

W3C WebRTC WG Meeting

October 17, 2023
8 AM - 10 AM

Chairs: Bernard Aboba
Harald Alvestrand
Jan-Ivar Bruaroey

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 October 2023 interim meeting of the W3C WebRTC WG, at which we will cover:
 - Congestion Control API, MediaCapture-ScreenShare, MediaCapture-Extensions, WebRTC-Extensions, WebRTC-PC, WebRTC Extended Use Cases, SDP negotiation
- Future meetings:
 - [November 21](#)
 - [December 12](#)
 - [January 16](#)
 - [February 20](#)
 - [March 19](#)

About this Virtual Meeting



- Meeting info:
 - Now available to WG members via the Calendar.
- Link to latest drafts:
 - <https://w3c.github.io/mediacapture-main/>
 - <https://w3c.github.io/mediacapture-extensions/>
 - <https://w3c.github.io/mediacapture-image/>
 - <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-extensions/>
 - <https://w3c.github.io/webrtc-stats/>
 - <https://w3c.github.io/mst-content-hint/>
 - <https://w3c.github.io/webrtc-priority/>
 - <https://w3c.github.io/webrtc-nv-use-cases/>
 - <https://github.com/w3c/webrtc-encoded-transform>
 - <https://github.com/w3c/mediacapture-transform>
 - <https://github.com/w3c/webrtc-svc>
 - <https://github.com/w3c/webrtc-ice>
- Link to Slides has been published on [WG wiki](#)
- Scribe? IRC <http://irc.w3.org/> Channel: [#webrtc](#)
- The meeting is (still) being recorded. The recording will be public.
- Volunteers for note taking?

W3C Code of Conduct

- This meeting operates under [W3C Code of Ethics and Professional Conduct](#)
- We're all passionate about improving WebRTC and the Web, but let's all keep the conversations cordial and professional

Virtual Interim Meeting Tips

This session is (still) being recorded

- Click  Raise hand to get into the speaker queue.
- Click  Lower hand to get out of the speaker queue.
- Please wait for microphone access to be granted before speaking.
- If you jump the speaker queue, you will be muted.
- Please use headphones when speaking to avoid echo.
- Please state your full name before speaking.
- Poll mechanism may be used to gauge the “sense of the room”.

Understanding Document Status

- Hosting within the W3C repo does ***not*** imply adoption by the WG.
 - WG adoption requires a Call for Adoption (CfA) on the mailing list.
- Editor's drafts do ***not*** represent WG consensus.
 - WG drafts ***do*** imply consensus, once they're confirmed by a Call for Consensus (CfC) on the mailing list.
 - Possible to merge PRs that may lack consensus, if a note is attached indicating controversy.

Issues for Discussion Today

- 08:10 - 08:30 AM Congestion Control API (Harald)
- 08:30 - 08:50 AM Mediacapture-screenshare (Elad)
- 08:50 - 09:20 AM Grab Bag (Henrik, Jan-Ivar & Fippo)
- 09:20 - 09:40 AM SDP Negotiation (Harald)
- 09:40 - 09:55 AM WebRTC Extended Use Cases (Bernard & Sun)
- 09:55 - 10:00 AM Wrapup and Next Steps (Chairs)

Time control:

- A warning will be given 2 minutes before time is up.
- Once time has elapsed we will move on to the next item.

Congestion Control API (Harald)

Start Time: 08:10 AM

End Time: 08:30 AM

Encoded Transform Congestion Control: Why

- Network capacity is (sometimes) limited
- Sending more than network capacity is Not A Good Thing
- Browser will police the sender - sending more causes discards

So what does this mean for EncodedTransform?

- With no transform: Transport divides capacity between channels, tells the encoders to behave themselves
 - Transport estimates overhead and subtracts
- With simple transform (no big change in frame size): Don't have to change
- If encoder's size and transform's size don't match at all, what?
 - Variable size "additional information" (Alpha channel, segmentation)
 - Frames injected from other sources (Late Fanout use case)
- Transform needs to know!

Congestion Control: How?

