W3C ORTC Community Group Meeting

Jan 27, 2015 9:00AM-10:30AM PDT

W3C CG IPR Policy

- See the Community License Agreement for details.
- Goals are
  - Enable rapid spec development
  - Safe to implement via royalty-free commitments from participants+employers
  - Comfort for committers by limiting scope to OWN contributions
  - Transparency about who is making commitments
- How it works in practice
  - Anyone can post to public-ortc@w3.org
  - CG members who have signed CLA can post to public-ortc-contrib
  - Editor should ensure that spec includes only “contributions”, CC-ing public-ortc-contrib makes that easier on the editor.

Welcome!

- Welcome to the 7th meeting of the W3C ORTC Community Group! Now 91 members!
- During this meeting, we hope to:
  - Bring you up to date on the status of the ORTC specification
  - Make progress on some outstanding issues
  - Organize/plan for implementation feedback

About this Virtual Meeting

Information on the meeting:

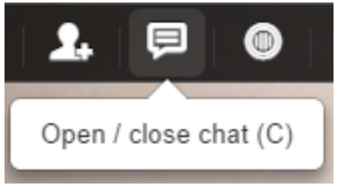
- ~~Jitsi Meet Link~~ Hangouts Link
- Link to Slides has been published on CG home page & ORTC.org
- Etherpad scribe?
- The meeting is being recorded.

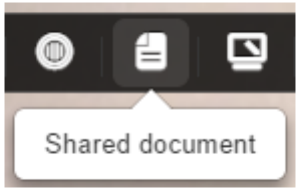
CG Chair

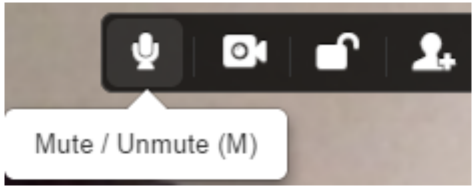
Erik Lagerway, Co-founder, Hookflash, erik@hookflash.com

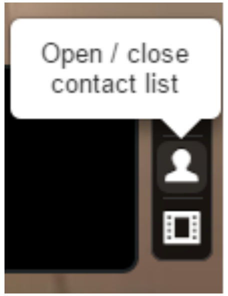
About Jitsi Meet

- Today's meeting is being hosted on an experimental server provided by Jitsi.org. Thank you! Thank you, CG participants for tolerating potential bugs! If something goes wrong, please reload the page (control-F5 or cmd-shift-R).
- To accommodate as many ORTC CG participants as possible, Voice Activity Detection ('Last N') has been enabled.
- Your video will only be sent to others if you leave it enabled and if you have spoken recently. For best results, please mute yourself if you are not speaking!
- Simulcast (but not SVC) will be enabled today to reduce bandwidth consumption.
- To allow people to see you in the contact list when you aren't speaking, please click on the Open/close chat icon and input your name. To see a list of participants, click on the contact list icon.
- To edit the Etherpad (for scribes) please click on the 'Shared Document' icon.









W3C ORTC Community Group Basics

- W3C ORTC CG website:
  - http://www.w3.org/community/ortc/
- Public mailing list: public-ortc@w3.org
  - Join Here - link on the right hand side
  - Non-members can post to this list.
  - Non-member contributions are problematic.
- Contributor’s mailing list: public-ortc-contrib@w3.org
  - Join Here - link on the right hand side
  - Members only, preferred list for contributions to the specification.

Associated Sites

- ORTC developer website: http://ortc.org/
  - Editor's drafts, pointers to github repos, etc.
- ORTC API Issues List: https://github.com/openpeer/ortc/issues?state=open

Editor’s Draft Changes

22 January 2015 Editor’s draft:

- http://ortc.org/wp-content/uploads/2015/01/ortc.html

Changes from the October 14 Editor’s Draft:

WebRTC 1.0 compatibility

1. Statistics API Update (Issue 85)
2. H.264 parameters update (Issue 158)
3. Support for *maxptime* (Issue 160)
4. RTCRtpUnhandledEvent update (Issue 163)
5. Support for RTCIceGatherer.state (Issue 164)

maxptime (Issue 160)

The SDP specification (RFC 4566) defines *ptime* and *maxptime* media-level attributes.

