W3C ORTC Community Group Meeting

May 15, 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 3rd 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.
  - Discuss use cases
  - Discuss difficulties with "params" vs "capabilities"
  - Discuss ORTC end game

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 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

14 May 2014 Editor’s draft:

- http://ortc.org/wp-content/uploads/2014/05/ortc.html

Changes since 29 April 2014 Editor’s draft:

- Fixed RTCRtpListener Example 5 (Issue 58)
- ICE restart explanation added (Issue 59)
- Fixes for error handling (Issue 75) and nits (Issue 76)
- Enable setting/retrieval of RTCP SSRC (Issue 77)
- Retrieval of audio and video capabilities (Issue 81)
- Partial update (needs more work) to the getStats(...) interface (Issues 82, 85), and SVC issues (Issue 83)

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

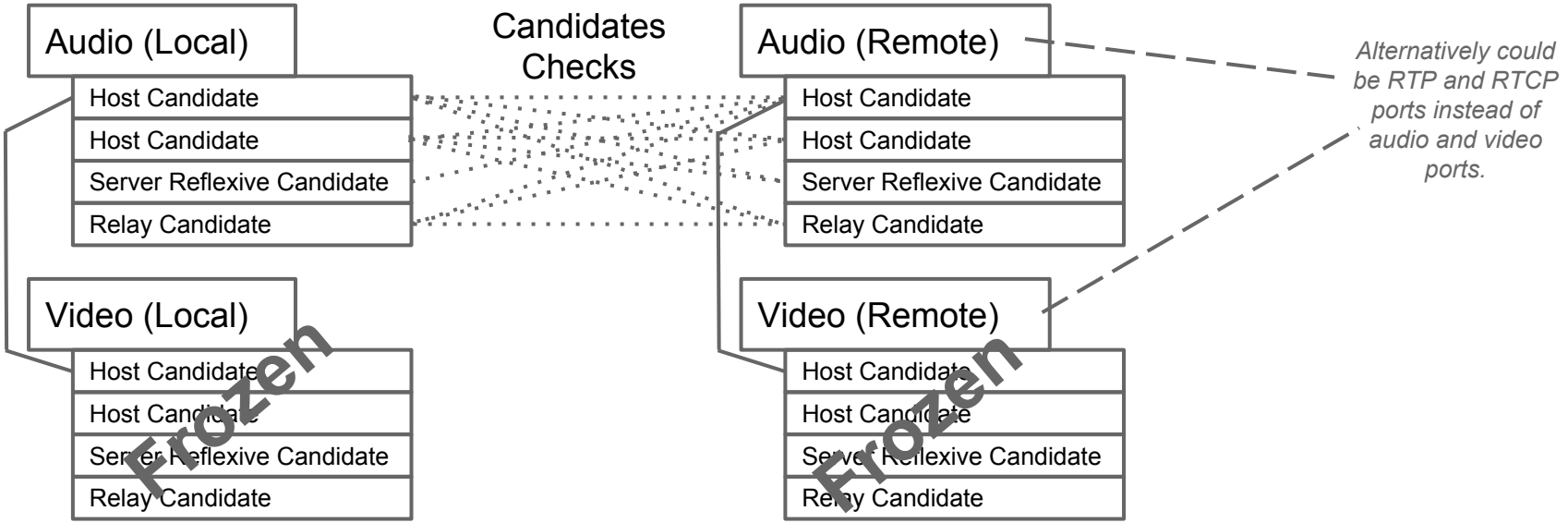
- Stats
- IdP
- RTCRtpListener behaviour and latching rules

Issues For Discussion Today

- RTCP non-mux / ICE freezing proposal
- Additional use cases (on top of WebRTC 1.0)
- Difficulties with "params" vs "capabilities"
- Explanation of "filterParams"
- JSL vs helper APIs
- ORTC end game

ICE Freezing Problem Review

- Need to control ICE setup ordering
- ICE checks need to happen in "m= line order"



ICE Freezing Proposal

We rejected the "implicit" approach, because 2 separate sessions should proceed with ICE in parallel. Per-session grouping is needed.

Thus, we introduce a 'session' object, called **RTCTransportController**.

Ice Freezing, Continued

RTCTransportController manages

- ICE freeze ordering
- BWE grouping

Maintains a list of RTP transports for a session, and their respective order.

