Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: May 1, 2012

C. Bran
Plantronics
C. Jennings
Cisco

October 29, 2011

WebRTC Codec and Media Processing Requirements
draft-cbran-rtcweb-codec-01

Abstract

This document outlines the codec and media processing requirements for WebRTC client application and endpoint devices.

Status of this Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79. This document may not be modified, and derivative works of it may not be created, and it may not be published except as an Internet-Draft.

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 May 1, 2012.

Copyright Notice

Table of Contents

1. Introduction

An integral part of the success and adoption of the Web Real Time Communications (WebRTC) will be the voice and video interoperability between WebRTC applications. This specification will outline the media processing and codec requirements for WebRTC client implementations.

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].

3. Codec Requirements

This section covers the audio and video codec requirements for WebRTC client applications. To ensure a baseline level of interoperability between WebRTC clients, a minimum set of required codecs are specified below. While this section specifies the codecs that will be mandated for all WebRTC client implementations, it leaves the question of supporting additional codecs to the will of the implementer.

3.1. Audio Codec Requirements

WebRTC clients are REQUIRED to implement the following audio codecs.

- PCMA/PCMU - 1 channel with a rate of 8000 Hz and a ptime of 20 - see section 4.5.14 of [RFC3551]
- Telephone Event - [RFC4734]
- Opus [draft-ietf-codec-opus]

3.2. Video Codec Requirements

If the MPEG-LA issues an intent to offer H.264 baseline profile on a royalty free basis for use in browsers before March 15, 2012, then the REQUIRED video codecs will be H.264 baseline. If this does not happen by that the date, then the REQUIRED video codec will be VP8 [I-D.webm].

The following feature list applies to all required video codecs.

Required video codecs:

- MUST support at least 10 frames per second (fps) and SHOULD support 30 fps
- If VP8, then MUST support a the bilinear and none reconstruction filters
- OPTIONALLY offer support for additional color spaces
- MUST support a minimum resolution of 320X240
- SHOULD support resolutions of 1280x720, 720x480, 1024x768, 800x600, 640x480, 640 x 360 , 320x240

4. WebRTC Client Requirements

It is plausible that the dominant near to mid-term WebRTC usage model will be people using the interactive audio and video capabilities to communicate with each other via web browsers running on a notebook computer that has built-in microphone and speakers. The notebook-as-communication-device paradigm presents challenging echo cancellation and audio gain problems, the specific remedy of which will not be mandated here. However, while no specific algorithm or standard will be required by WebRTC compatible clients, functionality such as automatic gain control, echo cancellation, headset detection and passing call control events to connected devices will improve the user experience and should be implemented by the endpoint device.

To address the problems outlined above, suitable implementations of the functionality listed below SHOULD be available within an RTC-Web endpoint device.

- Automatic gain control
- Ability to override automatic gain control to manually set gain
- Auto-adjustments to gain control and echo cancellation algorithms based on if a headset or internal speakers/microphone is being used
- Echo cancellation, including acoustic echo cancellation
- Headset detection
- Call control event notification to connected devices such as headsets

5. Legacy VoIP Interoperability

The codec requirements above will ensure, at a minimum, voice interoperability capabilities between WebRTC client applications and legacy phone systems.

Video interoperability will be dependent upon the MPEG-LA decision regarding H.264 baseline.

6. IANA Considerations

This document makes no request of IANA.

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

7. Security Considerations

The codec requirements have no additional security considerations other than those captured in [I-D.ekr-security-considerations-for-rtc-web].

8. Acknowledgements

This draft incorporates ideas and text from various other drafts. In particularly we would like to acknowledge, and say thanks for, work we incorporated from Harald Alvestrand.

9. Normative References

[I-D.ekr-security-considerations-for-rtc-web]
Rescorla, E., "Security Considerations for RTC-Web", May 2011.

[I-D.webm]
Google, Inc., "VP8 Data Format and Decoding Guide", July 2010.

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

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

[RFC4734] Schulzrinne, H. and T. Taylor, "Definition of Events for Modem, Fax, and Text Telephony Signals", RFC 4734, December 2006.

Authors' Addresses

Cary Bran
Plantronics
345 Encinial Street
Santa Cruz, CA 95060
USA

Phone: +1 206 661-2398
Email: cary.bran@plantronics.com

Cullen Jennings
Cisco
170 West Tasman Drive
San Jose, CA 95134
USA

Phone: +1 408 421-9990
Email: fluffy@cisco.com

MMUSIC Working Group
Internet-Draft
Intended status: Standards Track
Expires: April 21, 2012

C. Holmberg
Ericsson
H. Alvestrand
Google

October 19, 2011

Multiplexing Negotiation Using Session Description Protocol (SDP) Port Numbers

draft-holmberg-mmusic-sdp-bundle-negotiation-00.txt

Abstract

This specification defines a new SDP Grouping Framework SDP grouping framework extension, "BUNDLE", that can be used with the Session Description Protocol (SDP) Offer/Answer mechanism to negotiate the usage of bundled media, which refers to the usage of a single 5-tuple for media associated with multiple SDP media descriptions ("m=" lines).

Status of this Memo

This Internet-Draft is submitted to IETF 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 April 21, 2012.

Copyright Notice

Table of Contents

1. Introduction

In the IETF RTCWEB WG, a need to use a single 5-tuple for sending and receiving media associated with multiple SDP media descriptions ("m=" lines) has been identified. This would e.g. allow the usage of a single set of Interactive Connectivity Establishment (ICE) [RFC5245] candidates for multiple media descriptions. Normally different media types (audio, video etc) will be described using different media descriptions.

This specification defines a new SDP Grouping Framework SDP grouping framework [RFC5888] extension, "BUNDLE", that can be used with the Session Description Protocol (SDP) Offer/Answer mechanism [RFC3264] to negotiate the usage of bundled media, which refers to the usage of a single 5-tuple for media associated with multiple SDP media descriptions ("m=" lines).

When an endpoint generates an SDP Offer or SDP Answer [RFC3264], which includes a "BUNDLE" group, each "m=" line associated with the group will share a single port number value.

As defined in RFC 4566 [RFC4566], the semantics of multiple "m=" lines using the same port number value are undefined, and there is no grouping defined by such means. Instead, an explicit grouping mechanism needs to be used to express the intended semantics. This specification provides such extension.

When media is transported using the Real-Time Protocol (RTP) [RFC3550], the default assumption of the mechanism is that all media associated with a "BUNDLE" group will form a single RTP Session [RFC3550]. However, future specifications can extend the mechanism, in order to negotiate RTP Session multiplexing, i.e. "BUNDLE" groups where media associated with a group form multiple RTP Sessions.

The mechanism is backward compatible. Entities that do not support the "BUNDLE" grouping extension, or do not want to enable the mechanism for a given session, are expected to generate a "normal" SDP Answer, using different port number values for each "m=" line, to the SDP Offer. The SDP Offerer [RFC3264] will still use a single port number value for each media, but as the SDP Answerer [RFC3264] will use separate ports a single 5-tuple will not be used for media associated with multiple "m=" lines between the endpoints.

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].

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

Bundled media: Two or more RTP streams using a single 5-tuple. The RTCP streams associated with the RTP streams also use a single 5-tuple, which might be the same, but can also be different, as the one used by the RTP streams.

3. Conventions

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 BCP 14, RFC 2119 [RFC2119].

4. Applicability Statement

The mechanism in this specification only applies to the Session Description Protocol (SDP) [RFC4566], when used together with the SDP Offer/Answer mechanism [RFC3264].

5. SDP Grouping Framework BUNDLE Extension Semantics

This section defines a new SDP Grouping Framework extension, "BUNDLE".

The "BUNDLE" extension can be indicated using an SDP session-level 'group' attribute. Each SDP media description ("m=" line) that is grouped together, using an SDP media-level 'mid' attribute, is part of a specific "BUNDLE" group.

6. SDP Offer/Answer Procedures

6.1. General

When an SDP Offerer or SDP Answerer generates an SDP Offer or SDP Answer, that describes bundled media, it MUST insert an SDP session-level 'group' attribute, with a "BUNDLE" value, and assign SDP media-level 'mid' attribute values to each "m=" line associated with the "BUNDLE" group.

In addition, the entity that generates the SDP Offer or SDP Answer

MUST, for each "m=" line that is part of the "BUNDLE" group:

- 1. Use the same port number value.
- 2. Use the same connection data ("c=" line) value.
- 3. Use the same SDP 'rtcp' attribute value, when used.
- 4. Use the same ICE candidate values, when used.
- 5. Insert an SDP 'rtpc-mux' attribute.

NOTE: If an entity wants to disable specific media ("m=" line) associated with a "BUNDLE" group, it will use a zero port number value for the "m=" line associated with the media.

6.2. SDP Offerer Procedures

When an SDP Offerer creates an SDP Offer, that offers bundled media, it MUST create the SDP Offer according to the procedures in Section 6.1.

If the associated SDP Answer contains an SDP session-level 'group' attribute, with a "BUNDLE" value, and the SDP Answer is created according to the proceudres in Section 6.1 (the same port number value is used for each "m=" line associated with the "BUNDLE" group, etc), the SDP Offerer can start using the same 5-tuple for sending and receiving media, associated with the group, between the entities.

If the SDP Answer does not include a session-level SDP 'group' attribute, with a "BUNDLE" value, the SDP Offerer cannot use the same 5-tuple for media associated with multiple "m=" lines.

If the SDP Answererer indicates that it will not use bundled media, the SDP Offerer will still use the single port number value for each "m= line" associated with the offered "BUNDLE" group, and it will normally be able to separate each individual media. The default mechanism for separating media received on a single IP address and port doing this is by using a 5-tuple based mapping for each individual media. If the SDP Offerer is aware of the Synchronization Source (SSRC) [RFC3550] values that the SDP Answerer will use in the media it sends, and the SSRC values will be unique for each media, the SDP Offerer can separate media based on the SSRC values.

NOTE: Assuming symmetric media is used, the SDP Offerer can use the port information from the SDP Answer in order to create the 5-tuple mapping for each media.

If the SDP Offerer is not able to separate multiple media received on a single port, it MUST send a new SDP Offer, without offering bundled media, where a separate port number value is provide for each "m=" line of the SDP Offer.

If an SDP Offer, offering a "BUNDLE" group, and the SDP Offerer has reasons to believe that the rejection is due to the usage of a single port number value for multiple "m=" lines, the SDP Offerer SHOULD send a new SDP Offer, without a "BUNDLE" group, where a separate port number value is provide for each "m=" line of the SDP offer.

6.3. SDP Answerer Procedures

When an SDP Answerer receives an SDP Offer, which offers bundled media, and the SDP Answerer accepts the offered bundle group, the SDP Answerer MUST create an SDP Answer according to the procedures in Section 6.1.

If the SDP Answerer does not accept the "BUNDLE" group in the SDP Offer, it MUST NOT include a session-evel ’group’ attribute, with a "BUNDLE" value, in the associated SDP Answer. In addition, the SDP Answerer MUST provide separate port number values for each "m=" line of the SDP Answer.

6.4. Bundled SDP Information

6.4.1. General

This section describes how SDP information, given for each media description, is calculated into a single value for a "BUNDLE" group.

6.4.2. Bandwidth (b=)

The total proposed bandwidth is the sum of the proposed bandwidth for each "m=" line associated with a negotiated BUNDLE group.

7. Single vs Multiple RTP Sessions

7.1. General

When entities negotiate the usage of bundled media, the default assumption is that all media associated with the bundled media will form a single RTP session.

The usage of multiple RTP Sessions within a "BUNDLE" group is outside the scope of this specification. Other specification needs to extend the mechanism in order to allow negotiation of such bundle groups.

7.2. Single RTP Session

When a single RTP Session is used, media associated with all "m=" lines part of a bundle group share a single SSRC [RFC3550] numbering

space.

In addition, the following rules and restrictions apply for a single RTP Session:

- o - The dynamic payload type values used in the "m=" lines MUST NOT overlap.
- o - The "proto" value in each "m=" line MUST be identical (e.g. RTP/AVPF).
- o - A given SSRC SHOULD NOT transmit RTP packets using payload types that originates from different "m=" lines.

NOTE: The last bullet above is to avoid sending multiple media types from the same SSRC. If transmission of multiple media types are done with time overlap RTP and RTCP fails to function. Even if done in proper sequence this causes RTP Timestamp rate switching issues [ref to draft-ietf-avtext-multiple-clock-rates].

8. Usage With ICE

8.1. General

This section describes how to use the "BUNDLE" grouping mechanism together with the Interactive Connectivity Establishment (ICE) mechanism [RFC5245].

8.2. Candidates

When an ICE-enabled SDP Offerer sends an SDP offer, it MUST include ICE candidates for each "m=" line associated with a "BUNDLE" group. The candidate values MUST be identical for each "m=" line associated with the group. This rule applies also to subsequent SDP Offers, when the usage of bundled media has already been negotiated.

When an ICE-enabled SDP Answerer receives an SDP Offer, offering a "BUNDLE" group and ICE, if the SDP Answerer enables ICE, MUST include ICE candidates for each "m=" line of the SDP Answer. This also applies for "m=" lines that are part of a "BUNDLE" group, in which case the candidate values MUST be identical for each "m=" line associated with the group. This rule applies also to subsequent SDP Answers, when the usage of bundled media has already been negotiated.

Once the usage of bundled media has been negotiated, ICE connectivity checks and keep-alives only needs to be performed for the whole "BUNDLE" group, instead of for each individual m= line associated with the group.

9. Security Considerations

TBA

10. Example

The example below shows an SDP Offer, where bundled media is offered. The example also shows two SDP Answer alternatives: one where bundled media is accepted, and one where bundled media is rejected (or, not even supported) by the SDP Answerer.

SDP Offer (Bundled media offered)

```
v=0
o=alice 2890844526 2890844526 IN IP4 host.atlanta.com
s=
c=IN IP4 host.atlanta.com
t=0 0
a=group:BUNDLE foo bar
m=audio 10000 RTP/AVP 0 8 97
a=mid:foo
b=AS:200
a=rtpmap:0 PCMU/8000
a=rtpmap:8 PCMA/8000
a=rtpmap:97 iLBC/8000
m=video 10000 RTP/AVP 31 32
a=mid:bar
b=AS:1000
a=rtpmap:31 H261/90000
a=rtpmap:32 MPV/90000
```

SDP Answer (Bundled media accepted)

```
v=0
o=bob 2808844564 2808844564 IN IP4 host.biloxi.com
s=
c=IN IP4 host.biloxi.com
t=0 0
a=group:BUNDLE foo bar
m=audio 20000 RTP/AVP 0
a=mid:foo
b=AS:200
a=rtpmap:0 PCMU/8000
m=video 20000 RTP/AVP 32
a=mid:bar
b=AS:1000
```

```
a=rtpmap:32 MPV/90000
```

SDP Answer (Bundled media not accepted)

```
v=0
o=bob 2808844564 2808844564 IN IP4 host.biloxi.com
s=
c=IN IP4 host.biloxi.com
t=0 0
m=audio 20000 RTP/AVP 0
b=AS:200
a=rtpmap:0 PCMU/8000
m=video 30000 RTP/AVP 32
b=AS:1000
a=rtpmap:32 MPV/90000
```

SDP Offer with ICE (Bundled media offered)

```
v=0
o=alice 2890844526 2890844526 IN IP4 host.atlanta.com
s=
c=IN IP4 host.atlanta.com
t=0 0
a=group:BUNDLE foo bar
m=audio 10000 RTP/AVP 0 8 97
a=mid:foo
b=AS:200
a=rtpmap:0 PCMU/8000
a=rtpmap:8 PCMA/8000
a=rtpmap:97 iLBC/8000
a=candidate:1 1 UDP 1694498815 host.atlanta.com 10000 typ host
m=video 10000 RTP/AVP 31 32
a=mid:bar
b=AS:1000
a=rtpmap:31 H261/90000
a=rtpmap:32 MPV/90000
a=candidate:1 1 UDP 1694498815 host.atlanta.com 10000 typ host
```

11. IANA Considerations

This document requests IANA to register the new SDP Grouping semantic extension called BUNDLE.

12. Acknowledgements

The usage of the SDP grouping mechanism is based on a similar alternative proposed by Harald Alvestrand. The SDP examples are also modified versions from the ones in the Alvestrand proposal.

Thanks to the nice flight crew on AY 021 for providing good sparkling wine, and a nice working athmosphere, for working on this draft.

13. Change Log

[RFC EDITOR NOTE: Please remove this section when publishing]

Changes from draft-holmberg-mmusic-sdp-multiplex-negotiation-00

- Draft name changed.
- Harald Alvestrand added as co-author.
- "Multiplex" terminology changed to "bundle".
- Added text about single versus multiple RTP Sessions.
- Added reference to RFC 3550.

14. References

14.1. Normative References

[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.

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

14.2. Informative 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.

[RFC5245] Rosenberg, J., "Interactive Connectivity Establishment (ICE): A Protocol for Network Address Translator (NAT) Traversal for Offer/Answer Protocols", RFC 5245, April 2010.

Authors' Addresses

Christer Holmberg
Ericsson
Hirsalantie 11
Jorvas 02420
Finland

Email: christer.holmberg@ericsson.com

Harald Tveit Alvestrand
Google
Kungsbron 2
Stockholm 11122
Sweden

Email: harald@alvestrand.no

Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: March 31, 2012

H. Alvestrand
Google

September 28, 2011

Overview: Real Time Protocols for Brower-based Applications
draft-ietf-rtcweb-overview-02

Abstract

This document gives an overview and context of a protocol suite intended for use with real-time applications that can be deployed in browsers - "real time communication on the Web".

It intends to serve as a starting and coordination point to make sure all the parts that are needed to achieve this goal are findable, and that the parts that belong in the Internet protocol suite are fully specified and on the right publication track.

This work is an attempt to synthesize the input of many people, but makes no claims to fully represent the views of any of them. All parts of the document should be regarded as open for discussion, unless the RTCWEB chairs have declared consensus on an item.

This document is a work item of the RTCWEB working group.

Requirements Language

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].

Status of this Memo

This Internet-Draft will expire on March 31, 2012.

Copyright Notice

Table of Contents

1. Introduction

The Internet was, from very early in its lifetime, considered a possible vehicle for the deployment of real-time, interactive applications - with the most easily imaginable being audio conversations (aka "Internet telephony") and videoconferencing.

The first attempts to build this were dependent on special networks, special hardware and custom-built software, often at very high prices or at low quality, placing great demands on the infrastructure.

As the available bandwidth has increased, and as processors and other hardware has become ever faster, the barriers to participation have decreased, and it is possible to deliver a satisfactory experience on commonly available computing hardware.

Still, there are a number of barriers to the ability to communicate universally - one of these is that there is, as of yet, no single set of communication protocols that all agree should be made available for communication; another is the sheer lack of universal identification systems (such as is served by telephone numbers or email addresses in other communications systems).

Development of The Universal Solution has proved hard, however, for all the usual reasons. This memo aims to take a more building-block-oriented approach, and try to find consensus on a set of substrate components that we think will be useful in any real-time communications systems.

The last few years have also seen a new platform rise for deployment of services: The browser-embedded application, or "Web application". It turns out that as long as the browser platform has the necessary interfaces, it is possible to deliver almost any kind of service on it.

Traditionally, these interfaces have been delivered by plugins, which had to be downloaded and installed separately from the browser; in the development of HTML5, much promise is seen by the possibility of making those interfaces available in a standardized way within the browser.

This memo specifies a set of building blocks that can be made accessible and controllable through a Javascript API interface in a browser, and which together form a necessary and sufficient set of functions to allow the use of interactive audio and video in applications that communicate directly between browsers across the Internet. The resulting protocol suite is intended to enable all the applications that are described as required scenarios in the RTCWEB

use cases document [I-D.ietf-rtcweb-use-cases-and-requirements].

Other efforts, for instance the W3C WebRTC, Web Applications and Device API working groups, focus on making standardized APIs and interfaces available, within or alongside the HTML5 effort, for those functions; this memo concentrates on specifying the protocols and subprotocols that are needed to specify the interactions that happen across the network.

2. Principles and Terminology

2.1. Goals of this document

The goal of the RTCWEB protocol specification is to specify a set of protocols that, if all are implemented, will allow the implementation to communicate with another implementation using audio, video and auxiliary data sent along the most direct possible path between the participants.

This document is intended to serve as the roadmap to the RTCWEB specifications. It defines terms used by other pieces of specification, lists references to other specifications that don't need further elaboration in the RTCWEB context, and gives pointers to other documents that form part of the RTCWEB suite.

By reading this document and the documents it refers to, it should be possible to have all information needed to implement an RTCWEB compatible implementation.

2.2. Relationship between API and protocol

The total RTCWEB/WEBRTC effort consists of two pieces:

- A protocol specification, done in the IETF
- A Javascript API specification, done in the W3C [webrtc-api]

Together, these two specifications aim to provide an environment where Javascript embedded in any page, viewed in any compatible browser, when suitably authorized by its user, is able to set up communication using audio, video and auxiliary data, where the browser environment does not constrain the types of application in which this functionality can be used.

The protocol specification does not assume that all implementations implement this API; it is not intended to be possible by observing the bits on the wire whether they come from a browser or from another

device implementing this specification.

The goal of cooperation between the protocol specification and the API specification is that for all options and features of the protocol specification, it should be clear which API calls to make to exercise that option or feature; similarly, for any sequence of API calls, it should be clear which protocol options and features will be invoked. Both subject to constraints of the implementation, of course.

2.3. On interoperability and innovation

The "Mission statement of the IETF" [RFC3935] states that "The benefit of a standard to the Internet is in interoperability - that multiple products implementing a standard are able to work together in order to deliver valuable functions to the Internet's users."

Communication on the Internet frequently occurs in two phases:

- Two parties communicate, through some mechanism, what functionality they both are able to support
- They use that shared communicative functionality to communicate, or, failing to find anything in common, give up on communication.

There are often many choices that can be made for communicative functionality; the history of the Internet is rife with the proposal, standardization, implementation, and success or failure of many types of options, in all sorts of protocols.

The goal of having a mandatory to implement function set is to prevent negotiation failure, not to preempt or prevent negotiation.

The presence of a mandatory to implement function set serves as a strong changer of the marketplace of deployment - in that it gives a guarantee that, as long as you conform to a specification, and the other party is willing to accept communication at the base level of that specification, you can communicate successfully.

The alternative - that of having no mandatory to implement - does not mean that you cannot communicate, it merely means that in order to be part of the communications partnership, you have to implement the standard "and then some" - that "and then some" usually being called a profile of some sort; in the version most antithetical to the Internet ethos, that "and then some" consists of having to use a specific vendor's product only.

2.4. Terminology

The following terms are used in this document, and as far as possible across the documents specifying the RTCWEB suite, in the specific meanings given here. Not all terms are used in this document. Other terms are used in their commonly used meaning.

The list is in alphabetical order.

Agent: Undefined term. See "SDP Agent" and "ICE Agent".

API: Application Programming Interface - a specification of a set of calls and events, usually tied to a programming language or an abstract formal specification such as WebIDL, with its defined semantics.

ICE Agent: An implementation of the ICE [RFC5245] protocol. An ICE Agent may also be an SDP Agent, but there exist ICE Agents that do not use SDP (for instance those that use Jingle).

Interactive: Communication between multiple parties, where the expectation is that an action from one party can cause a reaction by another party, and the reaction can be observed by the first party, with the total time required for the action/reaction/ observation is on the order of no more than hundreds of milliseconds.

Media: Audio and video content. Not to be confused with "transmission media" such as wires.

Media path: The path that media data follows from one browser to another.

Protocol: A specification of a set of data units, their representation, and rules for their transmission, with their defined semantics. A protocol is usually thought of as going between systems.

Real-time media: Media where generation of content and display of content are intended to occur closely together in time (on the order of no more than hundreds of milliseconds).

SDP Agent: The protocol implementation involved in the SDP offer/ answer exchange, as defined in [RFC3264] section 3.

Signaling: Communication that happens in order to establish, manage and control media paths.

Signaling Path: The communication channels used between entities participating in signalling to transfer signaling. There may be more entities in the signaling path than in the media path.

NOTE: Where common definitions exist for these terms, those definitions should be used to the greatest extent possible.

TODO: Extend this list with other terms that might prove slippery.

3. Architecture and Functionality groups

The model of real-time support for browser-based applications does not envisage that the browser will contain all the functions that need to be performed in order to have a function such as a telephone or a videoconferencing unit; the vision is that the browser will have the functions that are needed for a Web application, working in conjunction with its backend servers, to implement these functions.

This means that two vital interfaces need specification: The protocols that browsers talk to each other, without any intervening servers, and the APIs that are offered for a Javascript application to take advantage of the browser’s functionality.

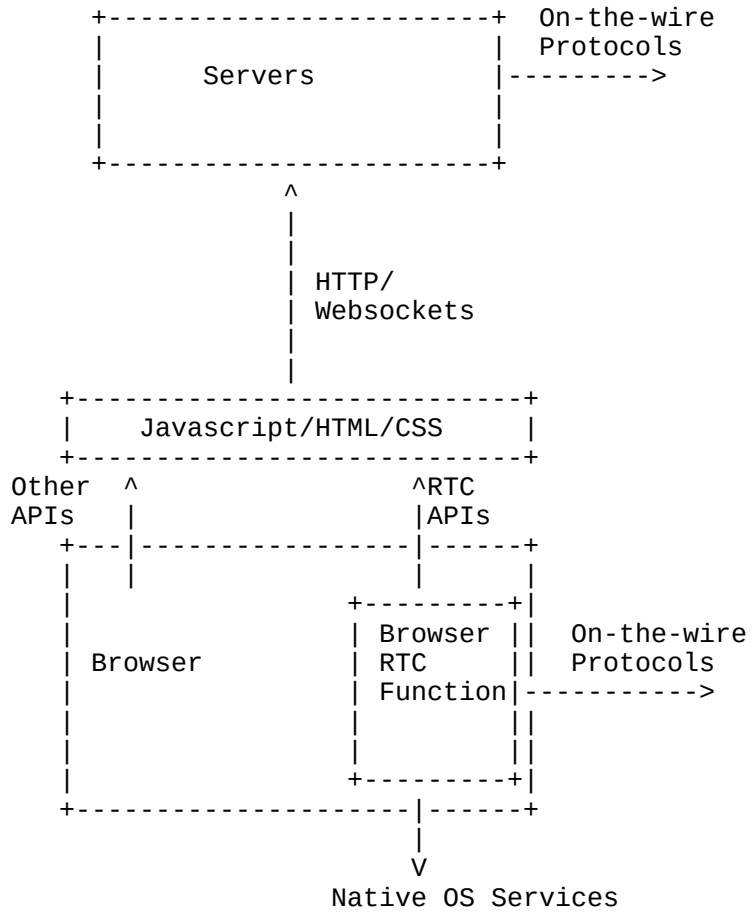


Figure 1: Browser Model

As for all protocol and API specifications, there is no restriction that the protocols can only be used to talk to another browser; since they are fully specified, any device that implements the protocols faithfully should be able to interoperate with the application running in the browser.

A commonly imagined model of deployment is the one depicted below.

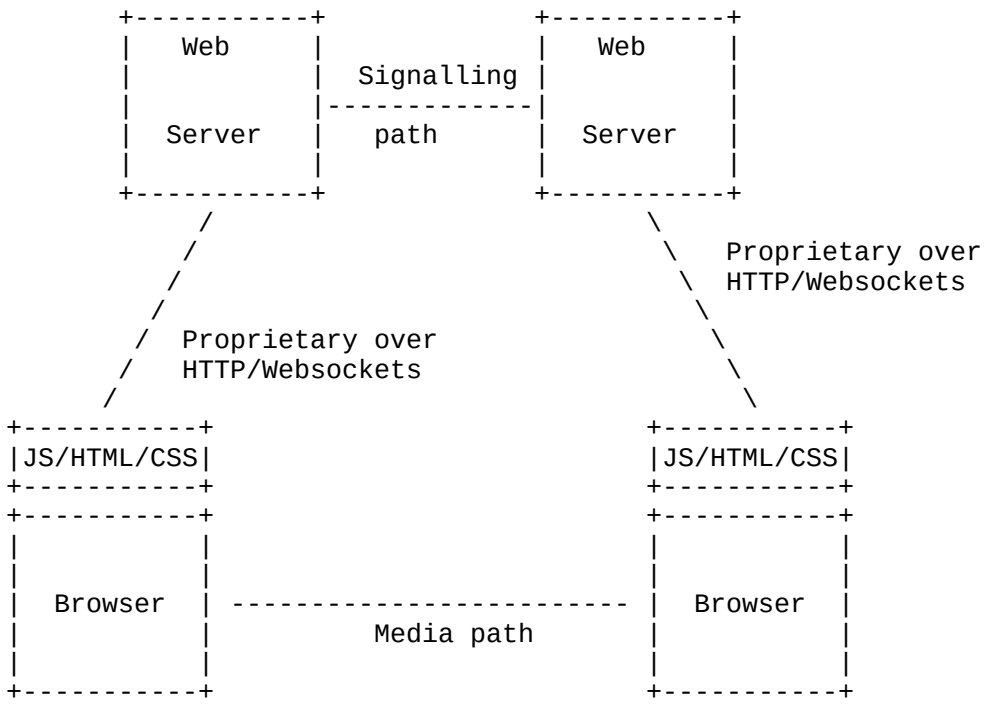


Figure 2: Browser RTC Trapezoid

If the two Web servers are operated by different entities, the signalling path needs to be agreed upon, either by standardization or by other means of agreement; for example, both servers might implement SIP, and the servers would talk SIP to each other, and each would translate between the SIP protocol and their proprietary representation for sending to their application running in the browser. This part is outside the scope of the RTCWEB standars suite.

On this drawing, the critical part to note is that the media path ("low path") goes directly between the browsers, so it has to be conformant to the specifications of the RTCWEB protocol suite; the signalling path ("high path") goes via servers that can modify, translate or massage the signals as needed.

The functionality groups that are needed in the browser can be specified, more or less from the bottom up, as:

- Data transport: TCP, UDP and the means to securely set up connections between entities, as well as the functions for deciding when to send data: Congestion management, bandwidth

estimation and so on.

- Data framing: RTP and other data formats that serve as containers, and their functions for data confidentiality and integrity.

- Data formats: Codec specifications, format specifications and functionality specifications for the data passed between systems. Audio and video codecs, as well as formats for data and document sharing, belong in this category. In order to make use of data formats, a way to describe them, a session description, is needed.

- Connection management: Setting up connections, agreeing on data formats, changing data formats during the duration of a call; SIP and Jingle/XMPP belong in this category.

- Presentation and control: What needs to happen in order to ensure that interactions behave in a non-surprising manner. This can include floor control, screen layout, voice activated image switching and other such functions - where part of the system require the cooperation between parties. Cisco/Tandberg's TIP was one attempt at specifying this functionality.

- Local system support functions: These are things that need not be specified uniformly, because each participant may choose to do these in a way of the participant's choosing, without affecting the bits on the wire in a way that others have to be cognizant of. Examples in this category include echo cancellation (some forms of it), local authentication and authorization mechanisms, OS access control and the ability to do local recording of conversations.

Within each functionality group, it is important to preserve both freedom to innovate and the ability for global communication. Freedom to innovate is helped by doing the specification in terms of interfaces, not implementation; any implementation able to communicate according to the interfaces is a valid implementation. Ability to communicate globally is helped both by having core specifications be unencumbered by IPR issues and by having the formats and protocols be fully enough specified to allow for independent implementation.

One can think of the three first groups as forming a "media transport infrastructure", and of the three last groups as forming a "media service". In many contexts, it makes sense to use a common specification for the media transport infrastructure, which can be embedded in browsers and accessed using standard interfaces, and "let a thousand flowers bloom" in the "media service" layer; to achieve interoperable services, however, at least the first five of the six groups need to be specified.

4. Data transport

Data transport refers to the sending and receiving of data over the network interfaces, the choice of network-layer addresses at each end of the communication, and the interaction with any intermediate entities that handle the data, but do not modify it (such as TURN relays).

It includes necessary functions for congestion control: When not to send data.

The data transport protocols used by RTCWEB are described in <WORKING GROUP DRAFT "TRANSPORTS">.

ICE is required for all media paths that use UDP; in addition to the ability to pass NAT boxes, ICE fulfils the need for guaranteeing that the media path is going to an UDP port that is willing to receive the data.

The details of interactions with intermediate boxes, such as firewalls, relays and NAT boxes, is described in <WORKING GROUP DRAFT "PEER TO PEER CONNECTIVITY">.

5. Data framing and securing

The format for media transport is RTP [RFC3550]. Implementation of SRTP [RFC3711] is required for all implementations.

The detailed considerations for usage of functions from RTP and SRTP are given in [I-D.ietf-rtcweb-rtp-usage]. Key negotiation for SRTP is described in <MISSING>. Transfer of data that is not in RTP format is described in <MISSING>.

6. Data formats

The intent of this specification is to allow each communications event to use the data formats that are best suited for that particular instance, where a format is supported by both sides of the connection. However, a minimum standard is greatly helpful in order to ensure that communication can be achieved. This document specifies a minimum baseline that will be supported by all implementations of this specification, and leaves further codecs to be included at the will of the implementor.

The mandatory to implement codecs, as well as any profiling requirements for both mandatory and optional codecs, is described in

<WORKING GROUP DRAFT "MEDIA PROCESSING">.

7. Connection management

The methods, mechanisms and requirements for setting up, negotiating and tearing down connections is a large subject, and one where it is desirable to have both interoperability and freedom to innovate.

The following principles apply:

1. The media negotiations will be done using the same SDP offer/ answer semantics that are used in SIP [RFC3264], in such a way that it is possible to build a signalling gateway between SIP and the RTCWEB media negotiation.

2. It will be possible to gateway between legacy SIP devices that support ICE and appropriate RTP / SDP mechanisms and codecs without using a media gateway. A signaling gateway to convert between the signaling on the web side to the SIP signaling may be needed.

3. When a new codec is specified, and the SDP for the new codec is specified in the MMUSIC WG, no other standardization would should be required for it to be possible to use that in the web browsers. Adding new codecs which might have new SDP parameters should not change the APIs between the browser and javascript application. As soon as the browsers support the new codecs, old applications written before the codecs were specified should automatically be able to use the new codecs where appropriate with no changes to the JS applications.

The particular choices made for RTCWEB are described in <WORKING GROUP DRAFT "SIGNALING AND NEGOTIATION">.

8. Presentation and control

The most important part of control is the user's control over the browser's interaction with input/output devices and communications channels. It is important that the user have some way of figuring out where his audio, video or texting is being sent, for what purported reason, and what guarantees are made by the parties that form part of this control channel. This is largely a local function between the browser, the underlying operating system and the user interface; this is being worked on as part of the W3C API effort, and will be part of [webrtc-api]

9. Local system support functions

These are characterized by the fact that the quality of these functions strongly influences the user experience, but the exact algorithm does not need coordination. In some cases (for instance echo cancellation, as described below), the overall system definition may need to specify that the overall system needs to have some characteristics for which these facilities are useful, without requiring them to be implemented a certain way.

Local functions include echo cancellation, volume control, camera management including focus, zoom, pan/tilt controls (if available), and more.

Certain parts of the system SHOULD conform to certain properties, for instance:

- Echo cancellation should be good enough to achieve the suppression of acoustical feedback loops below a perceptually noticeable level.
- Privacy concerns must be satisfied; for instance, if remote control of camera is offered, the APIs should be available to let the local participant to figure out who’s controlling the camera, and possibly decide to revoke the permission for camera usage.
- Automatic gain control, if present, should normalize a speaking voice into <whatever dB metrics makes sense here - most important that we have one only>

The requirements on RTCWEB systems in this category are found in <WORKING GROUP DRAFT "MEDIA PROCESSING">.

10. IANA Considerations

This document makes no request of IANA.

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

11. Security Considerations

Security of the web-enabled real time communications comes in several pieces:

- Security of the components: The browsers, and other servers involved. The most target-rich environment here is probably the browser; the aim here should be that the introduction of these components introduces no additional vulnerability.

- Security of the communication channels: It should be easy for a participant to reassure himself of the security of his communication - by verifying the crypto parameters of the links he himself participates in, and to get reassurances from the other parties to the communication that they promise that appropriate measures are taken.

- Security of the partners' identity: verifying that the participants are who they say they are (when positive identification is appropriate), or that their identity cannot be uncovered (when anonymity is a goal of the application).

The security analysis, and the requirements derived from that analysis, is contained in [I-D.ietf-rtcweb-security].

12. Acknowledgements

The number of people who have taken part in the discussions surrounding this draft are too numerous to list, or even to identify. The ones below have made special, identifiable contributions; this does not mean that others' contributions are less important.

Thanks to Cary Bran, Cullen Jennings, Colin Perkins, Magnus Westerlund and Joerg Ott, who offered technical contributions on various versions of the draft.

Thanks to Jonathan Rosenberg, Matthew Kaufman and others at Skype for the ASCII drawings in section 1.

Thanks to Justin Uberti, Henry Sinnreich, Colin Perkins and Simon Leinen for document review.

13. References

13.1. Normative References

[I-D.ietf-rtcweb-rtp-usage]
Perkins, C., Westerlund, M., and J. Ott, "RTP Requirements for RTC-Web", draft-ietf-rtcweb-rtp-usage-00 (work in progress), September 2011.

[I-D.ietf-rtcweb-security]
Rescorla, E., "Security Considerations for RTC-Web", draft-ietf-rtcweb-security-00 (work in progress), September 2011.

[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.

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

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

13.2. Informative References

[I-D.ietf-rtcweb-use-cases-and-requirements]
Holmberg, C., Hakansson, S., and G. Eriksson, "Web Real-Time Communication Use-cases and Requirements", draft-ietf-rtcweb-use-cases-and-requirements-05 (work in progress), September 2011.

[RFC3935] Alvestrand, H., "A Mission Statement for the IETF", BCP 95, RFC 3935, October 2004.

[RFC5245] Rosenberg, J., "Interactive Connectivity Establishment (ICE): A Protocol for Network Address Translator (NAT) Traversal for Offer/Answer Protocols", RFC 5245, April 2010.

[webrtc-api]
Bergkvist, Burnett, Jennings, Narayanan, "WebRTC 1.0: Real-time Communication Between Browsers", August 2011.

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

Appendix A. Change log

This section may be deleted by the RFC Editor when preparing for publication.

A.1. Changes from draft-alvestrand-dispatch-rtcweb-datagram-00 to -01

Added section "On interoperability and innovation"

Added data confidentiality and integrity to the "data framing" layer

Added congestion management requirements in the "data transport" layer section

Changed need for non-media data from "question: do we need this?" to "Open issue: How do we do this?"

Strengthened disclaimer that listed codecs are placeholders, not decisions.

More details on why the "local system support functions" section is there.

A.2. Changes from draft-alvestrand-dispatch-01 to draft-alvestrand-rtcweb-overview-00

Added section on "Relationship between API and protocol"

Added terminology section

Mentioned congestion management as part of the "data transport" layer in the layer list

A.3. Changes from draft-alvestrand-rtcweb-00 to -01

Removed most technical content, and replaced with pointers to drafts as requested and identified by the RTCWEB WG chairs.

Added content to acknowledgements section.

Added change log.

Spell-checked document.

A.4. Changes from draft-alvestrand-rtcweb-overview-01 to draft-ietf-rtcweb-overview-00

Changed draft name and document date.

Removed unused references

A.5. Changes from draft-ietf-rtcweb-overview -00 to -01

Added architecture figures to section 2.

Changed the description of "echo cancellation" under "local system support functions".

Added a few more definitions.

A.6. Changes from draft-ietf-rtcweb-overview -01 to -02

Added pointers to use cases, security and rtp-usage drafts (now WG drafts).

Changed description of SRTP from mandatory-to-use to mandatory-to-implement.

Added the "3 principles of negotiation" to the connection management section.

Added an explicit statement that ICE is required for both NAT and consent-to-receive.

Author's Address

Harald T. Alvestrand
Google
Kungsbron 2
Stockholm, 11122
Sweden

Email: harald@alvestrand.no

Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: May 3, 2012

C. Perkins
University of Glasgow
M. Westerlund
Ericsson
J. Ott
Aalto University
October 31, 2011

Web Real-Time Communication (WebRTC): Media Transport and Use of RTP
draft-ietf-rtcweb-rtp-usage-01

Abstract

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

Table of Contents

1. Introduction

The Real-time Transport Protocol (RTP) [RFC3550] was designed to provide a framework for delivery of audio and video teleconferencing data and other real-time media applications. This memo describes how RTP is to be used in the context of the Web Real-Time Communication (WebRTC) framework, a new activity that aims to provide support for direct, interactive, and rich communication using audio, video, collaboration, games, etc. between two peers' web-browsers.

Previous work in the IETF Audio/Video Transport Working Group, and it's successors, has been about providing a framework for real-time multimedia transport, but has not specified how the pieces of this framework should be combined. This is because the choice of building blocks and protocol features can really only be done in the context of some application. This memo proposes a set of RTP features and extensions to be implemented by applications that fit within the WebRTC application context. This includes applications such as voice over IP (VoIP), video teleconferencing, and on-demand multimedia streaming, delivered in the context of the WebRTC browser-based infrastructure.

2. Terminology

This memo is structured into different topics. For each topic, one or several recommendations from the authors are given. When it comes to the importance of extensions, or the need for implementation support, we use three requirement levels to indicate the importance of the feature to the WebRTC specification:

REQUIRED: Functionality that is absolutely needed to make the WebRTC solution work well, or functionality of low complexity that provides high value.

RECOMMENDED: Should be included as its brings significant benefit, but the solution can potentially work without it.

