

WEBRTC WG Meetings

TPAC

November 6-7, 2017
Burlingame, CA

Chairs: Stefan Hakansson

Bernard Aboba

Harald Alvestrand

Welcome to the WebRTC WG Meetings at TPAC!

- WebRTC-PC has been published as a Candidate Recommendation!
- Goal of this meeting is to make progress on remaining WebRTC-PC, Media Capture and Depth Capture issues and to discuss how the WG should proceed going forward, including:
 - Testing
 - Progress toward PR for WebRTC-PC and Media Capture
 - New work

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

About These Meetings

- Meeting info:
 - [https://www.w3.org/2011/04/webrtc/wiki/November_6 - 7 2017](https://www.w3.org/2011/04/webrtc/wiki/November_6_-_7_2017)
- Link to latest drafts:
 - <https://rawgit.com/w3c/mediacapture-main/master/getusermedia.html>
 - <https://rawgit.com/w3c/webrtc-pc/master/webrtc.html>
 - <https://rawgit.com/w3c/webrtc-stats/master/webrtc-stats.html>
- Link to Slides has been published on [WG wiki](#)
- Scribe? IRC <http://irc.w3.org/> Channel: [#webrtc](#)
- The meeting is being recorded.
- WebEx info [here](#)

Monday TPAC Agenda

- 8:30 - 9:00 Agenda bashing
- 9:00 - 10:30 Public session on WebRTC Testing (Alexandre, Huib, Soares)
 - a. Test suite
 - b. Introduction to the new testing tool
- 10:30 - 11:00 Coffee
- 11:00 - 11:30 Test-driven development (Philip Jägenstedt)
- 11:30 - 13:00 WebRTC-PC Issues (Bernard, Peter)
- 13:00 - 14:00 Lunch break
- 14:00 - 15:00 WebRTC-stats (Varun, Harald)
- 15:00 - 15:30 Coffee break
- 15:30 - 17:00 Media Capture Main (Dan, Jan-Ivar)

Tuesday TPAC Agenda

- 8:30 - 9:00 Agenda bashing
- 9:00 AM - 10:30 AM Media-Capture
- 10:30 AM - 11:00 AM Coffee break
- 11:00 AM - 12:00 PM WebRTC and WebRTC-stats next steps
- 12:00 PM - 1:00 PM Lunch
- 1:00 PM - 3:00 PM WebRTC new work
 - a. New functionality (QUIC, SVC)
 - b. Discussion of working methods and how to get started on NV
- 3:00 - 3:30 PM Coffee break
- 3:30 PM - 4:30 PM Wrap-up and action items

WebRTC Testing

W3C TPAC
November 6, 2017
Burlingame, CA

Presenters: Huib, Alexandre, Soares

About This Session

- WebRTC testing has potential impact beyond the W3C standards community, so we have opened this session to the developer community at large.
- The meeting is being recorded.
- WebEx info [here](#)

Path to WebRTC 1.0



Specs



Tests



Compliance



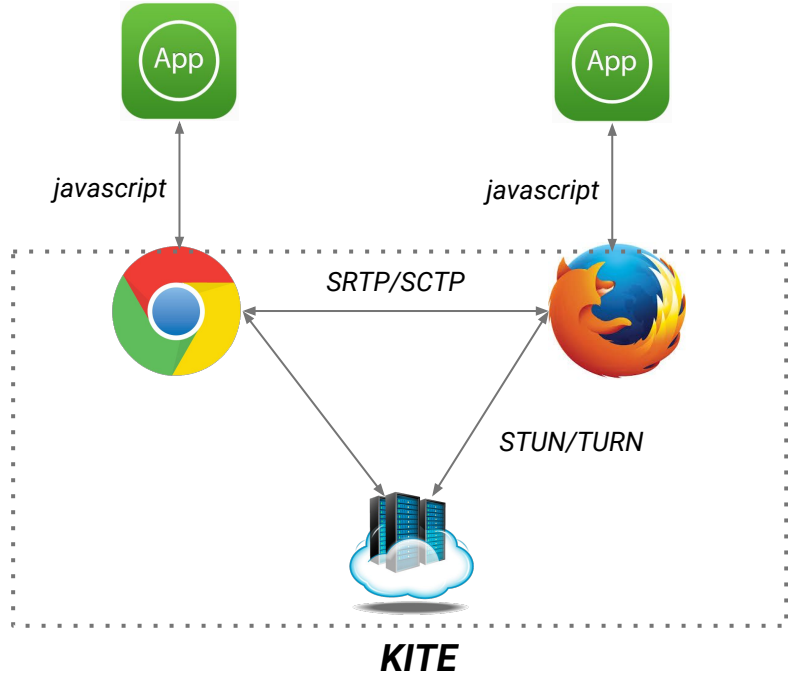
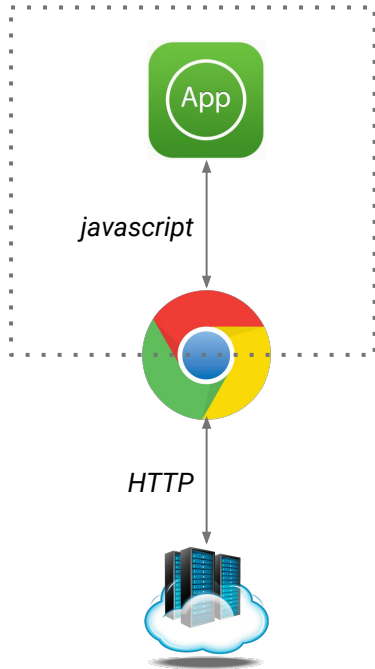
Reliability

Spec status

W3C		IETF	
WebRTC 1.0	CR	JSEP	→ RFC
Media Capture and Streams	CR	Data Channel	~RFC
Identifiers for WebRTC's Statistics	WD	RTP Usage	~RFC
		Transports	~RFC
		Audio/Video codecs	RFC
		Requirements	RFC

Tests tracks: WPT and Interop testing

Web Platform Tests



Compliance

WebRTC.org status

- **Available:** Spec-compliant getStats, MediaStreamTrack constraints, RTCRtpReceiver
- **Experimental:** RTCRtpSender, addTrack, ontrack
- **Under development:** Unified Plan

ETA: mostly completed end of Q1 2018

- Challenging for application developers without consistent compliance, native path chosen instead

Reliability

- Reliability and performance are critical
- Harder to get right than spec compliance
- Typical issues
 - Failing audio/camera subsystems
 - Jitter due in overloaded browser
 - Performance regressions across browsers

Web Platform Tests Progress

- May 2017: 293 tests
- Nov 2017: 1296 tests (+1003)

web-platform-tests dashboard

[Root](#) [About](#) [GitHub](#)

[WPT](#) /

				
Spec	chrome 63.0 linux 3.16 @7a0cf8ade7 Nov 02 2017	edge 15 windows 10 @053e5361f2 Nov 02 2017	firefox 57.0 linux * @1dfa574650 Nov 02 2017	safari 11.0 macos 10.12 @15cb61b5a7 Nov 03 2017
/webrtc	523 / 1296	121 / 574	539 / 1296	639 / 1296

