAC, or PCMU.
// When used as send parameters, this indicates that the first codec
// should be used, but the browser can use send other codecs if it
// needs to because of either bandwidth or CPU constraints.
MediaCodecDescription[] codecs;
}

dictionary MediaFlowGroup {
  DOMString type; // "SIM" for Simulcast, "FEC" for FEC, etc
  DOMString[] flowids;
}

dictionary MediaCodecDescription {
  unsigned byte payloadType;
  DOMString name;
  unsigned int? clockRate;
  unsigned int? bitRate;
  // A grab bag of other fmp that will need to be further defined.
  MediaCodecParam[] params;
}

dictionary MediaCodecParam {
  DOMString key;
  DOMString value;
}
}
```

Some additional notes:

- o When LocalMediaStreams are added using addStream, onnegotiatedneeded is not called, and those streams are never reflected in future SDP exchanges. Indeed, it would be impossible to put them in the SDP without first resolving if that would be Plan A SDP or Plan B SDP.

- o Just like piles of attributes would need to be defined for Plan A and for Plan B, similar attributes would need to be defined here (Luckily, much work has already been done figuring out what those parameters are :).

API Pros:

- o Either Plan A or Plan B or could be implemented in Javascript using this API
- o It exposes all the same functionality to the Javascript as SDP, but in a much nicer format that is much easier to work with.
- o Any other signalling mechanism, such as Jingle or CLUE could be implemented using this API.
- o There is almost no risk of signalling glare.
- o Debugging errors with misconfigured descriptions should be much easier with this than with large SDP blobs.

API Cons:

- o Now there are two slightly different ways to add streams: by creating a LocalMediaStream first, and not. This is, however, analogous to setting "negotiated: true" in createDataChannel. On way is "Just Work", and the other is more advanced control.
- o All the options in MediaCodecDescription are a bit complicated. Really, this is only necessary because Plan A requires being able to specify codec parameters per SSRC, and set each flow on different transports. If we did not have this requirement, we could simplify.

7.1.1. Example 2

Following is an example of how these API additions would be used:

```
// Imagine I have MyApp, handles creating a PeerConnection,
// signalling, and rendering streams. This is how the new API could be
// used.
var peerConnection = MyApp.createPeerConnection();

// On sender side:
var stream = MyApp.getMediaStream();
var localStream = peerConnection.createSendStream(stream);
sendStream.description = MyApp.modifyStream(localStream.description)
```

```
MyApp.signalAddStream(localStream.description, function(response)) {
  if (!response.rejected) {
    // Media will not be sent.
    peerConnection.addStream(localStream);
  }
}

// On receiver side:
MyApp.onAddStreamSignalled = function(streamDescription) {
  var stream = peerConnection.createReceiveStream(streamDescription);
  MyApp.renderStream(stream);
}

// In this exchange, the MediaStreamDescription signalled from the
// sender to the receiver may have looked something like this:

{
  tracks: [
    {
      id: "audio1",
      kind: "audio",
      flows: [
        {
          id: "main",
          transportId: "transport1",
          ssrc: 1111,
          codecs: [
            {
              payloadType: 111,
              name: "opus",
              // ... more codec details
            },
            {
              payloadType: 112,
              name: "pcmu",
              // ... more codec details
            }
          ]
        }
      ]
    }
  ],
},
{
  id: "video1",
  kind: "video",
  flows: [
    {
      id: "sim0",
      transportId: "transport2",
      ssrc: 2222,
      codecs: [
```



```
    {
      payloadType: 122,
      name: "vp8"
      // ... more codec details
    }
  ],
  {
    id: "sim1",
    transportId: "transport2",
    ssrc: 2223,
    codecs: [
      {
        payloadType: 122,
        name: "vp8",
        // ... more codec details
      }
    ]
  },
  {
    id: "sim2",
    transportId: "transport2",
    ssrc: 2224,
    codecs: [
      {
        payloadType: 122,
        name: "vp8",
        // ... more codec details
      }
    ]
  },
  {
    id: "sim0fec",
    transportId: "transport2",
    ssrc: 2225,
    codecs: [
      {
        payloadType: 122,
        name: "vp8",
        // ...
      }
    ]
  }
],
flowGroups: [
  {
    semantics: "SIM",
    ssrcs: [2222, 2223, 2224]
  },
  {
    semantics: "FEC",
    ssrcs: [2222, 2225]
  }
]
```