<https://github.com/w3c/webrtc-encoded-transform/pull/207>

- Expose attributes to say what the “downstream” available bandwidth is
- Add events to tell of “significant changes”
- Add methods to tell “upstream” of available bandwidth
- Use cancellable events to distinguish between “framework should do it” and “I will do it myself”

IDL of proposed API

```
interface mixin RTCRtpScriptSource {  
  readonly attribute ReadableStream readable;  
  Promise<unsigned long long> generateKeyFrame(optional DOMString  
rid);  
  
  Promise<undefined> sendKeyFrameRequest();  
  undefined sendBandwidthEstimate(BandwidthInfo info);  
};
```

```
interface mixin RTCRtpScriptSink {  
  readonly attribute WritableStream writable;  
  attribute BandwidthInfo bandwidthInfo;  
  
  attribute EventHandler onbandwidthestimate;  
  attribute EventHandler onkeyframerequest;  
};
```

```
[Exposed=DedicatedWorker]  
interface RTCRtpScriptTransformer {  
  readonly attribute any options;  
};  
RTCRtpScriptTransformer includes RTCRtpScriptSource;  
RTCRtpScriptTransformer includes RTCRtpScriptSink;
```

Blue: Restructuring for clarity

Yellow: New API pieces

```
interface BandwidthInfo {  
  readonly attribute long allocatedBitrate;  
  // bits per second  
  
  readonly attribute long availableOutgoingBitrate;  
  
  readonly attribute boolean writable;  
};
```

Example API usage

Passthrough, but want to handle it myself:

```
transformer.onbandwidthestimate = (ev) => {  
  ev.preventDefault();  
  sendBandwidthEstimate(this.bandwidthInfo);  
}
```

I'm adding a kilobyte per second:

```
Transformer.onbandwidthestimate = (ev) => {  
  let temp = this.bandwidthInfo;  
  temp.allocatedBitrate -= 8000;  
  this.sendBandwidthEstimate(temp);  
  ev.preventDefault();  
}
```

Not addressed in this proposal

- Reading and modifying allocation of bandwidth between senders
- Other actions than “modify upstream bandwidth targets”
 - However, all the other APIs for influencing senders remain available

Discussion (**End Time: 08:30**)

-

Mediacapture-screenshare (Elad)

Start Time: 08:30 AM

End Time: 08:50 AM

For Discussion Today

- MediaCapture-screenshare
 - [Issue 276](#): Handling of contradictory hints
 - [Issue 281](#): Distinguish cancellations from absent OS permissions

Contradictory hints - example #1

```
// Audio generally not requested,  
// but system-audio marked as desired.  
navigator.mediaDevices.getDisplayMedia({  
  audio: false,  
  systemAudio: "include",  
});
```

Contradictory hints - example #2

```
// Audio requested, including an explicit
// request for system-audio,
// but monitors asked to be excluded.
navigator.mediaDevices.getDisplayMedia({
  audio: true,
  systemAudio: "include",
  monitorTypeSurfaces: "exclude"
});
```

Contradictory hints - example #3

```
// Application requested monitors to be
// displayed most prominently,
// while simultaneously asking for monitors
// to not be offered.
navigator.mediaDevices.getDisplayMedia({
  video: { displaySurface: "monitor" },
  monitorTypeSurfaces: "exclude"
});
```

Handling of contradictory hints - proposal

4. Let *options* be the method's first argument.
 5. Let *constraints* be [*options.audio*, *options.video*].
 6. If *constraints.video* is *false*, return a promise rejected with a newly created TypeError.
-

Above is a snapshot of the `getDisplayMedia` algorithm in the spec.

Proposal:

- Add a step for validating the interaction between constraints and options and reject if a contradiction is detected.
- (Probably create a subroutine for it, naming specific possible contradictions.)

Issue 281: Missing OS permissions - problem description

Sometimes screen-sharing is blocked because the user presses cancel; maybe the user changed their mind.

Sometimes screen-sharing is blocked because the OS permissions are configured to block. In that case, some applications might wish to explain that to the user.

Issue 281: Missing OS permissions - proposal

```
enum GetDisplayMediaNotAllowedReason {
    "user-rejected",
    "operating-system-disallowed",
};

dictionary GetDisplayMediaNotAllowedErrorInit {
    required GetDisplayMediaNotAllowedReason reason;
};

interface GetDisplayMediaNotAllowedError : DOMException {
    constructor(optional DOMString message = "", GetDisplayMediaNotAllowedErrorInit init);
    readonly attribute GetDisplayMediaNotAllowedReason reason;
};
```

(The ctor sets the name attribute to the value “NotAllowedError”.)