<https://wpt.fyi/webrtc>

Coverage Status

- [PR #8051](#): Add coverage report and tools for WebRTC tests
- Coverage = (total - todo) / total

```
$ cd webrtc/tools
$ node scripts/overview.js
Overall Coverage
=====
todo          |      248
tested        |      315
trivial       |      173
untestable    |       79
=====
total         |      815
coverage      |    69.57%
=====
```

4. Peer-to-peer connections	67.83%
5. RTP Media API	67.01%
6. Peer-to-peer Data API	71.87%
7. Peer-to-peer DTMF	93.54%
8. Statistics Model	100.00%
9. Identity	86.04%
10. Media Stream API Extensions for Network Use	35.71%

Challenges to reach 100% coverage

- Missed out (124)
 - Difficulty to keep track of coverage, until now
- New steps added (97)
 - Long update between editor's draft
 - Most tests based on 20170605 draft - 5 months lag by now
 - Issue [#942](#) fixes this hopefully
- Race conditions (20)
 - Difficult to test correctly

Path to 100% coverage

- Straightforward tests to reach 90%
 - New steps resolved most testing issues
- Remaining 10% take some extra time
 - Tests involving other specs: SDP, ICE, HTML5, etc
- We should reach 100% by Q1 2018

Next Steps

- Coverage != correctness
 - Several issues have been reported to fix incorrect tests
 - Need validation from implementers and spec authors
- Discuss ways to keep track of coverage
- Automation with WebDriver & permissions API
- Testing other specs?
 - mediacapture-main ✓
 - mediastream-recording ?
 - Mediacapture-depth ?
 - Mediacapture-fromelement ?

Interop testing

https://drive.google.com/open?id=0B398g_p42xgrdWVxaXU3amh2VUU

Thank you

Special thanks to:

W3C/MIT for WebEx

Huib, Dr. Alex, Soares

Test As You Commit Policy

W3C TPAC
November 6, 2017
Burlingame, CA

Presenters: Bernard

For Discussion in this session

- [Issue 1617](#): Adopt “test as you commit” policy (foolip)

Issue 1617: Adopt “test as you commit” policy (foolip)

- WebRTC test suite is already in pretty good shape:
<https://rwaldron.github.io/webrtc-pc/>
- <https://wpt.fyi> running all tests
- Imported into at least Chromium, Gecko and WebKit
- We’re working on automation
- Similar policy already adopted by many other groups.

rwaldron.github.io/webrtc-pc/

4.2 Configuration

Incomplete

4.2.1 RTCConfiguration Dictionary

The **RTCConfiguration** defines a set of parameters to configure how the peer to peer communication established via [RTCPeerConnection](#) is established or re-established.

Incomplete

WebIDL

```
dictionary RTCConfiguration {
```

Tested

```
sequence<RTCIceServer> iceServers;  
RTCIceTransportPolicy iceTransportPolicy = "all";  
RTCBundlePolicy bundlePolicy = "balanced";  
RTCRtcpMuxPolicy rtcpMuxPolicy = "require";
```

```
DOMString peerIdentity;  
sequence<RTCCertificate> certificates;
```

Tested

```
[EnforceRange]
```


wpt.fyi (soon much less red)





web-platform-tests dashboard

[Root](#) [About](#) [GitHub](#)

Search test files, like cors/allow-headers.htm

[WPT](#) / [webrtc](#)

Data below are intended for web platform implementers and do not contain useful metrics for evaluation or comparison of web platform features. Also note that tested Edge and Safari are not pre-release versions (#109, #110).

Spec	 chrome 63.0 linux 3.16 @6ed1ba18d0 Nov 04 2017	 edge 15 windows 10 @790279663b Nov 06 2017	 firefox 57.0 linux * @1dfa574650 Nov 06 2017	 safari 11.0 macos 10.12 @790279663b Nov 05 2017
RTCCertificate.html	4 / 6	1 / 6	1 / 6	1 / 6
RTCConfiguration-bundlePolicy.html	11 / 16	1 / 16	8 / 16	14 / 16
RTCConfiguration-iceCandidatePoolSize.html	5 / 10	0 / 1	1 / 10	10 / 10
RTCConfiguration-iceServers.html	25 / 78	0 / 1	25 / 78	18 / 78
RTCConfiguration-iceTransportPolicy.html	7 / 17	0 / 1	11 / 17	14 / 17
RTCConfiguration-rtcpMuxPolicy.html	8 / 12	0 / 1	1 / 12	1 / 12
RTCDTMFSender-insertDTMF.https.html	1 / 7	/	1 / 7	0 / 1
RTCDTMFSender-ontonechange-long.https.html	1 / 2	/	1 / 2	1 / 2
RTCDTMFSender-ontonechange.https.html	1 / 12	/	1 / 12	1 / 12
RTCDataChannel-bufferedAmount.html	1 / 5	/	1 / 5	0 / 5
RTCDataChannel-id.html	3 / 3	/	1 / 3	1 / 3
RTCDataChannel-send.html	7 / 11	/	11 / 11	1 / 11
RTCDataChannelEvent-constructor.html	5 / 5	/	2 / 5	5 / 5
RTCDtlsTransport-	1 / 2	/	1 / 2	1 / 2

Automation using WebDriver

testdriver.js Automation

testdriver.js provides a means to automate tests that cannot be written purely using web platform APIs.

It is currently supported only for [testharness.js](#) tests.

API

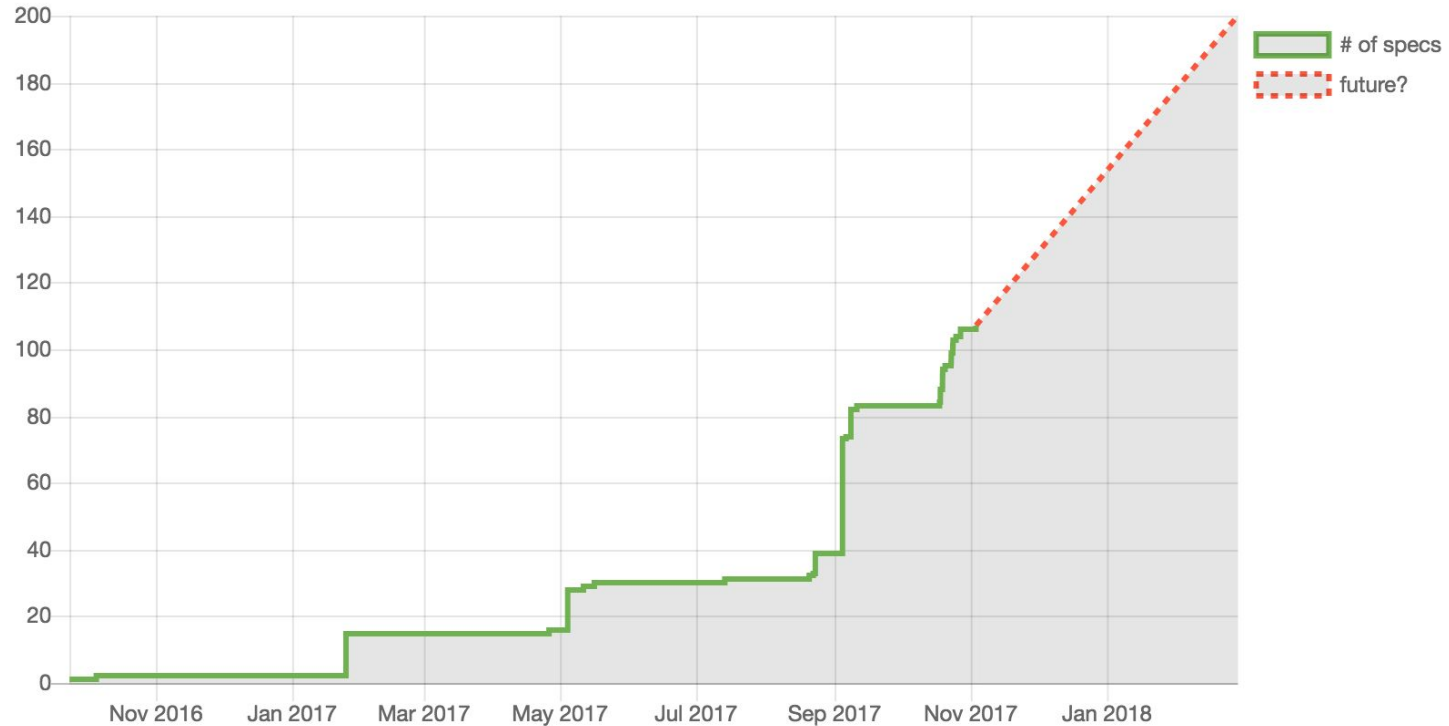
testdriver.js exposes its API through the `test_driver` variable in the global scope.

