W3C WebRTC WG Meeting

December 5, 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 first December interim meeting of the W3C WebRTC WG, at which we will cover:
    - MediaCapture-ScreenShare, WebRTC Extended Use Cases, RtpTransport, Mediacapture-Extensions, WebRTC-Extensions, Encrypted Transform, WebRTC-SVC
- Future meetings:
    - December 12
    - January 16
    - February 20
    - March 19

About this Virtual Meeting

- Meeting info:
  - https://www.w3.org/2011/04/webrtc/wiki/December_05_2023
- 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
  - https://github.com/w3c/webrtc-rtptransport
- 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 Grab Bag (Sameer, Jan-Ivar, Florent)
- 08:30 - 08:50 AM WebRTC Extended Use Cases (Bernard & Sun)
- 08:50 - 09:50 AM RtpTransport (Peter, Bernard, Jan-Ivar)
- 09:50 - 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.

Grab Bag

Start Time: 08:10 AM

End Time: 08:30 AM

For Discussion Today

- WebRTC-Encoded Transform (Fippo)
  - PR 212: Describe data attribute
- WebRTC-Extensions
  - PR 175: Add RTCIceTransport method to remove pairs
  - Issue 2170: Data channel default binaryType value is 'blob'
- MediaCapture-Extensions
  - PR 134: Introduce avg/min/max audio latency and resetLatency() (Henrik)

PR 212: Describe data

- The specification does not describe the format of the encoded frame data
  - Codec-specific, with some surprises
  - addition of mimeType allows describing this
  - PR adds table with informative references for a few mimeTypes
- SVC behavior needs to be described too
  - Called once per spatial layer with same RTP timestamp
  - "Two-time pad" if E2EE KDF depends only on RTP timestamp and not frameId
  - Q: does simulcast behavior need special mention too?
- Underlying packetizer makes assumptions about format
  - E.g. H264 packetization needs to retain annex-b NALUs with start codes
- Caveat: output and input may not be the same
  - packetization may drop some parts like AV1 temporal OBUs
- Please review!

PR 175: Add RTCIceTransport method to remove pairs

Promise<undefined> removeCandidatePair(RTCIceCandidatePair *pair*);

- What is the use case?
  - App cancels nomination & selects a different candidate pair
  - Now app wants to stop pinging other pairs and release resources
- What does "remove" mean?
  - *"Tell ICE agent that app does not want to use this pair in this session"*
  - Remove pair from (all) Checklists ⇒ no more pings
  - Update Checklist states ⇒ *Failed* if all pairs removed or *Failed*
  - Free unpaired candidates ⇒ release resources
  - Removed pairs cannot be added back *(unless regathering supported)*
- Is an Array argument needed?
  - Useful if app selects a pairs and wants to stop pinging all others
  - Not essential, but can reduce thread hops when bulk removing

Issue 2170: Data channel default binaryType value is 'blob'

- DataChannel.binaryType has 2 possible values “blob” and “arraybuffer”
- “arraybuffer” is implemented in all implementations.
- “blob” implementation is missing in Chromium based browsers.
- “blob” is the current standard default value.
    - Chromium and WebKit use “arraybuffer” by default.
    - Safari correctly use “blob” by default.
- Many applications rely on Chromium or WebKit’s default “arraybuffer” binaryType explicitly, breaking compatibility with Firefox. Changing the default value to “blob” would break those.
- Compatibility with WebSocket may not be considered as important now.

- Proposal: For the sake of interoperability, we propose that the default value is changed to “arraybuffer”.

PR 134: Introduce avg/min/max audio latency and resetLatency() (Henrik)

As of recently, `MediaStreamTrack`'s Audio Stats API define latest input latency:

> The ***latest input latency*** is the latest available input latency as estimated between the track's input device and delivery to any of its sinks.

✅ Which is exposed as `track.stats.latency` as an instantaneous value:

```
partial interface MediaStreamTrackAudioStats {
   readonly attribute DOMHighResTimeStamp latency;
}
```

❌ But…

- Apps care about average input latency over **app-defined time intervals**.
  - ➢ `sumOfLatencyMeasurements/numLatencyMeasurements` is not the most ergonomic.
- Average latency hides **peaks and troughs** and could be misleading on its own.
  - ➢ We need "min" and "max" too.

PR 134: Introduce avg/min/max audio latency and resetLatency() (Henrik)

PR 134: Expose average, minimum and maximum since method call:

```
partial interface MediaStreamTrackAudioStats {
   readonly attribute DOMHighResTimeStamp latency;

   // The avg/min/max latency since last call to resetLatency()
   readonly attribute DOMHighResTimeStamp averageLatency;
   readonly attribute DOMHighResTimeStamp minimumLatency;
   readonly attribute DOMHighResTimeStamp maximumLatency;

   undefined resetLatency();
}
```

Proposal: Keep it simple. Merge PR.