Issue 281: Possible objection

Possible objection: “Shouldn’t the UA inform the user?”

Answers:

- Not mutually exclusive - we could do both.
- Which change is more likely to happen in the foreseeable future?
 - A bespoke error is trivial to implement.
 - Custom UX to surface an error... Only sounds trivial until you try.

Discussion (**End Time: 08:50**)

-

Grab Bag (Henrik, Jan-Ivar & Fippo)

Start Time: 08:50 AM

End Time: 09:20 AM

For Discussion Today

- **Mediacapture-extensions**

- [PR #117](#): Add MediaStreamTrackAudioStats interface (Henrik)

- **Mediacapture-main**

- [Issue 972](#): Racy devicechange event design poor interop (Jan-Ivar)
- [Issue 966](#): Should devicechange fire when the device info changes? (Jan-Ivar)

- **WebRTC-Extensions**

- [Issue 146](#): Exposing decode errors / SW fallback as an event (Fippo)

- **WebRTC-PC**

- [Issue 2888](#): setCodecPreferences vs. unidirectional codecs (Fippo)

PR #117: Add `MediaStreamTrackAudioStats` interface (Henrik)

`track.stats` now exposes **video** frame counters (*delivered, discarded, total*).

What about **audio** stats? [Issue #96](#): *Migrate capture metrics from `RTCAudioSourceStats` to `MediaStreamTrack` method.*

- Previously covered in the [April 18th Virtual Interim](#).
- **Decision:** Move stats, but express them in terms of *audio frames*, not *audio samples*.

The PR for this never landed due to discussions around API shape. **New PR:**

WebIDL

```
partial interface MediaStreamTrack {  
  [SameObject] readonly attribute  
    (MediaStreamTrackAudioStats or MediaStreamTrackVideoStats) stats;  
};
```

PR #117: Add `MediaStreamTrackAudioStats` interface (Henrik)

Delivered and dropped frames definition are similar to the video stats.

Measuring delay:

$\text{deliveredFramesDelay} / \text{deliveredFrames}$

is the average delay of each audio frame from capture to delivery.

Measuring glitches:

$\text{droppedFramesDuration} / (\text{deliveredFramesDuration} + \text{droppedFramesDuration})$

is the percentage of audio that was dropped
(i.e. not processed in a timely manner).

Alternatively totalFrames instead of droppedFrames?

Where $\text{droppedFrames} = \text{totalFrames} - \text{deliveredFrames}$

For consistency with video stats.

WebIDL

```
[Exposed=Window]
interface MediaStreamTrackAudioStats {
  readonly attribute unsigned long long deliveredFrames;
  readonly attribute DOMHighResTimeStamp deliveredFramesDuration;
  readonly attribute DOMHighResTimeStamp deliveredFramesDelay;
  readonly attribute unsigned long long droppedFrames;
  readonly attribute DOMHighResTimeStamp droppedFramesDuration;
  [Default] object toJSON();
};
```

Issue 972: Racy devicechange event design has poor interoperability (Jan-Ivar)

The [devicechange](#) event follows [§ 7.7. Use plain Events for state](#), but its "state information in the [target](#) object." is **not** available synchronously, a footgun: 🦶🔫

```
navigator.mediaDevices.ondevicechange = async () => {  
  const devices = await navigator.mediaDevices.enumerateDevices();  
  // 100+ milliseconds may have passed (letting devicechange fire multiple times!)  
  // The app compares devices vs. app.oldDevices to detect insertion/removal  
  app.oldDevices = devices; // stored state  
}
```

100+ milliseconds may pass before the app examines `devices`, during which Chrome fires `devicechange` [7 more times](#) (for putting on my AirPods). The async app code runs 8 times interleaved with itself. Any application state access becomes suspect.

Hard to reason about for no reason → Trial & error to pass QA → reliance on browser side-effects undetected → fails in the field in other browsers → poor interoperability.