In 1.0, could be exposed from PeerConnection.

RTCTransportController API

```
interface RTCTransportController {
    sequence<IceTransport> getTransports();
    void addTransport(IceTransport transport, int index = null);
}

// Note: |transport| is always a RTP transport, never RTCP.
// RTCP handling is separate.
```

RTCP Mux Handling

When creating an IceTransport for RTCP, it needs to be told it is for RTCP, so it can set component = 2. Similarly, it needs to only unfreeze after its associated RTP transport unfreezes.

Need a way to tie RTCP to RTP transport.

RTCP Mux API

```
partial interface RTCIceTransport {
  IceTransport createAssociatedTransport(IceComponent newComp);
};
partial dictionary RTCRtpParameters {
  bool rtcpMux;
}
```

To support non-mux, create a RTCP IceTransport from the RTP transport. When remote side replies with params, RtpSender will do right thing based on rtcpMux value. If true, can .stop() the RTCP IceTransport.

Use Cases for ORTC API

- All WebRTC use cases:

http://tools.ietf.org/html/draft-ietf-rtcweb-use-cases-and-requirements

- Select "3" important use cases for ORTC API:
  - Lossy network/SVC/FEC
  - Multi-point conference/simulcast/layered
  - Non-call scenarios (e.g. Security Camera)

All Use Cases from WebRTC

http://tools.ietf.org/html/draft-ietf-rtcweb-use-cases-and-requirements

Q: Are we agreeing to supporting all WebRTC use cases as currently defined?

Q: If not, which do you believe should not be done (or cannot be done)?

Use Case #1: Lossy Networks/SVC/FEC/RTX

- Alice is sending to Bob over lossy wan network on mobile device

Use Case #2: Multipoint conferencing/ simulcast/layering

- Alice and friends use a conferencing service, each having different bandwidth capabilities (SFU with SVC)*
- Alice sends thumbnail and big picture video (spatial simulcast and temporal scalability)*

** could be simulcast, or SVC, or simulcast with SVC.*

Use Case #3: Non-Call Scenarios (e.g. Security Camera)

- Security company monitors varying security cameras (one-way video(s), low framerates, high quality)
- Panic button for 2 way audio to security guard

Made difficult to implement in WebRTC 1.0 with negotiation (m=line matching) when it's purely unidirectional.

What are Capabilities and Params?

Capabilities - Defines what RTP/codec features the browser engine is capable of doing

Parameters - Defines the exact usage configuration of each and every RTP/codec feature used

filterParams: The Big Picture

- Capabilities are how the browser tells JS what it can do
- Parameters are how JS tells the browser what to do
- createParameters and filterParameters are (optional) helper functions
- Together Capabilities, Parameters, createParameters, and filterParameters enable different signaling/negotiation methods

See Peter's clarification email: http://lists.w3.org/Archives/Public/public-ortc/2014May/0049.html

filterParams(...) vs createParams(...)

Three main differences from "createParams(remoteCaps)":

- filterParams manipulates an existing set of "params" (does not create "params" from scratch)
- filterParams in practice filters based on (implicit) local and remote capabilities, not just creation of "params" from one or the other set of "capabilities"
- filterParams works with both sender/receiver capabilities (senders capabilities are not always uniform with receivers)

filterParams: Examples 7 & 8

Alice:

```
var sendAudioParams = RTCRtpSender.createParameters(audioTrack);
 signaller.offerTracks({
   "rtpAudioCaps": RTCRtpReceiver.getCapabilities("audio"),
   "audio": sendAudioParams
 }, function(answer) {
     var audioSendParams = RTCRtpSender.filterParameters (sendAudioParams,
answer.rtpAudioCaps);
       var audioRecvParams = RTCRtpReceiver.filterParameters( answer.audio);

   audioSender.send(audioSendParams);
   audioReceiver.receive(audioRecvParams);
```

filterParams (cont’d)

Bob:

```
var audioSendParams = RTCRtpSender.createParameters( audioTrack, remote.rtpAudioCaps);
var audioRecvParams = RTCRtpReceiver.filterParameters(remote.audio);

audioSender.send(audioSendParams);
audioReceiver.receive(audioRecvParams);

signaller.answerTracks({
   "rtpAudioCaps": RTCRtpReceiver.getCapabilities("audio"), "audio": audioSendParams
  });
```

Potential Invariants

(a) filterParameters (createParameters(), remoteCaps) == createParameters (remoteCaps)

Filtering locally created parameters through remote capabilities produces the same result as creating local parameters constrained by remote capabilities.

