# W3C WebRTC WG Meeting

January 14, 2016  8:00AM-9:30AM PDT

Chairs: Harald Alvestrand
Stefan Hakansson
Erik Lagerway

# W3C WG IPR Policy

- This group abides by the W3C patent policy https://www.w3.org/Consortium/Patent-Policy-20040205
- Only people and companies listed at https://www.w3.org/2004/01/pp-impl/47318/status are allowed to make substantive contributions to the WebRTC specs

# Welcome!

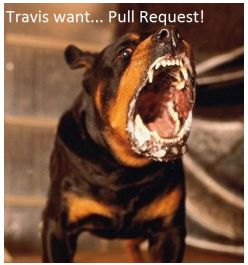
- Welcome to the interim meeting of the W3C WebRTC WG!
- During this meeting, we hope to make progress on some outstanding issues before transition to CR
- Editor’s Draft update to follow meeting

# About this Virtual Meeting

Information on the meeting:

- Hangouts Meeting
  - Participatory Hangout Link
- Link to Slides has been published on WG wiki
- Scribe? IRC http://irc.w3.org/ Channel: #webrtc
- The meeting is being recorded.

# Heads Up: Contribution Guidelines




- PR 447 includes new line wrapping guidelines (webrtc-respec-ci)
- Guide currently addresses:
  - W3C Legal
  - Line Wrapping
  - Pull Request Names
  - Linked Names
  - JSEP References
  - Refactoring: Moving Text

# For Discussion Today

- **Pull Requests**
  - **PR 434**: Change Params to Async (Cullen)
  - PR 454: Voice Activity Flag (Bernard)
  - PR 462: Add PeerConnection.activateSender() and update early media example (Peter Thatcher)
  - **PR 463**/**Issue 413**/**Issue 363**: Rollback (Peter Thatcher)
- **Issues**
  - **Issue 441**: RTCIceTransportPolicy needs to be DOMString not enum? (Bernard Aboba)
  - **Issue 389**: Should have a "closed" RTCPeerConnectionState (Peter Thatcher)
  - **Issue 412**: Framerate knob for simulcast (PR 430 exists) (Bernard Aboba and Peter Thatcher)
  - **Issue 370**: Add drop option for RTCDegradationPreference (Bernard Aboba)
  - **Issue 442**: Impossible to know if ICE agent is "finished checking", for "failed" and "completed" states (Taylor-b & Peter Thatcher)

# PR 434 - setParams Async

- Promise can fail if:
  - Changing any read-only parameters (.rtcp.cname, .rtcp.reducedSize, .headerExtensions, .encodings[i].ssrc, encodings[i].rtx.ssrc, encodings[i].fec.ssrc)
  - Trying to change codecs other than reordering or removing (adding or changing .payloadType, .mimeType, .clockRate, .channels, .sdpFmtpLine)
  - Setting writable parameters to something invalid (scaleResolutionDownBy < 0, invalid RID value)
    - Actually, should RID be read-only?
- Does anything above require async?

# PR 454 - Add contributing source voice activity flag

- For browser-browser communication, application may desire to access the “V” flag, in addition to the audioLevel.
  - “V” flag only available in client-mixer header extension (RFC 6464).
- Use case: Application wishes to show which peers have “voice activity”.
- Proposed change:

```
partial interface RTCRtpContributingSource {
    readonly        attribute boolean?              v;
};
```

# PR 462/Issue 415: Add activateSender() method

```
void activateSender()
```

```
The activateSender() method activates the RTCRtpTransceiver.sender such that
future calls to createOffer() and createAnswer() will mark the corresponding media
description as sendrecv or sendonly, as defined in JSEP.
```

```
If the sender is not already active and the PeerConnection is in the stable state,
this will trigger onnegotiationneeded.
```

From Harald: Shouldn't this say "this will set the negotiation-needed flag", and leave off mentioning the stable state?

From Peter: Yes, you are right.  I'll fix it.

# PR 463/Issue 413/Issue 363: Rollback

Text was mostly already there:

- RtpSender.mid and RtpReceiver.mid are gone
- RtpTransceiver.mid is nullable

Just adding some updates like:

"The initial value of .mid is null. setLocalDescription and setRemoteDescription may change it to a non-null value, as defined in JSEP."