Issue 972: Racy devicechange event design has poor interoperability (Jan-Ivar)

Proposal: Include the devices in the event:

```
navigator.mediaDevices.ondevicechange = ({devices}) => {  
  // The app compares devices vs. oldDevices to detect changes  
  app.oldDevices = devices;  
}
```

Avoids all races and is 100% backwards compatible (modeled on [trackEvent.streams](#)).

Issue 966: Should devicechange fire when the device info changes? (Jan-Ivar)

The spec [says](#): No.

Event name	Interface	Fired when...
devicechange	Event	The set of media devices, available to the User Agent , has changed. The current list devices can be retrieved with the enumerateDevices() method.

When new media input and/or output devices are [made available to the User Agent](#), or any available input and/or output device becomes [unavailable](#), or the [system default](#) for input and/or output devices of a [MediaDeviceKind](#) changed, the [User Agent](#) **MUST** run the following **device change notification steps** for each [MediaDevices](#) object, *mediaDevices*, for which [device enumeration can proceed](#) is **true**, but for no other [MediaDevices](#) object:

It indicates a change in devices available to the *browser* (i.e. OS change). Apps rely on it to support auto-switching in reaction to users inserting or removing devices during calls.

Safari incorrectly fires it as part of getUserMedia, [confusing](#) apps → poor interop.

Issue 966: Should devicechange fire when the device info changes? (Jan-Ivar)

To capture #966 discussion: imagine a rhetorical `enumeratedevicechange` event that fires for any delta in `enumerateDevices` output. It would include firing for non-OS reasons: changes in the site's device information exposure or its permission.

Use case: Refresh an in-content picker

Option A: no change (workaround: app calls `enumerateDevices` after gUM)

Option B: add a new `enumeratedevicechange` event

Option C: change `devicechange` to work like `enumeratedevicechange`

- C seems web incompatible (would interfere with auto-switching, e.g. Safari)
- Device information exposure is already deterministic (set in gUM)
- Permission loss is already detectable using `permissions.query`

Proposal: A

Issue 146: Exposing decode errors / SW fallback as an event

- [Discussed at TPAC](#)
 - Generally in favor of “Proposal B”
 - Also expose error event on RTCRtpSender and RTCRtpReceiver
 - Confirm that decision
 - Minor changes to naming
 - rtpTimestamp instead of timestamp
 - spatialIndex instead of rid
 - Distinct errors for sender & receiver or common one?
 - spatialIndex only known for sender
 - spatialIndex only for simulcast
 - SVC encoders considered monolithic
 - Ready for PR?
- Proposal B: Custom event (like [RTCPeerConnectionIceErrorEvent](#))

[Exposed=Window]

```
interface RTCRtpSenderErrorEvent : Event {  
  constructor(DOMString type, RTCRtpSenderErrorEventInit eventInitDict);  
  readonly attribute DOMString? rid;  
  readonly attribute unsigned short errorCode;  
  readonly attribute USVString errorText;  
  readonly attribute long long timestamp;  
};
```

Issue 2888: setCodecPreferences vs. unidirectional codecs

- libWebRTC: call setCodecPreferences with H264 only and get VP8
 - Drive-by discovery from H264 woes on Android
- Root cause: some codecs are send-only, others receive-only
 - H264 profiles in particular
 - but also VP9 and some FEC mechanisms
 - Windows: 21 receive codecs, 14 send codecs
- Where does input for setCodecPreferences come from?
 - Assumption: RTCRtpSender.getCapabilities('video').codecs
 - Then reorder or filter the list

Issue 2888: setCodecPreferences vs. unidirectional codecs

- setCodecPreferences algorithm:

6. If the intersection between *codecs* and `RTCRtpSender.getCapabilities(kind).codecs` or the intersection between *codecs* and `RTCRtpReceiver.getCapabilities(kind).codecs` only contains RTX, RED or FEC codecs or is an empty set, throw `InvalidModificationError`. This ensures that we always have something to offer, regardless of *transceiver.direction*.

- Looks at both send and receive sets
 - Regardless of transceiver direction
- Does not specify how codecs are compared
 - H264 profile-level-asymmetry-allowed

Issue 2888: setCodecPreferences vs. unidirectional codecs

- Take directionality into account (from [comment](#))
 - setCodecPreferences with a recv-only codec on a sendrecv or sendonly transceiver should throw
 - Incompatible changes to transceiver direction after setCodecPreferences should throw
- Specify how codecs are compared
 - Already done for SDP O/A

Ready for PR?