NB: presently, testdriver.js only works in the top-level test browsing context (and not therefore in any frame or window opened from it).

```
test_driver.click(element)
```

This function causes a click to occur on the target element (an `Element` object), potentially scrolling the document to make it possible to click it. It returns a `Promise` that resolves after the click has occurred or rejects if the element cannot be clicked (for example, it is obscured by an element on top of it).

Testing policy adoption (>50%!)



WebRTC-PC Session

W3C TPAC
November 6, 2017
Burlingame, CA

Presenters: Peter, Bernard

For Discussion in this session

● WebRTC-PC Issues

- [Issue 1194](#): AudioLevel of tracks, both send and receive (Peter)
- [Issue 1586](#): behavior of OfferToReceive* set to false (Jan-Ivar)
- [Issue 1625](#): RTCPriorityType combines relative bitrate with QoS priority, which applications may not want (Peter)
- [Issue 1635](#): Need for initial bitrate by the application/RtpSender (Peter)
- [Issue 1644](#): Adding more values to RTCIceTransportPolicy Enum (Peter)
- [Issue 1646](#): Isolated Media Streams require modification to permission algorithms (soareschen)

[Issue 1625/PR 1632](#): RTCPriorityType undesirably combines relative bitrate with QoS priority (Peter)

Problem: You want to send something with high priority for QoS but with fewer bits.

RTPEncodingParameters.priority controls both and makes higher QoS = more bits

Examples:

- Change the DSCP markings without changing the bits
- Change the bits without changing the DSCP markings
- Send low-bitrate/high-QoS audio with a high-bitrate/low-QoS audio
- Send low-bitrate/high-QoS video with a high-bitrate/low-QoS video
- Send low-bitrate/high-QoS audio with a high-bitrate/low-QoS video

Also: the ratios of 1:2:4:8, are not granular enough to be very useful.

[Issue 1625/PR 1632](#): RTCPriorityType undesirably combines relative bitrate with QoS priority (Peter)

Proposal: break up "more bits" and "higher QoS" into two separate controls

- Existing RtpEncodingParameters.priority is for "higher QoS" (It's like "QoS priority")
- New RtpEncodingParameters.relativeBitrate is for "more bits" (It's like "bits priority")

[PR 1632](#):

`relativeBitrate` of type `double`

Indicates the relative amount of bitrate that this encoding **should** be allocated when congestion occurs, relative to other encodings being sent under the same congestion control regime. For example, if two encodings use values of 1.0 and 1.5, respectively, and the congestion controller determines that 5Mbps are available to allocate, the encodings **should** be allocated 2Mbps and 3Mbps, respectively. The encoding may also be further constrained by other limits (such as `maxBitrate` or per-transport or per-session bandwidth limits), resulting in it using less than its available share.

[Issue 1625/PR 1632](#): RTCPriorityType undesirably combines relative bitrate with QoS priority (Peter)

Peter's recommendation: split "priority" into "QoS priority" and "bitrate priority". More flexible bitrate ratios via "relativeBitrate" seems like an elegant solution and nice bonus.

Issue 1635: Need for initial bitrate by the application/ RtpSender (Peter)

Should we add a way for the app to say "I think the bandwidth estimate is X. Start there"?

If so, where does it go and what does it mean? **It's complicated.**

- RTCCConfig: What does getConfig() return? What does setConfig(getConfig()) do?
- New method on PeerConnection? What about WebRTC NV?
- New method on IceTransport? It's more of an RTP thing than an ICE thing.
- New method on RtpSender? But it applies to all senders.

What if you're not bundling?

What does it do to SCTP?

What happens if you switch ICE network routes?

Issue 1635: Need for initial bitrate by the application/ RtpSender (Peter)

Peter's recommendation: Leave it out (for now?). We can try something in WebRTC NV (perhaps we'll have a separate RtpTransport where this would make sense).

Issue 1194: AudioLevel of tracks, both send and receive (Peter)

Would `RTCRtpReceiver.getSynchronizationSources()` be useful for getting the audio level of the last sent packet?

Or would it make more sense to get the audio level from `RTCRtpReceiver.track` (either using `WebAudio` or adding a new event to `MediaStreamTrack`)?

[Issue 1194](#): AudioLevel of tracks, both send and receive (Peter)

Peter's recommendation: leave it off RtpSender. Put it on MediaStreamTrack, or just let WebAudio handle it. Even the initial bug reporter (Henrik Bostrom) said that's fine.

Issue 1644: Adding more values to RTCIceTransportPolicy Enum (Peter)

Two separate things going on:

1. Data-channel-only applications want **a way to get more candidates** than the default behavior without going all the way to using getUserMedia to get all of them. This is something we've talked a lot about before.
2. Applications may want **a way to constrain the candidates** to something less than "all" and more than "relay". For example, "all for the default route, and relay for the others". This could not always be accomplished by JS filtering candidates, but it could maybe be accomplished by allowing JS to force the browser into a lower permission state than it has otherwise granted ("Even if I have been granted access through getUserMedia, pretend I haven't been and do your default thing"). This is something we've talk a lot about before.

[Issue 1644](#): Adding more values to RTCIceTransportPolicy Enum (Peter)

Peter's recommendation: Leave it out. Maybe try to handle it with the permissions spec.