Discussion (End Time: 08:30)

-

WebRTC Extended Use Cases

Start Time: 08:30 AM

End Time: 08:50 AM

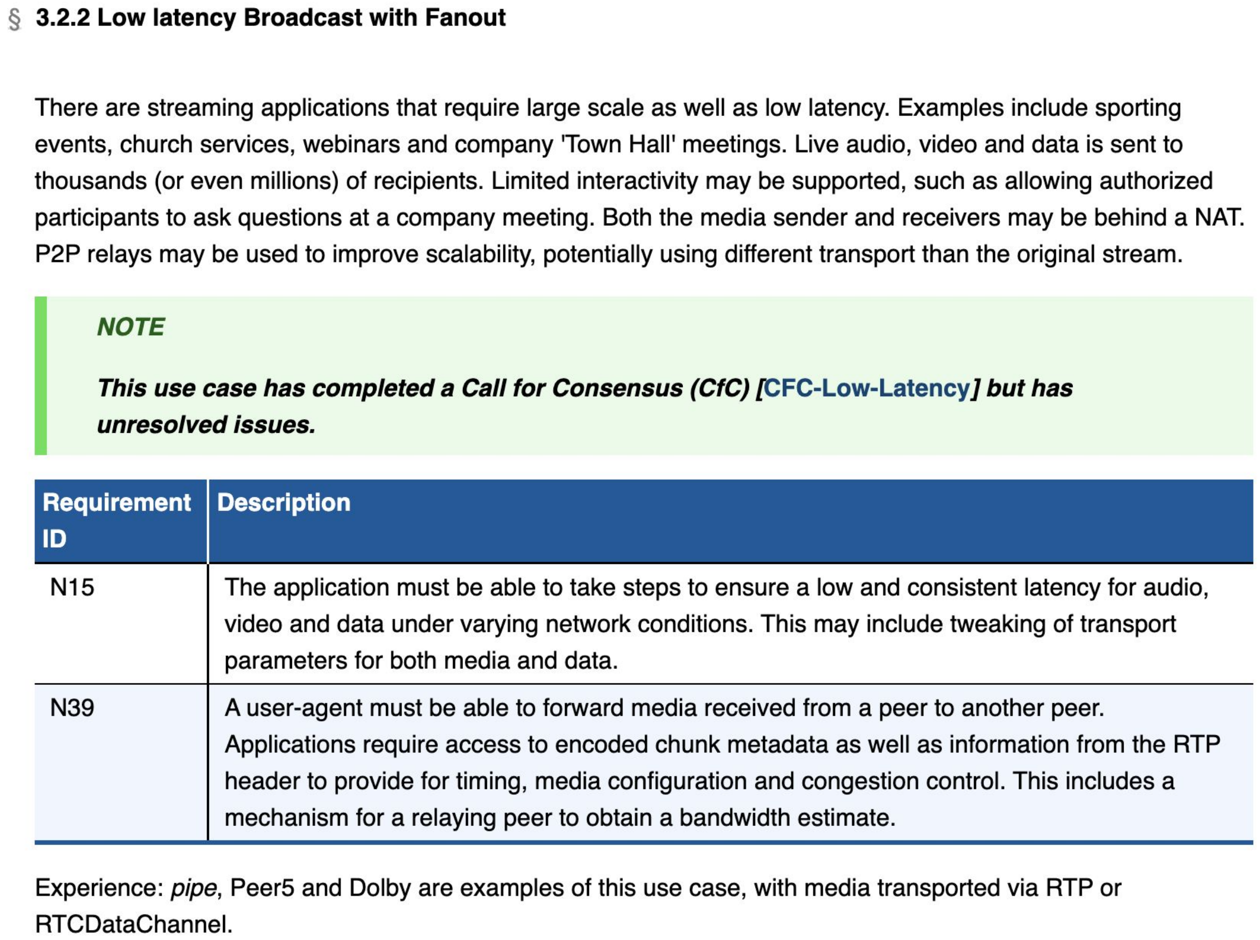
For Discussion Today

- 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 103: Feedback related to WebRTC-NV Low Latency Streaming Use Case
  - Moved issues
    - Issue 80: Access to raw audio data (TPAC 2023: move to audio codec use case)
  - Closed issues/PRs
    - 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
    - PR 124: Requirement N38 is satisfied by jitterBufferTarget (partial fix for Issue 103)
-

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.

