

W3C WebRTC WG Meeting

Thursday, April 11, 2019
9:30 AM Pacific Time

Chairs: Bernard Aboba
Harald Alvestrand

W3C WG IPR Policy

- This group abides by the W3C Patent Policy <https://www.w3.org/Consortium/Patent-Policy/>
- 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 open issues in webrtc-pc, webrtc-stats, mediacapture-main and screen capture.

About this Virtual Meeting

Information on the meeting:

- Meeting info:
 - https://www.w3.org/2011/04/webrtc/wiki/April_11_2019
- Link to latest drafts:
 - <https://w3c.github.io/mediacapture-main/>
 - <https://w3c.github.io/mediacapture-output/>
 - <https://w3c.github.io/mediacapture-screen-share/>
 - <https://w3c.github.io/mediacapture-record/>
 - <https://w3c.github.io/webrtc-pc/>
 - <https://w3c.github.io/webrtc-stats/>
 - <https://www.w3.org/TR/mst-content-hint/>
 - <https://w3c.github.io/webrtc-nv-use-cases/>
 - <https://w3c.github.io/webrtc-dscp-exp/>
- Link to Slides has been published on [WG wiki](#)
- Scribe? IRC <http://irc.w3.org/> Channel: [#webrtc](#)
- The meeting is being recorded.

For Discussion Today

- **WebRTC-PC**

- [Issue 2026](#): Should WPT webrtc folder be renamed to webrtc-pc? (Youenn)
- [Issue 2080](#): Simulcast: Which layer gets dropped first? (Harald)
- [Issue 2141](#): Missing specification for how to assign bandwidth between layers (Harald)
- [Issue 2116](#): Simulcast Stats Implications (Varun & Henrik)
- [Issue 2121](#): Constrainable properties on remote tracks are under-specified (Henrik)
- [Issue 2150](#): transceiver.stop() needs more work (Jan-Ivar)

- **Screen Sharing**

- [Issue 102](#): Clarify what audio is captured and what “application” means in the context of restrictOwnAudio (Henrik)
- [Issue 103](#): Which audio constraints from media capture are applicable? (Henrik)

- **Mediacapture-main**

- [Issue 551](#): Should we allow empty string device IDs? (Youenn)
- [Issue 559](#): Spec does not handle fingerprinting related to exposing non-default capture devices (Youenn)
- [Issue 561](#): enumerateDevices can be used to track user devices in background pages (Youenn)
- [Issue 562](#): What constraint name should be exposed in case of a getUserMedia query with multiple failing constraints (Youenn)
- [Issue 573](#): Why do we have overconstrained event? (Henrik)

If Time Permits...

- **WebRTC-PC issues relating to rollback and ICE restart**
 - [Issue 2165](#): A simpler glare-proof setLocalDescription() (Jan-Ivar)
 - [Issue 2166](#): A simpler non-racy rollback (Jan-Ivar)
 - [Issue 2167](#): {iceRestart: true} works poorly with ONN (Jan-Ivar)

WebRTC-PC Issues

- [Issue 2026](#): Should WPT webrtc folder be renamed to webrtc-pc? (Youenn)
- [Issue 2080](#): Simulcast: Which layer gets dropped first? (Harald)
- [Issue 2141](#): Missing specification for how to assign bandwidth between layers (Harald)
- [Issue 2116](#): Simulcast Stats Implications (Varun & Henrik)
- [Issue 2121](#): Constraining properties on remote tracks are under-specified (Henrik)
- [Issue 2150](#): `transceiver.stop()` needs more work (Jan-Ivar)

Issue 2026 : Should WPT webrtc folder be renamed to webrtc-pc? (Youenn)

- Advantages
 - Consistency with other specs
 - XMLHttpRequest folder was renamed to xhr
 - Allow running only the webrtc-pc tests in test runner
 - Easier tooling, for instance to allow the spec to refer to the WPT tests
- Downsides
 - Might disrupt existing WPT PRs
 - Might disrupt WPT users, browser vendor CIs in particular

Issue 2080: Simulcast: Which layer gets dropped first? (Harald)

- Proposal: Drop the last one in the list on the a=simulcast: line
 - This is consistent with the IETF specification
 - We already promise to follow the IETF specification. No new text needed.
- Corollary: Have to specify order on the a=simulcast: line
 - Suggestion: Same order as on the encodings array
- This is done and merged as [#2071](#)

No action needed on what the subject line says. The text of #2080 describes an editorial matter of getting things to be obvious in the spec.

