Workgroup: AVTCORE
Internet-Draft:
draft-murillo-avtcore-multi-codec-payload-
format-01
Published: 11 July 2021
Intended Status: Standards Track
Expires: 12 January 2022
Authors: S. Garcia Murillo Y. Fablet A. Gouaillard
CoSMo Apple Inc. CoSMo
J. Uberti
Clubhouse

Multi Codec RTP payload format

Abstract

RTP Media Chains usually rely on piping encoder output directly to packetizers. Media packetization formats often support a specific codec format and optimize RTP packets generation accordingly. With the development of Selective Forward Unit (SFU) solutions, RTP Media Chains used in WebRTC solutions are increasingly relying on application-specific transforms that sit between encoder and packetizer on one end and between depacketizer and decoder on the other end. These transforms are typically encrypting media content so that the media content is not readable from the SFU, for instance using [SFrame] or [WebRTCInsertableStreams]. In that context, RTP packetizers can no longer expect to use packetization formats that mandate media content to be in a specific codec format. This document provides a solution to that problem by describing a RTP packetization format that can be used for many media content, and how to negotiate use of this format. This document also describes a solution to allow SFUs to continue performing packet routing on top of this RTP packetization format.

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

As per Figure 1 of [RFC7656], a Media Packetizer transforms a single Encoded Stream into one or several RTP packets. The Encoded Stream is coming straight from the Media Encoder and is expected to follow the format produced by the Media Encoder. A number of Media Packetizer formats have been designed to process a specific format produced by Media Encoder. For instance [RFC6184] is dedicated to

the processing of content produced by H.264 Media Encoders, and generates packets following NALUs organization.

WebRTC applications are increasingly deploying end-to-end encryption solutions on top of RTP Media Chains. End-to-end encryption is implemented by inserting application-specific Media Transformers between Media Encoder and Media Packetizer on the sending side, and between Media Depacketizer and Media Decoder on the receiving side, as described in Figure 1 and Figure 2. To support end-to-end encryption, Media Transformers can use the [SFrame] format. In browsers, Media Transformers are implemented using [WebRTCInsertableStreams], for instance by injecting JavaScript code provided by web pages.