OPTIONAL: Something that is useful in some cases, but not always a benefit.

3. Media Transport in WebRTC

3.1. Expected Topologies

As WebRTC is focused on peer to peer connections established from clients in web browsers the following topologies further discussed in

RTP Topologies [RFC5117] are primarily considered. The topologies are depicted and briefly explained here for ease of the reader.

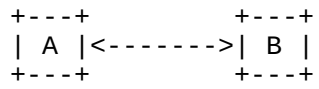


Figure 1: Point to Point

The point to point topology (Figure 1) is going to be very common in any single user to single user applications.

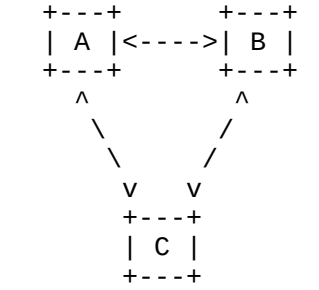


Figure 2: Multi-unicast

For small multiparty sessions it is practical enough to create RTP sessions by letting every participant send individual unicast RTP/UDP flows to each of the other participants. This is called multi-unicast (Figure 2), and is unfortunately not discussed in the RTP Topologies [RFC5117]. This topology has the benefit of not requiring central nodes. The downside is that it increases the used bandwidth at each sender by requiring one copy of the media streams for each participant that are part of the same session beyond the sender itself. Thus this is limited to scenarios with few end-points unless the media is very low bandwidth.

This topology may be implemented as a single RTP session, spanning multiple peer to peer transport layer connections, or as several pairwise RTP sessions, one between each pair of peers. The later approach simplifies rate adaptation, but reduces the effectiveness of RTCP for debugging purposes, by limiting the ability to send third-party RTCP reports.

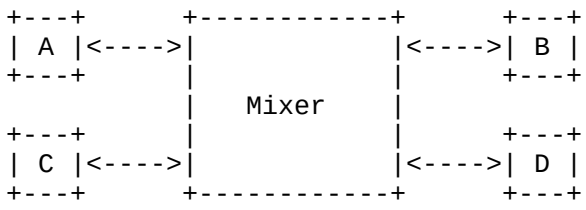


Figure 3: RTP Mixer with Only Unicast Paths

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

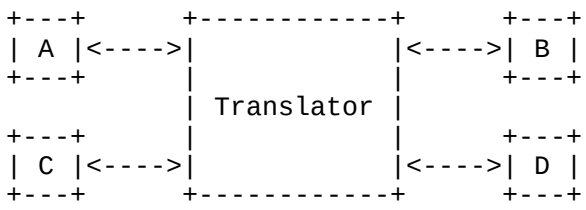


Figure 4: RTP Translator (Relay) with Only Unicast Paths

If one wants a less complex central node it is possible to use an relay (called an Transport Translator) (Figure 4) that takes on the role of forwarding the media to the other end-points but doesn't perform any media processing. It simply forwards the media from all other to all the other. Thus one endpoint A will only need to send a media once to the relay, but it will still receive 3 RTP streams with the media if B, C and D all currently transmits.

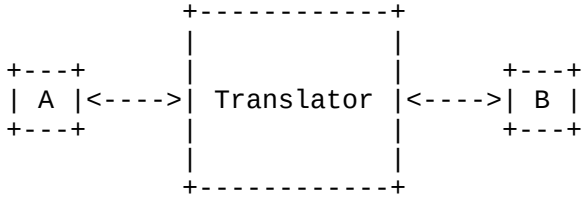


Figure 5: Translator towards Legacy end-point

To support legacy end-point (B) that don't fulfil the requirements of WebRTC it is possible to insert a Translator (Figure 5) that takes on the role to ensure that from A's perspective B looks like a fully compliant end-point. Thus it is the combination of the Translator and B that looks like the end-point B. The intention is that the presence of the translator is transparent to A, however it is not certain that is possible. Thus this case is include so that it can be discussed if any mechanism specified to be used for WebRTC results in such issues and how to handle them.

3.2. Requirements from RTP

This section discusses some requirements RTP and RTCP [RFC3550] place on their underlying transport protocol, the signalling channel, etc.

3.2.1. Signalling for RTP sessions

RTP is built with the assumption of an external to RTP/RTCP signalling channel to configure the RTP sessions and its functions. The basic configuration of an RTP session consists of the following parameters:

RTP Profile: The name of the RTP profile to be used in session. The RTP/AVP [RFC3551] and RTP/AVPF [RFC4585] profiles can interoperate on basic level, as can their secure variants RTP/SAVP [RFC3711] and RTP/SAVPF [RFC5124]. The secure variants of the profiles do not directly interoperate with the non-secure variants, due to the presence of additional header fields in addition to any cryptographic transformation of the packet content.

Transport Information: Source and destination address(s) and ports for RTP and RTCP MUST be signalled for each RTP session. If RTP and RTCP multiplexing [RFC5761] is to be used, such that a single port is used for RTP and RTCP flows, this MUST be signalled (see Section 5.1). If several RTP sessions are to be multiplexed onto a single transport layer flow, this MUST also be signalled (see Section 4.4).

RTP Payload Types, media formats, and media format parameters: The mapping between media type names (and hence the RTP payload formats to be used) and the RTP payload type numbers must be signalled. Each media type may also have a number of media type parameters that must also be signalled to configure the codec and RTP payload format (the "a=fmtp:" line from SDP).

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

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

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

3.2.2. (Lack of) Signalling for Payload Format Changes

As discussed in Section 3.2.1, the mapping between media type name, and its associated RTP payload format, and the RTP payload type number to be used for that format must be signalled as part of the session setup. An endpoint may signal support for multiple media formats, or multiple configurations of a single format, each using a different RTP payload type number. If multiple formats are signalled by an endpoint, that endpoint is REQUIRED to be prepared to receive data encoded in any of those formats at any time (this is slightly modified if several RTP sessions are multiplexed onto one transport layer connection, such that an endpoint must be prepared for a source to switch between formats of the same media type at any time; see Section 4.4). RTP does not require advance signalling for changes between formats that were signalled during the session setup. This is needed for rapid rate adaptation.

4. WebRTC Use of RTP: Core Protocols

4.1. RTP and RTCP

The Real-time Transport Protocol (RTP) [RFC3550] is REQUIRED to be implemented as the media transport protocol for WebRTC. RTP itself comprises two parts: the RTP data transfer protocol, and the RTP

control protocol (RTCP). RTCP is a fundamental and integral part of the RTP protocol, and is REQUIRED to be implemented.

RTP and RTCP are flexible and extensible protocols that allow, on the one hand, choosing from a variety of building blocks and combining those to meet application needs, but on the other hand, offer the ability to create extensions where existing mechanisms are not sufficient. This memo requires a number of RTP and RTCP extensions that have been shown to be provide important functionality in the WebRTC context be implemented. It is possible that future extensions will be needed: several documents provide guidelines for the use and extension of RTP and RTCP, including Guidelines for Writers of RTP Payload Format Specifications [RFC2736] and Guidelines for Extending the RTP Control Protocol [RFC5968], and should be consulted before extending this memo.

4.2. Choice of RTP Profile

The complete specification of RTP for a particular application domain requires the choice of an RTP Profile. For WebRTC use, the "Extended Secure RTP Profile for Real-time Transport Control Protocol (RTCP)-Based Feedback (RTP/SAVPF)" [RFC5124] is REQUIRED to be implemented. This builds on the basic RTP/AVP profile [RFC3551], the RTP/AVPF feedback profile [RFC4585], and the secure RTP/SAVP profile [RFC3711].

The RTP/AVPF part of RTP/SAVPF is required to get the improved RTCP timer model, that allows more flexible transmission of RTCP packets in response to events, rather than strictly according to bandwidth. This also saves RTCP bandwidth and will commonly only use the full amount when there is a lot of events on which to send feedback. This functionality is needed to make use of the RTP conferencing extensions discussed in Section 6.1. The improved RTCP timer model defined by RTP/AVPF is backwards compatible with legacy systems that implement only the RTP/AVP profile given some constraints on parameter configuration such as RTCP banwidth and "trr-int".

The RTP/SAVP part of RTP/SAVPF is for support for Secure RTP (SRTP) [RFC3711]. This provides media encryption, integrity protection, replay protection and a limited form of source authentication. It does not contain a specific keying mechanism, so that, and the set of security transforms, will be required to be chosen. It is possible that a security mechanism operating on a lower layer than RTP can be used instead and that should be evaluated. However, the reasons for the design of SRTP should be taken into consideration in that discussion. A mandatory to implement media security mechanism including keying must be required so that confidentialtiy, integrity protection and source authentication of the media stream can be

provided when desired by the user.

4.3. Choice of RTP Payload Formats

(tbd: say something about the choice of RTP Payload Format for WebRTC. If there is a mandatory to implement set of codecs, this should reference them. In any case, it should reference a discussion of signalling for the choice of codec, once that discussion reaches closure.)

4.4. RTP Session Multiplexing

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

In today's networks, however, with the prolific use of Network Address Translators (NAT) and Firewalls (FW), there is a desire to reduce the number of transport layer ports used by real-time media applications using RTP by combining multimedia traffic in a single RTP session. (Details of how this is to be done are tbd, but see [I-D.lennox-rtcweb-rtp-media-type-mux], [I-D.holmberg-mmusic-sdp-bundle-negotiation] and [I-D.westerlund-avtcore-multiplex-architecture].) WebRTC implementations of RTP are REQUIRED to support multiplexing of multimedia sessions onto a single RTP session according to (tbd). If such RTP session multiplexing is to be used, this MUST be negotiated during the signalling phase.

5. WebRTC Use of RTP: Optimisations

This section discusses some optimisations that makes RTP/RTCP work better and more efficient and therefore are considered.

5.1. RTP and RTCP Multiplexing

Historically, RTP and RTCP have been run on separate UDP ports. With the increased use of Network Address/Port Translation (NAPT) this has become problematic, since maintaining multiple NAT bindings can be costly. It also complicates firewall administration, since multiple

ports must be opened to allow RTP traffic. To reduce these costs and session setup times, support for multiplexing RTP data packets and RTCP control packets on a single port [RFC5761] is REQUIRED. Supporting this specification is generally a simplification in code, since it relaxes the tests in [RFC3550].

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

5.2. Reduced Size RTCP

RTCP packets are usually sent as compound RTCP packets; and RFC 3550 demands that those compound packets always start with an SR or RR packet. However, especially when using frequent feedback messages, these general statistics are not needed in every packet and unnecessarily increase the mean RTCP packet size and thus limit the frequency at which RTCP packets can be sent within the RTCP bandwidth share.

RFC5506 "Support for Reduced-Size Real-Time Transport Control Protocol (RTCP): Opportunities and Consequences" [RFC5506] specifies how to reduce the mean RTCP message and allow for more frequent feedback. Frequent feedback, in turn, is essential to make real-time application quickly aware of changing network conditions and allow them to adapt their transmission and encoding behaviour.

Support for RFC5506 is REQUIRED.

5.3. Symmetric RTP/RTCP

RTP entities choose the RTP and RTCP transport addresses, i.e., IP addresses and port numbers, to receive packets on and bind their respective sockets to those. When sending RTP packets, however, they may use a different IP address or port number for RTP, RTCP, or both; e.g., when using a different socket instance for sending and for receiving. Symmetric RTP/RTCP requires that the IP address and port number for sending and receiving RTP/RTCP packets are identical.

The reasons for using symmetric RTP is primarily to avoid issues with NAT and Firewalls by ensuring that the flow is actually bi-directional and thus kept alive and registered as flow the intended recipient actually wants. In addition it saves resources in the form of ports at the end-points, but also in the network as NAT mappings or firewall state is not unnecessary bloated. Also the number of QoS state are reduced.

Using Symmetric RTP and RTCP [RFC4961] is REQUIRED.

5.4. Generation of the RTCP Canonical Name (CNAME)

The RTCP Canonical Name (CNAME) provides a persistent transport-level identifier for an RTP endpoint. While the Synchronisation Source (SSRC) identifier for an RTP endpoint may change if a collision is detected, or when the RTP application is restarted, it's RTCP CNAME is meant to stay unchanged, so that RTP endpoints can be uniquely identified and associated with their RTP media streams. For proper functionality, RTCP CNAMEs should be unique among the participants of an RTP session.

The RTP specification [RFC3550] includes guidelines for choosing a unique RTP CNAME, but these are not sufficient in the presence of NAT devices. In addition, some may find long-term persistent identifiers problematic from a privacy viewpoint. Accordingly, support for generating a short-term persistent RTCP CNAMEs following method (b) as specified in Section 4.2 of "Guidelines for Choosing RTP Control Protocol (RTCP) Canonical Names (CNAMEs)" [RFC6222] is RECOMMENDED, since this addresses both concerns.

6. WebRTC Use of RTP: Extensions

There are a number of RTP extensions that could be very useful in the WebRTC context. One set is related to conferencing, others are more generic in nature.

6.1. Conferencing Extensions

RTP is inherently a group communication protocol. Groups can be implemented using a centralised server, multi-unicast, or IP multicast. While IP multicast was popular in early deployments, in today's practice, overlay-based conferencing dominates, typically using one or more central servers to connect endpoints in a star or flat tree topology. These central servers can be implemented in a number of ways [RFC5117], of which the following are the most common:

1. RTP Translator (Relay) with Only Unicast Paths ([RFC5117], section 3.3)

2. RTP Mixer with Only Unicast Paths ([RFC5117], section 3.4)

3. Point to Multipoint Using a Video Switching MCU ([RFC5117], section 3.5)

4. Point to Multipoint Using Content Modifying MCUs ([RFC5117], section 3.6)

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

RTP protocol extensions to be used with conferencing are included because they are important in the context of centralised conferencing, where one RTP Mixer (Conference Focus) receives a participants media streams and distribute them to the other participants. These messages are defined in the Extended RTP Profile for Real-time Transport Control Protocol (RTCP)-Based Feedback (RTP/AVPF) [RFC4585] and the "Codec Control Messages in the RTP Audio-Visual Profile with Feedback (AVPF)" (CCM) [RFC5104] and are fully usable by the Secure variant of this profile (RTP/SAVPF) [RFC5124].

6.1.1. Full Intra Request

The Full Intra Request is defined in Sections 3.5.1 and 4.3.1 of CCM [RFC5104]. It is used to have the mixer request from a session participants a new Intra picture. This is used when switching between sources to ensure that the receivers can decode the video or other predicted media encoding with long prediction chains. It is RECOMMENDED that this feedback message is supported.

6.1.2. Picture Loss Indication

The Picture Loss Indication is defined in Section 6.3.1 of the RTP/AVPF profile [RFC4585]. It is used by a receiver to tell the encoder that it lost the decoder context and would like to have it repaired somehow. This is semantically different from the Full Intra Request above. It is RECOMMENDED that this feedback message is supported as a loss tolerance mechanism.

6.1.3. Slice Loss Indication

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

6.1.4. Reference Picture Selection Indication

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

6.1.5. Temporary Maximum Media Stream Bit Rate Request

This feedback message is defined in Section 3.5.4 and 4.2.1 in CCM [RFC5104]. This message and its notification message is used by a media receiver, to inform the sending party that there is a current limitation on the amount of bandwidth available to this receiver. This can be for various reasons, and can for example be used by an RTP mixer to limit the media sender being forwarded by the mixer (without doing media transcoding) to fit the bottlenecks existing towards the other session participants. It is RECOMMENDED that this feedback message is supported.

6.2. Header Extensions

The RTP specification [RFC3550] provides a capability to extend the RTP header with in-band data, but the format and semantics of the extensions are poorly specified. Accordingly, if header extensions are to be used, it is REQUIRED that they be formatted and signalled according to the general mechanism of RTP header extensions defined in [RFC5285].

As noted in [RFC5285], the requirement from the RTP specification that header extensions are "designed so that the header extension may be ignored" [RFC3550] stands. To be specific, header extensions must only be used for data that can safely be ignored by the recipient without affecting interoperability, and must not be used when the presence of the extension has changed the form or nature of the rest of the packet in a way that is not compatible with the way the stream is signalled (e.g., as defined by the payload type). Valid examples might include metadata that is additional to the usual RTP information.

The RTP rapid synchronisation header extension [RFC6051] is recommended, as discussed in Section 6.3 we also recommend the client to mixer audio level [I-D.ietf-avtext-client-to-mixer-audio-level], and consider the mixer to client audio level [I-D.ietf-avtext-mixer-to-client-audio-level] as optional feature.

It is REQUIRED that the mechanism to encrypt header extensions [I-D.ietf-avtcore-srtp-encrypted-header-ext] is implemented when the client-to-mixer or mixer-to-client audio level indications are in use in SRTP encrypted sessions, since the information contained in these header extensions may be considered sensitive.

6.3. Rapid Synchronisation Extensions

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

6.4. Mixer Audio Level Extensions

6.4.1. Client to Mixer Audio Level

The Client to Mixer Audio Level [I-D.ietf-avtext-client-to-mixer-audio-level] is an RTP header extension used by a client to inform a mixer about the level of audio activity in the packet the header is attached to. This enables a central node to make mixing or selection decisions without decoding or detailed inspection of the payload. Thus reducing the needed complexity in some types of central RTP nodes.

Assuming that the Client to Mixer Audio Level [I-D.ietf-avtext-client-to-mixer-audio-level] is published as a finished specification prior to RTCWEB’s first RTP specification then it is RECOMMENDED that this extension is included.

6.4.2. Mixer to Client Audio Level

The Mixer to Client Audio Level header extension [I-D.ietf-avtext-mixer-to-client-audio-level] provides the client with the audio level of the different sources mixed into a common mix from the RTP mixer. Thus enabling a user interface to indicate the relative activity level of a session participant, rather than just being included or not based on the CSRC field. This is a pure optimisations of non critical functions and thus optional functionality.

Assuming that the Mixer to Client Audio Level [I-D.ietf-avtext-client-to-mixer-audio-level] is published as a finished specification prior to RTCWEB’s first RTP specification then

it is OPTIONAL that this extension is included.

7. WebRTC Use of RTP: Improving Transport Robustness

There are some tools that can make RTP flows robust against Packet loss and reduce the impact on media quality. However they all add extra bits compared to a non-robust stream. These extra bits needs to be considered and the aggregate bit-rate needs to be rate controlled. Thus improving robustness might require a lower base encoding quality but has the potential to give that quality with fewer errors in it.

7.1. Retransmission

Support for RTP retransmission as defined by "RTP Retransmission Payload Format" [RFC4588] is RECOMMENDED.

The retransmission scheme in RTP allows flexible application of retransmissions. Only selected missing packets can be requested by the receiver. It also allows for the sender to prioritise between missing packets based on senders knowledge about their content. Compared to TCP, RTP retransmission also allows one to give up on a packet that despite retransmission(s) still has not been received within a time window.

"WebRTC Media Transport Requirements" [I-D.cbran-rtcweb-data] raises two issues that they think makes RTP Retransmission unsuitable for RTCWEB. We here consider these issues and explain why they are in fact not a reason to exclude RTP retransmission from the tool box available to RTCWEB media sessions.

The additional latency added by [RFC4588] will exceed the latency threshold for interactive voice and video: RTP Retransmission will require at least one round trip time for a retransmission request and repair packet to arrive. Thus the general suitability of using retransmissions will depend on the actual network path latency between the end-points. In many of the actual usages the latency between two end-points will be low enough for RTP retransmission to be effective. Interactive communication with end-to-end delays of 400 ms still provide a fair quality. Even removing half of that in end-point delays allows functional retransmission between end-points on the same continent. In addition, some applications may accept temporary delay spikes to allow for retransmission of crucial codec information such an parameter sets, intra picture etc, rather than getting no media at all.

The undesirable increase in packet transmission at the point when congestion occurs: Congestion loss will impact the rate controls view of available bit-rate for transmission. When using retransmission one will have to prioritise between performing retransmissions and the quality one can achieve with ones adaptable codecs. In many use cases one prefer error free or low rates of error with reduced base quality over high degrees of error at a higher base quality.

The RTCWEB end-point implementations will need to both select when to enable RTP retransmissions based on API settings and measurements of the actual round trip time. In addition for each NACK request that a media sender receives it will need to make a prioritisation based on the importance of the requested media, the probability that the packet will reach the receiver in time for being usable, the consumption of available bit-rate and the impact of the media quality for new encodings.

To conclude, the issues raised are implementation concerns that an implementation needs to take into consideration, they are not arguments against including a highly versatile and efficient packet loss repair mechanism.

7.2. Forward Error Correction (FEC)

Support of some type of FEC to combat the effects of packet loss is beneficial, but is heavily application dependent. However, some FEC mechanisms are encumbered.

The main benefit from FEC is the relatively low additional delay needed to protect against packet losses. The transmission of any repair packets should preferably be done with a time delay that is just larger than any loss events normally encountered. That way the repair packet isn’t also lost in the same event as the source data.

The amount of repair packets needed varies depending on the amount and pattern of packet loss to be recovered, and on the mechanism used to derive repair data. The later choice also effects the the additional delay required to both encode the repair packets and in the receiver to be able to recover the lost packet(s).

7.2.1. Basic Redundancy

The method for providing basic redundancy is to simply retransmit a some time earlier sent packet. This is relatively simple in theory, i.e. one saves any outgoing source (original) packet in a buffer marked with a timestamp of actual transmission, some X ms later one transmit this packet again. Where X is selected to be longer than

the common loss events. Thus any loss events shorter than X can be recovered assuming that one doesn't get an another loss event before all the packets lost in the first event has been received.

The downside of basic redundancy is the overhead. To provide each packet with once chance of recovery, then the transmission rate increases with 100% as one needs to send each packet twice. It is possible to only redundantly send really important packets thus reducing the overhead below 100% for some other trade-off is overhead.

In addition the basic retransmission of the same packet using the same SSRC in the same RTP session is not possible in RTP context. The reason is that one would then destroy the RTCP reporting if one sends the same packet twice with the same sequence number. Thus one needs more elaborate mechanisms.

RTP Payload Format Support: Some RTP payload format do support basic redundancy within the RTP paylaod format itself. Examples are AMR-WB [RFC4867] and G.719 [RFC5404].

RTP Payload for Redundant Audio Data: This audio and text redundancy format defined in [RFC2198] allows for multiple levels of redundancy with different delay in their transmissions, as long as the source plus payload parts to be redundantly transmitted together fits into one MTU. This should work fine for most interactive audio and text use cases as both the codec bit-rates and the framing intervals normally allow for this requirement to hold. This payload format also don't increase the packet rate, as original data and redundant data are sent together. This format does not allow perfect recovery, only recovery of information deemed necessary for audio, for example the sequence number of the original data is lost.

RTP Retransmission Format: The RTP Retransmission Payload format [RFC4588] can be used to pro-actively send redundant packets using either SSRC or session multiplexing. By using different SSRCs or a different session for the redundant packets the RTCP receiver reports will be correct. The retransmission payload format is used to recover the packets original data thus enabling a perfect recovery.

Duplication Grouping Semantics in the Session Description Protocol: This [I-D.begen-mmusic-redundancy-grouping] is proposal for new SDP signalling to indicate media stream duplication using different RTP sessions, or different SSRCs to separate the source and the redundant copy of the stream.

7.2.2. Block Based FEC

Block based redundancy collects a number of source packets into a data block for processing. The processing results in some number of repair packets that is then transmitted to the other end allowing the receiver to attempt to recover some number of lost packets in the block. The benefit of block based approaches is the overhead which can be lower than 100% and still recover one or more lost source packet from the block. The optimal block codes allows for each received repair packet to repair a single loss within the block. Thus 3 repair packets that are received should allow for any set of 3 packets within the block to be recovered. In reality one commonly don't reach this level of performance for any block sizes and number of repair packets, and taking the computational complexity into account there are even more trade-offs to make among the codes.

One result of the block based approach is the extra delay, as one needs to collect enough data together before being able to calculate the repair packets. In addition sufficient amount of the block needs to be received prior to recovery. Thus additional delay are added on both sending and receiving side to ensure possibility to recover any packet within the block.

The redundancy overhead and the transmission pattern of source and repair data can be altered from block to block, thus allowing a adaptive process adjusting to meet the actual amount of loss seen on the network path and reported in RTCP.

The alternatives that exist for block based FEC with RTP are the following:

RTP Payload Format for Generic Forward Error Correction: This RTP payload format [RFC5109] defines an XOR based recovery packet. This is the simplest processing wise that an block based FEC scheme can be. It also results in some limited properties, as each repair packet can only repair a single loss. To handle multiple close losses a scheme of hierarchical encodings are need. Thus increasing the overhead significantly.

Forward Error Correction (FEC) Framework: This framework [I-D.ietf-fecframe-framework] defines how not only RTP packets but how arbitrary packet flows can be protected. Some solutions produced or under development in FECFRAME WG are RTP specific. There exist alternatives supporting block codes such as Reed-Salomon and Raptor.

7.2.3. Recommendations for FEC

(tbd)

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

WebRTC will be used in very varied network environment with a hetrogenous set of link technologies, including wired and wireless, interconnecting peers at different topological locations resulting in network paths with widely varying one way delays, bit-rate capacity, load levels and traffic mixes. In addition individual end-points will open one or more WebRTC sessions between one or more peers. Each of these session may contain different mixes of media and data flows. Assymetric usage of media bit-rates and number of media streams is also to be expected. A single end-point may receive zero to many simultanous media streams while itself transmitting one or more streams.

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

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

This hetrogenous situation results in a requirement to have functionality that adapts to the available capacity and that competes fairly with other network flows. If it would not compete fairly enough WebRTC could be used as an attack method for starving out other traffic on specific links as long as the attacker is able to create traffic across a specific link. This is not far-fetched for a web-service capable of attracting large number of end-points and use the service, combined with BGP routing state a server could pick client pairs to drive traffic to specific paths.

The above estalish a clear need based on several reasons why there need to be a well working media adaptation mechanism. This mechanism also have a number of requirements on what services it should provide and what performance it needs to provide.

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

8.1. Rate Control Requirements

Note: This section does not yet have WG consensus.

This section provides a number of requirements on an media adaptation/congestion control solution for WebRTC.

1. All WebRTC media streams MUST be congestion-controlled. (The same requirement apply to data streams)

2. The congestion algorithms used MUST cause WebRTC streams to act reasonably fairly with TCP and other congestion-controlled flows, such as DCCP and TFRC, and other WebRTC flows. Note that WebRTC involves multiple data flows which "normally" would be separately congestion-controlled.

3. The congestion control mechanism MUST be possible to realize between two indendently implemented WebRTC end-points.

4. The congestion control algorithm SHOULD attempt to minimize the media-stream end-to-end delays between the participants, by controlling bandwidth appropriately.

5. The congestion control SHOULD allow for prioritization and shifting of banwidth between media flows. In other words, if one flow on the same path as another has to adjust its bit-rate the other flow can perform that adjustment instead, or divided between the flows.

Thus it is REQUIRED to have an implementation of an RTP Rate Control mechanism fulfilling the above requirements.

8.2. RTCP Limiations

Experience with the congestion control algorithms of TCP [RFC5681], TFRC [RFC5348], and DCCP [RFC4341], [RFC4342], [RFC4828], has shown that feedback on packet arrivals needs to be sent roughly once per round trip time. We note that the capabilities of real-time media traffic to adapt to changing path conditions may be less rapid than

for the elastic applications TCP was designed for, but frequent feedback is still required to allow the congestion control algorithm to track the path dynamics.

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

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

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

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

8.3. Legacy Interop Limitations

Congestion control interoperability with most type of legacy devices, even using an translator could be difficult. There are numerous reasons for this:

No RTCP Support: There exist legacy implementations that does not even implement RTCP at all. Thus no feedback at all is provided.

RTP/AVP Minimal RTCP Interval of 5s: RTP [RFC3550] under the RTP/AVP profile specifies a recommended minimal fixed interval of 5 seconds. Sending RTCP report blocks as seldom as 5 seconds makes it very difficult for a sender to use these reports and react to any congestion event.

RTP/AVP Scaled Minimal Interval: If a legacy device uses the scaled minimal RTCP compound interval, the "RECOMMENDED value for the reduced minimum in seconds is 360 divided by the session bandwidth in kilobits/second" ([RFC3550], section 6.2). The minimal interval drops below a second, still several times the RTT in almost all paths in the Internet, when the session bandwidht becomes 360 kbps. A session bandwidth of 1 Mbps still has a minimal interval of 360 ms. Thus, with the exception for rather high bandwidth sessions, getting frequent enough RTCP Report Blocks to report on the order of the RTT is very difficult as long as the legacy device uses the RTP/AVP profile.

RTP/AVPF Supporting Legacy Device: If a legacy device supports RTP/AVPF, then that enables negotation of important parameters for frequent reporting, such as the "trr-int" parameter, and the possibility that the end-point supports some useful feedback format for congestion control purpose such as TMMBR [RFC5104].

It has been suggested on the RTCWEB mailing list that if interoperating with really limited legacy devices an WebRTC end-point may not send more than 64 kbps of media streams, to avoid it causing massive congestion on most paths in the Internet when communicating with a legacy node not providing sufficient feedback for effective congestion control. This warrants further discussion as there is clearly a number of link layers that don’t even provide that amount of bit-rate consistently, and that assumes no competing traffic.

9. WebRTC Use of RTP: Performance Monitoring

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

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

10. IANA Considerations

This memo makes no request of IANA.

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

RFC.

11. Security Considerations

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

The guidelines in [I-D.ietf-avtcore-srtp-vbr-audio] apply when using variable bit rate (VBR) audio codecs, for example Opus or the Mixer audio level header extensions.

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

12. Acknowledgements

The authors would like to thank Harald Alvestrand, Cary Bran, and Cullen Jennings for valuable feedback.

13. References

13.1. Normative References

[I-D.holmberg-mmusic-sdp-bundle-negotiation]
Holmberg, C. and H. Alvestrand, "Multiplexing Negotiation Using Session Description Protocol (SDP) Port Numbers", draft-holmberg-mmusic-sdp-bundle-negotiation-00 (work in progress), October 2011.

[I-D.ietf-avtcore-srtp-encrypted-header-ext]
Lennox, J., "Encryption of Header Extensions in the Secure Real-Time Transport Protocol (SRTP)", draft-ietf-avtcore-srtp-encrypted-header-ext-00 (work in progress), June 2011.

[I-D.ietf-avtcore-srtp-vbr-audio]

Perkins, C. and J. Valin, "Guidelines for the use of Variable Bit Rate Audio with Secure RTP", draft-ietf-avtcore-srtp-vbr-audio-03 (work in progress), July 2011.

[I-D.ietf-avtext-client-to-mixer-audio-level]
Lennox, J., Ivov, E., and E. Marocco, "A Real-Time Transport Protocol (RTP) Header Extension for Client-to-Mixer Audio Level Indication", draft-ietf-avtext-client-to-mixer-audio-level-03 (work in progress), July 2011.

[I-D.ietf-avtext-mixer-to-client-audio-level]
Ivov, E., Marocco, E., and J. Lennox, "A Real-Time Transport Protocol (RTP) Header Extension for Mixer-to-Client Audio Level Indication", draft-ietf-avtext-mixer-to-client-audio-level-03 (work in progress), July 2011.

[I-D.ietf-rtcweb-security]
Rescorla, E., "Security Considerations for RTC-Web", draft-ietf-rtcweb-security-01 (work in progress), October 2011.

[I-D.lennox-rtcweb-rtp-media-type-mux]
Lennox, J. and J. Rosenberg, "Multiplexing Multiple Media Types In a Single Real-Time Transport Protocol (RTP) Session", draft-lennox-rtcweb-rtp-media-type-mux-00 (work in progress), October 2011.

[I-D.westerlund-avtcore-multiplex-architecture]
Westerlund, M., Burman, B., and C. Perkins, "RTP Multiplexing Architecture", draft-westerlund-avtcore-multiplex-architecture-00 (work in progress), October 2011.

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

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

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

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

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

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

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

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

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

[RFC5109] Li, A., "RTP Payload Format for Generic Forward Error Correction", RFC 5109, December 2007.

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

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

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

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

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

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

13.2. Informative References

[I-D.begen-mmusic-redundancy-grouping]
Begen, A., Cai, Y., and H. Ou, "Duplication Grouping Semantics in the Session Description Protocol", draft-begen-mmusic-redundancy-grouping-01 (work in progress), June 2011.

[I-D.cbran-rtcweb-data]
Bran, C. and C. Jennings, "RTC-Web Non-Media Data Transport Requirements", draft-cbran-rtcweb-data-00 (work in progress), July 2011.

[I-D.ietf-fecframe-framework]
Watson, M., Begen, A., and V. Roca, "Forward Error Correction (FEC) Framework", draft-ietf-fecframe-framework-15 (work in progress), June 2011.

[RFC2198] Perkins, C., Kouvelas, I., Hodson, O., Hardman, V., Handley, M., Bolot, J., Vega-Garcia, A., and S. Fosse-Parisis, "RTP Payload for Redundant Audio Data", RFC 2198, September 1997.

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

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

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

[RFC4867] Sjoberg, J., Westerlund, M., Lakaniemi, A., and Q. Xie, "RTP Payload Format and File Storage Format for the Adaptive Multi-Rate (AMR) and Adaptive Multi-Rate Wideband (AMR-WB) Audio Codecs", RFC 4867, April 2007.

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

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

[RFC5404] Westerlund, M. and I. Johansson, "RTP Payload Format for G.719", RFC 5404, January 2009.

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

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

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

Authors' Addresses

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

Email: csp@csperkins.org

Magnus Westerlund
Ericsson
Farogatan 6
SE-164 80 Kista
Sweden

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

Joerg Ott
Aalto University
School of Electrical Engineering
Espoo 02150
Finland

Email: jorg.ott@aalto.fi

RTC-Web
Internet-Draft
Intended status: Standards Track
Expires: May 2, 2012

E. Rescorla
RTFM, Inc.

October 30, 2011

Security Considerations for RTC-Web
draft-ietf-rtcweb-security-01

Abstract

The Real-Time Communications on the Web (RTC-Web) working group is tasked with standardizing protocols for real-time communications between Web browsers. The major use cases for RTC-Web technology are real-time audio and/or video calls, Web conferencing, and direct data transfer. Unlike most conventional real-time systems (e.g., SIP-based soft phones) RTC-Web communications are directly controlled by some Web server, which poses new security challenges. For instance, a Web browser might expose a JavaScript API which allows a server to place a video call. Unrestricted access to such an API would allow any site which a user visited to "bug" a user's computer, capturing any activity which passed in front of their camera. This document defines the RTC-Web threat model and defines an architecture which provides security within that threat model.

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 May 2, 2012.

Copyright Notice

Table of Contents

1. Introduction

The Real-Time Communications on the Web (RTC-Web) working group is tasked with standardizing protocols for real-time communications between Web browsers. The major use cases for RTC-Web technology are real-time audio and/or video calls, Web conferencing, and direct data transfer. Unlike most conventional real-time systems, (e.g., SIP-based[RFC3261] soft phones) RTC-Web communications are directly controlled by some Web server. A simple case is shown below.

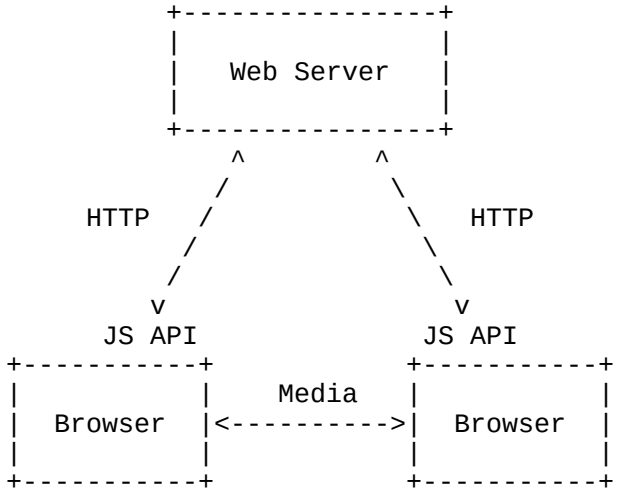


Figure 1: A simple RTC-Web system

In the system shown in Figure 1, Alice and Bob both have RTC-Web enabled browsers and they visit some Web server which operates a calling service. Each of their browsers exposes standardized JavaScript calling APIs which are used by the Web server to set up a call between Alice and Bob. While this system is topologically similar to a conventional SIP-based system (with the Web server acting as the signaling service and browsers acting as softphones), control has moved to the central Web server; the browser simply provides API points that are used by the calling service. As with any Web application, the Web server can move logic between the server and JavaScript in the browser, but regardless of where the code is executing, it is ultimately under control of the server.

It should be immediately apparent that this type of system poses new security challenges beyond those of a conventional VoIP system. In particular, it needs to contend with malicious calling services. For example, if the calling service can cause the browser to make a call at any time to any callee of its choice, then this facility can be

used to bug a user's computer without their knowledge, simply by placing a call to some recording service. More subtly, if the exposed APIs allow the server to instruct the browser to send arbitrary content, then they can be used to bypass firewalls or mount denial of service attacks. Any successful system will need to be resistant to this and other attacks.

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].

3. The Browser Threat Model

The security requirements for RTC-Web follow directly from the requirement that the browser's job is to protect the user. Huang et al. [huang-w2sp] summarize the core browser security guarantee as:

> Users can safely visit arbitrary web sites and execute scripts provided by those sites.

It is important to realize that this includes sites hosting arbitrary malicious scripts. The motivation for this requirement is simple: it is trivial for attackers to divert users to sites of their choice. For instance, an attacker can purchase display advertisements which direct the user (either automatically or via user clicking) to their site, at which point the browser will execute the attacker's scripts. Thus, it is important that it be safe to view arbitrarily malicious pages. Of course, browsers inevitably have bugs which cause them to fall short of this goal, but any new RTC-Web functionality must be designed with the intent to meet this standard. The remainder of this section provides more background on the existing Web security model.

In this model, then, the browser acts as a TRUSTED COMPUTING BASE (TCB) both from the user's perspective and to some extent from the server's. While HTML and JS provided by the server can cause the browser to execute a variety of actions, those scripts operate in a sandbox that isolates them both from the user's computer and from each other, as detailed below.

Conventionally, we refer to either WEB ATTACKERS, who are able to induce you to visit their sites but do not control the network, and NETWORK ATTACKERS, who are able to control your network. Network attackers correspond to the [RFC3552] "Internet Threat Model". In

general, it is desirable to build a system which is secure against both kinds of attackers, but realistically many sites do not run HTTPS [RFC2818] and so our ability to defend against network attackers is necessarily somewhat limited. Most of the rest of this section is devoted to web attackers, with the assumption that protection against network attackers is provided by running HTTPS.

3.1. Access to Local Resources

While the browser has access to local resources such as keying material, files, the camera and the microphone, it strictly limits or forbids web servers from accessing those same resources. For instance, while it is possible to produce an HTML form which will allow file upload, a script cannot do so without user consent and in fact cannot even suggest a specific file (e.g., /etc/passwd); the user must explicitly select the file and consent to its upload. [Note: in many cases browsers are explicitly designed to avoid dialogs with the semantics of "click here to screw yourself", as extensive research shows that users are prone to consent under such circumstances.]

Similarly, while Flash SWFs can access the camera and microphone, they explicitly require that the user consent to that access. In addition, some resources simply cannot be accessed from the browser at all. For instance, there is no real way to run specific executables directly from a script (though the user can of course be induced to download executable files and run them).

3.2. Same Origin Policy

Many other resources are accessible but isolated. For instance, while scripts are allowed to make HTTP requests via the XMLHttpRequest() API those requests are not allowed to be made to any server, but rather solely to the same ORIGIN from whence the script came.[I-D.abarth-origin] (although CORS [CORS] and WebSockets [I-D.ietf-hybi-thewebsocketprotocol] provides a escape hatch from this restriction, as described below.) This SAME ORIGIN POLICY (SOP) prevents server A from mounting attacks on server B via the user's browser, which protects both the user (e.g., from misuse of his credentials) and the server (e.g., from DoS attack).

More generally, SOP forces scripts from each site to run in their own, isolated, sandboxes. While there are techniques to allow them to interact, those interactions generally must be mutually consensual (by each site) and are limited to certain channels. For instance, multiple pages/browser panes from the same origin can read each other's JS variables, but pages from the different origins--or even iframes from different origins on the same page--cannot.

3.3. Bypassing SOP: CORS, WebSockets, and consent to communicate

While SOP serves an important security function, it also makes it inconvenient to write certain classes of applications. In particular, mash-ups, in which a script from origin A uses resources from origin B, can only be achieved via a certain amount of hackery. The W3C Cross-Origin Resource Sharing (CORS) spec [CORS] is a response to this demand. In CORS, when a script from origin A executes what would otherwise be a forbidden cross-origin request, the browser instead contacts the target server to determine whether it is willing to allow cross-origin requests from A. If it is so willing, the browser then allows the request. This consent verification process is designed to safely allow cross-origin requests.

While CORS is designed to allow cross-origin HTTP requests, WebSockets [I-D.ietf-hybi-thewebsocketprotocol] allows cross-origin establishment of transparent channels. Once a WebSockets connection has been established from a script to a site, the script can exchange any traffic it likes without being required to frame it as a series of HTTP request/response transactions. As with CORS, a WebSockets transaction starts with a consent verification stage to avoid allowing scripts to simply send arbitrary data to another origin.

While consent verification is conceptually simple--just do a handshake before you start exchanging the real data--experience has shown that designing a correct consent verification system is difficult. In particular, Huang et al. [huang-w2sp] have shown vulnerabilities in the existing Java and Flash consent verification techniques and in a simplified version of the WebSockets handshake. In particular, it is important to be wary of CROSS-PROTOCOL attacks in which the attacking script generates traffic which is acceptable to some non-Web protocol state machine. In order to resist this form of attack, WebSockets incorporates a masking technique intended to randomize the bits on the wire, thus making it more difficult to generate traffic which resembles a given protocol.

