W3C ORTC Community Group Meeting

July 20, 2014 10:00am-11:30am 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
  - 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 5th meeting of the W3C ORTC Community Group!
- During this meeting, we hope to:
  - Bring you up to date on the status of the ORTC specification
  - Make progress on some outstanding issues
  - Current plan / dependencies on WebRTC 1.0
  - Proposals: SS/MS, getParameters, ICE restart, CNAME
  - Organize/plan for implementation feedback

About this Virtual Meeting

Information on the meeting

- Hangout on Air Link (broadcasted publicly & recorded)
- Link to Slides has been published on CG home page & ORTC.org
- Scribe?

CG Chair

Robin Raymond, Chief Architect - Hookflash Inc.

robin@hookflash.com

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

15 July 2014 Editor’s draft:

- http://ortc.org/wp-content/uploads/2014/07/ortc.html

Changes since 16 June 2014 Editor’s draft:

Editorial

1. Fixed WebIDL issues noted in Issue 97
2. Addressed NITs described in Issue 99
3. Fixed 'Big Picture' issues described in Issue 105
4. Separated references into Normative and Informative, as noted in Issue 133

WebRTC 1.0 compatibility

5. Added section on WebRTC 1.0 compatibility issues, responding to Issue 66
6. Added Identity support, as described in Issue 78
7. Reworked getStats method, as described in Issue 85
8. Addressed WebRTC 1.0 Data Channel compatibility issue described in Issue 111
9. Addressed RTP/RTCP non-mux issues with IdP as described in Issue 114
10. Added explanation of Voice Activity Detection (VAD), responding to Issue 129

Editor's Draft Changes (cont'd)

Scalable Video Coding

1. Removed quality scalability capabilities and parameters, as described in Issue 109
2. Added scalability examples as requested in Issue 110
3. Added layering diagrams as requested in Issue 117
4. Clarified the meaning of maxTemporalLayers and maxSpatialLayers, as noted in Issue 130

ICE

1. Removed ICE restart method described in Issue 93
2. Moved onerror from the RTCIceTransport object to the RTCIceListener object as described in Issue 121
3. Addressed ICE terminology issues, as described in Issue 132
4. ICE transport issues fixed as described in Issue 101
5. ICE transport controller fixes made as described in Issue 102

Editor’s Draft Changes (cont’d)