Discussion (**End Time: 09:20**)

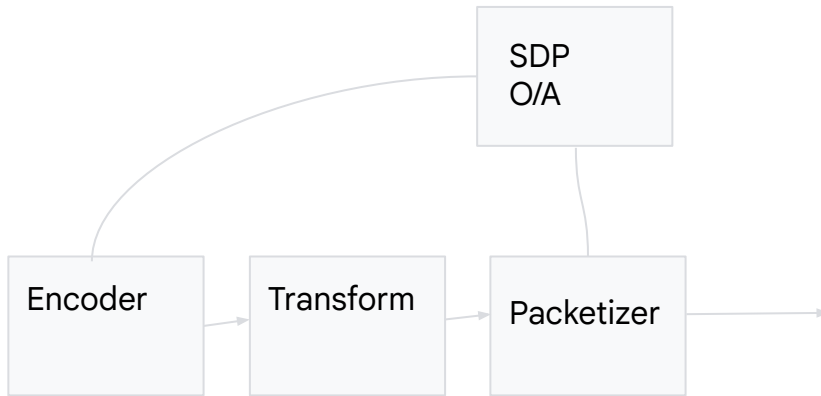
-

SDP Codec Negotiation (Harald)

Start Time: 09:20 AM

End Time: 09:40 AM

SDP Negotiation - The Model



- The SDP O/A configures the available codec lists for encoder and packetizer
- SetParameters() allows to pick the codec from that list (new API)
- New functionality is needed to get the right things to happen for E2EE and frame-level metadata

SDP negotiation - What To Do

Before negotiating with the peer, both sides of an app using a transform should:

- Add codec (name and parameters) that will be offered in O/A

Nothing in the platform understands those names; both sides have to add the same names.

If the peer agrees it understands those names, a PT will be assigned to it.

SDP negotiation - processing data

Before sending data:

- Choose explicitly the encoder to use for encoding, using `SetParameters`
- Tell the transform what PT to assign to post-transform frames
- Tell the packetizer how to packetize that PT

Before receiving data:

- Tell the depacketizer how to depacketize the PT agreed on
- Tell the transform what PT to expect, and what PT to produce
- Tell the decoder to decode frames using the produced PT using the relevant decoder

The transform transforms, modifying metadata as needed.

Timeline of proposal

- March 21, 2023: Issue and outline of proposal [presented](#) at March interim
- June 14, 2023: First version of PR [#186](#) uploaded
- June 27, 2023: Proposal [presented](#) at June interim.
 - [Minutes](#) say: “RESOLUTION: Adopt PR #186 with details to be discussed in the PR”
- Sept 12, 2023: Presented at TPAC. [Minutes](#) say:
 - “Summary: PT to MimeType -> arguments on both sides. Setting packetisers is also accepted but needs further discussion.”
- Oct 12, 2023: Revised PR with TPAC changes discussed at editors meeting.

The result of the Oct 12 discussion was a number of arguments that suggested the whole approach should be rethought and different approaches considered.

Some of those are now mentioned on the PR discussion.

A FAQ section has been added to the explainer on some of the choices.

Evaluation of status

- 7 months and 3 presentations to WG should be enough to expect that the basic approach has consensus.
- Details can always be changed later, but blocking the PR hampers experimentation.
- Proposal: Instruct the editors team to merge the PR as-is.

Discussion (**End Time: 09:40**)

-

WebRTC Extended Use Cases

Start Time: 09:40 AM

End Time: 09:55 AM

For Discussion Today

- [Section 3.2: Low Latency Streaming](#)

Status of Section 3.2: Low Latency Streaming

- [Section 3.2: Low Latency Streaming](#)
 - [Section 3.2.1: Game Streaming](#)
 - [Section 3.2.2: Low Latency Broadcast with Fanout](#)
- CfC concluded on January 16, 2023: [Summary](#)
 - 6 responses received, 5 in support, 1 no opinion
 - Open Issues mentioned in responses:
 - [Issue 80](#): Access to raw audio data (TPAC 2023: move to audio codec use case)
 - [Issue 103](#): Feedback related to WebRTC-NV Low Latency Streaming Use Case
 - [PR 124](#): Requirement N38 is satisfied by jitterBufferTarget
 - Closed issues mentioned in responses:
 - [Issue 85](#): What is a "node" in the low latency broadcast with fanout use case?
 - [Issue 86](#): Is the DRM requirement in the Low latency Broadcast with Fanout use case satisfied by data channels?
 - [Issue 91](#): N15 latency control should be formulated in a technology-agnostic way
 - [Issue 94](#): Improvements for game pad input
 - [Issue 95](#): Low-latency streaming: Review of requirements