Any comments/concerns?

Issue 2141: Missing specification for how to assign bandwidth between layers (Harald)

- At the Prague hackathon, it was discovered that Chrome was allocating all the bandwidth to one of the layers, starving others.
- This is a bug - but if we want to test for it as part of the WG's test suite, we'd better have a description that says what's right.
- Congestion control has traditionally had a lot of latitude. So we don't want to overconstrain too much. Still, it should be possible to do better than nothing.
- Drop order of layers is described in draft-ietf-mmusic-sdp-simulcast. Bandwidth allocation isn't.
- SFUs frequently send different layers to different clients, so having one good layer and one bad layer is a Bad Thing.
- Suggested text could be "If congestion occurs, each layer SHOULD be given a share of bandwidth such that it remains useful - for instance, if frame rate is reduced, it should be reduced proportionally on all non-stopped layers".
- Where should it go?

Issue 2116: Simulcast Stats Implications - Part 1 (Henrik)

Current stats is not ready to accurately describe Simulcast scenarios.

Problem 1: Simulcast adds multiple RTP streams per sender. The sender (and likewise receiver) stats dictionaries are currently a mix of “sender”, “RTP stream” and “track/source” stats.

- frameWidth, audioLevel, etc

Solution:

- Move RTP stream-specific stats to RTC[In/Out]boundRtpStreamStats ([Issue 402](#)).
 - This affects a few stats that have been there for over a year :(
 - But “sender” and “receiver” stats have not been implemented, only their predecessor: “track” stats :)
- Multiple simulcast streams = 1 sender but multiple outbound-rtps ([Issue 394](#)).
- Move track/source stats to a new dictionary, RTCMediaSourceStats ([Issue 400](#)).

[Issue 2116](#): Simulcast Stats Implications - Part 1 (Henrik)

Problem 2: How to know which simulcast stream maps to which outbound-rtp?

Solution:

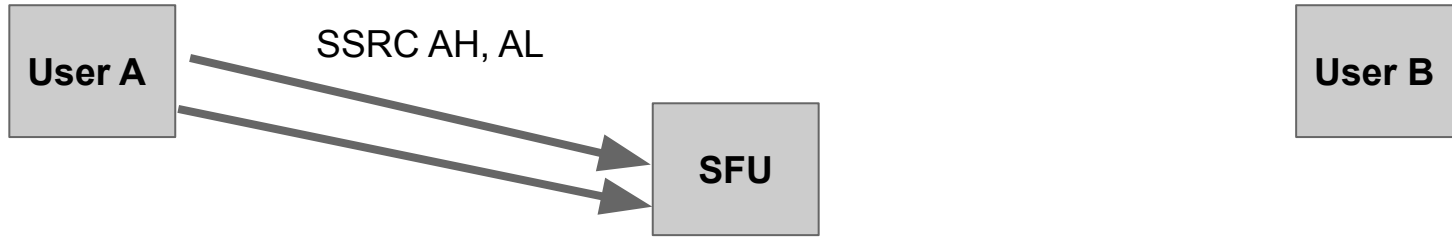
- Add RTCOutboundRtpStreamStats.rid ([Issue 395](#)).
- Related, add RTC[Audio/Video]SenderStats.mid ([Issue 396](#)).
 - Applies to “receiver” too.
- Add RTCEncodingParameterStats so that we know encoding properties of each outbound-rtp stream ([Issue 398](#)).