4. Security for RTC-Web Applications

4.1. Access to Local Devices

As discussed in Section 1, allowing arbitrary sites to initiate calls violates the core Web security guarantee; without some access restrictions on local devices, any malicious site could simply bug a user. At minimum, then, it MUST NOT be possible for arbitrary sites to initiate calls to arbitrary locations without user consent. This immediately raises the question, however, of what should be the scope

of user consent.

For the rest of this discussion we assume that the user is somehow going to grant consent to some entity (e.g., a social networking site) to initiate a call on his behalf. This consent may be limited to a single call or may be a general consent. In order for the user to make an intelligent decision about whether to allow a call (and hence his camera and microphone input to be routed somewhere), he must understand either who is requesting access, where the media is going, or both. So, for instance, one might imagine that at the time access to camera and microphone is requested, the user is shown a dialog that says "site X has requested access to camera and microphone, yes or no" (though note that this type of in-flow interface violates one of the guidelines in Section 3). The user's decision will of course be based on his opinion of Site X. However, as discussed below, this is a complicated concept.

4.1.1. Calling Scenarios and User Expectations

While a large number of possible calling scenarios are possible, the scenarios discussed in this section illustrate many of the difficulties of identifying the relevant scope of consent.

4.1.1.1. Dedicated Calling Services

The first scenario we consider is a dedicated calling service. In this case, the user has a relationship with a calling site and repeatedly makes calls on it. It is likely that rather than having to give permission for each call that the user will want to give the calling service long-term access to the camera and microphone. This is a natural fit for a long-term consent mechanism (e.g., installing an app store "application" to indicate permission for the calling service.) A variant of the dedicated calling service is a gaming site (e.g., a poker site) which hosts a dedicated calling service to allow players to call each other.

With any kind of service where the user may use the same service to talk to many different people, there is a question about whether the user can know who they are talking to. In general, this is difficult as most of the user interface is presented by the calling site. However, communications security mechanisms can be used to give some assurance, as described in Section 4.3.2.

4.1.1.2. Calling the Site You're On

Another simple scenario is calling the site you're actually visiting. The paradigmatic case here is the "click here to talk to a representative" windows that appear on many shopping sites. In this

case, the user's expectation is that they are calling the site they're actually visiting. However, it is unlikely that they want to provide a general consent to such a site; just because I want some information on a car doesn't mean that I want the car manufacturer to be able to activate my microphone whenever they please. Thus, this suggests the need for a second consent mechanism where I only grant consent for the duration of a given call. As described in Section 3.1, great care must be taken in the design of this interface to avoid the users just clicking through. Note also that the user interface chrome must clearly display elements showing that the call is continuing in order to avoid attacks where the calling site just leaves it up indefinitely but shows a Web UI that implies otherwise.

4.1.1.3. Calling to an Ad Target

In both of the previous cases, the user has a direct relationship (though perhaps a transient one) with the target of the call. Moreover, in both cases he is actually visiting the site of the person he is being asked to trust. However, this is not always so. Consider the case where a user is a visiting a content site which hosts an advertisement with an invitation to call for more information. When the user clicks the ad, they are connected with the advertiser or their agent.

The relationships here are far more complicated: the site the user is actually visiting has no direct relationship with the advertiser; they are just hosting ads from an ad network. The user has no relationship with the ad network, but desires one with the advertiser, at least for long enough to learn about their products. At minimum, then, whatever consent dialog is shown needs to allow the user to have some idea of the organization that they are actually calling.

However, because the user also has some relationship with the hosting site, it is also arguable that the hosting site should be allowed to express an opinion (e.g., to be able to allow or forbid a call) since a bad experience with an advertiser reflect negatively on the hosting site [this idea was suggested by Adam Barth]. However, this obviously presents a privacy challenge, as sites which host advertisements often learn very little about whether individual users clicked through to the ads, or even which ads were presented.

4.1.2. Origin-Based Security

As discussed in Section 3.2, the basic unit of Web sandboxing is the origin, and so it is natural to scope consent to origin. Specifically, a script from origin A MUST only be allowed to initiate communications (and hence to access camera and microphone) if the

user has specifically authorized access for that origin. It is of course technically possible to have coarser-scoped permissions, but because the Web model is scoped to origin, this creates a difficult mismatch.

Arguably, origin is not fine-grained enough. Consider the situation where Alice visits a site and authorizes it to make a single call. If consent is expressed solely in terms of origin, then at any future visit to that site (including one induced via mash-up or ad network), the site can bug Alice's computer, use the computer to place bogus calls, etc. While in principle Alice could grant and then revoke the privilege, in practice privileges accumulate; if we are concerned about this attack, something else is needed. There are a number of potential countermeasures to this sort of issue.

Individual Consent
: Ask the user for permission for each call.

Callee-oriented Consent
: Only allow calls to a given user.

Cryptographic Consent
: Only allow calls to a given set of peer keying material or to a cryptographically established identity.

Unfortunately, none of these approaches is satisfactory for all cases. As discussed above, individual consent puts the user's approval in the UI flow for every call. Not only does this quickly become annoying but it can train the user to simply click "OK", at which point the consent becomes useless. Thus, while it may be necessary to have individual consent in some case, this is not a suitable solution for (for instance) the calling service case. Where necessary, in-flow user interfaces must be carefully designed to avoid the risk of the user blindly clicking through.

The other two options are designed to restrict calls to a given target. Unfortunately, Callee-oriented consent does not work well because a malicious site can claim that the user is calling any user of his choice. One fix for this is to tie calls to a cryptographically established identity. While not suitable for all cases, this approach may be useful for some. If we consider the advertising case described in Section 4.1.1.3, it's not particularly convenient to require the advertiser to instantiate an iframe on the hosting site just to get permission; a more convenient approach is to cryptographically tie the advertiser's certificate to the communication directly. We're still tying permissions to origin here, but to the media origin (and-or destination) rather than to the Web origin.

Another case where media-level cryptographic identity makes sense is when a user really does not trust the calling site. For instance, I might be worried that the calling service will attempt to bug my computer, but I also want to be able to conveniently call my friends. If consent is tied to particular communications endpoints, then my risk is limited. However, this is also not that convenient an interface, since managing individual user permissions can be painful.

While this is primarily a question not for IETF, it should be clear that there is no really good answer. In general, if you cannot trust the site which you have authorized for calling not to bug you then your security situation is not really ideal. It is RECOMMENDED that browsers have explicit (and obvious) indicators that they are in a call in order to mitigate this risk.

4.1.3. Security Properties of the Calling Page

Origin-based security is intended to secure against web attackers. However, we must also consider the case of network attackers. Consider the case where I have granted permission to a calling service by an origin that has the HTTP scheme, e.g., http://calling-service.example.com. If I ever use my computer on an unsecured network (e.g., a hotspot or if my own home wireless network is insecure), and browse any HTTP site, then an attacker can bug my computer. The attack proceeds like this:

1. I connect to http://anything.example.org/. Note that this site is unaffiliated with the calling service.
2. The attacker modifies my HTTP connection to inject an IFRAME (or a redirect) to http://calling-service.example.com
3. The attacker forges the response apparently http://calling-service.example.com/ to inject JS to initiate a call to himself.

Note that this attack does not depend on the media being insecure. Because the call is to the attacker, it is also encrypted to him. Moreover, it need not be executed immediately; the attacker can "infect" the origin semi-permanently (e.g., with a web worker or a popunder) and thus be able to bug me long after I have left the infected network. This risk is created by allowing calls at all from a page fetched over HTTP.

Even if calls are only possible from HTTPS sites, if the site embeds active content (e.g., JavaScript) that is fetched over HTTP or from an untrusted site, because that JavaScript is executed in the security context of the page [finer-grained]. Thus, it is also dangerous to allow RTC-Web functionality from HTTPS origins that embed mixed content. Note: this issue is not restricted to PAGES

which contain mixed content. If a page from a given origin ever loads mixed content then it is possible for a network attacker to infect the browser's notion of that origin semi-permanently.

[[OPEN ISSUE: What recommendation should IETF make about (a) whether RTCWeb long-term consent should be available over HTTP pages and (b) How to handle origins where the consent is to an HTTPS URL but the page contains active mixed content?]]

4.2. Communications Consent Verification

As discussed in Section 3.3, allowing web applications unrestricted network access via the browser introduces the risk of using the browser as an attack platform against machines which would not otherwise be accessible to the malicious site, for instance because they are topologically restricted (e.g., behind a firewall or NAT). In order to prevent this form of attack as well as cross-protocol attacks it is important to require that the target of traffic explicitly consent to receiving the traffic in question. Until that consent has been verified for a given endpoint, traffic other than the consent handshake MUST NOT be sent to that endpoint.

4.2.1. ICE

Verifying receiver consent requires some sort of explicit handshake, but conveniently we already need one in order to do NAT hole-punching. ICE [RFC5245] includes a handshake designed to verify that the receiving element wishes to receive traffic from the sender. It is important to remember here that the site initiating ICE is presumed malicious; in order for the handshake to be secure the receiving element MUST demonstrate receipt/knowledge of some value not available to the site (thus preventing the site from forging responses). In order to achieve this objective with ICE, the STUN transaction IDs must be generated by the browser and MUST NOT be made available to the initiating script, even via a diagnostic interface. Verifying receiver consent also requires verifying the receiver wants to receive traffic from a particular sender, and at this time; for example a malicious site may simply attempt ICE to known servers that are using ICE for other sessions. ICE provides this verification as well, by using the STUN credentials as a form of per-session shared secret. Those credentials are known to the Web application, but would need to also be known and used by the STUN-receiving element to be useful.

There also needs to be some mechanism for the browser to verify that the target of the traffic continues to wish to receive it. Obviously, some ICE-based mechanism will work here, but it has been observed that because ICE keepalives are indications, they will not

work here, so some other mechanism is needed.

4.2.2. Masking

Once consent is verified, there still is some concern about misinterpretation attacks as described by Huang et al.[huang-w2sp]. As long as communication is limited to UDP, then this risk is probably limited, thus masking is not required for UDP. I.e., once communications consent has been verified, it is most likely safe to allow the implementation to send arbitrary UDP traffic to the chosen destination, provided that the STUN keepalives continue to succeed. In particular, this is true for the data channel if DTLS is used because DTLS (with the anti-chosen plaintext mechanisms required by TLS 1.1) does not allow the attacker to generate predictable ciphertext. However, with TCP the risk of transparent proxies becomes much more severe. If TCP is to be used, then WebSockets style masking MUST be employed.

4.2.3. Backward Compatibility

A requirement to use ICE limits compatibility with legacy non-ICE clients. It seems unsafe to completely remove the requirement for some check. All proposed checks have the common feature that the browser sends some message to the candidate traffic recipient and refuses to send other traffic until that message has been replied to. The message/reply pair must be generated in such a way that an attacker who controls the Web application cannot forge them, generally by having the message contain some secret value that must be incorporated (e.g., echoed, hashed into, etc.). Non-ICE candidates for this role (in cases where the legacy endpoint has a public address) include:

- STUN checks without using ICE (i.e., the non-RTC-web endpoint sets up a STUN responder.)
- Use or RTCP as an implicit reachability check.

In the RTCP approach, the RTC-Web endpoint is allowed to send a limited number of RTP packets prior to receiving consent. This allows a short window of attack. In addition, some legacy endpoints do not support RTCP, so this is a much more expensive solution for such endpoints, for which it would likely be easier to implement ICE. For these two reasons, an RTCP-based approach does not seem to address the security issue satisfactorily.

In the STUN approach, the RTC-Web endpoint is able to verify that the recipient is running some kind of STUN endpoint but unless the STUN responder is integrated with the ICE username/password establishment system, the RTC-Web endpoint cannot verify that the recipient

consents to this particular call. This may be an issue if existing STUN servers are operated at addresses that are not able to handle bandwidth-based attacks. Thus, this approach does not seem satisfactory either.

If the systems are tightly integrated (i.e., the STUN endpoint responds with responses authenticated with ICE credentials) then this issue does not exist. However, such a design is very close to an ICE-Lite implementation (indeed, arguably is one). An intermediate approach would be to have a STUN extension that indicated that one was responding to RTC-Web checks but not computing integrity checks based on the ICE credentials. This would allow the use of standalone STUN servers without the risk of confusing them with legacy STUN servers. If a non-ICE legacy solution is needed, then this is probably the best choice.

Once initial consent is verified, we also need to verify continuing consent, in order to avoid attacks where two people briefly share an IP (e.g., behind a NAT in an Internet cafe) and the attacker arranges for a large, unstoppable, traffic flow to the network and then leaves. The appropriate technologies here are fairly similar to those for initial consent, though are perhaps weaker since the threats is less severe.

[[OPEN ISSUE: Exactly what should be the requirements here? Proposals include ICE all the time or ICE but with allowing one of these non-ICE things for legacy.]]

4.2.4. IP Location Privacy

Note that as soon as the callee sends their ICE candidates, the callee learns the callee's IP addresses. The callee's server reflexive address reveals a lot of information about the callee's location. In order to avoid tracking, implementations may wish to suppress the start of ICE negotiation until the callee has answered. In addition, either side may wish to hide their location entirely by forcing all traffic through a TURN server.

4.3. Communications Security

Finally, we consider a problem familiar from the SIP world: communications security. For obvious reasons, it MUST be possible for the communicating parties to establish a channel which is secure against both message recovery and message modification. (See [RFC5479] for more details.) This service must be provided for both data and voice/video. Ideally the same security mechanisms would be used for both types of content. Technology for providing this service (for instance, DTLS [RFC4347] and DTLS-SRTP [RFC5763]) is

well understood. However, we must examine this technology to the RTC-Web context, where the threat model is somewhat different.

In general, it is important to understand that unlike a conventional SIP proxy, the calling service (i.e., the Web server) controls not only the channel between the communicating endpoints but also the application running on the user's browser. While in principle it is possible for the browser to cut the calling service out of the loop and directly present trusted information (and perhaps get consent), practice in modern browsers is to avoid this whenever possible. "In-flow" modal dialogs which require the user to consent to specific actions are particularly disfavored as human factors research indicates that unless they are made extremely invasive, users simply agree to them without actually consciously giving consent. [abarth-rtcweb]. Thus, nearly all the UI will necessarily be rendered by the browser but under control of the calling service. This likely includes the peer's identity information, which, after all, is only meaningful in the context of some calling service.

This limitation does not mean that preventing attack by the calling service is completely hopeless. However, we need to distinguish between two classes of attack:

Retrospective compromise of calling service.
: The calling service is is non-malicious during a call but subsequently is compromised and wishes to attack an older call.

During-call attack by calling service.
: The calling service is compromised during the call it wishes to attack.

Providing security against the former type of attack is practical using the techniques discussed in Section 4.3.1. However, it is extremely difficult to prevent a trusted but malicious calling service from actively attacking a user's calls, either by mounting a MITM attack or by diverting them entirely. (Note that this attack applies equally to a network attacker if communications to the calling service are not secured.) We discuss some potential approaches and why they are likely to be impractical in Section 4.3.2.

4.3.1. Protecting Against Retrospective Compromise

In a retrospective attack, the calling service was uncompromised during the call, but that an attacker subsequently wants to recover the content of the call. We assume that the attacker has access to the protected media stream as well as having full control of the calling service.

If the calling service has access to the traffic keying material (as in SDES [RFC4568]), then retrospective attack is trivial. This form of attack is particularly serious in the Web context because it is standard practice in Web services to run extensive logging and monitoring. Thus, it is highly likely that if the traffic key is part of any HTTP request it will be logged somewhere and thus subject to subsequent compromise. It is this consideration that makes an automatic, public key-based key exchange mechanism imperative for RTC-Web (this is a good idea for any communications security system) and this mechanism SHOULD provide perfect forward secrecy (PFS). The signaling channel/calling service can be used to authenticate this mechanism.

In addition, the system MUST NOT provide any APIs to extract either long-term keying material or to directly access any stored traffic keys. Otherwise, an attacker who subsequently compromised the calling service might be able to use those APIs to recover the traffic keys and thus compromise the traffic.

4.3.2. Protecting Against During-Call Attack

Protecting against attacks during a call is a more difficult proposition. Even if the calling service cannot directly access keying material (as recommended in the previous section), it can simply mount a man-in-the-middle attack on the connection, telling Alice that she is calling Bob and Bob that he is calling Alice, while in fact the calling service is acting as a calling bridge and capturing all the traffic. While in theory it is possible to construct techniques which protect against this form of attack, in practice these techniques all require far too much user intervention to be practical, given the user interface constraints described in [abarth-rtcweb].

4.3.2.1. Key Continuity

One natural approach is to use "key continuity". While a malicious calling service can present any identity it chooses to the user, it cannot produce a private key that maps to a given public key. Thus, it is possible for the browser to note a given user's public key and generate an alarm whenever that user's key changes. SSH [RFC4251] uses a similar technique. (Note that the need to avoid explicit user consent on every call precludes the browser requiring an immediate manual check of the peer's key).

Unfortunately, this sort of key continuity mechanism is far less useful in the RTC-Web context. First, much of the virtue of RTC-Web (and any Web application) is that it is not bound to particular piece of client software. Thus, it will be not only possible but routine

for a user to use multiple browsers on different computers which will of course have different keying material (SACRED [RFC3760] notwithstanding.) Thus, users will frequently be alerted to key mismatches which are in fact completely legitimate, with the result that they are trained to simply click through them. As it is known that users routinely will click through far more dire warnings [cranor-wolf], it seems extremely unlikely that any key continuity mechanism will be effective rather than simply annoying.

Moreover, it is trivial to bypass even this kind of mechanism. Recall that unlike the case of SSH, the browser never directly gets the peer's identity from the user. Rather, it is provided by the calling service. Even enabling a mechanism of this type would require an API to allow the calling service to tell the browser "this is a call to user X". All the calling service needs to do to avoid triggering a key continuity warning is to tell the browser that "this is a call to user Y" where Y is close to X. Even if the user actually checks the other side's name (which all available evidence indicates is unlikely), this would require (a) the browser to trusted UI to provide the name and (b) the user to not be fooled by similar appearing names.

4.3.2.2. Short Authentication Strings

ZRTP [RFC6189] uses a "short authentication string" (SAS) which is derived from the key agreement protocol. This SAS is designed to be read over the voice channel and if confirmed by both sides precludes MITM attack. The intention is that the SAS is used once and then key continuity (though a different mechanism from that discussed above) is used thereafter.

Unfortunately, the SAS does not offer a practical solution to the problem of a compromised calling service. "Voice conversion" systems, which modify voice from one speaker to make it sound like another, are an active area of research. These systems are already good enough to fool both automatic recognition systems [farus-conversion] and humans [kain-conversion] in many cases, and are of course likely to improve in future, especially in an environment where the user just wants to get on with the phone call. Thus, even if SAS is effective today, it is likely not to be so for much longer. Moreover, it is possible for an attacker who controls the browser to allow the SAS to succeed and then simulate call failure and reconnect, trusting that the user will not notice that the "no SAS" indicator has been set (which seems likely).

Even were SAS secure if used, it seems exceedingly unlikely that users will actually use it. As discussed above, the browser UI constraints preclude requiring the SAS exchange prior to completing

the call and so it must be voluntary; at most the browser will provide some UI indicator that the SAS has not yet been checked. However, it it is well-known that when faced with optional mechanisms such as fingerprints, users simply do not check them [whitten-johnny] Thus, it is highly unlikely that users will ever perform the SAS exchange.

Once uses have checked the SAS once, key continuity is required to avoid them needing to check it on every call. However, this is problematic for reasons indicated in Section 4.3.2.1. In principle it is of course possible to render a different UI element to indicate that calls are using an unauthenticated set of keying material (recall that the attacker can just present a slightly different name so that the attack shows the same UI as a call to a new device or to someone you haven't called before) but as a practical matter, users simply ignore such indicators even in the rather more dire case of mixed content warnings.

4.3.2.3. Recommendations

[[OPEN ISSUE: What are the best UI recommendations to make? Proposal: take the text from [I-D.kaufman-rtcweb-security-ui] Section 2]]

[[OPEN ISSUE: Exactly what combination of media security primitives should be specified and/or mandatory to implement? In particular, should we allow DTLS-SRTP only, or both DTLS-SRTP and SDES. Should we allow RTP for backward compatibility?]]

5. Security Considerations

This entire document is about security.

6. Acknowledgements

Bernard Aboba, Harald Alvestrand, Cullen Jennings, Hadriel Kaplan (S 4.2.1), Matthew Kaufman, Magnus Westerland.

7. References

7.1. Normative References

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

7.2. Informative References

[CORS] van Kesteren, A., "Cross-Origin Resource Sharing".

[I-D.abarth-origin]
Barth, A., "The Web Origin Concept", draft-abarth-origin-09 (work in progress), November 2010.

[I-D.ietf-hybi-thewebsocketprotocol]
Fette, I. and A. Melnikov, "The WebSocket protocol", draft-ietf-hybi-thewebsocketprotocol-17 (work in progress), September 2011.

[I-D.kaufman-rtcweb-security-ui]
Kaufman, M., "Client Security User Interface Requirements for RTCWEB", draft-kaufman-rtcweb-security-ui-00 (work in progress), June 2011.

[RFC2818] Rescorla, E., "HTTP Over TLS", RFC 2818, May 2000.

[RFC3261] Rosenberg, J., Schulzrinne, H., Camarillo, G., Johnston, A., Peterson, J., Sparks, R., Handley, M., and E. Schooler, "SIP: Session Initiation Protocol", RFC 3261, June 2002.

[RFC3552] Rescorla, E. and B. Korver, "Guidelines for Writing RFC Text on Security Considerations", BCP 72, RFC 3552, July 2003.

[RFC3760] Gustafson, D., Just, M., and M. Nystrom, "Securely Available Credentials (SACRED) - Credential Server Framework", RFC 3760, April 2004.

[RFC4251] Ylonen, T. and C. Lonvick, "The Secure Shell (SSH) Protocol Architecture", RFC 4251, January 2006.

[RFC4347] Rescorla, E. and N. Modadugu, "Datagram Transport Layer Security", RFC 4347, April 2006.

[RFC4568] Andreasen, F., Baugher, M., and D. Wing, "Session Description Protocol (SDP) Security Descriptions for Media Streams", RFC 4568, July 2006.

[RFC5245] Rosenberg, J., "Interactive Connectivity Establishment (ICE): A Protocol for Network Address Translator (NAT) Traversal for Offer/Answer Protocols", RFC 5245, April 2010.

[RFC5479] Wing, D., Fries, S., Tschofenig, H., and F. Audet, "Requirements and Analysis of Media Security Management Protocols", RFC 5479, April 2009.

[RFC5763] Fischl, J., Tschofenig, H., and E. Rescorla, "Framework for Establishing a Secure Real-time Transport Protocol (SRTP) Security Context Using Datagram Transport Layer Security (DTLS)", RFC 5763, May 2010.

[RFC6189] Zimmermann, P., Johnston, A., and J. Callas, "ZRTP: Media Path Key Agreement for Unicast Secure RTP", RFC 6189, April 2011.

[abarth-rtcweb]
Barth, A., "Prompting the user is security failure", RTC-Web Workshop.

[cranor-wolf]
Sunshine, J., Egelman, S., Almuhimedi, H., Atri, N., and L. cranor, "Crying Wolf: An Empirical Study of SSL Warning Effectiveness", Proceedings of the 18th USENIX Security Symposium, 2009.

[farus-conversion]
Farrus, M., Erro, D., and J. Hernando, "Speaker Recognition Robustness to Voice Conversion".

[finer-grained]
Barth, A. and C. Jackson, "Beware of Finer-Grained Origins", W2SP, 2008.

[huang-w2sp]
Huang, L-S., Chen, E., Barth, A., Rescorla, E., and C. Jackson, "Talking to Yourself for Fun and Profit", W2SP, 2011.

[kain-conversion]
Kain, A. and M. Macon, "Design and Evaluation of a Voice Conversion Algorithm based on Spectral Envelope Mapping and Residual Prediction", Proceedings of ICASSP, May 2001.

[whitten-johnny]
Whitten, A. and J. Tygar, "Why Johnny Can’t Encrypt: A Usability Evaluation of PGP 5.0", Proceedings of the 8th USENIX Security Symposium, 1999.

Appendix A. A Proposed Security Architecture [No Consensus on This]

This section contains a proposed security architecture, based on the considerations discussed in the main body of this memo. This section is currently the opinion of the author and does not have consensus though some (many?) elements of this proposal do seem to have general consensus.

A.1. Trust Hierarchy

The basic assumption of this proposal is that network resources exist in a hierarchy of trust, rooted in the browser, which serves as the user's TRUSTED COMPUTING BASE (TCB). Any security property which the user wishes to have enforced must be ultimately guaranteed by the browser (or transitively by some property the browser verifies). Conversely, if the browser is compromised, then no security guarantees are possible. Note that there are cases (e.g., Internet kiosks) where the user can't really trust the browser that much. In these cases, the level of security provided is limited by how much they trust the browser.

Optimally, we would not rely on trust in any entities other than the browser. However, this is unfortunately not possible if we wish to have a functional system. Other network elements fall into two categories: those which can be authenticated by the browser and thus are partly trusted--though to the minimum extent necessary--and those which cannot be authenticated and thus are untrusted. This is a natural extension of the end-to-end principle.

A.1.1. Authenticated Entities

There are two major classes of authenticated entities in the system:

- Calling services: Web sites whose origin we can verify (optimally via HTTPS).
- Other users: RTC-Web peers whose origin we can verify cryptographically (optimally via DTLS-SRTP).

Note that merely being authenticated does not make these entities trusted. For instance, just because we can verify that https://www.evil.org/ is owned by Dr. Evil does not mean that we can trust Dr. Evil to access our camera an microphone. However, it gives the user an opportunity to determine whether he wishes to trust Dr. Evil or not; after all, if he desires to contact Dr. Evil, it's safe to temporarily give him access to the camera and microphone for the purpose of the call. The point here is that we must first identify other elements before we can determine whether to trust them.

It’s also worth noting that there are settings where authentication is non-cryptographic, such as other machines behind a firewall. Naturally, the level of trust one can have in identities verified in this way depends on how strong the topology enforcement is.

A.1.2. Unauthenticated Entities

Other than the above entities, we are not generally able to identify other network elements, thus we cannot trust them. This does not mean that it is not possible to have any interaction with them, but it means that we must assume that they will behave maliciously and design a system which is secure even if they do so.

A.2. Overview

This section describes a typical RTCWeb session and shows how the various security elements interact and what guarantees are provided to the user. The example in this section is a "best case" scenario in which we provide the maximal amount of user authentication and media privacy with the minimal level of trust in the calling service. Simpler versions with lower levels of security are also possible and are noted in the text where applicable. It’s also important to recognize the tension between security (or performance) and privacy. The example shown here is aimed towards settings where we are more concerned about secure calling than about privacy, but as we shall see, there are settings where one might wish to make different tradeoffs--this architecture is still compatible with those settings.

For the purposes of this example, we assume the topology shown in the figure below. This topology is derived from the topology shown in Figure 1, but separates Alice and Bob’s identities from the process of signaling. Specifically, Alice and Bob have relationships with some Identity Provider (IDP) that supports a protocol such OpenID or BrowserID) that can be used to attest to their identity. This separation isn’t particularly important in "closed world" cases where Alice and Bob are users on the same social network and have identities based on that network. However, there are important settings where that is not the case, such as federation (calls from one network to another) and calling on untrusted sites, such as where two users who have a relationship via a given social network want to call each other on another, untrusted, site, such as a poker site.

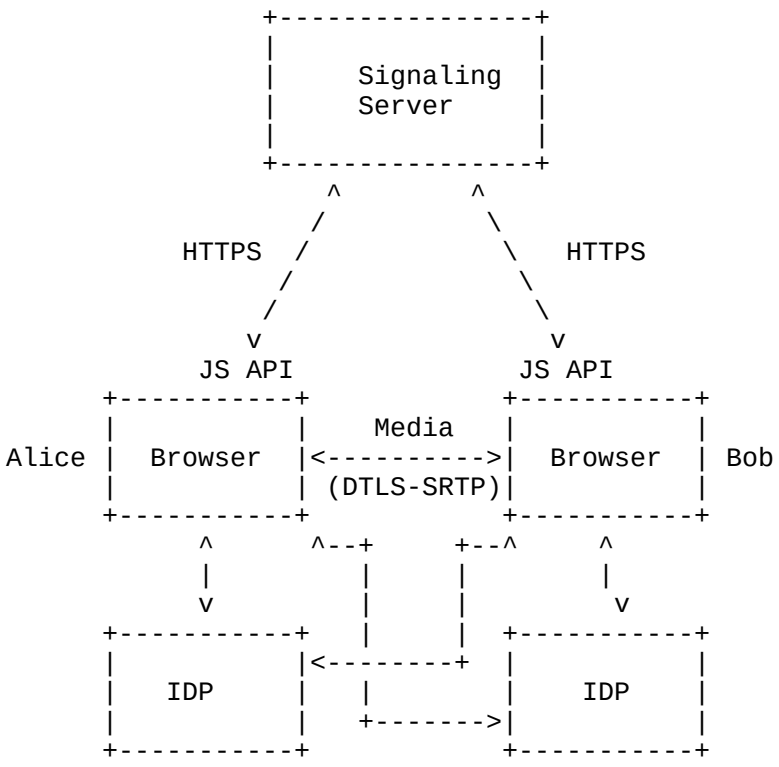


Figure 2: A call with IDP-based identity

A.2.1. Initial Signaling

Alice and Bob are both users of a common calling service; they both have approved the calling service to make calls (we defer the discussion of device access permissions till later). They are both connected to the calling service via HTTPS and so know the origin with some level of confidence. They also have accounts with some identity provider. This sort of identity service is becoming increasingly common in the Web environment in technologies such (BrowserID, Federated Google Login, Facebook Connect, OAuth, OpenID, WebFinger), and is often provided as a side effect service of your ordinary accounts with some service. In this example, we show Alice and Bob using a separate identity service, though they may actually be using the same identity service as calling service or have no identity service at all.

Alice is logged onto the calling service and decides to call Bob. She can see from the calling service that he is online and the calling service presents a JS UI in the form of a button next to Bob’s name

which says "Call". Alice clicks the button, which initiates a JS callback that instantiates a PeerConnection object. This does not require a security check: JS from any origin is allowed to get this far.

Once the PeerConnection is created, the calling service JS needs to set up some media. Because this is an audio/video call, it creates two MediaStreams, one connected to an audio input and one connected to a video input. At this point the first security check is required: untrusted origins are not allowed to access the camera and microphone. In this case, because Alice is a long-term user of the calling service, she has made a permissions grant (i.e., a setting in the browser) to allow the calling service to access her camera and microphone any time it wants. The browser checks this setting when the camera and microphone requests are made and thus allows them.

In the current W3C API, once some streams have been added, Alice's browser + JS generates a signaling message The format of this data is currently undefined. It may be a complete message as defined by ROAP [REF] or may be assembled piecemeal by the JS. In either case, it will contain:

- Media channel information
- ICE candidates
- A fingerprint attribute binding the message to Alice's public key [RFC5763]

Prior to sending out the signaling message, the PeerConnection code contacts the identity service and obtains an assertion binding Alice's identity to her fingerprint. The exact details depend on the identity service (though as discussed in Appendix A.3.6.4 I believe PeerConnection can be agnostic to them), but for now it's easiest to think of as a BrowserID assertion.

This message is sent to the signaling server, e.g., by XMLHttpRequest [REF] or by WebSockets [I-D.ietf-hybi-thewebsocketprotocol]. The signaling server processes the message from Alice's browser, determines that this is a call to Bob and sends a signaling message to Bob's browser (again, the format is currently undefined). The JS on Bob's browser processes it, and alerts Bob to the incoming call and to Alice's identity. In this case, Alice has provided an identity assertion and so Bob's browser contacts Alice's identity provider (again, this is done in a generic way so the browser has no specific knowledge of the IDP) to verity the assertion. This allows the browser to display a trusted element indicating that a call is coming in from Alice. If Alice is in Bob's address book, then this interface might also include her real name, a picture, etc. The calling site will also provide some user interface element (e.g., a

button) to allow Bob to answer the call, though this is most likely not part of the trusted UI.

If Bob agrees [I am ignoring early media for now], a PeerConnection is instantiated with the message from Alice's side. Then, a similar process occurs as on Alice's browser: Bob's browser verifies that the calling service is approved, the media streams are created, and a return signaling message containing media information, ICE candidates, and a fingerprint is sent back to Alice via the signaling service. If Bob has a relationship with an IDP, the message will also come with an identity assertion.

At this point, Alice and Bob each know that the other party wants to have a secure call with them. Based purely on the interface provided by the signaling server, they know that the signaling server claims that the call is from Alice to Bob. Because the far end sent an identity assertion along with their message, they know that this is verifiable from the IDP as well. Of course, the call works perfectly well if either Alice or Bob doesn't have a relationship with an IDP; they just get a lower level of assurance. Moreover, Alice might wish to make an anonymous call through an anonymous calling site, in which case she would of course just not provide any identity assertion and the calling site would mask her identity from Bob.

A.2.2. Media Consent Verification

As described in Section 4.2. This proposal specifies that that be performed via ICE. Thus, Alice and Bob perform ICE checks with each other. At the completion of these checks, they are ready to send non-ICE data.

At this point, Alice knows that (a) Bob (assuming he is verified via his IDP) or someone else who the signaling service is claiming is Bob is willing to exchange traffic with her and (b) that either Bob is at the IP address which she has verified via ICE or there is an attacker who is on-path to that IP address detouring the traffic. Note that it is not possible for an attacker who is on-path but not attached to the signaling service to spoof these checks because they do not have the ICE credentials. Bob's security guarantees with respect to Alice are the converse of this.

A.2.3. DTLS Handshake

Once the ICE checks have completed [more specifically, once some ICE checks have completed], Alice and Bob can set up a secure channel. This is performed via DTLS [RFC4347] (for the data channel) and DTLS-SRTP [RFC5763] for the media channel. Specifically, Alice and Bob perform a DTLS handshake on every channel which has been established

by ICE. The total number of channels depends on the amount of muxing; in the most likely case we are using both RTP/RTCP mux and muxing multiple media streams on the same channel, in which case there is only one DTLS handshake. Once the DTLS handshake has completed, the keys are extracted and used to key SRTP for the media channels.

At this point, Alice and Bob know that they share a set of secure data and/or media channels with keys which are not known to any third-party attacker. If Alice and Bob authenticated via their IDPs, then they also know that the signaling service is not attacking them. Even if they do not use an IDP, as long as they have minimal trust in the signaling service not to perform a man-in-the-middle attack, they know that their communications are secure against the signaling service as well.

A.2.4. Communications and Consent Freshness

From a security perspective, everything from here on in is a little anticlimactic: Alice and Bob exchange data protected by the keys negotiated by DTLS. Because of the security guarantees discussed in the previous sections, they know that the communications are encrypted and authenticated.

The one remaining security property we need to establish is "consent freshness", i.e., allowing Alice to verify that Bob is still prepared to receive her communications. ICE specifies periodic STUN keepalizes but only if media is not flowing. Because the consent issue is more difficult here, we require RTCWeb implementations to periodically send keepalives. If a keepalive fails and no new ICE channels can be established, then the session is terminated.

A.3. Detailed Technical Description

A.3.1. Origin and Web Security Issues

The basic unit of permissions for RTC-Web is the origin [I-D.abarth-origin]. Because the security of the origin depends on being able to authenticate content from that origin, the origin can only be securely established if data is transferred over HTTPS. Thus, clients MUST treat HTTP and HTTPS origins as different permissions domains and SHOULD NOT permit access to any RTC-Web functionality from scripts fetched over non-secure (HTTP) origins. If an HTTPS origin contains mixed active content (regardless of whether it is present on the specific page attempting to access RTC-Web functionality), any access MUST be treated as if it came from the HTTP origin. For instance, if a https://www.example.com/example.html loads https://www.example.com/example.js and

http://www.example.org/jquery.js, any attempt by example.js to access RTCWeb functionality MUST be treated as if it came from http://www.example.com/. Note that many browsers already track mixed content and either forbid it by default or display a warning.

A.3.2. Device Permissions Model

Implementations MUST obtain explicit user consent prior to providing access to the camera and/or microphone. Implementations MUST at minimum support the following two permissions models:

- Requests for one-time camera/microphone access.
- Requests for permanent access.

In addition, they SHOULD support requests for access to a single communicating peer. E.g., "Call customerservice@ford.com". Browsers servicing such requests SHOULD clearly indicate that identity to the user when asking for permission.

API Requirement: The API MUST provide a mechanism for the requesting JS to indicate which of these forms of permissions it is requesting. This allows the client to know what sort of user interface experience to provide. In particular, browsers might display a non-invasive door hanger ("some features of this site may not work..." when asking for long-term permissions) but a more invasive UI ("here is your own video") for single-call permissions. The API MAY grant weaker permissions than the JS asked for if the user chooses to authorize only those permissions, but if it intends to grant stronger ones SHOULD display the appropriate UI for those permissions.

API Requirement: The API MUST provide a mechanism for the requesting JS to relinquish the ability to see or modify the media (e.g., via MediaStream.record()). Combined with secure authentication of the communicating peer, this allows a user to be sure that the calling site is not accessing or modifying their conversion.

UI Requirement: The UI MUST clearly indicate when the user’s camera and microphone are in use. This indication MUST NOT be suppressable by the JS and MUST clearly indicate how to terminate a call, and provide a UI means to immediately stop camera/ microphone input without the JS being able to prevent it.

UI Requirement: If the UI indication of camera/microphone use are displayed in the browser such that minimizing the browser window would hide the indication, or the JS creating an overlapping window would hide the indication, then the browser SHOULD stop camera and microphone input.

Clients MAY permit the formation of data channels without any direct user approval. Because sites can always tunnel data through the server, further restrictions on the data channel do not provide any additional security. (though see Appendix A.3.3 for a related issue).

Implementations which support some form of direct user authentication SHOULD also provide a policy by which a user can authorize calls only to specific counterparties. Specifically, the implementation SHOULD provide the following interfaces/controls:

- Allow future calls to this verified user.
- Allow future calls to any verified user who is in my system address book (this only works with address book integration, of course).

Implementations SHOULD also provide a different user interface indication when calls are in progress to users whose identities are directly verifiable. Appendix A.3.5 provides more on this.

A.3.3. Communications Consent

Browser client implementations of RTC-Web MUST implement ICE. Server gateway implementations which operate only at public IP addresses may implement ICE-Lite.

Browser implementations MUST verify reachability via ICE prior to sending any non-ICE packets to a given destination. Implementations MUST NOT provide the ICE transaction ID to JavaScript. [Note: this document takes no position on the split between ICE in JS and ICE in the browser. The above text is written the way it is for editorial convenience and will be modified appropriately if the WG decides on ICE in the JS.]

Implementations MUST send keepalives no less frequently than every 30 seconds regardless of whether traffic is flowing or not. If a keepalive fails then the implementation MUST either attempt to find a new valid path via ICE or terminate media for that ICE component. Note that ICE [RFC5245]; Section 10 keepalives use STUN Binding Indications which are one-way and therefore not sufficient. We will need to define a new mechanism for this. [OPEN ISSUE: what to do here.]

A.3.4. IP Location Privacy

As mentioned in Section 4.2.4 above, a side effect of the default ICE behavior is that the peer learns one's IP address, which leaks large amounts of location information, especially for mobile devices. This has negative privacy consequences in some circumstances. The

following two API requirements are intended to mitigate this issue:

API Requirement: The API MUST provide a mechanism to suppress ICE negotiation (though perhaps to allow candidate gathering) until the user has decided to answer the call [note: determining when the call has been answered is a question for the JS.] This enables a user to prevent a peer from learning their IP address if they elect not to answer a call.

API Requirement: The API MUST provide a mechanism for the calling application to indicate that only TURN candidates are to be used. This prevents the peer from learning one's IP address at all.

A.3.5. Communications Security

Implementations MUST implement DTLS and DTLS-SRTP. All data channels MUST be secured via DTLS. DTLS-SRTP MUST be offered for every media channel and MUST be the default; i.e., if an implementation receives an offer for DTLS-SRTP and SDES and/or plain RTP, DTLS-SRTP MUST be selected.

[OPEN ISSUE: What should the settings be here? MUST?] Implementations MAY support SDES and RTP for media traffic for backward compatibility purposes.

API Requirement: The API MUST provide a mechanism to indicate that a fresh DTLS key pair is to be generated for a specific call. This is intended to allow for unlinkability. Note that there are also settings where it is attractive to use the same keying material repeatedly, especially those with key continuity-based authentication.

API Requirement: The API MUST provide a mechanism to indicate that a fresh DTLS key pair is to be generated for a specific call. This is intended to allow for unlinkability.

API Requirement: When DTLS-SRTP is used, the API MUST NOT permit the JS to obtain the negotiated keying material. This requirement preserves the end-to-end security of the media.

UI Requirements: A user-oriented client MUST provide an "inspector" interface which allows the user to determine the security characteristics of the media. [largely derived from [I-D.kaufman-rtcweb-security-ui]

The following properties SHOULD be displayed "up-front" in the browser chrome, i.e., without requiring the user to ask for them:

* A client MUST provide a user interface through which a user may determine the security characteristics for currently-displayed audio and video stream(s)
* A client MUST provide a user interface through which a user may determine the security characteristics for transmissions of their microphone audio and camera video.
* The "security characteristics" MUST include an indication as to whether or not the transmission is cryptographically protected and whether that protection is based on a key that was delivered out-of-band (from a server) or was generated as a result of a pairwise negotiation.
* If the far endpoint was directly verified Appendix A.3.6 the "security characteristics" MUST include the verified information.