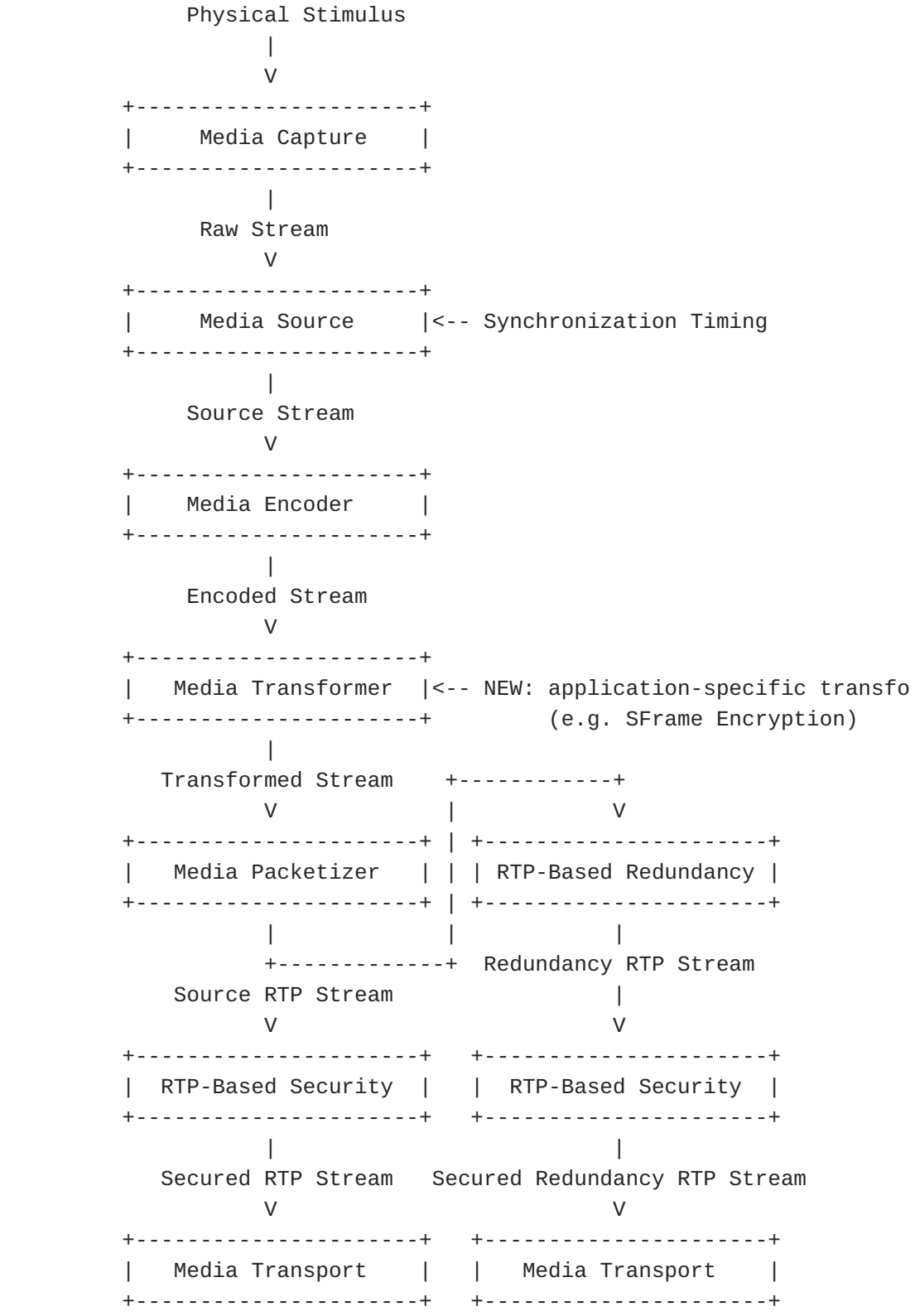
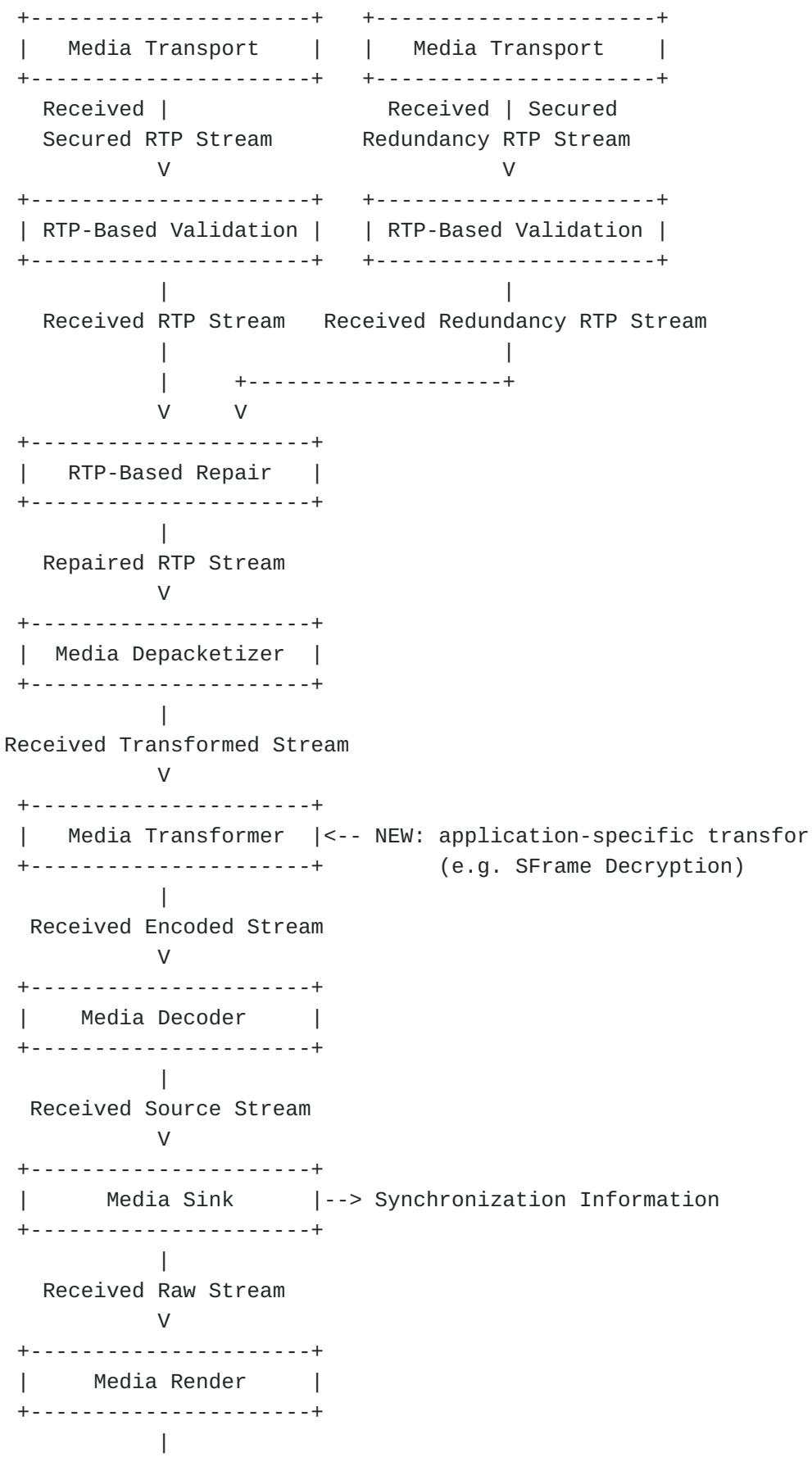