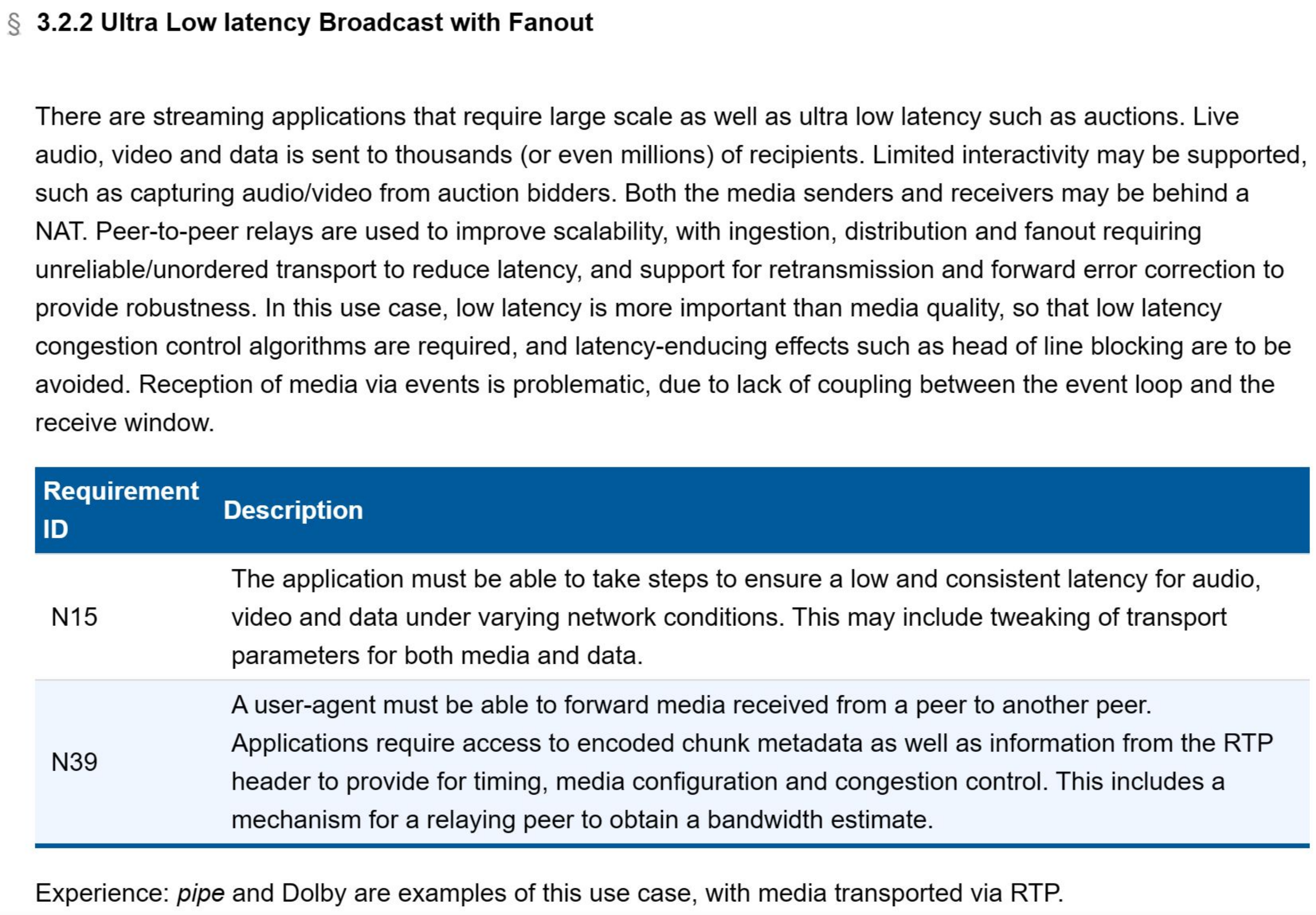
Requirement ID	Description
N15	The application must be able to take steps to ensure a low and consistent latency for audio, video and data under varying network conditions. This may include tweaking of transport parameters for both media and data.
N39	A user-agent must be able to forward media received 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.

PR 123: Use Case Goals

- Proposed refocus on auctions (originally suggested by Tim Panton).
    - Online auctions require ultra low latency (more important than quality)
    - Need for participant feedback
    - DRM typically not required
    - IETF WISH WG: ULL ingestion and distribution via WebRTC (WHIP/WHEP)
    - Low latency use cases like Church services, Webinars removed
        - Use streaming technology (e.g. LL-HLS), not WebRTC
        - Fanout requirements already covered by RTCDataChannel requirements (e.g. worker support) in Section 3.1: File Sharing.
- P2P Fanout for Auctions: Data Channel transport not a good fit
    - Issues with backpressure, due to decoupling of event loop and receive window
    - SCTP transport implements NewReno, but low latency congestion control required
    - Need to implement RTCP-style feedback (e.g. PLI) and FEC/RED in the application
    - Transport mode issues
        - Reliable/ordered transport: issues with latency, HoL blocking, buffer size
        - Unreliable/unordered transport: app needs to reimplement NACK/RTX

