W3C WebRTC TPAC 2023 Meeting

Tuesday, September 12, 2023

11:30 - 16:30 Seville Time

09:30 - 14:30 UTC

02:30 - 07:30 Pacific Time

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

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

Safety Reminders

While attending TPAC, follow the health rules:

- Authorized masks are required indoors at all times. If you need to remove your mask during the meeting, please keep it short
- Daily test is expected

Please be aware of and respect the personal boundaries of your fellow participants

https://www.w3.org/2023/09/TPAC/health.html

About TPAC 2023 Meetings

- TPAC 2023 Schedule
- Link to slides has been published on WG wiki
- Scribe? IRC http://irc.w3.org/ Channel: #webrtc
- Will we be recording the session?
- Volunteers for note taking?
- Future WebRTC WG meetings:
  - October 17
  - November 21
  - December 12

TPAC 2023 Meeting Schedule

- WebRTC WG: September 12, 2023
  - 11:30 - 16:30 Seville Time
  - Meeting info
  - Slides
- Joint WebRTC/SCCG: September 14, 2023
  - 17:00 - 18:30 Seville Time
  - Meeting Info
  - Slides
- Joint WebRTC/MEDIA September 15, 2023
  - 14:30 - 16:30 Seville Time
  - Meeting Info
  - Slides

WebRTC Use Case Breakout Session

- **“WebRTC use cases and requirements under high-demanding real-time communication scenarios”**
  - TPAC 2023: Breakouts schedule (w3.org)
- Wednesday, September 13, 11:00 - 12:00
  - Location: Nervion I, Level -1
  - Calendar entry
- Document:
    - WebRTC live-streaming use cases

HDR Breakout Session

- **“HDR on the Web”**
  - Discussion of rendering of existing HDR content (images/video), proposed HDR APIs (canvas/WebGL/GPU), the need for more expressive media queries, and interactions with CSS colors.
- Wednesday, September 13, 17:15 - 18:15
  - Location: Nervion-Arenal II Level -1
  - Calendar entry

WebCodecs Serialization Breakout Session

- **“WebCodecs Serialization Format”**
- Wednesday, September 13, 16:00 - 17:00
  - Location: Lebrija, First Floor
  - Calendar entry

Virtual Meeting Tips (Zoom)

- **Both local and remote participants need to be on irc.w3.org channel #webrtc.**
- **Use "raise hand" to get into the speaker queue and "lower hand" to get out of the speaker queue.**
- **To try out WebCodecs over RTCDatachannel (not RTP!) join using a Browser.**
- **Please use headphones when speaking to avoid echo.**
- **Please wait for microphone access to be granted before speaking.**
- **Please state your full name before speaking.**

Today's Agenda

- 11:40 - 12:00 State of the WG (Harald Alvestrand)
- 12:00 - 12:40 WebRTC Extended Use Cases (Bernard, Harald, Sun)
- 12:40 - 13:00 Modifications for low latency fanout (Palak)
- 13:00 - 14:00 Lunch Break
- 14:00 - 14:40 WebRTC & Media Capture Issues (Henrik, Jan-Ivar)
- 14:40 - 15:10 Ice Controller API (Sameer Vijakar)
- 15:10 - 15:40 RTPTransport (Peter & Stefan)
- 15:40 - 16:00 SDP negotiation for Encoded Transform (Harald)
- 16:00 - 16:20 Topic TBD (reserved for followup)
- 16:20 - 16:30 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.

State of the WEBRTC WG (Harald Alvestrand)

Start Time: 11:40

End Time: 12:00

External Environment

- WebRTC over RTP is the dominant browser VC platform (and decent chunks of non-browser)
- WebRTC over RTP is being used in many niche applications (ex WHIP for recording)
- Explorations of other protocols (MOQ, WebCodecs over WebTransport) ongoing, but have not achieved significant deployment traction

Activity since TPAC 2022

- Repo activity
  - Mediacapture-main
    - Getting ready for REC (still) - removing non-implemented features
  - Mediacapture-extensions
    - Holding pen for new ideas
  - Webrtc-pc
    - Merging some things from -extensions (when implemented)
  - Webrtc-extensions
    - Holding pen for new ideas
  - Webrtc-stats
    - Living Standard-like. Simplification and removal of Old Stuff
  - Webrtc-nv-use-cases
    - Restructure and attempt to make useful

Major new or expanded topics

- Webrtc-encoded-transform
  - New functionality desired
- Webrtc-ice
  - New direction on how to control pursued
- Platform processing for effects and faces
  - Being pursued in some sync with Media WG
- Screen capture
  - Largely pursued in new SCCG community group

Things that seem stable

- Mediacapture-transform
- Mediacapture-record
- Mediacapture-fromelement
- Mediacapture-image
- Webtc-priority
- Mst-content-hint
- WebRTC-SVC

Discussion (End Time: 12:00)

-

WebRTC Extended Use Cases

Start Time: 12:00

End Time: 12:40

Proposals from the May and July Meetings

- Rename it. Proposal: “WebRTC Extended Use Cases”. Done.

- Focus on things that can only/best be done by WebRTC (p2p etc)
- Remove use cases that are now met by other standards
- Include use cases that have no requirements but extend RFC 7478
- Remove use cases that don’t get consensus within a few months
- Remove requirements that don’t get consensus within a few months
- Remove use cases that don’t add new requirements. Done.
- Proposed API changes should include changes to the use-case doc
- **Define the relationship between this doc and explainers**
- Define the relationship between this doc and issues
- Define the relationship between this doc and API proposals
- Broaden the input somehow - perhaps via webrtc.nu ?
- Define what we do with aspirations.

What is the relationship of a Use Case To…

- Explainers (How the proposals relate to the use cases)
  - Should explainers link to use cases?
    - Clarifies whether an API proposal is solving a use case
    - Recommendation: Explainers (and API proposals) can link to use cases, but not required.

Proposals from the May and July Meetings

- Rename it. Proposal: “WebRTC Extended Use Cases”. Done.

- Focus on things that can only/best be done by WebRTC (p2p etc)
- Remove use cases that are now met by other standards
- Include use cases that have no requirements but extend RFC 7478
- Remove use cases that don’t get consensus within a few months
- Remove requirements that don’t get consensus within a few months
- Remove use cases that don’t add new requirements. Done.
- Proposed API changes should include changes to the use-case doc
- Define the relationship between this doc and explainers
- **Define the relationship between this doc and issues**
- Define the relationship between this doc and API proposals
- Broaden the input somehow - perhaps via webrtc.nu ?
- Define what we do with aspirations.

What is the relationship of a Use Case To…

- Issues (what specific problems are being referred to)
  - Should a use case link to related Issues?
    - Advantage: Links requirements to specific Issues whose resolution can be tracked.
    - Recommendation: Use cases may link to issues raised in a CfC (or to the CfC summary)

Proposals from the May and July Meetings

- Rename it. Proposal: “WebRTC Extended Use Cases”. Done.

- Focus on things that can only/best be done by WebRTC (p2p etc)
- Remove use cases that are now met by other standards
- Include use cases that have no requirements but extend RFC 7478
- Remove use cases that don’t get consensus within a few months
- Remove requirements that don’t get consensus within a few months
- Remove use cases that don’t add new requirements. Done.
- Proposed API changes should include changes to the use-case doc
- Define the relationship between this doc and explainers
- Define the relationship between this doc and issues
- **Define the relationship between this doc and API proposals**
- Broaden the input somehow - perhaps via webrtc.nu ?
- Define what we do with aspirations.

What is the relationship of a Use Case To…

- API Proposals (what API proposals relate to the problems)
  - Should a use case link to API proposals?
    - Clarifies whether a use case has API proposals
    - Links use case to an API proposal whose progress can be tracked
    - Recommendation:
      - API proposals can link to use cases.
      - However, use cases should not be required to link to API proposals.

For Discussion Today

- Section 3.6: Funny Hats
- Section 3.2: Low Latency Streaming

Section 3.6: Additional Metadata in Frames

The experience with “Funny Hats” implementation (use case 3.6) is that we frequently encounter the need to pass information across the wire together with the encoded frame. Just like E2EE, this means that what goes on the wire is NOT in a standard codec format; unlike E2EE, the additional overhead can be considerable - enough to influence congestion control.