"If the value of a RTCRtpTransceiver.mid was set to a non-null value by an SDP description that is rolled back, the RTCRtpTransceiver.mid will be set to null, as defined by JSEP."

# Issue 441: RTCIceTransportPolicy

- In response to Issue 384, PR 432 changed RTCIceTransportPolicy to sync with JSEP Section 4.1.1:

```
enum RTCIceTransportPolicy {
    "none",
    "relay",
    "all"
};
```

```
enum RTCIceTransportPolicy {
    "public",
    "relay",
    "all"
};
```

- Raised concerns about backward compatibility, as well as future extensibility.
- Harald suggested that RTCIceTransportPolicy be a DOMString instead of an enum.

# Issue 389: Should have “closed” RTCPeerConnectionState

- Jan-Ivar: The spec says: [RTCIceConnectionState's] "`closed` occurs only if `RTCPeerConnection.close()` has been called." With "closed" having more to do with the peer connection as a whole rather than the ICE agent specifically, and the rationale AFAIR for `connectionState` being to cover the peer connection as a whole more so than `iceConnectionState`, then it seems to follow that "closed" needs to be surfaced in (moved to) the RTCPeerConnectionState.
- Peter? (Peter here: ORTC has IceTransport.close(). WebRTC has PeerConnection.close(), but the state still applies)

# Issue 412: Framerate knob for Simulcast

- At TPAC there was consensus for a “framerate knob” for simulcast.
  Two knobs under consideration:

```
partial dictionary RTCRtpEncodingParameters {
        double          scaleFramerateDownBy;
        unsigned long   maxFramerate;
};
```

  - On-list “Consensus Call” produced only one response.
- When track frame rate changes:
  - scaleFramerateDownBy produces simulcast streams with different framerates.
  - maxFramerate produces streams below a maximum framerate (which could converge).

# Framerate Knob Use Cases

- Screen sharing conference with devices of different capabilities (e.g. PC and mobile d
  differing screen sizes and maximum framerates).
  - scaleResolutionDownBy generates streams with different resolutions.
  - maxFramerate generates streams within device maximum framerate.
  - Example:
    - PC stream: (scaleResolutionDownBy: 1.0, maxFramerate: 30)
    - Mobile stream: (scaleResolutionDownBy: 2.0, maxFramerate: 15)
- Video conference with speaker stream + thumbnail streams.
  - Speaker stream: (scaleResolutionDownBy: 1.0, scaleFramerateDownBy: 1.0)
  - Thumbnails: (scaleResolutionDownBy: 4.0, scaleFramerateDownBy: 4.0)

# Issue 370: Add drop option for RTCDegradationPreference

- How and when does a simulcast sender stop sending simulcast layers? Ways this can happen:
- Application-driven:
  - Application can set *encodings[j].active* to “false”
- Browser-driven:
  - Browsers MUST implement circuit breakers or congestion control.
  - Can browser decide to drop an encoding? If so, how does the application know it has been dropped?
- SFU-driven:
  - SFU can signal application to drop a layer, application then sets *encodings[j].active* = false.
  - draft-ietf-avtext-rtp-stream-pause provides TMMBR/TMMBN mechanism (no browser plan to implement?)

# Issue 370: Add drop option for RTCDegradationPreference

- Existing knobs:
  - *degradationPreference* (resolution/framerate tradeoff)
  - *priority* (relative importance of encodings)

```
enum RTCDegradationPreference {
    "maintain-framerate",
    "maintain-resolution",
    "balanced"
};
```

```
enum RTCPriorityType {
    "very-low",
    "low",
    "medium",
    "high"
};
```

# *degradationPreference & priority*

- *degradationPreference* indicates framerate/resolution preference of an RTCRtpSender:

```
partial dictionary RTCRtpParameters {
             sequence<RTCRtpEncodingParameters>        encodings;
             RTCDegradationPreference                  degradationPreference = "balanced";
};
```

  - *priority* indicates the relative importance of an encoding:

```
partial dictionary RTCRtpEncodingParameters {
             boolean           active;
             RTCPriorityType   priority;

};
```

# In the event of congestion….

- *parameters.degradationPreference* indicates whether the RTCRtpSender encoder should “maintain-resolution”, “maintain-framerate” (or be “balanced”).
  - All encodings have the same *degradationPreference*.
- *parameters.encodings[j].priority* indicates the relative importance of each encoding.
  - Determines QoS marking of each encoding.
  - Does it also guide encoder behavior (preference of encodings to drop if all cannot be sent simultaneously)?
  - Example:  “base” encoding at “high” priority, “medium” encoding at “medium” priority, best encoding at “low” priority.
    - Best encoding dropped first, then medium encoding, then base encoding.