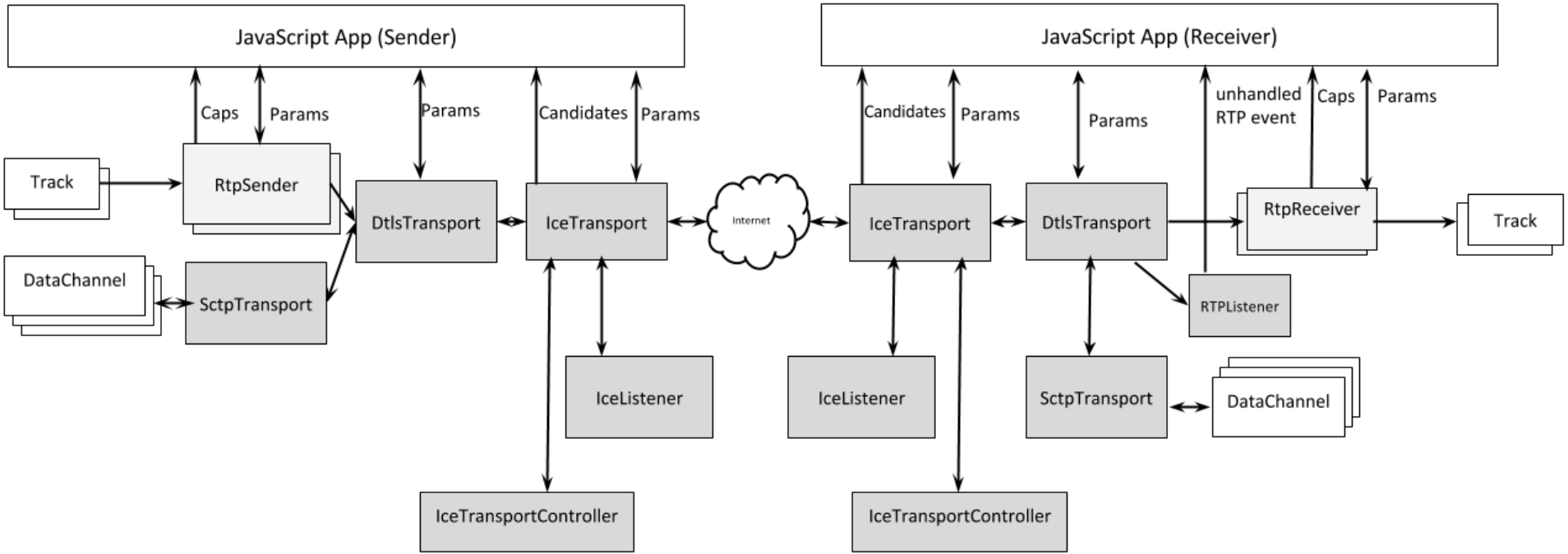
RTCRtpParameters

1. Addressed CNAME and synchronization context issues described in Issue 94
2. Sender and Receiver object fixes made as described in Issue 103
3. Fixed RTCRtpEncodingParameter default issues described in Issue 104
4. Fixed RTCRtpParameter default issues described in Issue 106
5. Removed header extensions from RTCRtpCodecParameters as described in Issue 113
6. Added getParameter methods to RTCRtpSender and RTCRtpReceiver objects, as described in Issue 116
7. Added a typedef for payload type, as described in Issue 118
8. Added RFC 6051 to the list of header extensions and removed RFC 5450, as noted in Issue 131
9. VP8, Opus and H.264 Capabilities and Parameters added, as noted in Issues 125, 126, 127

Other

10. DTLS transport issues fixed as described in Issue 100

Updated Big Picture (105)



Normative vs. Informative (131,133)

Denoted as Informative references that are:

- Informative in WebRTC 1.0 API;
- Not referenced normatively in any IETF RTCWEB WG work item;
- Are no better than OPTIONAL in any IETF RTCWEB WG work item.

Notes:

- RFC 3389 (CN) not mentioned in any RTCWEB document.
- Application-ID not mentioned in any RTCWEB document (discussion of removal from BUNDLE on MMUSIC WG list).
- (Still) No FEC recommendations in the latest RTP-Usage draft.
- RFC 5450 (Transmission Time Offsets) not mentioned in RTP-Usage
- RFC 6051 (Rapid Synchronization) mandated in RTP-usage but not in the list of header extensions

IdP Support (78)

```
[Constructor(RTCDtlsTransport transport)]
interface RTCIdentity {
    readonly    attribute RTCIdentityAssertion? peerIdentity;
    readonly    attribute RTCDtlsTransport      transport;
    Promise<DOMString>            getIdentityAssertion (DOMString provider, optional
DOMString protocol = "default", optional DOMString username);
    Promise<RTCIdentityAssertion> setIdentityAssertion (DOMString assertion);
};
```

- RTCIdentity object constructed from an RTCDtlsTransport.
- getIdentityAssertion(*provider, protocol, username*) method combines actions of setIdentityProvider(*provider, protocol, username*) and getIdentityAssertion() from WebRTC 1.0 API.
- setIdentityAssertion(*assertion*) validates the remote peer assertion.
- RTCIdentityError returned if Promise is rejected.

RTP and RTCP non-mux with IdP (114)

- If RTP and RTCP are not multiplexed and utilize distinct RTCDtlsTransports, then you can have an RTCIdentity object RTP and another for RTCP.
- For RTP and RTCP, RTCIdentity.getIdentityAssertion() can be called with different values for *provider, protocol, username.*
- RTCIdentity.setIdentityAssertion() could return a different assertion for RTP and RTCP.

Proposed Resolution

Add the following NOTEs:

- Obtaining an assertion: While it is possible for getIdentityAssertion() to be called with different values of provider, protocol and username for the RTP and RTCP RTCIdentity objects, application developers desiring backward compatibility with WebRTC 1.0 are strongly discouraged from doing so, since this is likely to result in an error.
  - TODO: Browser should prevent this from working.
- Validation: Where RTP and RTCP are not multiplexed, it is possible that the assertions for both the RTP and RTCP will be validated, but that the identities will not be equivalent. For applications requiring backward compatibility with WebRTC 1.0, this must be considered an error. However, if backward compatibility with WebRTC 1.0 is not required the application may consider an alternative, such as ignoring the RTCP identity assertion.