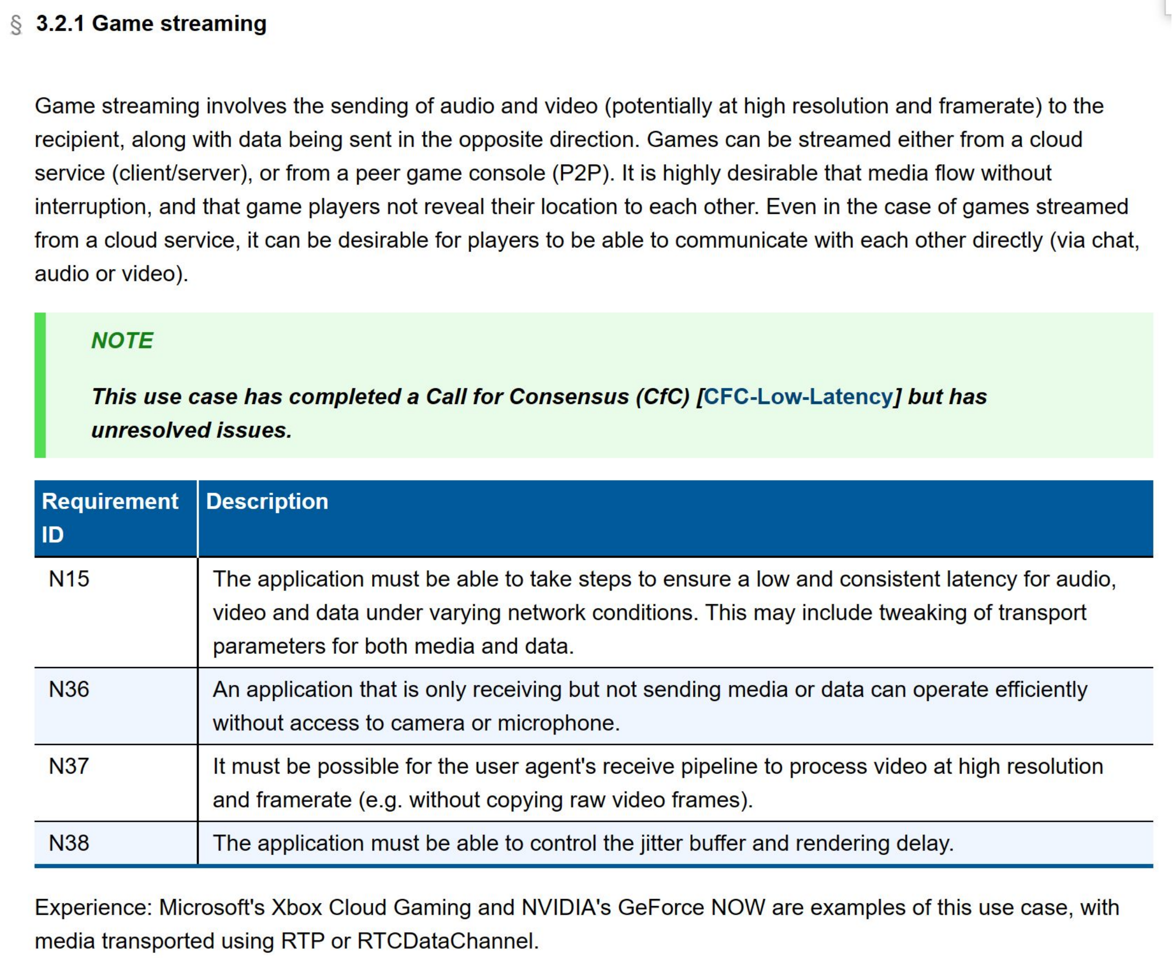
In Google experimentation, we have found that we need:

- SDP negotiation of non-standard codecs (PR #186)
- The ability to designate an outgoing frame as one of those non-standard codecs
- The ability to dispatch an incoming frame to appropriate processing
- The ability to control the decoding of the frame after processing
- The ability to control packetization and depacketization of those frames

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
    - Issue 103: Feedback related to WebRTC-NV Low Latency Streaming Use Case
  - 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 3.2.1: 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.

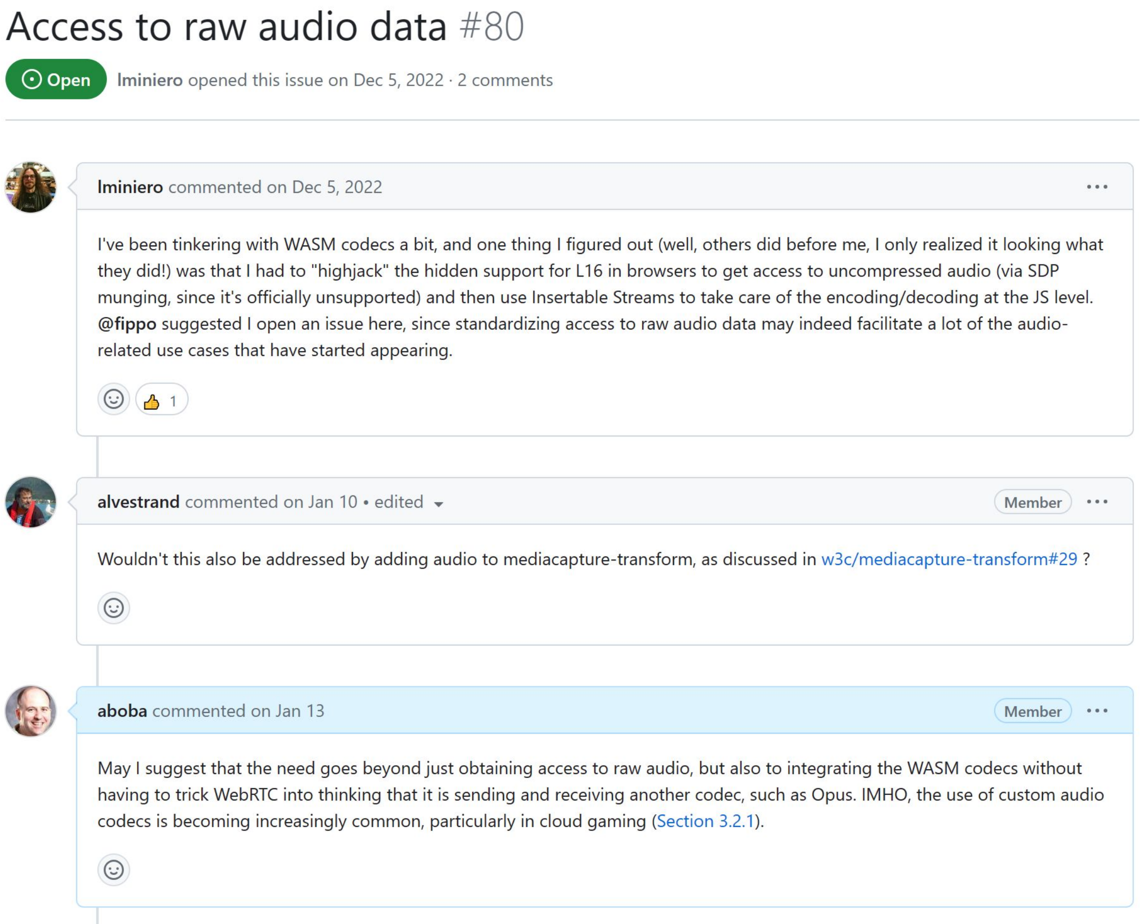
Requirement ID	Description
N15	The application must be able to take steps to ensure a low and consistent latency for audio, video and data under varying network conditions. This may include tweaking of transport parameters for both media and data.
N36	An application that is only receiving but not sending 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 80: Access to raw audio data
  - Issue 103: Section 3.2: Feedback relating to WebRTC-NV Low Latency Streaming Use Case
- PRs
  - PR 118: Clarify Game Streaming Requirements

Issue 80: Access to Raw Audio Data



Access to raw audio data #80

Open lminiero opened this issue on Dec 5, 2022 · 2 comments

lminiero commented on Dec 5, 2022

I've been tinkering with WASM codecs a bit, and one thing I figured out (well, others did before me, I only realized it looking what they did!) was that I had to "highjack" the hidden support for L16 in browsers to get access to uncompressed audio (via SDP munging, since it's officially unsupported) and then use Insertable Streams to take care of the encoding/decoding at the JS level. **@fippo** suggested I open an issue here, since standardizing access to raw audio data may indeed facilitate a lot of the audio-related use cases that have started appearing.

👍 1

alvestrand commented on Jan 10 • edited Member

Wouldn't this also be addressed by adding audio to mediacapture-transform, as discussed in w3c/mediacapture-transform#29 ?