The following properties are more likely to require some "drill-down" from the user:

* If the transmission is cryptographically protected, the The algorithms in use (For example: "AES-CBC" or "Null Cipher".)
* If the transmission is cryptographically protected, the "security characteristics" MUST indicate whether PFS is provided.
* If the transmission is cryptographically protected via an end-to-end mechanism the "security characteristics" MUST include some mechanism to allow an out-of-band verification of the peer, such as a certificate fingerprint or an SAS.

A.3.6. Web-Based Peer Authentication

A.3.6.1. Generic Concepts

In a number of cases, it is desirable for the endpoint (i.e., the browser) to be able to directly identity the endpoint on the other side without trusting only the signaling service to which they are connected. For instance, users may be making a call via a federated system where they wish to get direct authentication of the other side. Alternately, they may be making a call on a site which they minimally trust (such as a poker site) but to someone who has an identity on a site they do trust (such as a social network.)

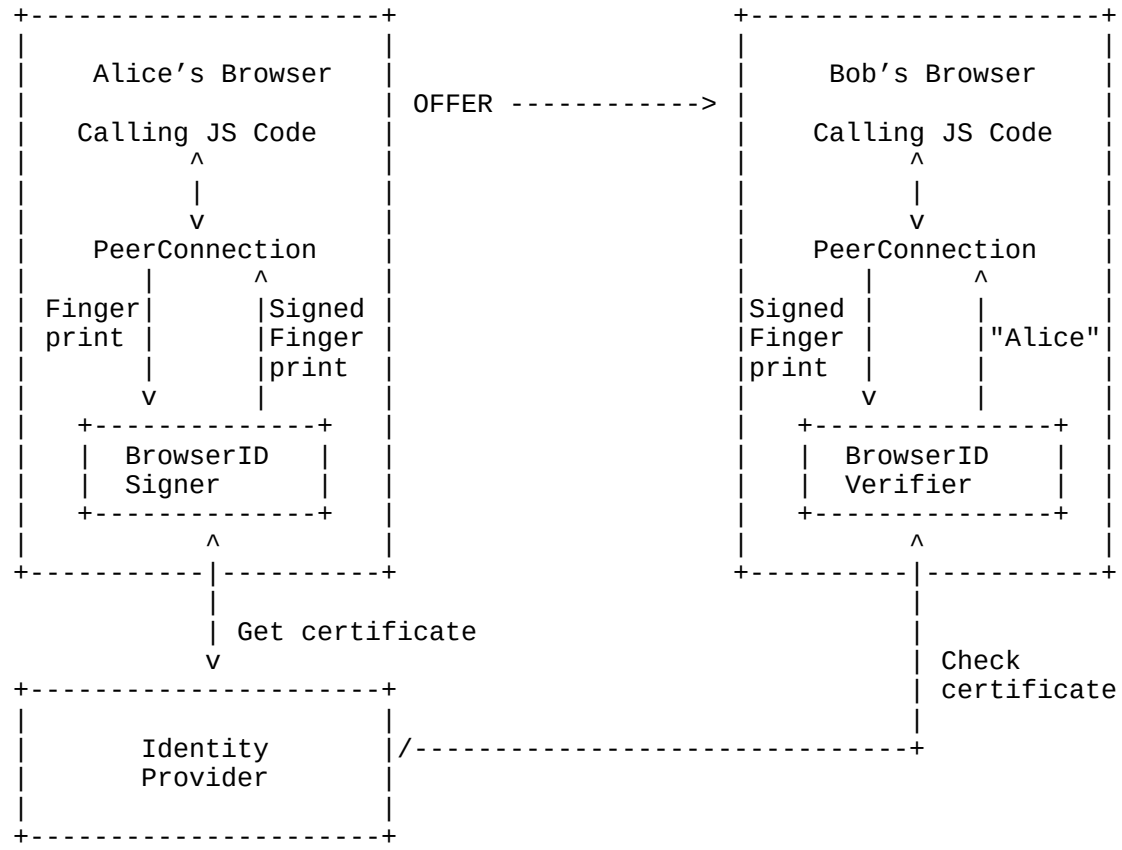
Recently, a number of Web-based identity technologies (OAuth, BrowserID, Facebook Connect), etc. have been developed. While the details vary, what these technologies share is that they have a Web-based (i.e., HTTP/HTTPS identity provider) which attests to your identity. For instance, if I have an account at example.org, I could

use the example.org identity provider to prove to others that I was alice@example.org. The development of these technologies allows us to separate calling from identity provision: I could call you on Poker Galaxy but identify myself as alice@example.org.

Whatever the underlying technology, the general principle is that the party which is being authenticated is NOT the signaling site but rather the user (and their browser). Similarly, the relying party is the browser and not the signaling site. This means that the PeerConnection API MUST arrange to talk directly to the identity provider in a way that cannot be impersonated by the calling site. The following sections provide two examples of this.

A.3.6.2. BrowserID

BrowserID [https://browserid.org/] is a technology which allows a user with a verified email address to generate an assertion (authenticated by their identity provider) attesting to their identity (phrased as an email address). The way that this is used in practice is that the relying party embeds JS in their site which talks to the BrowserID code (either hosted on a trusted intermediary or embedded in the browser). That code generates the assertion which is passed back to the relying party for verification. The assertion can be verified directly or with a Web service provided by the identity provider. It’s relatively easy to extend this functionality to authenticate RTC-Web calls, as shown below.



The way this mechanism works is as follows. On Alice's side, Alice goes to initiate a call.

1. The calling JS instantiates a PeerConnection and tells it that it is interested in having it authenticated via BrowserID.
2. The PeerConnection instantiates the BrowserID signer in an invisible IFRAME. The IFRAME is tagged with an origin that indicates that it was generated by the PeerConnection (this prevents ordinary JS from implementing it). The BrowserID signer is provided with Alice's fingerprint. Note that the IFRAME here does not render any UI. It is being used solely to allow the browser to load the BrowserID signer in isolation, especially from the calling site.
3. The BrowserID signer contacts Alice's identity provider, authenticating as Alice (likely via a cookie).
4. The identity provider returns a short-term certificate attesting to Alice's identity and her short-term public key.

5. The Browser-ID code signs the fingerprint and returns the signed assertion + certificate to the PeerConnection. [Note: there are well-understood Web mechanisms for this that I am excluding here for simplicity.]
6. The PeerConnection returns the signed information to the calling JS code.
7. The signed assertion gets sent over the wire to Bob's browser (via the signaling service) as part of the call setup.

Obviously, the format of the signed assertion varies depending on what signaling style the WG ultimately adopts. However, for concreteness, if something like ROAP were adopted, then the entire message might look like:

```
{
  "messageType":"OFFER",
  "callerSessionId":"13456789ABCDEF",
  "seq": 1
  "sdp":"
v=0\n
o=- 2890844526 2890842807 IN IP4 192.0.2.1\n
s= \n
c=IN IP4 192.0.2.1\n
t=2873397496 2873404696\n
m=audio 49170 RTP/AVP 0\n
a=fingerprint: SHA-1 \
4A:AD:B9:B1:3F:82:18:3B:54:02:12:DF:3E:5D:49:6B:19:E5:7C:AB\n",
 "identity":{
    "identityType":"browserid",
      "assertion": {
      "digest":"<hash of fingerprint and session IDs>",
      "audience": "[TBD]"
      "valid-until": 1308859352261,
     }, // signed using user's key
     "certificate": {
       "email": "rescorla@gmail.com",
       "public-key": "<ekrs-public-key>",
       "valid-until": 1308860561861,
     } // certificate is signed by gmail.com
     }
}
```

Note that we only expect to sign the fingerprint values and the session IDs, in order to allow the JS or calling service to modify the rest of the SDP, while protecting the identity binding. [OPEN ISSUE: should we sign seq too?]

[TODO: NEed to talk about Audience a bit.]

On Bob's side, he receives the signed assertion as part of the call setup message and a similar procedure happens to verify it.

1. The calling JS instantiates a PeerConnection and provides it the relevant signaling information, including the signed assertion.
2. The PeerConnection instantiates a BrowserID verifier in an IFRAME and provides it the signed assertion.
3. The BrowserID verifier contacts the identity provider to verify the certificate and then uses the key to verify the signed fingerprint.
4. Alice's verified identity is returned to the PeerConnection (it already has the fingerprint).
5. At this point, Bob's browser can display a trusted UI indication that Alice is on the other end of the call.

When Bob returns his answer, he follows the converse procedure, which provides Alice with a signed assertion of Bob's identity and keying material.

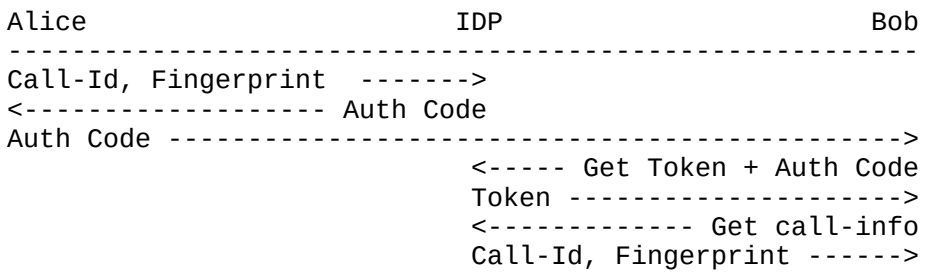
A.3.6.3. OAuth

While OAuth is not directly designed for user-to-user authentication, with a little lateral thinking it can be made to serve. We use the following mapping of OAuth concepts to RTC-Web concepts:

OAuth	RTCWeb
Client	Relying party
Resource owner	Authenticating party
Authorization server	Identity service
Resource server	Identity service

Table 1

The idea here is that when Alice wants to authenticate to Bob (i.e., for Bob to be aware that she is calling). In order to do this, she allows Bob to see a resource on the identity provider that is bound to the call, her identity, and her public key. Then Bob retrieves the resource from the identity provider, thus verifying the binding between Alice and the call.



This is a modified version of a common OAuth flow, but omits the redirects required to have the client point the resource owner to the IDP, which is acting as both the resource server and the authorization server, since Alice already has a handle to the IDP.

Above, we have referred to "Alice", but really what we mean is the PeerConnection. Specifically, the PeerConnection will instantiate an IFRAME with JS from the IDP and will use that IFRAME to communicate with the IDP, authenticating with Alice's identity (e.g., cookie). Similarly, Bob's PeerConnection instantiates an IFRAME to talk to the IDP.

A.3.6.4. Generic Identity Support

I believe it's possible to build a generic interface between the PeerConnection and any identity sub-module so that the PeerConnection just gets pointed to the IDP (which the relying party either trusts or not) and JS from the IDP provides the concrete interfaces. However, I need to work out the details, so I'm not specifying this yet. If it works, the previous two sections will just be examples.

Author's Address

Eric Rescorla
RTFM, Inc.
2064 Edgewood Drive
Palo Alto, CA 94303
USA

Phone: +1 650 678 2350
Email: ekr@rtfm.com

RTCWEB Working Group
Internet-Draft
Intended status: Informational
Expires: April 6, 2012

C. Holmberg
S. Hakansson
G. Eriksson
Ericsson

October 4, 2011

Web Real-Time Communication Use-cases and Requirements
draft-ietf-rtcweb-use-cases-and-requirements-06.txt

Abstract

This document describes web based real-time communication use-cases. Based on the use-cases, the document also derives requirements related to the browser, and the API used by web applications to request and control media stream services provided by the browser.

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 April 6, 2012.

Copyright Notice

Table of Contents

1. Introduction

This document presents a few use-cases of web applications that are executed in a browser and use real-time communication capabilities. Based on the use-cases, the document derives requirements related to the browser and the API used by web applications in the browser.

The requirements related to the browser are named "Fn" and are described in Section 5.2

The requirements related to the API are named "An" and are described in Section 5.3

The document focuses on requirements related to real-time media streams. Requirements related to privacy, signalling between the browser and web server etc. are currently not considered.

2. Conventions

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 BCP 14, RFC 2119 [RFC2119].

3. Definitions

TBD

4. Use-cases

4.1. Introduction

This section describes web based real-time communication use-cases, from which requirements are derived.

The following considerations are applicable to all use cases:

- Clients can be on IPv4-only
- Clients can be on IPv6-only
- Clients can be on dual-stack
- Clients can be on wideband (10s of Mbits/sec)
- Clients can be on narrowband (10s to 100s of Kbits/sec)
- Clients can be on variable-media-quality networks (wireless)
- Clients can be on congested networks

- Clients can be on firewalled networks with no UDP allowed
- Clients can be on networks with cone NAT
- Clients can be on networks with symmetric NAT

4.2. Browser-to-browser use-cases

4.2.1. Simple Video Communication Service

4.2.1.1. Description

Two or more users have loaded a video communication web application into their browsers, provided by the same service provider, and logged into the service it provides. The web service publishes information about user login status by pushing updates to the web application in the browsers. When one online user selects a peer online user, a 1-1 video communication session between the browsers of the two peers is initiated. The invited user might accept or reject the session.

During session establishment a self-view is displayed, and once the session has been established the video sent from the remote peer is displayed in addition to the self-view. During the session, each user can select to remove and re-insert the self-view as often as desired. Each user can also change the sizes of his/her two video displays during the session. Each user can also pause sending of media (audio, video, or both) and mute incoming media

It is essential that the communication cannot be eavesdropped.

Any session participant can end the session at any time.

The two users may be using communication devices of different makes, with different operating systems and browsers from different vendors.

One user has an unreliable Internet connection. It sometimes loses packets, and sometimes goes down completely.

One user is located behind a Network Address Translator (NAT).

4.2.1.2. Derived Requirements

F1, F2, F3, F4, F5, F6, F8, F9, F10, F20, F25, F28

A1, A2, A3, A4, A5, A6, A7, A8, A9, A10, A11, A12

4.2.2. Simple Video Communication Service, NAT/FW that blocks UDP

4.2.2.1. Description

This use-case is almost identical to the Simple Video Communication Service use-case (Section 4.2.1). The difference is that one of the users is behind a NAT that blocks UDP traffic.

4.2.2.2. Derived Requirements

F1, F2, F3, F4, F5, F6, F8, F9, F10, F20, F25, F28, F29

A1, A2, A3, A4, A5, A6, A7, A8, A9, A10, A11, A12

4.2.3. Simple Video Communication Service, global service provider

4.2.3.1. Description

This use-case is almost identical to the Simple Video Communication Service use-case (Section 4.2.1).

What is added is that the service provider is operating over large geographical areas (or even globally).

Assuming that ICE will be used, this means that the service provider would like to be able to provide several STUN and TURN servers (via the app) to the browser; selection of which one(s) to use is part of the ICE processing. Other reasons for wanting to provide several STUN and TURN servers include support for IPv4 and IPv6, load balancing and redundancy.

Note that the additional requirements derived are termed FaI/AaI where aI means "assuming ICE".

4.2.3.2. Derived Requirements

F1, F2, F3, F4, F5, F6, F8, F9, F10, F20, F25, F28

FaI1

A1, A2, A3, A4, A5, A6, A7, A8, A9, A10, A11, A12

AaI1

4.2.4. Simple Video Communication Service, enterprise aspects

4.2.4.1. Description

This use-case is similar to the Simple Video Communication Service use-case (Section 4.2.1).

What is added is aspects when using the service in enterprises. ICE is assumed in the further description of this use-case.

An enterprise that uses a RTCWEB based web application for communication desires to audit all RTCWEB based application session used from inside the company towards any external peer. To be able to do this they deploy a TURN server that straddle the boundary between the internal network and the external.

The firewall will block all attempts to use STUN with an external destination unless they go to the enterprise auditing TURN server. In cases where employees are using RTCWEB applications provided by an external service provider they still want to have the traffic to stay inside their internal network and in addition not load the straddling TURN server, thus they deploy a STUN server allowing the RTCWEB client to determine its server reflexive address on the internal side. Thus enabling cases where peers are both on the internal side to connect without the traffic leaving the internal network. It must be possibele to configure the browsers used in the enterprise with network specific STUN and TURN servers. This should be possible to achieve by autoconfiguration methods. The RTCWEB functionality will need to utilize both network specific STUN and TURN resources and STUN and TURN servers provisioned by the web application.

Note that the additional requirements derived are termed FaI/AaI where aI means "assuming ICE".

4.2.4.2. Derived Requirements

F1, F2, F3, F4, F5, F6, F8, F9, F10, F20, F25, F28

FaI2

A1, A2, A3, A4, A5, A6, A7, A8, A9, A10, A11, A12

4.2.5. Simple Video Communication Service, access change

4.2.5.1. Description

This use-case is almost identical to the Simple Video Communication Service use-case (Section 4.2.1).The difference is that the user changes network access during the session:

The communication device used by one of the users have several network adapters (Ethernet, WiFi, Cellular). The communication device is accessing the Internet using Ethernet, but the user has to start a trip during the session. The communication device automatically changes to use WiFi when the Ethernet cable is removed and then moves to cellular access to the Internet when moving out of WiFi coverage. The session continues even though the access method changes.

4.2.5.2. Derived Requirements

F1, F2, F3, F4, F5, F6, F8, F9, F10, F20, F25, F26, F28

A1, A2, A3, A4, A5, A6, A7, A8, A9, A10, A11, A12

4.2.6. Simple Video Communication Service, QoS

4.2.6.1. Description

This use-case is almost identical to the Simple Video Communication Service, access change use-case (Section 4.2.5). The use of Quality of Service (QoS) capabilities is added:

The user in the previous use case that starts a trip is behind a common residential router that supports prioritization of traffic. In addition, the user’s provider of cellular access has QoS support enabled. The user is able to take advantage of the QoS support both when accessing via the residential router and when using cellular.

4.2.6.2. Derived Requirements

F1, F2, F3, F4, F5, F6, F8, F9, F10, F20, F24, F25, F26, F28

A1, A2, A3, A4, A5, A6, A7, A8, A9, A10, A11, A12

4.2.7. Simple Video Communication Service with sharing

4.2.7.1. Description

This use-case has the audio and video communication of the Simple Video Communication Service use-case (Section 4.2.1).

But in addition to this, one of the users can share what is being displayed on her/his screen with a peer. The user can choose to share the entire screen, part of the screen (part selected by the user) or what a selected applicaton displays with the peer.

4.2.7.2. Derived Requirements

F1, F2, F3, F4, F5, F6, F8, F9, F10, F20, F25, F28, F30

A1, A2, A3, A4, A5, A6, A7, A8, A9, A10, A11, A12, A21

4.2.8. Simple video communication service with inter-operator calling

4.2.8.1. Description

Two users have logged into two different web applications, provided by different service providers.

The service providers are interconnected by some means, but exchange no more information about the users than what can be carried using SIP.

NOTE: More profiling of what this means may be needed.

For each user Alice who has authorized another user Bob to receive login status information, Alice’s service publishes Alice’s login status information to Bob. How this authorization is defined and established is out of scope.

The same functionality as in the the Simple Video Communication Service use-case (Section 4.2.1) is available.

The same issues with connectivity apply.

4.2.8.2. Derived requirements

F1, F2, F3, F4, F5, F6, F8, F9, F10, F20, F25, F27, F28

A1, A2, A3, A4, A5, A6, A7, A8, A9, A10, A11, A12, A20

4.2.9. Hockey Game Viewer

4.2.9.1. Description

An ice-hockey club uses an application that enables talent scouts to, in real-time, show and discuss games and players with the club manager. The talent scouts use a mobile phone with two cameras, one front facing and one rear facing.

The club manager uses a desktop, equipped with one camera, for viewing the game and discussing with the talent scout.

Before the game starts, and during game breaks, the talent scout and

the manager have a 1-1 video communication. Only the rear facing camera of the mobile phone is used. On the display of the mobile phone, the video of the club manager is shown with a picture-in-picture thumbnail of the rear facing camera (self-view). On the display of the desktop, the video of the talent scout is shown with a picture-in-picture thumbnail of the desktop camera (self-view).

When the game is on-going, the talent scout activates the use of the front facing camera, and that stream is sent to the desktop (the stream from the rear facing camera continues to be sent all the time). The video stream captured by the front facing camera (that is capturing the game) of the mobile phone is shown in a big window on the desktop screen, with picture-in-picture thumbnails of the rear facing camera and the desktop camera (self-view). On the display of the mobile phone the game is shown (front facing camera) with picture-in-picture thumbnails of the rear facing camera (self-view) and the desktop camera.

It is essential that the communication cannot be eavesdropped.

4.2.9.2. Derived Requirements

F1, F2, F3, F4, F5, F6, F8, F9, F10, F17, F20

A1, A2, A3, A4, A5, A7, A8, A9, A10, A11, A12, A17

4.2.10. Multiparty video communication

4.2.10.1. Description

In this use-case is the Simple Video Communication Service use-case (Section 4.2.1) is extended by allowing multiparty sessions. No central server is involved - the browser of each participant sends and receives streams to and from all other session participants. The web application in the browser of each user is responsible for setting up streams to all receivers.

In order to enhance intelligibility, the web application pans the audio from different participants differently when rendering the audio. This is done automatically, but users can change how the different participants are placed in the (virtual) room. In addition the levels in the audio signals are adjusted before mixing.

Another feature intended to enhance the use experience is that the video window that displays the video of the currently speaking peer is highlighted.

Each video stream received is by default displayed in a thumbnail

frame within the browser, but users can change the display size.

It is essential that the communication cannot be eavesdropped.

Note: What this use-case adds in terms of requirements is capabilities to send streams to and receive streams from several peers concurrently, as well as the capabilities to render the video from all recevied streams and be able to spatialize, level adjust and mix the audio from all received streams locally in the browser. It also adds the capability to measure the audio level/activity.

4.2.10.2. Derived Requirements

F1, F2, F3, F4, F5, F6, F8, F9, F10, F11, F12, F13, F14, F15, F16, F17, F20, F25

A1, A2, A3, A4, A5, A6, A7, A8, A9, A10, A11, A12, A13, A14, A15, A16, A17

4.2.11. Multiparty on-line game with voice communication

4.2.11.1. Description

This use case is based on the previous one. In this use-case, the voice part of the multiparty video communication use case is used in the context of an on-line game. The received voice audio media is rendered together with game sound objects. For example, the sound of a tank moving from left to right over the screen must be rendered and played to the user together with the voice media.

Quick updates of the game state is required.

It is essential that the communication cannot be eavesdropped.

Note: the difference regarding local audio processing compared to the "Multiparty video communication" use-case is that other sound objects than the streams must be possible to be included in the spatialization and mixing. "Other sound objects" could for example be a file with the sound of the tank; that file could be stored locally or remotely.

4.2.11.2. Derived Requirements

F1, F2, F3, F4, F5, F6, F8, F9, F11, F12, F13, F14, F15, F16, F18, F20, F23

A1, A2, A3, A4, A5, A7, A8, A9, A10, A11, A12, A13, A14, A15, A16, A17, A18

4.2.12. Distributed Music Band

4.2.12.1. Description

In this use-case, a music band is playing music while the members are at different physical locations. No central server is used, instead all streams are set up in a mesh fashion.

Discussion: This use-case was briefly discussed at the Quebec webrtc meeting and it got support. So far the only concrete requirement (A17) derived is that the application must be able to ask the browser to treat the audio signal as audio (in contrast to speech). However, the use case should be further analysed to determine other requirements (could be e.g. on delay mic->speaker, level control of audio signals, etc.).

4.2.12.2. Derived Requirements

F1, F2, F3, F4, F5, F6, F8, F9, F11, F12, F13, F14, F15, F16

A1, A2, A3, A4, A5, A7, A8, A9, A10, A11, A12, A13, A14, A15, A16, A19

4.3. Browser - GW/Server use cases

4.3.1. Telephony terminal

4.3.1.1. Description

A mobile telephony operator allows its customers to use a web browser to access their services. After a simple log in the user can place and receive calls in the same way as when using a normal mobile phone. When a call is received or placed, the identity is shown in the same manner as when a mobile phone is used.

It is essential that the communication cannot be eavesdropped.

Note: With "place and receive calls in the same way as when using a normal mobile phone" it is meant that you can dial a number, and that your mobile telephony operator has made available your phone contacts on line, so they are available and can be clicked to call, and be used to present the identity of an incoming call. If the callee is not in your phone contacts the number is displayed. Furthermore, your call logs are available, and updated with the calls made/ received from the browser. And for people receiving calls made from the web browser the usual identity (i.e. the phone number of the mobile phone) will be presented.

4.3.1.2. Derived Requirements

F1, F2, F3, F4, F5, F6, F8, F9, F10, F20, F21

A1, A2, A3, A4, A7, A8, A9, A10, A11, A12

4.3.2. Fedex Call

4.3.2.1. Description

Alice uses her web browser with a service something like Skype to be able to phone PSTN numbers. Alice calls 1-800-gofedex. Alice should be able to hear the initial prompts from the fedex IVR and when the IVR says press 1, there should be a way for Alice to navigate the IVR.

4.3.2.2. Derived Requirements

F1, F2, F3, F4, F5, F6, F8, F9, F10, F21, F22

A1, A2, A3, A4, A7, A8, A9, A10, A11, A12

4.3.3. Video conferencing system with central server

4.3.3.1. Description

An organization uses a video communication system that supports the establishment of multiparty video sessions using a central conference server.

The browser of each participant send an audio stream (type in terms of mono, stereo, 5.1, ... depending on the equipment of the participant) to the central server. The central server mixes the audio streams (and can in the mixing process naturally add effects such as spatialization) and sends towards each participant a mixed audio stream which is played to the user.

The browser of each participant sends video towards the server. For each participant one high resolution video is displayed in a large window, while a number of low resolution videos are displayed in smaller windows. The server selects what video streams to be forwarded as main- and thumbnail videos respectively, based on speech activity. As the video streams to display can change quite frequently (as the conversation flows) it is important that the delay from when a video stream is selected for display until the video can be displayed is short.

The organization has an internal network set up with an aggressive

firewall handling access to the Internet. If users cannot physically access the internal network, they can establish a Virtual Private Network (VPN).

It is essential that the communication cannot be eavesdropped.

All participants are authenticated by the central server, and authorized to connect to the central server. The participants are identified to each other by the central server, and the participants do not have access to each others' credentials such as e-mail addresses or login IDs.

Note: This use-case adds requirements on support for fast stream switches F7, on encryption of media and on ability to traverse very restrictive FWs. There exist several solutions that enable the server to forward one high resolution and several low resolution video streams: a) each browser could send a high resolution, but scalable stream, and the server could send just the base layer for the low resolution streams, b) each browser could in a simulcast fashion send one high resolution and one low resolution stream, and the server just selects or c) each browser sends just a high resolution stream, the server transcodes into low resolution streams as required.

4.3.3.2. Derived Requirements

F1, F2, F3, F4, F5, F6, F7, F8, F9, F10, F17, F19, F20

A1, A2, A3, A4, A5, A7, A8, A9, A10, A11, A12, A17

5. Requirements

5.1. General

This section contains the requirements derived from the use-cases in section 4.

NOTE: It is assumed that the user applications are executed on a browser. Whether the capabilities to implement specific browser requirements are implemented by the browser application, or are provided to the browser application by the underlying operating system, is outside the scope of this document.

5.2. Browser requirements

REQ-ID	DESCRIPTION

F1	The browser MUST be able to use microphones and cameras as input devices to generate streams.
F2	The browser MUST be able to send streams to a peer in the presence of NATs.
F3	Transmitted streams MUST be rate controlled.
F4	The browser MUST be able to receive, process and render streams from peers.
F5	The browser MUST be able to render good quality audio and video even in the presence of reasonable levels of jitter and packet losses. TBD: What is a reasonable level?
F6	The browser MUST be able to handle high loss and jitter levels in a graceful way.
F7	The browser MUST support fast stream switches.
F8	The browser MUST detect when a stream from a peer is not received anymore
F9	When there are both incoming and outgoing audio streams, echo cancellation MUST be made available to avoid disturbing echo during conversation. QUESTION: How much control should be left to the web application?
F10	The browser MUST support synchronization of audio and video. QUESTION: How much control should be left to the web application?
F11	The browser MUST be able to transmit streams to several peers concurrently.
F12	The browser MUST be able to receive streams from multiple peers concurrently.
F13	The browser MUST be able to apply spatialization effects to audio streams.

F14	The browser MUST be able to measure the level in audio streams.
F15	The browser MUST be able to change the level in audio streams.
F16	The browser MUST be able to render several concurrent video streams
F17	The browser MUST be able to mix several audio streams.
F18	The browser MUST be able to process and mix sound objects (media that is retrieved from another source than the established media stream(s) with the peer(s) with audio streams.
F19	Streams MUST be able to pass through restrictive firewalls.
F20	It MUST be possible to protect streams from eavesdropping.
F21	The browser MUST support an audio media format (codec) that is commonly supported by existing telephony services. QUESTION: G.711?
F22	There should be a way to navigate the IVR
F23	The browser must be able to send short latency datagram traffic to a peer browser
F24	The browser MUST be able to take advantage of capabilities to prioritize voice and video appropriately.
F25	The browser SHOULD use encoding of streams suitable for the current rendering (e.g. video display size) and SHOULD change parameters if the rendering changes during the session
F26	It MUST be possible to move from one network interface to another one

F27	The browser MUST be able to initiate and accept a media session where the data needed for establishment can be carried in SIP.
F28	The browser MUST support a baseline audio and video codec
F29	The browser MUST be able to send streams to a peer in the presence of NATs that block UDP traffic.
F30	The browser MUST be able to use the screen (or a specific area of the screen) or what a certain application displays on the screen to generate streams.
FaI1	The browser MUST be able to use several STUN and TURN servers
FaI2	There browser MUST support that STUN and TURN servers to use are supplied by other entities than the service provided (i.e. the network provider)

5.3. API requirements

REQ-ID	DESCRIPTION
A1	The Web API MUST provide means for the application to ask the browser for permission to use cameras and microphones as input devices.
A2	The Web API MUST provide means for the web application to control how streams generated by input devices are used.
A3	The Web API MUST provide means for the web application to control the local rendering of streams (locally generated streams and streams received from a peer).
A4	The Web API MUST provide means for the web application to initiate sending of stream/stream components to a peer.
A5	The Web API MUST provide means for the web application to control the media format (codec) to be used for the streams sent to a peer.

	NOTE: The level of control depends on whether the codec negotiation is handled by the browser or the web application.
A6	The Web API MUST provide means for the web application to modify the media format for streams sent to a peer after a media stream has been established.
A7	The Web API MUST provide means for informing the web application of whether the establishment of a stream with a peer was successful or not.
A8	The Web API MUST provide means for the web application to mute/unmute a stream or stream component(s). When a stream is sent to a peer mute status must be preserved in the stream received by the peer.
A9	The Web API MUST provide means for the web application to cease the sending of a stream to a peer.
A10	The Web API MUST provide means for the web application to cease processing and rendering of a stream received from a peer.
A11	The Web API MUST provide means for informing the web application when a stream from a peer is no longer received.
A12	The Web API MUST provide means for informing the web application when high loss rates occur.
A13	The Web API MUST provide means for the web application to apply spatialization effects to audio streams.
A14	The Web API MUST provide means for the web application to detect the level in audio streams.
A15	The Web API MUST provide means for the web application to adjust the level in audio streams.

A16	The Web API MUST provide means for the web application to mix audio streams.
A17	For each stream generated, the Web API MUST provide an identifier that is accessible by the application. The identifier MUST be accessible also for a peer receiving that stream and MUST be unique relative to all other stream identifiers in use by either party.
A18	In addition to the streams listed elsewhere, the Web API MUST provide a mechanism for sending and receiving isolated discrete chunks of data.
A19	The Web API MUST provide means for the web application indicate the type of audio signal (speech, audio)for audio stream(s)/stream component(s).
A20	It must be possible for an initiator or a responder Web application to indicate the types of media he's willing to accept incoming streams for when setting up a connection (audio, video, other). The types of media he's willing to accept can be a subset of the types of media the browser is able to accept.
A21	The Web API MUST provide means for the application to ask the browser for permission to the screen, a certain area on the screen or what a certain application displays on the screen as input to streams.
AaI1	The Web API MUST provide means for the application to specify several STUN and/or TURN servers to use.

6. IANA Considerations

TBD

7. Security Considerations

7.1. Introduction

A malicious web application might use the browser to perform Denial Of Service (DOS) attacks on NAT infrastructure, or on peer devices.

Also, a malicious web application might silently establish outgoing, and accept incoming, streams on an already established connection.

Based on the identified security risks, this section will describe security considerations for the browser and web application.

7.2. Browser Considerations

The browser is expected to provide mechanisms for getting user consent to use device resources such as camera and microphone.

The browser is expected to provide mechanisms for informing the user that device resources such as camera and microphone are in use ("hot").

The browser is expected to provide mechanisms for users to revise and even completely revoke consent to use device resources such as camera and microphone.

The browser is expected to provide mechanisms for getting user consent to use the screen (or a certain part of it) or what a certain application displays on the screen as source for streams.

The browser is expected to provide mechanisms for informing the user that the screen, part thereof or an application is serving as a stream source ("hot").

The browser is expected to provide mechanisms for users to revise and even completely revoke consent to use the screen, part thereof or an application is serving as a stream source.

The browser is expected to provide mechanisms in order to assure that streams are the ones the recipient intended to receive.

The browser needs to ensure that media is not sent, and that received media is not rendered, until the associated stream establishment and handshake procedures with the remote peer have been successfully finished.

The browser needs to ensure that the stream negotiation procedures are not seen as Denial Of Service (DOS) by other entities.

7.3. Web Application Considerations

The web application is expected to ensure user consent in sending and receiving media streams.

8. Additional use-cases

Several additional use-cases have been discussed. At this point these use-cases are not included as requirement deriving use-cases for different reasons (lack of documentation, overlap with existing use-cases, lack of consensus). For completeness these additional use-cases are listed below:

1. Use-cases regarding different situations when being invited to a "session", e.g. browser open, browser open but another tab active, browser open but active in session, browser closed, ... (Matthew Kaufman); discussed at webrtc meeting
2. E911 (Paul Beaumont) http://www.ietf.org/mail-archive/web/rtcweb/current/msg00525.html, followed up by Stephan Wenger
3. Local Recording and Remote recording (John): Discussed a _lot_ on the mail lists (rtcweb as well as public-webrtc) lAugust and September 2011. Concrete proposal: http://www.ietf.org/mail-archive/web/rtcweb/current/msg01006.html (remote) and http://www.ietf.org/mail-archive/web/rtcweb/current/msg00734.html (local)
4. Emergency access for disabled (Bernard Aboba) http://www.ietf.org/mail-archive/web/rtcweb/current/msg00478.html
5. Clue use-cases (Roni Even) http://tools.ietf.org/html/draft-ietf-clue-telepresence-use-cases-01
6. Rohan red cross (Cullen Jennings); http://www.ietf.org/mail-archive/web/rtcweb/current/msg00323.html
7. Security camera/baby monitor usage http://www.ietf.org/mail-archive/web/rtcweb/current/msg00543.html
8. Large multiparty session http://www.ietf.org/mail-archive/web/rtcweb/current/msg00530.html

9. Acknowledgements

Dan Burnett has reviewed and proposed a lot of things that enhances the document. Most of this has been incorporated in rev -05.

Stephan Wenger has provided a lot of useful input and feedback, as well as editorial comments.

Harald Alvestrand and Ted Hardie have provided comments and feedback on the draft.

Harald Alvestrand and Cullen Jennings have provided additional use-cases.

Thank You to everyone in the RTCWEB community that have provided comments, feedback and improvement proposals on the draft content.

10. Change Log

[RFC EDITOR NOTE: Please remove this section when publishing]

Changes from draft-ietf-rtcweb-use-cases-and-requirements-05

- Added use-case "global service provider", derived reqs associated with several STUN/TURN servers
- Added use-case "enterprise aspects", derived req associated with enabling the network provider to supply STUN and TURN servers
- The requirements from the above are ICE specific and labeled accordingly
- Separated the requirements phrased like "processing such as pan, mix and render" for audio to be specific reqs on spatialization, level measurement, level adjustment and mixing (discussed on the lists in http://www.ietf.org/mail-archive/web/rtcweb/current/msg01648.html and http://lists.w3.org/Archives/Public/public-webrtc/2011Sep/0102.html)
- Added use-case on sharing as decided in http://www.ietf.org/mail-archive/web/rtcweb/current/msg01700.html, derived reqs F30 and A21
- Added the list of common considerations proposed in mail http://www.ietf.org/mail-archive/web/rtcweb/current/msg01562.html to the Introduction of the use-case section

Changes from draft-ietf-rtcweb-use-cases-and-requirements-04

- Most changes based on the input from Dan Burnett http://www.ietf.org/mail-archive/web/rtcweb/current/msg00948.html
- Many editorial changes
- 4.2.1.1 Clarified
- Some clarification added to 4.3.1.1 as a note
- F-requirements updated (see reply to Dan’s mail).
- Almost all A-requirements updated to start "The Web API MUST provide ..."
- A8 removed, A9 rephrased to cover A8 and old A9
- A15 rephrased
- For more details, and discussion, look att the response to Dan’s mail http://www.ietf.org/mail-archive/web/rtcweb/current/msg01177.html

Changes from draft-ietf-rtcweb-use-cases-and-requirements-03

- Editorials
- Changed when the self-view is displayed in 4.2.1.1, and added words about allowing users to remove and re-insert it.

- Clarified 4.2.6.1
- Removed the "mono" stuff from 4.2.7.1
- Added that communication should not be possible to eavesdrop to most use cases - and req. F17
- Re-phrased 4.3.3.1 to not describe the technical solution so much, and removed "stereo" stuff. Solution possibilities are now in a note.
- Re-inserted API requirements after discussion in the W3C webrtc WG. (Re-phrased A15 and added A18 compared to version -02).

Changes from draft-ietf-rtcweb-use-cases-and-requirements-02

- Removed desrciption/list of API requirements, instead
- Reference to W3C webrtc_reqs document for API requirements

Changes from draft-ietf-rtcweb-ucreqs-01

- Changed Intended status to Information
- Changed "Ipr" to "trust200902"
- Added use case "Simple video communication service, NAT/FW that blocks UDP", and derived new req F26
- Added use case "Distributed Music Band" and derived new req A17
- Added F24 as requirement derived from use case "Simple video communication service with inter-operator calling"
- Added section "Additional use cases"
- Added text about ID handling to multiparty with central server use case
- Re-phrased A1 slightly

Changes from draft-ietf-rtcweb-ucreqs-00

- - Reshuffled: Just two main groups of use cases (b2b and b2GW/ Server); removed some specific use cases and added them instead as flavors to the base use case (Simple video communciation)
- - Changed the fromulation of F19
- - Removed the requirement on an API for DTMF
- - Removed "FX3: There SHOULD be a mapping of the minimum needed data for setting up connections into SIP, so that the restriction to SIP-carriable data can be verified. Not a rew on the browser but rather on a document"
- - (see http://www.ietf.org/mail-archive/web/rtcweb/current/msg00227.html for more details)
- -Added text on informing user of that mic/cam is being used and that it must be possible to revoce permission to use them in section 7.

Changes from draft-holmberg-rtcweb-ucreqs-01

- o - Draft name changed to draft-ietf-rtcweb-ucreqs
- o - Use-case grouping introduced
- o - Additional use-cases added
- o - Additional reqs added (derived from use cases): F19-F25, A16-A17

Changes from draft-holmberg-rtcweb-ucreqs-00

- o - Mapping between use-cases and requirements added (Harald Alvestrand, 090311)
- o - Additional security considerations text (Harald Alvestrand, 090311)
- o - Clarification that user applications are assumed to be executed by a browser (Ted Hardie, 080311)
- o - Editorial corrections and clarifications

11. References

11.1. Normative References

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

11.2. Informative References

[webrtc_reqs] "Webrt requirements, http://dev.w3.org/2011/webrtc/editor/webrtc_reqs.html".

Authors' Addresses

Christer Holmberg
Ericsson
Hirsalantie 11
Jorvas 02420
Finland

Email: christer.holmberg@ericsson.com

Stefan Hakansson
Ericsson
Laboratoriegrand 11
Lulea 97128
Sweden

Email: stefan.lk.hakansson@ericsson.com

Goran AP Eriksson
Ericsson
Farogatan 6
Stockholm 16480
Sweden

Email: goran.ap.eriksson@ericsson.com

Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: May 2, 2012

C. Jennings
Cisco
J. Rosenberg
jdrosen.net
J. Uberti
Google
R. Jesup
Mozilla

October 30, 2011

RTCWeb Offer/Answer Protocol (ROAP)
draft-jennings-rtcweb-signaling-01

Abstract

This document describes an protocol used to negotiate media between browsers or other compatible devices. This protocol provides the state machinery needed to implement the offer/answer model (RFC 3264), and defines the semantics and necessary attributes of messages that must be exchanged. The protocol uses an abstract transport in that it does not actually define how these messages are exchanged. Rather, such exchanges are handled through web-based transports like HTTP or WebSockets. The protocol focuses solely on media negotiation and does not handle call control, call processing, or other functions.

Table of Contents

1. Introduction

This specification defines a protocol that allows an RTCWeb browser to exchange information to control the set up of media to another browser or device. The scope of this protocol is limited to functionality required for the setup and negotiation of media and the associated transports, referred to as media control. The protocol defines the minimum set of messages and state machinery necessary to implement the offer/answer model as defined in [RFC3264]. The offer answer model specifies rules for the bilateral exchange of Session Description Protocol (SDP) messages [RFC4566] for creation of media streams.