Issue 1586: behavior of OfferToReceive* set to false (Jan-Ivar)

History: offerToReceive was removed when Transceivers were introduced

- Not needed; same functionality available with transceiver.direction attribute
- {offerToReceiveAudio: true} (and video) as a way to offer to receive audio/video (even though not intending to send) later re-introduced
- However 3 browsers support {offerToReceiveAudio: false} = sendonly
- Usage: Broadcast; “I want to send audio but not receive it” offer and other end-point isn’t browser (= doesn’t know to reply with recvonly)
- Complex to specify - Usage?

Questions:

- 1) Should we re-introduce {offerToReceiveAudio: false} as a way to make the SDP indicate you are not willing to receive even though you’re sending?
- 2) Spec currently disallows browsers to continue legacy support. Allow?
- “In all other situations, it MAY be disregarded.”?

Issue 1646: Isolated Media Streams require modification to permission algorithms (soareschen)

- The `peerIdentity` field in `MediaStreamConstraints` allows applications to request for media device permission for specific `peerIdentity` only
- Permission algorithms are specified in `mediacapture-main` and `w3c/permissions`
 - Not taken in consideration of `peerIdentity` constraints
- `peerIdentity`-specific permission grant may not be reused for subsequent request for general usage
- Caching algorithms only use `media kind` and `deviceId` as keys
- Multiple Options
 - Modify other specs?
 - Augment algorithms from `webrtc-pc`?
 - Remove `peerIdentity`-specific permission prompt?
 - Suggestions?

WebRTC-Stats Session

W3C TPAC
November 6, 2017
Burlingame, CA

Presenters: Varun, Harald

We've made a lot of progress

- 3 updates:
 - 14 December 2016, 30 March 2017, 14 June 2017
 - 120 issues closed.
 - Many thanks to contributors, especially: Henrik, Taylor, and Jan-Ivar.

Major changes

- Added new RTCStatsType
 - csrc, split some of the RTP objects into local and remote objects
 -
- Added a section on Obsolete Stats
 - Current reason is renaming and unit changes
 - rtt had ambiguous units - some used milliseconds and others used seconds, was changed to be in seconds as per RFC3550 and was renamed roundTripTime.
 -

Path to CR

- Need review on the existing document
 - From implementers (naturally)
 - From users of stats (important!)
- Get Security and Privacy section in a better shape
 - Self review started: <https://docs.google.com/spreadsheets/d/1zJy-zVohPIVcGtOUewYxrHBjTKQwG9Y0KGsK48E5a3g/edit#gid=0>
- 30 Open issues
 - 6 icebox issues
 - Some of these will get done, however, others **require much wider review or discussions** → to make sure some of these get resolved in time for CR.
 - Example: concealedAudibleSamples in review for a few months.
<https://github.com/w3c/webrtc-stats/pull/215>

Open issues to discuss at this meeting

- Structural issues - guidelines for new stats, format of specs
 - Need input from spec-writers/readers and folks who know how to do process
 - May be CR blockers
- Stats proposed that are hard to clearly define
 - Need subject matter expert input
 - Not CR blockers
- CR blockers: 99/251, 177/243, 131/262, 231, 230
 - Issue 99/PR 251: Security and privacy considerations
 - Issue 177: Caching and consistency of getStats()
 - Issue 131/PR 262: “objectDeleted” marker
 - Issue 231: Sender and Receiver stats vs Track stats
 - Issue 230: RTCMediaStreamTrackStats to 4 dicts
 - Any others?

Issue 99/PR 251: Security and privacy considerations

- CR blocker
- Self analysis done by Harald and Varun,
 - https://docs.google.com/a/callstats.io/spreadsheets/d/1zJy-zVohPIVcGtOUewYxrHBJTKQwG9Y0KGSK48E5a3g/edit?usp=drive_web
- Does this specification expose any other data to an origin that it doesn't currently have access to?
 - The properties exposed by `RTCReceivedRTPStreamStats`, `RTCRemoteInboundRTPStreamStats`, `RTCSentRTPStreamStats`, `RTCOutboundRTPStreamStats`, `RTCRemoteOutboundRTPStreamStats`, `RTCIceCandidatePairStats`, `RTCTransportStats` **expose network-layer data not currently available to the JavaScript layer.**
-

Issue 99/PR 251: Security and privacy considerations

- Does this specification allow an origin access to a user's location?
 - For instance, the [round-trip time] exposed in `RTCRemoteInboundRTPStreamStats` can give some coarse indication on how far apart the peers are located, and thus, if one of the peer's location is known, this may reveal information about the other peer.

Issue 177: Caching and consistency of getStats()

- CR blocker
- Proposal: Require objects to have updated stats after object state transitions; permit some caching when there is no state transition.
- PR #243
 - When the state of the PeerConnection visibly changes as a result of an API call, a promise resolving or an event firing, subsequent new getStats() calls MUST return up-to-date dictionaries for the affected objects.
 - *For example, if a track is added with addTrack() subsequent getStats() calls MUST resolve with a corresponding RTCMediaStreamTrackStats object. If you call setRemoteDescription() removing a remote track, upon the promise resolving or an associated event (stream's onremovetrack or track's onmute) firing, calling getStats() MUST resolve with an up-to-date RTCMediaStreamTrackStats object.*

Issue 131/PR 262: “objectDeleted” marker

This reflects explicitly the model that stats exist for objects that are gone.

Is this marker useful? Correctly defined?

CR blocker?

Editors think this is ready to merge.

Issue 231/[PR 273](#): Sender & Receiver stats vs Track stats

- CR BLOCKER

Track stats were introduced before senders/receivers. Later we made each track per-attachment to the PC, a compromise. A more accurate view is to have stats per sender, receiver and track objects.

This is an issue because ReplaceTrack will make the two stats different

Much simpler, removes the need for [detached](#), and accumulation of stale objects.

Have to decide one way or the other where we want to end up

Might present all stats on both objects for a while as a transition strategy?

Issue 230/[PR 272](#): RTCMediaStreamTrackStats to 4 dicts

- CR blocker (even though not visible on the wire)

Removes "Only valid for..." prose in nearly every RTCMediaStreamTrackStats member by using WebIDL instead (dictionary inheritance).

kind and remoteSource stay. API unaffected. Minor JS observable change (order).

(If we go for #231 remoteSource disappears naturally: remoteSource == "receiver").

Editors like it.

Issue 133/135: DSCP information

Not CR blocker

Gives information that can diagnose remarking and codepoint-blocking problems.

Belongs on RTCTransportStats or RTCIceCandidatePairStats?

Information is not accessible on all platforms.

Proposal: Add a record<USVString, long>, where key is the DSCP codepoint in string-numeric form and the value is # of packets sent / received with that marking.

Alternative: define an interface that is a maplike<short, long>, where key is DSCP codepoint and value is # of packets sent/received.

Issue 202: concealedAudibleSamples

Concealed samples are samples synthesized to conceal packet loss.

concealedAudibleSamples are such samples synthesized during “audible” portions of the stream, such as when someone is speaking.

Hard to find a definition on “audible” that’s comprehensible and supportable.

PR #215 is a proposal

Not a CR blocker

Issue 1613: Stats and Isolated Streams (varun)

- Some aspects (like volume) allow JS to learn stuff about what goes on inside isolated streams (see RFC 6562 for a similar case involving RTP)
- Two possible approaches:
 - a) Minimize the information leaked (e.g by dithering, caching) and live with it
 - b) Do not report sensitive stats on isolated streams, live without it
- Suggested approach: b) List sensitive items in doc, say “will not report on isolated streams”
 - Sensitive items beyond “volume” need suggesting!