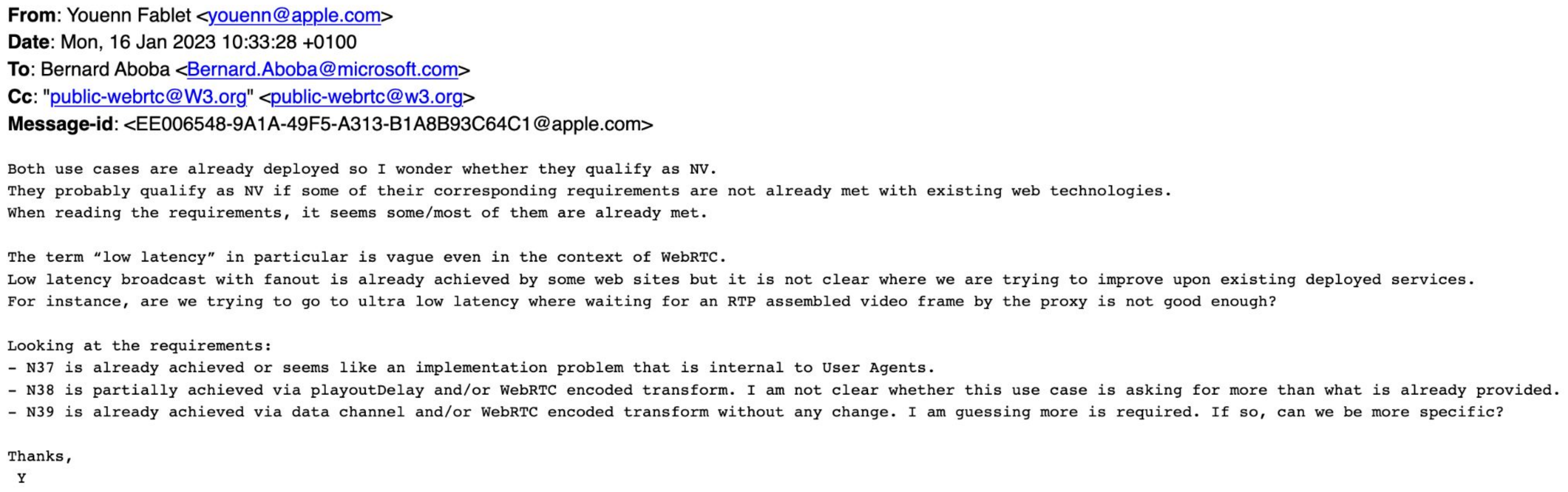
aboba commented on Jan 13 Member

May I suggest that the need goes beyond just obtaining access to raw audio, but also to integrating the WASM codecs without having to trick WebRTC into thinking that it is sending and receiving another codec, such as Opus. IMHO, the use of custom audio codecs is becoming increasingly common, particularly in cloud gaming (Section 3.2.1).

Issue 80: Access to Raw Audio Data

- Should we add a requirement for access to raw audio data?
- Access to raw audio data often required to implement custom audio codecs
  - Example: Spatial audio
  - Problem: this requires ‘tricking’ WebRTC into thinking it is sending and receiving another codec (e.g. Opus).
- Potential requirements
  - SDP negotiation of non-standard codecs (PR #186)
  - The ability to designate an outgoing frame as one of those non-standard codecs
  - The ability to dispatch an incoming frame to appropriate processing
  - The ability to control the decoding of the frame after processing
  - The ability to control packetization and depacketization of those frames