Issue 2116: Simulcast Stats Implications - Part 2 (Varun)

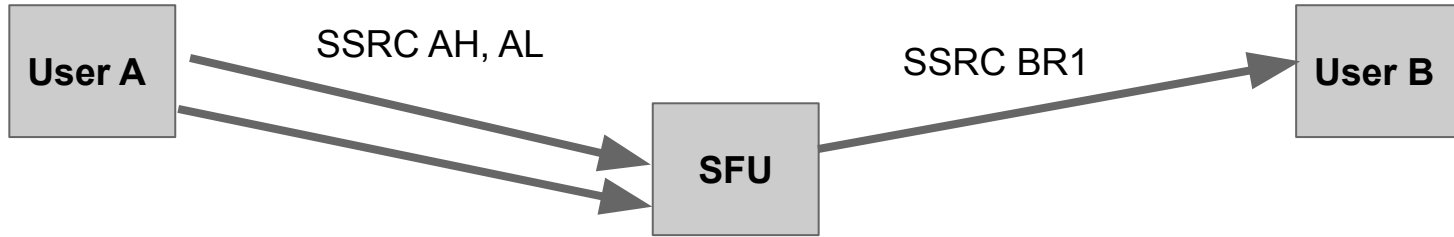
Can the receiver know encoding/source stream it is receiving via getStats?

- **Can we correlate which SSRC's data is being forwarded by the SFU (in the case of simulcast), so that receiver is able to log it via getStats on the receiver side?**

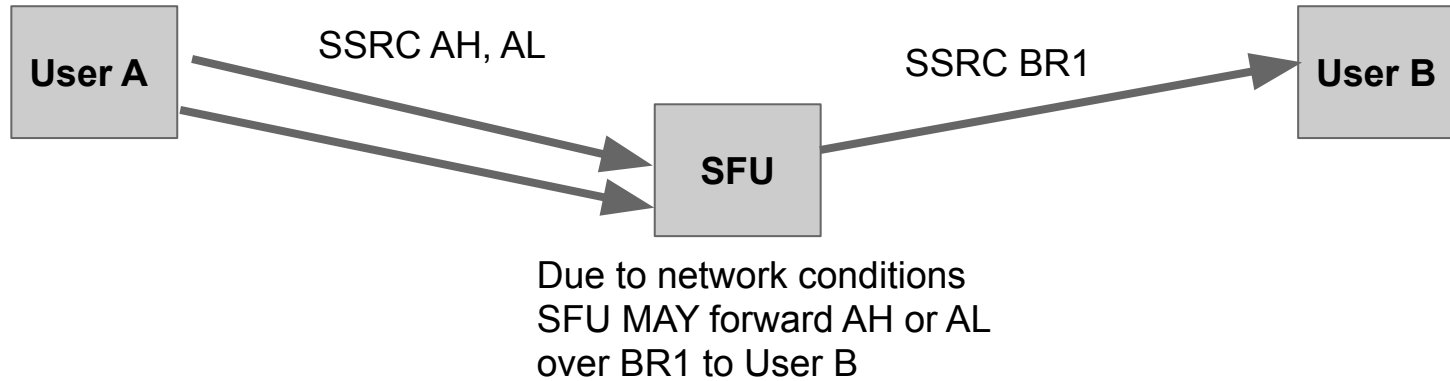
huh?



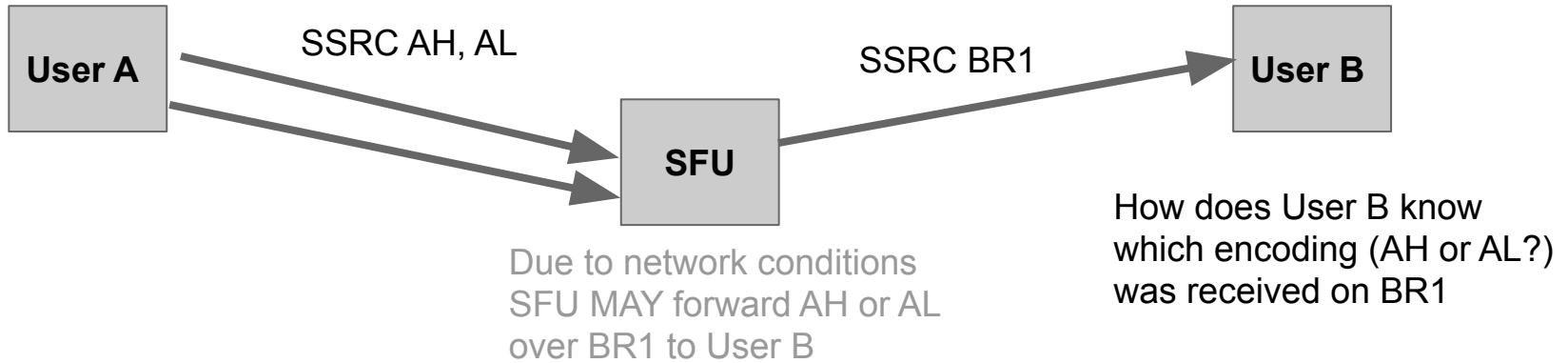
huh?



huh?



huh?