The protocol specified here defines the state machines, semantic behaviors, and messages that are exchanged between instances of the state machines. However, it does not specify the actual on the wire transport of these messages. Rather, it assumes that the implementation of this protocol would occur within the browser itself, and then browser APIs would allow the application's JavaScript to request creation of messages and insert messages into the state machine. The actual transfer of these messages would be the responsibility of the web application, and would utilize protocols such as HTTP and WebSockets. To facilitate implementation within a browser, messages are encoded in JSON [RFC4627]. This protocol, with appropriate selected transports, could also be implemented by a signalling gateway that converts ROAP to SIP or Jingle.

This protocol is designed to be closely aligned with the PeerConnection API defined in the RTCWeb API[webrtc-api] specification. It is important to note that while ROAP does not require what has been referred to as a low level API for media manipulation, ROAP does not prevent having a such an API as well and both styles of API could coexist and be used where appropriate.

The protocol defined here does not provide any call control. Concepts like ringing of phones, user search, call forwarding, redirection, transfer, hold, and so on, are all the domain of call processing and are out of scope for this specification. It is assumed that the application running within the browser provides any call control based on the needs of the application, the scope of which is not a matter for standardization.

Despite that fact that it has an abstract transport, ROAP is still a protocol. This means it has state machines, and it has rules governing the behavior of those state machines which guarantee that system operates properly based on any set of inputs. It is assumed that this state machinery is implemented in the browser and thus

immutable by the application, which can then guarantee proper behavior regardless of the operation of the resident JavaScript.

The protocol is designed to operate between two entities (browsers for example), which exchange messages "directly" - meaning that a message output by one entity is meant to be directly processed by the other entity without further modification. In practice, this means that a web server can treat ROAP messages as opaque and just shuffle them between browser instances. This allows for simple implementations. However, more powerful applications can be built in which the web server or JavaScript can modify the messages in order to provide more complex features. As long as those modifications produce messages compliant to this specification, SDP Offer/Answer [RFC3264], SDP [RFC4566], ICE [RFC5245] and any other dependencies, interoperability is still possible.

This protocol is designed for two major use cases:

- Browser to browser
- Browser to SIP device via a SIP gateway

In the browser to SIP use case, the gateway obviously needs to be somewhat more sophisticated. However, because this design is a small subset of the design space covered by SIP [RFC3261], it is intended to be simple to translate to and from/SIP via a signalling gateway. Moreover, many of the elements in messages have clear mappings to elements in SIP messages, thus allowing simple, stateless translation.

2. Requirements and Design Goals

There has been extensive debate about the best architecture for RTCWeb signaling. To a great extent this decision is dictated by the requirements that the signaling mechanism is intended to fit. The protocol in this document was designed to minimize the amount of implementation effort required outside the browser and RTC-Web signaling gateways. This implies the following requirements:

It should be possible to develop a simple browser to browser voice and video service in a small amount of code. In particular, it MUST be possible to implement a functional service such that:

- It’s possible to build a web service that maintains only transaction state, not call state;
- In the browser to browser case, the web server can simply pass protocol messages between the browser agents without examining or modifying them;

- The service operates without needing to examine the details of the browser capabilities (e.g., new codecs should be automatically accommodated without modifying either the service or the associated JS.

It should be possible to implement a simple RTC-Web gateway that:

- Connects to legacy SIP devices ranging from multiscreen video phones to PSTN gateways;
- Has a deterministic mapping between RTC-Web messages and SIP messages;
- Permits the mechanical translation of messages without knowledge of the details of all the browser capabilities;
- is only required to maintains transaction state, not call state (note is fine if an implementation want to maintain call state); and
- Does not need to send or receive the media (unless also acting as a relay or a translator for codecs which are not jointly supported).

Finally it seems clear that SDP is too complicated to reinvent, so despite its manifest deficiencies we opt to take it as-is rather than trying to reinvent it.

3. 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].

4. Protocol Overview

We start with a simple example. Consider the case where browser A wishes to setup up a media session with browser B. At the high level, A needs to communicate the following information:

- This is a new media session and not an update to a different session.
- Here is A’s SDP offer, including media parameters and ICE candidates.

The OFFER message is used to carry this information. For example, A might send B:

```
{
  "messageType":"OFFER",
  "offererSessionId":"13456789ABCDEF",
  "seq": 1,
  "sdp":"
v=0\n
o=- 2890844526 2890842807 IN IP4 192.0.2.1\n
s= \n
c=IN IP4 192.0.2.1\n
t=2873397496 2873404696\n
m=audio 49170 RTP/AVP 0"
}
```

The messageType field indicates that this is an OFFER and the offererSessionId indicates the media session that this OFFER is associated with. B can tell that this is for a new media session because it contains a offererSessionId that he has not seen before. The sdp field contains the offer itself, which is just an ordinary SDP offer rendered as a string.

If B elects to start a media session, B responds with an ANSWER message containing SDP, as shown below.

```
{
  "messageType":"ANSWER",
  "offererSessionId":"13456789ABCDEF",
  "answererSessionId":"abc1234356",
  "seq": 1,
  "sdp":"
v=0\n
o=- 2890844526 2890842807 IN IP4 192.0.2.3\n
s= \n
c=IN IP4 192.0.2.3\n
t=2873397496 2873404696\n
m=audio 49175 RTP/AVP 0"
}
```

The contents of this message are more or less the same as those in the OFFER, except that B also includes a answererSessionId to uniquely identify the session from B’s perspective. The combination of offererSessionId and answererSessionId uniquely identifies this session.

Finally, in order to confirm that A has seen B’s ANSWER, A responds with an OK message.

```
{
  "messageType":"OK",
  "offererSessionId":"13456789ABCDEF",
  "answererSessionId":"abc1234356",
  "seq": 1
}
```

Note that all of these messages contain a seq field which contains a transaction sequence number. The seq field makes it possible to correlate messages which belong to the same transaction, as well as to detect duplicates, which is described later in section Section 5.1.

The messageType value of "OFFER" will always contain an SDP offer, and an object with a messageType value of "ANSWER" will always contain an SDP answer. The complete list of message types is defined in Section 5. Only a small number of messages are permitted and much of the message set is devoted to error handling.

In building web systems it is often useful for a request to contain some state that is passed back in future messages. This system includes two types of state: session state and request state. If a browser receives a message that contains state in a setSessionState attribute, any future messages it sends that have the same offererSessionId MUST include this state in a sessionState attribute. Similarly if a request contains an setResponseState attribute, that state MUST be included in any response to that request in a responseState attribute.

Once a session has been set up, additional rounds of offer/answer can be sent using the OFFER/ANSWER/OK sequence. Note that the seq attribute makes it easy to differentiate these additional rounds from the initial exchange and from each other.

At the point that one side which to end the session, it simply sends a SHUTDOWN message which is responded to with an OK response. A SHUTDOWN can be sent regardless of it any response has been received to the initial OFFER. The key purpose of the SHUTDOWN messages is to allow the other side to know they can clean up any state associated with the session.

5. Semantics & Syntax

5.1. Reliability Model

ROAP messages are typically carried over a reliable transport (likely HTTP via XMLHttpRequest or WebSockets), so the chance of message loss

is low (though non-zero), provided that the signaling service is up. However, the common web reliability and scaleability model is based on the principle that transactions are idempotent and that requests can just be discarded and will be retried. A retry of a transaction might happened if a given host was down and the DNS round robin approach wanted to move to the next server, or if a server was overloaded, or if there was a hiccup in the network. Web applications that want to work well need to deal with theses issues to get the advantages of the general web design pattern for scaleability and reliability. Because only the application knows what its internal reliability characteristics are, the JS application (and whatever associated servers it uses) are ultimately responsible for ensuring end-to-end delivery; the browser simply assumes that messages which are provided to the JS will be delivered eventually.

However, in order to maintain OFFER/ANSWER transaction state, the SDP state machine does need to understand when the far end has received an ANSWER if it caused an error or not. To support this model, OFFER and ANSWER messages are acknowledged end to end with an ANSWER or OK however any retransmission need to be handled by the JS or whatever is providing the transport of the ROAP messages. The combination of the sessionID and seq allow the browser to detect and discard duplicate requests and to detect glare.

NOTE: The split of the reliability model between the JS and browser is something where implementations are playing around with and trying to get some experience with what works best. This is an area that is highly likely to change as understanding of the implications evolves.

5.2. Common Fields

5.2.1. Session IDs

Each call is identified by a pair of session identifiers:

offererSessionId The offerer’s half of the session ID (supplied in the OFFER)

answererSessionId The answerer’s half of the session ID (supplied in the response to an OFFER)

The session ID values MUST be generated so that they are globally unique. Thus, the combination of both sessionIds is itself globally unique. Session IDs never change for the duration of an media session.

All messages MUST contain the "offererSessionId", and all messages

other than OFFER or an error in response to an OFFER MUST contain both "offererSessionId" and "answererSessionId".

5.2.2. Seq

This is a sequence counter for the key requests that helps correlate responses to the correct request.

This is a 32-bit unsigned integer. On each new OFFER (from either browser) it is incremented by one. The Seq of an OK or ANSWER is set to the same Seq that was used in the OFFER which caused it. When a PeerConnection objects originates a new session by sending an OFFER type message, it starts the Seq at 1.

Note: If browser A starts an OFFER/ANSWER/OK transaction with a seq of 1 to browser B, then later B initiates a second OFFER/ANSWER?/OK transaction, it will have a seq of 2.

5.2.3. Session Tokens

While session IDs serve to uniquely identify a session, it may be useful to allow one or another sides to offload state onto the other side (for instance to enable a stateless gateway). The "setSessionToken" and "sessionToken" fields are used for this purpose. When an implementation receives a message with a "setSessionToken" field, it MUST associate the field value with the session. For all future messages in the session MUST send the associated value in the "sessionToken" field (unless the session token is reset by another "setSessionToken" value). If no session token has yet been received, the "sessionToken" field MUST be omitted.

5.2.4. Response Tokens

In addition to tokens which persist for the life of a session, it is also possible to have tokens which are only valid for the lifetime of a given request/response pair. The "setResponseToken" and "responseToken" fields are used for this purpose.

When an implementation responds to a message from the other side (e.g., supplies an answer to an offer, or replies to an answer with an OK), it MUST copy into the "responseToken" field any value found in a "setResponseToken" field in the message being responded to. If no "setResponseToken" field is present, then the "responseToken" field MUST be omitted.

5.3. Media Setup

In order to initiate sending media between the browsers, the offerer sends an OFFER message. In order to accept the media, the answerer responds with an ANSWER message. A sample message flow for this is shown below:

```
participant OffererUA
participant OffererJS
participant AnswererJS
participant AnswererUA
OffererJS->OffererUA: peer=new PeerConnection();

OffererJS->OffererUA: peer->addStream();
OffererUA->OffererJS: sendSignalingChannel();
OffererJS->AnswererJS: {"type":"OFFER", "sdp":"..."}
AnswererJS->AnswererUA: peer=new PeerConnection();
AnswererJS->AnswererUA: peer->processSignalingMessage();
AnswererUA->AnswererJS: onconnecting();

AnswererUA->OffererUA: ICE starts checking

note right of AnswererUA: User decides it is OK to send video
AnswererJS->AnswererUA: peer->addStream();
AnswererUA->OffererUA: Media

AnswererUA->AnswererJS: sendSignalingChannel();
AnswererJS->OffererJS: {"type":"ANSWER","sdp":"..."}
OffererJS->OffererUA: peer->processSignalingMessage();
OffererUA->OffererJS: onaddstream();
OffererUA->AnswererUA: Media

AnswererUA->OffererUA: ICE Completes
AnswererUA->AnswererJS: onopen();
OffererUA->OffererJS: onopen();

OffererUA->OffererJS: sendSignalingChannel();
OffererJS->AnswererJS: {"type":"OK" }
AnswererJS->AnswererUA: peer->processSignalingMessage();
AnswererUA->AnswererJS: onaddstream();
```

The above figure shows a simple message flow for negotiating media:

- The offerer sends an OFFER to initiate the call;
- At this point, ICE negotiation starts;
- Once the browser authorizes sending media to the far side, the answerer sends an ANSWER containing the media parameters; and finally,

- Once ICE is completed and an OK to the ANSWER is received, both sides know that media can flow.

The contents of each of these messages is detailed below.

5.3.1. OFFER Message

The first OFFER message with a given offererSessionId is used to indicate the desire to start a media session.

5.3.1.1. Offerer Behavior

In order to start a new media session, a offerer constructs a new OFFER message with a fresh offererSessionId. The answererSessionId field MUST be empty. Like all SDP offers, the message MUST contain an "sdp" field with the offerer's offer. It MUST also contain the tieBreaker field, containing a 32 bit random integer used for glare resolution as described in Section 5.4.1.

5.3.1.2. Answerer Behavior

A answerer can receive an OFFER in three cases:

- A new session (this is detected by seeing a new offererSessionId value);
- A retransmit of a new OFFER (known offererSessionId, empty answererSessionId); or
- A request to change media parameters (known offererSessionId, known answererSessionId, new seq value).

The first two situations are described in this section. The third case is described in Section 5.4. Any other condition represents an alien packet and SHOULD be rejected with Error:NOMATCH

If no media session exists with the given "offererSessionId" value, then this is a new media session. The answerer has three primary options:

- Reject the request, either silently with no response or with an Error:REFUSED message;
- Reply to the OFFER message with a final ANSWER message; or Section 5.3.2
- Send back a non final ANSWER message and then later respond with an final ANSWER.

In either of the latter two cases, the answerer performs the following steps:

1. Generate a "answererSessionId" value;
2. Create some local call state (i.e., a PeerConnection object) and bind it to the "offererSessionId"/"answererSessionId" pair. All future messages on this session MUST then be delivered to that PeerConnection object;
3. Start ICE handshaking with the offerer; and finally,
4. Respond with a message containing an SDP answer in the "sdp" field. This will contain the answerer's (potentially with moreComing=true) media information and the ICE parameters.

If an OFFER is received that has already been received and responded to and the media session still exists, then the answerer MUST respond with the same message as before. If the session has been terminated in the meantime, then an Error:NOMATCH message SHOULD be sent.

5.3.2. ANSWER

The ANSWER message is used by the receiver of an OFFER message to indicate that the offer has been accepted. The ANSWER message MUST contain the answererSessionId for this media session and an sdp parameter containing ICE candidates and the final media parameters for the session (although of course these can be adjusted by a new OFFER/ANSWER exchange. See Section 5.4). In addition, ANSWERs MAY contain the moreComing flag, as described below.

5.3.2.1. moreComing Flag

This is a boolean flag that can only appear in an ANSWER and, if set to true, indicates that this answer is not the final answer that will be sent for the associated OFFER. If this flag is not present, it is assumed to be false.

One motivating use case for moreComing is where an Agent wishes to respond immediately to an OFFER in order to start ICE checking before the user has provided authorization to send media. The Agent cannot send an ANSWER containing media information but can send ICE candidate. In this case, the Agent could send an ANSWER that had moreComing=true but that allowed ICE to start. Then later, when the user had authorized the media, the Agent could send an ANSWER with the moreComing flag=false that indicated this was the final media selection.

To see why simply having multiple independent offers (as opposed to multiple answers for a single offer), consider the case where browser A requests video with B. When the A side that sent the initial OFFER gets an ANSWER that rejects the video, it may very well present a UI indication that there is no media. Five seconds later when browser B sends an OFFER requesting video, browser A may present a UI element

that asks is OK to do the video that was just rejected. This results in a bad user experience and in the extreme can result in both sides always rejecting the other side's OFFER of video, then waiting for the user to authorize video that results in a new OFFER that is always rejected.

It easier to be able to indicate that OFFER resulted in one valid ANSWER, but that the OFFER needs to be held open as other valid ANSWERS which would replace the current one. This stops the other side from generating new a new OFFER while this is taking place. This is also needed to support a SIP gateway doing early media.

5.3.3. OK

The OK message is used by the receiver of an ANSWER message to indicate that it has received the ANSWER message. It has no contents itself and is merely used to stop the retransmissions of the ANSWER.

5.3.4. ERROR

The ERROR message is used to indicate that there has been an error. The contents and semantics of this message are defined in Section 5.6.

5.4. Changing Media Parameters

Once a call has been set up, it is common to want to adjust the media parameters, e.g., to add video to an audio-only call. This is also done with the OFFER/ANSWER/OK sequence of messages, though the details are slightly different.

Either side may initiate a new OFFER/ANSWER exchange by sending an OFFER message. However, implementations MUST NOT attempt this for sessions which are still in active negotiation. Specifically, the offerer MUST NOT send a new OFFER until it has received the ANSWER, and the answerer MUST NOT send a new OFFER until it has received the OK indicating receipt of the ANSWER.

A new OFFER MUST contain a complete set of media parameters describing the proposed new media configuration as well as a full set of ICE parameters. The recipient of a new OFFER on a valid connection MUST respond with an appropriate ANSWER message. However that message MAY refuse to accept the proposed new configuration. If the session has been terminated in the meantime, then an Error: NOMATCH message SHOULD be sent.

5.4.1. Conflicting OFFERS (glare)

Because a change of media parameters may be initiated by either side, there is a potential for the change requests to occur simultaneously (i.e., "glare"). This document defines a glare handling procedure that results in immediate resolution of the glare condition allowing one OFFER message to continue to be processed while the other is terminated. It is defined in such a way that it can interwork with SIP's glare handling mechanism. However SIP's timer based mechanism aren't suitable for the ROAP as strict requirements on ROAP message transport between end-points are not possible and thus easily could result in an repeated glare situation.

To achieve immediate resolution each OFFER message includes a 32 unsigned integer value, the tie breaker, that is randomly generated for each new OFFER message an end-point issues. Whenever a end-point receives an OFFER message that has the same sequence number as an outstanding OFFER the end-point itself sent, a glare condition has arisen. In a glare condition the end-point compares the received OFFER's tiebreaker value with the tiebreaker value of the tiebreaker in the OFFER outstanding. The OFFER with the greatest numerical value wins and that OFFER is allowed to continue being processed. IF the received OFFER lost the tie breaking an Error:CONFLICT message is sent. If it is the outstanding OFFER that lost, the end-point can expect an Error:CONFLICT message to be eventually received. However, that OFFER can immediately be considered as terminated.

Some special considerations has been made in this glare handling for interworking well with SIP glare handling as currently specified. Thus it has the notion of a gateway that converts the ROAP message into SIP message. This process is discussed in more detail below after the basic rules are defined normatively.

A regular end-point SHALL generate a random 32-bit unsigned numerical value for each OFFER message. In the case the random value becomes 0 or 4,294,967,295 a new random value SHALL be generated until it is neither values. The values 0 and 4,294,967,295 MAY be assigned to ROAP messages generated by gateways to ensure efficient glare handling towards other systems.

An ROAP message end-point that has an outstanding OFFER, i.e. an OFFER where it has not yet received an ANSWER SHALL upon receiving an OFFER perform the following processing:

1 Check if the incomming OFFER has a answererSessionId, if not it is an initial offer. If the outstanding OFFER also is an intial OFFER there is an Error. If the outstanding OFFER is not an initial OFFER and the outstanding OFFER do have answererSessionId equal to the offererSessionId in the received message then the sequence numbers are checked. In case the incomming OFFER's sequence number is equal to the sequence number of the outstanding OFFER there is glare. If the sequence number is not the same and the sequence number of the incomming is larger than the outstanding OFFER's sequence number, then this message is out of order with an ANSWER to the out-standing message. If the sequence number of the incomming is lower than the outstanding, then this is a old request.

2 In case of glare, compare the tie-breaker values for each OFFER. The tie-breaker value that is greater than the other wins. The OFFER with the winning value is processed as if there was no glare. The OFFER with the losing value is terminated, see 3A or 3B. In case the tie-breaker values are equal the double-glare case in 3C is invoked.

3A The OFFER being terminated is the received one: The end-point SHALL send a Error:CONFLICT response message.

3B The OFFER being terminated is this end-points outstanding OFFER: The end-point knows the OFFER will be terminated and can expect an Error:CONFLICT response. The end-point can assume this termination and MAY issue a new OFFER as soon as possible after having concluded the transactions for the winning OFFER.

3C The two tie-breaker values where equal, in this case both OFFERs are terminated and a Error:DOUBLCONFLICT message is sent. Both of the Offerer SHOULD re-attempt their offers by generating new OFFER messages, these messages SHALL have new tie-breaker values and incremented sequence number. Also gateways SHOULD generate random values, as one reason for this double conflict is that two gateways have become interconnected and both selects either 0 or 4,294,967,295.

The following figure assumes the previous message flow has happened and media is flowing.

```
  participant OffererUA
  participant OffererJS
  participant AnswererJS
  participant AnswererUA

  note left of OffererJS: "Hi, Let's do video"
  note right of AnswererJS: "Sounds great"
  OffererJS->OffererUA: peer->addStream( new MediaStream() );
  OffererUA->OffererJS: sendSignalingChannel();
  AnswererJS->AnswererUA: peer->addStream( new MediaStream() );
  AnswererUA->AnswererJS: sendSignalingChannel();
  OffererJS->AnswererJS: {"type":"OFFER", tiebreaker="123", "sdp":"..."}
  AnswererJS->OffererJS: {"type":"OFFER", tiebreaker="456", "sdp":"..."}
  AnswererJS->AnswererUA: peer->processSignalingMessage();
  OffererJS->OffererUA: peer->processSignalingMessage();


  OffererUA->OffererJS: sendSignalingChannel();
  AnswererUA->AnswererJS: sendSignalingChannel();
  OffererJS->AnswererJS: {"type":"ERROR",error="conflict","sdp":"..."}
  AnswererJS->OffererJS: {"type":"ANSWER", "sdp":"..."}
  AnswererJS->AnswererUA: peer->processSignalingMessage();
  OffererJS->OffererUA:  peer->processSignalingMessage();

  OffererUA->OffererJS: sendSignalingChannel();
  OffererJS->AnswererJS: {"type":"OK"}
  AnswererJS->AnswererUA: peer->processSignalingMessage();
  AnswererUA->AnswererJS: onaddstream();


  AnswererUA->AnswererJS: sendSignalingChannel();
  AnswererJS->OffererJS: {"type":"OFFER", tiebreaker="789", "sdp":"..."}
  OffererJS->OffererUA: peer->processSignalingMessage();
  OffererUA->OffererJS: sendSignalingChannel();
  OffererJS->AnswererJS: {"type":"ANSWER", "sdp":"..."}
  AnswererJS->AnswererUA: peer->processSignalingMessage();
  AnswererUA->OffererUA: Both way Video
  AnswererUA->AnswererJS: sendSignalingChannel();
  AnswererJS->OffererJS: {"type":"OK"}
  OffererJS->OffererUA: peer->processSignalingMessage();
  OffererUA->OffererJS: onaddstream();
```

5.4.2. Premature OFFER

It is an error, though technically possible, for an agent to generate a second OFFER while it already has an unanswered OFFER pending. An agent which receives such an offer MUST respond with an Error:FAILED

message containing a "RetryAfter" attribute generated as a random value from 0 to 10 seconds.

5.5. Notification of Media Termination

The SHUTDOWN message is used to indicate the termination of an existing session. Either side may initiate a SHUTDOWN at any time during the session, including while the initial OFFER is outstanding (i.e., before an ANSWER has been sent/received.)
TODO - FIX NAMES

```
participant OffererUA
participant OffererJS
participant AnswererJS
participant AnswererUA

OffererJS->OffererUA: peer->close();
OffererUA->OffererJS: sendSignalingChannel();
OffererJS->AnswererJS: { "type":"SHUTDOWN" }
AnswererJS->AnswererUA: peer->processSignalingMessage();
AnswererUA->AnswererJS: onclose();

AnswererUA->AnswererJS: sendSignalingChannel();
AnswererJS->OffererJS: {"type":"OK"}
OffererJS->OffererUA: peer->processSignalingMessage();
OffererUA->OffererJS: onclose();
```

Upon receipt of a SHUTDOWN which corresponds to an existing session, an agent MUST immediately terminate the session and send an OK message. Subsequent messages directed to this session MUST result in an Error:NOMATCH message. Similarly, on receipt of the OK, the agent which sent the SHUTDOWN MUST terminate the session and SHOULD respond to future messages with Error:NOMATCH.

5.6. Errors

Errors are indicated by the messageType "ERROR". All errors MUST contain an "errorType" field indicating the type of error which occurred and echo the "seq" value (if any) and the session id values of the message which generated the error. The following sections describe each error type.

5.6.1. NOMATCH

An implementation which receives a message with either an unknown offererSessionId (for an OFFER) or an unknown offererSessionId/ answererSessionId pair SHOULD respond with a NOMATCH error.

5.6.2. TIMEOUT

The TIMEOUT error is used to indicate that the corresponding message required some processing which timed out. For instance, an agent which is a SIP gateway translates ROAP signaling messages into SIP messages. If those SIP messages time out, the gateway would generate a TIMEOUT error.

5.6.3. REFUSED

An agent which has received an initial OFFER MAY indicate its refusal of the media session by sending a REFUSED error. Note that this error is not required; an agent MAY simply drop the OFFER with no acknowledgement at all. However, agents which do not wish to accept subsequent OFFERS SHOULD [OPEN ISSUE: MUST?] send a REFUSED in order to avoid timeouts and confusion on the offerer side.

5.6.4. CONFLICT

The CONFLICT error is used to indicate that an agent has received an OFFER while it has its own OFFER outstanding. The offerer's behavior in response to this error is defined in Section 5.4.1.

5.6.5. DOUBLECONFLICT

The DOUBLECONFLICT error is used to indicate the tiebreaker values in CONFLICT were the same. See Section 5.4.1.

5.6.6. FAILED

FAILED is a catch-all error indicating that something went wrong while processing a message. A FAILED error MAY contain a "retryAfter" field, which indicates the time (in seconds) after which the message MAY be retried (though retries are OPTIONAL).

6. Security Considerations

TBD

7. Companion APIs

Note: This section may need to move to the requirements draft[I-D.ietf-rtcweb-use-cases-and-requirements] but for now it is convenient to put it here just to help see how all the pieces fit together.

The offer / answer concepts in this draft are not enough to meet all the use cases of RTCWeb. They need to be combined with some additional functionality that the browser exposes to the JavaScript applications. This additional functionality loosely falls into three categories: capabilities, hints, and stats. The capabilities allow the JS application to find out what video codecs and capabilities a given browser supports before initiating a media session. The hints provide a way for the JS application to provide useful information to the browser about how the media will be used so that the browser can negotiate appropriate codecs and modes. Stats provides statistics about what the current media sessions. The capabilities, hints, and stats do not need to be communicated between the two browsers, so they are not specified in this draft. However, this drafts assumes the existence of API so that these three can be used to build complete systems. Some of the assumptions about these APIs are described in the following sections.

7.1. Capabilities

The APIs need to provide a way to find out the capabilities as defined in section 9 of RFC 3264. This allows the JS to find out the codecs that the browser supports.

7.2. Hints

When creating a new PeerConenction in a browser, the application needs to be able to provide optional hints to the browser about preferences for the media to be negotiated. These include:

1. Whether the session has audio, video, or both;
2. Whether the audio is spoken voice or music;
3. Preferred video resolution and frame rate (perhaps these just come from the MediaTrack objects);
4. Whether the video should prefer temporal or spatial fidelity;
5. <add more here>

The JS applications should also be able to update and change these hints mid-session. Some types of hint changes may simply impact the parameter on various codecs and require no signalling to the other end of the media stream. Other types of hint changes may cause a new offer answer exchange.

7.3. Stats

Several parts of the media session create statistics that are important to some applications. APIs should provide the JS applications with information on the following statistics:

1. Total IP data rate for the session;
2. ICE statistics including current candidates, active pairs, RTT;
3. RTP statistics including codecs selected, parameters, and bit rates;
4. RTCP statistics including packet loss rate; and
5. SRTP statistics.

8. Relationship with SIP & Jingle

The SIP [RFC3261] specifies an application protocol that provides a complete solution for setting up and managing communications on the Internet. It combines both "call processing" functions - identity and name spaces, call routing, user search, call features, authentication, and so on - as well as media processing through its transport of SDP and support for the offer/answer model.

In a web context, application processing can be done through proprietary logic implemented in Javascript/HTML, along with proprietary logic implemented in the web server, and proprietary messaging transported through HTTP and WebSockets. One of the advantages of the web is to allow a rich set of applications to be built without changing the browser. Although application processing and be done in JavaScript and the web servers, we do require raw media control in the browser. ROAP basically extracts the offer/answer media control processing used in SIP, and puts it into an protocol that can operate independently of SIP itself.

The information contained in ROAP messages corresponds closely to the offer/answer information carried by complete solutions such as SIP and Jingle, so it is straightforward to build gateways to and from ROAP. These gateways need only translate the signaling, while allowing end-to-end media without the need for media relays (except, of course, for NAT traversal.) In the case of SIP, which uses SDP directly, such gateways would translate between SIP and ROAP, while transporting SDP end-to-end. In the case of Jingle [XEP-0166], it would also be necessary to translate between SDP and the Jingle offer/answer format; [XEP-0167] describes such a mapping.

9. IANA Considerations

This document requires no actions from IANA.

10. Acknowledgments

The text for the glare resoltuion section was provided by Magnus

Westerlund. Many thanks for comment, ideas, and text from Eric Rescorla, Harald Alvestrand, Magnus Westerlund, Ted Hardie, and Stefan Hakansson.

11. Open Issues

How to negotiate support for enhancements to this JSON message. (consider supported / required)

Common way to indicate destination in offer going to a signalling gateway.

Need to generate proper ASCII art version of message flows.

12. References

12.1. Normative References

[RFC4627] Crockford, D., "The application/json Media Type for JavaScript Object Notation (JSON)", RFC 4627, July 2006.

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

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

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

12.2. Informative References

[RFC3261] Rosenberg, J., Schulzrinne, H., Camarillo, G., Johnston, A., Peterson, J., Sparks, R., Handley, M., and E. Schooler, "SIP: Session Initiation Protocol", RFC 3261, June 2002.

[XEP-0166] Ludwig, S., Beda, J., Saint-Andre, P., McQueen, R., Egan, S., and J. Hildebrand, "Jingle", XSF XEP 0166, December 2009.

[XEP-0167] Ludwig, S., Saint-Andre, P., Egan, S., McQueen, R., and D. Cionoiu, "Jingle RTP Sessions", XSF XEP 0167,

December 2008.

[RFC5245] Rosenberg, J., "Interactive Connectivity Establishment (ICE): A Protocol for Network Address Translator (NAT) Traversal for Offer/Answer Protocols", RFC 5245, April 2010.

[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

[I-D.ietf-rtcweb-use-cases-and-requirements] Holmberg, C., Hakansson, S., and G. Eriksson, "Web Real-Time Communication Use-cases and Requirements", draft-ietf-rtcweb-use-cases-and-requirements-06 (work in progress), October 2011.

Authors' Addresses

Cullen Jennings
Cisco
170 West Tasman Drive
San Jose, CA 95134
USA

Phone: +1 408 421-9990
Email: fluffy@cisco.com

Jonathan Rosenberg
jdrosen.net

Email: jdrosen@jdrosen.net
URI: http://www.jdrosen.net

Justin Uberti
Google, Inc.

Randell Jesup
Mozilla

Email: randell-ietf@jesup.org

RTCWeb Working Group
Internet-Draft
Intended status: Informational
Expires: May 3, 2012

R. Jesup
Mozilla
S. Loreto
Ericsson
M. Tuexen
Muenster University of Applied Sciences

October 31, 2011

RTCWeb Datagram Connection
draft-jesup-rtcweb-data-01.txt

Abstract

This document investigates the possibilities for designing a generic transport service that allows Web Browser to exchange generic data in a peer to peer way. Several, already standardized by IETF, transport protocols and their properties are investigated in order to identify the most appropriate one.

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 May 3, 2012.

Copyright Notice

Table of Contents

1. Introduction

The issue of how best to handle non-media data types in the context of RTCWEB is still under discussion in the mailing list; there have been several proposals on how to address this problem, but there is not yet a clear consensus on the actual solution.

However it seems to be a general agreement that for NAT traversal purpose it has to be:

FOO/UDP/IP

or most likely:

FOO/DTLS/UDP/IP (for confidentiality, source authenticated, integrity protected transfers)

where FOO is a protocol that is supposed to provide congestion control and possible some type of framing or stream concept.

Moreover there has been a clear interest for both an unreliable and a reliable peer to peer datagram based channel.

This document provides Requirement and use cases for both unreliable and reliable peer to peer datagram base channel, provide an overview of the pro and cons of the different proposed solutions, and finally analyze in more detail the SCTP based solution.

2. Requirements

This section lists the requirements for P2P data connections between two browsers.

Req. 1 Multiple simultaneous datagram streams must be supported. Note that there may 0 or more media streams in parallel with the data streams, and the number and state (active/inactive) of the media streams may change at any time.

Req. 2 Both reliable and unreliable datagram streams must be supported.

Req. 3 Data streams must be congestion controlled; either individually, as a class, or in conjunction with the media streams, to ensure that datagram exchanges don’t cause congestion problems for the media streams, and that the rtcweb PeerConnection as a whole is fair with competing streams such as TCP.

Req. 4 The application should be able to provide guidance as to the relative priority of each datagram stream relative to each other, and relative to the media streams. [TBD: how this is encoded and what the impact of this is.] This will interact with the congestion control.

Req. 5 Datagram streams must be encrypted; allowing for confidentiality, integrity and source authentication. See the security spec [xxx] for detailed info.

Req. 6 Consent and NAT traversal mechanism: These are handled by the PeerConnection's ICE [RFC5245] connectivity checks and optional TURN servers.

Req. 7 Data streams MUST provide message fragmentation support such that IP-layer fragmentation does not occur no matter how large a message/buffer the Javascript application passes down to the Browser to be sent out.

Rec. 8 The data stream transport protocol MUST NOT encode local IP addresses inside its protocol fields; doing so reveals potentially private information, and leads to failure if the address is depended upon.

Req. 9 The data stream protocol SHOULD support unbounded-length "messages" (i.e., a virtual socket stream) at the application layer, for such things as image-file-transfer; or else it MUST support at least a maximum application-layer message size of 4GB.

Req. 10 The data stream packet format/encoding MUST be such that it is impossible for a malicious Javascript to generate an application message crafted such that it could be interpreted as a native protocol over UDP - such as UPnP, RTP, SNMP, STUN, etc.

Req. 10 The data stream transport protocol MUST start with the assumption of a PMTU of 1280 [*** need justification ***] bytes until measured otherwise.

Req. 11 The data stream transport protocol MUST NOT rely on ICMP being generated or being passed back, such as for PMTU discovery.

3. Use cases.

3.1. Use cases for unreliable datagram based channel

U-C 1 A real-time game where position and object state information is sent via one or more unreliable data channels. Note that at any time there may be no media channels, or all media channels may be inactive.

U-C 2 Non-critical state updates about a user in a video chat or conference, such as Mute state.

3.2. Use cases for reliable channels (datagram or stream).

Note that either reliable datagrams or streams are possible; reliable streams would be fairly simple to layer on top of SCTP reliable datagrams with in-order delivery.

U-C 3 A real-time game where critical state information needs to be transferred, such as control information. Typically this would be datagrams. Such a game may have no media channels, or they may be inactive at any given time, or may only be added due to in-game actions.

U-C 4 Non-realtime file transfers between people chatting. This could be datagrams or streaming; streaming might be an easier fit

U-C 5 Realtime text chat while talking with an individual or with multiple people in a conference. Typically this would be datagrams.

U-C 6 Renegotiation of the set of media streams in the PeerConnection. Typically this would be datagrams

4. Protocol alternatives

4.1. Datagrams over DTLS over DCCP over UDP.

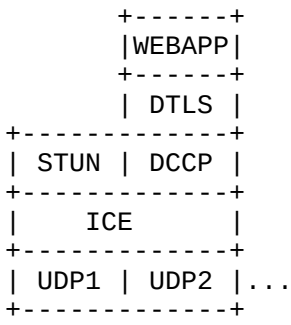


Figure 1: stack diagram

DCCP [RFC4340] adds to a UDP-like foundation the minimum mechanisms necessary to support congestion control. It is a unicast, connection-oriented protocol with bidirectional data flow.

The main downside of DCCP is the uncertainty of the DCCP implementations. Moreover DCCP only meets the requirements for the unreliable data channel, so in order to satisfy the reliable data channel requirements there is a need to build a new mechanism on top of DCCP or use a different protocol for it.

The main advantage of DCCP is that the Congestion Control (CC) methods are modularly separated from its core, that allows each application to choose a different congestion control methods it prefers. Each congestion control method is denoted by unique ID (CCID): a number from 0-255. CCIDs 2, 3 and 4 are current defined; CCIDs 0, 1, and 5-255 are reserved. The end-points negotiate their CCIDs during connection initiation and achieve agreement through the exchange of feature negotiation options in DCCP headers.

CCID 2 [RFC4341] denotes Additive Increase, Multiplicative Decrease congestion control with feature modelled directly on TCP. It achieves the maximum bandwidth over the long term consistent with the use of end-to-end congestion control but reduces the congestion window by half upon congestion detection. Applications that prefer a large amount of bandwidth as feasible over the longer terms should use CCID2.

CCID 3 [RFC4342], TCP friendly rate control mechanism(TFRC), denotes an equation-based and rate controlled congestion control mechanisms designed to be reasonably fair when competing for bandwidth with TCP

like flows. It shows much lower variation of throughput over time compared with TCP that makes CCID 3 more suitable than CCID 2 for applications such as streaming media content that prefers to minimize abrupt changes in the sending rate.

CCID 4 [RFC5622] is a modified version of CCID 3 and designed for applications that use a small fixed segment size, or change their sending rate by varying the segment size. It denotes TFRC-SP (TCP friendly rate control for small packets)

Both CCID 3 and 4 uses the TCP throughput equation for CC; the former is based on the calculation of loss event rate but the later also include nominal packet size of 1460 bytes, a round trip estimate in TCP throughput calculation. In contrast to CCID 3, the CCID 4 sender imposes a minimum interval of 10 ms between data packets regardless of the allowed transmit rate.

4.2. Datagrams over SCTP over DTLS over UDP.

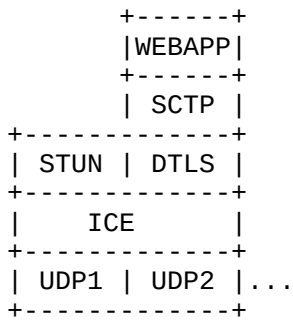


Figure 2: stack diagram

An SCTP [RFC4960] based solution will provide several interesting features among the others: Multistreaming, Ordered and Unordered delivery, Reliability and partial-Reliability [RFC3758], and Dynamic Address Reconfiguration [RFC5061].

Moreover SCTP provides the possibility to transport different "protocols" over multiple streams and associations using the ppid (Payload Protocol Identifier). An application can set a different PPID with each send call. This allows the receiving application to look at this information (as well as the stream id/seq) on receiving to know what type of protocol the data payload has.

The SCTP feature so seems to satisfy all the requirements for both

the unreliable and the reliable scenario.

There are SCTP implementations for all the different OS: Linux (mainline kernel 2.6.36), FreeBSD (release kernel 8.2), Mac OS X, Windows (SctpDrv4) and Solaris (OpenSolaris 2009.06), as well as a user-land SCTP implementation based on the FreeBSD implementation).

The SCTP solution is analyzed in more detail in Section 5.

4.3. A new protocol on top of UDP.

This option requires to at least build a congestion control (CC) mechanism on top of the plain UDP.

In designing it we have to follow carefully the guidelines provided in [RFC5405].

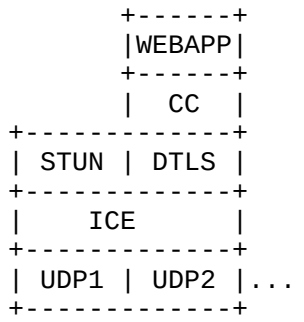


Figure 3: stack diagram UDP

4.3.1. TCP over DTLS over UDP.

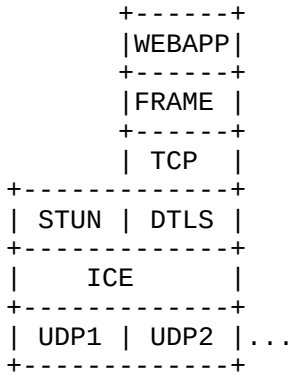


Figure 4: stack diagram

Layering TCP atop DTLS or UDP is an approach that has been suggested several times in the past, including TCP-over-UDP [I-D.baset-tsvwg-tcp-over-udp] and UDP-Encapsulated Transport Protocols [I-D.denis-udp-transport].

A similar mechanism has also been used for Google Talk's peer-to-peer file transfer protocol, implemented in the libjingle library as "PseudoTcp" [http://code.google.com/p/libjingle/source/browse/trunk/talk/p2p/base/pseudotcp.cc]. In this implementation, a lightweight userspace TCP stack has been developed with support for a fairly minimal set of TCP options, namely delayed acknowledgements, Nagle, fast retransmit, fast recovery (NewReno), timestamps, and window scaling. Some features have been removed, such as urgent data. And as in the aforementioned drafts, the TCP header has been tweaked slightly to remove fields redundant with the UDP header, namely source/destination port and checksum.

The advantage of this approach is clear; TCP is well-known and understood, and its congestion control is by definition TCP-fair. User-space implementations of TCP exist, e.g. as PseudoTcp, which has considerable deployment experience. It is also possible to support multiplexing of datagram flows with this approach, either by adding a stream identifier to the TCP header (in place of the port numbers, perhaps), or doing the same in a higher-level framing layer.