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
```

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

- With respect to Game Streaming (Section 3.2.1):
    - N37: Current text is not specific.
        - Potential next steps:
            - Leave it alone?
            - Add/replace with more specific performance requirements (**PR 118**)
    - N38: Requirement is partially satisfied by jitterBufferTarget.
        - Use case document does not currently link to APIs that relate to the requirements.
        - Recommendation: no action.

PR 118: Clarify Game Streaming requirements (Section 3.2.1)

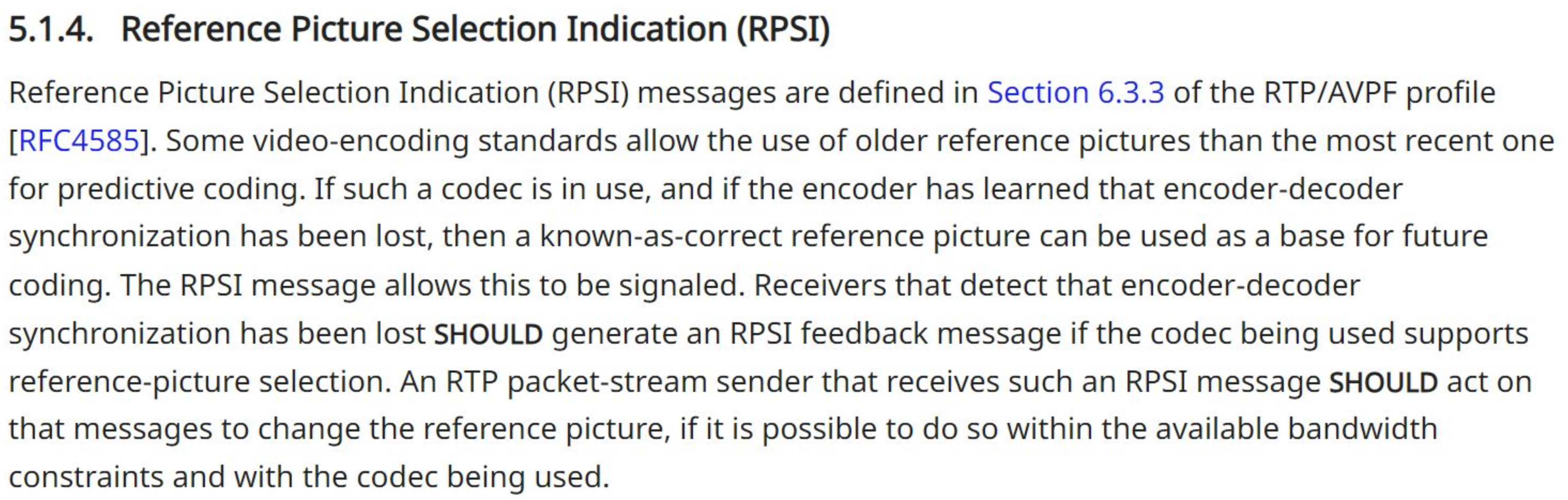
- Rationale: Cloud Game Characteristics
  - A highly interactive application that depends on **continuous visual feedback** to user inputs.
  - The cloud gaming latency KPI would track Click to Pixel latency - time elapsed between user input to when the game response is available at the user display (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 (Ex: Transport-wide RTCP Feedback Message) - Currently under field trial **"WebRTC-SendNackDelayMs".**	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 - Chromium Issue: 1327251	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

- At the July meeting, it was pointed out that this is an IETF issue.
- From "Media Transport and Use of RTP in WebRTC" RFC 8834, Section 5.1.4:



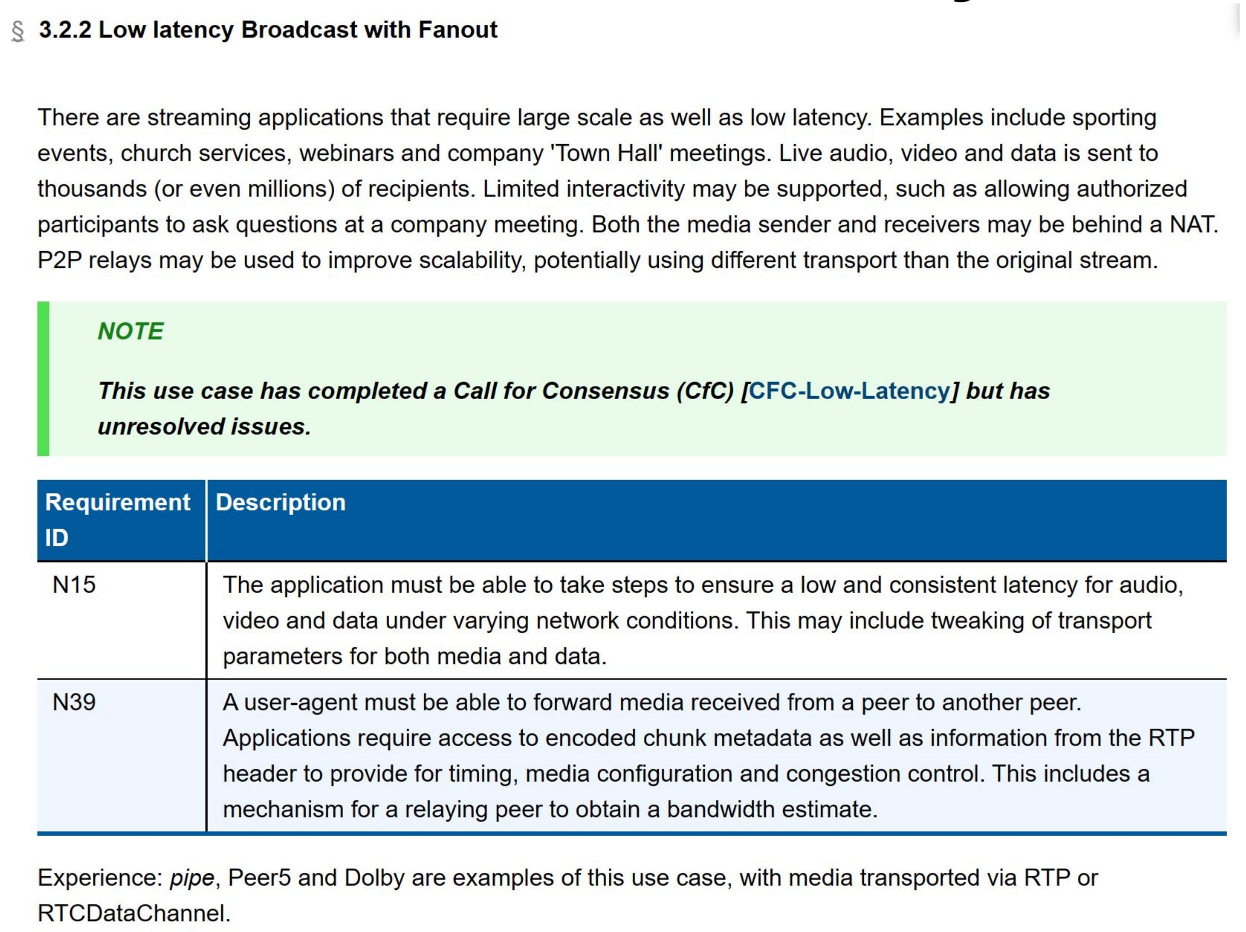
5.1.4. Reference Picture Selection Indication (RPSI)

Reference Picture Selection Indication (RPSI) messages are defined in Section 6.3.3 of the RTP/AVPF profile [RFC4585]. Some video-encoding standards allow the use of older reference pictures than the most recent one for predictive coding. If such a codec is in use, and if the encoder has learned that encoder-decoder synchronization has been lost, then a known-as-correct reference picture can be used as a base for future coding. The RPSI message allows this to be signaled. Receivers that detect that encoder-decoder synchronization has been lost **SHOULD** generate an RPSI feedback message if the codec being used supports reference-picture selection. An RTP packet-stream sender that receives such an RPSI message **SHOULD** act on that messages to change the reference picture, if it is possible to do so within the available bandwidth constraints and with the codec being used.

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

- draft-aboba-avtcore-hevc-webrtc in "Call for Adoption" in IETF AVTCORE WG
    - Issue 13 filed relating to RPSI support in HEVC
- Excerpts from GitHub:
    - "RPSI used to be implemented for VP8 and VP9 as feedback of successfully decoded picture id; It was later removed from the implementation as there seems no good usage of it for VP8/VP9."
    - "Currently, libwebrtc does not support RPSI at all. There was an effort many years ago to try out LTR, but then a custom RTCP message [Loss Notification] was implemented as RPSI was found to be insufficient in some ways."
        - See: https://bugs.chromium.org/p/webrtc/issues/detail?id=10336

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 RTCDataChannel.

Use Case: Low Latency Broadcast with Fanout

Webrtc-NV-Use-Cases 3.2.2 - completed CfC January 2023
Issues mentioned: #80 ~~#85~~ ~~#86~~ ~~#91 #94~~ ~~#95~~ #103

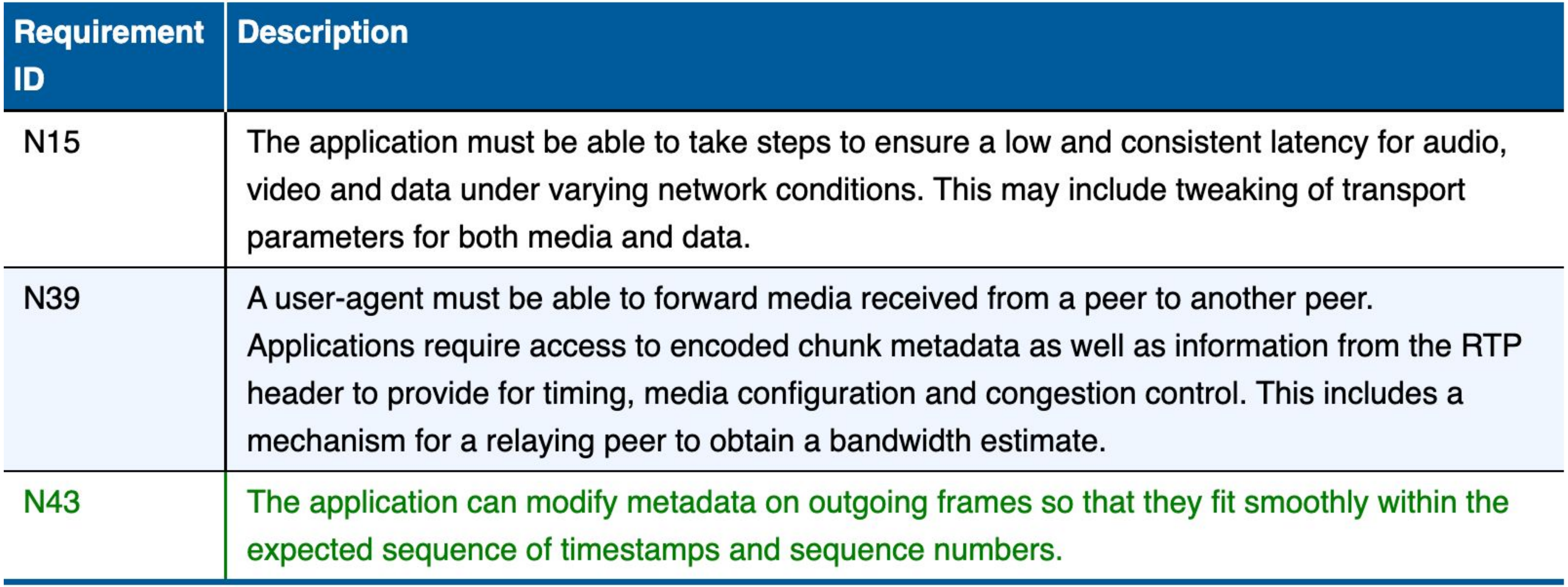
Google has experimented with this use case.

Conclusions: We need webrtc-encoded-stream modifications for:

- Ability to move frames between PCs (#200)
- Ability to structuredClone frames (#181)
- Ability to modify metadata (#162, use-cases #122)

For our sample use case, congestion management is optional.

Proposed requirement change



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.
N43	The application can modify metadata on outgoing frames so that they fit smoothly within the expected sequence of timestamps and sequence numbers.

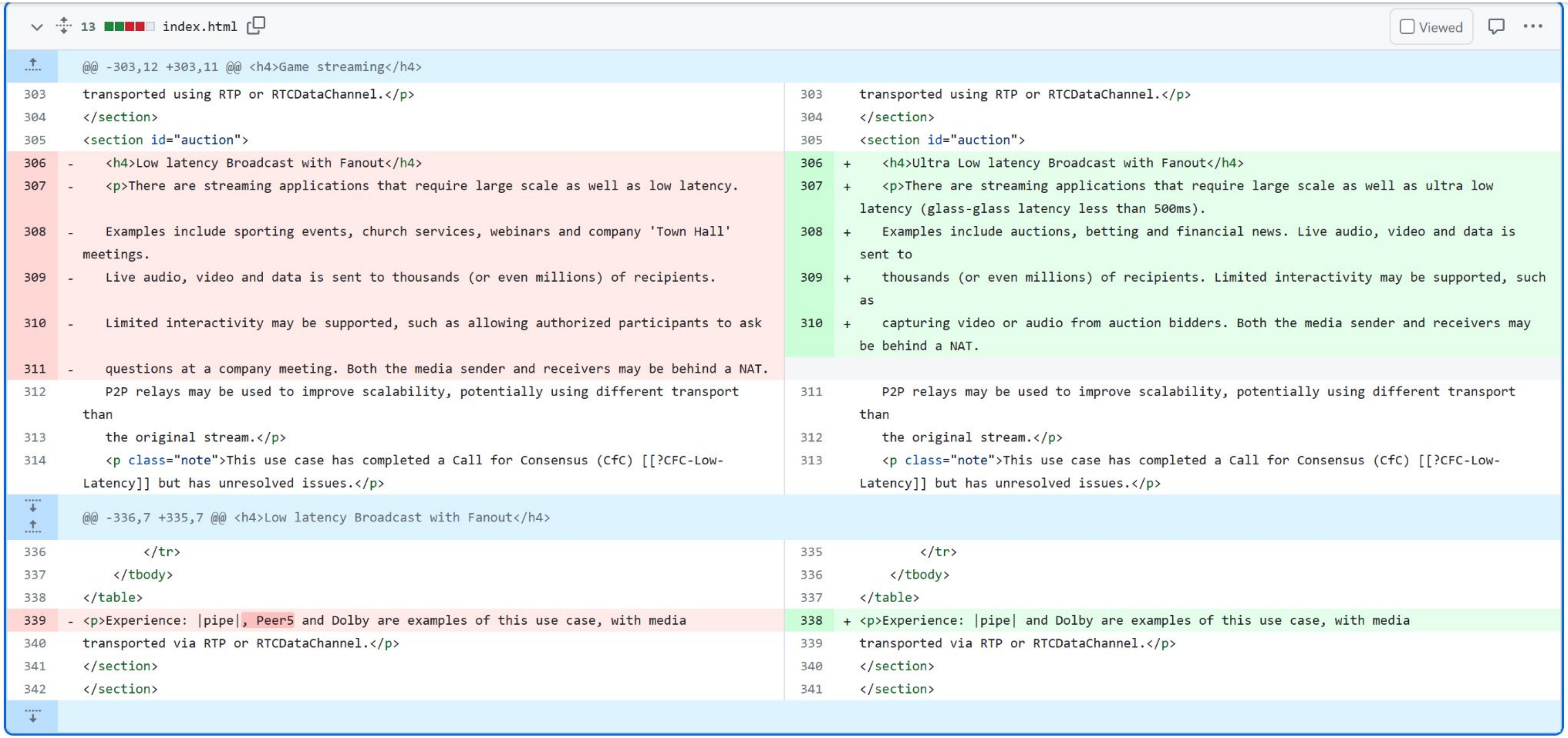
Section 3.2.2: Low latency Broadcast with Fanout

- Open Issues
    - Issue 80: Access to raw audio data (discussed previously)
    - Issue 103: Section 3.2: Feedback relating to WebRTC-NV Low Latency Streaming Use Case
- PRs
    - PR 123: Clarify Use Case

PR 123: Section 3.2.2: Clarify use case (Bernard)

- Goal: Focus on “ultra low latency” streaming
  - This use case was originally focused on auctions/betting, which require “ultra low latency” (glass-glass latency < 500 ms).
    - WebRTC is popular for these “ultra low latency” use cases (e.g. WHIP/WHEP).
    - For ULL streaming, data channel fanout adds too much latency, even with unreliable/unordered transport:
      - Overhead of CMAF containerization/decontainerization
        - DRM may not be needed.
      - Dependency on “Low latency MSE” (not standardized)
- Requirements for “low latency” using data channel fanout are covered elsewhere
  - “File Sharing” use case (Section 3.1): Requirement N13 (datachannel in workers)
  - “IoT” use case (Section 3.3): Requirement N16 (max retransmissions/timeout)

PR 123: Section 3.2.2: Clarify use case



13 index.html

Viewed