Section 3.2.2: Low Latency Broadcast w/fanout

§ 3.2.2 Low latency Broadcast with Fanout

There are streaming applications that require large scale as well as low latency. Examples include sporting events, church services, webinars and company 'Town Hall' meetings. Live audio, video and data is sent to thousands (or even millions) of recipients. Limited interactivity may be supported, such as allowing authorized participants to ask questions at a company meeting. Both the media sender and receivers may be behind a NAT. P2P relays may be used to improve scalability, potentially using different transport than the original stream.

NOTE

This use case has completed a Call for Consensus (CfC) [CFC-Low-Latency] but has unresolved issues.

Requirement ID	Description
N15	The application must be able to take steps to ensure a low and consistent latency for audio, video and data under varying network conditions. This may include tweaking of transport parameters for both media and data.
N39	A user-agent must be able to forward media received from a peer to another peer. Applications require access to encoded chunk metadata as well as information from the RTP header to provide for timing, media configuration and congestion control. This includes a mechanism for a relaying peer to obtain a bandwidth estimate.

Experience: *pipe*, Peer5 and Dolby are examples of this use case, with media transported via RTP or RTCDatChannel.

Section 3.2.2: Low Latency Broadcast w/fanout

- [PR 123](#): Section 3.2.2: Clarify Use Cases
 - Focus is now on auctions (as originally suggested in Tim's PR).
 - Church services, Webinars and Town Hall meetings removed
 - These use cases typically do not require ultra low latency and can be addressed without RTP fanout. Examples:
 - RTCDataChannel fanout (Peer5)
 - IETF MoQ, using WebCodecs + WebTransport with CDN caching
 - Sporting events removed
 - Typically requires content protection, which RTP does not support.
 - DRM supported via CMAF transport (MoQ or RTCDataChannel)
 - Support for content protection under investigation in WebCodecs.
 - [Issue 41](#): Support for content protection

PR 123: Section 3.2.2: Clarify Use Cases

§ 3.2.2 ~~Ultra~~ Low latency Broadcast with Fanout

There are streaming applications that require large scale as well as ~~ultra~~ low ~~latency~~ ~~latency (glass-glass latency less than 300ms)~~. Examples include ~~sporting events, church services, webinars and company 'Town Hall' meetings~~ ~~auctions~~. Live audio, video and data is sent to thousands (or even millions) of recipients. Limited interactivity may be supported, such as ~~allowing authorized participants to ask questions at a company meeting~~ ~~capturing audio/video from auction bidders~~. Both the media sender and receivers may be behind a NAT. ~~P2P~~ ~~Peer-to-peer~~ relays may be used to improve scalability, ~~potentially using different transport than the original stream~~ ~~with ingestion, distribution and fanout utilizing RTP~~.

~~Note This use case has completed a Call for Consensus (CfC) [CFC Low Latency] but has unresolved issues.~~

Requirement ID	Description
N15	The application must be able to take steps to ensure a low and consistent latency for audio, video and data under varying network conditions. This may include tweaking of transport parameters for both media and data.
N39	A user-agent must be able to forward media received from a peer to another peer. Applications require access to encoded chunk metadata as well as information from the RTP header to provide for timing, media configuration and congestion control. This includes a mechanism for a relaying peer to obtain a bandwidth estimate.

Experience: *pipe* ~~Peer5~~ and Dolby are examples of this use case, with media transported via ~~RTP or RTCDataChannel~~ ~~RTP~~.

Ready to merge?

Section 3.2.1: Game Streaming

3.2.1 Game streaming

Game streaming involves the sending of audio and video (potentially at high resolution and framerate) to the recipient, along with data being sent in the opposite direction. Games can be streamed either from a cloud service (client/server), or from a peer game console (P2P). It is highly desirable that media flow without interruption, and that game players not reveal their location to each other. Even in the case of games streamed from a cloud service, it can be desirable for players to be able to communicate with each other directly (via chat, audio or video).