SVC use cases for an SFU developer

Sergio Garcia Murillo - CosMo Software

SVC Use cases

Simulcast and SVC use cases are very similar (if not identical): **content adaptation without transcoding**

Adaptation will happen as a combination and/or trade off of the following items:

- Bitrate
- Image size
- FPS
- Quality
- Decoding complexity

Typically you would always love to have the best quality, with biggest image size and most fps, but most of the times you have will be restricted to do that and have to adapt video stream to the match certain limits.

Also, another interesting use case would be for **increasing reliability against packet losses** by applying different protection levels to each layer and maximizing the probability that at least the base layer is received.

Content Adaptation (I)

Bitrate adaptation is the most common one:

- Each participant on a multiconference will have different available bandwidth so the SFU must select the appropriate layers to forward to each participant based on the bandwidth estimation.
- The SFU will either choose to select a lower spatial layer (reducing image size/quality) or temporal layer (reducing fps) depending on its business logic.

Image size adaptation is also very common:

- The SFU will optimize network and cpu resources by sending a lower resolution matching the UI needs and service logic requirements.
- Typically the SFU will forward a bigger image for the active speaker than the non-active ones.

Content Adaptation (II)

Adapt on FPS and/or Quality

- I have not found any use case for them (Thumbnailing?).
- Typically it is a trade off in other uses cases: When adapting bitrate, what should be preserved quality or movement?

Adapt based on decoding complexity

- Sometimes you do not want to overflow low end devices (mobile phones most probably).
- Combination of image size and fps adaptation.

What do SFU developers need?

- Enabling/Disabling temporal/spatial scalability (or simulcast/temporal scalability).
- Set desired targets on image size (not really on fps or bitrate).
- Set desired granularity for adaptation (number of layers/steps or and relative difference between them in terms of bitrate/fps/size).
- Set priority between encodings/layers to control how they prefer to degrade quality/movement and/or split bitrate between.
- We don't really care about dependencies between layers really, it is a codec thing.

Media Capture Session

W3C TPAC
November 6-7, 2017
Burlingame, CA

Presenters: Peter T, Jan-Ivar, Peter B

For Discussion In This Session

- **Media Capture Issues**

- [Issue 495/Issue 472](#): Video dimension constraints (Peter Thatcher)
- [Issue 470](#): Does `getSettings()` reflect configured or actual settings? (Jan-Ivar)
- [Issue 466](#): Question about setting belong to source in Section 3 (Jan-Ivar)
- [Issue 441](#): Interop for muted tracks (Jan-Ivar)
- [Issue 478](#): Content hints for `MediaStreamTrack` (Peter Boström)

[Issue 495](#)/[Issue 472](#): Video dimension constraints (Peter Thatcher)

Scenario: Sender has a 16:9 track. Receiver wants to receive a 1:1 track.

Solution we *don't* want: something on RtpSender to change 16:9 into 1:1 when sending it.

Solution we *do* want: a way to get a 1:1 track from a 16:9 track.

```
// Or maybe height and width instead of aspectRatio  
track.applyConstraints({aspectRatio: {exact: 1.0}})
```

Problem: does the app prefer adding pixels (pillar/letterbox) or removing pixels (crop)?

Related problem: is the browser allowed to scale to avoid overly adding or removing?

[Issue 495](#)/[Issue 472](#): Video dimension constraints (Peter Thatcher)

Proposal: let the app tell you what it's happy with a new constraint.

Let's call it `resizeMode`.

```
enum ResizeModeEnum {  
    "crop-and-scale", // Removes pixels; allows scaling  
    "box-and-scale",  // Adds pixels; allows scaling  
    "none",           // No adding or removing; no scaling  
};
```

```
track.applyConstraints({aspectRatio: {exact: 1.0},  
                       resizeMode: "crop-and-scale"})
```

Last question: what's the default? I prefer `"crop-and-scale"`.

Issue 472: No constraint defaults. When do we downscale? (Jan-Ivar)

- Problem: Web devs get vastly different results from browsers that don't support downscaling.
- New constraints aside, can we we converge on default behavior for downscaling in all browsers? SHOULD minimize fitness distance, or SHOULD minimize processing?

1. `await track.applyConstraints({width: 640, height: 360}); // does it downscale/crop?`
2. `await track.applyConstraints({width: {exact: 640}, height: {exact: 360}}); // this?`

Two models being explored today:

- Chrome: downscales on both above
- Firefox: downscales on neither today, may downscale soon only in second case.
- How do you avoid downscaling/processed modes today?

```
({width: {min: 640, max: 1024}, height: {min: 360, max: 720}}); // trust browser  
({width: {min: 640, max: 1024, ideal: 1024}, height: {min: 360, max: 720, ideal: 720}})
```


Issue 470: Does `getSettings()` reflect configured or actual settings? (Jan-Ivar)

- Is it a settings arbitration API (between concurrent users) or a measurement API?
- Spec says *"To check which ConstraintSets are currently in effect, the application should use `getSettings`."*, suggesting deterministic target “settings” values.
- Live measured values sometimes deviate from their (target) “settings”, like during:
 - [camera motor pan](#) 30 → 60: actual < setting, actual > old {pan: {max: 30}}
 - System overload or low light: measured frameRate fluctuates below target.
 - Live volume vs volume setting.
- If `getSettings()` were to return live values, then the spec text above doesn't hold.
- Does any browser implement actual values? (`getSettings()` in Firefox returns setting).
- Does any browser implement aggressive `OverconstrainedError` or `onoverconstrained`?
No, cause auto-disabling != useful, would surprise users at this point who might use `exact` to force rescaling/decimating. Rare users who want this behavior are better off [JS measuring](#).
- Must agree when promise resolves, or suffer users polling `getSettings()` waiting on motor.

Issue 466: Question about setting belong to source in Section 3 (Jan-Ivar)

- (Note: Github discussion devolved into discussing previous slide and `ideal`)
- `@guido` and I are on the same page on core question above:
 - Sources practically support more than one setting concurrently. True for audio filters.
 - `track.getSettings()` should return what's relevant to consumers of that track.
 - Which means `getSettings()` from different tracks from the same source may return different values due to downscaling/decimating/audio processing.
 - Tracks are useful abstraction APIs for browsers. Actual hardware source settings seem of little relevance, and mandating their examination is a cross-origin security issue (can be used to detect concurrent use, e.g. is user using a particular site atm, or even be used to morse-code data across origins, bypassing cross-origin protections).
- Suggestion: Massage language as needed to reflect this. If this means settings “belong” to tracks, so be it, but if there are other ways to refine it, swell.