PR 123: Section 3.2.2: Clarify Use Cases



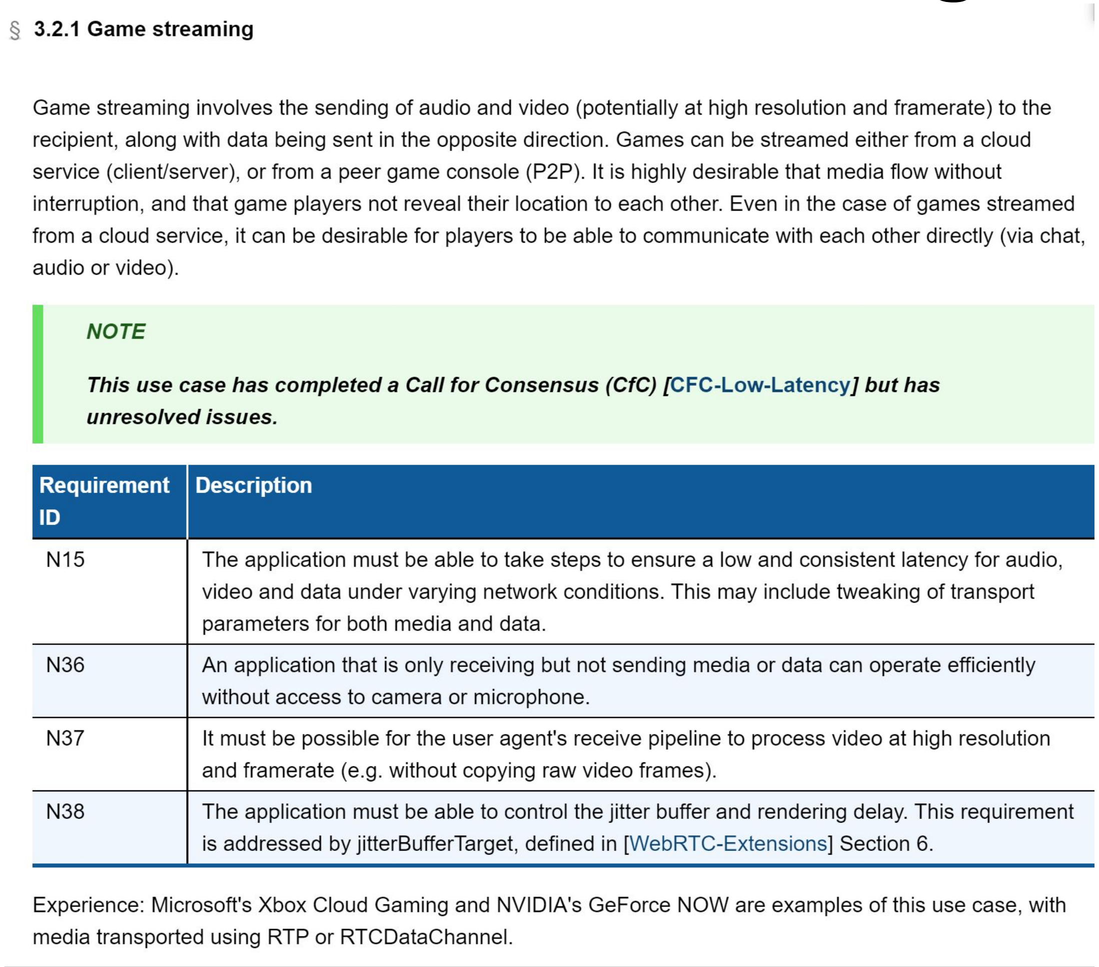
§ **3.2.2 Ultra Low latency Broadcast with Fanout**

There are streaming applications that require large scale as well as ultra low latency such as auctions. Live audio, video and data is sent to thousands (or even millions) of recipients. Limited interactivity may be supported, such as capturing audio/video from auction bidders. Both the media senders and receivers may be behind a NAT. Peer-to-peer relays are used to improve scalability, with ingestion, distribution and fanout requiring unreliable/unordered transport to reduce latency, and support for retransmission and forward error correction to provide robustness. In this use case, low latency is more important than media quality, so that low latency congestion control algorithms are required, and latency-enducing effects such as head of line blocking are to be avoided. Reception of media via events is problematic, due to lack of coupling between the event loop and the receive window.

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* and Dolby are examples of this use case, with media transported via RTP.

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.