Proposal

- SFU in RTP packets with SSRC BR1 add the CSRC of the forwarded Stream SSRC AH or AL?

Proposal

- SFU in RTP packets with SSRC BR1 add the CSRC of the forwarded Stream SSRC AH or AL?
- If this makes sense,
 - a. Expose CSRC info in `getStats()` → which we probably already do.
 - b. Do we support CSRC for video?

Issue 401 SVC Stats

- There can be multiple scalable layers within a single RtpStream
- In getStats we want to know
 - what is the layer dependency (e.g. scalability mode)
 - Counter for how many **packets or frames were sent or received**
 - for a particular resolution
 - for a particular frame rate

Scalability Mode	Spatial Layers	Resolution Ratio	Temporal Layers	Inter-layer dependency
L1T2	1		2	
L1T3	1		3	
L2T1	2	2:1	1	Yes
L2T2	2	2:1	2	Yes
L2T3	2	2:1	3	Yes
L3T1	3	2:1	1	Yes
L3T2	3	2:1	2	Yes
L3T3	3	2:1	3	Yes
L2T1h	2	1.5:1	1	Yes
L2T2h	2	1.5:1	2	Yes
L2T3h	2	1.5:1	3	Yes
L3T2_KEY	3	2:1	2	Yes
L3T3_KEY	3	2:1	3	Yes
L4T5_KEY	4	2:1	5	Yes
L4T7_KEY	4	2:1	7	Yes
L3T2_KEY_SHIFT	3	2:1	2	Yes
L3T3_KEY_SHIFT	3	2:1	3	Yes
L4T5_KEY_SHIFT	4	2:1	5	Yes
L4T7_KEY_SHIFT	4	2:1	7	Yes

Issue 401 SVC Stats: Layer dependency

- Add scalability mode (e.g., L1T2, L1T3) to `RTCInboundRtpStreamStats` and `RTCOutboundRtpStreamStats`

Issue 401 SVC Stats: Resolution Stats

- Add Counter for how many packets or frames were sent or received for a particular resolution
 - perResolutionFramesSent/Received
 - perResoultionPacketsSent/Received

- We have encountered this before with DSCPs

```
record<USVString, unsigned long> perDscpPacketsSent;
```

- PROPOSAL:

```
record<USVString, unsigned long> perResolutionFramesSent;
```

- USVString can be “height”, or “width”, or “height x width”
- If we just report height or width, we could use unsigned long instead

Issue 401 SVC Stats: FrameRate Stats

- Add Counter for how many packets or frames were sent or received for a particular resolution
 - `distributionOfTimeSinceLastFrameSent`
 - `distributionOfTimeSinceLastFrameReceived`

- PROPOSAL:

```
record<double, unsigned long> distributionOfTimeSinceLastFrame*;
```

- `double` represents `interframedelay` (so this can be 16.66, 33.33, 66.66 etc)
 - representing 15 FPS (66.66ms), 30 FPS (33.33ms) and 60 FPS (16.66ms)
 - Example: `{ 33.33: 50, 66.66: 2000 }`
 - `UVString` or `double`?

Issue 2121: Constrainable properties on remote tracks are under-specified (Henrik)

Currently, the [spec says](#) `MediaStreamTrackSettings` of remote tracks “*will only be populated with members to the extent that data is supplied [via SDP and RTP data]. This means that certain members, such as `facingMode`, `echoCancellation`, `latency`, `deviceId` and `groupId`, will always be missing*”.

A reading of the spec allows `remoteTrack.applyConstraints()` to have an affect by transforming the track, but it's under-specified, perhaps it should always reject? Chrome assumed constraints that “make sense” are still applicable.

Status:

- Chrome has already shipped `width`, `height`, `aspectRatio`, `resizeMode` and `frameRate`.
 - Implemented as downscaling and dropping frames.