Issue 470: Does `getSettings()` reflect configured or actual settings? (Jan-Ivar)

- Is it a settings arbitration API (between concurrent users) or a measurement API?
- Spec says *"To check which ConstraintSets are currently in effect, the application should use `getSettings`."*, suggesting deterministic target “settings” values.
- Live measured values sometimes deviate from their (target) “settings”, like during:
 - [camera motor pan](#) 30 → 60: actual < setting, actual > old {pan: {max: 30}}
 - System overload or low light: measured frameRate fluctuates below target.
 - Live volume vs volume setting.
- If `getSettings()` were to return live values, then the spec text above doesn't hold.
- Does any browser implement actual values? (`getSettings()` in Firefox returns setting).
- Does any browser implement aggressive `OverconstrainedError` or `onoverconstrained`?
No, cause auto-disabling != useful, would surprise users at this point who might use `exact` to force rescaling/decimating. Rare users who want this behavior are better off [JS measuring](#).
- Must agree when promise resolves, or suffer users polling `getSettings()` waiting on motor.

Issue 441: Interop for muted/track.enabled=false tracks (Jan-Ivar)

- The remaining concern (interop) seems like a WebRTC issue, not mediacapture.
 - WebRTC could encode 1 fps while MediaRecorder sees 30 fps.
- In WebRTC, there will be cases where the webrtc backend will stop receiving frames:
 - track being muted or track.enabled=false (hardware may be off),
 - manual canvas capture stream.
- Concern is whether an implementation choice (in this case not sending redundant frames) might cause interop troubles if the infrastructure is not expecting that kind of scenario.
- Spec:
 - 1 frame per second?
 - No frames at all?
 - audio/video?
- Are frames on wire needed to keep connection alive? To keep ports open?

Issue 478: Content hints for MediaStreamTrack (Peter Boström)

Lorem Ipsum is simply dummy text of the printing industry. Lorem Ipsum has been the industry's standard dummy text ever since the 1500s, when a random selection of type and scrambled it to make a type specimen book. It has survived not only five centuries, but also the leap into electronic typesetting, remaining essentially unchanged. It was popularised in the 1960s with the release of Letraset typefaces. Many desktop publishing packages and web page editors now use Lorem Ipsum as their default text dummy text, and a search for Lorem Ipsum will find more than 100 instances of it on the Internet.

- Browsers configure implicit settings based on content source.

Chrome/WebRTC: UVC -> webcam, tab/desktop capture -> screenshare, all audio -> speech

Wrong for capture cards, **wrong** behavior when screensharing video / game content, **wrong** for music.

- Use a content “hint” to help the browser make implicit decisions.

Using a MST property -> can be used by MediaStreamRecorder and other APIs outside WebRTC with less flexible controls, without having to modify their specifications (scales better). Informs “balanced”.

- Example behavior:

Motion video: Downscale / use higher max QP to preserve motion.

Detail video: Drop frames / use lower max QPs to preserve individual frame quality.

Speech: Use noise suppression and echo cancellation by default. Maybe enhance intelligibility?

Music: Turn off noise suppression (preserve snares), tune echo cancellation differently / turn it off.

Issue 478: Content hints for MediaStreamTrack (Peter Boström)

- October interim discussion:
 - Does this belong on MSTs or on sources and/or sinks?
 - Might need an 'auto' setting for any constraint that could be affected by this to deal with otherwise unpredictable defaults
 - How would this work with the already complex selectSettings() algorithm?
 - What, if any, is the impact of adding this into the gUM spec on extension specs that currently don't know anything about it?
 - How would conflicting settings of this value be addressed?
 - What are the testing implications?

Issue 1533: Live RTCRtpContributingSource confusing (Jan-Ivar)

- No value in its liveness when one already has to poll to discover new participants.
- When participants left, objects remained stuck with their last values.
- To store the data, you'd need a copy algorithm, or data would plot a straight line.
- Solution: `s/interface/dictionary/`

```
dictionary RTCRtpContributingSource {
    DOMHighResTimeStamp timestamp;
    unsigned long    source;
    byte?           audioLevel;
};
```

```
dictionary RTCRtpSynchronizationSource : RTCRtpContributingSource {
    boolean?         voiceActivityFlag;
};
```

Splitting Priority Part 2

draft-ietf-rtcweb-transports-17 says "The priority settings affect two pieces of behavior: Packet send sequence decisions and packet markings. Each is described in its own section below". We can change that to say "there are two priority settings":

<https://github.com/rtcweb-wg/rtcweb-transport/pull/50>

Meanwhile, we can add this to RtpEncodingParameters. The existing priority sets both, except when overridden by either or both.

<https://github.com/w3c/webrtc-pc/pull/1659>

```
RTCPriorityType bitratePriority = "low";  
RTCPriorityType markingPriority = "low";
```


Media Capture, WebRTC-PC and WebRTC-Stats Next Steps

W3C TPAC
November 7, 2017
Burlingame, CA

Presenters: Bernard, Harald, Varun

For Discussion Today

- **Media Capture next steps**
- **WebRTC-PC next steps**
- **WebRTC-stats next steps**
- **Other specifications**

Media Capture Next Steps

- **Recycled at CR**
 - Complete test suite
 - Address outstanding issues
 - Recycle at CR again (to handle substantive issues)
- **Next goal: PR**

WebRTC-PC Status

- **96 open issues. Breakdown**
 - 25 labelled “editorial”
 - 15 labelled “test suite issue”
 - 10 labelled “question”
 - 3 labelled “PR exists”
- **By topic:**
 - 15 relating to peer identity
 - 9 relating to object model
 - 6 relating to ICE
 - 5 relating to data channel/SCTP

WebRTC-PC Next Steps

- **Immediate goals:**
 - Complete test suite
 - Require tests for new PRs
 - Address outstanding issues
 - Recycle at CR
- **Ultimate goal: PR**
 - Interoperability requirements
 - WPT to run in two browsers (dashboard)
 - KITE test requirements?

Other specifications

- **Wide review for:**
 - Media recording
 - Image capture
 - From element
 - Screen capture?
 - Depth?

WebRTC-Stats Status

- **30 open issues. Breakdown**
 - 12 labelled “CR blocker”, 3 are bugs
 - 6 labelled “submitter input/list discussion needed”
 - 10 labelled “PR exists/ready”
 - 4 labelled “icebox”

WebRTC-Stats Next Steps

- **Immediate goals:**
 - Wide review
 - Address CR-blocking issues
 - Get started on the test suite?
- **Candidate Recommendation**
 - MTI stats documented in webrtc-pc
 - How to test, verify interoperability
- **Ultimate goal: PR**
 - Interoperability requirements
 - Currently, browsers implement some parts of the getStats() Promise API

WebRTC-Stats Experimentation

- **We have running code, sometimes we want to measure things, how do we do that?**
- **How to experiment with non-standard metrics?**
 - **Need to have a definition for the experimental metrics.**
 - **Return the data in the current objects?**
 - **How do we name these?**
 - **Prefix experiments with “exp” before the metric name.**
- **How to document these, in a**
 - **separate section of webrtc-stats, or**
 - **separate doc**

WebRTC New Work

W3C TPAC
November 7, 2017
Burlingame, CA

Presenters: Peter, Sergio

For Discussion Today

- Potential new functionality
 - Scalable video coding (Sergio Garcia Murillo)
 - Full (fuller?) simulcast support
 - QUIC (Peter Thatcher)
 - Fine grain media stack control (not based on SDP)
 - Working methods

Objectives for this session

1. **What:** What new things are potentially of interest?
2. **Working methods:** When we want to add things, what do we do? Options:
 - A. Add to existing docs
 - B. Create new docs
 - C. ???

