W3C ORTC Community Group Meeting

June 24, 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 4th 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

16 June 2014 Editor’s draft:

- http://ortc.org/wp-content/uploads/2014/06/ortc.html

Changes since 14 May 2014 Editor’s draft:

1. Added support for non-multiplexed RTP/RTCP and ICE freezing, as described in Issue 57
2. Added support for getRemoteCertificates(), as described in Issue 67
3. Removed filterParameters and createParameters functions, as described in Issue 80
4. Partially addressed capabilities issues, as described in Issue 84
5. Addressed WebIDL type issues described in Issue 88
6. Addressed Overview section issues described in Issue 91
7. Address readonly attribute issues described in Issue 92
8. Added ICE restart method to address the issue described in Issue 93
9. Added onerror eventhandler to sender and receiver objects as described in Issue 95

Read-only Attribute Issues

```
partial interface RTCDtlsTransport {
    readonly    attribute RTCIceTransport       transport;
    void                  setTransport (RTCIceTransport transport);
    };

partial interface RTCRtpSender {
    readonly    attribute MediaStreamTrack track;
    readonly    attribute RTCDtlsTransport transport;
    readonly    attribute RTCDtlsTransport rtcpTransport;
    void                    setTransport (RTCDtlsTransport transport, optional RTCDtlsTransport rtcpTransport);
    void                    setTrack (MediaStreamTrack track);
};

partial interface RTCRtpReceiver {
    readonly    attribute MediaStreamTrack? track;
    readonly    attribute RTCDtlsTransport  transport;
    readonly    attribute RTCDtlsTransport  rtcpTransport;
    void                    setTransport (RTCDtlsTransport transport, optional RTCDtlsTransport rtcpTransport);
};
```

Non-multiplexed RTP/RTCP

```
partial interface RTCIceTransport {
   // Keep track of what component this ICE Transport is for
   readonly attribute RTCIceComponent component;

    // Creates associated “RTCP” transport
    RTCIceTransport createAssociatedTransport();
}

enum RTCIceComponent {
    "RTP",
    "RTCP"
};

[Constructor(RTCDtlsTransport transport, optional RTCDtlsTransport rtcpTransport)]
partial interface RTCRtpSender {
      readonly attribute RTCDtlsTransport rtcpTransport;
      void   setTransport(RTCDtlsTransport transport, optional RTCDtlsTransport rtcpTransport);
}

[Constructor(MediaStreamTrack track, RTCDtlsTransport transport, optional RTCDtlsTransport rtcpTransport)]
partial interface RTCRtpReceiver {
      readonly attribute RTCDtlsTransport rtcpTransport;
      void   setTransport(RTCDtlsTransport transport, optional RTCDtlsTransport rtcpTransport);
}
```

ICE Freezing

```
[Constructor()]
interface RTCIceTransportController {
    sequence<RTCIceTransport> getTransports ();
    void                      addTransport (RTCIceTransport transport, unsigned long index = null);
};
```

Note: RTCIceTransportController object is need whenever the application is not multiplexing A/V as well as RTP/RTCP. Without this, ICE freezing may not happen consistently between endpoints!

Question from Shijun Sun: Is the index argument needed? Why can't addTransport push an RTCIceTransport onto the end?

Impact of Removal of create/filterParams

- Need to be able to create JS library version of create/filterParams
- Rewrote Examples 7 and 8 to create parameters based on an exchange of audio/video send and receive capabilities.
  - TODO: Need to code myCapsToSendParams and myCapsToRecvParams JS methods (Section 15.2).

Thus:

- Need "capabilities" with enough details to create "parameters".
- Can work without forward-looking knowledge of codecs, headers extensions, etc…
- Should not need specialized knowledge of specific codecs

What do the JS methods do?

1. Determine the codecs that the sender and receiver have in common.
2. Within each common codec, determine the intersection of supported parameters, header extensions and rtcpFeedback mechanisms, and configure them.
3. For each common codec, determine the payloadType to be used, e.g. based on the receiver preferredPayloadType.
4. Set RTCRtcpParameters such as rtcp.compound and rtcp.mux to their default values.
5. Return RTCRtpParameters enabling the jointly supported features and codecs.

ORTC Capabilities/Parameters Deficiencies

By attempting to code JS library methods, deficiencies were discovered:

- No way to choose payload number consistently.
- Feedback capabilities were global and not possible to derive per codec.
- Header extensions that apply to a particular codec were not known.
- Header extension preferred assigned id and encryption flag were not known.
- Capabilities/Settings objects from "gum" were difficult to manipulate.
- RTCP needs its own grouping of parameters.
- Encoding parameters needs to point to "codec" by payload id and not by name as multiple codecs with same name can be defined for different usages of same codec.