Support for *maxptime* is required in WebRTC 1.0 API. From https://tools.ietf.org/html/draft-ietf-rtcweb-jsep Section 5.2.1 (Initial Offers):

“Each m= section MUST include the following attribute lines:...
o If this m= section is for media with configurable frame sizes, e.g. audio, an "a=maxptime" line, indicating the smallest of the maximum supported frame sizes out of all codecs included above, as specified in [RFC4566], Section 6.”

Question: How does the ORTC API handle ptime/maxptime capabilities and settings?

Proposal for maxptime

```
partial dictionary RTCRtpCodecCapability {
unsigned long maxptime;
}

partial dictionary RTCRtpCodecParameters {
unsigned long maxptime;
}
```

By including a *maxptime* attribute for each codec, the largest *maxptime* among all codecs can be computed, for inclusion in the a=maxptime line, as required by JSEP.

Proposal for ptime

Define ptime capabilities and settings for each codec. For Opus and G.711, a summary is available here:
https://www.ietf.org/mail-archive/web/rtcweb/current/msg11485.html

Issue noted by Magnus Westerlund:

“The fact that ptime only can indicate a single rate becomes a potential issue as you can't determine a remote peer preferences for other rates, if an WebRTC endpoint likes to modify its rate due to congestion control reasons. Changing the packetization rate is one of the tools that give a most significant bit-rate change for audio, and it can even be applied without changing the encoding rate, something crucial for doing any bit-rate adaptation for G.711.”

Implication: ptime *capabilities* describe supported values; *settings*, the value in use. Exchanging ptime capabilities enables rate adaptation (adjustment to ptime settings).

Unhandled RTP streams (Issue 163)

A pull request has been submitted to add a RTCMediaDiscardedEvent to WebRTC 1.0, to indicate reception of "packets having an unknown SSRC, payload type, or any other error that makes it impossible to attribute an RTP packet to a specific MediaStreamTrack."

Pull request: https://github.com/w3c/webrtc-pc/pull/29

ORTC API Section 8.5 defines RTCRtpUnhandledEvent with the same attributes (ssrc, payloadType, mid).

Question: what is the difference?

Answer: In ORTC API, there is no *requirement* that unhandled RTP packets be discarded. Modest buffering is permitted.

Editor’s Draft Changes (cont’d)