```
@@ -303,12 +303,11 @@ <h4>Game streaming</h4>
  transported using RTP or RTCDataChannel.</p>
  </section>
  <section id="auction">
-    <h4>Low latency Broadcast with Fanout</h4>
-    <p>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.
+    <h4>Ultra Low latency Broadcast with Fanout</h4>
+    <p>There are streaming applications that require large scale as well as ultra low latency (glass-glass latency less than 500ms).
+    Examples include auctions, betting and financial news. Live audio, video and data is sent to
+    thousands (or even millions) of recipients. Limited interactivity may be supported, such as
+    capturing video or audio from auction bidders. 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.</p>
     <p class="note">This use case has completed a Call for Consensus (CfC) [[?CFC-Low-Latency]] but has unresolved issues.</p>
@@ -336,7 +335,7 @@ <h4>Low latency Broadcast with Fanout</h4>
          </tr>
       </tbody>
  </table>
- <p>Experience: |pipe|, Peer5 and Dolby are examples of this use case, with media
+ <p>Experience: |pipe| and Dolby are examples of this use case, with media
  transported via RTP or RTCDataChannel.</p>
  </section>
  </section>
```

Discussion (End Time: 12:40)

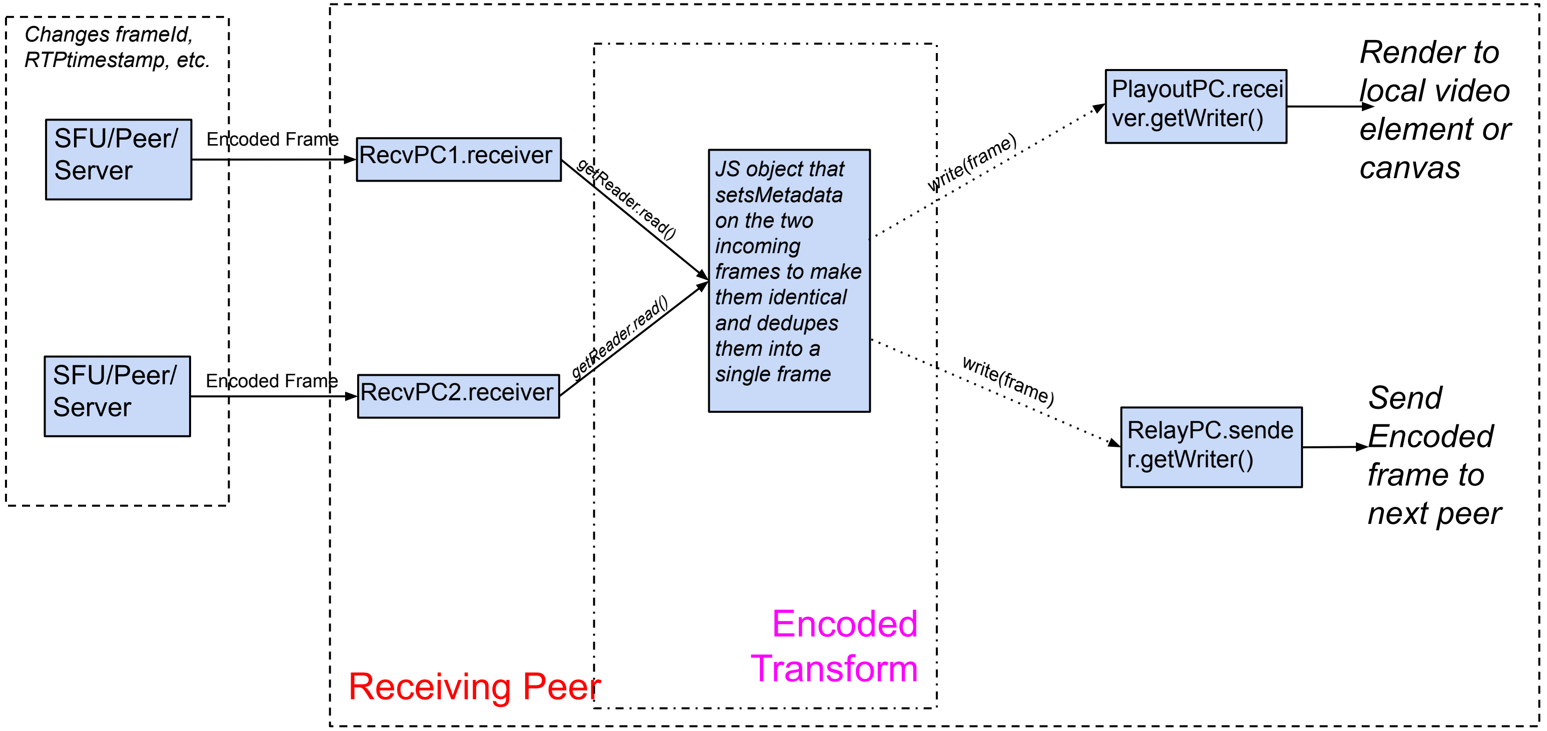
-

Modifications for Low Latency Fanout (Palak)

Start Time: 12:40

End Time: 13:00

Example setup for redundant transmission


Changes frameId, RTPtimestamp, etc.
SFU/Peer/ Server
Encoded Frame
SFU/Peer/ Server
Encoded Frame
RecvPC1.receiver
RecvPC2.receiver
getReader.read()
getReader.read()
JS object that setsMetadata on the two incoming frames to make them identical and dedupes them into a single frame
write(frame)
write(frame)
PlayoutPC.receiver.getWriter()
Render to local video element or canvas
RelayPC.sender.getWriter()
Send Encoded frame to next peer
Encoded Transform
Receiving Peer

Functionality Needed for Low Latency Fanout

We need webrtc-encoded-stream modifications for:

- Ability to move frames between PCs (#160, #200)
- Ability to call structuredClone() on frames (#181)
- Ability to modify metadata (#162, use-cases #122)

Ability to move frames between PCs (#160, #200)

- Allows nodes to forward received encoded frame to a relay peer by reading the encoded frame from the receivingPC and writing it to the relaySenderPC

- Remove restriction clause on streams being limited to only one PC. PR: #201

Ability to call structuredClone() on frames (#181)

- Allows a peer to use an encoded frame and its clone for relaying to other peers and for local rendering
- Only requires marking frames Serializable. PR: #182

Ability to modify metadata (#162, use-cases #122)

- Allow a node to update the metadata of frames from different incoming PCs such that frames with the same payload become interchangeable

- Limit changes to rtpTimestamp (audio and video), frameID and dependencies (video only)

- PR: #202

Discussion (End Time: 13:00)

-

Lunch Break

End Time: 14:00

WebRTC & Media Capture (Henrik, Jan-Ivar)

Start Time: 14:00

End Time: 14:40

For Discussion Today

- **MediaCapture-Extensions**
  - MediaStreamTrack Frame Stats API shape (Henrik)
- **MediaCapture-Output**
  - Issue 137: Undesirable prompt from selectAudioOutput({deviceId}) if valid device removed (Jan-Ivar)
- **WebRTC-PC**
  - Issue 2899: No way to observe DataChannel-only transport events in initial negotiation (Jan-Ivar)

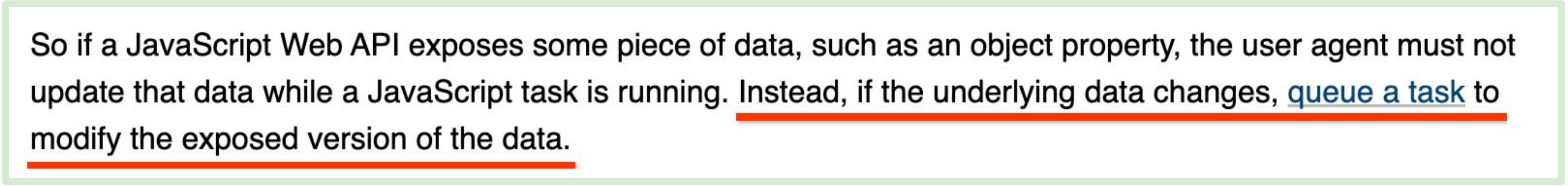
MediaStreamTrack Frame Stats API shape (Henrik)

Issue #105 and #98, PR #106

In a previous Virtual Interim we've discussed if track stats should be an async or sync API.

The frame counters live on the media thread, not JS.
The API shape affects how to surface the data (promise or instant get).

§5.2 Preserve run-to-completion semantics (*Web Platform Design Principles*) says data accessible to JS must not update while a JavaScript task is running:



So if a JavaScript Web API exposes some piece of data, such as an object property, the user agent must not update that data while a JavaScript task is running. Instead, if the underlying data changes, queue a task to modify the exposed version of the data.

To avoid excessive post tasking, we can *cross-thread get* the data and then cache it and clear it in the next task execution cycle. (Atomics, lockless ring buffer, mutex, etc.)

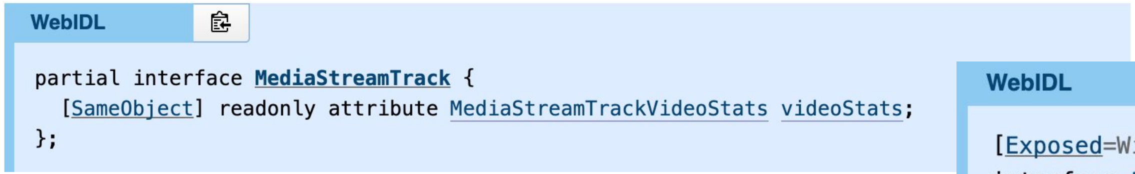
MediaStreamTrack Frame Stats API shape (Henrik)

Sync or async?