Voice Activity Detection (129)

- What is the equivalent of RTCOfferOptions. voiceActivityDetection in ORTC API?

Added Section 16.1 with the following text:

Within ORTC API, equivalent behavior can be obtained by configuring the Comfort Noise codec for use within the RTCRtpParameters object, or configuring a codec with built-in support for Comfort Noise (such as Opus) to enable comfort noise.

Should we add a knob to RTCRtpEncodingParameters to indicate that VAD is desired for a given encoding?

CNAME (94)

```
dictionary RTCRtcpParameters {
    unsigned long ssrc;
    DOMString     cname;
    boolean       reducedSize = false;
    boolean       mux = true;
};
```

Operation:

- Developer must fill CNAME value
- Developer cannot get "parameters" on RTCRtpSender/Receiver to learn value

Simulcast and SVC Examples (110, 117)

Added examples and layer diagrams for:

- Temporal Scalability (Section 9.9.2)
- Spatial Simulcast (Section 9.9.3)
- Spatial Scalability (Section 9.9.4)

Layering Capabilities (109, 130)

RTCRtpCodecCapability has maxTemporalLayers, maxSpatialLayers and maxQualityLayers to indicate codec SVC capabilities.

- Is there a capability for the maximum number of simulcast layers? Answer: no
- Will implementations support Quality scalability?
  - Proposal: Remove maxQualityLayers
- How does maxSpatialLayers = 0 differ from 1?
  - Proposal: Make the default to be 1; 0 is illegal.

Codec Capabilities and Parameters (125,126, 127)

Added Capabilities for the Opus, VP8 and H.264 codecs:

VP8

max-fr unsigned long

This capability indicates the maximum frame rate in frames per second that the decoder is capable of decoding.

max-fs

unsigned long long

This capability indicates the maximum frame size in macroblocks that the decoder is capable of decoding.

H.264

max-recv-level unsigned long

Indicates the highest level a receiver supports.

packetization-mode unsigned short

An integer in the range of 0 to 2 which indicates which packetization-mode this implementation supports.

Receiver codec capabilities become Sender codec parameters;

Sender codec capabilities become Receiver codec parameters.

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?

Coming Attractions

- RTCRtpListener behaviour / latching rules
- RTCRtpReceiver behaviour / latching rules
- DTMF (if updated in WebRTC 1.0)
- Stats (if updated in WebRTC 1.0)
- IdP (if updated WebRTC 1.0)
- Data Channel (if updated WebRTC 1.0)

Issues For Discussion Today

- ICE restart
- ICE gather
- ICE states
- RTCRtpParameter handling
- What does encoding parameter "scale" value mean?
- RTCRtpSender/Receiver send() after send(), receive() after receive()

ICE Restart - What does it mean?

RFC 5245 9.1.1.1:

"To restart ICE, an agent MUST change both the ice-pwd and the ice-ufrag for the media stream in an offer."

Why? RFC examples are:

- changing target of media stream
- agent is changing implementation level

SIP ICE Restart + Trickle ICE Means?

draft-ietf-mmusic-rfc5245bis-02 removed 9.1.1.1 but SIP ICE "restart" will find a new RFC home.

ICE Restart as defined for SIP has specific implications:

- ufrag-password on ICE must change
- all remote candidates are flushed
- local candidates are gathered again
- affects dependency components (like RTCP)
- may want to perform on all related transports at once

ICE Restart

```
partial interface RTCIceTransportController {
    void                        restart ();
};
```

If needed, proposal:

- Add method "restart"
- Restarts ICE to gathering state and flushes all remote candidates
- Affects all transports within RTCIceTransportController

Outstanding Question:

- Does this change the RTCIceListener? (No - just ufrag/password)

ICE Gather Policy Change

Two important operations:

- from more restrictive to less restrictive
- from less restrictive go more restrictive

E.g. change policy from "nohost" to "all" or vice versa.