# Example: Sign language interpretation

- Framerate more important than resolution
  - Possible use for minimum framerate constraint in gUM ( overconstrained exceptio minimum is unattainable)
  - *parameters.degradationPreference* = “maintain-framerate”;
- Two simulcast streams sent (one for PC, one for mobile)
  - In event of congestion, degrade resolution in both streams first, then drop the hig resolution stream entirely.
    - *encodings[0].scaleResolutionDownBy* = 1.0;
    - *encodings[0].priority* = “low”; // Drop high resolution simulcast stream if need
    - *encodings[1].scaleResolutionDownBy* = 2.0;
    - *encodings[1].priority* = “high”; // Keep low resolution simulcast stream if poss

# Issue 442: Impossible to know if ICE agent is “finished” checking

The problem:

> The ICE failed state is defined as:
>
> **The ICE Agent is finished checking all candidate pairs and failed to find a connection for at least one component.**
>
> However, “finished checking” is somewhat ambiguous. We could be “finished checking” all existing candidate pairs, but then get a new ICE candidate (either local or remote), and start checking again.

# What trickle ICE says about this

In section 8.1, the trickle ICE spec says that the Failed state is reached when:

- **all candidate harvesters have completed and the agent is not expecting to discover any new local candidates;**
- **the remote agent has sent an end-of-candidates indication for that check list as described in Section 9.3.**

However, besides doing another offer/answer with “a=end-of-candidates”, there’s no way to tell the ICE agent it can expect no more remote candidates.

# Question 1: Should “failed” only occur after local and remote gathering is done?

Pros:

- “failed” is now definitive. It can only be recovered through an ICE restart.
- It matches the trickle ICE definition of “Failed”.
- It allows the ICE agent to speed up ICE conclusion (in the case of non-aggressive nomination)

Cons:

- If someone forgets to signal remote end-of-candidates, they’ll never see “failed”.
- For most cases, this change won’t make a difference, since gathering is usually finished by the time the “failed” state is reached anyway.
- Applications can already determine this condition themselves, so this change doesn’t give app developers much advantage.

# Question 2 (if answer to question 1 is “yes”): How does an application indicate end of remote candidates?

In order from smallest to biggest change:

1. Just rely on “a=end-of-candidates”. If someone wants to see “failed”, they need to do another offer/answer when they see gathering is done.
2. Have a timeout (no candidates for 30 seconds == done)
3. Allow passing in a special candidate to “addIceCandidate” (perhaps with an mid and ufrag)
4. Add a new method (“candidatesComplete()”, for example)

# Question 2b (if answer to question 1 is “no”):
# Is this an acceptable definition of “failed”?

**The ICE Agent is done gathering, and finished checking all known candidate pairs, and failed to find a connection. If a new remote candidate is added while in this state, the state may return to “checking”.**

# Question 3: Should “failed” also occur if permanently “disconnected”?

“Disconnected” is defined as:

**Liveness checks have failed for one or more components. This is more aggressive than failed, and may trigger intermittently (and resolve itself without action) on a flaky network.**

However, if the 30-second consent interval elapses for all connections, “disconnected” becomes unresolvable (without an ICE restart). Should “disconnected” become “failed” in this case?

Pros:

- Gives applications a way to know when an ICE restart is *absolutely* needed.

Cons:

- By the time 30 seconds have passed, applications should probably have already done an ICE restart. Or the user has already refreshed the browser tab.
- “failed”’s meaning becomes even more complex.

# And what about “completed”?

Most of the answers to these questions also apply to “completed”.
If we decide “failed” means “finished gathering and checking and failed to find a connection”, then “completed” should probably mean “finished gathering and checking and found a connection”.

# What if one side never stops gathering?

If one side or the other decides it will keep trickling as new networking interfaces come up, then it will never signal "end of candidates", and the failed/completed state will never happen.

# Thank you


Special thanks to:

Google - for the Hangout

WG Participants, Editors & Chairs