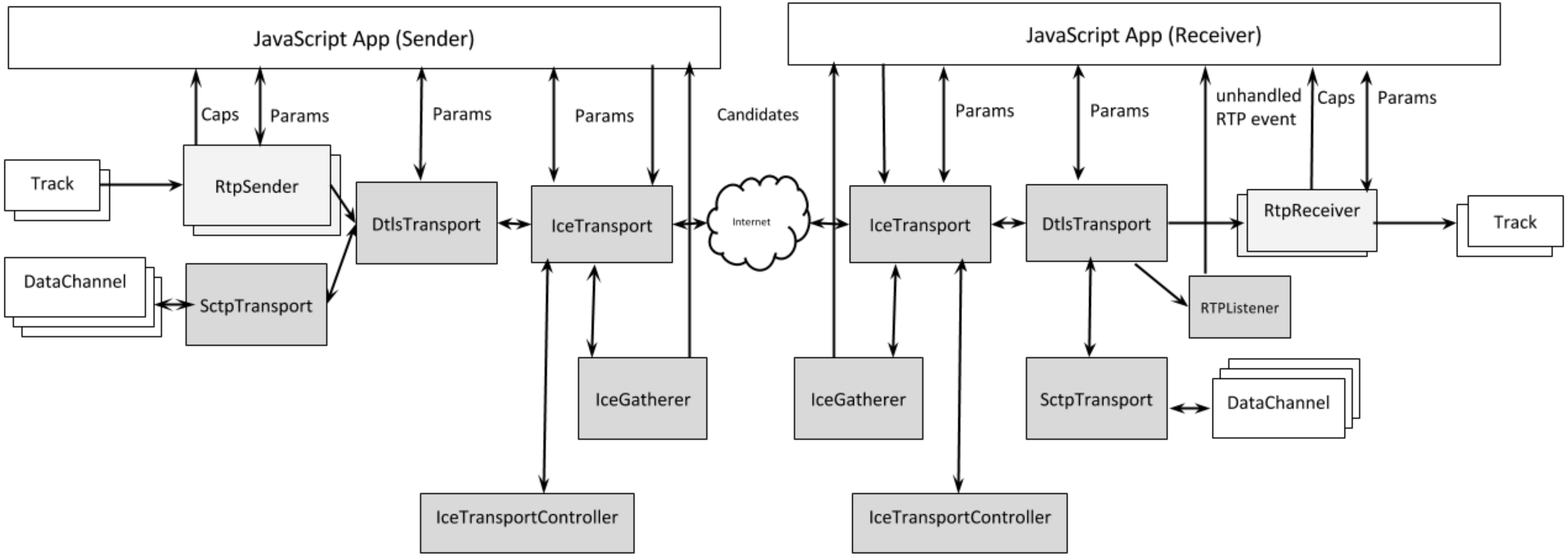
Editorial

6. Changed 'RTCIceTransportEvent' to 'RTCIceGathererEvent' as noted in: Issue 161

Clarifications

7. Updated Section 8.3 to reflect proposed matching rules, reflecting: Issue 48
8. Update on 'automatic' use of scalable video coding, as noted in: Issue 156
9. Update to the 'Big Picture', as noted in: Issue 159
10. Revised the text relating to RTCIceTransport.start() as noted in: Issue 166
11. Added text relating to DTLS interoperability with WebRTC 1.0, as noted in: Issue 167
12. Revised the text relating to RTCDtlsTransport.start() as noted in: Issue 168
13. Clarified handling of incoming connectivity checks by the RTCIceGatherer as noted in: Issue 170
14. Added reference to ICE consent document, as noted in Issue 171.

Updated Big Picture (Issue 159)



Questions for the CG

- Is the CG generally OK with the direction in which the Editor’s draft is headed?
- Do you have questions about general aspects of the spec?

For Discussion Today

- ICE related issues (164, 166, 170, 171)
- DTLS demuxing upon second start()
- Matching rules

Coming Attractions

- DTMF (if updated in WebRTC 1.0)
- IdP (if updated in WebRTC 1.0)
- Data Channel (if updated in WebRTC 1.0)

IceGatherer state / candidate pruning (Issues 164)

In clarification of the IceGatherer.state, pruning of local candidates within an IceTransport has no affect on the local candidates available with the associated IceGatherer.

If this were not true then any forked offer would require an IceGatherer per potential forked answer (as pruning candidates within one fork would affect other forks thus making ICE negotiation impossible for normal forking).

Ice re-start (Issues 166)

Calling IceTransport.start() with an new IceGatherer flushes all remote candidates, adopts all non-pruned local candidates available from the IceGather and uses the new ufrag/password.

Calling IceTransport.start() with the existing IceGatherer flushes all remote candidates, refreshes all non-pruned local candidates available from the IceGather and uses the existing ufrag/password.

In both cases, the existing functional nominated pair remains in use until a new nominated candidate pair is found from the new set of local/remote candidate pairs. Thus "connected" remains the state of the IceTransport if the transport was already "connected".

IceGatherer respond to connectivity checks? (Issues 170)

In theory, the IceGatherer has all information required to respond to connectivity checks prior to calling IceTransport.start() but cannot issue connectivity checks.

Currently the IceGatherer **may** respond to connectivity checks and pass the pre-answered checks to the IceTransport upon start() being called (so that appropriate peer reflexive outgoing checks can occur).