Things definitely "post 1.0"

- Fine grain media stack control (not based on SDP)
- QUIC data channels
- Controls on congestion control (such as a start BWE)
- More flexible simulcast
- Exposing encoders/decoders directly?
- Plugging in parts with WebASM (encoders, encryption)

QUIC data channels

Idea: Send data with QUIC instead of SCTP + DTLS

- Lots of app developers are asking for it
- Easier to terminate than DTLS+SCTP (esp. on server)
- Faster to setup (fewer round trips)
- Easier to get real-time congestion control (BBR)
- We can add back-pressure from the receive side (better support for sending large files, etc)
- New possibilities: HTTP from browser?

Proposal: add QUIC data channels to WebRTC

- It's easy and simple!
- **Add a QuicTransport and a QuicStream** independent of PC
 - Defined in ORTC CG already
 - Prototype in Chrome implemented
- Maybe some minimal IETF work as well to along
 - a draft of multiplexing QUIC with other things
 - The QUIC WG has a lot of work to do, but we mostly just wait and benefit from what they define

Optional long-term work not proposed here

- Standardize a mapping from DataChannel to QUIC streams
- Define a way to offer/answer QUIC in SDP
- Add QUIC to PeerConnection
- It's more difficult and complex, but doable if desired

QuicTransport

Like DtlsTransport: does a crypto handshake based on local certificates and remote fingerprints.

Like SctpTransport: creates data channels/streams and has an event for when the remote side creates channels/streams

QuicTransport

[Constructor(IceTransport, sequence<Certificate>)]

```
interface QuicTransport {  
    readonly attribute QuicTransportState state;  
    QuicParameters getLocalParameters();  
    sequence<ArrayBuffer> getRemoteCertificates();  
    void start(QuicParameters remoteParameters);  
    void stop();  
    QuicStream createStream();  
    attribute EventHandler onstream;  
    attribute EventHandler onstatechange;  
};
```

Example: setup

```
var ice = ...;
var certs = ...;
var quic = new QuicTransport(ice, certs);
var remoteParameters = signalExchange(quic.getLocalParameters())
quic.start(remoteParameters);
quic.onstatechange = (evt) => {
    if (evt.state == "connected") {
        // :)
    }
}
```

QuicStream

- Maps very closely to a QUIC stream
 - A bidirectional stream of bytes, like a TCP stream
 - But you can have *a lot* of them
- An RTCDataChannel can be built on top
- Other things can be built on top
 - WHAT WG streams
 - HTTP server?
- Backpressure from the receive-side is possible

QuicStream

```
interface QuicStream {
    readonly attribute QuicStreamState state;
    readonly attribute unsigned long readBufferedAmount;
    readonly attribute unsigned long writeBufferedAmount;
    setTargetReadBufferedAmount(unsigned long amount);
    unsigned long readInto(Uint8Array data);
    void write(Uint8Array data);
    void finish();
    void reset();
    void attribute EventHandler onstatechange;
    Promise waitForReadable(amount); // like waitForReadBufferedAmountBelow
    Promise waitForWritable(amount, ...); // like waitForWriteBufferedAmountBelow
```

Example: small, unordered, unreliable

```
// Send side for each message
var qstream = quic.createStream();
qstream.write(data);
qstream.finish();
// Could be replaced with a max time or max rtx policy
setTimeout(() => qstream.reset(), 5000);

// Receive side for each message
quic.onstream = (qstream) => {
    qstream.readInto(buffer);
};
```

Example: big, reliable, ordered

```
let qstream = quic.createStream();
for chunk in chunks {
  await qstream.waitForWritable(chunk.byteLength, targetBuffer); // back pressure
  qstream.write(chunk);
}
qstream.finish();
```

```
quic.onstream = qstream => {
  await qstream.waitForReadable();
  while (qstream.state == "open") {
    qstream.readInto(buffer); // back pressure
    await qstream.waitForReadable();
  }
}
```

Example:ordered data channels

- One QuicStream
- Framing in the stream to separate messages
- OPEN message is the first message

Could be defined in IETF, but doesn't need to be. Can be implemented in JS.

Example:unordered channels

- One QuicStream per message
- Data Channel ID at front of stream
- OPEN message is separate message

Could be defined in IETF, but doesn't need to be. Can be implemented in JS.

Roadblocks

- QUIC transport protocol stabilization
- Demuxing with RTP/RTCP, DTLS, and ICE
- QuicTransport needs an IceTransport, preferably without a PeerConnection

QUIC transport protocol stabilization

Still stabilizing in regards to:

- Unidirectional vs. bidirectional vs. both
- QUIC stream states (reset vs. finish)

Demuxing with RTP/RTCP, DTLS, and ICE

Good news: it *probably* can be done with no change to QUIC:
draft-aboba-avtcore-quic-multiplexing-01

But it could be cleaner with changes to QUIC or an extra byte of framing. Read the draft if you're interested.

IceTransport

A QuicTransport needs an IceTransport. Where does it get it?

The easiest thing to do is to allow constructing an IceTransport without a PeerConnection.

With something like the next slide.

IceTransport

```
partial interface IceTransport {  
    void start(IceGatherer, IceParameters remote, IceRole);  
    void stop();  
    void addRemoteCandidate(IceCandidate);  
}
```

```
interface RTCIceGatherer {  
    IceParameters getLocalParameters();  
    void gather(optional RTCIceGatherOptions);  
    void close();  
    attribute EventHandler onlocalcandidate;  
}
```

IceTransport example

```
var gatherer = new IceGatherer();
gatherer.gather();
signalLocalIceParameters(gatherer.getLocalParameters());
gatherer.onlocalcandidate = signalLocalIceCandidate;
var ice = new IceTransport();
onSignaledRemoteIceParameters = (remoteParameters) => {
    ice.start(gatherer, remoteParameters, role);
};
onSignaledRemoteIceCandidate = (remoteCandidate) => {
    ice.addRemoteCandidate(remoteCandidate);
}
```

Summary of Proposal

Add QuicTransport

Add QuicStream

Add IceTransport constructor/methods

Add IceGatherer

Great! Now we'll have p2p QUIC data streams in WebRTC.

Question: Should this be added to the CR, or in a separate doc?
(or hybrid: ICE in CR; QUIC in separate doc)?

Fine grain media stack control (not SDP)

Goals:

- interop with PeerConnection
- no SDP/RtpTransceiver
- Can construct and configure
 - IceTransport
 - DtlsTransport
 - RtpSender
 - RtpReceiver
 - SctpTransport
 - DataChannel
- possible to add new transport protocols

One last attempt at content hint

Use case

Has a capture card
sets "detailed"

Has a game screencapture
sets "motion"

Has a stream music app
sets "music"

Effect on PC (especially w/ balanced):

Gets same behavior as screencast:
high resolution/quality; fps degrades;
denoising off; maybe special code mode

Gets same behavior as camera:
high fps; resolution/quality degrades;
denoising off

Gets intelligibility enhancement off (and
other non-standard implicit speech
processing); noise suppression; special codec
mode; higher bitrate;

One last attempt at content hint

Use case

Has a capture card
sets "detailed"

Has a game screencapture
sets "motion"

Has a music app sets "music"

Effect on MediaRecorder

Gets same implicit encoding behavior as tab / screen capture (can't be overridden otherwise).