ORTC Capabilities Cleanup

```
dictionary RTCRtpCapabilities {
    sequence<RTCRtpCodecCapability> codecs;
    sequence<RTCRtpHeaderExtension> headerExtensions;
    sequence<DOMString>             fecMechanisms;
};

dictionary RTCRtpCodecCapability {
    DOMString                    name = "";
    DOMString                    kind;
    unsigned long?               clockRate = null;
    unsigned short               preferredPayloadType;
    unsigned short?              numChannels = 1;
    sequence<RTCRtcpFeedback>    rtcpFeedback;
    Dictionary                   parameters;
    unsigned short               maxTemporalLayers = 0;
    unsigned short               maxSpatialLayers = 0;
    unsigned short               maxQualityLayers = 0;
    boolean?                     multiStreamSupport =
false;
};
```

```
dictionary RTCRtpHeaderExtension {
    DOMString      kind;
    DOMString      uri;
    unsigned short preferredId;
    boolean        preferredEncrypt = false;
};

dictionary RTCRtcpFeedback {
    DOMString type;
    DOMString parameter;
};
```

ORTC Parameters Cleanup

```
dictionary RTCRtpParameters {
    DOMString                                   receiverId = "";
    sequence<RTCRtpCodecParameters>             codecs;
    sequence<RTCRtpHeaderExtensionParameters>   headerExtensions;
    sequence<RTCRtpEncodingParameters>          encodings;
    RTCRtcpParameters                           rtcp;
};

dictionary RTCRtpCodecParameters {
    DOMString                   name = "";
    unsigned short              payloadType;
    unsigned long?              clockRate = null;
    unsigned short?             numChannels = 1;
    sequence<RTCRtcpFeedback>   rtcpFeedback;
    Dictionary                  parameters;
    sequence<DOMString>         headerExtensionURIs;
};

dictionary RTCRtpHeaderExtensionParameters {
    DOMString      uri;
    unsigned short id;
    boolean        encrypt = false;
};
```

```
dictionary RTCRtpEncodingParameters {
    unsigned long?         ssrc = null;
    unsigned short?        codecPayloadType = null;
    RTCRtpFecParameters?   fec = null;
    RTCRtpRtxParameters?   rtx = null;
    double                 priority = 1.0;
    double?                maxBitrate = null;
    double                 minQuality = 0;
    double                 framerateBias = 0.5;
    double                 resolutionScale = null;
    double                 framerateScale = null;
    double                 qualityScale = null;
    boolean                active = true;
    DOMString?             encodingId;
    sequence<DOMString>    dependencyEncodingIds;
};

dictionary RTCRtcpParameters {
    unsigned long ssrc;
    boolean       compound = true;
    boolean       mux = true;
};

dictionary RTCRtpRtxParameters {
    unsigned long? ssrc = null;
};
```

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

- WebRTC 1.0 Dependencies
- ICE restart
- CNAME
- RTCRtpParameter defaulting
- MST/SST SS vs MS

WebRTC 1.0 Dependencies

- IdP
- Stats
- DataChannel
- DTMF

General proposal / philosophy:

- ORTC API works "as is".
- ORTC API is "as close" to WebRTC 1.0 as possible / logical.
- Synchronize ORTC API with WebRTC 1.0 if/when updates are completed.

IdP

```
partial interface RTCDtlsTransport {
    Promise<DOMString> getIdentityAssertion(
                                            DOMString provider,
                                            optional DOMString protocol = "default",
                                            optional DOMString username
                                            );
    // this encapsulates onidentityresult and onidpassertionerror in the promise

    Promise setIdentityAssertion(DOMString assertion);
    // this encapsulates onidentityresult and onidpvalidationerror

    readonly attribute RTCIdentityAssertion? remoteIdentity;
};
```

Proposal:

- IdP related assertions added to RTCDtlsTransport.
  - Shijun Sun: Could RTCDtlsTransport be used as an argument instead?
- Semantically identical to WebRTC 1.0 concept (although syntactically slightly different).
- All other related IdP interfaces are identical to WebRTC 1.0

Stats Proposal

Currently:

```
typedef (RTCRtpSender or RTCRtpReceiver or RTCDtlsTransport or RTCIceTransport or RTCSctpTransport) RTCStatsObject;
interface RTCStats {
    void getStats (RTCStatsObject statsObject, RTCStatsCallback successCallback, RTCErrorCallback failureCallback);
};
```

Proposed:

```
interface RTCStatsBase {
    Promise<RTCStatsReport> getStats ();
};
```

Interfaces implementing RTCStatsBase:

- RTCRtpSender
- RTCRtpReceiver
- RTCDtlsTransport
- RTCIceTransport
- RTCSctpTransport (via RTCDataTransport)