```
    }]  
  }]  
}
```

8. IANA Considerations

None.

9. Informative References

- [CLUE] Duckworth, M., Pepperell, A., and S. Wenger, "Framework for Telepresence Multi-Streams", reference.I-D.ietf-clue-framework (work in progress), May 2013, <reference.I-D.ietf-clue-framework>.
- [COIN] Ivov, E. and E. Marocco, "XEP-0298: Delivering Conference Information to Jingle Participants (Coin)", XSF XEP 0298, June 2011, <reference.I-D.ietf-coin-framework>.
- [MAX-SSRC] Westerlund, M., Burman, B., and F. Jansson, "Multiple Synchronization sources (SSRC) in RTP Session Signaling ", reference.I-D.westerlund-avtcore-max-ssrc (work in progress), July 2012, <reference.I-D.westerlund-avtcore-max-ssrc>.
- [MSID] Alvestrand, H., "Cross Session Stream Identification in the Session Description Protocol", reference.I-D.ietf-mmusic-msid (work in progress), February 2013, <reference.I-D.ietf-mmusic-msid>.
- [PlanA] Roach, A. and M. Thomson, "Using SDP with Large Numbers of Media Flows", reference.I-D.roach-rtcweb-plan-a (work in progress), May 2013, <reference.I-D.roach-rtcweb-plan-a>.
- [PlanB] Uberti, J., "Plan B: a proposal for signaling multiple media sources in WebRTC.", reference.I-D.uberti-rtcweb-plan (work in progress), May 2013, <reference.I-D.uberti-rtcweb-plan>.
- [RFC3264] Rosenberg, J. and H. Schulzrinne, "An Offer/Answer Model with Session Description Protocol (SDP)", RFC 3264, June 2002.

- [RFC3551] Schulzrinne, H. and S. Casner, "RTP Profile for Audio and Video Conferences with Minimal Control", STD 65, RFC 3551, July 2003.
- [RFC4575] Rosenberg, J., Schulzrinne, H., and O. Levin, "A Session Initiation Protocol (SIP) Event Package for Conference State", RFC 4575, August 2006.
- [RFC5285] Singer, D. and H. Desineni, "A General Mechanism for RTP Header Extensions", RFC 5285, July 2008.
- [RFC5761] Perkins, C. and M. Westerlund, "Multiplexing RTP Data and Control Packets on a Single Port", RFC 5761, April 2010.
- [RFC5956] Begen, A., "Forward Error Correction Grouping Semantics in the Session Description Protocol", RFC 5956, September 2010.
- [RFC6015] Begen, A., "RTP Payload Format for 1-D Interleaved Parity Forward Error Correction (FEC)", RFC 6015, October 2010.
- [RFC6051] Perkins, C. and T. Schierl, "Rapid Synchronisation of RTP Flows", RFC 6051, November 2010.
- [ROACH-GLARELESS-ADD]
Roach, A., "An Approach for Adding RTCWEB Media Streams without Glare", reference.I-D.roach-rtcweb-glareless-add (work in progress), May 2013, <reference.I-D.roach-rtcweb-glareless-add>.
- [SRCNAME] Westerlund, M., Burman, B., and P. Sandgren, "RTCP SDES Item SRCNAME to Label Individual Sources ", reference.I-D.westerlund-avtext-rtcp-sdes-srcname (work in progress), October 2012, <reference.I-D.westerlund-avtext-rtcp-sdes-srcname>.
- [TRICKLE-ICE]
Ivov, E., Rescorla, E., and J. Uberti, "Trickle ICE: Incremental Provisioning of Candidates for the Interactive Connectivity Establishment (ICE) Protocol ", reference.I-D.ivov-mmusic-trickle-ice (work in progress), March 2013, <reference.I-D.ivov-mmusic-trickle-ice>.
- [XCON] , "Centralized Conferencing (XCON) Status Pages", , <<http://tools.ietf.org/wg/xcon/>>.

Appendix A. Acknowledgements

Many thanks to Bernard Aboba and Mary Barnes, for reviewing this document and providing numerous comments and substantial input.

Authors' Addresses

Emil Ivov
Jitsi
Strasbourg 67000
France

Phone: +33-177-624-330
Email: emcho@jitsi.org

Enrico Marocco
Telecom Italia
Via G. Reiss Romoli, 274
Turin 10148
Italy

Email: enrico.marocco@telecomitalia.it

Peter Thatcher
Google
747 6th St S
Kirkland, WA 98033
USA

Phone: +1 857 288 8888
Email: pthatcher@google.com

Network Working Group
Internet-Draft
Intended status: Informational
Expires: January 16, 2014

J. Lennox
Vidyo
K. Gross
AVA
S. Nandakumar
G. Salgueiro
Cisco Systems
B. Burman
Ericsson
July 15, 2013

A Taxonomy of Grouping Semantics and Mechanisms for Real-Time Transport
Protocol (RTP) Sources
draft-lennox-raiarea-rtp-grouping-taxonomy-01

Abstract

The terminology about, and associations among, Real-Time Transport Protocol (RTP) sources can be complex and somewhat opaque. This document describes a number of existing and proposed relationships among RTP sources, and attempts to define common terminology for discussing protocol entities and their relationships.

This document is still very rough, but is submitted in the hopes of making future discussion productive.

Status of This Memo

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

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

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

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

Copyright Notice

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

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

Table of Contents

1. Introduction	3
2. Concepts	3
2.1. End Point	4
2.1.1. Alternate Usages	4
2.1.2. Characteristics	4
2.2. Capture Device	4
2.2.1. Alternate Usages	4
2.2.2. Characteristics	5
2.3. Media Source	5
2.3.1. Alternate Usages	5
2.3.2. Characteristics	5
2.4. Media Stream	6
2.4.1. Alternate Usages	6
2.4.2. Characteristics	6
2.5. Media Provider	6
2.5.1. Alternate Usages	7
2.5.2. Characteristics	7
2.6. RTP Session	7
2.6.1. Alternate Usages	7
2.6.2. Characteristics	7
2.7. Media Transport	8
2.7.1. Characteristics	8
2.8. Rendering Device	8
2.8.1. Characteristics	8
2.9. Media Renderer	8
2.9.1. Alternate Usages	8
2.9.2. Characteristics	9
2.10. Participant	9
2.10.1. Characteristics	9
2.11. Multimedia Session	9
2.11.1. Alternate Usages	9
2.11.2. Characteristics	10
2.12. Communication Session	10
2.12.1. Alternate Usages	10
2.12.2. Characteristics	10
3. Relationships	10

3.1.	Synchronization Context	11
3.1.1.	RTCP CNAME	12
3.1.2.	Clock Source Signaling	12
3.1.3.	CLUE Scenes	12
3.1.4.	Implicitly via RtcMediaStream	12
3.1.5.	Explicitly via SDP Mechanisms	12
3.2.	Containment Context	12
3.2.1.	Media Stream Multiplexing	13
3.2.2.	RTP Session Multiplexing	13
3.2.3.	Multiple Media Sources in a WebRTC PeerConnection . .	13
3.3.	Equivalence Context	13
3.3.1.	Simulcast	14
3.3.2.	Layered MultiStream Transmission	14
3.3.3.	Robustness and Repair	15
3.3.4.	SDP FID Semantics	17
3.4.	Session Context	17
3.4.1.	Point-to-Point Session	18
3.4.2.	Full Mesh Session	19
3.4.3.	Centralized Conference Session	20
4.	Security Considerations	20
5.	Acknowledgement	21
6.	Open Issues	21
7.	IANA Considerations	21
8.	References	21
8.1.	Normative References	21
8.2.	Informative References	21
Appendix A.	Changes From Earlier Versions	23
A.1.	Changes From Draft -00	23
Authors' Addresses	23

1. Introduction

The existing taxonomy of sources in RTP is often regarded as confusing and inconsistent. Consequently, a deep understanding of how the different terms relate to each other becomes a real challenge. Frequently cited examples of this confusion are (1) how different protocols that make use of RTP use the same terms to signify different things and (2) how the complexities addressed at one layer are often glossed over or ignored at another.

This document attempts to provide some clarity by reviewing the semantics of various aspects of sources in RTP. As an organizing mechanism, it approaches this by describing various ways that RTP sources can be grouped and associated together.

2. Concepts

This section defines concepts that serve to identify various components in a given RTP usage. For each concept an attempt is made to list any alternate definitions and usages that co-exist today along with various characteristics that further describes the concept.

All references to ControLling mUltiple streams for tElepresence (CLUE) in this document map to [I-D.ietf-clue-framework] and all references to Web Real-Time Communications (WebRTC) map to [I-D.ietf-rtcweb-overview].

2.1. End Point

A single entity sending or receiving RTP packets. It may be decomposed into several functional blocks, but as long as it behaves as a single RTP stack entity it is classified as a single "End Point".

2.1.1. Alternate Usages

The CLUE Working Group (WG) uses the terms "Media Provider" and "Media Consumer" to describes aspects of End Point pertaining to sending and receiving functionalities.

2.1.2. Characteristics

End Points can be identified in several different ways. While RTCP Canonical Names (CNAMEs) [RFC3550] provide a globally unique and stable identification mechanism for the duration of the Communication Session (See Section 2.12), their validity applies exclusively within a synchronization context. Therefore, a mechanisms outside the scope of RTP, such as an application defined mechanisms, must be depended upon to ensure End Point identification when outside this synchronization context.

2.2. Capture Device

The physical source of stream of media data of one type such as camera or microphone.

2.2.1. Alternate Usages

The CLUE WG uses the term "Capture Device" to identify a physical capture device.

WebRTC WG uses the term "Recording Device" to refer to the locally available capture devices in an end-system.

2.2.2. Characteristics

- o A Capture Device is identified either by hardware/manufacture ID or via a session-scoped device identifier as mandated by the application usage.
- o A Capture Device always corresponds to a Media Source (See Section 2.3 for a definition of this term) but vice-versa might not always be true. For example, in the cases of output from a media production function (i.e., an audio mixer) or a video editing function which can represent data from several Media Sources.

2.3. Media Source

A Media Source logically defines the source of a raw stream of media data as generated either by a single capture device or by a conceptual source. A Media Source represents an Audio Source or a Video Source.

2.3.1. Alternate Usages

The CLUE WG uses the term "Media Capture" for this purpose. A CLUE Media Capture is identified via indexed notation. The terms Audio Capture and Video Capture are used to identify Audio Sources and Video Sources respectively. Concepts such as "Capture Scene", "Capture Scene Entry" and "Capture" provide a flexible framework to represent media captured spanning spatial regions.

The WebRTC WG defines the term "RtcMediaStreamTrack" to refer to a Media Source. An "RtcMediaStreamTrack" is identified by the ID attribute on it.

Typically a Media Source is mapped to a single m=line via the Session Description Protocol (SDP) [RFC4566] unless mechanisms such as Source-Specific attributes are in place [RFC5576]. In the latter cases, an m=line can represent either multiple Media Sources or multiple Media Streams (See Section 2.4 for a definition of this term).

2.3.2. Characteristics

- o A Media Source represents a real-time source of raw stream of audio or video media data.
- o At any point, it can represent a physical capture source or conceptual source.

- o Typically raw media from a Media Source is compressed via the application of an appropriate encoding mechanism, thus creating an RTP payload for Media Streams (See Section 2.4 for a definition of this term).
- o Multiple transformations can be applied to the data from a Media Source, thus creating several Media Streams.
- o Some notable transformations are described in Section 3.3.

2.4. Media Stream

Media from a Media Source is encoded and packetized to produce one or more Media Streams representing a sequence of RTP packets.

2.4.1. Alternate Usages

The term "Stream" is used by the CLUE WG to define a encoded Media Source sent via RTP. "Capture Encoding", "Encoding Groups" are defined to capture specific details of the encoding scheme.

RFC3550 [RFC3550] uses the term Source for this purpose.

The equivalent mapping of Media Stream in SDP [RFC4566] is defined per usage. For example, each m=line can describe one Media Stream and hence one Media Source OR a single m=line can describe properties for multiple Media Streams (via [RFC5576] mechanisms for example).

2.4.2. Characteristics

- o Each Media Stream is identified by a unique Synchronization source (SSRC) [RFC3550] that is carried in every RTP and Real-time Transport Control Protocol (RTCP) packet header.
- o At any given point, an Media Stream can have one and only SSRC.
- o Each Media Stream defines a unique RTP sequence numbering and timing space.
- o Several Media Streams could potentially map to a single Media Source via the source transformations (See Section 3.3).
- o Several Media Streams can be carried over a single RTP Session.

2.5. Media Provider

A Media Provider is a logical component within the RTP Stack that is responsible for encoding the media data from one or more Media Sources to generate RTP Payload for the outbound Media Streams.

2.5.1. Alternate Usages

Within the SDP usage, an m=line describes the necessary configuration required for encoding purposes.

CLUE's "Capture Encoding" provides specific encoding configuration for this purpose.

WebRTC WG uses the term "RtcMediaStreamTrack" to qualify as source of the media data that is encoded via the Media Provider.

2.5.2. Characteristics

- o A Media Source can be multiply encoded by a given Media Provider on-the-fly by allowing various encoded representations.

2.6. RTP Session

An RTP session is an association among a group of participants communicating with RTP. It is a group communications channel which can potentially carry a number of Media Streams. Within an RTP session, every participant finds out meta-data and control information (over RTCP) about all the Media Streams in the RTP session. The bandwidth of the RTCP control channel is shared within an RTP Session.

2.6.1. Alternate Usages

Within the context of SDP a single m=line can map to a single RTP Session or multiple m=lines can map to a single RTP Session. The latter is enabled via multiplexing schemes such as BUNDLE [I-D.ietf-mmusic-sdp-bundle-negotiation], for example, that allows mapping of multiple m=lines to a single RTP Session.

2.6.2. Characteristics

- o Typically an RTP Session can carry one or more Media Streams, the latter is also termed "SSRC Multiplexing".
- o Each RTP Session is carried by a single underlying Media Transport unless multiple RTP sessions are multiplexed over a single Transport Flow. Such a scheme is alternatively called "Session Multiplexing" in the RTP context [I-D.westerlund-avtcore-transport-multiplexing].

- o An RTP Session shares a single SSRC space as defined in RFC3550 [RFC3550]. That is, those End Points can see an SSRC identifier transmitted by any of the other End Points. An End Point can receive an SSRC either as SSRC or as a Contributing source (CSRC) in RTP and RTCP packets, as defined by the endpoints' network interconnection topology.
- o Multiple RTP Sessions can be related to one another via mechanisms defined in Section 3.

2.7. Media Transport

A Media Transport defines an end-to-end transport association for carrying one or more RTP Sessions. The combination of a network address and port uniquely identifies such a transport association, for example an IP address and a UDP port.

2.7.1. Characteristics

- o Media Transport transmits RTP Packets from a source transport address to a destination transport address.
- o RTP may depend upon the lower-layer protocol to provide mechanism such as ports to multiplex the RTP and RTCP packets of an RTP Session.

2.8. Rendering Device

Represents a physical rendering device such display or speaker.

2.8.1. Characteristics

- o An End Point can potentially have multiple rendering devices of each type.
- o Incoming Media Streams are decoded by one or more Media Renderers to provide a representation suitable for rendering the media data over one or more Rendering Devices, as defined by the application usage or system-wide configuration.

2.9. Media Renderer

A Media Renderer is a logical component within the RTP Stack that is responsible for decoding the RTP Payload within the incoming Media Streams to generate media data suitable for eventual rendering.

2.9.1. Alternate Usages

Within the context of SDP, an m=line describes the necessary configuration required to decode either one or more incoming Media Streams.

The WebRTC WG uses the term "RtcMediaStreamTrack" to qualify the media data decoded via the Media Renderer corresponding to the incoming Media Stream.

2.9.2. Characteristics

- o The output from the Media Renderer is usually rendered to a Rendering Device via appropriate mechanisms as explained in Section 2.8
- o Incoming Media Streams decoded by the Media Renderer are typically identified via the SSRC.

2.10. Participant

A participant is an entity reachable by a single signaling address, and is thus related more to the signaling context than to the media context.

2.10.1. Characteristics

- o A single signaling-addressable entity, using an application-specific signaling address space, for example a SIP URI.
- o A participant can have several associated transport flows, including several separate local transport addresses for those transport flows.
- o A participant can have several multimedia sessions.

2.11. Multimedia Session

A multimedia session is an association among a group of participants engaged in the conversation via one or more RTP Sessions. It defines logical relationships among Media Sources that appear in multiple RTP Sessions.

2.11.1. Alternate Usages

RFC4566 [RFC4566] defines a multimedia session as a set of multimedia senders and receivers and the data streams flowing from senders to receivers.

RFC3550 [RFC3550] defines it as set of concurrent RTP sessions among a common group of participants. For example, a videoconference (which is a multimedia session) may contain an audio RTP session and a video RTP session.

2.11.2. Characteristics