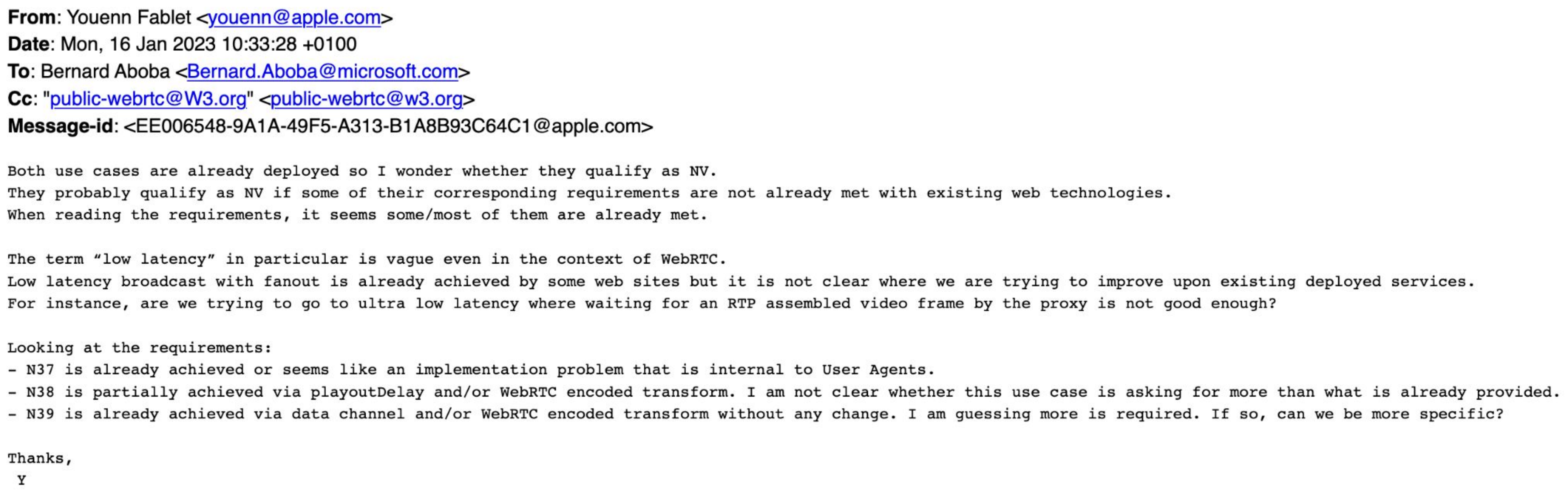
Requirement ID	Description
N15	The application must be able to take steps to ensure a low and consistent latency for audio, video and data under varying network conditions. This may include tweaking of transport parameters for both media and data.
N36	An application that is only receiving but not sending 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. This requirement is addressed by jitterBufferTarget, defined in [WebRTC-Extensions] Section 6.

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 125: Clarify Requirement N37
  - 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>

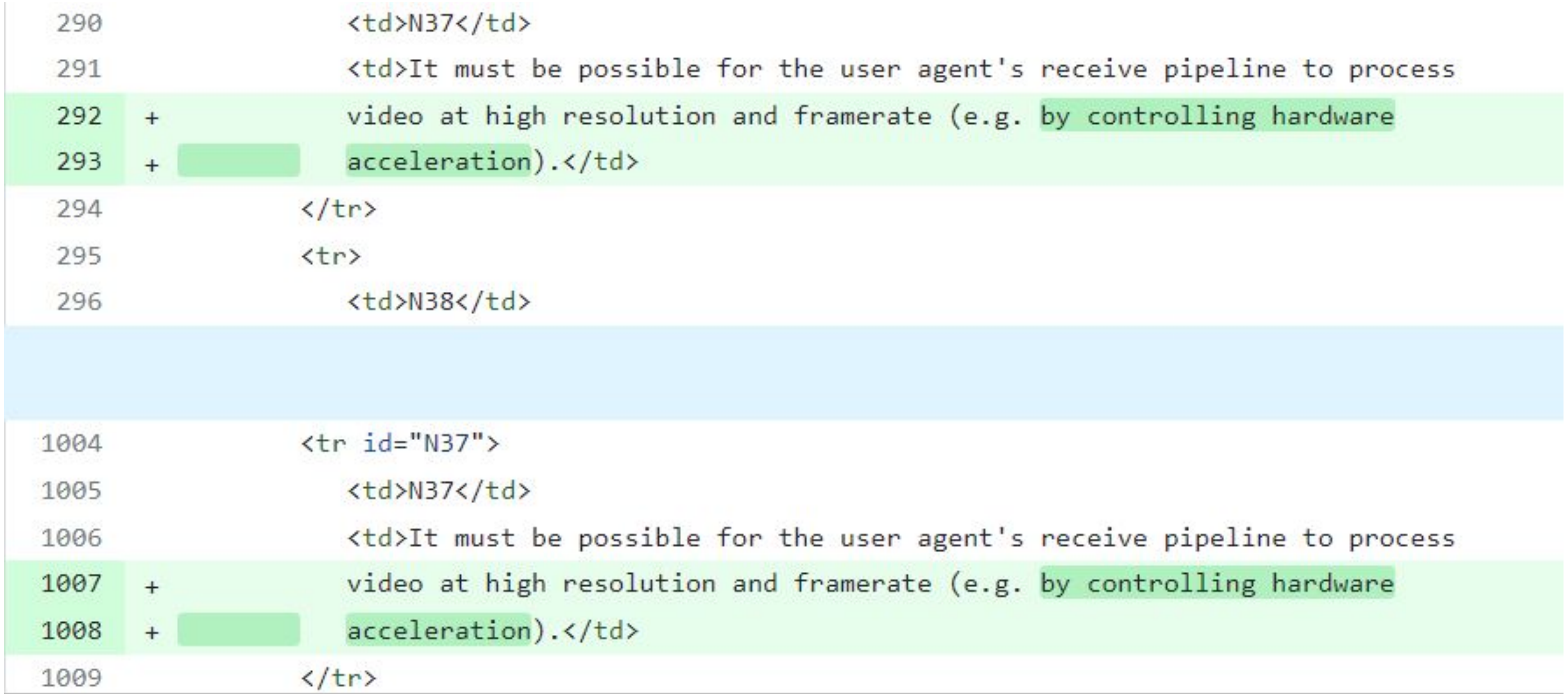
Both use cases are already deployed so I wonder whether they qualify as NV.
They probably qualify as NV if some of their corresponding requirements are not already met with existing web technologies.
When reading the requirements, it seems some/most of them are already met.