Proposal:

- Specify these constraints as downscaling/dropping frames.
 - Use case: Receive video in HD, but save a video to file in SD.
- Discuss other constraints separately, perhaps we don't want to support them.

[Issue 2150](#) / [PR 2168](#): `transceiver.stop()` needs more work (jib)

PROBLEM: The BUNDLE spec has painted us in a corner where calling `stop()` on the first transceiver in “have-remote-offer” signalingState, is **lethal**: It stops all transceivers. Happens ONLY in that state! Racy.

Impossible to fix in BUNDLE. Yet this flies in the face of the design of `negotiationneeded`, which was to [manage the negotiation state-machine](#), separately from high-level actions.

The primary use-cases for `transceiver.stop()` are:

1. High-level (everyone): Relinquish resources after an app is done with a transceiver:

```
button.onclick = () => {
  if (button.checked) {
    this.transceiver = pc.addTransceiver(track, {streams: [stream]});
  } else {
    this.transceiver.stop();
  }
}
```

 The above code will work 99.9% of the time, but once in a blue moon it will stop all transceivers; footgun!

2. Low-level (expert): Reject an offered m-line in time for the answer (in “have-remote-offer”).

Issue 2150 / PR 2168: `transceiver.stop()` needs more work (jib)

1. Modifying JSEP's definition of **stopped** seems fraught with peril. Best to leave it alone.
2. Some differences from `direction`:
 - a. `stop()` is terminal, and has instant (RTCP BYE) as well as negotiated effects.
 - b. `currentDirection` is a *result* of negotiation that happens after, whereas `stopped` is an *input* to negotiation, that needs to happen ahead of negotiation.

Best to design something on top of JSEP, and leave JSEP alone.

REVISED GOAL: Have **stopped** miss the hazardous "have-remote-offer" window.

[Issue 2150](#) / [PR 2168](#): `transceiver.stop()` needs more work (jib)

SOLUTION: A 2-step stopping procedure, inspired by `direction` vs. `currentDirection`.

Leave JSEP alone and define a new **stopping** property (in `webrtc-pc` only):

stopping of type `boolean`, `readonly`

The `stopping` attribute indicates that the `stop()` method has been called on this transceiver, but the transceiver has not yet been `stopped`. If `true`, this transceiver will be `stopped` in the queued task that fires the `negotiationneeded` event in the `stable` signaling state. On getting, this attribute *MUST* return the value of the `[[Stopping]]` slot.

In short, a **stopping** transceiver will be **stopped** on the next tick or once we're in "stable" state, whichever is later, thus missing the "have-remote-offer" danger window.


Proposals:

- A:** Everything like today, but define a new `stableStop()` method that sets this new **stopping** attribute, which triggers negotiation, causing the transceiver to always be **stopped** from stable state.
- B:** Like (A) except rename `stableStop()` to `stop()`, and rename old hazardous `stop()` to `reject()`.

Only B is the real proposal. A is just a rhetorical device.

[Issue 2150](#) / [PR 2168](#): `transceiver.stop()` needs more work (jib)

SOLUTION: A 2-step `stop()` that sets **stopping** immediately, and queues **stopped** to stable.

```
WebIDL 
[Exposed=Window] interface RTCRtpTransceiver {
  readonly attribute DOMString? mid;
  [SameObject]
  readonly attribute RTCRtpSender sender;
  [SameObject]
  readonly attribute RTCRtpReceiver receiver;
  readonly attribute boolean stopping;
  readonly attribute boolean stopped;
  attribute RTCRtpTransceiverDirection direction;
  readonly attribute RTCRtpTransceiverDirection? currentDirection;
  void stop ();
  void reject ();
  void setCodecPreferences (sequence<RTCRtpCodecCapability> codecs);
};
```

Experts may use `transceiver.reject()` to reject m-lines like today (they're experts!)

[Issue 2150](#) / [PR 2168](#): `transceiver.stop()` needs more work (jib)

FAQ:

Q: When exactly will the transceiver be stopped?

Right before your `negotiationneeded` callback is called, on the same queued task. Doesn't matter if you have one.

Q: I don't use `negotiationneeded`. Will this break me?