Figure 1: Sender side concepts in the Media Chain with application-level Media Transform

These RTP packets are sent over the wire to a receiver media chain matching the sender side, reaching the Media Depacketizer that will

reconstruct the Encoded Stream before passing it to the Media Decoder.


Media Transport
Media Transport
Received Secured RTP Stream
Received Secured Redundancy RTP Stream
RTP-Based Validation
RTP-Based Validation
Received RTP Stream
Received Redundancy RTP Stream
RTP-Based Repair
Repaired RTP Stream
Media Depacketizer
Received Transformed Stream
Media Transformer
NEW: application-specific transfor
(e.g. SFrame Decryption)
Received Encoded Stream
Media Decoder
Received Source Stream
Media Sink
Synchronization Information
Received Raw Stream
Media Render

V

Physical Stimulus

Figure 2: Receiver side concepts in the Media Chain with application-level Media Transform

This packetization does not change how the mapping between one or several encoded or dependant streams are mapped to the RTP streams or how the synchronization sources(s) (SSRC) are assigned.

Given the use of post-encoder application-specific transforms, the whole Media Chain needs to be made aware of it. This includes the sender post-transform Media Chain, Media Transport intermediaries (SFUs typically) and receiver pre-transform Media Chain.

As these transforms can alter Encoded Streams in any possible way, the use of codec-specific Media Packetizers like [RFC6184] on Transformed Stream may be suboptimal on sender side. It may also be problematic on the receiving side in case codec-specific processing is done prior the Media Transformer. Media Transport intermediaries are often looking at the Media Content itself to fuel their packet selection algorithms.

2. Goals

The objective of this document is to support inserting any application-specific transform between encoders and packetizers in the Media Chain. For that purpose, this document will: 1. Provide a packetization format that supports multiple media content used by WebRTC applications (audio compressed by Opus, video compressed by H264 or VP8, encrypted content...) that allows reuse of existing RTP mechanisms in place in WebRTC applications such as RTX, RED or FEC. 2. Provide a way to negotiate use of this packetization format between sender and receiver, with minimum impact on existing negotiation approaches. 3. Provide a side-channel information so that network intermediaries (SFU in particular) can do their existing packet routing strategies without inspecting the media content.

3. RTP Packetization

This packetizer, by design, is not expected to understand the format of the media to transmit. The unit used by the packetizer to do processing is called a frame in the remainder of the document.

It is the responsibility of the application using the packetizer to group media content in meaningful frames. In the common case of a video codec, the packetizer frame is the frame in byte format (h264 annex b for example) generated by the encoder.

If the application wants to transform encoded content, the application needs to split the encoded content into frames prior the transform. Each frame is then transformed independently, for

instance encrypted using [SFrame]. The content of each transformed frame is then processed by the packetizer.

In the case of a video codec supporting spatial scalability, each spatial layer MUST be split in its own frame by the application before passing it to the packetizer.

When the packetizer receives a frame from the application, it MUST fragment the frame content in multiple RTP packets to ensure packets do not exceed the network maximum transmission unit. The content of the frame will be treated as a binary blob by the packetizer, so the decision about the boundaries of each fragment is decided arbitrarily by the packetizer. The packetizer or any relying server MUST NOT modify the frame content and concatenating the RTP payload of the RTP packets for each frame MUST produce the exact binary content of the input frame content.