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 125: Clarify Requirement N37



```
          <td>N37</td>
          <td>It must be possible for the user agent's receive pipeline to process
+         video at high resolution and framerate (e.g. by controlling hardware
+         acceleration).</td>
        </tr>
        <tr>
          <td>N38</td>
```

```
        <tr id="N37">
          <td>N37</td>
          <td>It must be possible for the user agent's receive pipeline to process
+         video at high resolution and framerate (e.g. by controlling hardware
+         acceleration).</td>
        </tr>
```

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)

- Fast Recovery

ID	Requirement	Description	Benefits to Cloud Gaming	Cloud Gaming Specific?
N48	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	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.	

PR 118: Clarify Game Streaming requirements (Section 3.2.1)

- Consistent Latency

ID	Requirement	Description	Benefits to Cloud Gaming	Cloud Gaming Specific?
N50	Configurable RTCP transmission interval	The application must be able to configure RTCP feedback transmission interval (e.g., Transport-wide RTCP Feedback Message).	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).	In general, short latency is very important, but consistent latency is even more important for the cloud gaming.
N51	Improve accuracy of Jitter buffer control	The user agent needs to provide the jitter buffer to account for jitter in the pipeline up to the frame render stage - WebRTC Issue: 15535	Increases accuracy of jitter buffer adaptation and helps maintain consistent latency	

N48 Recovery using non-key frames

- 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)

→ Is there enough consensus to add this requirement to WebRTC requirements list? Exactly how it is solved (if solvable) can be discussed later, we are working with the use cases and requirements in this document.

The application must be able to control video decoding to continue even after a frame-loss without waiting for a key frame. This enables fast recovery from lossy network conditions.

N49: Loss of encoder -decoder synchronicity notification

- IETF discussion relating to reference feedback in HEVC
  - Draft adopted in AVTCORE WG as draft-ietf-avtcore-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)
    - 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

→ Ongoing discussions are about "how" to implement this. Is there consensus about the requirement. We would like to conclude the PR?

: The application must be able to generate signals that indicate to the encoder the loss of encoder-decoder synchronicity (DPB buffers) and the sequence of frame loss using the platform-agnostic protocols. This helps the application choose the right recovery method in a lossy network.

N50: Configurable RTCP transmission interval

- We found the implementation and need to confirm the Working Group feedback
    - RFC 4585: Extended RTP Profile for Real-time Transport Control Protocol (RTCP)-Based Feedback (RTP/AVPF) (rfc-editor.org)
        - The trr-int parameter indicates the interval between regular RTCP packets in milliseconds. The syntax is as follows:
            - a=rtcp-fb:pt trr-int interval
            - where pt is the payload type and interval is the desired value in milliseconds. If the interval is zero, it means that regular RTCP packets are not expected. The trr-int parameter can be specified at the media level or at the payload type level.
            - a=rtcp-fb:97 trr-int 100 // **regular RTCP packets are expected every 100 milliseconds for payload type 97**
    - Current syntax does not satisfy our requirements since it is generic for all RTCP messages.

→ Ongoing discussions are about "how" to implement this. Is there consensus about the requirement. We would like to conclude the PR? Why can't we have a requirement on enabling quicker reacting (transport wide) congestion control (presumably enabled by frequent RTCP reports)?

: The application must be able to configure RTCP feedback transmission interval (e.g., Transport-wide RTCP Feedback Message). This helps the application adapt the video quality to the varying network and maintain consistent latency.

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 : `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

→ Is there enough consensus to add this requirement to WebRTC requirements list?

The user agent needs to provide the jitter buffer to account for jitter in the pipeline up to the frame render stage. This helps the application adapt the video quality to the varying network and maintain consistent latency.

Discussion (End Time: 08:50)

-

RtpTransport
Start Time: 08:50 AM
End Time: 09:50 AM

GitHub Repo: https://github.com/w3c/webrtc-rtptransport/

RtpTransport

1. Review Current State
2. Addressing Previous Comments
3. Issues
   a. Issue 9: Customizing piecemeal
   b. Issue 10: SDP “Bumper lanes”
   c. Issue 7: Support for SRTP/cryptex
   d. Issue 13: Workers
   e. Issue 8: WHATWG streams
4. Jan-Ivar's Slides

RtpTransport

1. **Review Current State**
2. Addressing Previous Comments
3. Issues
    a. Issue 9: Customizing piecemeal
    b. Issue 10: SDP “Bumper lanes”
    c. Issue 7: Support for SRTP/cryptex
    d. Issue 13: Workers
    e. Issue 8: WHATWG streams
4. Jan-Ivar's Slides

Review Current State

To swap back into your memory the current state:

RtpTransport Reminder

TL;DR: Rough Consensus So Far

- We want some kind of packet-level API for sending and receiving RTP/RTCP.
- Packets must always be encrypted using SRTP/SRTCP.
- Packets must be congestion controlled (as much as RTP/RTCP usually is).