No, you'll be fine. The following should continue to work:

```
transceiver.stop();
await pc.setLocalDescription(await pc.createOffer());
```

...because the `createOffer/Answer` algorithms are [written to pick up on state changes](#) before succeeding. The order of queued tasks in JS guarantee we pick up the queued **stopped** from `stop()`.

Q: What happens if I call `stop()` in “have-remote-offer”?

The answer will be unaffected. You won't be stopped until you return to stable, where `negotiationneeded` fires again to trigger a second negotiation. This allows for safe stopping of any transceiver, even in BUNDLE. (Odd behavior? No. It's what already happens today if you were a few milliseconds later).

Screen Sharing Issues

- [Issue 102](#): Clarify what audio is captured and what “application” means in the context of `restrictOwnAudio` (Henrik)
- [Issue 103](#): Which audio constraints from media capture are applicable? (Henrik)

Issue 102: Clarify what audio is captured and what “application” means in the context of restrictOwnAudio (Henrik)

Recap:

- `getDisplayMedia({video:true,audio:true})` is supported, but audio is optional.
 - Up to the browser whether or not to return any audio.
 - When returning audio, up to the browser what audio is included.
- `restrictOwnAudio:true` means:
 - The user agent **MUST** attempt to remove audio produced by the application that called `getDisplayMedia()`. If unable to do so through processing, the user agent **SHOULD** exclude the application's audio from being captured.

Problem: “application” is vague... *does it mean browser, document origin, document, window/tab, etc?*

1. Does `restrictOwnAudio` require suppressing audio from child iframes? From parent iframes?
2. Does `restrictOwnAudio` require suppressing audio from other tabs of the same origin?

Proposal:

- Clarify that “application” means “document”.
 - The answer to 1) and 2) is **NO**.
 - Note: The optional nature of audio capture allows a UA to suppress more audio than “document” if this is needed to achieve no application audio. A UA is *allowed* to implement this as “exclude tab”.

Issue 103: Which audio constraints from media capture are applicable? (Henrik)

We have not defined which audio constraints are applicable to `getDisplayMedia()`:

- `volume`
- `sampleRate`
- `sampleSize`
- `latency`
- `channelCount`
- *(`echoCancellation` - microphone-specific, N/A)*
- *(`autoGainControl` - microphone-specific, N/A)*
- *(`noiseSuppression` - microphone-specific, N/A)*

We do not want filter sources for privacy reasons. But are any of these useful for processing?
Ideal values? Resampling?

Probably not very useful.

Proposal: Don't support any of them.

Mediacapture-main Issues

- [Issue 551](#): Should we allow empty string device IDs? (Youenn)
- [Issue 559](#): Spec does not handle fingerprinting related to exposing non-default capture devices (Youenn)
- [Issue 561](#): enumerateDevices can be used to track user devices in background pages (Youenn)
- [Issue 562](#): What constraint name should be exposed in case of a getUserMedia query with multiple failing constraints (Youenn)
- [Issue 573](#): Why do we have overconstrained event? (Henrik)

Issue 551: Should we allow empty string device IDs? (Youenn)

- Chrome exposes device ids 'default' values
- Safari exposes device ids "" values
 - In case a page does not have 'device-info' permission
 - To enforce mitigations for origins that had pages with 'device-info' permission
- Specification expects device ids to be per-origin unique identifiers
 - “The origin-unique identifier for the source of the MediaStreamTrack. The same identifier MUST be valid between browsing sessions of this origin, but MUST also be different for other origins”
- Proposal
 - Allow specific values to be non unique
 - As long as these values do not relate to a specific user

Issue 559: Spec does not handle fingerprinting related to exposing non-default capture devices (Youenn)

- enumerateDevices/getUserMedia can be used for fingerprinting purposes
 - enumerateDevices IS used for fingerprinting purposes
- Safari is building some mitigations
 - Try to hit a sweet spot between protection and existing apps needs
 - In particular, enumerateDevices will provide little information if 'device-info' or 'camera' or 'microphone' permission is not granted
 - Enumeration of default devices only with " deviceId values
- These mitigations do not always match the spec, hence the following issues

Issue 561: enumerateDevices can be used to track user devices in background pages (Youenn)