The marker bit of each RTP packet in a frame MUST be set according to the audio and video profiles specified in [RFC3551].

The spatial layer frames are sent in ascending order, with the same RTP timestamp, and only the last RTP packet of the last spatial layer frame will have the marker bit set to 1.

4. Payload Multiplexing

In order to reduce the number of payload type in the SDP exchange, a single payload type code for this multi-codec packetization can be used for all negotiated media formats that the multi-codec packetization supports. That requires to identify the original payload type code of the frame negotiated media format, called the associated payload type (APT) hereunder. The APT value is the payload type code of the associated format passed to the multi-codec Media Packetizer before any transformation is applied.

The APT value is sent in a dedicated header extension. The payload of this header extension can be encoded using either the one-byte or two-byte header defined in [RFC5285]. Figures 3 and 4 show examples with each one of these examples.

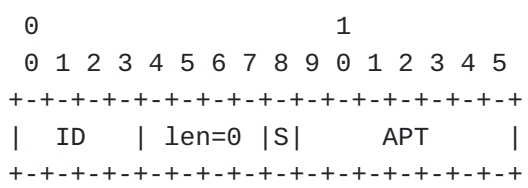


Figure 3: Frame associated payload type encoding using the One-Byte header format

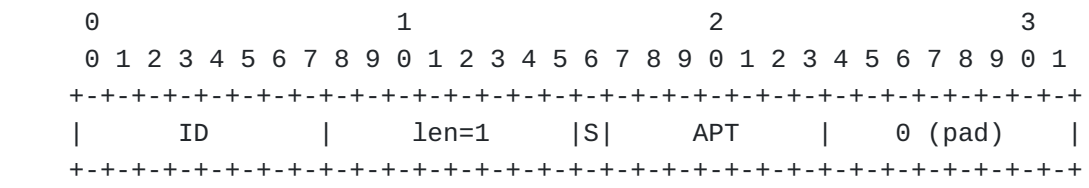


Figure 4: Frame associated payload type encoding using the Two-Byte header format

The APT value is the associated payload type value. The S bit indicates if the media stream can be forwarded safely starting from this RTP packet. Typically, it will be set to 1 on the first RTP packet of an intra video frame and in all RTP audio packets.

Receivers MUST be ready to receive RTP packets with different associated payload types in the same way they would receive different payload type codes on the RTP packets.

The URI for declaring this header extension in an extmap attribute is "urn:ietf:params:rtp-hdrext:associated-payload-type".

5. SDP Negotiation

To use the multi-codec packetization, the SDP Offer/Answer exchange MUST negotiate: - The payload type of the negotiated codec format - The multi-codec payload type - The associated payload type header extension

Only the negotiated payload types are allowed to be used as associated payload types. Figure 5 illustrates a SDP that negotiates exchange of video using either VP8 or VP9 codecs with the possibility to use the multi-codec packetization. In this example, RTX is also negotiated and will be applied normally on each associated payload type.

```
m=video 9 UDP/TLS/RTP/SAVPF 96 97 98 99 100 101
c=IN IP4 0.0.0.0
a=rtcp:9 IN IP4 0.0.0.0
a=setup:actpass
a=mid:1
a=extmap:1 urn:ietf:params:rtp-hdrext:sdes:mid
a=extmap:2 urn:ietf:params:rtp-hdrext:sdes:rtp-stream-id
a=extmap:3 urn:ietf:params:rtp-hdrext:sdes:repaired-rtp-stream-id
a=extmap:4 urn:ietf:params:rtp-hdrext:associated-payload-type
a=sendrecv
a=rtpmap:96 vp9/90000
a=rtpmap:97 vp8/90000
a=rtpmap:98 generic/90000
a=rtpmap:99 rtx/90000
a=fmtp:99 apt=96
a=rtpmap:100 rtx/90000
a=fmtp:100 apt=97
a=rtpmap:101 rtx/90000
a=fmtp:101 apt=98
```

Figure 5: SDP example negotiating the multi-codec payload type and related header extension for video

6. SFU Packet Selection

SFUs need to have a basic understanding of each frame they receive so they can decide to forward it or not and to which endpoint. They might need similar information to support media content recording. This information is either generic to a group of frames (called a stream hereafter) or specific to each frame.