TL;DR: Explainer

https://github.com/w3c/webrtc-rtptransport/blob/main/explainer.md

Summary: The goal is for the web app to be able to do custom things:

- custom payloads/codecs
- custom packetization
- custom FEC
- custom NACK/RTX
- custom metadata
- custom RTCP messages
- etc

TL;DR: Explainer (cont'd)

https://github.com/w3c/webrtc-rtptransport/blob/main/explainer.md

Code examples are *speculative* and *immature*. They assume the model presented at TPAC, don't represent subsequent discussion, and probably have mistakes:

- RtpTransport is constructed via `RTCPeerConnection.createRtpTransport()`
- RtpTransport has a `writable` stream for sending
- RtpTransport receives via an `onrtppacket` EventHandler

This is relevant to issues 14 and 15

TL;DR: Spec

https://github.com/w3c/webrtc-rtptransport/blob/main/index.bs

Summary: Just a skeleton so far. But hopefully soon:

- RtpPackets: payload type, timestamp, header extensions, payload, …
- RtcpPackets: payload type, subtype, payload, …

But that may change with discussions we have today…

RtpTransport

1. Review Current State
2. **Addressing Previous Comments**
3. Issues
    a. Issue 9: Customizing piecemeal
    b. Issue 10: SDP “Bumper lanes”
    c. Issue 7: Support for SRTP/cryptex
    d. Issue 13: Workers
    e. Issue 8: WHATWG streams
4. Jan-Ivar's Slides

Addressing Previous Comments

From https://www.youtube.com/watch?v=xJMXnf3Qwh8 around 1:08

- Harald: "There might be two modes here"
    - A mode that can only send/receive what is negotiated in SDP for an m-line (e.g. “bumper lanes”).
    - A mode that can send/receive anything on the PeerConnection.
- Jan-Ivar: "I imagined a new type of data channel"
    - Replacing the source of RTP rather than providing control of RTP.
    - The input is a MediaStreamTrack or a writable (pre-encoded data)
- Lots of suggestions for specifics (API shape, RTP specifics)
- "Piecemeal Customization"

Before specifics, what general direction?

New type of Data Channel

Piecemeal Customization

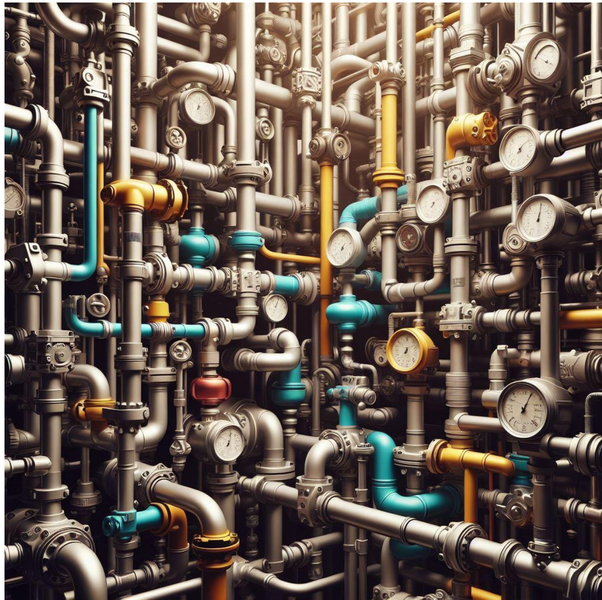
TPAC RtpTransport

Strawman: A new type of data channel

- PeerConnection.createRtpDataChannel()
  - Can only send RTP, not RTCP
  - You don't pick payload type, SSRC, seqnum, header extensions, …
  - You pick the payload and maybe the timestamp
- A sufficiently sophisticated app could use this by just putting everything into the payload.
- It's not as compatible with existing endpoints.
- But maybe that's enough.

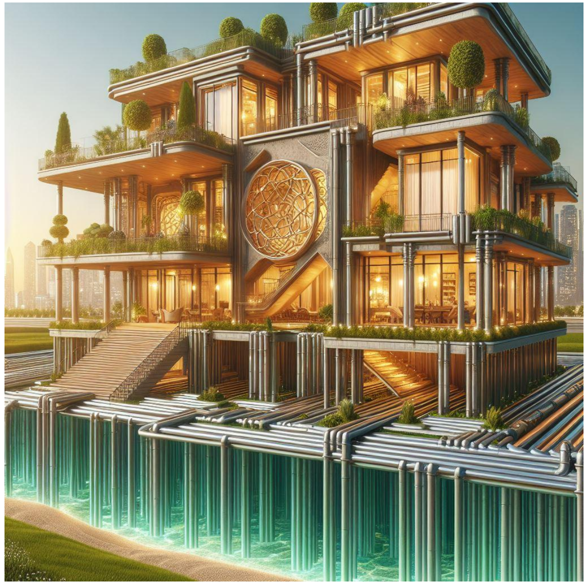
Strawman: Full Piecemeal Customization

- RtpSender.getRtpStream("rid")
  - Capture the RTP packets that *would have been sent*
  - Insert the RTP packets that *will be sent* (maybe leave unmodified; maybe replace)
- Let's you customize things *piecemeal.* e.g:
  - Keep packetization but change RTX
  - Keep RTX but change packetization
- Probably more convenient for an app
- But must have a *way more complex API* (because RTP is complex)