NOTE

This use case has completed a Call for Consensus (CfC) [CFC-Low-Latency] but has unresolved issues.

Requirement ID	Description
N15	The application must be able to take steps to ensure a low and consistent latency for audio, video and data under varying network conditions. This may include tweaking of transport parameters for both media and data.
N36	An application that is only receiving but not sending media or data can operate efficiently without access to camera or microphone.
N37	It must be possible for the user agent's receive pipeline to process video at high resolution and framerate (e.g. without copying raw video frames).
N38	The application must be able to control the jitter buffer and rendering delay.

Experience: Microsoft's Xbox Cloud Gaming and NVIDIA's GeForce NOW are examples of this use case, with media transported using RTP or RTCDataChannel.

Section 3.2.1: Game Streaming

- Issues
 - [Issue 103](#): Section 3.2: Feedback relating to WebRTC-NV Low Latency Streaming Use Case
- PRs
 - [PR 124](#): Requirement N38 is satisfied by jitterBufferTarget
 - [PR 118](#): Clarify Game Streaming Requirements
 - Follow up N48, 49, 50 feedback
 - Clarification on N51 requirement

Issue #103: Feedback related to WebRTC-NV Low Latency Streaming Use Case

From: Youenn Fablet <youenn@apple.com>

Date: Mon, 16 Jan 2023 10:33:28 +0100

To: Bernard Aboba <Bernard.Aboba@microsoft.com>

Cc: "public-webrtc@W3.org" <public-webrtc@w3.org>

Message-id: <EE006548-9A1A-49F5-A313-B1A8B93C64C1@apple.com>

Both use cases are already deployed so I wonder whether they qualify as NV.

They probably qualify as NV if some of their corresponding requirements are not already met with existing web technologies.

When reading the requirements, it seems some/most of them are already met.

The term "low latency" in particular is vague even in the context of WebRTC.

Low latency broadcast with fanout is already achieved by some web sites but it is not clear where we are trying to improve upon existing deployed services.

For instance, are we trying to go to ultra low latency where waiting for an RTP assembled video frame by the proxy is not good enough?

Looking at the requirements:

- N37 is already achieved or seems like an implementation problem that is internal to User Agents.
- N38 is partially achieved via `playoutDelay` and/or WebRTC encoded transform. I am not clear whether this use case is asking for more than what is already provided.
- N39 is already achieved via data channel and/or WebRTC encoded transform without any change. I am guessing more is required. If so, can we be more specific?

Thanks,

Y

PR 124: Requirement N38 is satisfied by jitterBufferTarget

N38

The application must be able to control the jitter buffer and rendering delay. [↑This requirement is satisfied by jitterBufferTarget, defined in \[↑↑WebRTC-Extensions↑↑\] Section 6. ↑](#)

Ready to merge?

PR 118: Clarify Game Streaming requirements (Section 3.2.1)

- Rationale: Cloud Game Characteristics
 - A highly interactive application that depends on **continuous visual feedback** to user inputs.
 - The cloud gaming latency KPI would track Click to Pixel latency - time elapsed between user input to when the game response is available at the user display (where as non-interactive applications may track G2G latency as the KPI).
 - Requires low and consistent latency. Desirable C2P latency range is typically 30 - 150ms. A latency higher than 170 ms makes high precision games unplayable.
 - Loss of video is highly undesirable. Garbled or corrupt video with fast recovery may be preferable in comparison to a video freeze.
 - Motion complexity can be high during active gameplay scenes.
 - Consistent latency is critical for player adaptability. Varying latency requires players to adapt continuously which can be frustrating and break gameplay.
 - The combination of high complexity, ultra low latency and fast recovery will require additional adaptive streaming and recovery techniques.