- o Participants in RTP multimedia sessions are identified via mechanisms such as RTCP CNAME or other application level identifiers as appropriate.
- o A multimedia session can be composed of several parallel RTP Sessions with potentially multiple Media Streams per RTP Session.
- o Each participant in a multimedia sessions can have multitude of Media Captures and Media Rendering devices.

2.12. Communication Session

A communication session is an association among group of participants communicating with each other via a set of multimedia sessions.

2.12.1. Alternate Usages

The Session Description Protocol RFC4566 [RFC4566] defines a multimedia session as a set of multimedia senders and receivers and the data streams flowing from senders to receivers. In that definition it is however not clear if a multimedia session includes both the sender's and the receiver's view of the same RTP Stream.

2.12.2. Characteristics

- o Each participant in a Communication Session is identified via an application-specific signaling address.
- o A Communication Session is composed of at least one multimedia session per participant, involving one or more parallel RTP Sessions with potentially multiple Media Streams per RTP Session.

For example, in a full mesh communication, the Communication Session consists of a set of separate Multimedia Sessions between each pair of Participants. Another example is a centralized conference, where the Communication Session consists of a set of Multimedia Sessions between each Participant and the conference handler.

3. Relationships

This section provides various relationships that can co-exist between the aforementioned concepts in a given RTP usage. Using Unified Modeling Language (UML) class diagrams [UML], Figure 1 below depicts general relations between a Media Source, its Media Provider(s) and the resulting Media Stream(s).

Note: The RTCP Stream related to the RTP Stream is not shown in the figure.

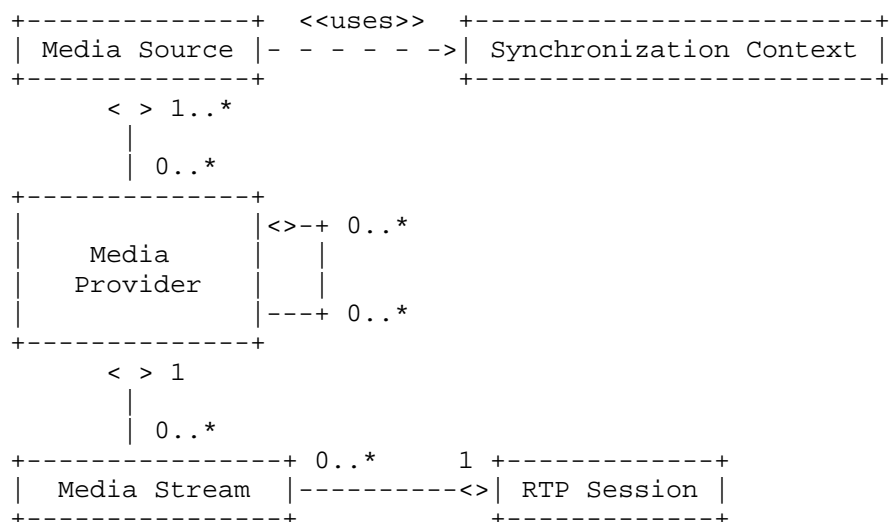


Figure 1: Media Source Relations

Media sources can have a large variety of relationships among them. These relationships can apply both between sources within a single RTP Session, and between Media Sources that occur in multiple RTP Session. Ways of relating them typically involve groups: a set of Media Sources has some relationship that applies to all those in the group, and no others. (Relationships that involve arbitrary non-grouping associations among Media sources, such that e.g., A relates to B and B to C, but A and C are unrelated, are uncommon if not nonexistent.) In many cases, the semantics of groups are not simply that the the members form an undifferentiated group, but rather that members of the group have certain roles.

3.1. Synchronization Context

A synchronization context defines requirement on a strong timing relationship between the related entities, typically requiring alignment of clock sources. Such relationship can be identified in multiple ways as listed below. A single Media Source can only belong

to a single Synchronization Context, since it is assumed that a single Media Source can only have a single media clock and requiring alignment to several Synchronization Contexts will effectively merge those into a single Synchronization Context.

A single Multimedia session can contain media from one or more Synchronization Contexts. An example of that is a Multimedia Session containing one set of audio and video for communication purposes belonging to one Synchronization context, and another set of audio and video for presentation purposes (like playing a video file) that has no strong timing relationship and need not be strictly synchronized with the audio and video used for communication.

3.1.1. RTCP CNAME

RFC3550 [RFC3550] describes Inter-media synchronization between RTP Sessions based on RTCP CNAME, RTP and Network Time Protocol (NTP) [RFC5905] timestamps.

3.1.2. Clock Source Signaling

[I-D.ietf-avtcore-clksrc] provides a mechanism to signal the clock source in SDP, thus allowing a synchronized context to be defined.

3.1.3. CLUE Scenes

In CLUE "Capture Scene", "Capture Scene Entry" and "Captures" define an implied synchronization context.

3.1.4. Implicitly via RtcMediaStream

The WebRTC WG defines "RtcMediaStream" with one or more "RtcMediaStreamTracks". All tracks in a "RtcMediaStream" are intended to be synchronized when rendered.

3.1.5. Explicitly via SDP Mechanisms

RFC5888 [RFC5888] defines m=line grouping mechanism called "Lip Synchronization (LS)" for establishing the synchronization requirement across m=lines when they map to individual sources.

RFC5576 [RFC5576] extends the above mechanism when multiple media sources are described by a single m=line.

3.2. Containment Context

A containment relationship allows composing of multiple concepts into a larger concept.

3.2.1. Media Stream Multiplexing

Multiple Media Streams can be contained within a single RTP Session via unique SSRC per Media Stream.

[I-D.ietf-mmusic-sdp-bundle-negotiation] provides SDP based signaling mechanism to enable this across several m=lines.

RFC5576 [RFC5576] enables the same for multiple Media Sources described in a single m=line.

3.2.2. RTP Session Multiplexing

[I-D.westerlund-avtcore-transport-multiplexing], for example, describes a mechanism that allow several RTP Sessions to be carried over a single underlying Media Transport.

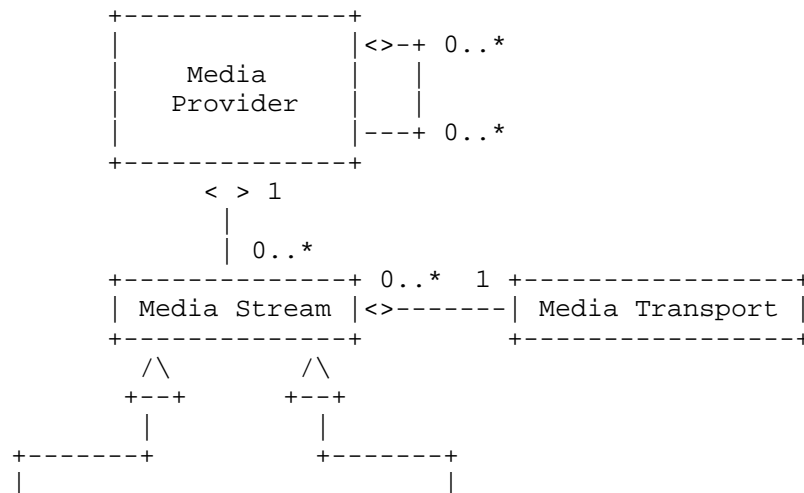
3.2.3. Multiple Media Sources in a WebRTC PeerConnection

The WebRTC WG defines a containment object named "RTCPeerConnection" that can potentially contain several Media Sources mapped to a single RTP Session or spread across several RTP Sessions.

3.3. Equivalence Context

In this relationship different instances of a concept are treated to be equivalent for the purposes of relating them to the Media Source.

Figure 2 below depicts in UML notation the general relation between a Media Provider and its Media Stream(s), including the Media Stream specializations Source Stream and RTP Repair Stream.



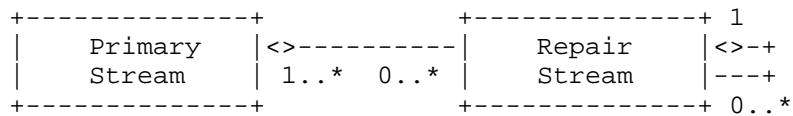


Figure 2: Media Stream Relations

This relation can in combination with Figure 1 be used to achieve a set of functionalities, described below.

3.3.1. Simulcast

A Media Source represented as multiple independent Encodings constitutes a simulcast of that Media Source. The figure below represents an example of a Media Source that is encoded into three separate simulcast streams that are in turn sent on the same transport flow.

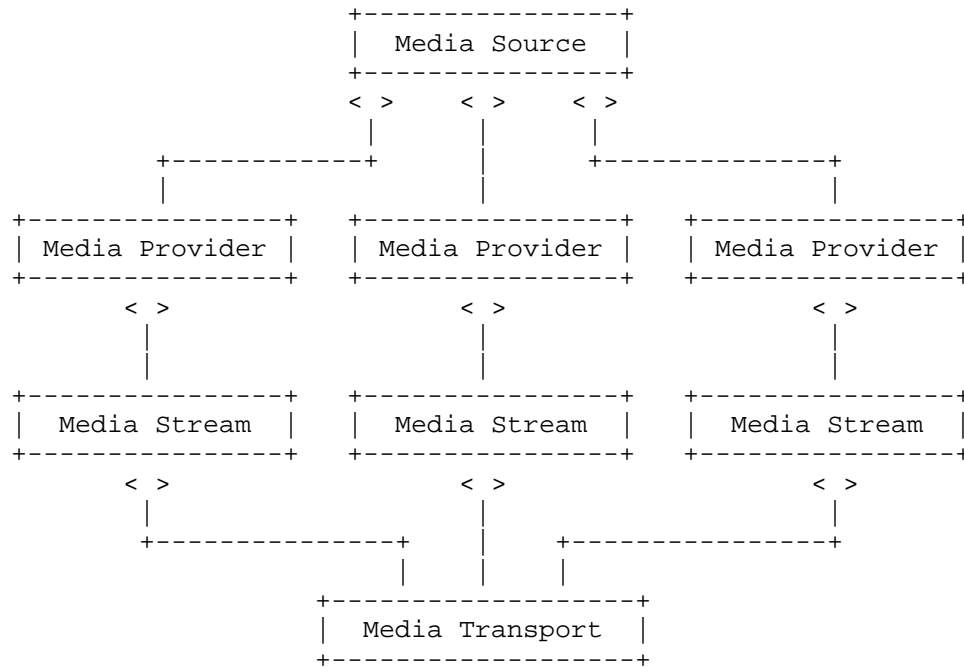


Figure 3: Example of Media Source Simulcast

3.3.2. Layered MultiStream Transmission

Multi-stream transmission (MST) is a mechanism by which different portions of a layered encoding of a media stream are sent using separate Media Streams (sometimes in separate RTP sessions). MSTs are useful for receiver control of layered media.

A Media Source represented as multiple dependent Encodings constitutes a Media Source that has layered dependency. The figure below represents an example of a Media Source that is encoded into three dependent layers, where two layers are sent on the same transport flow and the third layer is sent on a separate transport flow.

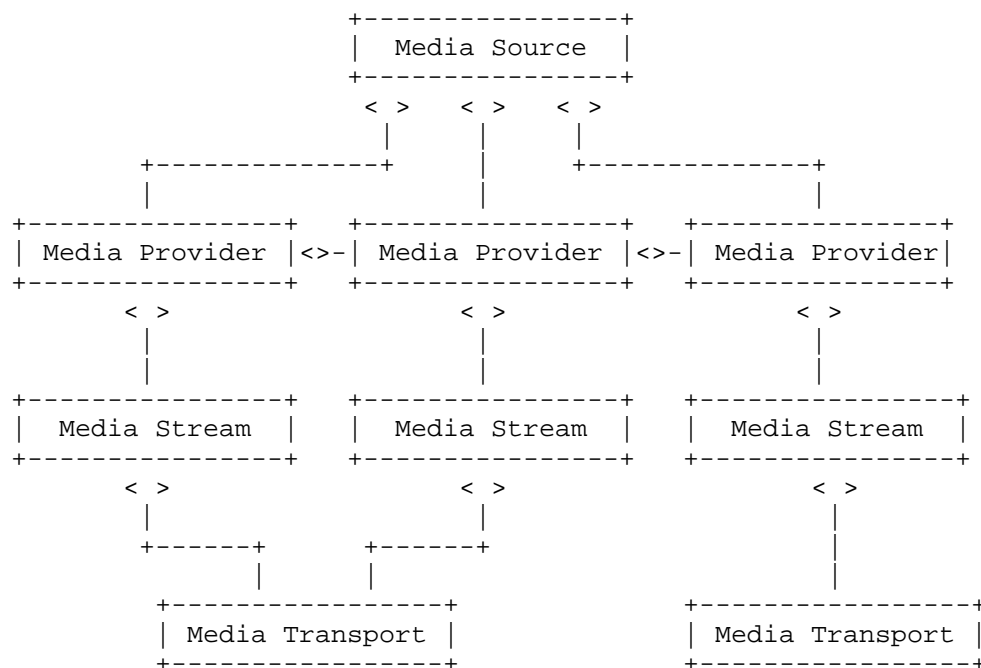


Figure 4: Example of Media Source Layered Dependency

3.3.3. Robustness and Repair

A Media Source may be protected by repair streams during transport. Several approaches listed below can achieve the same result

- o Duplication of the original Media Stream
- o Duplication of the original Media Stream with a time offset,
- o forward error correction (FEC) techniques, and.

- o retransmission of lost packets (either globally or selectively).

The figure below represents an example where a Media Source is protected by a retransmission (RTX) flow. In this example the primary Media Stream and the RTP RTX Stream share the same Media Transport.

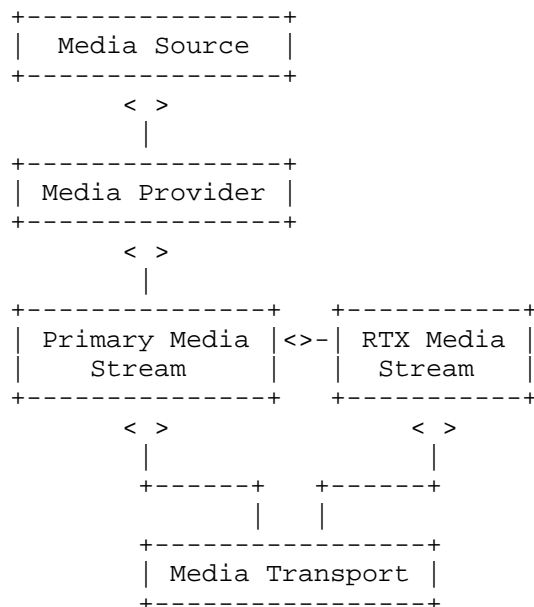


Figure 5: Example of Media Source Retransmission Flows

The figure below represents an example where two Media Sources are protected by individual FEC flows as well as one additional FEC flow that protects the set of both Media Sources (a FEC group). There are several possible ways to map those Media Streams to one or more Media Transport, but that is omitted from the figure for clarity.



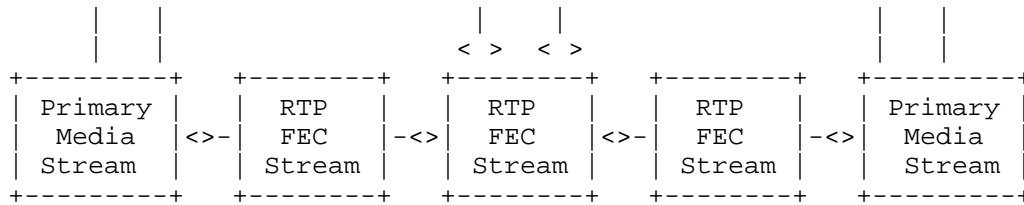


Figure 6: Example of Media Source FEC Flows

3.3.4. SDP FID Semantics

RFC5888 [RFC5888] defines m=line grouping mechanism called "FID" for establishing the equivalence of Media Streams across the m=lines under grouping.

RFC5576 [RFC5576] extends the above mechanism when multiple media sources are described by a single m=line.

3.4. Session Context

There are different ways to construct a Communication Session. The general relation in UML notation between a Communication Session, Participants, Multimedia Sessions and RTP Sessions is outlined below.

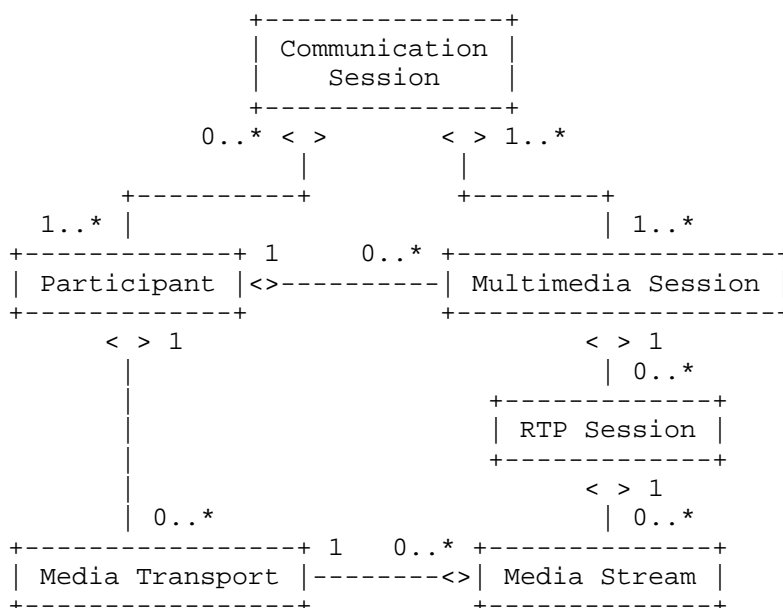
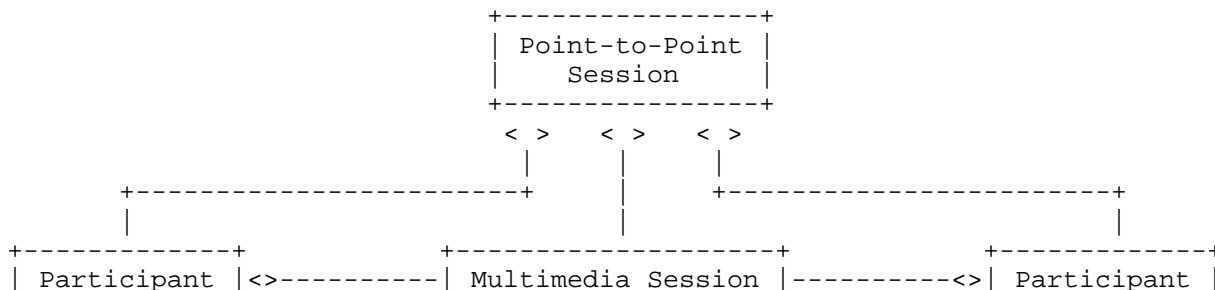


Figure 7: Session Relations

Several different flavors of Session can be possible. A few typical examples are listed in the below sub-sections, but many other are possible to construct.

3.4.1. Point-to-Point Session

In this example, a single Multimedia Session is shared between the two Participants. That Multimedia Session contains a single RTP Session with two Media Streams from each Participant. Each Participant has only a single Media Transport, carrying those Media Streams, which is the main reason why there is only a single RTP Session.



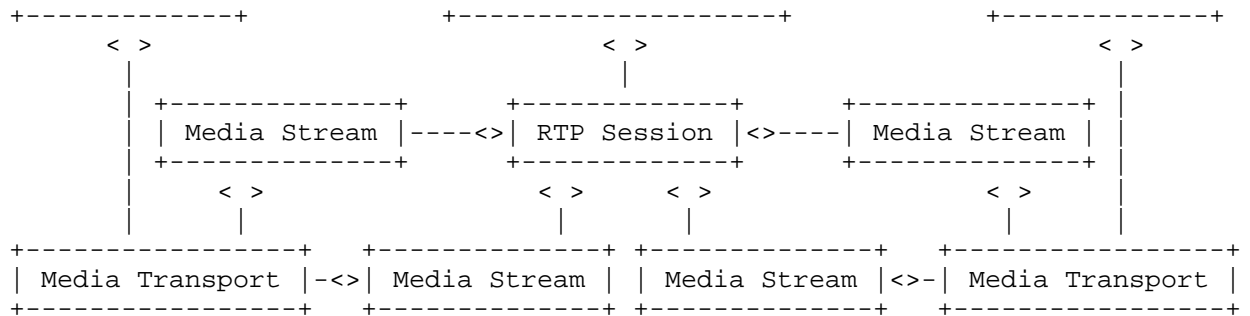
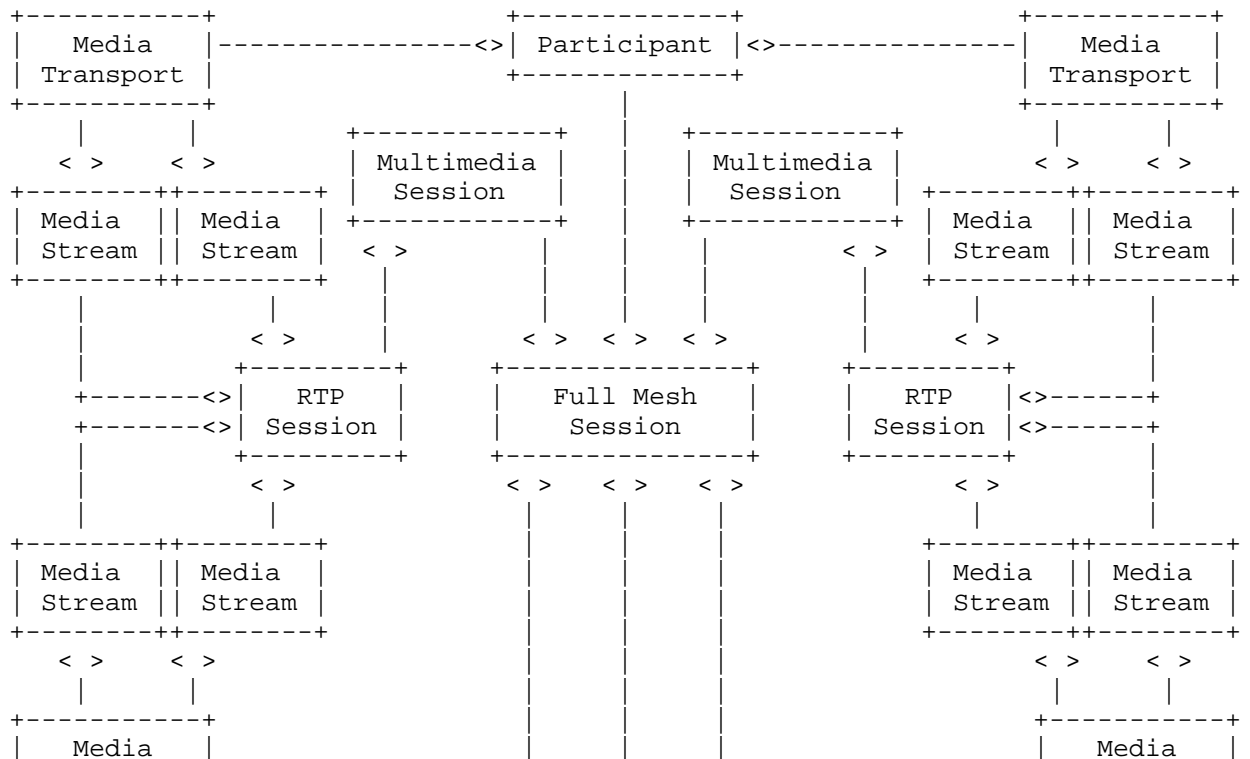


Figure 8: Example Point-to-Point Session

3.4.2. Full Mesh Session

In this example, the Full Mesh Session has three Participants, each of which has the same characteristics as the example in the previous section; a single Media Transport per peer Participant, resulting in a single RTP session between each pair of Participants.



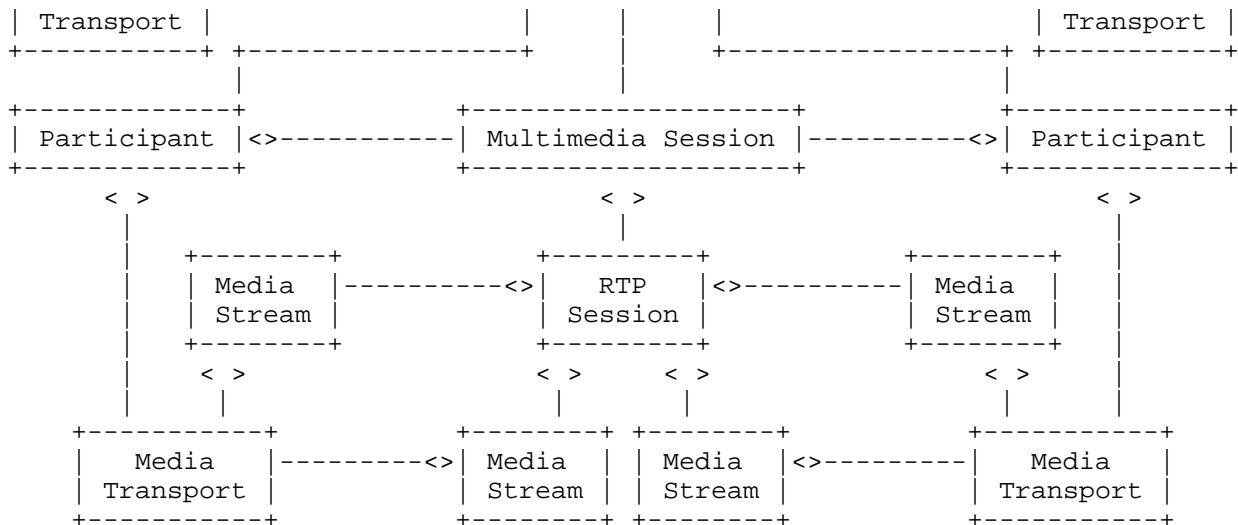


Figure 9: Example Full Mesh Session

3.4.3. Centralized Conference Session

Text to be provided

TBD

Figure 10: Example Centralized Conference Session

4. Security Considerations

This document simply tries to clarify the confusion prevalent in RTP taxonomy because of inconsistent usage by multiple technologies and protocols making use of the RTP protocol. It does not introduce any new security considerations beyond those already well documented in the RTP protocol [RFC3550] and each of the many respective specifications of the various protocols making use of it.

Hopefully having a well-defined common terminology and understanding of the complexities of the RTP architecture will help lead us to better standards, avoiding security problems.

5. Acknowledgement

This document has many concepts borrowed from several documents such as WebRTC [I-D.ietf-rtcweb-overview], CLUE [I-D.ietf-clue-framework], Multiplexing Architecture [I-D.westerlund-avtcore-transport-multiplexing]. The authors would like to thank all the authors of each of those documents.

The authors would also like to acknowledge the insights, guidance and contributions of Magnus Westerlund, Roni Even, Colin Perkins, Keith Drage, and Harald Alvestrand.

6. Open Issues

Much of the terminology is still a matter of dispute.

It might be useful to distinguish between a single endpoint's view of a source, or RTP session, or multimedia session, versus the full set of sessions and every endpoint that's communicating in them, with the signaling that established them.

(Sure to be many more...)

7. IANA Considerations

This document makes no request of IANA.

8. References

8.1. Normative References

- [RFC3550] Schulzrinne, H., Casner, S., Frederick, R., and V. Jacobson, "RTP: A Transport Protocol for Real-Time Applications", STD 64, RFC 3550, July 2003.
- [UML] Object Management Group, "OMG Unified Modeling Language (OMG UML), Superstructure, V2.2", OMG formal/2009-02-02, February 2009.

8.2. Informative References

- [I-D.ietf-avtcore-clksrc] Williams, A., Gross, K., Brandenburg, R., and H. Stokking, "RTP Clock Source Signalling", draft-ietf-avtcore-clksrc-05 (work in progress), July 2013.
- [I-D.ietf-clue-framework]

Duckworth, M., Pepperell, A., and S. Wenger, "Framework for Telepresence Multi-Streams", draft-ietf-clue-framework-11 (work in progress), July 2013.

[I-D.ietf-mmusic-sdp-bundle-negotiation]

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

[I-D.ietf-rtcweb-overview]

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

[I-D.westerlund-avtcore-transport-multiplexing]

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

[RFC3264] Rosenberg, J. and H. Schulzrinne, "An Offer/Answer Model with Session Description Protocol (SDP)", RFC 3264, June 2002.

[RFC4566] Handley, M., Jacobson, V., and C. Perkins, "SDP: Session Description Protocol", RFC 4566, July 2006.

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

[RFC5888] Camarillo, G. and H. Schulzrinne, "The Session Description Protocol (SDP) Grouping Framework", RFC 5888, June 2010.

[RFC5905] Mills, D., Martin, J., Burbank, J., and W. Kasch, "Network Time Protocol Version 4: Protocol and Algorithms Specification", RFC 5905, June 2010.

[RFC6222] Begen, A., Perkins, C., and D. Wing, "Guidelines for Choosing RTP Control Protocol (RTCP) Canonical Names (CNAMEs)", RFC 6222, April 2011.

Appendix A. Changes From Earlier Versions

NOTE TO RFC EDITOR: Please remove this section prior to publication.

A.1. Changes From Draft -00

- o Too many to list
- o Added new authors
- o Updated content organization and presentation

Authors' Addresses

Jonathan Lennox
Vidyo, Inc.
433 Hackensack Avenue
Seventh Floor
Hackensack, NJ 07601
US

Email: jonathan@vidyo.com

Kevin Gross
AVA Networks, LLC
Boulder, CO
US

Email: kevin.gross@avanw.com

Suhas Nandakumar
Cisco Systems
170 West Tasman Drive
San Jose, CA 95134
US

Email: snandaku@cisco.com

Gonzalo Salgueiro
Cisco Systems
7200-12 Kit Creek Road
Research Triangle Park, NC 27709
US

Email: gsalguei@cisco.com

Bo Burman
Ericsson
Farogatan 6
SE-164 80 Kista
Sweden

Phone: +46 10 714 13 11
Email: bo.burman@ericsson.com

Network Working Group
Internet-Draft
Intended status: Informational
Expires: January 16, 2014

A. B. Roach
Mozilla
J. Uberti
Google
M. Thomson
Microsoft
July 15, 2013

A Unified Plan for Using SDP with Large Numbers of Media Flows
draft-roach-mmusic-unified-plan-00

Abstract

A recurrent theme in emerging real-time communications use cases, such as RTCWEB, has been the need to handle very large numbers of media flows. Unfortunately, naive uses of SDP do not handle this case particularly well. This document describes a modest set of extensions to SDP which allow it to cleanly handle arbitrary numbers of flows while still retaining a large degree of backward compatibility with existing and non-RTCWEB endpoints.

Status of This Memo

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

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

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

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

Copyright Notice

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

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

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

Table of Contents

1. Introduction	3
1.1. Design Goals	4
1.1.1. Support for a large number of arbitrary sources	4
1.1.2. Support for fine-grained receiver control of sources . .	5
1.1.3. Glareless addition and removal of sources	5
1.1.4. Interworking with other devices	5
1.1.5. Avoidance of excessive port allocation	6
1.1.6. Simple binding of MediaStreamTrack to SDP	6
1.1.7. Support for RTX, FEC, simulcast, layered coding	6
1.2. Terminology	6
1.3. Syntax Conventions	7
2. Solution Overview	7
3. Detailed Description	8
3.1. Bundle-Only M-Lines	8
3.2. Correlation	12
3.2.1. Correlating RTP Sources with m-lines	12
3.2.1.1. RTP Header Extension Correlation	13
3.2.1.2. Payload Type Correlation	14
3.2.2. Correlating Media Stream Tracks with m-lines	16
3.2.3. Correlating Media Stream Tracks with RTP Sources . . .	16
3.3. Handling of Simulcast, Forward Error Correction, and Retransmission Streams	16
3.4. Glare Minimization	18
3.4.1. Adding a Stream	19
3.4.2. Changing a Stream	19
3.4.3. Removing a Stream	20
3.5. Negotiation of Stream Ordinality	20
3.6. Compatibility with Legacy uses	22
4. Examples	23
4.1. Simple example with one audio and one video	23
4.2. Multiple Videos	26
4.3. Many Videos	28
4.4. Multiple Videos with Simulcast	30
4.5. Video with Simulcast and RTX	31
4.6. Video with Simulcast and FEC	32
4.7. Video with Layered Coding	33
5. Security Considerations	35
6. IANA Considerations	35
7. Acknowledgements	35
8. References	35

8.1. Normative References	35
8.2. Informative References	36
Authors' Addresses	37

1. Introduction

A recurrent theme in new RTC technologies has been the need to cleanly handle very large numbers of media flows. For instance, a videoconferencing application might have a main display plus thumbnails for 10 or more other speakers all displayed at the same time. If each video source is encoded in multiple resolutions (e.g., simulcast or layered coding) and also has FEC or RTX, this could easily add up to 30 or more independent RTP flows.

This document focuses on the WebRTC use cases, and uses its terminology to discuss key concepts. The approach described herein, however, is not intended to be WebRTC specific, and should be generalized to other SDP-using applications.

The standard way of encoding this information in SDP is to have each RTP flow (i.e., SSRC) appear on its own m-line. For instance, the SDP for two cameras with audio from a device with a public IP address could look something like:

```
v=0
o=- 20518 0 IN IP4 203.0.113.1
s=
t=0 0
c=IN IP4 203.0.113.1
a=ice-ufrag:F7gI
a=ice-pwd:x9cml/YzichV2+XlhiMu8g
a=fingerprint:sha-1
    42:89:c5:c6:55:9d:6e:c8:e8:83:55:2a:39:f9:b6:eb:e9:a3:a9:e7

m=audio 54400 RTP/SAVPF 0 96
a=msid:ma ta
a=extmap:1 urn:ietf:params:rtp-hdrext:stream-correlator 52595
a=rtpmap:0 PCMU/8000
a=rtpmap:96 opus/48000
a=ptime:20
a=sendrecv
a=candidate:0 1 UDP 2113667327 203.0.113.1 54400 typ host
a=candidate:1 2 UDP 2113667326 203.0.113.1 54401 typ host

m=video 55400 RTP/SAVPF 96 97
a=msid:ma tb
a=extmap:1 urn:ietf:params:rtp-hdrext:stream-correlator 56036
a=rtpmap:96 H264/90000
```

```
a=fmtp:96 profile-level-id=4d0028;packetization-mode=1
a=rtpmap:97 VP8/90000
a=sendrecv
a=candidate:0 1 UDP 2113667327 203.0.113.1 55400 typ host
a=candidate:1 2 UDP 2113667326 203.0.113.1 55401 typ host