This approach has some disadvantages as well; since TCP is a stream,

rather than datagram-oriented protocol, some framing needs to be added on top of TCP to provide the necessary datagram interface to the calling API. In addition, TCP only provides reliable delivery; there is no provision for a unreliable channel. This deficiency could be remedied by providing a separate protocol for unreliable channels.

Such a protocol could be a lightweight datagram protocol that implements the sequence number and other fields necessary to run TFRC-SP.

4.4. A RTP compatible protocol.

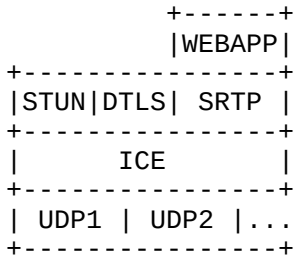


Figure 5: stack diagram

When sending RTP with DTLS, rather than encapsulating RTP in DTLS, SRTP keys are extracted from the DTLS key material, allowing use of SRTP's more efficient format. This same approach could be extended for sending of application data as a RTP payload.

The benefits of this approach are centered around the ability to reuse the existing mechanisms for audio/video data for application data, and the resultant simplification that occurs. If everything ends up RTP, and RTP provides information about loss and timing, we can have common encryption, congestion control, and multiplexing mechanisms for all types of data. For example, we could use RTP SSRC to demux different application data streams, and RTCP NACK to faciliate reliable delivery.

On the other hand, RTP has a number of semantics associated with it that aren't necessarily a good fit for arbitrary application data. While the RTP timestamp, sequence number and SSRC fields are meaningful, there are a number of other header fields that may not make sense, and the applicability of RTP's notions of timing, media playout, and control feedback is unclear.

5. Datagrams over SCTP over DTLS over UDP.

5.1. User Space vs Kernel implementation.

Even kernel implementation of SCTP are already available for all the different platforms (see Section 4.2), there are compelling reasons that incline towards for a SCTP stack that works well in user land.

The main reason is deployability.

There are many applications today that are expected to run on a wide range of old and new operating systems. Web browsers are an excellent example. They support operating systems 10 years old or more, they run on mobile and desktop operating systems, and they are highly portable to new operating systems. This is achieved by having a fairly narrow portability layer to minimize what needs to be supported on old operating systems and ported to new ones. This creates a strong desire to implemented as much functionality as possible inside the application instead of relying on the operating system for it.

This leads to a situation where there is a desire for the SCTP stack to be implemented in the user space instead of the kernel space. For many applications that require support of operating systems without SCTP (insert whatever stack order is - UDP, DTLS - whatever), there is no way to deploy this unless it can be implemented in the application. The traditional reasons for kernel implementations, such as mux of many application using transport port numbers, does not particularly apply here where that level of multiplexing between application was provided by the underling UDP that is tunneled over. The requirement is:

It MUST be possible to implement the SCTP and DTLS portion of the stack in the user application space.

5.2. The envisioned usage of SCTP in the RTCWeb context

The appealing features of SCTP in the RTCWeb context are:

- the congestion control which is TCP friendly.
- the congestion control is modifiable for integration with media stream congestion control.
- support for multiple channels with different characteristics.
- support for out-of-order delivery.

- support for large datagrams and PMTU-discovery and fragmentation.
- the reliable or partial reliability support.

Multi streaming is probably also of interest.

Multihoming will not be used in this scenario. The SCTP layer would simply act as if it were running on a single-homed host, since that is the abstraction that the lower layers (e.g. UDP) would expose.

5.3. SCTP/DTLS/UDP vs DTLS/SCTP/UDP

The two alternatives being discussed in this subsection are shown in the following Figure 6.

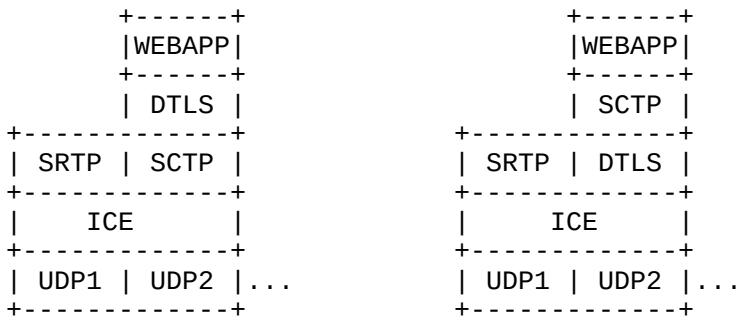


Figure 6: Two variants of SCTP and DTLS usage

The UDP encapsulation of SCTP used in the protocol stack shown on the left hand side of Figure 6 is specified in [I-D.ietf-tsvwg-sctp-udp-encaps] and the usage of DTLS over SCTP is specified in [RFC6083]. Using the UDP encapsulation of SCTP allows SCTP implementations to run in user-land without any special privileges, but still allows the support of SCTP kernel implementations. This also requires no SCTP specific support in middleboxes like firewalls and NATs. Multihoming and failover support is implemented via the ICE layer, and SCTP associations are single-homed at the SRTP layer. The SCTP payload is protected by DTLS, which provides confidentiality, integrity and source authentication. SCTP-AUTH as specified in [RFC4895] is used to provide integrity of SCTP control information related to the user messages like the SCTP stream identifier. Please note that the SCTP control information (like the SCTP stream identifier) is sent unencrypted.

Considering the protocol stack on the right hand side of Figure 6, the usage of DTLS over UDP is specified in

[I-D.ietf-tls-rfc4347-bis]. Using SCTP on top of DTLS is currently unspecified. Since DTLS is typically implemented in user-land, an SCTP user-land implementation has also to be used. Kernel SCTP implementations can't be used. When using DTLS as the lower layer, only single homed SCTP associations can be used, since DTLS does not expose any any address management to its upper layer. DTLS implementations used for this stack must support controlling fields of the IP layer like the DF-bit in case of IPv4 and the DSCP field. This is required for performing path MTU discovery. The DTLS implementation must also support sending user messages exceeding the path MTU. When supporting multiple SCTP associations over a single DTLS connection, incoming ICMP or ICMPv6 messages can't be processed by the SCTP layer, since there is no way to identify the corresponding association. Therefore the number of SCTP associations should be limited to one or ICMP and ICMPv6 messages should be ignored. In general, the lower layer interface of an SCTP implementation has to be adopted to address the differences between IPv4 or IPv6 (being connection-less) or DTLS (being connection-oriented). When this stack is used, DTLS protects the complete SCTP packet, so it provides confidentiality, integrity and source authentication of the complete SCTP packet.

It should be noted that both stack alternatives support the usage of multiple SCTP streams. A user message can be sent ordered or unordered and, if the SCTP implementations support [RFC3758], with partial reliability. When using partial reliability, it might make sense to use a policy limiting the number of retransmissions. Limiting the number of retransmissions to zero provides a UDP like service where each user messages is sent exactly once.

SCTP provides congestion control on a per-association base. This means that all SCTP streams within a single SCTP association share the same congestion window. Traffic not being sent over SCTP is not covered by the SCTP congestion control.

6. Message Format.

TBD if we need also to design a framing or not.

At time of writing nobody has identified a real need for masking of UDP, and DTLS is a much-preferred solution for encryption/ authentication.

More SCTP already provides sequence number information (e.g. stream sequence numbers). However there could be a need to support different kind of message (link in WebSocket where there is support to distinguish between Unicode text and binary frames), since for

SCTP they are all user message. In the case there is this need a possibility is to map types to streams.

7. Security Considerations

To be done.

8. IANA Considerations

This document does not require any actions by the IANA.

9. Acknowledgments

Many thanks for comments, ideas, and text from Cullen Jennings, Eric Rescorla, Randall Stewart and Justin Uberti.

10. Informational References

[RFC3758] Stewart, R., Ramalho, M., Xie, Q., Tuexen, M., and P. Conrad, "Stream Control Transmission Protocol (SCTP) Partial Reliability Extension", RFC 3758, May 2004.

[RFC4340] Kohler, E., Handley, M., and S. Floyd, "Datagram Congestion Control Protocol (DCCP)", RFC 4340, March 2006.

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

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

[RFC5622] Floyd, S. and E. Kohler, "Profile for Datagram Congestion Control Protocol (DCCP) Congestion ID 4: TCP-Friendly Rate Control for Small Packets (TFRC-SP)", RFC 5622, August 2009.

[RFC4895] Tuexen, M., Stewart, R., Lei, P., and E. Rescorla, "Authenticated Chunks for the Stream Control Transmission Protocol (SCTP)", RFC 4895, August 2007.

[RFC4960] Stewart, R., "Stream Control Transmission Protocol",

RFC 4960, September 2007.

[RFC5061] Stewart, R., Xie, Q., Tuexen, M., Maruyama, S., and M. Kozuka, "Stream Control Transmission Protocol (SCTP) Dynamic Address Reconfiguration", RFC 5061, September 2007.

[RFC5245] Rosenberg, J., "Interactive Connectivity Establishment (ICE): A Protocol for Network Address Translator (NAT) Traversal for Offer/Answer Protocols", RFC 5245, April 2010.

[RFC5389] Rosenberg, J., Mahy, R., Matthews, P., and D. Wing, "Session Traversal Utilities for NAT (STUN)", RFC 5389, October 2008.

[RFC5405] Eggert, L. and G. Fairhurst, "Unicast UDP Usage Guidelines for Application Designers", BCP 145, RFC 5405, November 2008.

[RFC6083] Tuexen, M., Seggelmann, R., and E. Rescorla, "Datagram Transport Layer Security (DTLS) for Stream Control Transmission Protocol (SCTP)", RFC 6083, January 2011.

[I-D.ietf-tls-rfc4347-bis] Rescorla, E. and N. Modadugu, "Datagram Transport Layer Security version 1.2", draft-ietf-tls-rfc4347-bis-06 (work in progress), July 2011.

[I-D.ietf-tsvwg-sctp-udp-encaps] Tuexen, M. and R. Stewart, "UDP Encapsulation of SCTP Packets", draft-ietf-tsvwg-sctp-udp-encaps-01 (work in progress), October 2011.

[I-D.baset-tsvwg-tcp-over-udp] Baset, S. and H. Schulzrinne, "TCP-over-UDP", draft-baset-tsvwg-tcp-over-udp-01 (work in progress), June 2009.

[I-D.denis-udp-transport] Denis-Courmont, R., "UDP-Encapsulated Transport Protocols", draft-denis-udp-transport-00 (work in progress), July 2008.

Authors' Addresses

Randell Jesup
Mozilla

Email: randell-ietf@jesup.org

Salvatore Loreto
Ericsson
Hirsalantie 11
Jorvas 02420
Finland

Email: salvatore.loreto@ericsson.com

Michael Tuexen
Muenster University of Applied Sciences
Stegerwaldstrasse 39
Steinfurt 48565
Germany

Email: tuexen@fh-muenster.de

RTCWEB
Internet-Draft
Intended status: Standards Track
Expires: April 26, 2012

J. Lennox
Vidyo
J. Rosenberg
Skype

October 24, 2011

Multiplexing Multiple Media Types In a Single Real-Time Transport Protocol (RTP) Session
draft-lennox-rtcweb-rtp-media-type-mux-00

Abstract

This document describes mechanisms and recommended practice for transmitting media streams of multiple media types (e.g., audio and video) over a single Real-Time Transport Protocol (RTP) session, primarily for the use of Real-Time Communication for the Web (rtcweb).

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 April 26, 2012.

Table of Contents

1. Introduction

Classically, multimedia sessions using the Real-Time Transport Protocol (RTP) [RFC3550] have transported different media types (most commonly, audio and video) in different RTP sessions, each with in own transport flow. At the time RTP was designed, this was a reasonable design decision, reducing system variability and adding flexibility ([RFC3550] discusses the motivation for this design decision in section 5.2).

However, the de facto architecture of the Internet has changed substantially since RTP was originally designed, nearly twenty years ago. In particular, Network Address Translators (NATs) and firewalls are now ubiquitous, and IPv4 address space scarcity is becoming more severe. As a consequence, the network resources used up by an application, and its probability of failure, are directly proportional to the number of distinct transport flows it uses.

Furthermore, applications have developed mechanisms (notably Interactive Connectivity Establishment (ICE) [RFC5245]) to traverse NATs and firewalls. The time such mechanisms need to perform the traversal process is proportional to the number of distinct transport flows in use.

As a result, in the modern Internet, it is advisable and useful to revisit the transport-layer separation of media in a multimedia session. Fortunately, the architecture of RTP allows this to be done in a straightforward and natural way: by placing multiple sources of different media types in the same RTP session.

Since this is architecturally somewhat different from existing RTP deployments, however, this decision has some consequences that may be non-obvious. Furthermore, it is somewhat complex to negotiate such flows in signaling protocols that assumed the older architecture, most notably the Session Description Protocol (SDP) [RFC4566]. The rest of this document discusses these issues.

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] and indicate requirement levels for compliant implementations.

3. Transmitting multiple types of media in a single RTP session

RTP [RFC3550] supports the notion of multiple sources within a session. Historically, this was typically used for distinct users within a group to send media of the same type. Each source has its own synchronization source (SSRC) value and has a distinct sequence number and timestamp space. This document specifies that this same mechanism is used to allow sources of multiple media types in the same RTP session, even if they come from the same user. For example, in a call containing audio and video between two users, each sending a single audio and a single video source, there would be a single RTP session containing two sources (one audio, one video) from each user, for a total of four sources (and thus four SSRC values) within the RTP session.

Transmitting multiple types of media in a single RTP [RFC3550] session is done using the same RTP mechanisms as are used to transmit multiple sources of the same media type on a session. Notably:

- Each stream (of every media type) is a distinct source (distinct stream of consecutive packets to be sent to a decoder) and is given a distinct synchronization source ID (SSRC), and has its own distinct timestamp and sequence number space.
- Every media type (full media type and subtype, e.g. video/h264 or audio/pcmu) has a distinct payload type value. The same payload type value mappings apply across all sources in the session.
- RTP SSRCs, initial sequence numbers, and initial timestamps are chosen at random, independently for each source (of each media type).
- RTCP bandwidth is five percent of the total RTP session bandwidth.
- RTP session bandwidth and RTCP bandwidth are divided among all the sources in the session.
- RTCP sender report (SR) or receiver report (RR) packets, and source description (SDES) packets, are sent periodically for every source in the session.

In other words, no special RTP mechanisms are specifically needed for senders of multiplexed media. The only constraint is that senders sources MUST NOT change the top-level media type (e.g. audio or video) of a given source. (It remains valid to change a source’s subtype, e.g. switching between audio/pcmu and audio/g729.)

For a receiver, the primary complexity of multiplexing is knowing how to process a received source. Without multiplexing, all sources in an RTP session can (in theory) be processed the same manner; e.g., all audio sources can be fed to an audio mixer, and all video sources displayed on a screen. With multiplexing, however, receivers must apply additional knowledge.

If the streams being multiplexed are simply audio and video, this processing can decision can be made based simply on a source's payload type. For more complex situations (for example, simultaneous live-video and shared-application sources, both sent as video), signaling-level descriptions of sources would be needed, using a mechanism such as SDP Source Descriptions [RFC5576].

Additionally, due to the large difference in typical bitrate between different media (video can easily use a bit rate an order of magnitude or more larger than audio), some complications arise with RTCP timing. Because RTCP bandwidth is shared evenly among all sources in a session, the RTCP for an audio source can end up being sent significantly more frequently than it would in a non-multiplexed session. (The RTCP for video will, correspondingly, be sent slightly less frequently; this is not nearly as serious an issue.)

For RTP sessions that use RTP's recommended minimum fixed timing interval of 5 seconds, this problem is not likely to arise, as most sessions' bandwidth is not so low that RTCP timing exceeds this limit. The RTP/AVP [RFC3551] or RTP/SAVP [RFC3711] profiles use this minimum interval by default, and do not have a mechanism in SDP to negotiate an alternate interval.

For sessions using the RTP/AVPF [RFC4585] and RTP/SAVPF [RFC5124] profiles, however, endpoints SHOULD set the minimum RTCP regular reporting interval trr-int to 5000 (5 seconds), unless they explicitly need it to be lower. This minimizes the excessive RTCP bandwidth consumption, as well as aiding compatibility with AVP endpoints. Since this value only affects regular RTCP reports, not RTCP feedback, this does not prevent AVPF feedback messages from being sent as needed.

3.1. Optimizations

For multiple sources in the same session, several optimizations are possible. (Most of these optimizations also apply to multiple sources of the same type in a session.) In all cases, endpoints MUST be prepared for their peers to be using these optimizations.

An endpoint sending multiple sources MAY, as needed, reallocate media bandwidth among the RTP sources it is sending. This includes adding or removing sources as more or less bandwidth becomes available.

An endpoint MAY choose to send multiple sources' RTCP messages in a single compound RTCP packet (though such compound packets SHOULD NOT exceed the path MTU, if avoidable and if it is known). This will reduce the average compound RTCP packet size, and thus increase the frequency with which RTCP messages can be sent. Regular (non-

feedback) RTCP compound packets MUST still begin with an SR or RR packet, but otherwise may contain RTCP packets in any order. Receivers MUST be prepared to receive such compound packets.

An endpoint SHOULD NOT send reception reports from one of its own sources about another one ("cross-reports"). Such reports are useless (they would always indicate zero loss and jitter) and use up bandwidth that coud more profitably be used to send information about remote sources. Endpoints receiving reception reports MUST be prepared that their peers might not be sending reception reports about their own sources. (A naive RTCP monitor might think that there is a network disconnection between these sources; however, architecturally it is very unclear if such monitors actually exist, or would care about a disconnection of this sort.)

Similarly, an endpoint sending multiple sources SHOULD NOT send reception reports about a remote source from more than one of its local sources. Instead, it SHOULD pick one of its local sources as the "reporting" source for each remote source, which sends full report blocks; all its other sources SHOULD be treated as if they were disconnected, and never saw that remote source. An endpoint MAY choose different local sources as the reporting source for different remote sources (for example, it could choose to send reports about remote audio sources from its local audio source, and reports about remote video sources from its local video source), or it MAY choose a single local source for all its reports. If the reporting source leaves the session (sends BYE), another reporting source MUST be chosen. This "reporting" source SHOULD also be the source for any AVPF feedback messages about its remote sources, as well. Endpoints interpreting reception reports MUST be prepared to receive RTCP SR or RR messages where only one remote source is reporting about its sources.

4. Backward compatibility

In some circumstances, the offerer in an offer/answer exchange [RFC3264] will not know whether the peer which will receive its offer supports media type multiplexing.

In scenarios where endpoints can rely on their peers supporting Interactive Connectivity Establishment (ICE) [RFC5245], even if they might not support multiplexing, this should not be a problem. An endpoint could construct a list of ICE candidates for its single session, and then offer that list, for backward compatibility, toward each of the peers; it would disambiguate the flows based on the ufrag fields in the received ICE connectivity checks. (This would result in the chosen ICE candidates participating in multiple RTP sessions,

in much the same manner as following a forked SIP offer.) For RTCWeb, it is currently anticipated that ICE will be required in all cases, for consent verification.

The more difficult case is if an offerer cannot reply on its potential peers supporting any features beyond baseline RTP (i.e., neither ICE nor multiplexing). In this case, it would either need to be prepared to use only a single media type (e.g., audio) with such a peer, or else will need to do the pre-offer steps to set up all the non-multiplexed sessions. Notably, this would include opening local ports, and doing ICE address gathering (collecting candidate addresses from STUN and/or TURN servers) for each session, even if it is anticipated that in most cases backward compatibility is not going to be necessary.

If the signaling protocol in use supports sending additional ICE candidates for an ongoing ICE exchange, or updating the destination of a non-ICE RTP session, it is instead possible for an offerer to do such gathering lazily, e.g. opening only local host candidates for the non-default RTP sessions, and gathering and offering additional candidates or public relay addresses once it becomes clear that they are needed. (With SIP, sending updated candidates or RTP destinations prior to the call being answered is possible only if both peers support the SIP 100rel feature [RFC3262], i.e. PRACK and UPDATE; otherwise, the initial offer cannot be updated until after the 200 OK response to the initial INVITE.)

5. Signaling

There is a need to signal multiplexed media in the Session Description Protocol (SDP) [RFC4566] -- for inter-domain federation in the case of RTCWeb, as well as for "pure" SIP endpoints that also want to use media-multiplexed sessions.

To signal multiplexed sessions, two approaches seem to present themselves: either using the SDP grouping framework [RFC5888], as in [I-D.holmberg-mmusic-sdp-bundle-negotiation], or directly representing the multiplexed sessions in SDP.

Directly encoded multiplexed sessions would have some grammar issues in SDP, as the syntax of SDP mixes together top-level media types and transport information in the m= line, splitting media types to be partially described in the m= line and partially in the a=rtpmap attribute. New SDP attributes would need to be invented to describe the top-level media types for each source.

```
m=multiplex 49170 RTP/AVP 96 97
a=mediamap:96 video
a=rtpmap:96 H264/90000
a=mediamap:97 audio
a=rtpmap:97 pcmu/8000
```

Figure 1: Hypothetical syntax for describing multiplexed media lines in SDP

If single-pass backward compatibility is (ever) a goal, directly encoding multiplexed sessions in SDP m= lines becomes much more complex, as it would require SDP Capability Negotiation [RFC5939] in order to offer both the legacy and the multiplexed streams.

Using SDP grouping seems to rule out the possibility of non-backward-compatible multiplexed streams. Other than that, however, it seems that it would be the easier path to signal multiplexed sessions.

6. Protocols with SSRC semantics

There are some RTP protocols that impose semantics on SSRC values. Most notably, there are several protocols (for instance, FEC [RFC5109], layered codecs [RFC5583], or RTP retransmission [RFC4588]) have modes that require that sources in multiple RTP sessions have the same SSRC value.

When multiplexing, this is impossible. Fortunately, in each case, there are alternative ways to do this, by explicitly signaling RTP SSRC values [RFC5576]. Thus, when multiplexing, these modes need to be used instead.

It is unclear how to signal this in a backward-compatible way (falling back to session-multiplexed modes) if SDP grouping semantics are used to described multiplexed sources in SDP.

7. Security Considerations

The security considerations of a muxed stream appear to be similar to those of multiple sources of the same media type in an RTP session.

Notably, it is crucial that SSRC values are never used more than once with the same SRTP keys.

8. IANA Considerations

The IANA actions required depend on the decision about how muxed streams are signaled.

9. References

9.1. 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.

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

9.2. Informative References

[I-D.holmberg-mmusic-sdp-bundle-negotiation] Holmberg, C. and H. Alvestrand, "Multiplexing Negotiation Using Session Description Protocol (SDP) Port Numbers", draft-holmberg-mmusic-sdp-bundle-negotiation-00 (work in progress), October 2011.

[RFC3262] Rosenberg, J. and H. Schulzrinne, "Reliability of Provisional Responses in Session Initiation Protocol (SIP)", RFC 3262, June 2002.

[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.

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

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

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

[RFC5109] Li, A., "RTP Payload Format for Generic Forward Error Correction", RFC 5109, December 2007.

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

[RFC5245] Rosenberg, J., "Interactive Connectivity Establishment (ICE): A Protocol for Network Address Translator (NAT) Traversal for Offer/Answer Protocols", RFC 5245, April 2010.

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

[RFC5583] Schierl, T. and S. Wenger, "Signaling Media Decoding Dependency in the Session Description Protocol (SDP)", RFC 5583, July 2009.

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

[RFC5939] Andreasen, F., "Session Description Protocol (SDP) Capability Negotiation", RFC 5939, September 2010.

Authors' Addresses

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

Email: jonathan@vidyo.com

Jonathan Rosenberg
Skype

Email: jdrosen@skype.net
URI: http://www.jdrosen.net

Network Working Group
Internet-Draft
Intended status: BCP
Expires: April 26, 2012

M. Westerlund
B. Burman
Ericsson
C. Perkins
University of Glasgow
October 24, 2011

RTP Multiplexing Architecture
draft-westerlund-avtcore-multiplex-architecture-00

Abstract

RTP has always been a protocol that supports multiple participants each sending their own media streams in an RTP session. Thus relying on the three main multiplexing points in RTP; RTP session, SSRC and Payload Type for their various needs. However, most usages of RTP have been less complex often with a single SSRC in each direction, with a single RTP session per media type. But the more complex usages start to be more common and thus guidance on how to use RTP in various complex cases are needed. This document analyzes a number of cases and discusses the usage of the various multiplexing points and the need for functionality when defining RTP/RTCP extensions that utilize multiple RTP streams and multiple RTP sessions.

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 April 26, 2012.

Table of Contents

1. Introduction

This document focuses at issues of non-basic usage of RTP [RFC3550] where multiple media sources of the same media type are sent over RTP. Separation of different media types is another issue that will be discussed in this document. The intended uses include for example multiple sources from the same end-point, multiple streams from a single media source, multiple end-points each having a source, or an application that needs multiple representations (encodings) of a particular source. It will be shown that these uses are inter-related and need a common discussion to ensure consistency. In general, usage of the RTP session and media streams will be discussed in detail.

RTP is already designed for multiple participants in a communication session. This is not restricted to multicast, as many believe, but also provides functionality over unicast, using either multiple transport flows below RTP or a network node that re-distributes the RTP packets. The node can for example be a transport translator (relay) that forwards the packets unchanged, a translator performing media translation in addition to forwarding, or an RTP mixer that creates new conceptual sources from the received streams. In addition, multiple streams may occur when a single end-point have multiple media sources of the same media type, like multiple cameras or microphones that need to be sent simultaneously.

Historically, the most common RTP use cases have been point to point Voice over IP (VoIP) or streaming applications, commonly with no more than one media source per end-point and media type (typically audio and video). Even in conferencing applications, especially voice only, the conference focus or bridge has provided a single stream with a mix of the other participants to each participant. It is also common to have individual RTP sessions between each end-point and the RTP mixer.

SSRC is the RTP media stream identifier that helps to uniquely identify media sources in RTP sessions. Even though available SSRC space can theoretically handle more than 4 billion simultaneous sources, the perceived need for handling multiple SSRCs in implementations has been small. This has resulted in an installed legacy base that isn’t fully RTP specification compliant and will have different issues if they receive multiple SSRCs of media, either simultaneously or in sequence. These issues will manifest themselves in various ways, either by software crashes or simply in limited functionality, like only decoding and playing back the first or latest received SSRC and discarding media related to any other SSRCs.

There have also arisen various cases where multiple SSRCs are used to

represent different aspects of what is in fact a single underlying media source. A very basic case is RTP retransmission [RFC4588] which have one SSRC for the original stream, and a second SSRC either in the same session or in a different session to represent the retransmitted packets to ensure that the monitoring functions still function. Another use case is scalable encoding, such as the RTP payload format for Scalable Video Coding (SVC) [RFC6190], which has an operation mode named Multiple Session Transmission (MST) that uses one SSRC in each RTP session to send one or more scalability layers. A third example is simulcast where a single media source is encoded in different versions and transmitted to an RTP mixer that picks which version to actually distribute to a given receiver part of the RTP session.

This situation has created a need for a document that discusses the existing possibilities in the RTP protocol and how these can and should be used in applications. A new set of applications needing more advanced functionalities from RTP is also emerging on the market, such as telepresence and advanced video conferencing. Thus furthering the need for a more common understanding of how multiple streams are handled in RTP to ensure media plane interoperability.

The document starts with some definitions and then goes into the existing RTP functionalities around multiplexing. Both the desired behavior and the implications of a particular behavior depend on which topologies are used, which requires some consideration. This is followed by a discussion of some choices in multiplexing behavior and their impacts. Finally, some recommendations and examples are provided.

2. Definitions

2.1. Requirements Language

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].

2.2. Terminology

The following terms and abbreviations are used in this document:

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

Media Stream: A sequence of RTP packets using a single SSRC that together carry part or all of the content of a specific Media Type from a specific sender source within a given RTP session.

Media Aggregate: All Media Streams related to a particular Source.

Media Type: Audio, video, text or data whose form and meaning are defined by a specific real-time application.

Source: The source of a particular media stream. Either a single media capturing device such as a video camera, or a microphone, or a specific output of a media production function, such as an audio mixer, or some video editing function.

3. RTP Multiplex Points

This section describes the existing RTP tools that enable multiplexing of different media streams and RTP functionalities.

3.1. Session

The RTP Session is the highest semantic level in RTP and contains all of the RTP functionality.

RTP in itself does not contain any Session identifier, but relies on the underlying transport. For example, when running RTP on top of UDP, an RTP endpoint can identify and delimit an RTP Session from other RTP Sessions through the UDP source and destination transport address, consisting of network address and port number(s). Most commonly only the destination address, i.e. all traffic received on a particular port, is defined as belonging to a specific RTP Session. It is worth noting that in practice a more narrow definition of the transport flows that are related to a give RTP session is possible. An RTP session can for example be defined as one or more 5-tuples (Transport Protocol, Source Address, Source Port, Destination Address, Destination Port). Any set of identifiers of RTP and RTCP packet flows are sufficient to determine if the flow belongs to a particular session or not.

Commonly, RTP and RTCP use separate ports and the destination transport address is in fact an address pair, but in the case of RTP/RTCP multiplex [RFC5761] there is only a single port.

A source that changes its source transport address during a session must also choose a new SSRC identifier to avoid being interpreted as a looped source.

The set of participants considered part of the same RTP Session is defined by[RFC3550] as those that share a single SSRC space. That is, those participants that can see an SSRC identifier transmitted by any one of the other participants. A participant can receive an SSRC either as SSRC or CSRC in RTP and RTCP packets. Thus, the RTP Session scope is decided by the participants' network interconnection topology, in combination with RTP and RTCP forwarding strategies deployed by end-points and any interconnecting middle nodes.

3.2. SSRC

The Synchronization Source (SSRC) identifier is used to identify individual sources within an RTP Session. The SSRC number is globally unique within an RTP Session and all RTP implementations must be prepared to use procedures for SSRC collision handling, which results in an SSRC number change. The SSRC number is randomly chosen, carried in every RTP packet header and is not dependent on network address. SSRC is also used as identifier to refer to separate media streams in RTCP.

A media source having an SSRC identifier can be of different types:

Real: Connected to a "physical" media source, for example a camera or microphone.

Conceptual: A source with some attributed property generated by some network node, for example a filtering function in an RTP mixer that provides the most active speaker based on some criteria, or a mix representing a set of other sources.

Virtual: A source that does not generate any RTP media stream in itself (e.g. an end-point only receiving in an RTP session), but anyway need a sender SSRC for use as source in RTCP reports.

Note that a "multimedia source" that generates more than one media type, e.g. a conference participant sending both audio and video, need not (and commonly should not) use the same SSRC value across RTP sessions. RTCP Compound packets containing the CNAME SDES item is the designated method to bind an SSRC to a CNAME, effectively cross-correlating SSRCs within and between RTP Sessions as coming from the same end-point. The main property attributed to SSRCs associated with the same CNAME is that they are from a particular synchronization context and may be synchronized at playback. There exist a few other methods to relate different SSRC where use of CNAME is inappropriate, such as session-based RTP retransmission [RFC4588].

Note also that RTP sequence number and RTP timestamp are scoped by SSRC and thus independent between different SSRCs.

An RTP receiver receiving a previously unseen SSRC value must interpret it as a new source. It may in fact be a previously existing source that had to change SSRC number due to an SSRC conflict. However, the originator of the previous SSRC should have ended the conflicting source by sending an RTCP BYE for it prior to starting to send with the new SSRC, so the new SSRC is anyway effectively a new source.

Some RTP extension mechanisms already require the RTP stacks to handle additional SSRCs, like SSRC multiplexed RTP retransmission [RFC4588]. However, that still only requires handling a single media decoding chain per pair of SSRCs.

3.3. CSRC

The Contributing Source (CSRC) can arguably be seen as a sub-part of a specific SSRC and thus a multiplexing point. It is optionally included in the RTP header, shares the SSRC number space and specifies which set of SSRCs that has contributed to the RTP payload. However, even though each RTP packet and SSRC can be tagged with the contained CSRCs, the media representation of an individual CSRC is in general not possible to extract from the RTP payload since it is typically the result of a media mixing (merge) operation (by an RTP mixer) on the individual media streams corresponding to the CSRC identifiers. Due to these restrictions, CSRC will not be considered a fully qualified multiplex point and will be disregarded in the rest of this document.

3.4. Payload Type

The Payload Type number is also carried in every RTP packet header and identifies what format the RTP payload has. The term "format" here includes whatever can be described by out-of-band signaling means for dynamic payload types, as well as the statically allocated payload types in [RFC3551]. In SDP the term "format" includes media type, RTP timestamp sampling rate, codec, codec configuration, payload format configurations, and various robustness mechanisms such as redundant encodings [RFC2198].

The meaning of a Payload Type definition (the number) is re-used between all media streams within an RTP session, when the definition is either static or signaled through SDP. There however do exist cases where each end-point have different sets of payload types due to SDP offer/answer.

Although Payload Type definitions are commonly local to an RTP Session, there are some uses where Payload Type numbers need be unique across RTP Sessions. This is for example the case in Media

Decoding Dependency [RFC5583] where Payload Types are used to describe media dependency across RTP Sessions.

Given that multiple Payload Types are defined in an RTP Session, a media sender is free to change the Payload Type on a per packet basis. One example of designed per-packet change of Payload Type is a speech codec that makes use of generic Comfort Noise [RFC3389].

The RTP Payload Type in RTP is designed such that only a single Payload Type is valid at any time instant in the SSRC's timestamp time line, effectively time-multiplexing different Payload Types if any switch occurs. Even when this constraint is met, having different rates on the RTP timestamp clock for the RTP Payload Types in use in the same RTP Session have issues such as loss of synchronization. Payload Type clock rate switching requires some special consideration that is described in the multiple clock rates specification [I-D.ietf-avtext-multiple-clock-rates].

If there is a true need to send multiple Payload Types for the same SSRC that are valid for the same RTP Timestamps, then redundant encodings [RFC2198] can be used. Several additional constraints than the ones mentioned above need to be met to enable this use, one of which are that the combined payload sizes of the different Payload Types must not exceed the transport MTU.

Other aspects of RTP payload format use are described in RTP Payload HowTo [I-D.ietf-payload-rtp-howto].

4. Multiple Streams Alternatives

This section reviews the alternatives to enable multi-stream handling. Let's start with describing mechanisms that could enable multiple media streams, independent of the purpose for having multiple streams.

SSRC Multiplexing: Each additional Media Stream gets its own SSRC within a RTP Session.

Session Multiplexing: Using additional RTP Sessions to handle additional Media Streams

Payload Type Multiplexing: Using different RTP payload types for different additional streams.

Independent of the reason to use additional media streams, achieving it using payload type multiplexing is not a good choice as can be seen in the below section (Section 6). The RTP payload type alone is

not suitable for cases where additional media streams are required. Streams need their own SSRCs, so that they get their own sequence number space. The SSRC itself is also important so that the media stream can be referenced and reported on.

This leaves us with two choices, either using SSRC multiplexing to have multiple SSRCs from one end-point in one RTP session, or create additional RTP sessions to hold that additional SSRC. As the below discussion will show, in reality we cannot choose a single one of the two solutions. To utilize RTP well and as efficiently as possible, both are needed. The real issue is finding the right guidance on when to create RTP sessions and when additional SSRCs in an RTP session is the right choice.

In the below discussion, please keep in mind that the reasons for having multiple media streams vary and include but are not limited to the following:

- Multiple Media Sources of the same media type
- Retransmission streams
- FEC stream
- Alternative Encoding
- Scalability layer

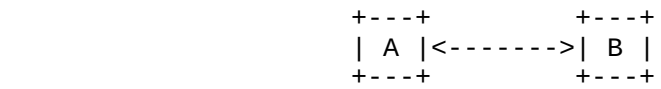
Thus the choice made due to one reason may not be the choice suitable for another reason. In the above list, the different items have different levels of maturity in the discussion on how to solve them. The clearest understanding is associated with multiple media sources of the same media type. However, all warrant discussion and clarification on how to deal with them.

5. RTP Topologies and Issues

The impact of how RTP Multiplex is performed will in general vary with how the RTP Session participants are interconnected; the RTP Topology [RFC5117]. This section describes the topologies and attempts to highlight the important behaviors concerning RTP multiplexing and multi-stream handling. It lists any identified issues regarding RTP and RTCP handling, and introduces additional topologies that are supported by RTP beyond those included in RTP Topologies [RFC5117]. The RTP Topologies that do not follow the RTP specification or do not attempt to utilize the facilities of RTP are ignored in this document.

5.1. Point to Point

This is the most basic use case with an RTP session containing of two end-points. Each end-point has one or more SSRCs.



Point to Point

5.1.1. RTCP Reporting

In cases when an end-point uses multiple SSRCs, we have found two closely related issues. The first is if every SSRC shall report on all other SSRC, even the ones originating from the same end-point. The reason for this would be ensure that no monitoring function should suspect a breakage in the RTP session.

The second issue around RTCP reporting arise when an end-point receives one or more media streams, and when the receiving end-point itself sends multiple SSRC in the same RTP session. As transport statistics are gathered per end-point and shared between the nodes, all the end-point's SSRC will report based on the same received data, the only difference will be which SSRCs sends the report. This could be considered unnecessary overhead, but for consistency it might be simplest to always have all sending SSRCs send RTCP reports on all media streams the end-point receives.

The current RTP text is silent about sending RTCP Receiver Reports for an endpoint's own sources, but does not preclude either sending or omitting them. The uncertainty in the expected behavior in those cases have likely caused variations in the implementation strategy. This could cause an interoperability issue where it is not possible to determine if the lack of reports are a true transport issue, or simply a result of implementation.

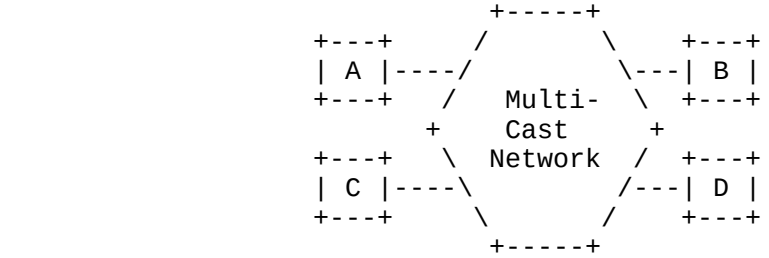
Although this issue is valid already for the simple point to point case, it needs to be considered in all topologies. From the perspective of an end-point, any solution needs to take into account what a particular end-point can determine without explicit information of the topology. For example, a Transport Translator (Relay) topology will look quite similar as point to point on an RTP level but is different. The main difference between a point to point with two SSRC being sent from the remote end-point and a Transport Translator with two single SSRC remote clients are that the RTT may vary between the SSRCs (but it is not guaranteed), and that the SSRCs may have different CNAMEs.

5.1.2. Compound RTCP Packets

When an end-point has multiple SSRCs and it needs to send RTCP packets on behalf of these SSRCs, the question arises if and how RTCP packets with different source SSRCs can be sent in the same compound packet. If it is allowed, then some consideration of the transmission scheduling is needed.

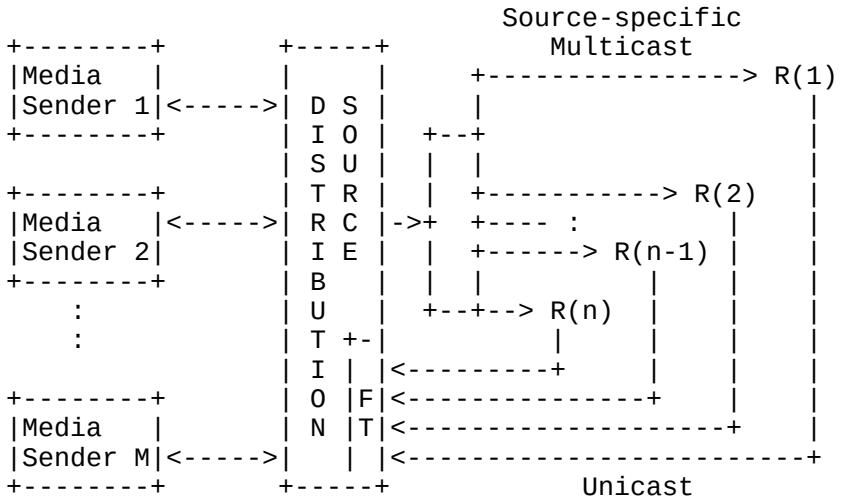
5.2. Point to Multipoint Using Multicast

This section discusses the Point to Multi-point using Multicast to interconnect the session participants. This needs to consider both Any Source Multicast (ASM) and Source-Specific Multicast (SSM).



Point to Multipoint Using Any Source Multicast

In Any Source Multicast, any of the participants can send to all the other participants, simply by sending a packet to the multicast group. That is not possible in Source Specific Multicast [RFC4607] where only a single source (Distribution Source) can send to the multicast group, creating a topology that looks like the one below:



FT = Feedback Target
Transport from the Feedback Target to the Distribution Source is via unicast or multicast RTCP if they are not co-located.

Point to Multipoint using Source Specific Multicast

In this topology a number of Media Senders (1 to M) are allowed to send media to the SSM group, sends media to the distribution source which then forwards the media streams to the multicast group. The media streams reach the Receivers (R(1) to R(n)). The Receiver's RTCP cannot be sent to the multicast group. To support RTCP, an RTP extension for SSM [RFC5760] was defined that use unicast transmission to send RTCP from the receivers to one or more Feedback Targets (FT).