(b) filterParameters(remoteCreateParameters()) == ??

Filtering remotely created parameters through local capabilities produces the same result as:

1. **remotecreateParameters(localCaps):** ***creating remote parameters constrained by local capabilities (Examples 7 & 8)***
2. **createParameters(remoteCaps):** ***creating local parameters constrained by remote capabilities? (*****Rewrite of Examples 7 and 8:**

   http://lists.w3.org/Archives/Public/public-ortc/2014May/0044.html

Implications

- **Invariant a requires consistency *within a given browser implementation.***
- **Invariant b (either flavor) requires compatibility *between browser implementations.***
- **True if and only if Invariant b (either flavor) holds:**

1. **Exchange of capabilities or parameters produce equivalent results.**
2. **Both capabilities and parameter exchanges interoperate.**

Is filterParams definable?

- What does filtering receiver params in a sender do?
- What does filtering simulcast streams do?
- What does filtering SVC layering do?

Too difficult to define exact behaviour for many complex cases. Is it acceptable if it only works in simple use cases?

Is filterParams definable?

Every browser would need to operate consistently for all of these combinations:

```
sender.filterParams(senderParams, senderCaps);
sender.filterParams(receiverParams, senderCaps);
sender.filterParams(senderParams, receiverCaps);
sender.filterParams(receiverParams, receiverCaps);
```

```
receiver.filterParams(senderParams, senderCaps);
receiver.filterParams(receiverParams, senderCaps);
receiver.filterParams(senderParams, receiverCaps);
receiver.filterParams(receiverParams, receiverCaps);
```

Where senderParams was created using sender.createParams();

Where receiverParams was created using receiver.createParams();

-OR-

Where senderParams was created using remoteSender.createParams();

Where receiverParams was created using remoteReceiver.createParams();

Is filtering params helpful?

- May provide easy signalling negotiation for some simple use cases
- Does not cover all signalling needs
- May not work for complex scenarios (simulcast, SVC, etc)
- createParams(...) can be made to cover many use cases without ever needing to use filterParams(...)*

* even for complex use cases if given additional param creation guidelines

Purpose of createParams(...)

- Optional to use helper method to generate "parameters" given "capabilities"
- Makes generating ready-to-use complex RTP related dictionaries easy

Is createParams(...) helpful?

- Works for simple use cases but doesn't handle complex scenarios (e.g. simulcast, SVC, FEC, RTX)
- Has no simple knobs for generating more complex scenarios

filterParams/createParams Take Aways

- Only works for simple use cases (right now)
- Optional to use for application developer
- filterParams(...) still isn't clear enough to be able to define for ORTC implementers
- Might be possible to expand use case coverage for other scenarios (at great effort)
- Getting all browsers to consistently implement createParams/filterParams is hard

JavaScript Library vs Baked-In APIs

- createParams(...) and filterParams(...) are optional helper functions to make simple use cases easier
- Could be implemented as JSL instead of baked-in API

As JavaScript Library...

- Expandable to work in other use cases / scenarios
- Does not burden ORTC implementers
- JSL (or equivalent) would be required even for simple use cases
- Requires expanding definition of "Capabilities" to include everything a JSL might need to know about parameters

As Built-in API...

- Expanding usage beyond simple cases puts burden on ORTC API implementations
- Less flexible / forward / backward compatible
- Doesn't require expanding "Capabilities" as it can assume a ton of details about "parameters"
- Might be impossible to create an alternative JSL versions if "Capabilities" definitions are not expanded
- Won't be useful for many use cases (unless expanded)

Implement as JSL or Built-in API?

Should we:

- implement "createParams(...) and "filterParams(...)" as is?
- attempt to make more useful for more use cases?
- ship as a reference JSL only (which can optionally be made more useful)?
- remove both functions and leave them out entirely?

ORTC End Game

- Implementable specification
- Published as final report for CG (after implementation feedback)
- Potential standardization in a future W3C WG (TBD)

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 website

http://ortc.org