DataChannel

```
[Constructor(RTCDataTransport transport, RTCDataChannelParameters parameters)]
interface RTCDataChannel : EventTarget {
    readonly    attribute RTCDataTransport          transport;
    readonly    attribute RTCDataChannelParameters parameters;
    readonly    attribute RTCDataChannelState      readyState;
    readonly    attribute unsigned long            bufferedAmount;
                attribute DOMString                binaryType;
    void    close ();
                attribute EventHandler             onopen;
                attribute EventHandler             onerror;
                attribute EventHandler             onclose;
                attribute EventHandler             onmessage;
    Promise send (DOMString data);
    Promise send (Blob data);
    Promise send (ArrayBuffer data);
    Promise send (ArrayBufferView data);
};
```

Proposal: Similar to WebRTC 1.0 but send returns **Promise** to indicate when data is delivered. Further changes pending WebRTC 1.0 synchronization.

DTMF Sender

```
[Constructor(RTCRtpSender sender)]
interface RTCDTMFSender {
    readonly    attribute boolean           canInsertDTMF;
    void insertDTMF (DOMString tones, optional long duration = 100, optional 
long interToneGap = 70);
    readonly    attribute MediaStreamTrack track;
                attribute EventHandler      ontonechange;
    readonly    attribute DOMString         toneBuffer;
    readonly    attribute long              duration;
    readonly    attribute long              interToneGap;
};
```

Proposal:

- Identical to WebRTC 1.0, except construction
- Constructed from RTCRtpSender object

ORTC Specific Proposals

- ICE restart
- CNAME
- RTCRtpParameter setting
- SST-SS / SST-MS

ICE restart

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

Proposal:

- Add method "restart"
- Restarts ICE to gathering state and flushes all remote candidates

CNAME

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

Need for compatibility as both a get and set mechanism for CNAME.

Proposal:

- Add "cname" to RTCRtcpParameters
- Auto-filled with a CNAME unique per each JavaScript sandbox
- Developer can get "parameters" on RTCRtpSender/Receiver to learn value
- Developer can set "cname" on RTCRtpSender to specify value

RTCRtpParameters defaulting

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 get defaulted RTCRtpParameters?

Proposal (to get "current" parameters defaults):

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

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

When are RTCRtpParameters available?

Specifically:

- For RTCRtpSender, are all parameters immediately set once "send(..)" completes, or may some parameters get filled in asynchronously?
- 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?

Promise vs Change Event

As a promise:

```
partial interface RTCRtpSender {
    Promise<RTCRtpParameters>  send(RTCRtpParameters params);
};
```

As an event:

```
partial interface RTCRtpSender {
    attribute EventHandler?     onparameterschanged;
};
```

NOTES:

- Use a **Promise** if parameters are asynchronously set but developer only needs a final parameters event.
- Use an **event** if parameters are asynchronously set and developers may need events as more parameters are set.

RTCRtpSender Parameters Promise vs Event

RTCRtpSender

- Is an event really needed?
  - Are all parameters set once send returns?
- Might some implementations fill in parameters asynchronously (thus needing a Promise or event)?
  - Would send benefit from a Promise anyway? (e.g. for consistency?)
  - Will parameters continue to be filled in after the .then success function is called?

RTCRtpReceiver Parameters Promise vs Event

RTCRtpReceiver

- Likely needs multiple events as latching rules fill in defaulted parameters.
- Is there a 'final' change event (i.e. will all parameters eventually default and never change)?
- Would receive benefit from a Promise? (e.g. for consistency?)

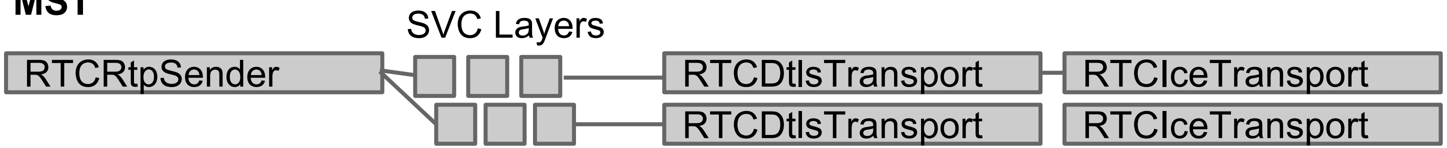
MST vs SST

MST = Multiple Session Transmission [RFC6190]

SST = Single Session Transmission [RFC6190]

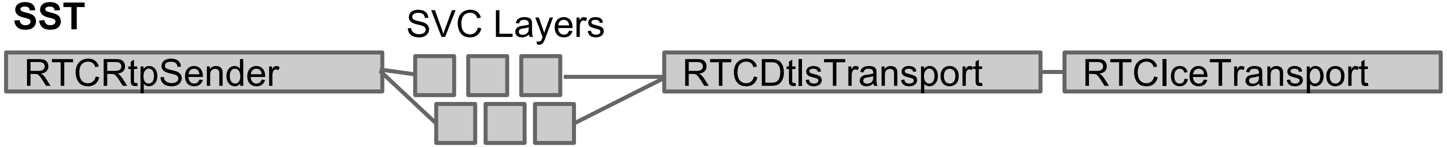
"Session" is ORTC = 1 ICE Transport (IP:port pair)

MST



NOT SUPPORTED!