ICE Gather Policy: To Less Restrictive

If new candidates are allowed because gather policy becomes less restrictive:

- emit new candidates
- prioritized as any other candidate to RFC rules

ICE Gather Policy: To More Restrictive

If existing candidates are forbidden because gather policy becomes less restrictive:

- flush all local candidates
- re-emit all local candidates

Problem:

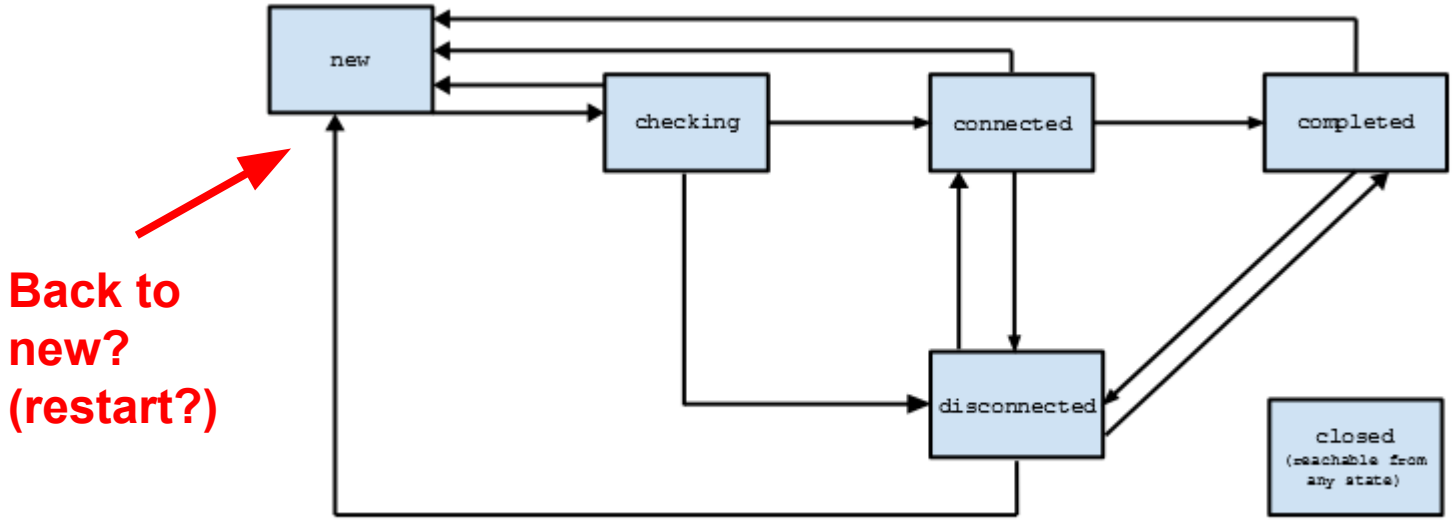
How does JS developer know when to flush?

ICE Gather Policy Change - Conclusion

Possible resolution:

- gather is an implicit "restart"