```
const {deliveredFrames, discardedFrames, …} = track.videoStats;
const {deliveredFrames, discardedFrames, …} = await track.getStats();
```

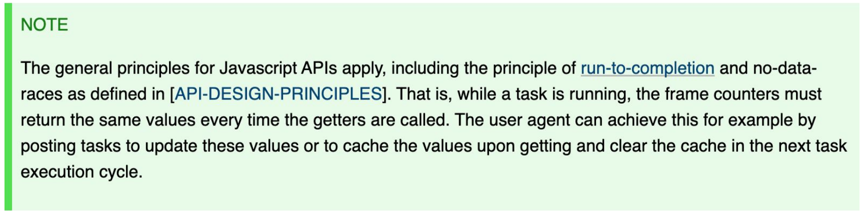
A desire to use this API outside of an async context has been expressed.

Extra implementation effort is small (caching). **PR #106:**



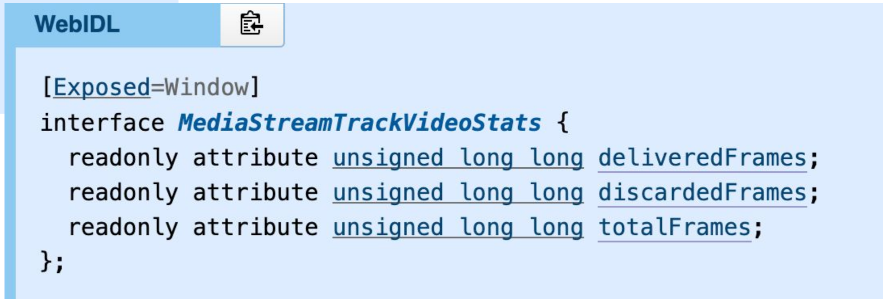
WebIDL

```
partial interface MediaStreamTrack {
  [SameObject] readonly attribute MediaStreamTrackVideoStats videoStats;
};
```



NOTE

The general principles for Javascript APIs apply, including the principle of run-to-completion and no-data-races as defined in [API-DESIGN-PRINCIPLES]. That is, while a task is running, the frame counters must return the same values every time the getters are called. The user agent can achieve this for example by posting tasks to update these values or to cache the values upon getting and clear the cache in the next task execution cycle.



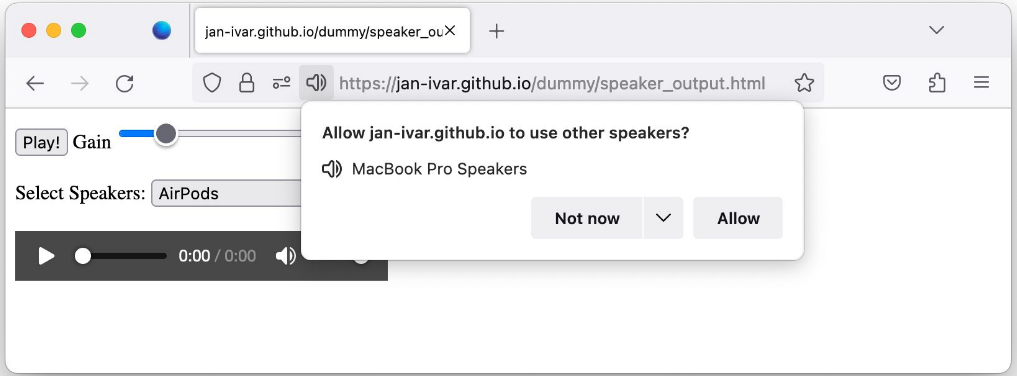
WebIDL

```
[Exposed=Window]
interface MediaStreamTrackVideoStats {
  readonly attribute unsigned long long deliveredFrames;
  readonly attribute unsigned long long discardedFrames;
  readonly attribute unsigned long long totalFrames;
};
```

Issue 137: Undesirable prompt from selectAudioOutput({deviceId}) if valid device removed

Firefox (116) implements `selectAudioOutput`!

Here's a demo with a PLAY button and a SELECT SPEAKERS button:


jan-ivar.github.io/dummy/speaker_ou
https://jan-ivar.github.io/dummy/speaker_output.html

Play! Gain

Select Speakers: AirPods

0:00 / 0:00

Allow jan-ivar.github.io to use other speakers?

MacBook Pro Speakers

Not now | Allow

This works:

1. Choose SELECT SPEAKERS, pick "Airpods" in prompt, then PLAY → 🎧🎵
2. Refresh page, and hit PLAY → 🎧🎵

But there's a problem:

3. Refresh page, put Airpods in their case, and hit PLAY → prompt

Users expect a prompt from SELECT SPEAKERS, *not* from PLAY button. Should 🔊🎵

Issue 137: Undesirable prompt from selectAudioOutput({deviceId}) if valid device removed (Jan-Ivar)

Behavior here should be an app decision.

Proposal:

- If the User Agent recognizes a removed deviceId (one it used to satisfy), then reject `selectAudioOutput({deviceId})` with `"NotFoundError"` as a one-time courtesy instead of prompting.
- Subsequent calls would continue to prompt

This provides apps an opportunity to remove this deviceId from localStorage, while still detering trackers.

Implementing this should be possible by tracking recently removed devices.

Issue 2899: No way to observe DataChannel-only transport events in initial negotiation

On a DataChannel-only peer connection, it’s impossible to access the IceTransport early enough to observe its events reliably on initial negotiation. E.g.:

```
  await pc.setLocalDescription();
// const {transport} = pc.getTransceivers()[0]; // works for audio & video
  const {transport} = pc.sctp; // TypeError: pc.sctp is null until "stable"!
  const {iceTransport} = transport;

  iceTransport.onstatechange = () => {...};            // missed events
  iceTransport.ongatheringstatechange = () => {...}; // missed events
  iceTransport.onselectedcandidatepairchange = () => {..}; // can miss prflx?
```

This seems inconsistent. Should we surface the sctp transport in sLD (and roll it back if need be) like the other transports?

Issue 2899: No way to observe DataChannel-only transport events in initial negotiation

maxMessageSize would need to become nullable like maxChannels is today. E.g.:

```
  // assuming state is unknown
- if (pc.sctp) console.log(pc.sctp.maxMessageSize);
+ if (pc.sctp?.maxMessageSize != null) console.log(pc.sctp.maxMessageSize);
  if (pc.sctp?.maxChannels != null) console.log(pc.sctp.maxChannels);
```

Proposal A: do this (surface sctp transport in sLD like other transports)

Proposal B: Do nothing. For DataChannel-only tell people to use:

```
  pc.oniceconnectionstatechange = () => {...};
  pc.onicegatheringstatechange = () => {...};
```

Discussion (End Time: 14:40)

-

Ice Controller API (Sameer & Peter)

Start Time: 14:40

End Time: 15:10

IceController road map

- Prevent removal of candidate pairs
- Remove candidate pairs
- Control selection of candidate pair
- (?) Observe candidate pair states
- Observe result/RTT of outgoing checks
- Control frequency of outgoing checks of particular candidate pairs
- Prevent outgoing checks of particular candidate pairs
- Control order and timing of outgoing checks
- Observe presence of of incoming checks or media for particular candidate pairs
- Gather local candidates for new network interfaces
- Re-gather local candidates of previously failed network interfaces
- Prevent removal of local candidates
- Remove local candidates
- Construct IceTransport without PeerConnection
- Support forking

Done-ish!

Follow-up!

New!

Next time!

Issue 171 - ICE candidate pair selection

- Goal: Application can choose which candidate pair to use to send data
- ICE RFC 8445 (and subsequent updates - RFC 8838 Trickle ICE and RFC 8863 ICE PAC) describes candidate pair nomination and selection
  - ICE agent performs connectivity checks and identifies valid candidate pairs
  - At some (unspecified) point, controlling agent picks one candidate pair to nominate
  - Prior to nomination, data can be sent on any valid pair
  - If nomination succeeds, data can only be sent on the nominated pair
  - Post nomination, the ICE agent must not nominate another pair
  - Post nomination, an ICE restart is required to change the selected pairs
- Preferable to avoid ICE restart to change the selected pair
  - Expensive to gather candidate pairs, perform connectivity checks and nominate again
- Sending data on any valid pair prior to nomination is an intentional decision in RFC 8445

PR 174 - ICE candidate pair selection - Proposal

- Application can defer nomination of a candidate pair picked by the controlling ICE agent
  - cancelable `icecandidatepairnominate` event
- Application can select any valid candidate pair to send data
  - `setSelectedCandidatePair` sets the selected pair without nomination
  - Both controlling and controlled side can select a candidate pair to send data
  - Only the responsibility to nominate differentiates the two sides
  - Application now has the responsibility & flexibility to synchronize send/receive
- ICE agent may nominate the candidate pair selected by the Application
  - Application can choose to defer this nomination again

```
partial interface RTCIceTransport {
  attribute EventHandler /* RTCIceCandidatePairEvent */ onicecandidatepairnominate;
  undefined setSelectedCandidatePair(RTCIceCandidatePair candidatePair);
};
```

PR 175 - ICE candidate pair removal

- Goal: Application can remove unused candidate pairs
- Proposal
  - `removeCandidatePairs` immediately removes the provided pairs
  - non-cancelable `icecandidatepairremove` event(s) fired after removal

```
partial interface RTCIceTransport {
   attribute EventHandler /* RTCIceCandidatePairEvent */ onicecandidatepairremove;
   undefined removeCandidatePairs(sequence<RTCIceCandidatePair> candidatePairs);
};
```

- *…and previously (PR 168)…*

```
partial interface RTCIceTransport {
   attribute EventHandler /* RTCIceCandidatePairEvent */ onicecandidatepairadd;
};
```

Sample application - only use a UDP port 3478 candidate pair

```
const pc = new RTCPeerConnection({iceServers: [/* ice servers */]});
const transceiver = pc.addTransceiver("video");
await pc.setLocalDescription();

const udp3478Pair = new class {
  constructor() {
    this.promise = new Promise((resolve, reject) => this.resolve = resolve);
  }
  matches(e) {
    return e.local.protocol === 'udp' && e.remote.port === 3478;
  }
}();
const unusedPairs = [];

transceiver.sender.transport.iceTransport.onicecandidatepairadd = (e) => {
  const addedPair = {
    local: e.local,
    remote: e.remote
  };
  if (udp3478Pair.matches(e)) {
    udp3478Pair.resolve(addedPair);
  }
  else {
    unusedPairs.push(addedPair);
  }
};
```

```
transceiver.sender.transport.iceTransport.onicecandidatepairremove =
  (e) => {
    if (e.cancelable && udp3478Pair.matches(e)) {
      e.preventDefault();
    }
    // else candidate pair announced by `e` is removed
  };

transceiver.sender.transport.iceTransport.onicecandidatepairnominate =
  (e) => {
    if (!udp3478Pair.matches(e)) {
      e.preventDefault();
    }
    // else UDP 3478 pair announced by `e` is nominated
  };

udp3478Pair.promise.then(pair => {
  transceiver.sender.transport.iceTransport
      .setSelectedCandidatePair(pair);
  transceiver.sender.transport.iceTransport
      .removeCandidatePairs(unusedPairs);
});
```

Observe result/RTT of outgoing checks

```
const pc = …;
const ice = pc.getTransceivers()[0].sender.transport.iceTransport;
ice.onchecksend = async(event) => {
  const check = await event.check;
  const response = await check.response;
  if (response) {
    const rtt = response.receivedTime - check.sentTime;
    // … do something with rtt
    if (response.error) {
      // … do something with error …
    }
  } else {
    // … do something with timeout …
  }
```

Prevent outgoing checks

```
const pc = …;
const ice = pc.getTransceivers()[0].sender.transport.iceTransport;
ice.onchecksend = (event) => {
  if (iDontLikeThisCheck(event.check)) {
    event.preventDefault();
  }
}
```

Control when ICE checks are sent

```
const pc = …;
const ice = pc.getTransceivers()[0].sender.transport.iceTransport;
while (true) {
  const candidatePair = await nextTimeToSendCheck();
  const checkSent = ice.sendCheck(candidatePair);
  (async () => {
    const check = await checkSent;
    const response = await check.response;
    if (response) {
      const rtt = response.receivedTime - check.sentTime;
      // … do something with rtt
    }
    … do more stuff …
  })();
```

The WebIDL

```
partial interface RTCIceTransport {
  Promise<RTCIceCheck> sendCheck()
  attribute EventHandler onchecksend;
}
interface RTCIceCheckSend : Event {  // Cancellable
  readonly attribute Promise<RTCIceCheck> check;  // Resolves when actually sent
}
interface RTCIceCheck {
  readonly attribute ArrayBuffer transactionId;
  readonly attribute DOMHighResTimeStamp sentTime;
  readonly attribute Promise<RTCIceCheckResponse?> response; // No response == timeout
}
interface RTCIceCheckResponse {
  readonly attribute DOMHighResTimeStamp receivedTime;
  readonly attribute RTCIceCheckResponseError? error; // No error == success
}
```

Discussion (End Time: 15:10)

-

RTPTransport (Peter & Stefan)

Start Time: 15:10

End Time: 15:40

Why RtpTransport?

Provide flexibility and control to web apps that want it:

- Custom payloads (perhaps ML-based audio codec)
- Custom packetization
- Custom FEC (perhaps ML-based)
- Custom RTX
- Custom Jitter Buffer (perhaps ML-based)
- Custom BWE (perhaps ML-based)
- Custom bitrate allocation
- Custom metadata (header extensions)
- Custom RTCP messages
- Forwarding

Why "progressive version"?

Make the transition simple

- Works with PeerConnection
- Works with Encoded Streams
- Works with WebCodecs
- Pick which parts you want to replace or keep

PeerConnection.createRtpTransport()

- Like .createDataChannel, sets up ICE, DTLS, and SRTP
- Returns RtpTransport object, with which:
    - You can receive all RTP and RTCP packets for the entire bundle group (IceTransport/DtlsTransport)
    - You can send any RTP or RTCP packet on with the following restrictions:
        i. SRTP (SSRC, seqnum, ROC) can't be reused (would break SRTP)
        ii. SRTCP seqnum can't be set (would break SRTCP)
        iii. Header extensions used for congestion control can't be set (would break CC)
        iv. RTCP feedback used for congestion control can't be sent (would break CC)
        v. If you send more than the BWE, packets may be queued/dropped
    - You can get a BWE

Things you can do

- Encode and packetize with custom WASM/ML codec and send
- Get frames from Encoded Streams, packetize yourself, and send
- Get frames from Encoded Streams, apply custom FEC, and send
- Observe incoming NACK and resend with custom RTX behavior
- Receive packets and put in custom jitter buffer implementation
  - Which can use WebCodecs for decode
- Receive packets, depacketize yourself, and inject into Encoded Streams
  - **Requires a constructor for EncodedAudioFrame/EncodeVideoFrame**
- Observe incoming feedback and do custom BWE
  - as long as it's *lower* than the built-in CC
- Get frames from Encoded Streams, packetize yourself, attach custom metadata, and send
- Get BWE from RtpTransport, do bitrate allocation yourself, and set bitrates of RtpSenders
- Forward RTP/RTCP packets from one PeerConnection to another, with full control over the entire packet (modulo SRTP/CC exceptions)

Example: Send with custom packetization