m=video 56400 RTP/SAVPF 96 97
a=msid:ma tc
a=extmap:1 urn:ietf:params:rtp-hdrext:stream-correlator 21909
a=rtpmap:96 H264/90000
a=fmtp:96 profile-level-id=4d0028;packetization-mode=1
a=rtpmap:97 VP8/90000
a=sendrecv
a=candidate:0 1 UDP 2113667327 203.0.113.1 56400 typ host
a=candidate:1 2 UDP 2113667326 203.0.113.1 56401 typ host
```

Unfortunately, as the number of independent media sources starts to increase, the scaling properties of this approach become problematic. In particular, SDP currently requires that each m-line have its own transport parameters (port, ICE candidates, etc.), which can get expensive. For instance, the [RFC5245] pacing algorithm requires that new STUN transactions be started no more frequently than 20 ms; with 30 RTP flows, which would add 600 ms of latency for candidate gathering alone. Moreover, having 30 persistent flows might lead to excessive consumption of NAT binding resources.

This document specifies a small number of modest extensions to SDP which are intended to reduce the transport impact of using a large number of flows. The general design philosophy is to maintain the existing SDP negotiation model (inventing as few new mechanisms as possible) while simply reducing the consumption of network resources.

1.1. Design Goals

The mechanism described in this document is meant to address the following goals:

1.1.1. Support for a large number of arbitrary sources

In cases such as a video conference, there may be dozens or hundreds of participants, each with their own audio and video sources. A participant may even want to browse conferences before joining one, meaning that there may be cases where there are many such conferences displayed simultaneously.

In these conferences, participants may have varying capabilities and therefore video resolutions. In addition, depending on conference

policy, user preference, and the desired UI, participants may be displayed in various layouts, including:

- o A single large main speaker with thumbnails for other participants
- o Multiple medium-sized main speakers, with or without thumbnails
- o Large slides + medium speaker, without thumbnails

These layouts can change dynamically, depending on the conference content and the preferences of the receiver. As such, there are not well-defined 'roles', that could be used to group sources into specific 'large' or 'thumbnail' categories. As such, the requirement we attempt to satisfy is support for sending and receiving up to hundreds of simultaneous, heterogeneous sources.

1.1.2. Support for fine-grained receiver control of sources

Since there may be large numbers of sources, which can be displayed in different layouts, it is imperative that the receiver can easily control which sources are received, and what resolution or quality is desired for each (for both audio and video). The receiver should also be able to prioritize the source it requests, so that if system limits or bandwidth force a reduction in quality, the sources chosen by the receiver as important will receive the best quality. These details must be exposed to the application via the API.

1.1.3. Glareless addition and removal of sources

Sources may come and go frequently, as is the case in a conference where various participants are presenting, or an interaction between multiple distributed conference servers. Because of this, it is desirable that sources can be added to SDP in a way that avoids signaling glare.

1.1.4. Interworking with other devices

When interacting with devices that do not apply all of the techniques described in this document, it must be possible to degrade gracefully to a usable basic experience. At a minimum, this basic experience should support setting up one audio stream and more than one video stream with existing videoconferencing equipment designed to establish a small number of simultaneous audio and video flows. For the remainder of this document, we will call these devices "legacy devices," although it should be understood that statements about legacy devices apply equally to future devices that elect not to use the techniques described in this document.

1.1.5. Avoidance of excessive port allocation

When there are dozens or hundreds of streams, it is desirable to avoid creating dozens or hundreds of transports, as empirical data shows a clear inverse relationship between number of transports (NAT bindings) and call success rate. While BUNDLE helps avoid creating large numbers of transports, it is also desirable to avoid creating large numbers of ports during call setup.

1.1.6. Simple binding of MediaStreamTrack to SDP

In WebRTC, each media source is identified by a MediaStreamTrack object. In order to ensure that the MSTs created by the sender show up at the receiver, each MST's id attribute needs to be reflected in SDP.

1.1.7. Support for RTX, FEC, simulcast, layered coding

For robust applications, techniques like RTX and FEC are used to protect media, and simulcast/layered coding can be used to provide support to heterogeneous receivers. It needs to be possible to support these techniques, allow the recipient to optionally use or not use them on a source-by-source basis; and for simulcast/layered scenarios, to control which simulcast streams or layers are received.

1.2. Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119].

This draft uses the API and terminology described in [webrtc-api].

5-tuple: A collection of the following values: source IP address, source transport port, destination IP address, destination transport port and transport protocol.

Transport-Flow: An transport 5 Tuple representing the UDP source and destination IP address and port over which RTP is flowing.

m-line: An SDP [RFC4566] media description identifier that starts with an "m=" field and conveys the following values: media type, transport port, transport protocol and media format descriptions.

Offer: An [RFC3264] SDP message generated by the participant who wishes to initiate a multimedia communication session. An Offer describes the participant's capabilities for engaging in a multimedia session.

Answer: An [RFC3264] SDP message generated by the participant in response to an Offer. An Answer describes the participant's capabilities in continuing with the multimedia session with in the constraints of the Offer.

This draft avoids using terms that implementors do not have a clear idea of exactly what they are - for example RTP Session.

1.3. Syntax Conventions

The SDP examples given in this document deviate from actual on-the-wire SDP notation in several ways. This is done to facilitate readability and to conform to the restrictions imposed by the RFC formatting rules. These deviations are as follows:

- o Any line that is indented (compared to the initial line in the SDP block) is a continuation of the preceding line. The line break and indent are to be interpreted as a single space character.
- o Empty lines in any SDP example are inserted to make functional divisions in the SDP clearer, and are not actually part of the SDP syntax.
- o Excepting the above two conventions, line endings are to be interpreted as <CR><LF> pairs (that is, an ASCII 13 followed by an ASCII 10).
- o Any text starting with the string "/" to the end of the line is inserted for the benefit of the reader, and is not actually part of the SDP syntax.

2. Solution Overview

At a high level, the solution described in this document can be summarized as follows:

1. Each media stream track is represented by its own unique m-line. This is a strict one-to-one mapping; a single media stream track cannot be spread across several m-lines, nor may a single m-line represent multiple media stream tracks. Note that this requires a modification to the way simulcast is currently defined by the individual draft [I-D.westerlund-avtcore-rtp-simulcast]. This does not preclude "application level" simulcasting; i.e., the creation of multiple media stream tracks from a single source.
2. Each m-line is marked with an a=ssrc attribute to correlate it with its RTP packets. Absent any other signaled extension, multiple SSRCs in a single m-line are interpreted as alternate

sources for the same media stream track: although senders can switch between the SSRCs as frequently as desired, only one should be sent at any given time.

3. Each m-line contains an MSID value to correlate it with a Media Stream ID and the Media Stream Track ID.
4. To minimize port allocation during a call, we rely on the BUNDLE [I-D.ietf-mmusic-sdp-bundle-negotiation] mechanism.
5. To reduce port allocation during call set-up, applications can mark less-critical media stream tracks in such a way that they will not require any port allocation, with the resulting property that such streams only work in the presence of the BUNDLE mechanism.
6. To address glare, we define a procedure via which partial offer/answer exchanges may take place. These exchanges operate on a single m-line at a time, rather than an entire SDP body. These operations are defined in a way that can completely avoid glare for stream additions and removals, and which reduces the chance of glare for changes to active streams. This approach requires all m-lines to contain an a=mid attribute.
7. All sources in a single bundle are required to contain identical attributes except for those that apply directly to a media stream track (such as label, msid, and resolution). See those attributes marked "IDENTICAL" in [I-D.nandakumar-mmusic-sdp-mux-attributes] for details.
8. RTP and RTCP streams are demultiplexed strictly based on their SSRC. However, to handle legacy cases and signaling/media races, correlation of streams to m-sections can use other mechanisms, as described in Section 3.2.

3. Detailed Description

3.1. Bundle-Only M-Lines

Even with the use of BUNDLE, it is expensive to allocate ICE candidates for a large number of m-lines. An offer can contain "bundle-only" m-lines which will be negotiated only by endpoints which implement this specification and ignored by other endpoints.

OPEN ISSUE: While it's probably pretty clear that this behavior will be controlled, in WebRTC, via a constraint, the "default" behavior -- that is, whether a line is "bundle-only" when there is no constraint present -- needs to be settled. This is a balancing

act between maximizing interoperability with legacy equipment by default or minimizing port use during call setup by default.

In order to offer such an m-line, the offerer does two things:

- o Sets the port in the m-line to 0. This indicates to old endpoints that the m-line is not to be negotiated.
- o Adds an a=bundle-only line. This indicates to new endpoints that the m-line is to be negotiated if (and only if) bundling is used.

An example offer that uses this feature looks like this:

```
v=0
o=- 20518 0 IN IP4 203.0.113.1
s=
t=0 0
c=IN IP4 203.0.113.1
a=group:BUNDLE S1 S2 S3
a=ice-ufrag:F7gI
a=ice-pwd:x9cml/YzichV2+XlhiMu8g
a=fingerprint:sha-1
    42:89:c5:c6:55:9d:6e:c8:e8:83:55:2a:39:f9:b6:eb:e9:a3:a9:e7

m=audio 54400 RTP/SAVPF 0 96
a=msid:ma ta
a=extmap:1 urn:ietf:params:rtp-hdrext:stream-correlator 20970
a=mid:1
a=rtpmap:0 PCMU/8000
a=rtpmap:96 opus/48000
a=ptime:20
a=sendrecv
a=rtcp-mux
a=ssrc:53280
a=candidate:0 1 UDP 2113667327 203.0.113.1 54400 typ host
a=candidate:1 2 UDP 2113667326 203.0.113.1 54401 typ host

m=video 0 RTP/SAVPF 96 97
a=msid:ma tb
a=extmap:1 urn:ietf:params:rtp-hdrext:stream-correlator 1714
a=mid:2
a=rtpmap:96 H264/90000
a=fmtp:96 profile-level-id=4d0028;packetization-mode=1
a=rtpmap:97 VP8/90000
a=sendrecv
a=rtcp-mux
a=ssrc:49152
a=bundle-only
```

```
m=video 0 RTP/SAVPF 96 97
a=msid:ma tc
a=extmap:1 urn:ietf:params:rtp-hdext:stream-correlator 57067
a=mid:3
a=rtpmap:96 H264/90000
a=fmtp:96 profile-level-id=4d0028;packetization-mode=1
a=rtpmap:97 VP8/90000
a=sendrecv
a=rtcp-mux
a=ssrc:32768
a=bundle-only
```

An old endpoint simply rejects the bundle-only m-lines by responding with a 0 port. (This isn't a normative statement, just a description of the way the older endpoints are expected to act.)

```
v=0
o=- 20518 0 IN IP4 203.0.113.1
s=
t=0 0
c=IN IP4 203.0.113.2
a=ice-ufrag:F7gI
a=ice-pwd:x9cml/YzichV2+XlhiMu8g
a=fingerprint:sha-1
    42:89:c5:c6:55:9d:6e:c8:e8:83:55:2a:39:f9:b6:eb:e9:a3:a9:e7
```

```
m=audio 55400 RTP/SAVPF 0 96
a=rtpmap:0 PCMU/8000
a=rtpmap:96 opus/48000
a=ptime:20
a=sendrecv
a=candidate:0 1 UDP 2113667327 203.0.113.2 55400 typ host
a=candidate:1 2 UDP 2113667326 203.0.113.2 55401 typ host
```

```
m=video 0 RTP/SAVPF 96 97
a=rtpmap:96 H264/90000
a=fmtp:96 profile-level-id=4d0028;packetization-mode=1
a=rtpmap:97 VP8/90000
a=sendrecv
```

```
m=video 0 RTP/SAVPF 96 97
a=rtpmap:96 H264/90000
a=fmtp:96 profile-level-id=4d0028;packetization-mode=1
a=rtpmap:97 VP8/90000
a=sendrecv
```

A new endpoint accepts the m-lines (both bundle-only and regular) by offering m-lines with a valid port, though this port may be duplicated as specified in Section 6 of [I-D.ietf-mmusic-sdp-bundle-negotiation]. For instance:

```
v=0
o=- 20518 0 IN IP4 203.0.113.2
s=
t=0 0
c=IN IP4 203.0.113.2
a=group:BUNDLE B1 B2 B3
a=ice-ufrag:F7gI
a=ice-pwd:x9cml/YzichV2+XlhiMu8g
a=fingerprint:sha-1
    42:89:c5:c6:55:9d:6e:c8:e8:83:55:2a:39:f9:b6:eb:e9:a3:a9:e7

m=audio 55400 RTP/SAVPF 0 96
a=msid:ma ta
a=extmap:1 urn:ietf:params:rtp-hdrext:stream-correlator 24860
a=mid:1
a=rtpmap:0 PCMU/8000
a=rtpmap:96 opus/48000
a=ptime:20
a=sendrecv
a=rtcp-mux
a=ssrc:35987
a=candidate:0 1 UDP 2113667327 203.0.113.2 55400 typ host

m=video 55400 RTP/SAVPF 96 97
a=msid:ma tb
a=extmap:1 urn:ietf:params:rtp-hdrext:stream-correlator 49811
a=mid:B2
a=rtpmap:96 H264/90000
a=fmtp:96 profile-level-id=4d0028;packetization-mode=1
a=rtpmap:97 VP8/90000
a=sendrecv
a=rtcp-mux
a=ssrc:9587
a=bundle-only

m=video 55400 RTP/SAVPF 96 97
a=msid:ma tc
a=extmap:1 urn:ietf:params:rtp-hdrext:stream-correlator 9307
a=mid:3
a=rtpmap:96 H264/90000
a=fmtp:96 profile-level-id=4d0028;packetization-mode=1
a=rtpmap:97 VP8/90000
a=sendrecv
```

```
a=rtcp-mux
a=ssrc:21389
a=bundle-only
```

Endpoints MUST NOT accept bundle-only m-lines if they are not part of an accepted bundle group.

3.2. Correlation

The system under consideration has three constructs the need to be mutually correlated for proper functioning: m-lines, media stream tracks, and RTP sources. These correlations are described in the following sections.

3.2.1. Correlating RTP Sources with m-lines

Sending several media streams over a single transport 5-tuple can pose challenges in the form of stream identification and correlation. This proposal maintains the use of SSRC as the single demultiplexing point for multiple streams sent between a transport 5-tuple.

Nominally, this correlation is performed by including a=ssrc attributes in the SDP. Under ideal circumstances, the use of a=ssrc in the SDP exchanged between endpoints is sufficient to correlate a demultiplexed stream to its m-line. However, at least three unrelated situations can arise that make correlation using an alternate mechanism advantageous.

During call establishment, circumstances may arise under which an endpoint can send an offer for a new stream, and begin receiving that media stream prior to receiving the SDP that correlates its SSRC to the m-line. For such cases, the endpoint will not know how to handle the media, and will most probably be forced to discard it. This can lead to media stream "clipping," which has a strongly negative impact on user experience. For audio streams, the "hello" of the answering party can be lost; for video streams, the initial I-frame can be lost, leading to corrupted or missing video until another I-frame is sent.

In the rare circumstance that a SSRC change for an existing media source is required, then any party that has changed its SSRC needs to inform the remote participants of the updated mapping, e.g. via a new SDP offer. Since any media sent with the new SSRC cannot be rendered until the new offer/answer exchange takes place, the clipping concern mentioned above exists here as well.

A different problem can arise when interoperating with legacy equipment. A number of circumstances can lead to the inability of a legacy endpoint to include SSRC information in its SDP. For example, in a system that decomposes signaling and media into different network devices, the protocol used to communicate between the boxes frequently will not include SSRC information, making it impossible to include in the SDP. If these devices choose to implement bundling, correlation of media streams to m-lines requires an alternate correlator.

These cases (and possibly other similar situations) can be ameliorated by using information in the media stream itself as a correlator to the SDP offer. If a packet arrives with an SSRC that is not yet associated with an m-line, we would ideally have some means of correlating it prior to the arrival of the answer.

The authors reiterate and emphasize that this technique is used solely for the purposes of correlation of an RTP stream to an SDP m-line after that stream has already been demultiplexed. Demultiplexing of multiple streams on a single transport address continues to be based on SSRC values.

3.2.1.1. RTP Header Extension Correlation

The preferred mechanism for such correlation is a new RTP header extension [RFC5285] that can be used near the beginning of an RTP stream to correlate RTP packets for which SSRC mapping information is not available. We propose that WebRTC implementations **MUST** implement this mechanism. We expect and that all other users of the BUNDLE extension **SHOULD** make use of it.

Although additional specification for this mechanism would be required for interoperability, the thumbnail sketch of such correlation is described below.

An implementation making use of this mechanism for local correlation includes an a=extmap attribute in the m-lines for which it wishes to use the mechanism. This attribute includes a mapping from the RTP header ID to the URL, as well as a 16-bit identifier (expressed as an integer) used for correlation; one such m-line would look like this:

```
m=audio 55400 RTP/SAVPF 0 96
a=msid:ma ta
a=extmap:1 urn:ietf:params:rtp-hdrex:stream-correlator 7582 // NEW
a=mid:1
a=rtpmap:0 PCMU/8000
a=rtpmap:96 opus/48000
a=ptime:20
```