SST



SUPPORTED

MST Not Supported by ORTC

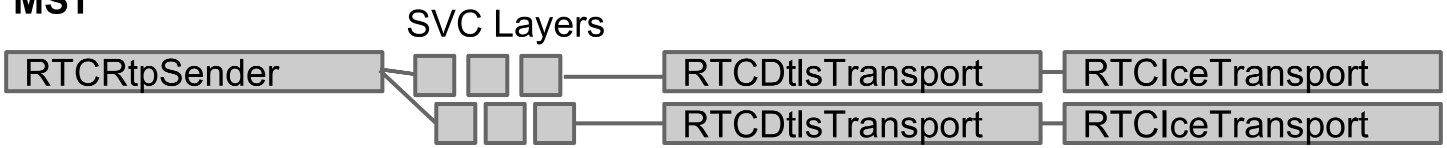
ORTC 1.1 does not support MST (at this time)

In theory ORTC could, but no use case demand

All known VP8 and H.264/SVC implementations use SST.

NOT SUPPORTED!

MST

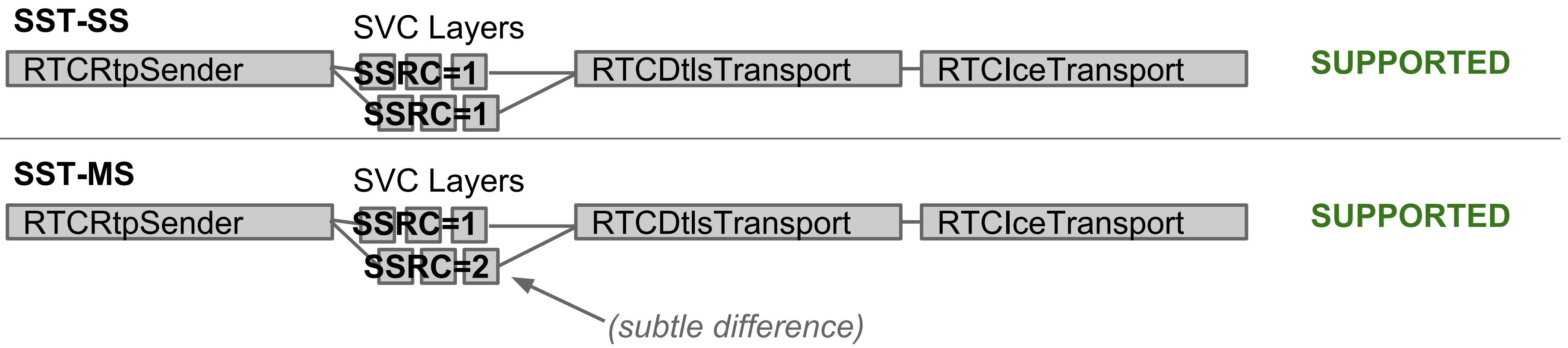


SST-SS vs SST-MS

SST = Single Session Transmission (i.e. 1 ICE Transport)

SS = Single Stream (1 SSRC per encoding)

MS = Multi-Stream (multiple SSRC per encoding)



SST-SS vs SST-MS

SST = Single Session Transmission (i.e. 1 ICE Transport)

SS = Single Stream (1 SSRC per encoding)

MS = Multi-Stream (multiple SSRC per encoding)

Issues:

- How to advertise which codecs support SS vs MS?
- How to define in codec parameters SS vs MS usage?

Codec Capability for SS vs MS

To advertise SS vs MS support in a codec list, add to RTCRtpCapability:

boolean svcWithMultipleSsrcs

If codec supports both SS and MS, then list codec twice one with true, one with false (as ORTC does when codec supports multiple selectable Hz rates).

Codec Parameters for SS vs MS

To specify SS vs MS in codec parameters, add to RTCRtpCodecParameters:

boolean svcWithMultipleSsrcs

By specifying "true" the codec will use a unique SSRC per encoding SVC layer.

ORTC CG Last Call - What does this mean?

What it is:

- Request for comments to flush out any remaining deficiencies of initial API
- Call for implementation feedback for any remaining deficiencies (that are only discovered by actually using the API)
- Complete enough to be a working, implementable public draft API
- "Working" public draft ORTC Community Group proposal for possible eventual standardization

What it is **not**:

- Absolutely final version of ORTC API (implementation feedback is needed)
- Final proposal from ORTC Community Group for possible standardization

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