The information is transmitted as a RTP header extension as the RTP packet payload should be treated as opaque by the SFU. This is especially necessary if the payload is end-to-end encrypted. The amount of information should be limited to what is strictly necessary to the SFU task since it is not always as trusted as individual peers.

For audio, configuration information such as Opus TOC might be useful. For video, configuration information might include: - Stream configuration information: resolution, quality, frame rate... - Codec specific configuration information: codec profile like profile_idc... - Frame specific information: whether the stream is decodable when starting from this frame, whether the frame is skippable...

For video content, this information is sent using a Dependency Descriptor header extension. In that case, the first RTP packet of

the frame will have its start_of_frame equal to 1 and the last packet will have its end_of_frame equal to 1.

7. Sender Processing Rules

The sender identifies the use of the multi-codec payload format by using the urn:ietf:params:rtp-hdrext:associated-payload-type extension. When doing so, the sender follows these additional rules: - For audio content, the associated payload type MUST reference an audio codec in the supported audio codec list. The supported audio codec list contains the audio codecs enumerated in [RFC7874]. This list may be extended in future versions of this specification. - For video content, H.264 and VP8 are supported as described in [RFC7742], as well as VP9 and AV.1. In the case scalable video coding is used, the sender MUST generate a Dependency Descriptor header extension. This requires the associated payload type to reference a video codec that can be described using the Dependency Descriptor header extension. This also requires the sender to split the video encoder output in frames that can each be described using the Dependency Descriptor header extension.

These rules apply to both the originator of the content as well as SFUs that might route the content to end receivers.

8. Redundancy Techniques Considerations

The solution described in this document is expected to integrate well with the existing RTP ecosystem. This section describes how the multi-codec packetizer can be used jointly with existing techniques that allow to mitigate unreliable transports.

8.1. Retransmission Techniques

[RFC4588] defines a retransmission payload format (RTX) that can be used in case of packet loss. As defined in [RFC4588], RTX is able to handle any payload format, including the format described in this document. Given RTX preserves both RTP packet payload and headers, the receiver will be able to identify the payload type of the recovered packet and whether multi-codec packetization is used. RTX will also allow recovering RTP header extensions that convey information on the media content itself.

8.2. Forward Error Correction (FEC) Techniques

FEC is another technique used in RTP Media Chains to protect media content against packet loss. [RFC5109] defines such a payload format used to transmit FEC for specific packets protection.

FEC may protect some parts of the media content more than others. For instance, intra video frame encoded data or important network

abstraction layer units (NALUs) like SPS/PPS may be more protected. With a post-encoder transform and the use of a multi-codec packetization, the granularity of the recovery mechanism is no longer at the NALU level but at the level of the frame generated by the post-encoder transform. In case a SVC codec is used, each spatial layer will be processed as an independent frame. In that case, base layers can be protected more heavily than higher resolution layers.

8.3. Redundant Audio Data Techniques

As defined in [RFC7656] RTP-based redundancy is defined here as a transformation that generates redundant or repair packets sent out as a Redundancy RTP Stream to mitigate Network Transport impairments, like packet loss and delay.

[RFC2198] defines a payload format for sending the same audio data encoded multiple times at different quality levels. This allows to use a lower quality encoding of the audio data, should the higher quality encoding of the audio data is lost during the transmission.

If a Media Transformation is in use, both the primary and redundant encoding must be transformed independently and the redundant packet created normally. As the RTP headers present in the redundant packet are only applicable to the primary encoding, if the payload type for a redundant encoding block is mapped to the multi-codec packetizer, the value of the associated payload type for the primary encoding is applied to the redundant encoding block as well.

9. Alternatives

Various alternatives can be used to implement and negotiate multi-codec packetization. This section describes a few additional alternatives. This section is to be removed before finalization of the document.

9.1. Generic Packetization With In-Payload APT

Instead of using a RTP header extension to convey the APT value, it is prepended in the RTP payload itself. As the value cannot change for a whole frame, its value is prepended to the first packet generated of the frame only. This removes the need to negotiate a dedicated header extension, but may require the SFU to update the payload when sending or recording content.