- devicechange event can only be fired on this condition:
 - Page has focus or page is capturing or page has 'device-info' permission
- Spec tries to prevent device addition/removal leakage to background pages
 - By not updating [[storedDeviceList]]
- The [[storedDeviceList]] protection can be bypassed and is not easy to fix
 - Through polling, a background page knows when devices are added/removed
- Proposal
 - Mandate enumerateDevices to delay processing until the devicechange condition is met
 - Mandate getUserMedia to delay processing similarly

Issue 562: What constraint name should be exposed in case of a `getUserMedia` query with multiple failing constraints (Youenn)

- `OverconstrainedError.constraint` allows a web page to identify why a query fails
 - This can be used to silently gather information on the device setup without user consent
- Mitigating this while still providing meaningful information to rightful apps is difficult if not contradictory
- Proposal
 - User agents MAY/SHOULD/MUST not report `OverconstrainedError.constraint` if the context does not have 'device-info' permission

Issue 573: Why do we have overconstrained event? 1/2 (Henrik)

getUserMedia() gives you a track with the capabilities/settings you asked for. But even with the right settings, you might not get what you asked for.

Example: poor lighting condition => less FPS than the camera aims for. This should trigger “onoverconstrained”.

Problems:

- Overconstrained mutes the track... (foot-gun!)
 - ...making it unusable (silent/black).
 - ...which contradicts the definition of “mute”:
The muted/unmuted state of a track reflects whether the source provides any media at this moment.
 - If applicable to remote WebRTC tracks, “onmute” has a different meaning.
- Quoting Jan-Ivar:
 - *It's undesired: demand has not materialized in 5 years.*
 - *It's redundant: just measure the output directly and react to it.*

Issue 573: Why do we have overconstrained event? 2/2 (Henrik)

Devil's advocate counter-arguments:

- *"It's undesired"*: This *may* only true because the track gets muted, you might still be interested in whether or not you get what you ask for if it didn't get muted.
- *"It's redundant"*: Having the application measure the output directly and reacting may require application logic to mimic parts of the constraints algorithm - this would be redundant too.
 - Application-measurements means polling, which means there may be a delay before overconstrained is noticed.

It might be possible to "fix overconstrained" by...

- Not muting the track on overconstraining.
- Add a boolean isOverconstrained attribute.
- Add an onunoverconstrained event.

But this API is overkill for "am I getting the right framerate?" and is largely untestable.

Proposal: Remove "overconstrained".

Remove it from the spec, no browser has implemented this.

Call for Consensus to remove has been out for a week, closing Real Soon Now.

Jan-Ivar's extra slides

I'm available to present these if there's time and interest. Otherwise please delete (I have copies).

[Issue 2165](#): A simpler glare-proof setLocalDescription() (Jan-Ivar)

[Issue 2166](#): A simpler non-racy rollback (Jan-Ivar)

[Issue 2167](#): {iceRestart: true} works poorly with ONN (Jan-Ivar)

Issue 2165: A simpler glare-proof `setLocalDescription()` (jib)

In "[Perfect negotiation in WebRTC](#)", I discovered to my horror that the following, sadly, is glare-prone:

```
pc.onnegotiationneeded = async () => {
  await pc.setLocalDescription(await pc.createOffer());
  io.send({desc: pc.localDescription});
}
```

A remote offer may come in between `createOffer` & `setLocalDescription`, causing it to fail. To safeguard we need:

```
pc.onnegotiationneeded = async () => {
  const offer = await pc.createOffer();
  if (pc.signalingState !== "stable") return; // ←- safeguard!
  await pc.setLocalDescription(offer);
  io.send({desc: pc.localDescription});
}
```

But who's going to remember that? Over an intermittent? Instead, I propose we allow a simpler *and* **safe** API:

```
pc.onnegotiationneeded = async () => io.send({desc: await pc.setLocalDescription()});
```

...and have it mean “*call createOffer or createAnswer implicitly, if needed, based on signalingState*”. Glare-proof!⁴¹

Issue 2165: A simpler glare-proof `setLocalDescription()` (jib)

Farfetched? No. **Fun fact**: the `sdp` argument to `setLocalDescription()` is already **unused**! a ritual:

```
await pc.setLocalDescription(await pc.createOffer());
```

...is *identical* to:

```
await pc.createOffer(); await pc.setLocalDescription({type: "offer"});
```

...because the spec already says to fish out **[[LastCreatedOffer]]** and use that here. Ditto answer.