```
const pc = new RTCPeerConnection({encodedInsertableStreams: true});
const rtpTransport = pc.createRtpTransport();
pc.getSenders().forEach((sender) => {
  pc.createEncodedStreams().readable.
      pipeThrough(createPacketizingTransformer()).pipeTo(rtpTransport.writable);
});

function createPacketizingTransformer() {
  return new TransformStream({
    async transform(encodedFrame, controller) {
      let rtpPackets = myPacketizer.packetize(frame);
      rtpPackets.forEach(controller.enqueue);
    }
  });
}
```

Example: Receive with custom depacketization

```
const pc = new RTCPeerConnection({encodedInsertableStreams: true});
const rtpTransport = pc.createRtpTransport();
receiver.ontrack = event => {
  const esWriter = event.receiver.createEncodedStreams().writable.getWriter();
  rtpTransport.onrtppacket = (rtpPacket) => {
    let {vBuffer, esWriter} = receivers[getUniqueStreamIdentifier(rtpPacket)];
    vBuffer.insertPacket(rtpPacket);
    // Requires a constructor for EncodedVideoFrame/EncodedAudioFrame
    while (vBuffer.nextFrameReady()) esWriter.write(vBuffer.getFrame());
  }
}
```

Example: Receive with custom jitter buffer and built-in depacketization

```
const receiver = new RTCPeerConnection({encodedInsertableStreams: true});
receiver.ontrack = e => {
  if (e.track.kind == "video") {
    const es = event.receiver.createEncodedStreams({jitterBuffer: false});
    receiveVideo(es.readable.getReader(), es.writable.getWriter());
  }
  else {
    const es = event.receiver.createEncodedStreams();
    receiveAudio(es.readable.getReader(), es.writable.getWriter());
  }
}
function receiveVideo(reader, writer) {
  while (true) {
    const {value: frame, done} = await reader.read();
    if (done) return;
    vBuffer.insertFrame(frame);
    while (vBuffer.nextFrameReady()) writer.write(vBuffer.getFrame());
  }
}
```

Example: Custom bitrate allocation

```
const pc = new RTCPeerConnection();
const rtpTransport = pc.createRtpTransport();
rtpTransport.ontargetsendratechanged = () => {
  const rtpSender = pc.getTransceivers()[0];
  const parameters = rtpSender.getParameters();
  parameters.encodings[0].maxBitrate = rtpTransport.targetSendRate;
  rtpSender.setParameters(parameters);
};
```

RtpPacket

When receiving, the following are "parsed" for you:

- SSRC
- seqnum (+ROC)
- timestamp
- marker bit
- payload type
- CSRCs
- header extensions (ID, value)
- payload

When sending, you provide them and it's serialized for you.

RtcpPacket

When receiving, the following are "parsed" for you (as an array):

- SSRC
- Payload Type
- Payload Subtype (AKA reception report count)
- Payload

When sending, you provide them (as an array) and they are serialized for you

SDP

```
v=0
o=- 0 0 IN IP4 0.0.0.0
s=
t=0 0
a=ice-ufrag:K86o
a=ice-pwd:j1T4YePMF3i2hYFrV7lmsD
a=fingerprint:sha-256 00:CA:79:9D:AC:D9:0A:B7:36:C9:92:4D:D5:25:FD:47:01:F8:AA:87:A0:0D:B1:DF:B5:20:E8:CD:6B:C4:26:A3
a=setup:passive
m=application 9 UDP/TLS/RTP/SAVPF *
a=extmap:1 http://www.ietf.org/id/draft-holmer-rmcat-transport-wide-cc-extensions-01
```

- "m=application" means "can be audio, video, or anything"
- "*" means "could be any payload type, with any SSRC, and any header extension
  - a=sendrecv and a=rtcp-mux are assumed/required
- The only header extensions specified are for congestion control
- Can be in a BUNDLE group or not
- This could be the entirety of the SDP (for a forwarder, for example, or when using WebCodecs), or it could be in addition to m=audio and m=video lines, just like with an SCTP m=application line.

SDP combined with m=audio and m=video

```
v=0
o=- 0 0 IN IP4 0.0.0.0
s=
t=0 0
a=group:BUNDLE 0 1 2
a=ice-ufrag:K86o
a=ice-pwd:j1T4YePMF3i2hYFrV7lmsD
a=fingerprint:sha-256 00:CA:79:9D:AC:D9:0A:B7:36:C9:92:4D:D5:25:FD:47:01:F8:AA:87:A0:0D:B1:DF:B5:20:E8:CD:6B:C4:26:A3
a=setup:passive
m=application 9 UDP/TLS/RTP/SAVPF *
a=mid:0
a=extmap:1 http://www.ietf.org/id/draft-holmer-rmcat-transport-wide-cc-extensions-01
m=audio 9 UDP/TLS/RTP/SAVPF 99
a=mid:1
a=rtcp-mux
a=sendrecv
a=rtpmap:99 opus/48000/2
m=video 9 UDP/TLS/RTP/SAVPF 100
a=mid:2
a=rtcp-mux
a=sendrecv
a=rtpmap:100 vp8/90000
a=rtcp-fb:100 nack
```

Discussion (End Time: 15:40)

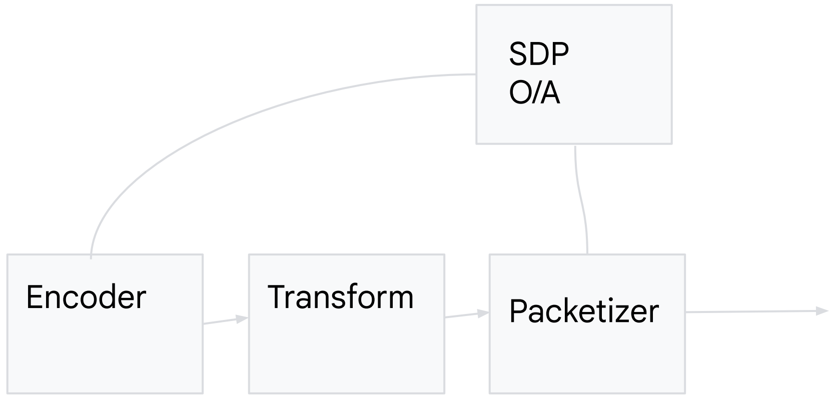
-

SDP Negotiation for Encoded Transform (Harald Alvestrand)

Start Time: 15:40

End Time: 16:00

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 - the aftermath

Before sending data:

- Choose explicitly the encoder to use for encoding, using SetParameters
- Tell the transform what PT to assign to post-transform frames

Before receiving data:

- 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.

Proposed Changes since July

- Setting packetizer: Go from “negotiate this codec with a packetizer attribute” to “Set packetizer / depacketizer for this PT”
  - Lines up better with setting encoding codec via SetParameters
- Identifying frames: Switch from PT to MIME type?
  - Would allow MIME type to PT mapping to be done purely in encoder
  - Adding MIME type to frame has been suggested for other reasons

Details: Payload Types in Transforms

A PayloadType (PT) is a 7-bit integer that indicates the payload embedded in an RTP frame.

It is used to guide the decoder in selecting the correct depacketization procedure, and, following that, the correct decoder for the payload; it follows that the packetizer must know what PT to apply.

The current proposal lets the transform set the frame PT and choose the PT/packetizer association.

The association between MIME types and PT is set up by SDP offer/answer.

So far, metadata on encoded frames has contained PT, and not MIME type.

QUESTION: Should frames be tagged by MIME type and not by PT?

Discussion (End Time: 16:00)

-

Topic TBD

Start Time: 16:00

End Time: 16:20

Slide Title Goes Here

- Content goes here

Discussion (End Time: 16:20)

-

Wrapup and Next Steps (Chairs)

Start Time: 16:20

End Time: 16:30

Next Steps

- Content goes here

Discussion (End Time: 16:30)

-

Thank you

Special thanks to:

WG Participants, Editors & Chairs