9.2. A Payload Type for Generic Packetization AND Media Format

The payload type is negotiated in the SDP so as to identify both the negotiated codec format and the multi-codec packetization use. There

is no network cost but this increases the number of payload types used in the SDP.

```
m=video 9 UDP/TLS/RTP/SAVPF 96 97 98 99 100 101
c=IN IP4 0.0.0.0
a=rtcp:9 IN IP4 0.0.0.0
a=setup:actpass
a=mid:1
a=extmap:1 urn:ietf:params:rtp-hdrext:sdes:mid
a=extmap:2 urn:ietf:params:rtp-hdrext:sdes:rtp-stream-id
a=extmap:3 urn:ietf:params:rtp-hdrext:sdes:repaired-rtp-stream-id
a=sendrecv
a=rtpmap:96 vp9/90000
a=rtpmap:97 generic/90000
a=fmtp:97 apt=96
a=rtpmap:98 vp8/90000
a=rtpmap:99 generic/90000
a=fmtp:99 apt=98
a=rtpmap:100 rtx/90000
a=fmtp:100 apt=96
a=rtpmap:101 rtx/90000
a=fmtp:101 apt=97
a=rtpmap:102 rtx/90000
a=fmtp:102 apt=98
a=rtpmap:103 rtx/90000
a=fmtp:103 apt=99
```

Figure 6: SDP example negotiating a payload type for format and multi-codec packetization

A variation of this approach is to consider defining several multi-codec payload types, each of them having an identified codec format.

```
m=video 9 UDP/TLS/RTP/SAVPF 96 97 98 99 100 101
c=IN IP4 0.0.0.0
a=rtcp:9 IN IP4 0.0.0.0
a=setup:actpass
a=mid:1
a=extmap:1 urn:ietf:params:rtp-hdrext:sdes:mid
a=extmap:2 urn:ietf:params:rtp-hdrext:sdes:rtp-stream-id
a=extmap:3 urn:ietf:params:rtp-hdrext:sdes:repaired-rtp-stream-id
a=sendrecv
a=rtpmap:96 generic/90000
a=fmtp:96 codec=vp9
a=rtpmap:97 generic/90000
a=fmtp:97 codec=vp8
a=rtpmap:98 rtx/90000
a=fmtp:98 apt=96
a=rtpmap:99 rtx/90000
a=fmtp:99 apt=97
```

Figure 7: Alternative SDP example negotiating a payload type for format and multi-codec packetization

9.3. A RTP Header To Choose Packetization

A RTP header extension can be used to flag content as opaque so that the receiver knows whether to use or not the multi-codec packetization. As for the API header extension, the RTP header extension may not need to be sent for every packet, it could for instance be sent for the first packet of every intra video frame. The main advantage of this approach is the reduced impact on SDP negotiation.

```
m=video 9 UDP/TLS/RTP/SAVPF 96 97 98 99 100 101
c=IN IP4 0.0.0.0
a=rtcp:9 IN IP4 0.0.0.0
a=setup:actpass
a=mid:1
a=extmap:1 urn:ietf:params:rtp-hdrext:sdes:mid
a=extmap:2 urn:ietf:params:rtp-hdrext:sdes:rtp-stream-id
a=extmap:3 urn:ietf:params:rtp-hdrext:sdes:repaired-rtp-stream-id
a=extmap:4 urn:ietf:params:rtp-hdrext:multi-codec-packetization-use
a=sendrecv
a=rtpmap:96 vp9/90000
a=rtpmap:97 vp8/90000
a=rtpmap:98 rtx/90000
a=fmtp:98 apt=96
a=rtpmap:99 rtx/90000
a=fmtp:99 apt=97
```

Figure 8: SDP example negotiating multi-codec packetization as RTP header extension

10. Security Considerations

RTP packets using the payload format defined in this specification are subject to the general security considerations discussed in [RFC3550]. It is not expected that the proposed solution presented in this document can create new security threats. The use and implementation of RTP Media Chains containing Media Transformers needs to be done carefully. It is important to refer to the security considerations discussed in [SFrame] and [WebRTCInsertableStreams]. In particular Media Transformers on the receiver side need to be prepared to receive arbitrary content, like decoders already do. Similarly, since Media Transformers can be implemented as JavaScript in browsers, RTP Packetizers should be prepared to receive arbitrary content.

11. IANA Considerations

Two new media subtypes have been registered with IANA, as described in this section.

11.1. Registration of audio/generic

Type name: audio

Subtype name: generic

Required parameters: none

Optional parameters: none

Encoding considerations: This format is framed (see Section 4.8 in the template document) and contains binary data.

Security considerations: TBD.

Interoperability considerations: TBD

Published specification: TBD.

Applications that use this media type: TBD.