As multicast is a one to many distribution system this must be taken into consideration. For example, the only practical method for adapting the bit-rate sent towards a given receiver is to use a set of multicast groups, where each multicast group represents a particular bit-rate. The media encoding is either scalable, where multiple layers can be combined, or simulcast where a single version is selected. By either selecting or combing multicast groups, the receiver can control the bit-rate sent on the path to itself. It is also common that transport robustification is sent in its own multicast group to allow for interworking with legacy or to support different levels of protection.

The result of this is three common behaviors for RTP multicast:

1. Use of multiple RTP sessions for the same media type.

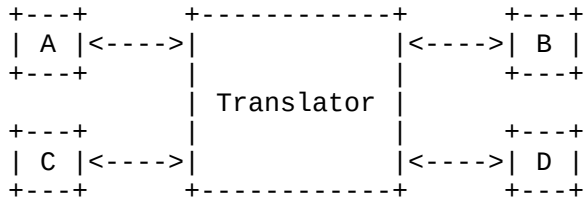
2. The need for identifying RTP sessions that are related in one of several ways.

3. The need for binding related SSRCs in different RTP sessions together.

This indicates that Multicast is an important consideration when working with the RTP multiplexing and multi stream architecture questions. It is also important to note that so far there is no special mode for basic behavior between multicast and unicast usages of RTP. Yes, there are extensions targeted to deal with multicast specific cases but the general applicability does need to be considered.

5.3. Point to Multipoint Using an RTP Translator

Transport Translators (Relays) are a very important consideration for this document as they result in an RTP session situation that is very similar to how an ASM group RTP session would behave.



Transport Translator (Relay)

One of the most important aspects with the simple relay is that it is both easy to implement and require minimal amount of resources as only transport headers are rewritten, no RTP modifications nor media transcoding occur. Thus it is most likely the cheapest and most generally deployable method for multi-point sessions. The most obvious downside of this basic relaying is that the translator has no control over how many streams needs to be delivered to a receiver. Nor can it simply select to deliver only certain streams, at least not without new RTCP extensions to coherently handle the fact that some middlebox temporarily stops a stream, preventing some receivers from reporting on it. This consistency problem in RTCP reporting needs to be handled.

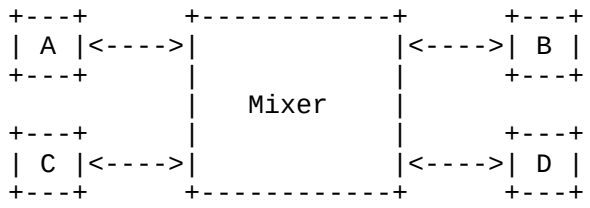
The Transport Translator does not need to have an SSRC of itself, nor need it send any RTCP reports on the flows that passes it, but it may choose to do that.

Use of a transport translator results in that any of the end-points will receive multiple SSRCs over a single unicast transport flow from the translator. That is independent of the other end-points having only a single or several SSRCs. End-points that have multiple SSRCs put further requirements on how SSRCs can be related or bound within and across RTP sessions and how they can be identified on an application level.

A Media Translator can perform a large variety of media functions affecting the media stream passing the translator, coming from one source and destined to a particular end-point. The media stream can be transcoded to a different bit-rate, change to another encoder, change the packetization of the media stream, add FEC streams, or terminate RTP retransmissions. The latter behaviors require the translator to use SSRCs that only exist in a particular sub-domain of the RTP session, and it may also create additional sessions when the mechanism applied on one side so requires.

5.4. Point to Multipoint Using an RTP Mixer

The most commonly used topology in centralized conferencing is based on the RTP Mixer. The main reason for this is that it provides a very consistent view of the RTP session towards each participant. That is accomplished through the mixer having its own SSRCs and any media sent to the participants will be sent using those SSRCs. If the mixer wants to identify the underlying media sources for its conceptual streams, it can identify them using CSRC. The media stream the mixer provides can be an actual media mixing of multiple media sources. It might also be as simple as selecting one of the underlying sources based on some mixer policy or control signalling.

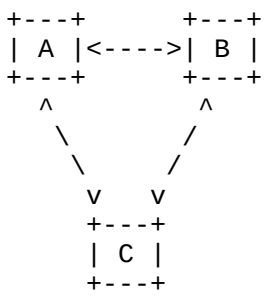


RTP Mixer

In the case where the mixer does stream selection, an application may in fact desire multiple simultaneous streams but only as many as the mixer can handle. As long as the mixer and an end-point can agree on the maximum number of streams and how the streams that are delivered are selected, this provides very good functionality. As these streams are forwarded using the mixer’s SSRCs, there are no inconsistencies within the session.

5.5. Point to Multipoint using Multiple Unicast flows

Based on the RTP session definition, it is clearly possible to have a joint RTP session over multiple transport flows like the below three end-point joint session. In this case, A needs to send its' media streams and RTCP packets to both B and C over their respective transport flows. As long as all participants do the same, everyone will have a joint view of the RTP session.

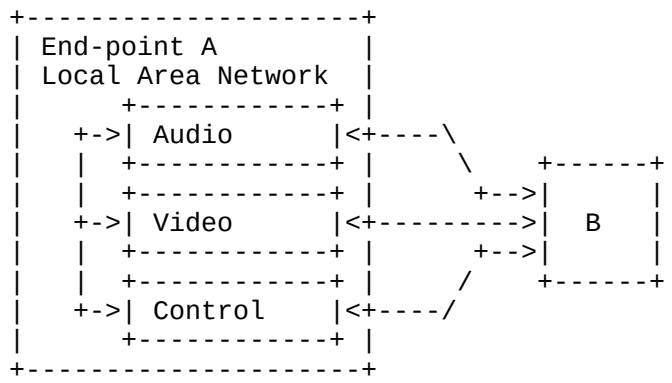


Point to Multi-Point using Multiple Unicast Transprots

This doesn't create any additional requirements beyond the need to have multiple transport flows associated with a single RTP session. Note that an end-point may use a single local port to receive all these transport flows, or it might have separate local reception ports for each of the end-points.

5.6. Decomposited End-Point

There is some possibility that an RTP end-point implementation in fact reside on multiple devices, each with their own network address. A very basic use case for this would be to separate audio and video processing for a particular end-point, like a conference room, into one device handling the audio and another handling the video being interconnected by some control functions allowing them to behave as a single end-point.

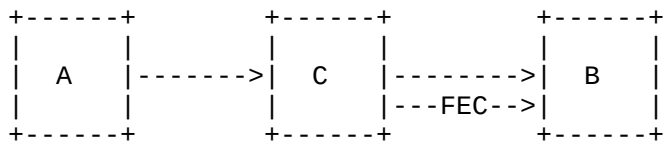


Decomposited End-Point

In the above usage, let us assume that the RTP sessions are different for audio and video. The audio and video parts will use a common CNAME and also have a common clock to ensure that synchronization and clock drift handling works despite the decomposition. However, if the audio and video were in a single RTP session then this use case becomes problematic. This as all transport flow receivers will need to receive all the other media streams that are part of the session. Thus the audio component will receive also all the video media streams, while the video component will receive all the audio ones, thus doubling the site's bandwidth requirements from all other session participants. With a joint RTP session it also becomes evident that a given end-point, as interpreted from a CNAME perspective, has two sets of transport flows for receiving the streams and the decomposition isn't hidden.

The requirements that can derived from the above usage is that the transport flows for each RTP session might be under common control but still go to what looks like different end-points based on addresses and ports. A conclusion can also be reached that decomposition without using separate RTP sessions has downsides and potential for RTP/RTCP issues.

There exist another use case which might be considered as a decomposited end-point. However, as will be shown this should be considered a translator instead. An example of this is when an end-point A sends a media flow to B. On the path there is a device C that on A's behalf does something with the media streams, for example adds an RTP session with FEC information for A's media streams. C will in this case need to bind the new FEC streams to A's media stream by using the same CNAME as A.



When Decomposition is a Translator

This type of functionality where C does something with the media stream on behalf of A is clearly covered under the media translator definition (Section 5.3).

6. Dismissing Payload Type Multiplexing

Before starting a discussion on when to use what alternative, we will first document a number of reasons why using the payload type as a multiplexing point for anything related to multiple streams is unsuitable and will not be considered further.

If one attempts to use Payload type multiplexing beyond it's defined usage, that has well known negative effects on RTP. To use Payload type as the single discriminator for multiple streams implies that all the different media streams are being sent with the same SSRC, thus using the same timestamp and sequence number space. This has many effects:

1. Putting restraint on RTP timestamp rate for the multiplexed media. For example, media streams that use different RTP timestamp rates cannot be combined, as the timestamp values need to be consistent across all multiplexed media frames. Thus streams are forced to use the same rate. When this is not possible, Payload Type multiplexing cannot be used.

2. Many RTP payload formats may fragment a media object over multiple packets, like parts of a video frame. These payload formats need to determine the order of the fragments to correctly decode them. Thus it is important to ensure that all fragments related to a frame or a similar media object are transmitted in sequence and without interruptions within the object. This can relatively simple be solved on the sender side by ensuring that the fragments of each media stream are sent in sequence.

3. Some media formats require uninterrupted sequence number space between media parts. These are media formats where any missing RTP sequence number will result in decoding failure or invoking of a repair mechanism within a single media context. The text/

T140 payload format [RFC4103] is an example of such a format. These formats will need a sequence numbering abstraction function between RTP and the individual media stream before being used with Payload Type multiplexing.

4. Sending multiple streams in the same sequence number space makes it impossible to determine which Payload Type and thus which stream a packet loss relates to.

5. If RTP Retransmission [RFC4588] is used and there is a loss, it is possible to ask for the missing packet(s) by SSRC and sequence number, not by Payload Type. If only some of the Payload Type multiplexed streams are of interest, there is no way of telling which missing packet(s) belong to the interesting stream(s) and all lost packets must be requested, wasting bandwidth.

6. The current RTCP feedback mechanisms are built around providing feedback on media streams based on stream ID (SSRC), packet (sequence numbers) and time interval (RTP Timestamps). There is almost never a field to indicate which Payload Type is reported, so sending feedback for a specific media stream is difficult without extending existing RTCP reporting.

7. The current RTCP media control messages [RFC5104] specification is oriented around controlling particular media flows, i.e. requests are done addressing a particular SSRC. Such mechanisms would need to be redefined to support Payload Type multiplexing.

8. The number of payload types are inherently limited. Accordingly, using Payload Type multiplexing limits the number of streams that can be multiplexed and does not scale. This limitation is exacerbated if one uses solutions like RTP and RTCP multiplexing [RFC5761] where a number of payload types are blocked due to the overlap between RTP and RTCP.

9. At times, there is a need to group multiplexed streams and this is currently possible for RTP Sessions and for SSRC, but there is no defined way to group Payload Types.

10. It is currently not possible to signal bandwidth requirements per media stream when using Payload Type Multiplexing.

11. Most existing SDP media level attributes cannot be applied on a per Payload Type level and would require re-definition in that context.

12. A legacy end-point that doesn't understand the indication that different RTP payload types are different media streams may be slightly confused by the large amount of possibly overlapping or identically defined RTP Payload Types.

7. Multiple Streams Discussion

7.1. Introduction

Using multiple media streams is a well supported feature of RTP. However, what can be unclear for most implementors or people writing RTP/RTCP extensions attempting to apply multiple streams, is when it is most appropriate to add an additional SSRC in an existing RTP session and when it is better to use multiple RTP sessions. This section tries to discuss the various considerations needed. The next section then concludes with some guidelines.

7.2. RTP/RTCP Aspects

This section discusses RTP and RTCP aspects worth considering when selecting between SSRC multiplexing and Session multiplexing.

7.2.1. The RTP Specification

RFC 3550 contains some recommendations and a bullet list with 5 arguments for different aspects of RTP multiplexing. Let's review Section 5.2 of [RFC3550], reproduced below:

"For efficient protocol processing, the number of multiplexing points should be minimized, as described in the integrated layer processing design principle [ALF]. In RTP, multiplexing is provided by the destination transport address (network address and port number) which is different for each RTP session. For example, in a teleconference composed of audio and video media encoded separately, each medium SHOULD be carried in a separate RTP session with its own destination transport address.

Separate audio and video streams SHOULD NOT be carried in a single RTP session and demultiplexed based on the payload type or SSRC fields. Interleaving packets with different RTP media types but using the same SSRC would introduce several problems:

1. If, say, two audio streams shared the same RTP session and the same SSRC value, and one were to change encodings and thus acquire a different RTP payload type, there would be no general way of identifying which stream had changed encodings.

2. An SSRC is defined to identify a single timing and sequence number space. Interleaving multiple payload types would require different timing spaces if the media clock rates differ and would require different sequence number spaces to tell which payload type suffered packet loss.

3. The RTCP sender and receiver reports (see Section 6.4) can only describe one timing and sequence number space per SSRC and do not carry a payload type field.

4. An RTP mixer would not be able to combine interleaved streams of incompatible media into one stream.

5. Carrying multiple media in one RTP session precludes: the use of different network paths or network resource allocations if appropriate; reception of a subset of the media if desired, for example just audio if video would exceed the available bandwidth; and receiver implementations that use separate processes for the different media, whereas using separate RTP sessions permits either single- or multiple-process implementations.

Using a different SSRC for each medium but sending them in the same RTP session would avoid the first three problems but not the last two.

On the other hand, multiplexing multiple related sources of the same medium in one RTP session using different SSRC values is the norm for multicast sessions. The problems listed above don’t apply: an RTP mixer can combine multiple audio sources, for example, and the same treatment is applicable for all of them. It may also be appropriate to multiplex streams of the same medium using different SSRC values in other scenarios where the last two problems do not apply."

Let’s consider one argument at a time. The first is an argument for using different SSRC for each individual media stream, which still is very applicable.

The second argument is advocating against using payload type multiplexing, which still stands as can been seen by the extensive list of issues found in Section 6.

The third argument is yet another argument against payload type multiplexing.

The fourth is an argument against multiplexing media streams that require different handling into the same session. This is to simplify the processing at any receiver of the media stream. If all media streams that exist in an RTP session is of one media type and

one particular purpose, there is no need for deeper inspection of the packets before processing them in both end-points and RTP aware middle nodes.

The fifth argument discusses network aspects that we will discuss more below in Section 7.4. It also goes into aspects of implementation, like decomposed end-points where different processes or inter-connected devices handle different aspects of the whole multi-media session.

A summary of RFC 3550's view on multiplexing is to use unique SSRCs for anything that is its' own media/packet stream, and secondly use different RTP sessions for media streams that don't share media type and purpose, to maximize flexibility when it comes to processing and handling of the media streams.

This mostly agrees with the discussion and recommendations in this document. However, there has been an evolution of RTP since that text was written which needs to be reflected in the discussion. Additional clarifications for specific cases are also needed.

7.2.2. Multiple SSRC Legacy Considerations

When establishing RTP sessions that may contain end-points that aren't updated to handle multiple streams following these recommendations, a particular application can have issues with multiple SSRCs within a single session. These issues include:

1. Need to handle more than one stream simultaneously rather than replacing an already existing stream with a new one.

2. Be capable of decoding multiple streams simultaneously.

3. Be capable of rendering multiple streams simultaneously.

RTP Session multiplexing could potentially avoid these issues if there is only a single SSRC at each end-point, and in topologies which appears like point to point as seen the end-point. However, forcing the usage of session multiplexing due to this reason would be a great mistake, as it is likely that a significant set of applications will need a combination of SSRC multiplexing of several media sources and session multiplexing for other aspects such as encoding alternatives, robustification or simply to support legacy. However, this issue does need consideration when deploying multiple media streams within an RTP session where legacy end-points may occur.

7.2.3. RTP Specification Clarifications Needed

The RTP specification contains a few things that are potential interoperability issues when using multiple SSRCs within a session. These issues are described and discussed in Section 9. These should not be considered strong arguments against using SSRC multiplexing when otherwise appropriate, and there are some issues we expect to be solved in the near future.

7.2.4. Handling Varying sets of Senders

Another potential issue that needs to be considered is where a limited set of simultaneously active sources varies within a larger set of session members. As each media decoding chain may contain state, it is important that this type of usage ensures that a receiver can flush a decoding state for an inactive source and if that source becomes active again, it does not assume that this previous state exists.

This behavior might in certain applications be possible to limit to a particular RTP Session and instead use multiple RTP sessions. But in some cases it is likely unavoidable and the most appropriate thing is to SSRC multiplex.

7.2.5. Cross Session RTCP requests

There currently exist no functionality to make truly synchronized and atomic RTCP requests across multiple RTP Sessions. Instead separate RTCP messages will have to be sent in each session. This gives SSRC multiplexed streams a slight advantage as RTCP requests for different streams in the same session can be sent in a compound RTCP packet. Thus providing an atomic operation if different modifications of different streams are requested at the same time.

In Session multiplexed cases, the RTCP timing rules in the sessions and the transport aspects, such as packet loss and jitter, prevents a receiver from relying on atomic operations, instead more robust and forgiving mechanisms need to be used.

7.2.6. Binding Related Sources

A common problem in a number of various RTP extensions has been how to bind together related sources. This issue is common independent of SSRC multiplexing and Session Multiplexing, and any solution and recommendation to the problem should work equally well for both to avoid creating barriers between using session multiplexing and SSRC multiplexing.

The current solutions don't have these properties. There exist one solution for grouping RTP session together in SDP [RFC5888] to know which RTP session contains for example the FEC data for the source data in another session. However, this mechanism does not work on individual media flows and is thus not directly applicable to the problem. The other solution is also SDP based and can group SSRCs within a single RTP session [RFC5576]. Thus this mechanism can bind media streams in SSRC multiplexed cases. Both solutions have the shortcoming of being restricted to SDP based signalling and also do not work in cases where the session's dynamic properties are such that it is difficult or resource consuming to keep the list of related SSRCs up to date.

One possible solution could be to mandate the same SSRC being used in all RTP session in case of session multiplexing. We do note that Section 8.3 of the RTP Specification [RFC3550] recommends using a single SSRC space across all RTP sessions for layered coding. However this recommendation has some downsides and is less applicable beyond the field of layered coding. To use the same sender SSRC in all RTP sessions from a particular end-point can cause issues if an SSRC collision occurs. If the same SSRC is used as the required binding between the streams, then all streams in the related RTP sessions must change their SSRC. This is extra likely to cause problems if the participant populations are different in the different sessions. For example, in case of large number of receivers having selected totally random SSRC values in each RTP session as RFC 3550 specifies, a change due to a SSRC collision in one session can then cause a new collision in another session. This cascading effect is not severe but there is an increased risk that this occurs for well populated sessions. In addition, being forced to change the SSRC affects all the related media streams; instead of having to re-synchronize only the originally conflicting stream, all streams will suddenly need to be re-synchronized with each other. This will prevent also the media streams not having an actual collision from being usable during the re-synchronization and also increases the time until synchronization is finalized. In addition, it requires exception handling in the SSRC generation.

The above collision issue does not occur in case of having only one SSRC space across all sessions and all participants will be part of at least one session, like the base layer in layered encoding. In that case the only downside is the special behavior that needs to be well defined by anyone using this. But, having an exception behavior where the SSRC space is common across all session an that doesn't fit all the RTP extensions or payload formats present in the sessions is a issue. It is possible to create a situation where the different mechanisms can't be combined due to the non standard SSRC allocation behavior.

Existing mechanisms with known issues:

RTP Retransmission (RFC4588): Has two modes, one for SSRC multiplexing and one for Session multiplexing. The session multiplexing requires the same CNAME and mandates that the same SSRC is used in both sessions. Using the same SSRC does work but will potentially have issues in certain cases. In SSRC multiplexed mode the CNAME is used, and when the first retransmission request is sent, one must not have another retransmission request outstanding for an SSRC which don't have a the binding between the original SSRC and the retransmission stream's SSRC. This works but creates some limitations that can be avoided by a more explicit mechanism. The SDP based ssrc-group mechanism is sufficient in this case as long as the application can rely on the signalling based solution.

Scalable Video Coding (RFC6190): As an example of scalable coding, SVC [RFC6190] has various modes. The Multi Session Transmission (MST) uses Session multiplexing to separate scalability layers. However, this specification has failed to explicit how these layers are bound together in cases where CNAME isn't sufficient. CNAME is no longer sufficient when more than one media source occur within a session that have the same CNAME, for example due to multiple video cameras capturing the same lecture hall. This likely implies that a single SSRC space as recommend by Section 8.3 of RTP [RFC3550] is to be used.

Forward Error Correction: If some type of FEC or redundancy stream is being sent, it needs it's own SSRC, with the exception of constructions like redundancy encoding [RFC2198]. Thus in case of transmitting the FEC in the same session as the source data, the inter SSRC relation within a session is needed. In case of sending the redundant data in a separate session from the source, the SSRC in each session needs to be related. This occurs for example in RFC5109 when using session separation of original and FEC data. SSRC multiplexing is not supported, only using redundant encoding is supported.

This issue appears to need action to harmonize and avoid future shortcomings in extension specifications. A proposed solution for handling this issue is [I-D.westerlund-avtext-rtcp-sdes-srcname].

7.2.7. Forward Error Correction

There exist a number of Forward Error Correction (FEC) based schemes for how to reduce the packet loss of the original streams. Most of the FEC schemes will protect a single source flow. The protection is achieved by transmitting a certain amount of redundant information

that is encoded such that it can repair one or more packet loss over the set of packets they protect. This sequence of redundant information also needs to be transmitted as its own media stream, or in some cases instead of the original media stream. Thus many of these schemes creates a need for binding the related flows as discussed above. They also create additional flows that need to be transported. Looking at the history of these schemes, there is both SSRC multiplexed and Session multiplexed solutions and some schemes that support both.

Using a Session multiplexed solution provides good support for legacy when deploying FEC or changing the scheme used so that some set of receivers may not be able to utilize the FEC information. By placing it in a separate RTP session, it can easily be ignored.

In usages involving multicast, having the FEC information on its own multicast group and RTP session allows for flexibility, for example when using Rapid Acquisition of Multicast Groups (RAMS) [RFC6285]. During the RAMS burst where data is received over unicast and where it is possible to combine with unicast based retransmission [RFC4588], there is no need to burst the FEC data related to the burst of the source media streams needed to catch up with the multicast group. This saves bandwidth to the receiver during the burst, enabling quicker catch up. When the receiver has catched up and joins the multicast group(s) for the source, it can at the same time join the multicast group with the FEC information. Having the source stream and the FEC in separate groups allow for easy separation in the Burst/Retransmission Source (BRS) without having to individually classify packets.

7.2.8. Transport Translator Sessions

A basic Transport Translator relays any incoming RTP and RTCP packets to the other participants. The main difference between SSRC multiplexing and Session multiplexing resulting from this use case is that for SSRC multiplexing it is not possible for a particular session participant to decide to receive a subset of media streams. When using separate RTP sessions for the different sets of media streams, a single participant can choose to leave one of the sessions but not the other.

7.2.9. Multiple Media Types in one RTP session

Having different media types, like audio and video, in the same RTP sessions is not forbidden, only recommended against as can be seen in Section 7.2.1. When using multiple media types, there are a number of considerations:

Payload Type gives Media Type: This solution is dependent on getting the media type from the Payload Type. Thus overloading this de-multiplexing point in a receiver for two purposes. First for the main media type and determining the processing chain, then later for the exact configuration of the encoder and packetization.

Payload Type field limiations: The total number of Payload Types available to use in an RTP session is fairly limited, especially if Multiplexing RTP Data and Control Packets on a Single Port [RFC5761] is used. For certain applications negotiating a large set of codes and configuration may become an issue.

Don’t switch media types for an SSRC: The primary reasons to avoid switching from sending for example audio to sending video using the same SSRC is the implications on a receiver. When this happens, the processing chain in the receiver will have to switch from one media type to another. As the different media type’s entire processing chains are different and are connected to different outputs it is difficult to reuse the decoding chain, which a normal codec change likely can. Instead the entire processing chain has to be torn down and replaced. In addition, there is likely a clock rate switching problem, possibly resulting in synchronization loss at the point of switching media type if some packet loss occurs.

RTCP Bit-rate Issues: If the media types are significantly different in bit-rate, the RTCP bandwidth rates assigned to each source in a session can result in interesting effects, like that the RTCP bit-rate share for an audio stream is larger than the actual audio bit-rate. In itself this doesn’t cause any conflicts, only potentially unnecessary overhead. It is possible to avoid this using AVPF or SAVPF and setting trr-int parameter, which can bring down unnecessary regular reporting while still allowing for rapid feedback.

Decomposited end-points: Decomposited nodes that rely on the regular network to separate audio and video to different devices do not work well with this session setup. If they are forced to work, all media receiver parts of a decomposited end-point will receive all media, thus doubling the bit-rate consumption for the end-point.

RTP Mixers and Translators: An RTP mixer or Media Translator will also have to support this particular session setup, where it before could rely on the RTP session to determine what processing options should be applied to the incoming packets.

As can be seen, there is nothing in here that prevents using a single

RTP session for multiple media types, however it does create a number of limitations and special case implementation requirements. So anyone considering to use this setup should carefully review if the reasons for using a single RTP session is sufficient to motivate this special case.

7.3. Signalling Aspects

There exist various signalling solutions for establishing RTP sessions. Many are SDP [RFC4566] based, however SDP functionality is also dependent on the signalling protocols carrying the SDP. Where RTSP [RFC2326] and SAP [RFC2974] both use SDP in a declarative fashion, SIP [RFC3261] uses SDP with the additional definition of Offer/Answer [RFC3264]. The impact on signalling and especially SDP needs to be considered as it can greatly affect how to deploy a certain multiplexing point choice.

7.3.1. Session Oriented Properties

One aspect of the existing signalling is that it is focused around sessions, or at least in the case of SDP the media description. There are a number of things that are signalled on a session level/ media description but that are not necessarily strictly bound to an RTP session and could be of interest to signal specifically for a particular media stream within the session. The following properties have been identified as being potentially useful to signal not only on RTP session level:

- Bitrate/Bandwidth exist today only at aggregate or a common any media stream limit

- Which SSRC that will use which RTP Payload Types

Some of these issues are clearly SDP’s problem rather than RTP limitations. However, if the aim is to deploy an SSRC multiplexed solution that contains several sets of media streams with different properties (encoding/packetization parameter, bit-rate, etc), putting each set in a different RTP session would directly enable negotiation of the parameters for each set. If insisting on SSRC multiplexing, a number of signalling extensions are needed to clarify that there are multiple sets of media streams with different properties and that they shall in fact be kept different, since a single set will not satisfy the applications requirements.

This does in fact create a strong driver to use RTP session multiplexing for any case where different sets of media streams with different requirements exist.

7.3.2. SDP Prevents Multiple Media Types

SDP encoded in its structure a prevention against using multiple media types in the same RTP session. A media description in SDP can only have a single media type; audio, video, text, image, application. This media type is used as the top-level media type for identifying the actual payload format bound to a particular payload type using the rtpmap attribute. Thus a high fence against using multiple media types in the same session was created.

There is a proposal in the MMUSIC WG for how one could allow multiple media lines describe a single underlying transport [I-D.holmberg-mmusic-sdp-bundle-negotiation] and thus support either one RTP session with multiple media types. There is also a solution for multiplexing multiple RTP sessions onto the same transport [I-D.westerlund-avtcore-single-transport-multiplexing].

7.4. Network Apsects

The multiplexing choice has impact on network level mechanisms that need to be considered by the implementor.

7.4.1. Quality of Service

When it comes to Quality of Service mechanisms, they are either flow based or marking based. RSVP [RFC2205] is an example of a flow based mechanism, while Diff-Serv [RFC2474] is an example of a Marking based one. For a marking based scheme, the method of multiplexing will not affect the possibility to use QoS.

However, for a flow based scheme there is a clear difference between the methods. SSRC multiplexing will result in all media streams being part of the same 5-tuple (protocol, source address, destination address, source port, destination port) which is the most common selector for flow based QoS. Thus, separation of the level of QoS between media streams is not possible. That is however possible for session based multiplexing, where each different version can be in a different RTP session that can be sent over different 5-tuples.

7.4.2. NAT and Firewall Traversal

In today’s network there exist a large number of middleboxes. The ones that normally have most impact on RTP are Network Address Translators (NAT) and Firewalls (FW).

Below we analyze and comment on the impact of requiring more underlying transport flows in the presence of NATs and Firewalls:

End-Point Port Consumption: A given IP address only has 65536 available local ports per transport protocol for all consumers of ports that exist on the machine. This is normally never an issue for an end-user machine. It can become an issue for servers that handle large number of simultaneous streams. However, if the application uses ICE to authenticate STUN requests, a server can serve multiple end-points from the same local port, and use the whole 5-tuple (source and destination address, source and destination port, protocol) as identifier of flows after having securely bound them to the remote end-point address using the STUN request. In theory the minimum number of media server ports needed are the maximum number of simultaneous RTP Sessions a single end-point may use. In practice, implementation will probably benefit from using more server ports to simplify implementation or avoid performance bottlenecks.

NAT State: If an end-point sits behind a NAT, each flow it generates to an external address will result in a state that has to be kept in the NAT. That state is a limited resource. In home or Small Office/Home Office (SOHO) NATs, memory or processing are usually the most limited resources. For large scale NATs serving many internal end-points, available external ports are typically the scarce resource. Port limitations is primarily a problem for larger centralized NATs where end-point independent mapping requires each flow to use one port for the external IP address. This affects the maximum number of internal users per external IP address. However, it is worth pointing out that a real-time video conference session with audio and video is likely using less than 10 UDP flows, compared to certain web applications that can use 100+ TCP flows to various servers from a single browser instance.

NAT Traversal Excess Time: Making the NAT/FW traversal takes a certain amount of time for each flow. It also takes time in a phase of communication between accepting to communicate and the media path being established which is fairly critical. The best case scenario for how much extra time it can take following the specified ICE procedures are: 1.5*RTT + Ta*(Additional_Flows-1), where Ta is the pacing timer, which ICE specifies to be no smaller than 20 ms. That assumes a message in one direction, and then an immediate triggered check back. This as ICE first finds one candidate pair that works prior to establish multiple flows. Thus, there are no extra time until one has found a working candidate pair. Based on that working pair the extra time is to in parallel establish the, in most cases 2-3, additional flows.

NAT Traversal Failure Rate: Due to the need to establish more than a single flow through the NAT, there is some risk that establishing the first flow succeeds but that one or more of the additional flows fail. The risk that this happens is hard to quantify, but it should be fairly low as one flow from the same interfaces has just been successfully established . Thus only rare events such as NAT resource overload, or selecting particular port numbers that are filtered etc, should be reasons for failure.

SSRC multiplexing keeps additional media streams within one RTP Session and does not introduce any additional NAT traversal complexities per media stream. In contrast, the session multiplexing is using one RTP session per media stream. Thus additional lower layer transport flows will be required, unless an explicit de-multiplexing layer is added between RTP and the transport protocol. A proposal for how to multiplex multiple RTP sessions over the same single lower layer transport exist in [I-D.westerlund-avtcore-single-transport-multiplexing].

7.4.3. Multicast

Multicast groups provides a powerful semantics for a number of real-time applications, especially the ones that desire broadcast-like behaviors with one end-point transmitting to a large number of receivers, like in IPTV. But that same semantics do result in a certain number of limitations.

One limitation is that for any group, sender side adaptation to the actual receiver properties causes a degradation for all participants to what is supported by the receiver with the worst conditions among the group participants. In most cases this is not acceptable. Instead various receiver based solutions are employed to ensure that the receivers achieve best possible performance. By using scalable encoding and placing each scalability layer in a different multicast group, the receiver can control the amount of traffic it receives. To have each scalability layer on a different multicast group, one RTP session per multicast group is used.

If instead a single RTP session over multiple transports were to be deployed, i.e. multicast groups with each layer as it’s own SSRC, then very different views of the RTP session would exist. That as one receiver may see only a single layer (SSRC), while another may see three SSRCs if it joined three multicast groups. This would cause disjoint RTCP reports where a management system would not be able to determine if a receiver isn’t reporting on a particular SSRC due to that it is not a member of that multicast group, or because it doesn’t receive it as a result of a transport failure.

Thus it appears easiest and most straightforward to use multiple RTP sessions. In addition, the transport flow considerations in multicast are a bit different from unicast. First of all there is no shortage of port space, as each multicast group has its own port space.

7.4.4. Multiplexing multiple RTP Session on a Single Transport

For applications that doesn't need flow based QoS and like to save ports and NAT/FW traversal costs, there is a proposal for how to achieve multiplexing of multiple RTP sessions over the same lower layer transport [I-D.westerlund-avtcore-single-transport-multiplexing]. Using such a solution would allow session multiplexing without most of the perceived downsides of additional RTP sessions creating a need for additional transport flows.

7.5. Security Aspects

On the basic level there is no significant difference in security when having one RTP session and having multiple. However, there are a few more detailed considerations that might need to be considered in certain usages.

7.5.1. Security Context Scope

When using SRTP [RFC3711] the security context scope is important and can be a necessary differentiation in some applications. As SRTP's crypto suites (so far) is built around symmetric keys, the receiver will need to have the same key as the sender. This results in that none in a multi-party session can be certain that a received packet really was sent by the claimed sender or by another party having access to the key. In most cases this is a sufficient security property, but there are a few cases where this does create situations.

The first case is when someone leaves a multi-party session and one wants to ensure that the party that left can no longer access the media streams. This requires that everyone re-keys without disclosing the keys to the excluded party.

A second case is when using security as an enforcing mechanism for differentiation. Take for example a scalable layer or a high quality simulcast version which only premium users are allowed to access. The mechanism preventing a receiver from getting the high quality stream can be based on the stream being encrypted with a key that user can't access without paying premium, having the key-management limit access to the key.

In the latter case it is likely easiest from signalling, transport (if done over multicast) and security to use a different RTP session. That way the user(s) not intended to receive a particular stream can easily be excluded. There is no need to have SSRC specific keys, which many of the key-management systems cannot handle.

7.5.2. Key-Management for Multi-party session

Performing key-management for Multi-party session can be a challenge. This section considers some of the issues.

Transport translator based session cannot use Security Description [RFC4568] nor DTLS-SRTP [RFC5764] without an extension as each end-point provides it's set of keys. In centralized conference, the signalling counterpart is a conference server and the media plane unicast counterpart (to which DTLS messages would be sent) is the translator. Thus an extension like Encrypted Key Transport [I-D.ietf-avt-srtp-ekt] are needed or a MIKEY [RFC3830] based solution that allows for keying all session participants with the same master key.

Keying of multicast transported SRTP face similar challenges as the transport translator case.

8. Guidelines

This section contains a number of recommendations for implementors or specification writers when it comes to handling multi-stream.

Don't Require the same SSRC across Sessions: As discussed in Section 7.2.6 there exist drawbacks in using the same SSRC in multiple RTP sessions as a mechanism to bind related media streams together. Instead a mechanism to explicitly signal the relation SHOULD be used, either in RTP/RTCP or in the used signalling mechanism that establish the RTP session(s).

Use SSRC multiplexing for additional Media Sources: In the cases an RTP end-point needs to transmit additional media source(s) of the same media type and purpose in the application it is RECOMMENDED to send them as additional SSRCs in the same RTP session. For example a telepresence room where there are three cameras, and each camera captures 2 persons sitting at the table, sending each camera as its own SSRC within a single RTP session is recommended.

Use additional RTP sessions for streams with different purposes: When media streams have different purpose or processing requirements it is RECOMMENDED that the different types of streams are put in different RTP sessions.

When using Session Multiplexing use grouping: When using Session Multiplexing solutions it is RECOMMENDED to be explicitly group the involved RTP sessions using the signalling mechanism, for example The Session Description Protocol (SDP) Grouping Framework. [RFC5888]

RTP/RTCP Extensions May Support SSRC and Session Multiplexing: When defining an RTP or RTCP extension, the creator needs to consider if this extension is applicable in both SSRC multiplexed and Session multiplexed usages. If it is, then any generic extensions are RECOMMENDED to support both. Applications that are not as generally applicable will have to consider if interoperability is better served by defining a single solution or providing both options.

Transport Support Extensions: When defining new RTP/RTCP extensions intended for transport support, like the retransmission or FEC mechanisms, they are RECOMMENDED to include support for both SSRC and Session multiplexing so that application developers can choose freely from the set of mechanisms without concerning themselves with if a particular solution only supports one of the multiplexing choices.

This discussion and guidelines points out that a small set of extension mechanisms could greatly improve the situation when it comes to using multiple streams independently of Session multiplexing or SSRC multiplexing. These extensions are:

Media Source Identification: A Media source identification that can be used to bind together media streams that are related to the same media source. A proposal [I-D.westerlund-avtext-rtcp-sdes-srcname] exist for a new SDES item SRCNAME that also can be used with the a=ssrc SDP attribute to provide signalling layer binding information.

SSRC limiations within RTP sessions: By providing a signalling solution that allows the signalling peers to explicitly express both support and limitations on how many simultaneous media streams an end-point can handle within a given RTP Session. That ensures that usage of SSRC multiplexing occurs when supported and without overloading an end-point. This extension is proposed in [I-D.westerlund-avtcore-max-ssrc].

9. RTP Specification Clarifications

This section describes a number of clarifications to the RTP specifications that are likely necessary for aligned behavior when RTP sessions contains more SSRCs than one local and one remote.

9.1. RTCP Reporting from all SSRCs

When one have multiple SSRC in an RTP node, then all these SSRC must send RTCP SR or RR as long as the SSRC exist. It is not sufficient that only one SSRC in the node sends report blocks on the incoming RTP streams. The reason for this is that a third party monitor may not necessarily be able to determine that all these SSRC are in fact co-located and originate from the same stack instance that gather report data.

9.2. RTCP Self-reporting

For any RTP node that sends more than one SSRC, there exist the question if SSRC1 needs to report its reception of SSRC2 and vice versa. The reason that they in fact need to report on all other local streams as being received is report consistency. A third party monitor that considers the full matrix of media streams and all known SSRC reports on these media streams would detect a gap in the reports which could be a transport issue unless identified as in fact being sources from same node.

9.3. Combined RTCP Packets

When a node contains multiple SSRCs, it is questionable if an RTCP compound packet can only contain RTCP packets from a single SSRC or if multiple SSRCs can include their packets in a joint compound packet. The high level question is a matter for any receiver processing on what to expect. In addition to that question there is the issue of how to use the RTCP timer rules in these cases, as the existing rules are focused on determining when a single SSRC can send.

10. IANA Considerations

This document makes no request of IANA.

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

11. Security Considerations

12. Acknowledgements

13. References

13.1. 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.

13.2. Informative References

[ALF] Clark, D. and D. Tennenhouse, "Architectural Considerations for a New Generation of Protocols", SIGCOMM Symposium on Communications Architectures and Protocols (Philadelphia, Pennsylvania), pp. 200--208, IEEE Computer Communications Review, Vol. 20(4), September 1990.

[I-D.holmberg-mmusic-sdp-bundle-negotiation] Holmberg, C. and H. Alvestrand, "Multiplexing Negotiation Using Session Description Protocol (SDP) Port Numbers", draft-holmberg-mmusic-sdp-bundle-negotiation-00 (work in progress), October 2011.

[I-D.ietf-avt-srtp-ekt] McGrew, D., Andreasen, F., Wing, D., and K. Fischer, "Encrypted Key Transport for Secure RTP", draft-ietf-avt-srtp-ekt-02 (work in progress), March 2011.

[I-D.ietf-avtext-multiple-clock-rates] Petit-Huguenin, M., "Support for multiple clock rates in an RTP session", draft-ietf-avtext-multiple-clock-rates-01 (work in progress), July 2011.

[I-D.ietf-payload-rtp-howto] Westerlund, M., "How to Write an RTP Payload Format", draft-ietf-payload-rtp-howto-01 (work in progress), July 2011.

[I-D.westerlund-avtcore-max-ssrc]

Westerlund, M., Burman, B., and F. Jansson, "Multiple Synchronization sources (SSRC) in RTP Session Signaling", draft-westerlund-avtcore-max-ssrc (work in progress), October 2011.

[I-D.westerlund-avtcore-single-transport-multiplexing] Westerlund, M., "Multiple RTP Session on a Single Lower-Layer Transport", draft-westerlund-avtcore-transport-multiplexing (work in progress), October 2011.

[I-D.westerlund-avtext-rtcp-sdes-srcname] Westerlund, M., Burman, B., and P. Sandgren, "RTCP SDES Item SRCNAME to Label Individual Sources", draft-westerlund-avtext-rtcp-sdes-srcname (work in progress), October 2011.

[RFC2198] Perkins, C., Kouvelas, I., Hodson, O., Hardman, V., Handley, M., Bolot, J., Vega-Garcia, A., and S. Fosse-Parisis, "RTP Payload for Redundant Audio Data", RFC 2198, September 1997.

[RFC2205] Braden, B., Zhang, L., Berson, S., Herzog, S., and S. Jamin, "Resource ReSerVation Protocol (RSVP) -- Version 1 Functional Specification", RFC 2205, September 1997.

[RFC2326] Schulzrinne, H., Rao, A., and R. Lanphier, "Real Time Streaming Protocol (RTSP)", RFC 2326, April 1998.

[RFC2474] Nichols, K., Blake, S., Baker, F., and D. Black, "Definition of the Differentiated Services Field (DS Field) in the IPv4 and IPv6 Headers", RFC 2474, December 1998.

[RFC2974] Handley, M., Perkins, C., and E. Whelan, "Session Announcement Protocol", RFC 2974, October 2000.

[RFC3261] Rosenberg, J., Schulzrinne, H., Camarillo, G., Johnston, A., Peterson, J., Sparks, R., Handley, M., and E. Schooler, "SIP: Session Initiation Protocol", RFC 3261, June 2002.