PR 118: Clarify Game Streaming requirements (Section 3.2.1)

ID	Requirement	Description	Benefits to Cloud Gaming	Is it Cloud Gaming Specific?
N48 (New)	Recovery using non-key frames	WebRTC must support a mode allows video decoding to continue even after a frame loss without waiting for a key frame. This enables addition of recovery methods such as using frames containing intra coded macroblocks and coding units - WebRTC Issue: 15192	Players can continue to game with partially intelligible video. Fast recovery from losses on the network	Can be used by any application where video corruption is preferred to video freezes
N49 (New)	Loss of encoder -decoder synchronicity notification	The WebRTC connection should generate signals indicating to encoder about loss of encoder-decoder synchronicity (DPB buffers) and sequence of the frame loss.(RFC 4585 section-6.3.3: Reference Picture Selection Indication) - Delete of RPSI (Mar/2017)	Fast recovery from losses on network. Helps application to choose right recovery method in lossy network.	Can be used by any application where video corruption is preferred to video freezes
N50 (New)	Configurable RTCP transmission interval	Application must be able to configure RTCP feedback transmission interval - Need to set transmission interval for Transport-wide RTCP along with “WebRTC-SendNackDelayMs(field-trial)” .	Gaming is sensitive to congestion and packet loss resulting in higher latency. Consistent RTCP feedback helps application to adapt video quality to varying network (BWE and packet loss).	Can be used by any application where latency buildup is not acceptable.
N51 (New)	Improve accuracy of Jitter buffer control	Extend adaptation of the jitter buffer to account for jitter in the pipeline upto the frame render stage - WebRTC Issue: 15535	Increases accuracy of jitter buffer adaptation and helps maintain consistent latency	Helps all low latency applications, but is necessary for Cloud gaming

N48 and N49: Recovery using non-key frames (cont'd)

- IETF discussion relating to reference feedback in HEVC
 - Draft adopted in AVTCORE WG as draft-ietf-avtc core-hevc-webrtc
 - Github issues: <https://github.com/aboba/hevc-webrtc/issues>
 - In RFC 7798 Section 8.3, use of RPSI for positive acknowledgment is deprecated, used only to indicate decoding of a reference by the client.
 - HEVC usage of RPSI different from VP8 (positive acknowledgement)
 - [Guidance on RPSI implementation RP#17](#) will meet our requirements
 - Will pursue LNTF RTCP message as a short-term solution
 - Will continue to pursue on the RPSI approach and find a way to meet the codec agnostic concern raised by [RPSI RTCP feedback support · Issue #13](#)

N48 and N49: Recovery using non-key frames (cont'd)

- Regarding Non-Keyframe based Recovery
 - RTP de-packetization and framing would need to be updated to recover using non-key frame.
 - Currently RTP receiver stops providing frames to decoder on packet loss.
 - Need a way to start providing subsequent completely received non-key frames to decoder.
 - Requires decoder API support (only encoder API discussed at TPAC)

N50: Configurable RTCP transmission interval (cont'd)

- Currently [Transport-wide congestion control 02](#) provides way to configure Transport wide CC feedback
 - This is not reliable as sender request might get lost in network resulting in no feedback.
 - Sending extra headers results into reduced payload size.
 - Sender request jitter shows up in feedback receive jitter.
- We want to have a API to set interval for Transport wide CC feedback

N51: Improve accuracy of Jitter buffer control

- [As the Cloud gaming service supports higher resolution\(4K\) and higher frame rate\(120p\)](#), we found that webrtc has many assumptions on the it's implementation assuming default video frame rates 60fps and render delay as 10ms etc.
 - `third_party/webrtc/modules/video_coding/timing/timing.h`
 - `static constexpr TimeDelta kDefaultRenderDelay = TimeDelta::Millis(10);`
 - `static constexpr int kDelayMaxChangeMsPerS = 100;`
 - [1327251 - Use render time and RTP timestamps in low-latency video path - chromium](#)
 - This bug tracks the work with making the signalling to the compositor explicit and always setting a reference time as well as removing some 60fps assumptions by instead using the actual RTP timestamps to determine the frame rate.
- So it make hard to control the latency through the Jitter buffer, so want to propose the implementation for getting the correct value on the rendering on the device.
 - [15535:Jitter buffer to account for jitter in the pipeline up to the frame render stage](#)

Discussion (**End Time: 09:55**)

-

Wrapup and Next Steps

Start Time: 09:55 AM

End Time: 10:00 AM

Title Goes Here

- Content goes here

Thank you

Special thanks to:

WG Participants, Editors & Chairs