Additional information: none

Intended usage: COMMON

Restrictions on usage: TBD

Author:

Change controller:

12. Registration of video/generic

Type name: video

Subtype name: generic

Required parameters: none

Optional parameters: none

Encoding considerations: This format is framed (see Section 4.8 in the template document) and contains binary data.

Security considerations: TBD.

Interoperability considerations: TBD

Published specification: TBD.

Applications that use this media type: TBD.

Additional information: none

Intended usage: COMMON

Restrictions on usage: TBD

Author:

Change controller:

13. References

13.1. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/info/rfc2119>.

[RFC3550] Schulzrinne, H., Casner, S., Frederick, R., and V. Jacobson, "RTP: A Transport Protocol for Real-Time Applications", STD 64, RFC 3550, DOI 10.17487/RFC3550, July 2003, <https://www.rfc-editor.org/info/rfc3550>.

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

3551, DOI 10.17487/RFC3551, July 2003, <https://www.rfc-editor.org/info/rfc3551>.

[RFC3711] Baugher, M., McGrew, D., Naslund, M., Carrara, E., and K. Norrman, "The Secure Real-time Transport Protocol (SRTP)", RFC 3711, DOI 10.17487/RFC3711, March 2004, <https://www.rfc-editor.org/info/rfc3711>.

[RFC4566] Handley, M., Jacobson, V., and C. Perkins, "SDP: Session Description Protocol", RFC 4566, DOI 10.17487/RFC4566, July 2006, <https://www.rfc-editor.org/info/rfc4566>.

[RFC5285] Singer, D. and H. Desineni, "A General Mechanism for RTP Header Extensions", RFC 5285, DOI 10.17487/RFC5285, July 2008, <https://www.rfc-editor.org/info/rfc5285>.

[RFC7656] Lennox, J., Gross, K., Nandakumar, S., Salgueiro, G., and B. Burman, Ed., "A Taxonomy of Semantics and Mechanisms for Real-Time Transport Protocol (RTP) Sources", RFC 7656, DOI 10.17487/RFC7656, November 2015, <https://www.rfc-editor.org/info/rfc7656>.

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

13.2. Informative References

[RFC2198] Perkins, C., Kouvelas, I., Hodson, O., Hardman, V., Handley, M., Bolot, J.C., Vega-Garcia, A., and S. Fosse-Parisis, "RTP Payload for Redundant Audio Data", RFC 2198, DOI 10.17487/RFC2198, September 1997, <https://www.rfc-editor.org/info/rfc2198>.

[RFC4588] Rey, J., Leon, D., Miyazaki, A., Varsa, V., and R. Hakenberg, "RTP Retransmission Payload Format", RFC 4588, DOI 10.17487/RFC4588, July 2006, <https://www.rfc-editor.org/info/rfc4588>.

[RFC5109] Li, A., Ed., "RTP Payload Format for Generic Forward Error Correction", RFC 5109, DOI 10.17487/RFC5109, December 2007, <https://www.rfc-editor.org/info/rfc5109>.

[RFC6184] Wang, Y.-K., Even, R., Kristensen, T., and R. Jesup, "RTP Payload Format for H.264 Video", RFC 6184, DOI 10.17487/RFC6184, May 2011, <https://www.rfc-editor.org/info/rfc6184>.

[RFC6464] Lennox, J., Ed., Ivov, E., and E. Marocco, "A Real-time Transport Protocol (RTP) Header Extension for Client-to-Mixer Audio Level Indication", RFC 6464, DOI 10.17487/RFC6464, December 2011, <https://www.rfc-editor.org/info/rfc6464>.

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

[RFC6904] Lennox, J., "Encryption of Header Extensions in the Secure Real-time Transport Protocol (SRTP)", RFC 6904, DOI 10.17487/RFC6904, April 2013, <https://www.rfc-editor.org/info/rfc6904>.

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

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

[SFrame] "Secure Frame (SFrame)", n.d., <https://tools.ietf.org/html/draft-omara-sframe>.

[WebRTCInsertableStreams] "WebRTC Insertable Media using Streams", n.d., <https://w3c.github.io/webrtc-insertable-streams>.

Authors' Addresses

Sergio Garcia Murillo
CoSMo

Email: sergio.garcia.murillo@cosmosoftware.io

Youenn Fablet
Apple Inc.

Email: youenn@apple.com

Alex Gouaillard
CoSMo

Email: alex.gouaillard@cosmosoftware.io

Justin Uberti

Clubhouse

Email: justin@uberti.name