Proposal A: The next natural step here is...

If **[[LastCreatedOffer]]** is `null`, instead of rejecting with `InvalidModificationError`, just invoke the `createdOffer` algorithm implicitly to set it. Duh! Ditto answer.

Proposed B: Proposal A +

Default `{type}` to (effectively) `signalingState.includes("offer") ? "answer" : "offer"`

100% backwards compatible. `setRemoteDescription()` would remain unchanged.

Issue 2166: A simpler non-racy rollback (Jan-Ivar)

In "[Perfect negotiation in WebRTC](#)", I roll back offers to solve glare (*"the polite peer"*), but, sadly, rollback is racy:

```
io.onmessage = async ({data: {description, candidate}}) => {
  if (description) {
    if (description.type == "offer" && pc.signalingState == "have-local-offer"){
      if (!polite) return;
      await Promise.all([
        pc.setLocalDescription({type: "rollback"}), // ←- safeguard!
        pc.setRemoteDescription(description)       // ←- safeguard!
      ]);
    }
  }
};
```

A remote candidate may come in between rollback & *setRemoteDescription*, causing it to be missed! To safeguard, I use `Promise.all` to enqueue both methods ahead of any *addIceCandidate* that may come in. But needing `Promise.all` to avoid intermittents is messed up! I propose we allow a simpler *and safe* API:

```
io.onmessage = async ({data: {description, candidate}}) => {
  if (description) {
    if (description.type == "offer" && pc.signalingState == "have-local-offer"){
      if (!polite) return;
      pc.allowRollback();
      await pc.setRemoteDescription(description);
    }
  }
};
```

Issue 2167: {iceRestart: true} works poorly with ONN (Jan-Ivar)

How does one restart ICE today when using negotiationneeded? Here's a good trick (but spot the bug!):

```
pc.onnegotiationneeded = async options => {
  await pc.setLocalDescription(await pc.createOffer(options));
  io.send({desc: pc.localDescription});
};
pc.oniceconnectionstatechange = () => {
  if (pc.iceConnectionState == "failed") {
    pc.onnegotiationneeded({iceRestart: true});
  }
};
```

Clever reuse... Except this will fail if iceconnectionstatechange fires outside of "stable" state!

Furthermore, what if your ONN uses rollback (e.g. to implement "the polite peer")? Your ICE restart just got rolled back! What do you do? You need to write app logic to persist until the offer is applied and not rolled back by the other peer. You will most likely never do this, leaving you open to intermittents.

Proposal:

```
pc.restartIce(); // sets [[RestartIce]], fires ONN. Cleared in SRD(answer)
```

Bonus slide: Perfect negotiation with a pushy SFU

Something I don't cover in "[Perfect negotiation in WebRTC](#)", is dealing with an SFU. Fippo explained that SFUs can be "pushy": They'll send an offer, followed immediately by second offer, a.k.a. a "better offer".

Two strategies come to mind: A) "The uber-polite peer" who rolls back the first offer, or B) the "submissive FIFO peer" who queues the offers. Which strategy to pick depends on how many answers the SFU expects.

But here's how to implement the "submissive FIFO peer":

```
io.onmessage = async ({data: {description, candidate}}) => {
  if (description) {
    if (description.type == "offer" && pc.signalingState == "have-remote-offer"){
      await Promise.all([
        pc.setLocalDescription({type: "rollback"}), // ←- safeguard!
        pc.setRemoteDescription(description),       // ←- safeguard!
        pc.createAnswer(),                          // ←- safeguard!
        pc.setLocalDescription({type: "answer"})    // ←- safeguard!
      ]);
    }
  }
}
```

A second offer may come in while we're busy responding to the first offer. To safeguard, I use `Promise.all` to front-load the peer connection's queue with all the methods I want done in sequence. This gets us all the way back to "stable" before any new peer connection methods get a go! Race solved.

For extra extra credit



Name that bird!

Thank you

Special thanks to:

WG Participants, Editors & Chairs

The bird