Two issues:

- A **must** requirement might be too complicated a requirement for ORTC implementations but a **may** requirement can lead to small inconsistencies in ICE functionality across implementations.
- A time lag can occur from issuing the connectivity response to the "start()" being called thus affecting the scheduling of RFC 5245 Section 2.2: “As an optimization, as soon as R gets L's check message, R schedules a connectivity check message to be sent to L on the same candidate pair.”

ICE consent failure (Issues 171)

What happens if ICE consent fails?

If a consent failure occurs, then the tested pair is considered invalid. If the pair is the only warm nominated ICE pair then the IceTransport.state will transition to "disconnected" (although the "disconnected" state may resolve itself if other candidate pairs prove available).

DTLS Transport De-Multiplexing (Issue 168)

Two DtlsTransport objects (A & B) are constructed using the same IceTransport object. dtlsTransportB.start() is called before dtlsTransportA.stop().

Problem: It is not possible to demultiplex DtlsTransport packets between objects A & B.

Question: What should happen in this case? Alternatives:

1. dtlsTransportB.start() throws an *InvalidState* exception.
2. dtlsTransportA.stop() is implicitly called before allowing dtlsTransportB.start() to proceed.

Matching Rules

WARNING: Matching rules (Section 8.3) have been known to cause headaches and fatigue. We will only touch on a limited use cases today out of concern for the health of ORTC CG participants.

If symptoms persist, please post questions, concerns, suggestions to the mailing list.

Matching Rules (Issue 48)

Matching rules route incoming RTP packets to Receiver objects (which emit a single MediaStreamTrack) and where no route is found an RtpUnhandledEvent is fired.

Peter Thatcher's proposal is now included in Section 8.3. Errors are the responsibility of the editors.

Three tables are used to determine which RtpReceiver an RTP packet is routed to:

1. ssrc_table[ssrc] → *receiver*, which maps the SSRC field in incoming packets to **RtpReceiver** objects;
2. muxId_table[muxId] → *receiver*, which maps values of the MID RTP header extension in incoming packets to **RtpReceiver** objects;
3. pt_table[pt] → *receiver*, which maps payload type field values in incoming packets to **RtpReceiver** objects.

Matching Rules (cont'd)

Table entries are created or revised whenever RtpReceiver.receiver() is called and table entries are removed when RtpReceiver.stop() is called.

Once the RTP packet is routed to the correct RtpReceiver object, the receiver object maintains its own mappings of SSRCs or PTs (Payload Types) to decode the RTP using the correct corresponding RtpEncodingParameters.

Matching Rules (cont’d)

RTP packets are first checked in this order:

- ssrc_table[packet.ssrc]
- muxId_table[packet.muxId]
- pt_table[packet.pt]

If a muxId_table[packet.muxId] is found then an ssrc_table[packet.ssrc] → *receiver* entry is created.

If pt_table[packet.pt] is found then ssrc_table[packet.ssrc] → *receiver* entry is created and pt_table[packet.pt] → *receiver is removed.*

Matching Rules (cont’d)

Questions:

1. What is the impact of this rule in common use case(s)?

2. Should a pt_table[packet.pt] → *receiver* be removed when a match is found? (i.e. causes an RtpReceiver to become "consumed" once only)

3. If pt_table[packet.pt] → *reciever* is removed, should only pt_table[packet.pt] be removed or *all* entries pt_table[pt] → *reciever* to the same receiver be removed?

Use Case: Distinct Audio SSRCs

Alice constructs an RtpReceiver with Opus as PT 101. Bob creates an RtpSender with PT 101 and SSRC 5000 and streams Opus. Bob creates a secondary RtpSender with PT 101 and SSRC 6000 and streams Opus.

If the pt_table[packet.pt] mapping to the first receiver is NOT removed upon receiving the first packet.pt then the second Opus stream with PT 101 will be confused with the first Opus stream and all packets from two SSRCs will go to the same RtpReceiver.