Gets same implicit encoding preferences as UVC (can't be overridden otherwise).

More bits allocated to the audio stream (w/o audioBitsPerSecond set). Default codec set as appropriate for music if not provided (user doesn't need to know what Opus is or good music target bitrate for specific codecs).

Wrap-up and Action Items

W3C TPAC
November 7, 2017
Burlingame, CA

Presenters: Bernard, Stefan, Harald

Wrap-up items

- Jan-Ivar: MediaCapture and Streams additional
- Peter: an even scarier proposal
- Cullen: scary ICE
- Dom: Whither NV?
- Peter: content hints redux

Issue 434 / PR 440 allowUserMedia

Mediacapture-main still references allowUserMedia:

```
<iframe allowUserMedia=true sandbox="allow-scripts" src="/foo">
```

Chrome 64 implements* different syntax for iframes as part of Feature Policy:

```
<iframe allow="camera" allow="microphone" sandbox="allow-scripts" src="/foo">
```

Firefox and Safari are looking to do the same

*) Chrome warns: "VideoCapture permission has been blocked because of a Feature Policy applied to the current document. See <https://goo.gl/EuHzyy> for more details."

Action Items

- * SVC + Simulcast few features: Not considering new features for WebRTC 1.0 (could be considered as a separate document)
- * QUIC: QUIC Transport document, ICE transport extension and consider an ICE constructor in WebRTC
- * Preference for Scary A over Scary B
- * Project snowflake: put together an IETF document on a subset of ICE, W3C API document to access the subset.

More scary proposal: audio/video over QUIC

Two options:

Option A: low-level access to encoders/decoders

Option B: higher-level access to send/receive

Option A: low-level access to encoders/decoders

```
interface VideoEncoder {  
    Promise<EncodedVideoFrame> encodeVideoFrame(track, VideoEncodeParameters);  
}  
  
interface VideoDecoder { // Also a jitter buffer  
    void decodeVideoFrame(EncodedVideoFrame frame, MediaStreamTrack sink);  
}
```

Option A: low-level access to encoders/decoders

```
// To send a video frame
var encoded_frame = await encoder.encodeVideoFrame(track, {bitrate: ...});
var serialized_frame = serialize_frame(encoded_frame);
var qstream = quicTransport.createQuicStream();
qstream.write(serialized_frame); qstream.finish();
```

```
// To receive a video frame
var track = new MediaStreamTrack("video");
var buffer = ...; // Anyone remember how to make a buffer?
quicTransport.onstream = (qstream) => {
  qstream.readInto(buffer);
  var encoded_frame = deserialize_frame(buffer);
  decoder.decodeVideoFrame(encoded_frame, track);
}
render_track(track);
```

Option A: low-level access to encoders/decoders

```
dictionary VideoEncodeParameters {
    unsigned long bitrate;
    boolean generateKeyFrame;
    // TODO: resolutionScale, framerateScale, ...
}

dictionary EncodedVideoFrame {
    unsigned long id;
    unsigned short width;
    unsigned short height;
    unsigned short rotation;
    unsigned long capturedTimestampMs;
    ByteArray encodedData;
    // TODO: SVC, vp8, vp9, h264, and h265-specific parameters, ...
}
```

Option A: low-level access to encoders/decoders

```
interface AudioEncoder {  
    Promise<EncodedAudioFrame> encodeAudioFrame(track, AudioEncodeParameters);  
}  
  
interface AudioDecoder { // Also a jitter buffer  
    void decodeAudioFrame(EncodedAudioFrame, MediaStreamTrack sink);  
}
```

Option A: low-level access to encoders/decoders

```
// Send an audio frame
var encoded_frame = encoder.encodeAudioFrame(track, {ptime: 20});
var serialized_samples = serialize_samples(encoded_frame);
var qstream = quicTransport.createStream();
qstream.write(encoded_samples); qstream.finish();
```

```
// Receive an audio frame
var track = new MediaStreamTrack("audio");
quic.onstream = function(qstream) {
  var buffer = ...; // Anyone remember how?
  serialized_frame = qstream.readInto(buffer);
  encoded_frame = deserialize_frame(serialized_frame);
  decoder.decodeAudioFrame(encoded_samples, track);
}
layout_track(track);
```

Option A: low-level access to encoders/decoders

```
dictionary AudioEncodeParameters {  
    unsigned long frameSize; // aka ptime, in ms  
    unsigned long bitrate;  
}
```

```
dictionary EncodedAudioSamples {  
    // Effectively a start and end time stamp, but in terms of a "sample rate clock"  
    unsigned long startSampleIndex;  
    // Theoretical endSampleIndex == startSampleIndex + sampleCount  
    unsigned int sampleCount;  
    ByteArray encodedData;  
    // TODO: opus-specific parameters?  
}
```

Option B: high-level access to send/receive

```
interface VideoSender {  
    void setTransport(QuicTransport);  
    void setTrack(MediaStreamTrack);  
    void sendVideo(VideoSendParameters);  
}
```

```
dictionary VideoSendParameters {  
    DOMString muxId;  
    DOMString codecId; // Obtained from capabilities  
    unsigned long maxBitrate;  
    RTCDegradationPreference degradationPreference = "balanced";  
    // TODO: fec, resolutionScale, framerateScale  
}
```

Option B: high-level access to send/receive

```
interface VideoReceiver {  
  readonly attribute MediaStreamTrack track;  
  void setTransport(QuicTransport);  
  Promise receiveVideo(); // Resolves when the remote side sends a stream  
}
```


Scary ICE

Api to:

- Gather candidates (filtered like today)
- Send STUN requests (restricted like today)
- Notify of STUN requests

Advantages:

- Not bilateral
- Testable
- Very little new code

One last attempt at content hint

Use case

Has a capture card
sets "detailed"

Has a game screencapture
sets "motion"

Has a stream music app
sets "music"

Effect on PC (especially w/ balanced):

Gets same behavior as screencast:
high resolution/quality; fps degrades;
denoising off; maybe special code mode

Gets same behavior as camera:
high fps; resolution/quality degrades;
denoising off

Gets intelligibility enhancement off (and
other non-standard implicit speech
processing); noise suppression; special codec
mode; higher bitrate;

One last attempt at content hint

Use case

Has a capture card
sets "detailed"

Has a game screencapture
sets "motion"

Has a music app sets "music"

Effect on MediaRecorder

Gets same implicit encoding behavior as tab / screen capture (can't be overridden otherwise).

Gets same implicit encoding preferences as UVC (can't be overridden otherwise).

More bits allocated to the audio stream (w/o audioBitsPerSecond set). Default codec set as appropriate for music if not provided (user doesn't need to know what Opus is or good music target bitrate for specific codecs).

For extra credit



Name that bird!

Thank you

Special thanks to:

W3C/MIT for WebEx

WG Participants, Editors & Chairs

The bird