```
a=sendrecv
a=rtcp-mux
a=ssrc:35987
a=candidate:0 1 UDP 2113667327 203.0.113.2 55400 typ host
```

The remote endpoint, if it supports this extension, MUST include an RTP header extension in several (on the order of 3 to 10) of the initial RTP packets in the stream. The value of this header extension will contain the correlator from the extmap line (in the above example, 7582).

3.2.1.2. Payload Type Correlation

To support implementations that cannot implement the RTP header extension described in Section 3.2.1.1 but which wish to use the BUNDLE mechanism, we allow an alternate (but less-preferred) means of correlation using payload type. This approach takes advantage of the fact that the offer contains payload types chosen by its creator, which will be present in any RTP received from the remote party. If these payload types are unique, then they can be used to reliably correlate incoming RTP streams to their m= lines.

Because of its inherent limitations, it is advisable to use other correlation techniques than PT multiplexing if at all possible. In order to accomplish this, we propose, for WebRTC, that use of this technique be controlled by an additional constraint passed to createOffer by the Web application.

If this constraint is set, the browser MUST behave as described in this section. If the constraint is not set, the browser MUST use identical PTs for the same codec values within each m-line bundle.

When such a constraint is present, implementations attempt to entirely exhaust the dynamic payload type numbering space before re-using a payload type within the scope of a local transport address. If such a constraint is present and the payload type space would ordinarily be exhausted within the scope of a local transport address, the implementation MAY (at its discretion) take any of the following actions:

1. Bind to multiple local transport addresses (using different BUNDLE groups) for the purpose of keeping the {payload type, transport address} combination unique.
2. Signal a failure to the application.

OPEN ISSUE: The above text specifically calls out "dynamic payload type numbering space," which consists of payload types 96 through 127. This is the most conservative range of payload types possible, with the greatest chance of exhaustion in normal use. In practice, it may make more sense to use a different range. The canonical description of payload type allocation strategies for RTP/AVP and its related profiles is given in section 3 of [RFC3551]. Roughly summarized: all values from 0 to 127 can be dynamically bound to codecs; codes from 96 to 127 should be preferred, followed by previously unassigned values, followed by statically assigned values. This is, however, modified by [RFC5761], which effectively eliminates payload types 64 through 95. Given these constraints, reasonable proposals (in order of most conservative to most aggressive) would include:

1. The dynamic range (96-127), for 32 usable payload types. This is meant to accommodate the most naive implementation possible, which is only capable of dynamically binding payload types in the dynamic range. Although not supported by current specifications, such limitations are suspected to exist in some modern RTP libraries.
2. The dynamic range (96-127), followed by the contiguous unassigned range (35-63), for 61 usable payload types. This approach is intended to accommodate those implementations that do not support dynamic binding for payload types for which an "audio/video" type is registered in the IANA registry.
3. The dynamic range (96-127) followed by all unassigned payload types (20-24, 27, 29, 30, and 35-63), for 69 usable payload types. This approach is intended to accommodate those implementations that are incapable of re-binding statically assigned payload types, while making use of all other available values.
4. The dynamic range (96-127) followed by all unassigned payload types (20-24, 27, 29, 30, and 35-63), followed by the statically assigned payload types (0-19, 25, 26, 28, and 31-34) for 96 usable payload types. This approach is most consistent with current IETF specifications, but is expected to cause interoperability issues with existing implementations (including libraries currently in use in early WebRTC implementations).

Note that the presence or absence of the aforementioned flag does not affect how incoming streams are correlated: if the RTP header extension for correlation is present, it is used in preference to the payload type. Conversely, if the flag is absent, and the RTP

contains no such header, then the payload type may be used for correlation inasmuch as a media line can be unambiguously identified. Of course, if the SSRC information has been made available in SDP prior to a need for stream correlation, then it can also be used for this purpose.

3.2.2. Correlating Media Stream Tracks with m-lines

Media Stream Tracks IDs are correlated with M-Lines directly by including an MSID in each m-line. The MSID also provides the Media Stream ID. (Note the format of the MSID used here is slightly different than what was proposed in the current MSID draft as that draft assumed multiple tracks in a single m-line and this proposal moves to a solution where there is a one to one relation between the Track and MSID. This work assumes the MSID draft will be updated to match the syntax used user which simply provides the value of the MediaStream ID and MediaStreamTrack ID on an "a=msid" line.)

3.2.3. Correlating Media Stream Tracks with RTP Sources

Media Stream Tracks are correlated with RTP sources transitively through the RTP-Source \Leftrightarrow M-Line \Leftrightarrow Media-Stream-Track relationship. Since the Media-Stream-Track \Leftrightarrow M-Line binding is established in the SDP offer, and the M-Line \Leftrightarrow RTP-Source binding can be handled as described in Section 3.2.1, none of the previously identified issues arise.

3.3. Handling of Simulcast, Forward Error Correction, and Retransmission Streams

Simulcast refers to taking a single capture (e.g., a camera), and encoding it multiple times at different resolutions and / or frame rates. For example, a device with a single HD camera may send one version of the video at full HD resolution, and a second version encoded at a low resolution. This would allow a video conferencing bridge to be able to send the high resolution copy to some destination and low resolution copy to other destinations without having to recode the video at the conference bridge.

Forward Error Correction (FEC) and Retransmission (RTX) streams are techniques that can provide stream robustness in the face of packet loss. These approaches frequently make use of different payload types and different SSRC values than the stream to which they apply.

In cases where a media source needs to correspond to more than one RTP flow, e.g. RTX, FEC, or simulcast, the a=ssrc-group [RFC5576] concept is used to create a grouping of SSRCs for a single media stream track. Each SSRC is declared using a=ssrc attributes, the

same MSID is shared between the SSRCs, and the a=ssrc-group attribute defines the behavior of the grouped SSRCs.

These groupings are used to perform demux of the incoming RTP streams and associate them (by SSRC) with their primary flows (modulo the behavior described in Section 3.2.1, if applicable). This multiplexing of RTX and FEC in a single RTP session is already well-defined; RTX SSRC-multiplexing behavior is defined in [RFC4588], and FEC SSRC-multiplexing behavior is defined in [RFC5956].

Note that both RTC and FEC also include SDP expressions that use different m= lines for the correction streams (cf. [RFC4588], section 8.7 and [RFC5956], section 4.2). These formats intend for correlation of streams to be based on transport addresses, which is inapplicable for bundled media streams. Our specific proposal is: (1) bundling implementations will never generate such a format; and (2) bundling implementations MAY choose to accept SDP in such a format or MAY simply reject the repair streams and proceed as if the indicated repair format is not supported.

For multi-resolution simulcast, we can create a similar ssrc-group, and adapt the imageattr attribute defined in [RFC6236] for the a=ssrc line attribute to indicate the send resolution for a given simulcast stream. (This will be added to [I-D.westerlund-avtcore-rtp-simulcast], as outlined in Section 2, bullet 1). In the example below, the SDP advertises a simulcast of a camera source at two different resolutions, as well as a screen-share source that supports RTX; a=ssrc-group is used to correlate the different SSRCs as part of a single media source.

Note that a characteristic of this approach is that it does not allow for independently setting attributes for simulcast, FEC, and RTX streams aside from those in fmlt. In particular, attributes such as ptime and framerate are shared between the streams that are grouped together for a simulcast group.

```
m=video 62537 RTP/SAVPF 96          // main video
a=msid:ma ta
a=extmap:1 urn:ietf:params:rtp-hdrext:stream-correlator 15955
a=mid:1
a=rtpmap:96 VP8/90000
a=sendrecv
a=rtcp-mux
a=ssrc:29154 imageattr:96 [x=1280,y=720]
a=ssrc:47182 imageattr:96 [x=640,y=360]
a=ssrc-group:SIMULCAST 29154 47182
```

```
m=video 0 RTP/SAVPF 96 97          // slide video
```

```
a=msid:ma tb
a=extmap:1 urn:ietf:params:rtp-hdrext:stream-correlator 26267
a=mid:2
a=rtpmap:96 VP8/90000
a=rtpmap:97 rtx/90000
a=sendrecv
a=rtcp-mux
a=fmtp:97 apt=96;rtx-time=3000
a=ssrc:45982
a=ssrc:9827
a=ssrc-group:FID 45982 9827          // FID provides SSRC correlation
a=bundle-only
```

Providing explicit resolutions on a per-SSRC basis for SIMULCAST groupings allows an intermediary (such as a Media Translator [RFC5117]) to be able to select an appropriate SIMULCAST layer without inspecting the media stream, which could otherwise require decrypting and possibly partially decoding media packets.

3.4. Glare Minimization

To allow for guaranteed glareless addition and removal of streams, and to provide for a reduced chance of glare in stream attribute changes, we propose a technique that allows for m-lines to be changed independently of each other.

The proposal for doing so is performed using "partial offers" and "partial answers." Using this technique has two key prerequisites: (1) all offer/answer exchanges in the session have contained "a=mid" attributes [RFC5888] for each m-line, and (2) both sides are known to support the partial offer/answer technique (either because they are part of a single domain of control, or because use of this technique has been explicitly signaled).

The use of a partial SDP body will be explicitly signaled, e.g., using a different MIME type for SIP, or using a different "type" for the WebRTC API.

The authors recognize that further formal definition would be required to describe this technique. These are left as future study for the appropriate venues, such as the W3C WebRTC WG and the SIPCORE WG. As a thumbnail sketch: For WebRTC, we envision that we would add a new constraint to `createOffer`, requesting that a partial offer be generated (if possible). The resulting `RTCSessionDescription` would contain only the m-lines that have changed since the most recent offer/answer exchange, and would have a type of "partialOffer." When `createAnswer` is called after receipt of a `partialOffer`, it would

create a partialAnswer, containing only the m-lines referenced in the partial offer, that can be provided to the remote party.

3.4.1. Adding a Stream

To add a stream glarelessly, a party creates a "partial offer" consisting of an m-line and all of its attributes. This m-line contains an mid that has not yet been used in the session. To reduce the chance of collision to effectively zero, this mid MUST contain at least 32 characters chosen randomly from full set of 79 characters allowed in a token. It then sends this partial offer to the remote party and awaits a partial answer.

Upon receipt of a partial offer, an implementation examines the mid in it. If the mid does not match any existing mid in the session, then it represents a new media stream. Assuming the recipient does not have an outstanding, unanswered partial offer that also adds a stream, this new m-line is simply appended to the end of the existing session description, the SDP version is incremented by one, and a partial answer is created. This partial answer consists of an m-line and its attributes, and has an mid matching the one from the partial offer.

If the recipient of a partial offer that contains a new mid has also sent a partial offer adding a new stream to the session, then ambiguity can arise regarding the canonical ordering of m-lines within the session. In this situation, both partial offer/answer exchanges are allowed to complete independently (as no fundamental data glare has occurred). However, the order in which they are appended to the session description is synchronized by performing a lexical comparison between each m-lines mid attribute: the m-line with the lexically smaller mid attribute is appended first, while the other m-line is appended after it.

3.4.2. Changing a Stream

Partial offers may also be generated for modification of an existing stream. In this case, the mid in the partial offer will match an existing mid in the session description.

Upon receipt of a partial offer, an implementation examines the mid in it. If the mid matches any existing mid in the session, then it represents a modification to that m-line. Assuming the recipient does not have an outstanding, unanswered partial offer that also modifies that exact same stream, this m-line is treated as an independent renegotiation of that stream (only). The SDP version is incremented by one, and a partial answer is created. This partial answer consists of an m-line and its attributes, and has an mid matching the one from the partial offer.

If the recipient of a partial offer that contains an existing mid has also sent a partial offer to change that exact same stream, and neither the received nor the sent partial offer contains an "a=inactive" attribute, then a legitimate glare condition has arisen. Normal glare recovery procedures -- e.g., using a tie-breaker token or a back-off timer -- must be engaged to resolve the conflict.

3.4.3. Removing a Stream

To remove a stream in a way that eliminates the chance of glare, an implementation generates a new partial offer, with an mid matching the m-line it wants to remove. This partial offer contains an a=inactive attribute, indicating that the stream is being deactivated.

If the recipient of a partial offer that contains an existing mid has also sent a partial offer to change that exact same stream, and either one of the received or the sent partial offer contains an "a=inactive" attribute, then the stream is deactivated. At this point, both partial offers are discarded, the corresponding m-line in the session is modified by changing any a=sendonly, a=recvonly, or a=sendrecv attribute to a=inactive (or, if no such attributes are present, an a=inactive attribute is added), and a partial answer is generated representing this single change.

3.5. Negotiation of Stream Ordinality

Within advanced applications, circumstances can easily arise in which the party creating the offer does not know ahead of time the number of streams the remote party will desire. For example, in a meet-me videoconference application that sends a separate stream for each participant, a client creating an offer to send to the conference focus does not necessarily know how many video streams to indicate in its SDP. Although this can be potentially be solved in an application-specific way (e.g., by always offering the maximum number of streams known to be supported by the application), this is not always desirable or even possible.

To address this situation, a three-way handshake can be employed.

	Calling Party	Called Party
Calling party creates offer with audio and video. Since it does not know how many streams, it "guesses" one of each.	--- Offer (1 video, 1 audio) -->	
[Call starts now]	<-- Answer (1 video, 1 audio) -- <-- Offer (8 video, 1 audio) ---	Called party desires eight video streams. So it creates an answer for the "one of each" offer and an offer for the total number of streams it wants.
Calling party answers for all eight video streams.	-- Answer (8 video, 1 audio) -->	

The first leg of this handshake consists of an offer sent by the calling party. This offer contains at least one m-line for each type of media the offerer wishes to use in the session. The authors draw special attention to the clause "at least" in the preceding sentence: offerers can use external knowledge, hinting, or simple guesses to offer additional m-lines.

Upon receipt of such an offer, the called party examines the number of streams of each media type being requested. If the number of streams is equal to or greater than the number of total streams that the called party desires at this time, it simply forms an answer to complete the offer/answer exchange [RFC3264], and the call is set up.

On the other hand, if the called party determines that more streams are necessary than are indicated in the initial offer, it responds by first creating an answer with the same number of streams as were present in the initial offer. It additionally creates a new answer that contains the number of streams it desires. This answer/offer pair is sent to the calling party, in a single message if supported by the signaling protocol (as will frequently be the case for WebRTC), or in two consecutive messages in a way that guarantees in-order delivery.

When the calling party receives this answer, it establishes the session, and all of the streams that were negotiated in this first offer/answer exchange. So, within a single signaling round trip, the initial set of streams (consisting of those the calling party included in its initial offer) are established.

When the calling party receives the subsequent offer, it comprises the beginning of a completely new RFC 3264 offer/answer exchange [RFC3264]. The calling party creates an answer that fully describes all of the streams in the session, and sends it to the called party. Consequently, within 1.5 round trips, the entire call is set up and all associated streams can be sent and received.

Of particular note is the fact that this model does not deviate from normal RFC3264 offer/answer handling, even when three-way handshaking is necessary.

3.6. Compatibility with Legacy uses

Due to the fact that this approach re-uses existing SDP constructs for indicating parameters in a media section, it remains compatible with legacy clients. Of particular note is the handling of "bundle-only" media sections, described in Section 3.1. Offers generated by an RTCWEB client and sent to a legacy client will simply negotiate those media the RTCWEB client did not use the "bundle-only" extension with. This allows RTCWEB clients to select which media streams are important for interoperability with legacy clients (by not making them bundle-only), and which ones are not. Offers generated by legacy clients will simply omit any bundle-related attributes, and the RTCWEB client will be able to process the SDP otherwise identically to the SDP received from RTCWEB clients: each m-line represents a different media stream, and contains a description of that stream in a syntax identical to the syntax used between RTCWEB clients.

With the bundle-only approach, only those streams that are "important for interoperability" will require allocation of ports and ICE exchanges. By doing so, working with non-multiplexing clients is

enabled without requiring excess resource allocation for those streams that are not critical for proper user experience.

4. Examples

In all of these examples, there are many lines that are wrapped due to column width limitation. It should be understood these lines are not wrapped in the real SDP.

The convention used for IP addresses in this drafts is that private IP behind a NAT come from 192.0.2.0/24, the public side of a NAT comes from 198.51.100.0/24 and the TURN servers have addresses from 203.0.113.0/24. Typically the offer has an IP ending in .1 and the answer has an IP ending in .2.

The examples do not include all the parts of SDP that are used in RTCWeb (See [I-D.ietf-rtcweb-rtp-usage]) as that makes the example unwieldy to read but instead focuses on showing the parts that are key for the multiplexing.

4.1. Simple example with one audio and one video

The following SDP shows an offer that offers one audio stream and one video steam with both a STUN and TURN address. It also shows unique payload across the audio and video m=lines for the Answerer that does not support BUNDLE semantics.