To enable this use case, a receiver would

1. Construct a distinct RtpReceiver object with a distinct PT per RtpReceiver even though the SSRCs are unique for the same codec.
2. RtpSender would have to be configured to use a different PT per RtpSender for the same codec even though the SSRC was already distinct.

Should an ssrc_table[pt.ssrc] → *receiver* be created and the pt_table[packet.pt] removed upon receiving packets to allow distinct SSRCs to fire unhandled events? Is this scenario legal in SDP?

Use Case: Audio Codec Switching

When multiple audio codecs are specified in a SIP Offer and Answer, RFC 3264 Section 5.1 says:
"If multiple formats are listed, it means that the offerer is capable of making use of any of those formats during the session. In other words, the answerer MAY change formats in the middle of the session, making use of any of the formats listed, without sending a new offer."

To enable this use case, a receiver might:

1. Construct a distinct RtpReceiver object for each of the payload types in the Answer OR
2. Construct a single RtpReceiver object, specifying all of the payload types in the Answer.

Depending on the matching rules, these approaches can behave differently.

Audio Codec Switching (cont'd)

1. Let us assume that we have a distinct RtpReceiver object for each of the payload types in the Answer (approach 1).

a. The Answerer starts by sending with the preferred codec (e.g. Opus) using an initial SSRC.

b. Initial RTP packets match the Opus payload type so packet.ssrc is inserted into the ssrc_table and the Opus PT is removed from pt_table (though not from receiver_table).

c. Answerer switches to G.711. If it does not change the SSRC, packets using the new codec are still routed to the initial receiver (and are decoded properly). If the SSRC changes, packets are routed to a distinct RtpReceiver object specifying the G.711 payload type, and the new packet.ssrc is inserted into the ssrc_table and the G.711 PT is removed from the pt_table.

d. The audio tracks from the two RtpReceiver objects are mixed by the browser.

Audio Codec Switching (cont'd)

2. What If there had been only one RtpReceiver object with entries for both Opus and G.711?

a. If <u>only</u> an individual pt_table[packet.pt] entry is removed, each time an SSRC latches when each PT is received and any packets with the same SSRC continue to be routed to correct *receiver* even if the G711 PT is received. But if another RtpSender uses the the G711 PT with a different SSRC then the initial Opus receiver will receive both the Opus RTP traffic from the first RtpSender and the G711 RTP traffic from the second RtpSender and decoding will be confused.

b. If <u>all</u> pt_table[pt] → receiver entries are removed when Opus is received then when a G.711 packet is received from a different SSRC then an RtpUnhandledEvent will correctly be fired, and a distinct RtpReceiver object will need to be constructed.

Question for Community Group:

Are there any comments regarding known deficiencies of the ORTC API at this time?

Good time to speak before implementation is too far along!

Organization / Call for implementation feedback

Mobile C++ ORTC implementation:
https://github.com/openpeer/ortc-lib

ORTC JS "shims" (i.e. downshim and upshim to / from WebRTC 1.0)
https://github.com/openpeer/ortc-js-shim

ORTC specification and createParams / filterParams replacement equivalency:
https://github.com/openpeer/ortc

ORTC Node JS implementations:
https://github.com/openpeer/ortc-node

Browser Implementations:
Requested at this time (status.modern.ie lists ORTC as "Under Consideration")

ORTC Implementation Volunteers

If you would like to participate in coding any of the ORTC open source projects then:

1. Join ORTC Community Group
2. Join developer mailing list / group
   http://ortc.org/dev
3. Start helping!

Thank you

<u>Special thanks to:</u>

Emil Ivov - Jitsi

Bernard Aboba - Microsoft

Michael Champion - MS Open Tech

Justin Uberti - Google

Peter Thatcher - Google

Robin Raymond - Hookflash

Erik Lagerway - Hookflash

For More Information

ORTC Community Group

http://www.w3.org/community/ortc/

ORTC Developer Website

http://ortc.org