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

[RFC3389] Zopf, R., "Real-time Transport Protocol (RTP) Payload for Comfort Noise (CN)", RFC 3389, September 2002.

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

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

[RFC3830] Arkko, J., Carrara, E., Lindholm, F., Naslund, M., and K. Norrman, "MIKEY: Multimedia Internet KEYing", RFC 3830, August 2004.

[RFC4103] Hellstrom, G. and P. Jones, "RTP Payload for Text Conversation", RFC 4103, June 2005.

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

[RFC4568] Andreasen, F., Baugher, M., and D. Wing, "Session Description Protocol (SDP) Security Descriptions for Media Streams", RFC 4568, July 2006.

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

[RFC4607] Holbrook, H. and B. Cain, "Source-Specific Multicast for IP", RFC 4607, August 2006.

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

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

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

[RFC5583] Schierl, T. and S. Wenger, "Signaling Media Decoding Dependency in the Session Description Protocol (SDP)", RFC 5583, July 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.

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

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

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

[RFC6190] Wenger, S., Wang, Y., Schierl, T., and A. Eleftheriadis, "RTP Payload Format for Scalable Video Coding", RFC 6190, May 2011.

[RFC6285] Ver Steeg, B., Begen, A., Van Caenegem, T., and Z. Vax, "Unicast-Based Rapid Acquisition of Multicast RTP Sessions", RFC 6285, June 2011.

Authors' Addresses

Magnus Westerlund
Ericsson
Farogatan 6
SE-164 80 Kista
Sweden

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

Bo Burman
Ericsson
Farogatan 6
SE-164 80 Kista
Sweden

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

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

Email: csp@csperkins.org

Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: May 3, 2012

M. Westerlund
Ericsson
C. Perkins
University of Glasgow
October 31, 2011

Multiple RTP Session on a Single Lower-Layer Transport
draft-westerlund-avtcore-transport-multiplexing-01

Abstract

This document specifies how multiple RTP sessions are to be multiplexed on the same lower-layer transport, e.g. a UDP flow. It discusses various requirements that have been raised and their feasibility, which results in a solution with a certain applicability. A solution is recommended and that solution is provided in more detail, including signalling and examples.

Table of Contents

1. Introduction

There has been renewed interest for having a solution that allows multiple RTP sessions [RFC3550] to use a single lower layer transport, such as a bi-directional UDP flow. The main reason is the cost of doing NAT/FW traversal for each individual flow. ICE and other NAT/FW traversal solutions are clearly capable of attempting to open multiple flows. However, there is both increased risk for failure and an increased cost in the creation of multiple flows. The increased cost comes as slightly higher delay in establishing the traversal, and the amount of consumed NAT/FW resources. The latter might be an increasing problem in the IPv4 to IPv6 transition period.

This document draws up some requirements for consideration on how to transport multiple RTP sessions over a single lower-layer transport. These requirements will have to be weighted as the combined set of requirements result in that no known solution exist that can fulfill them completely.

A number of possible solutions are then considered and discussed with respect to their properties. Based on that, the authors recommends a shim layer variant as single solution, which is described in more detail including signalling solution and examples.

2. Conventions

2.1. Terminology

Some terminology used in this document.

Multiplexing: Unless specifically noted, all mentioning of multiplexing in this document refer to the multiplexing of multiple RTP Sessions on the same lower layer transport. It is important to make this distinction as RTP does contain a number of multiplexing points for various purposes, such as media formats (Payload Type), media sources (SSRC), and RTP sessions.

2.2. Requirements Language

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. Requirements

This section lists and discusses a number of potential requirements.

However, it is not difficult to realize that it is in fact possible to put requirements that makes the set of feasible solutions an empty set. It is thus necessary to consider which requirements that are essential to fulfill and which can be compromised on to arrive at a solution.

3.1. Support Use of Multiple RTP Sessions

This may at first glance appear to be an obvious requirement. Although the authors are convinced it is a mandatory requirement for a solution, it warrants some discussion around the implications of not having multiple RTP sessions and instead use a single RTP session.

The main purpose of RTP sessions is to allow separation of streams that have different purposes, for example different media types. A big reason for establishing this is the knowledge that any SSRC within the session is supposed to be processed in a similar way.

For simpler cases, where the streams within each media type need the same processing, it is clearly possible to find other multiplex solutions, for example based on the Payload Type and the differences in encoding that the payload type allows to describe. This may anyhow be insufficient when you get into more advanced usages where you have multiple sources of the same media type, but for different purposes or as alternatives. For example when you have one set of video sources that shows session participants and another set of video sources that shares an application or slides, you likely want to separate those streams for various reasons such as control, prioritization, QoS, methods for robustification, etc. In those cases, using the RTP session for separation of properties is a powerful tool. A tool with properties that need to be preserved when providing a solution for how to use only a single lower-layer transport.

For more discussion of the usage of RTP sessions verses other multiplexing we recommend RTP Multiplexing Architecture [I-D.westerlund-avtcore-multiplex-architecture].

3.2. Same SSRC Value in Multiple RTP Sessions

Two different RTP sessions being multiplexed on the same lower layer transport need to be able to use the same SSRC value. This is a strong requirement, for two reasons:

1. To avoid mandating SSRC assignment rules that are coordinated between the sessions. If the RTP sessions multiplexed together must have unique SSRC values, then additional code that works

between RTP Sessions is needed in the implementations. Thus raising the bar for implementing this solution. In addition, if one gateways between parts of a system using this multiplexing and parts that aren't multiplexing, the part that isn't multiplexing must also fulfil the requirements on how SSRC is assigned or force the gateway to translate SSRCs. Translating SSRC is actually hard as it requires one to understand the semantics of all current and future RTP and RTCP extensions. Otherwise a barrier for deploying new extensions is created.

2. There are some few RTP extensions that currently rely on being able to use the same SSRC in different RTP sessions:

   * XOR FEC (RFC5109)

   * RTP Retransmission in session mode (RFC4588)

   * Certain Layered Coding

3.3. SRTP

SRTP [RFC3711] is one of the most commonly used security solutions for RTP. In addition, it is the only one recommended by IETF that is integrated into RTP. This integration has several aspects that needs to be considered when designing a solution for multiplexing RTP sessions on the same lower layer transport.

Determing Crypto Context: SRTP first of all needs to know which session context a received or to-be-sent packet relates to. It also normally relies on the lower layer transport to identify the session. It uses the MKI, if present, to determine which key set is to be used. Then the SSRC and sequence number are used by most crypto suites, including the most common use of AES Counter Mode, to actually generate the correct cipher stream.

Unencrypted Headers: SRTP has chosen to leave the RTP headers and the first two 32-bit words of the first RTCP header unencrypted, to allow for both header compression and monitoring to work also in the presence of encryption. As these fields are in clear text they are used in most crypto suites for SRTP to determine how to protect or recover the plain text.

It is here important to contrast SRTP against a set of other possible protection mechanisms. DTLS, TLS, and IPsec are all protecting and encapsulating the entire RTP and RTCP packets. They don't perform any partial operations on the RTP and RTCP packets. Any change that is considered to be part of the RTP and RTCP packet is transparent to them, but possibly not to SRTP. Thus the impact on SRTP operations

must be considered when defining a mechanism.

3.4. Don't Redefine Used Bits

As the core of RTP is in use in many systems and has a really large deployment story and numerous implementations, changing any of the field definitions is highly problematic. First of all, the implementations need to change to support this new semantics. Secondly, you get a large transition issue when you have some session participants that support the new semantics and some that don't. Combing the two behaviors in the same session can force the deployment of costly and less than perfect translation devices.

3.5. Firewall Friendly

It is desirable that current firewalls will accept the solutions as normal RTP packets. However, in the authors' opinion we can't let the firewall stifle invention and evolution of the protocol. It is also necessary to be aware that a change that will make most deep inspecting firewall consider the packet as not valid RTP/RTCP will have more difficult deployment story.

3.6. Monitioring and Reporting

It is desirable that a third party monitor can still operate on the multiplexed RTP Sessions. It is however likely that they will require an update to correctly monitor and report on multiplexed RTP Sessions.

Another type of function to consider is packet sniffers and their selector filters. These may be impacted by a change of the fields. An observation is that many such systems are usually quite rapidly updated to consider new types of standardized or simply common packet formats.

3.7. Usable Also Over Multicast

It is desirable that a solution should be possible to use also when RTP and RTCP packets are sent over multicast, both Any Source Multicast (ASM) and Single Source Multicast (SSM). The reason for this requirement is to allow a system using RTP to use the same configuration regardless of the transport being done over unicast or multicast. In addition, multicast can't be claimed to have an issue with using multiple ports, as each multicast group has a complete port space scoped by address.

3.8. Incremental Deployment

A good solution has the property that in topologies that contains RTP mixers or Translators, a single session participant can enable multiplexing without having any impact on any other session participants. Thus a node should be able to take a multiplexed packet and then easily send it out with minimal or no modification on another leg of the session, where each RTP session is transported over its own lower-layer transport. It should also be as easy to do the reverse forwarding operation.

4. Possible Solutions

This section looks at a few possible solutions and discusses their feasibility.

4.1. Header Extension

One proposal is to define an RTP header extension [RFC5285] that explicitly enumerates the session identifier in each packet. This proposal has some merits regarding RTP, since it uses an existing extension mechanism; it explicitly enumerates the session allowing for third parties to associate the packet to a given RTP session; and it works with SRTP as currently defined since a header extension is by default not encrypted, and is thus readable by the receiving stack without needing to guess which session it belongs to and attempt to decrypt it. This approach does, however, conflict with the requirement from [RFC5285] that "header extensions using this specification MUST only be used for data that can be safely ignored by the recipient", since correct processing of the received packet depends on using the header extension to demultiplex it to the correct RTP session.

Using a header extension also result in the session ID is in the integrity protected part of the packet. Thus a translator between multiplexed and non-multiplexed has the options:

1. to be part of the security context to verify the field

2. to be part of the security context to verify the field and remove it before forwarding the packet

3. to be outside of the security context and leave the header extension in the packet. However, that requires successful negotiation of the header extension, but not of the functionality, with the receiving end-points.

The biggest existing hurdle for this solution is that there exist no header extension field in the RTCP packets. This requires defining a solution for RTCP that allows carrying the explicit indicator, preferably in a position that isn't encrypted by SRTCP. However, the current SRTCP definition does not offer such a position in the packet.

Modifying the RR or SR packets is possible using profile specific extensions. However, that has issues when it comes to deployability and in addition any information placed there would end up in the encrypted part.

Another alternative could be to define another RTCP packet type that only contains the common header, using the 5 bits in the first byte of the common header to carry a session id. That would allow SRTCP to work correctly as long it accepts this new packet type being the first in the packet. Allowing a non-SR/RR packet as the first packet in a compound RTCP packet is also needed if an implementation is to support Reduced Size RTCP packets [RFC5506]. The remaining downside with this is that all stack implementations supporting multiplexing would need to modify its RTCP compound packet rules to include this packet type first. Thus a translator box between supporting nodes and non-supporting nodes needs to be in the crypto context.

This solution's per packet overhead is expected to be 64-bits for RTCP. For RTP it is 64-bits if no header extension was otherwise used, and an additional 16 bits (short header), or 24 bits plus (if needed) padding to next 32-bits boundary if other header extensions are used.

4.2. Multiplexing Shim

This proposal is to prefix or postfix all RTP and RTCP packets with a session ID field. This field would be outside of the normal RTP and RTCP packets, thus having no impact on the RTP and RTCP packets and their processing. An additional step of demultiplexing processing would be added prior to RTP stack processing to determine in which RTP session context the packet shall be included. This has also no impact on SRTP/SRTCP as the shim layer would be outside of its protection context. The shim layer's session ID is however implicitly integrity protected as any error in the field will result in the packet being placed in the wrong or non-existing context, thus resulting in a integrity failure if processed by SRTP/SRTCP.

This proposal is quite simple to implement in any gateway or translating device that goes from a multiplexed to a non-multiplexed domain or vice versa, as only an additional field needs to be added to or removed from the packet.

The main downside of this proposal is that it is very likely to trigger a firewall response from any deep packet inspection device. If the field is prefixed, the RTP fields are not matching the heuristics field (unless the shim is designed to look like an RTP header, in which case the payload length is unlikely to match the expected value) and thus are likely preventing classification of the packet as an RTP packet. If it is postfixed, it is likely classified as an RTP packet but may not correctly validate if the content validation is such that the payload length is expected to match certain values. It is expected that a postfixed shim will be less problematic than a prefixed shim in this regard, but we are lacking hard data on this.

This solution's per packet overhead is 1 byte.

4.3. Single Session

Given the difficulty of multiplexing several RTP sessions onto a single lower-layer transport, it's tempting to send multiple media streams in a single RTP session. Doing this avoids the need to de-multiplex several sessions on a single transport, but at the cost of losing the RTP session as a separator for different type of streams. Lacking different RTP sessions to demultiplex incoming packets, a receiver will have to dig deeper into the packet before determining what to do with it. Care must be taken in that inspection. For example, you must be careful to ensure that each real media source uses its own SSRC in the session and that this SSRC doesn't change media type.

The loss of the RTP session as a purpose separator is likely not a big issue if the only difference between RTP Sessions is the media type. In this case, you can use the Payload Type field to identify the media type. The loss of the RTP Session functionality is more severe, however, if you actually use the RTP Session for separating different treatments, contexts etc. Then you would need additional signalling to bind the different sources to groups which can help make the necessary distinctions.

This approach has been proposed in the RTCWeb context in [I-D.lennox-rtcweb-rtp-media-type-mux] and [I-D.holmberg-mmusic-sdp-bundle-negotiation]. These drafts describe how to signal multiple media streams multiplexed into a single RTP session, and address some of the issues raised here and in Section 7.2.9 of the RTP Multiplexing Architecture [I-D.westerlund-avtcore-multiplex-architecture] draft. However, they fail to discuss maybe the largest issue with this solution: how to do incremental deployment and transition.

Many transition scenarios use an RTP translator as a gateway between a single RTP session containing multiple media types multiplexed together, and several separate RTP sessions each using a single media type. In this scenario, it is possible that a legacy device that uses one RTP session for each media type will use the same SSRC in each session. When translating these into a single RTP session, it will be necessary to rewrite one of the SSRCs, so that each stream has a unique SSRC. This SSRC translation process is straight-forward for RTP packets, but is very complex for RTCP packets. It also hinders innovation, since such a gateway will not be able to translate new RTCP extensions that it is unaware of, even if they are supported by devices on both sides of the gateway.

This method has several limitations that makes it unsuitable as general mechanism to provide multiple RTP sessions on the same lower layer transport. However, we acknowledge that there are some uses for which this method may be sufficient and which can accept the method limitations and other downsides. The RTCWEB WG has a working assumption to support this method. For more details of this method, see the relevant drafts under development.

This solution has no per packet overhead. The signalling overhead will be a different question.

4.4. Use the SRTP MKI field

This proposal is to overload the MKI SRTP/SRTCP identifier to not only identify a particular crypto context, but also identify the actual RTP Session. This clearly is a miss use of the MKI field, however it appears to be with little negative implications. SRTP already supports handling of multiple crypto contexts.

The two major downsides with this proposal is first the fact that it requires using SRTP/SRTCP to multiplex multiple sessions on a single lower layer transport. The second issue is that the session ID parameter needs to be put into the various key-management schemes and to make them understand that the reason to establish multiple crypto contexts is because they are connected to various RTP Sessions. Considering that SRTP have at least 3 used keying mechanisms, DTLS-SRTP [RFC5764], Security Descriptions [RFC4568], and MIKEY [RFC3830], this is not an insignificant amount of work.

This solution has 32-bit per packet overhead, but only if the MKI was not already used.

4.5. Use an Octet in the Padding

The basics of this proposal is to have the RTP packet and the last (required by RFC3550) RTCP packet in a compound to include padding, at least 2 bytes. One byte for the padding count (last byte) and one byte just before the padding count containing the session ID.

This proposal uses bytes to carry the session ID that have no defined value and is intended to be ignored by the receiver. From that perspective it only causes packet expansion that is supported and handled by all existing equipment. If an implementation fails to understand that it is required to interpret this padding byte to learn the session ID, it will see a mostly coherent RTP session except where SSRCs overlap or where the payload types overlap. However, reporting on the individual sources or forwarding the RTCP RR are not completely without merit.

There is one downside of this proposal and that has to do with SRTP. To be able to determine the crypto context, it is necessary to access to the encrypted payload of the packet. Thus, the only mechanism available for a receiver to solve this issue is to try the existing crypto contexts for any session on the same lower layer transport and then use the one where the packet decrypts and verifies correctly. Thus for transport flows with many crypto contexts, an attacker could simply generate packets that don’t validate to force the receiver to try all crypto contexts they have rather than immediately discard it as not matching a context. A receiver can mitigate this somewhat by using hueristics based on the RTP header fields to determine which context applies for a received packet, but this is not a complete solution.

This solution has a 16-bit per packet overhead.

4.6. Redefine the SSRC field

The Rosenberg et. al. Internet draft "Multiplexing of Real-Time Transport Protocol (RTP) Traffic for Browser based Real-Time Communications (RTC)" [I-D.rosenberg-rtcweb-rtpmux] proposed to redefine the SSRC field. This has the advantage of no packet expansion. It also looks like regular RTP. However, it has a number of implications. First of all it prevents any RTP functionality that require the same SSRC in multiple RTP sessions.

Secondly its interoperability with normal RTP is problematic. Such interoperability requires an SSRC translator function in the gateway to ensure that the SSRCs fulfill the requirements of the different domains. That translator is actually far from easy as it needs to understand the semantics of all RTP and RTCP extensions that include

SSRC/CSRC. This as it is necessary to know when a particular matching 32-bit pattern is an SSRC field and when the field is just a combination of other fields that create the same matching 32-bit pattern. Thus any future RTCP extension might not work through the translator, causing a barrier for deployment of future extensions.

This solution has no per packet overhead.

5. Recommendation

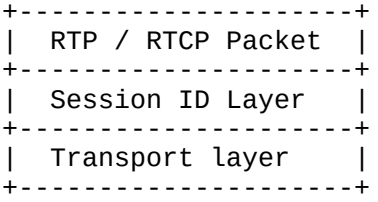
Considering these options, the authors would recommend that AVTCORE standardize a solution based on a postfixed multiplexing field, i.e. a shim approach combined with the appropriate signalling as described in Section 4.2.

6. Specification

This section contains the specification of the solution based on a SHIM, with the explicit session identifier at the end of the encapsulated payload.

6.1. Shim Layer

This solution is based on a shim layer that is inserted in the stack between the regular RTP and RTCP packets and the transport layer being used by the RTP sessions. Thus the layering looks like the following:



Stack View with Session ID SHIM

The above stack is in fact a layered one as it does allow multiple RTP Sessions to be multiplexed on top of the Session ID shim layer. This enables the example presented in Figure 1 where four sessions, S1-S4 is sent over the same Transport layer and where the Session ID layer will combine and encapsulate them with the session ID on transmission and separate and decapsulate them on reception.

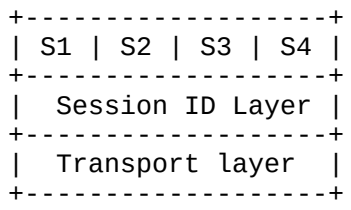
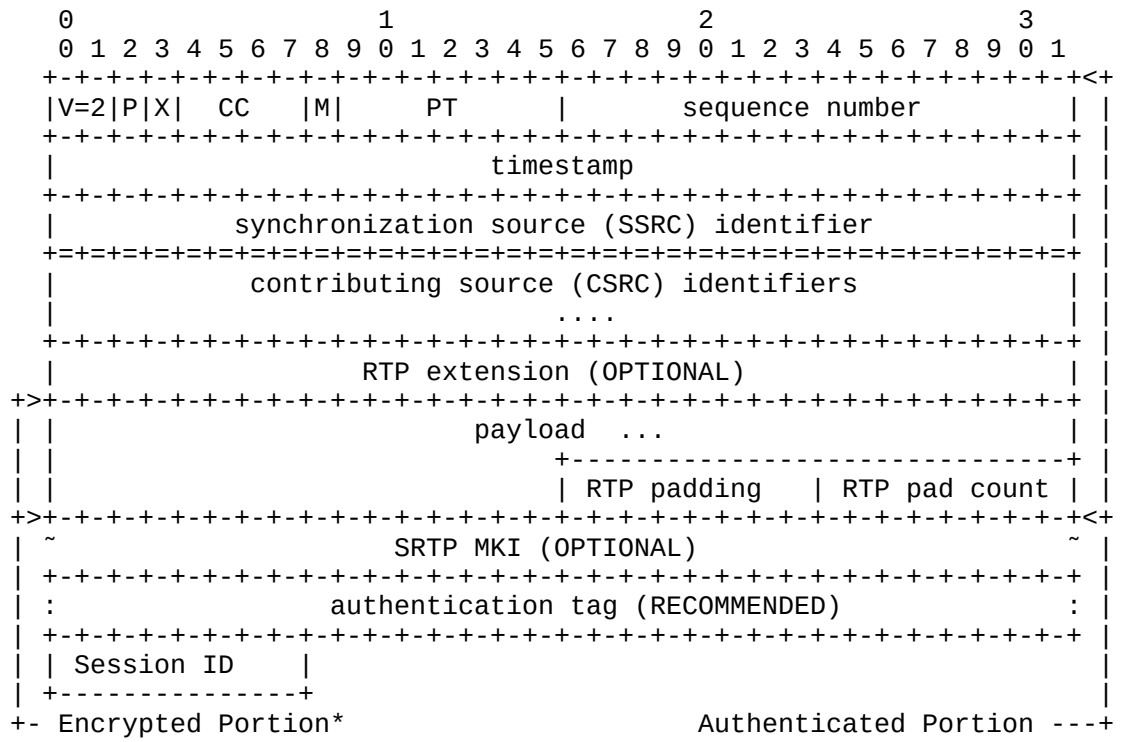


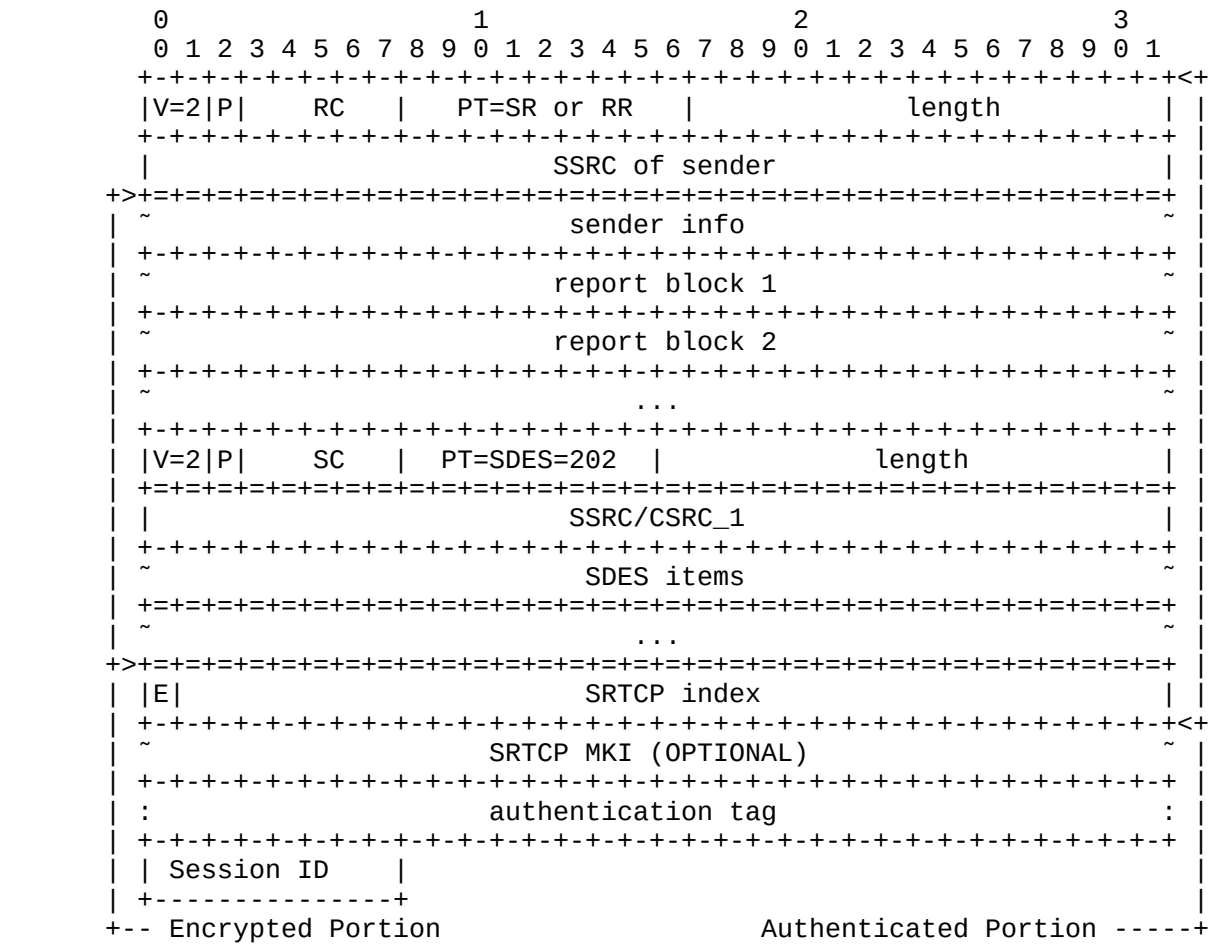
Figure 1: Multiple RTP Session On Top of Session ID Layer

The Session ID layer encapsulates one RTP or RTCP packet from a given RTP session and postfixes a one byte Session ID (SID) field to the packet. Each RTP session being multiplexed on top of a given transport layer is assigned either a single or a pair of unique SID in the range 0-255. The reason for assigning a pair of SIDs to a given RTP session are for RTP Sessions that doesn't support "Multiplexing RTP Data and Control Packets on a Single Port" [RFC5761] to still be able to use a single 5-tuple. The reasons for supporting this extra functionality is that RTP and RTCP multiplexing based on the payload type/packet type fields enforces certain restrictions on the RTP sessions. These restrictions may not be acceptable. As this solution does not have these restrictions, performing RTP and RTCP multiplexing in this way has benefits.

Each Session ID value space is scoped by the underlying transport protocol. Common transport protocols like UDP, DCCP, TCP, and SCTP can all be scoped by one or more 5-tuple (Transport protocol, source address and port, destination address and port). The case of multiple 5-tuples occur in the case of multi-unicast topologies, also called meshed multiparty RTP sessions.



SRTP Packet encapsulated by Session ID Layer



SRTCP packet encapuslated by Session ID layer

The processing in a receiver when the Session ID layer is present will be to

1. Pick up the packet from the lower layer transport

2. Inspect the SID field value

3. Strip the SID field from the packet

4. Forward it to the (S)RTP Session context identified by the SID value

6.2. Signalling

The use of the Session ID layer needs to be explicitly agreed on between the communicating parties. Each RTP Session the application uses must in addition to the regular configuration such as payload types, RTCP extension etc, have both the underlying 5-tuple (source address and port, destination address and port, and transport protocol) and the Session ID used for the particular RTP session. The signalling requirement is to assign unique Session ID values to all RTP Sessions being sent over the same 5-tuple. The same Session ID shall be used for an RTP session independently of the traffic direction. Note that nothing prevents a multi-media application from using multiple 5-tuples if desired for some reason, in which case each 5-tuple has its own session ID value space.

This section defines how to negotiate the use of the Session ID layer, using the Session Description Protocol (SDP) Offer/Answer mechanism [RFC3264]. A new media-level SDP attribute, 'session-mux-id', is defined, in order to be used with the media BUNDLE mechanism defined in [I-D.holmberg-mmusic-sdp-bundle-negotiation]. The attribute allows each media description ("m=" line) associated with a 'BUNDLE' group to form a separate RTP session.

The 'session-mux-id' attribute is included for a media description, in order to indicate the Session ID for that particular media description. Every media description that shares a common attribute value is assumed to be part of a single RTP session. An SDP Offerer MUST include the 'session-mux-id' attribute for every media description associated with a 'BUNDLE' group. If the SDP Answer does not contain 'session-mux-id' attributes, the SDP Offerer MUST NOT assume that separate RTP sessions will be used. If the SDP Answer still describes a 'BUNDLE' group, the procedures in [I-D.holmberg-mmusic-sdp-bundle-negotiation] apply.

An SDP Answerer MUST NOT include the 'session-mux-id' attribute in an SDP Answer, unless included in the SDP Offer.

The attribute has the following ABNF [RFC5234] definition.

```
Session-mux-id-attr = "a=session-mux-id:" SID *SID-prop
SID                 = SID-value / SID-pairs
SID-value           = 1*3DIGIT / "NoN"
SID-pairs           = SID-value "/" SID-value ; RTP/RTCP SIDs
SID-prop            = SP assignment-policy / prop-ext
prop-ext            = token "=" value
assignment-policy   = "policy=" ("tentative" / "fixed")
```

The following parameters MUST be configured as specified:

- RTP Profile SHOULD be the same, but MUST be compatible, like AVP and AVPF.
- RTCP bandwidth parameters are the same
- RTP Payload type values are not overlapping

In declarative SDP usage, there is clearly no method for fallback unless some other negotiation protocol is used.

The SID property "policy" is used in negotiation by an end-point to indicate if the session ID values are merely a tentative suggestion or if they must have these values. This is used when negotiating SID for multi-party RTP sessions to support shared transports such as multicast or RTP translators that are unable to produce renumbered SIDs on a per end-point basis. The normal behavior is that the offer suggest a tentative set of values, indicated by "policy=tentative". These SHOULD be accepted by the peer unless that peer negotiate session IDs on behalf of a centralized policy, in which case it MAY change the value(s) in the answer. If the offer represents a policy that does not allow changing the session ID values, it can indicate that to the answerer by setting the policy to "fixed". This enables the answering peer to either accept the value or indicate that there is a conflict in who is performing the assignment by setting the SID value to NoN (Not a Number). Offerer and answerer SHOULD always include the policy they are operating under. Thus, in case of no centralized behaviors, both offerer and answerer will indicate the tentative policy.

6.3. SRTP Key Management

Key management for SRTP do needs discussion as we do cause multiple SRTP sessions to exist on the same underlying transport flow. Thus we need to ensure that the key management mechanism still are properly associated with the SRTP session context it intends to key. To ensure that we do look at the three SRTP key management mechanism that IETF has specified, one after another.

6.3.1. Security Description

Session Description Protocol (SDP) Security Descriptions for Media Streams [RFC4568] as being based on SDP has no issue with the RTP session multiplexing on lower layer specified here. The reason is that the actual keying is done using a media level SDP attribute. Thus the attribute is already associated with a particular media description. A media description that also will have an instance of

the "a=session-mux-id" attribute carrying the SID value/pair used with this particular crypto parameters.

6.3.2. DTLS-SRTP

Datagram Transport Layer Security (DTLS) Extension to Establish Keys for the Secure Real-time Transport Protocol (SRTP) [RFC5764] is a keying mechanism that works on the media plane on the same lower layer transport that SRTP/SRTCP will be transported over. Thus each DTLS message must be associated with the SRTP and/or SRTCP flow it is keying.

The most direct solution is to use the SHIM and the SID context identifier to be applied also on DTLS packets. Thus using the same SID that is used with RTP and/or RTCP also for the DTLS message intended to key that particular SRTP and/or SRTCP flow(s).

6.3.3. MIKEY

MIKEY: Multimedia Internet KEYing [RFC3830] is a key management protocol that has several transports. In some cases it is used directly on a transport protocol such as UDP, but there is also a specification for how MIKEY is used with SDP "Key Management Extensions for Session Description Protocol (SDP) and Real Time Streaming Protocol (RTSP)" [RFC4567].

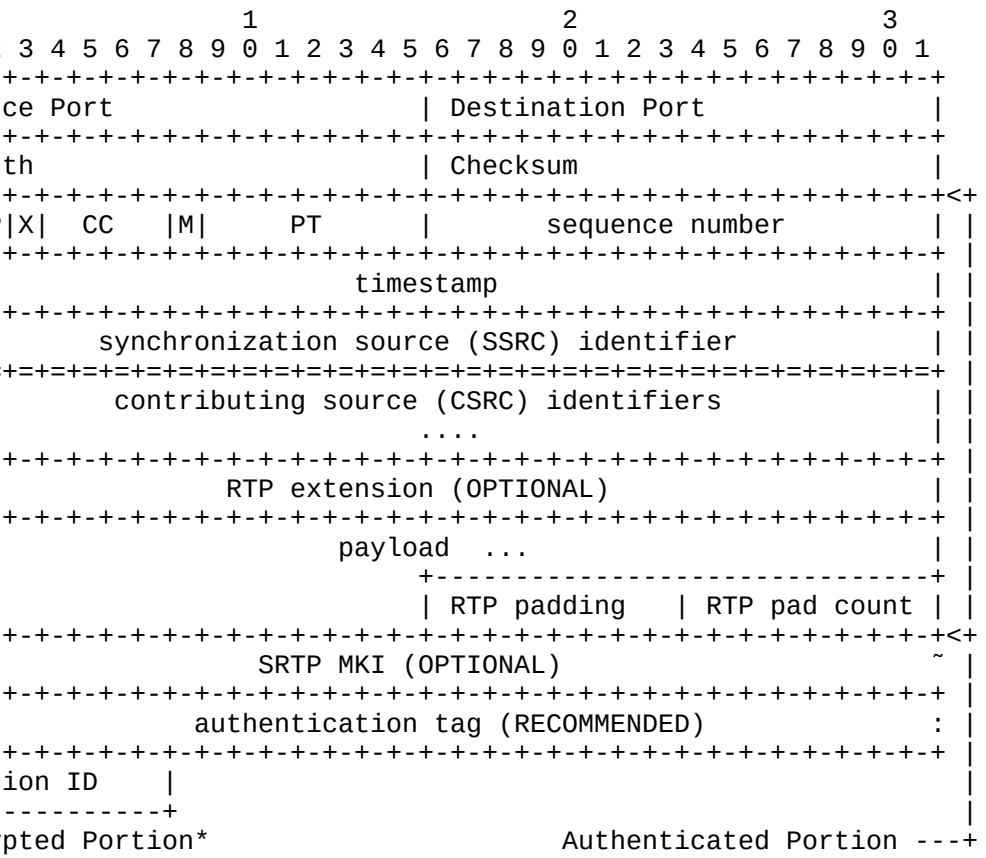
Lets start with the later, i.e. the SDP transport, which shares the properties with Security Description in that is can be associated with a particular media description in a SDP. As long as one avoids using the session level attribute one can be certain to correctly associate the key exchange with a given SRTP/SRTCP context.

It does appear that MIKEY directly over a lower layer transport protocol will have similar issues as DTLS.

6.4. Examples

6.4.1. RTP Packet with Transport Header

The below figure contains an RTP packet with SID field encapsulated by a UDP packet (added UDP header).



SRTP Packet Encapsulated by Session ID Layer

6.4.2. SDP Offer/Answer example

This section contains SDP offer/answer examples. First one example of successful BUNDLEing, and then two where fallback occurs.

In the below SDP offer, one audio and one video is being offered. The audio is using SID 0, and the video is using SID 1 to indicate that they are different RTP sessions despite being offered over the same 5-tuple.

```
v=0
o=alice 2890844526 2890844526 IN IP4 atlanta.example.com
s=
c=IN IP4 atlanta.example.com
t=0 0
a=group:BUNDLE foo bar
m=audio 10000 RTP/AVP 0 8 97
b=AS:200
a=mid:foo
a=session-mxu-id:0 policy=suggest
a=rtpmap:0 PCMU/8000
a=rtpmap:8 PCMA/8000
a=rtpmap:97 iLBC/8000
m=video 10000 RTP/AVP 31 32
b=AS:1000
a=mid:bar
a=session-mxu-id:1 policy=suggest
a=rtpmap:31 H261/90000
a=rtpmap:32 MPV/90000
```

The SDP answer from an end-point that supports this BUNDLEing:

```
v=0
o=bob 2808844564 2808844564 IN IP4 biloxi.example.com
s=
c=IN IP4 biloxi.example.com
t=0 0
a=group:BUNDLE foo bar
m=audio 20000 RTP/AVP 0
b=AS:200
a=mid:foo
a=session-mux-id:0 policy=suggest
a=rtpmap:0 PCMU/8000
m=video 20000 RTP/AVP 32
b=AS:1000
a=mid:bar
a=session-mux-id:1 policy=suggest
a=rtpmap:32 MPV/90000
```

The SDP answer from an end-point that does not support this BUNDLEing or the general signalling of [I-D.holmberg-mmusic-sdp-bundle-negotiation].

```
v=0
o=bob 2808844564 2808844564 IN IP4 biloxi.example.com
s=
c=IN IP4 biloxi.example.com
t=0 0
m=audio 20000 RTP/AVP 0
b=AS:200
a=rtpmap:0 PCMU/8000
m=video 30000 RTP/AVP 32
b=AS:1000
a=rtpmap:32 MPV/90000
```

The SDP answer of a client supporting [I-D.holmberg-mmusic-sdp-bundle-negotiation] but not this BUNDLEing would look like this:

```
v=0
o=bob 2808844564 2808844564 IN IP4 biloxi.example.com
s=
c=IN IP4 biloxi.example.com
t=0 0
a=group:BUNDLE foo bar
m=audio 20000 RTP/AVP 0
a=mid:foo
b=AS:200
a=rtpmap:0 PCMU/8000
m=video 20000 RTP/AVP 32
a=mid:bar
b=AS:1000
a=rtpmap:32 MPV/90000
```

In this last case, the result is a sing RTP session with both media types being established. If that isn't supported or desired, the offerer will have to either re-invite without the BUNDLE grouping to force different 5-tuples, or simply terminate the session.

7. Open Issues

This is the first version of this draft. It will obviously have a number of open issues. This section contains a list of open issues where the author desires some input.

1. Should RTP and RTCP multiplexing without RFC 5761 support be included?

8. IANA Considerations

This document request the registration of one SDP attribute. Details of the registration to be filled in.

9. Security Considerations

The security properties of the Session ID layer is depending on what mechanism is used to protect the RTP and RTCP packets of a given RTP session. If IPsec or transport layer security solutions such as DTLS or TLS are being used then both the encapsulated RTP/RTCP packets and the session ID layer will be protected by that security mechanism. Thus potentially providing both confidentiality, integrity and source authentication. If SRTP is used, the session ID layer will not be directly protected by SRTP. However, it will be implicitly integrity protected (assuming the RTP/RTCP packet is integrity protected) as the only function of the field is to identify the session context. Thus any modification of the SID field will attempt to retrieve the wrong SRTP crypto context. If that retrieval fails, the packet will be anyway be discarded. If it is successful, the context will not lead to successful verification of the packet.

10. Acknowledgements

This document is based on the input from various people, especially in the context of the RTCWEB discussion of how to use only a single lower layer transport. The RTP and RTCP packet figures are borrowed from RFC3711. The SDP example is extended from the one present in [I-D.holmberg-mmusic-sdp-bundle-negotiation]. The authors would like to thank Christer Holmberg for assistance in utilizing the BUNDLE grouping mechanism.

The proposal in Section 4.5 is original suggested by Colin Perkins. The idea in Section 4.6 is from an Internet Draft [I-D.rosenberg-rtcweb-rtpmux] written by Jonathan Rosenberg et. al. The proposal in Section 4.3 is a result of discussion by a group of people at IETF meeting #81 in Quebec.

11. References

11.1. Normative References

[I-D.holmberg-mmusic-sdp-bundle-negotiation]
Holmberg, C. and H. Alvestrand, "Multiplexing Negotiation Using Session Description Protocol (SDP) Port Numbers",

draft-holmberg-mmusic-sdp-bundle-negotiation-00 (work in progress), October 2011.

[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.

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

[RFC5234] Crocker, D. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", STD 68, RFC 5234, January 2008.

11.2. Informational References

[I-D.lennox-rtcweb-rtp-media-type-mux]
Lennox, J. and J. Rosenberg, "Multiplexing Multiple Media Types In a Single Real-Time Transport Protocol (RTP) Session", draft-lennox-rtcweb-rtp-media-type-mux-00 (work in progress), October 2011.

[I-D.rosenberg-rtcweb-rtpmux]
Rosenberg, J., Jennings, C., Peterson, J., Kaufman, M., Rescorla, E., and T. Terriberry, "Multiplexing of Real-Time Transport Protocol (RTP) Traffic for Browser based Real-Time Communications (RTC)", draft-rosenberg-rtcweb-rtpmux-00 (work in progress), July 2011.

[I-D.westerlund-avtcore-multiplex-architecture]
Westerlund, M., Burman, B., and C. Perkins, "RTP Multiplexing Architecture", draft-westerlund-avtcore-multiplex-architecture-00 (work in progress), October 2011.

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

[RFC3830] Arkko, J., Carrara, E., Lindholm, F., Naslund, M., and K. Norrman, "MIKEY: Multimedia Internet KEYing", RFC 3830, August 2004.

[RFC4567] Arkko, J., Lindholm, F., Naslund, M., Norrman, K., and E.

Carrara, "Key Management Extensions for Session Description Protocol (SDP) and Real Time Streaming Protocol (RTSP)", RFC 4567, July 2006.

[RFC4568] Andreasen, F., Baugher, M., and D. Wing, "Session Description Protocol (SDP) Security Descriptions for Media Streams", RFC 4568, July 2006.

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

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

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

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

Authors' Addresses

Magnus Westerlund
Ericsson
Farogatan 6
SE-164 80 Kista
Sweden

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

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

Email: csp@csperkins.org