```
v=0
o=- 20518 0 IN IP4 198.51.100.1
s=
t=0 0
c=IN IP4 203.0.113.1
a=ice-ufrag:074c6550
a=ice-pwd:a28a397a4c3f31747dlee3474af08a068
a=fingerprint:sha-1
          99:41:49:83:4a:97:0e:1f:ef:6d:f7:c9:c7:70:9d:1f:66:79:a8:07
a=group:BUNDLE m1 m2

m=audio 56600 RTP/SAVPF 0 109
a=msid:ma ta
a=extmap:1 urn:ietf:params:rtp-hdext:stream-correlator 33424
a=mid:m1
a=ssrc:53280
a=rtpmap:0 PCMU/8000
a=rtpmap:109 opus/48000
a=ptime:20
a=sendrecv
a=rtcp-mux
```

```
a=candidate:0 1 UDP 2113667327 192.0.2.1 54400 typ host
a=candidate:1 2 UDP 2113667326 192.0.2.1 54401 typ host
a=candidate:0 1 UDP 694302207 198.51.100.1 55500 typ srflx raddr
    192.0.2.1 rport 54400
a=candidate:1 2 UDP 169430220 198.51.100.1 55501 typ srflx raddr
    192.0.2.1 rport 54401
a=candidate:0 1 UDP 73545215 203.0.113.1 56600 typ relay raddr
    192.0.2.1 rport 54400
a=candidate:1 2 UDP 51989708 203.0.113.1 56601 typ relay raddr
    192.0.2.1 rport 54401

m=video 56602 RTP/SAVPF 99 120
a=msid:ma tb
a=extmap:1 urn:ietf:params:rtp-hdrext:stream-correlator 35969
a=mid:m2
a=ssrc:49843
a=rtpmap:99 H264/90000
a=fmtp:99 profile-level-id=4d0028;packetization-mode=1
a=rtpmap:120 VP8/90000
a=sendrecv
a=rtcp-mux
a=candidate:3 1 UDP 2113667327 192.0.2.1 54402 typ host
a=candidate:4 2 UDP 2113667326 192.0.2.1 54403 typ host
a=candidate:3 1 UDP 694302207 198.51.100.1 55502 typ srflx raddr
    192.0.2.1 rport 54402
a=candidate:4 2 UDP 169430220 198.51.100.1 55503 typ srflx raddr
    192.0.2.1 rport 54403
a=candidate:3 1 UDP 73545215 203.0.113.1 56602 typ relay raddr
    192.0.2.1 rport 54402
a=candidate:4 2 UDP 51989708 203.0.113.1 56603 typ relay raddr
    192.0.2.1 rport 54403
```

The following shows an answer to the above offer from a device that does not support bundle or rtcp-mux.

```
v=0
o=- 16833 0 IN IP4 198.51.100.2
s=
t=0 0
c=IN IP4 203.0.113.2
a=ice-ufrag:c300d85b
a=ice-pwd:de4e99bd291c325921d5d47efbabd9a2
a=fingerprint:sha-1
    91:41:49:83:4a:97:0e:1f:ef:6d:f7:c9:c7:70:9d:1f:66:79:a8:03

m=audio 60600 RTP/SAVPF 109
a=msid:ma ta
```

```
a=rtpmap:109 opus/48000
a=ptime:20
a=sendrecv
a=candidate:0 1 UDP 2113667327 192.0.2.2 60400 typ host
a=candidate:1 2 UDP 2113667326 192.0.2.2 60401 typ host
a=candidate:0 1 UDP 1694302207 198.51.100.2 60500 typ srflx raddr
    192.0.2.2 rport 60400
a=candidate:1 2 UDP 1694302206 198.51.100.2 60501 typ srflx raddr
    192.0.2.2 rport 60401
a=candidate:0 1 UDP 73545215 203.0.113.2 60600 typ relay raddr
    192.0.2.1 rport 60400
a=candidate:1 2 UDP 51989708 203.0.113.2 60601 typ relay raddr
    192.0.2.1 rport 60401

m=video 60602 RTP/SAVPF 99
a=msid:ma tb
a=rtpmap:99 H264/90000
a=fmtp:99 profile-level-id=4d0028;packetization-mode=1
a=sendrecv
a=candidate:2 1 UDP 2113667327 192.0.2.2 60402 typ host
a=candidate:3 2 UDP 2113667326 192.0.2.2 60403 typ host
a=candidate:2 1 UDP 694302207 198.51.100.2 60502 typ srflx raddr
    192.0.2.2 rport 60402
a=candidate:3 2 UDP 169430220 198.51.100.2 60503 typ srflx raddr
    192.0.2.2 rport 60403
a=candidate:2 1 UDP 73545215 203.0.113.2 60602 typ relay raddr
    192.0.2.2 rport 60402
a=candidate:3 2 UDP 51989708 203.0.113.2 60603 typ relay raddr
    192.0.2.2 rport 60403
```

The following shows answer to the above offer from a device that does support bundle.

```
v=0
o=- 16833 0 IN IP4 198.51.100.2
s=
t=0 0
c=IN IP4 203.0.113.2
a=ice-ufrag:c300d85b
a=ice-pwd:de4e99bd291c325921d5d47efbabd9a2
a=fingerprint:sha-1
    91:41:49:83:4a:97:0e:1f:ef:6d:f7:c9:c7:70:9d:1f:66:79:a8:03
a=group:BUNDLE m1 m2

m=audio 60600 RTP/SAVPF 109
a=msid:ma ta
a=extmap:1 urn:ietf:params:rtp-hdrext:stream-correlator 39829
```

```
a=mid:m1
a=ssrc:35856
a=rtpmap:109 opus/48000
a=ptime:20
a=sendrecv
a=rtcp-mux
a=candidate:0 1 UDP 2113667327 192.0.2.2 60400 typ host
a=candidate:0 1 UDP 1694302207 198.51.100.2 60500 typ srflx raddr
    192.0.2.2 rport 60400
a=candidate:0 1 UDP 73545215 203.0.113.2 60600 typ relay raddr
    192.0.2.1 rport 60400

m=video 60600 RTP/SAVPF 99
a=msid:ma tb
a=extmap:1 urn:ietf:params:rtp-hdrext:stream-correlator 45163
a=mid:m2
a=ssrc:2638
a=rtpmap:99 H264/90000
a=fmtp:99 profile-level-id=4d0028;packetization-mode=1
a=sendrecv
a=rtcp-mux
a=candidate:3 1 UDP 2113667327 192.0.2.2 60400 typ host
a=candidate:3 1 UDP 694302207 198.51.100.2 60500 typ srflx raddr
    192.0.2.2 rport 60400
a=candidate:3 1 UDP 73545215 203.0.113.2 60600 typ relay raddr
    192.0.2.2 rport 60400
```

4.2. Multiple Videos

Simple example showing an offer with one audio stream and two video streams.

```
v=0
o=- 20518 0 IN IP4 198.51.100.1
s=
t=0 0
c=IN IP4 203.0.113.1
a=ice-ufrag:F7gI
a=ice-pwd:x9cml/YzichV2+XlhiMu8g
a=fingerprint:sha-1
    42:89:c5:c6:55:9d:6e:c8:e8:83:55:2a:39:f9:b6:eb:e9:a3:a9:e7
a=group:BUNDLE m1 m2 m3

m=audio 56600 RTP/SAVPF 0 96
a=msid:ma ta
a=extmap:1 urn:ietf:params:rtp-hdrext:stream-correlator 47434
a=mid:m1
```

```
a=ssrc:32385
a=rtpmap:0 PCMU/8000
a=rtpmap:96 opus/48000
a=ptime:20
a=sendrecv
a=rtcp-mux
a=candidate:0 1 UDP 2113667327 192.0.2.1 54400 typ host
a=candidate:1 2 UDP 2113667326 192.0.2.1 54401 typ host
a=candidate:0 1 UDP 694302207 198.51.100.1 55500 typ srflx raddr
192.0.2.1 rport 54400
a=candidate:1 2 UDP 169430220 198.51.100.1 55501 typ srflx raddr
192.0.2.1 rport 54401
a=candidate:0 1 UDP 73545215 203.0.113.1 56600 typ relay raddr
192.0.2.1 rport 54400
a=candidate:1 2 UDP 51989708 203.0.113.1 56601 typ relay raddr
192.0.2.1 rport 54401

m=video 56602 RTP/SAVPF 96 98
a=msid:ma tb
a=extmap:1 urn:ietf:params:rtp-hdrext:stream-correlator 22705
a=mid:m2
a=ssrc:43985
a=rtpmap:96 H264/90000
a=fmtp:96 profile-level-id=4d0028;packetization-mode=1
a=rtpmap:98 VP8/90000
a=sendrecv
a=rtcp-mux
a=candidate:2 1 UDP 2113667327 192.0.2.1 54402 typ host
a=candidate:3 2 UDP 2113667326 192.0.2.1 54403 typ host
a=candidate:2 1 UDP 694302207 198.51.100.1 55502 typ srflx raddr
192.0.2.1 rport 54402
a=candidate:3 2 UDP 169430220 198.51.100.1 55503 typ srflx raddr
192.0.2.1 rport 54403
a=candidate:2 1 UDP 73545215 203.0.113.1 56602 typ relay raddr
192.0.2.1 rport 54402
a=candidate:3 2 UDP 51989708 203.0.113.1 56603 typ relay raddr
192.0.2.1 rport 54403
a=ssrc:11111 cname:45:5f:fe:cb:81:e9

m=video 56604 RTP/SAVPF 96 98
a=msid:ma tc
a=extmap:1 urn:ietf:params:rtp-hdrext:stream-correlator 64870
a=mid:m3
a=ssrc:54269
a=rtpmap:96 H264/90000
a=fmtp:96 profile-level-id=4d0028;packetization-mode=1
a=rtpmap:98 VP8/90000
a=sendrecv
```

```
a=rtcp-mux
a=candidate:4 1 UDP 2113667327 192.0.2.1 54404 typ host
a=candidate:5 2 UDP 2113667326 192.0.2.1 54405 typ host
a=candidate:4 1 UDP 694302207 198.51.100.1 55504 typ srflx raddr
    192.0.2.1 rport 54404
a=candidate:5 2 UDP 169430220 198.51.100.1 55505 typ srflx raddr
    192.0.2.1 rport 54405
a=candidate:4 1 UDP 73545215 203.0.113.1 56604 typ relay raddr
    192.0.2.1 rport 54404
a=candidate:5 2 UDP 51989708 203.0.113.1 56605 typ relay raddr
    192.0.2.1 rport 54405
a=ssrc:22222 cname:45:5f:fe:cb:81:e9
```

4.3. Many Videos

This section adds three video streams and one audio. The video streams are sent in such a way that they are only accepted if the far side supports bundle using the "bundle only" approach described in Section 3.1. The video streams also use the same payload types so it will not be possible to demux the video streams from each other without using the SSRC values.

```
v=0
o=- 20518 0 IN IP4 198.51.100.1
s=
t=0 0
c=IN IP4 203.0.113.1
a=ice-ufrag:F7gI
a=ice-pwd:x9cml/YzichV2+XlhiMu8g
a=fingerprint:sha-1
    42:89:c5:c6:55:9d:6e:c8:e8:83:55:2a:39:f9:b6:eb:e9:a3:a9:e7
a=group:BUNDLE m0 m1 m2 m3

m=audio 56600 RTP/SAVPF 0 96
a=msid:ma ta
a=extmap:1 urn:ietf:params:rtp-hdrext:stream-correlator 6614
a=mid:m0
a=ssrc:12359
a=rtpmap:0 PCMU/8000
a=rtpmap:96 opus/48000
a=ptime:20
a=sendrecv
a=rtcp-mux
a=ssrc:12359 cname:45:5f:fe:cb:81:e9
a=candidate:0 1 UDP 2113667327 192.0.2.1 54400 typ host
a=candidate:1 2 UDP 2113667326 192.0.2.1 54401 typ host
a=candidate:0 1 UDP 694302207 198.51.100.1 55500 typ srflx raddr
```



```
192.0.2.1 rport 54400
a=candidate:1 2 UDP 169430220 198.51.100.1 55501 typ srflx raddr
192.0.2.1 rport 54401
a=candidate:0 1 UDP 73545215 203.0.113.1 56600 typ relay raddr
192.0.2.1 rport 54400
a=candidate:1 2 UDP 51989708 203.0.113.1 56601 typ relay raddr
192.0.2.1 rport 54401
```

```
m=video 0 RTP/SAVPF 96 98
a=msid:ma tb
a=extmap:1 urn:ietf:params:rtp-hdrext:stream-correlator 24147
a=mid:m1
a=ssrc:26989
a=rtpmap:96 H264/90000
a=fmtp:96 profile-level-id=4d0028;packetization-mode=1
a=rtpmap:98 VP8/90000
a=sendrecv
a=rtcp-mux
a=bundle-only
a=ssrc:26989 cname:45:5f:fe:cb:81:e9
```

```
m=video 0 RTP/SAVPF 96 98
a=msid:ma tc
a=extmap:1 urn:ietf:params:rtp-hdrext:stream-correlator 33989
a=mid:m2
a=ssrc:32986
a=rtpmap:96 H264/90000
a=fmtp:96 profile-level-id=4d0028;packetization-mode=1
a=rtpmap:98 VP8/90000
a=sendrecv
a=rtcp-mux
a=bundle-only
a=ssrc:32986 cname:45:5f:fe:cb:81:e9
```

```
m=video 0 RTP/SAVPF 96 98
a=msid:ma td
a=extmap:1 urn:ietf:params:rtp-hdrext:stream-correlator 61408
a=mid:m3
a=ssrc:46986
a=rtpmap:96 H264/90000
a=fmtp:96 profile-level-id=4d0028;packetization-mode=1
a=rtpmap:98 VP8/90000
a=sendrecv
a=rtcp-mux
a=bundle-only
a=ssrc:46986 cname:45:5f:fe:cb:81:e9
```

4.4. Multiple Videos with Simulcast

This section shows an offer with with audio and two video each of which can send it two resolutions as described in Section 3.3. One video stream supports VP8, while the other supports H.264. All the video is bundle-only. Note that the use of different codec-specific parameters causes two different payload types to be used.