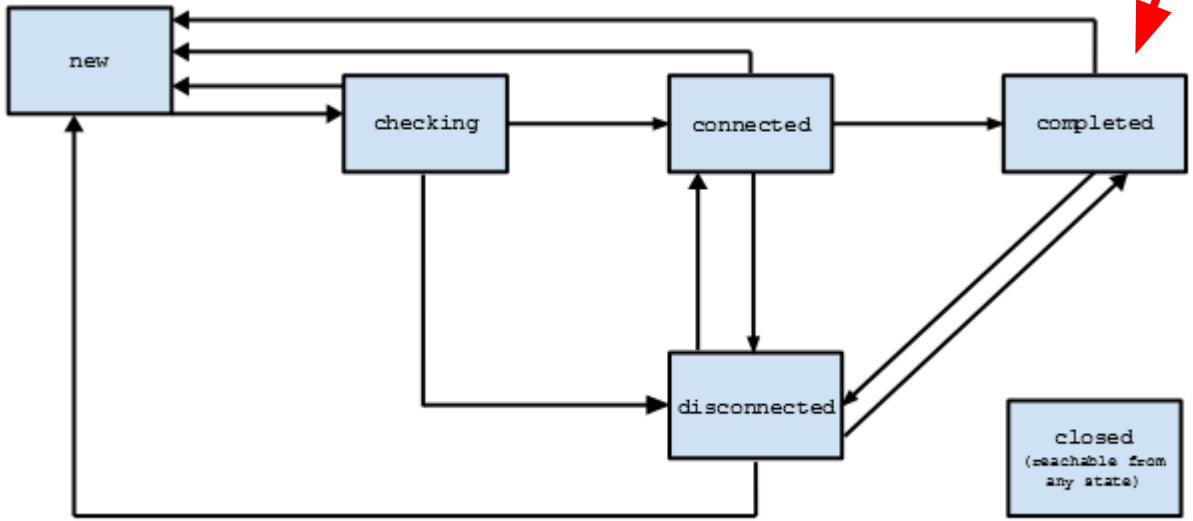
ICE States



If have "restart", should it go back to "new"?

ICE States

Only possible on ICE "controlling" side



Does this have value? Needed for "compatibility"?

RTCRtpParameters Behavior (106, 116)

RTCRtpParameters passed as an argument to RTCRtpSender.send() and RTCRtpReceiver.receive().

Assumptions:

- RTCRtpParameters contains optional parameters (e.g. SSRCs).
- Optional parameters can be set by the browser.
  - Example: SSRCs used to send simulcast and/or scalable video coding layers.
- Need to learn SSRCs and other information set by the browser engine.

Questions:

- How do we retrieve the RTCRtpParameter entries set by the browser?
- When are the values available for an RTCRtpSender? For RTCRtpReceiver?

How to retrieve RTCRtpParameters?

Proposal (to get "current" RTCRtpParameters):

```
partial interface RTCRtpSender {
    RTCRtpParameters          getParameters ();
};

partial interface RTCRtpReceiver {
    RTCRtpParameters          getParameters ();
};
```

BUT: Are these really needed?

When are RTCRtpParameters available?

Specifically:

- For RTCRtpSender, are all parameters are preconfigured before "send (..)"?
- For RTCRtpReceiver, will parameters get filled in over time as latching rules cause values to fill in (e.g. base SSRC vs FEC SSRC)?
- For RTCRtpSender/Receiver, is there a "final" parameters state?

Some other questions...

- For some codecs (e.g. VP8) decoder can decode whatever encoder can send. So receive(*RTCRtpParameters*) may not include layering, but send(*RTCRtpParameters*) might.
  - What does RTCRtpReceiver.getParameters() return?
  - Proposal: Return *RTCRtpParameters* provided to receive, no dynamic update.
- What if the browser can't do what the developer asks for (e. g. too many layers requested)?
  - Are RTCRtpParameters considered a hint? Answer: not a hint, bad RTCRtpParameters result in an exception or

RTCRtpEncodingParameters "scale"

Use case:

Programmer wants stream at 33% original size with 2 spatial layers:

Base = 33% x ¼ = 0.0825 resolution scale

1st layer = 33% x ½ = 0.165 resolution scale

2nd layer = 33% x 1/1 = 0.33 resolution scale

RTCRtpEncodingParameters "scale"

"Resolution scale" is a fixed geometric relationship. Value MUST be set precisely, but may not be because of:

- programmer error
- floating point rounding issues

- How "forgiving" is API to input error?
- Do we throw exceptions if input wrong or "fix" the values?
- Do we throw if value is "close" to proper value?

RTCRtpEncodingParameters "scale"

Resolution:

The denominator value becomes the "scale"

Base = ⅓ x ¼ = 12 resolution scale

1st layer = ⅓ x ½ = 6 resolution scale

2nd layer = ⅓ x 1/1 = 3 resolution scale

RTCRtpSender/Receiver send, send() / receive, receive()

Calling send() after send() is legal. Encoder changes engine encoding to adapt to new scenario.

Calling receive() after receive() is legal. Decoder adapts to new scenario.

2nd receive() cannot change all things, like payload type. Exception will be called.

Question for Community Group:

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

Good time to speak before implementation starts!

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

Special thanks to:

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