```
v=0
o=- 20518 0 IN IP4 198.51.100.1
s=
t=0 0
c=IN IP4 203.0.113.1
a=ice-ufrag:F7gI
a=ice-pwd:x9cml/YzichV2+XlhiMu8g
a=fingerprint:sha-1
    42:89:c5:c6:55:9d:6e:c8:e8:83:55:2a:39:f9:b6:eb:e9:a3:a9:e7
a=group:BUNDLE m0 m1 m2

m=audio 56600 RTP/SAVPF 0 96
a=msid:ma ta
a=extmap:1 urn:ietf:params:rtp-hdrext:stream-correlator 31727
a=mid:m0
a=rtpmap:0 PCMU/8000
a=rtpmap:96 opus/48000
a=ptime:20
a=sendrecv
a=rtcp-mux
a=candidate:0 1 UDP 2113667327 192.0.2.1 54400 typ host
a=candidate:1 2 UDP 2113667326 192.0.2.1 54401 typ host
a=candidate:0 1 UDP 694302207 198.51.100.1 55500 typ srflx raddr
    192.0.2.1 rport 54400
a=candidate:1 2 UDP 169430220 198.51.100.1 55501 typ srflx raddr
    192.0.2.1 rport 54401
a=candidate:0 1 UDP 73545215 203.0.113.1 56600 typ relay raddr
    192.0.2.1 rport 54400
a=candidate:1 2 UDP 51989708 203.0.113.1 56601 typ relay raddr
    192.0.2.1 rport 54401

m=video 0 RTP/SAVPF 96 100
a=msid:ma tb
a=extmap:1 urn:ietf:params:rtp-hdrext:stream-correlator 41664
b=AS:1756
a=mid:m1
a=rtpmap:96 VP8/90000
a=ssrc-group:SIMULCAST 58949 28506
a=ssrc:58949 imageattr:96 [x=1280,y=720]
a=ssrc:28506 imageattr:96 [x=640,y=480]
```

```
a=sendrecv
a=rtcp-mux
a=bundle-only

m=video 0 RTP/SAVPF 96 100
a=msid:ma tc
a=extmap:1 urn:ietf:params:rtp-hdext:stream-correlator 14460
b=AS:1756
a=mid:m2
a=rtpmap:96 H264/90000
a=fmtp:96 profile-level-id=4d0028;packetization-mode=1;max-fr=30
a=rtpmap:100 H264/90000
a=fmtp:100 profile-level-id=4d0028;packetization-mode=1;max-fr=15
a=ssrc-group:SIMULCAST 18875 54986
a=ssrc:18875
a=ssrc:54986
a=sendrecv
a=rtcp-mux
a=bundle-only
```

4.5. Video with Simulcast and RTX

This section shows an SDP offer that has an audio and a single video stream. The video stream that is simulcast at two resolutions and has [RFC4588] style re-transmission flows.

```
v=0
o=- 20518 0 IN IP4 198.51.100.1
s=
t=0 0
c=IN IP4 203.0.113.1
a=ice-ufrag:F7gI
a=ice-pwd:x9cml/YzichV2+XlhiMu8g
a=fingerprint:sha-1
    42:89:c5:c6:55:9d:6e:c8:e8:83:55:2a:39:f9:b6:eb:e9:a3:a9:e7
a=group:BUNDLE m0 m1

m=audio 56600 RTP/SAVPF 0 96
a=msid:ma ta
a=extmap:1 urn:ietf:params:rtp-hdext:stream-correlator 42123
a=mid:m0
a=rtpmap:0 PCMU/8000
a=rtpmap:96 opus/48000
a=ptime:20
a=sendrecv
a=rtcp-mux
a=candidate:0 1 UDP 2113667327 192.0.2.1 54400 typ host
```

```
a=candidate:1 2 UDP 2113667326 192.0.2.1 54401 typ host
a=candidate:0 1 UDP 694302207 198.51.100.1 55500 typ srflx raddr
    192.0.2.1 rport 54400
a=candidate:1 2 UDP 169430220 198.51.100.1 55501 typ srflx raddr
    192.0.2.1 rport 54401
a=candidate:0 1 UDP 73545215 203.0.113.1 56600 typ relay raddr
    192.0.2.1 rport 54400
a=candidate:1 2 UDP 51989708 203.0.113.1 56601 typ relay raddr
    192.0.2.1 rport 54401

m=video 0 RTP/SAVPF 96 101
a=msid:ma tb
a=extmap:1 urn:ietf:params:rtp-hdrext:stream-correlator 60725
b=AS:2500
a=mid:m1
a=rtpmap:96 VP8/90000
a=rtpmap:101 rtx/90000
a=fmtp:101 apt=96;rtx-time=3000
a=ssrc-group:SIMULCAST 78909 43567
a=ssrc-group:FID 78909 56789
a=ssrc-group:FID 43567 13098
a=ssrc:78909
a=ssrc:43567
a=ssrc:13098
a=ssrc:56789
a=sendrecv
a=rtcp-mux
a=bundle-only
```

4.6. Video with Simulcast and FEC

This section shows an SDP offer that has an audio and a single video stream. The video stream that is simulcast at two resolutions and has [RFC5956] style FEC flows.

```
v=0
o=- 20518 0 IN IP4 198.51.100.1
s=
t=0 0
c=IN IP4 203.0.113.1
a=ice-ufrag:F7gI
a=ice-pwd:x9cml/YzichV2+XlhiMu8g
a=fingerprint:sha-1
    42:89:c5:c6:55:9d:6e:c8:e8:83:55:2a:39:f9:b6:eb:e9:a3:a9:e7
a=group:BUNDLE m0 m1
```

```
m=audio 56600 RTP/SAVPF 0 96
a=msid:ma ta
a=extmap:1 urn:ietf:params:rtp-hdrext:stream-correlator 42123
a=mid:m0
a=rtpmap:0 PCMU/8000
a=rtpmap:96 opus/48000
a=ptime:20
a=sendrecv
a=rtcp-mux
a=candidate:0 1 UDP 2113667327 192.0.2.1 54400 typ host
a=candidate:1 2 UDP 2113667326 192.0.2.1 54401 typ host
a=candidate:0 1 UDP 694302207 198.51.100.1 55500 typ srflx raddr
    192.0.2.1 rport 54400
a=candidate:1 2 UDP 169430220 198.51.100.1 55501 typ srflx raddr
    192.0.2.1 rport 54401
a=candidate:0 1 UDP 73545215 203.0.113.1 56600 typ relay raddr
    192.0.2.1 rport 54400
a=candidate:1 2 UDP 51989708 203.0.113.1 56601 typ relay raddr
    192.0.2.1 rport 54401

m=video 0 RTP/SAVPF 96 101
a=msid:ma tb
a=extmap:1 urn:ietf:params:rtp-hdrext:stream-correlator 60725
b=AS:2500
a=mid:m1
a=rtpmap:96 VP8/90000
a=rtpmap:101 ld-interleaved-parityfec/90000
a=fmtp:96 max-fr=30;max-fs=8040
a=fmtp:101 L=5; D=10; repair-window=200000
a=ssrc-group:SIMULCAST 56780 34511
a=ssrc-group:FEC-FR 56780 48675
a=ssrc-group:FEC-FR 34511 21567
a=ssrc:56780
a=ssrc:34511
a=ssrc:21567
a=ssrc:48675
a=sendrecv
a=rtcp-mux
a=bundle-only
```

4.7. Video with Layered Coding

This section shows an SDP offer that has an audio and a single video stream. The video stream that is layered coding at 3 different resolutions based on [RFC5583]. The video m=lines shows 3 streams with last stream (payload 100) dependent on streams with payload 96 and 97 for decoding.

```
v=0
o=- 20518 0 IN IP4 198.51.100.1
s=
t=0 0
c=IN IP4 203.0.113.1
a=ice-ufrag:F7gI
a=ice-pwd:x9cml/YzichV2+XlhiMu8g
a=fingerprint:sha-1
    42:89:c5:c6:55:9d:6e:c8:e8:83:55:2a:39:f9:b6:eb:e9:a3:a9:e7
a=group:BUNDLE m0 m1

m=audio 56600 RTP/SAVPF 0 96
a=msid:ma ta
a=extmap:1 urn:ietf:params:rtp-hdrext:stream-correlator 42123
a=mid:m0
a=rtpmap:0 PCMU/8000
a=rtpmap:96 opus/48000
a=ptime:20
a=sendrecv
a=rtcp-mux
a=candidate:0 1 UDP 2113667327 192.0.2.1 54400 typ host
a=candidate:1 2 UDP 2113667326 192.0.2.1 54401 typ host
a=candidate:0 1 UDP 694302207 198.51.100.1 55500 typ srflx raddr
    192.0.2.1 rport 54400
a=candidate:1 2 UDP 169430220 198.51.100.1 55501 typ srflx raddr
    192.0.2.1 rport 54401
a=candidate:0 1 UDP 73545215 203.0.113.1 56600 typ relay raddr
    192.0.2.1 rport 54400
a=candidate:1 2 UDP 51989708 203.0.113.1 56601 typ relay raddr
    192.0.2.1 rport 54401

m=video 0 RTP/SAVPF 96 97 100
a=msid:ma tb
a=extmap:1 urn:ietf:params:rtp-hdrext:stream-correlator 60725
b=AS:2500
a=mid:m1
a=rtpmap:96 H264/90000
a=fmtp:96 max-fr=30;max-fs=8040
a=rtpmap:97 H264/90000
a=fmtp:97 max-fr=15;max-fs=1200
a=rtpmap:100 H264-SVC/90000
a=fmtp:100 max-fr=30;max-fs=8040
```

```
a=depend:100 lay m1:96,97;  
a=ssrc:48970  
a=ssrc:90898  
a=ssrc:66997  
a=sendrecv  
a=rtcp-mux  
a=bundle-only
```

5. Security Considerations

TBD

6. IANA Considerations

TBD

7. Acknowledgements

Thanks to Cullen Jennings and Suhas Nandakumar for their assistance in generating the SDP examples in this document.

Some of the material in this document was taken from [I-D.jennings-rtcweb-plan].

8. References

8.1. Normative References

- [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-03 (work in progress), February 2013.
- [I-D.jennings-mmusic-media-req]
Jennings, C., Uberti, J., and E. Rescorla, "Requirements
from various WG for MMUSIC", draft-jennings-mmusic-media-
req-00 (work in progress), February 2013.
- [I-D.nandakumar-mmusic-sdp-mux-attributes]
Nandakumar, S., "A Framework for SDP Attributes when
Multiplexing", draft-nandakumar-mmusic-sdp-mux-
attributes-02 (work in progress), April 2013.
- [I-D.westerlund-avtcore-rtp-simulcast]

Westerlund, M., Lindqvist, M., and F. Jansson, "Using Simulcast in RTP Sessions", draft-westerlund-avtcore-rtp-simulcast-02 (work in progress), February 2013.

- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.
- [RFC3264] Rosenberg, J. and H. Schulzrinne, "An Offer/Answer Model with Session Description Protocol (SDP)", RFC 3264, June 2002.
- [RFC4566] Handley, M., Jacobson, V., and C. Perkins, "SDP: Session Description Protocol", RFC 4566, July 2006.
- [RFC5576] Lennox, J., Ott, J., and T. Schierl, "Source-Specific Media Attributes in the Session Description Protocol (SDP)", RFC 5576, June 2009.
- [RFC5888] Camarillo, G. and H. Schulzrinne, "The Session Description Protocol (SDP) Grouping Framework", RFC 5888, June 2010.

8.2. Informative References

- [I-D.ietf-rtcweb-rtp-usage] Perkins, C., Westerlund, M., and J. Ott, "Web Real-Time Communication (WebRTC): Media Transport and Use of RTP", draft-ietf-rtcweb-rtp-usage-06 (work in progress), February 2013.
- [I-D.jennings-rtcweb-plan] Jennings, C., "Proposed Plan for Usage of SDP and RTP", draft-jennings-rtcweb-plan-01 (work in progress), February 2013.
- [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.
- [RFC4588] Rey, J., Leon, D., Miyazaki, A., Varsa, V., and R. Hakenberg, "RTP Retransmission Payload Format", RFC 4588, July 2006.
- [RFC5117] Westerlund, M. and S. Wenger, "RTP Topologies", RFC 5117, January 2008.

- [RFC5245] Rosenberg, J., "Interactive Connectivity Establishment (ICE): A Protocol for Network Address Translator (NAT) Traversal for Offer/Answer Protocols", RFC 5245, April 2010.
- [RFC5285] Singer, D. and H. Desineni, "A General Mechanism for RTP Header Extensions", RFC 5285, July 2008.
- [RFC5583] Schierl, T. and S. Wenger, "Signaling Media Decoding Dependency in the Session Description Protocol (SDP)", RFC 5583, July 2009.
- [RFC5761] Perkins, C. and M. Westerlund, "Multiplexing RTP Data and Control Packets on a Single Port", RFC 5761, April 2010.
- [RFC5956] Begen, A., "Forward Error Correction Grouping Semantics in the Session Description Protocol", RFC 5956, September 2010.
- [iana.rtp-pt]
IANA, "RTP Payload types (PT) for standard audio and video encodings", July 2013.

Available at <http://www.iana.org/assignments/rtp-parameters/rtp-parameters.xhtml#rtp-parameters-1>
- [webrtc-api]
Bergkvist, Burnett, Jennings, Narayanan, , "WebRTC 1.0: Real-time Communication Between Browsers", October 2011.

Available at <http://dev.w3.org/2011/webrtc/editor/webrtc.html>

Authors' Addresses

Adam Roach
Mozilla
Dallas, TX
US

Phone: +1 650 903 0800 x863
Email: adam@nostrum.com

Justin Uberti
Google
747 6th St. S
Kirkland, WA 98033
USA

Email: justin@uberti.name

Martin Thomson
Microsoft
3210 Porter Drive
Palo Alto, CA 94304
US

Phone: +1 650 353 1925
Email: martin.thomson@skype.net