Strawman: RtpTransport from TPAC

- Roughly halfway between the previous two
  - It's "you do everything" kind of like the new type of data channel
  - With the low-level RTP packet control of full piecemeal customization



Another way to think about it…

	Can customize RTP to/from RtpReceiver/RtpSender	Cannot
High Level	🤨	New Type of Data Channel
Low Level	Full Piecemeal Customization	TPAC RtpTransport

Or another way to think about it…

Simple

Complex

New Type of Data Channel

TPAC RtpTransport

Full Piecemeal Customization

Strawman Battle

- Are there other ideas?
- Which do we want?
- How do we decide?

RtpTransport

1. Review Current State
2. Addressing Comments from Last Time
3. **Issues**
    a. Issue 9: Customizing piecemeal
    b. Issue 10: SDP “Bumper lanes”
    c. Issue 7: Support for SRTP/cryptex
    d. Issue 13: Workers
    e. Issue 8: WHATWG streams
4. Jan-Ivar's Slides

Issue 9: Customizing Piecemeal

- If new type of data channel or TPAC RtpTransport:
    - Not possible
- If full piecemeal customization
    - Built into the API from the beginning
    - Could match RFC 7656 with types like:
        - `RtpStream`: 1 ssrc; N packets
        - `RtpRepairedStream`: 1 RtpStream; 1+ RTX RtpStreams
        - RtpSender: N RtpRepairedStreams (outbound)
        - RtpReceiver: 1 RtpRepairedStream (inbound)

Issue 10: SDP “Bumper lanes”

- If new type of data channel:
    - Bumper lanes strictly enforced (you can't control much)
- If full piecemeal customization:
    - Bumper lanes lightly enforced
      (perhaps SSRCs, PTs, header ext IDs)
- If TPAC RtpTransport:
    - No bumper lanes

Issue 7: SRTP/Cryptex (RFC 9335)

- If strawman new type of data channel or TPAC RtpTransport:
  - Add an API point to turn on or off (probably for all packets, not per-packet)
- If full piecemeal customization:
  - Just use what's negotiated?
- In all cases:
  - Applied at SRTP layer
  - Web app never sees encrypted packet
  - Like DTLS/SCTP for data channels or QUIC for WebTransport
  - It just happens underneath

Issue 13: Workers

- If new type of data channel:
  - Transfer the new "data channel object" (BTW, we probably want undirectional objects)
- If full piecemeal customization:
  - Transfer the "RtpStream"?
- If TPAC RtpTransport:
  - Transfer the whole RtpTransport?

Issue 11: Arbitrary RTCP

Can a web app send arbitrary bytes in RTCP messages? Such as XR or LNTF?

- If new type of data channel:
    - No (must be embedded in RTP payload)
- If full piecemeal customization:
    - Maybe (if we want to customize RTCP)
- If TPAC RtpTransport:
    - Yes

Issue 12: Arbitrary RTP Header Extensions

Can a web app send arbitrary RTP header extensions? Like custom metadata?

- If new type of data channel:
  - No (must be embedded in RTP payload)
- If full piecemeal customization:
  - Maybe (if we want to customize RTP metadata outside payload)
- If TPAC RtpTransport:
  - Yes

Issue 8: WHATWG Streams

We could spend hours debating the merits of WHATWG streams (or demerits).

Let's put this aside until we figure out the more important matters.

We could do *any* direction with or without WHATWG streams
(depending on how much pain we want to inflict upon everybody)

Back to the question of which path to take

RtpTransport

1. Review Current State
2. Addressing Comments from Last Time
3. Issues
    a. Issue 9: Customizing piecemeal
    b. Issue 10: SDP “Bumper lanes”
    c. Issue 7: Support for SRTP/cryptex
    d. Issue 13: Workers
    e. Issue 8: WHATWG streams
4. **Jan-Ivar's Slides**

(Jan-Ivar) RtpTransport

More open issues

a. Issue 13: Workers
b. Issue 14: Update examples with standard APIs
c. Issue 15: Examples pipe multiple streams into a single writable

(Jan-Ivar) WHATWG Streams

The RtpTransport explainer examples use streams. That's good!
WebTransport and WebSocket/WebRTC already have two different data input APIs. Let's not invent a third.

- RTP could have back pressure on the send side, but do you really want it?

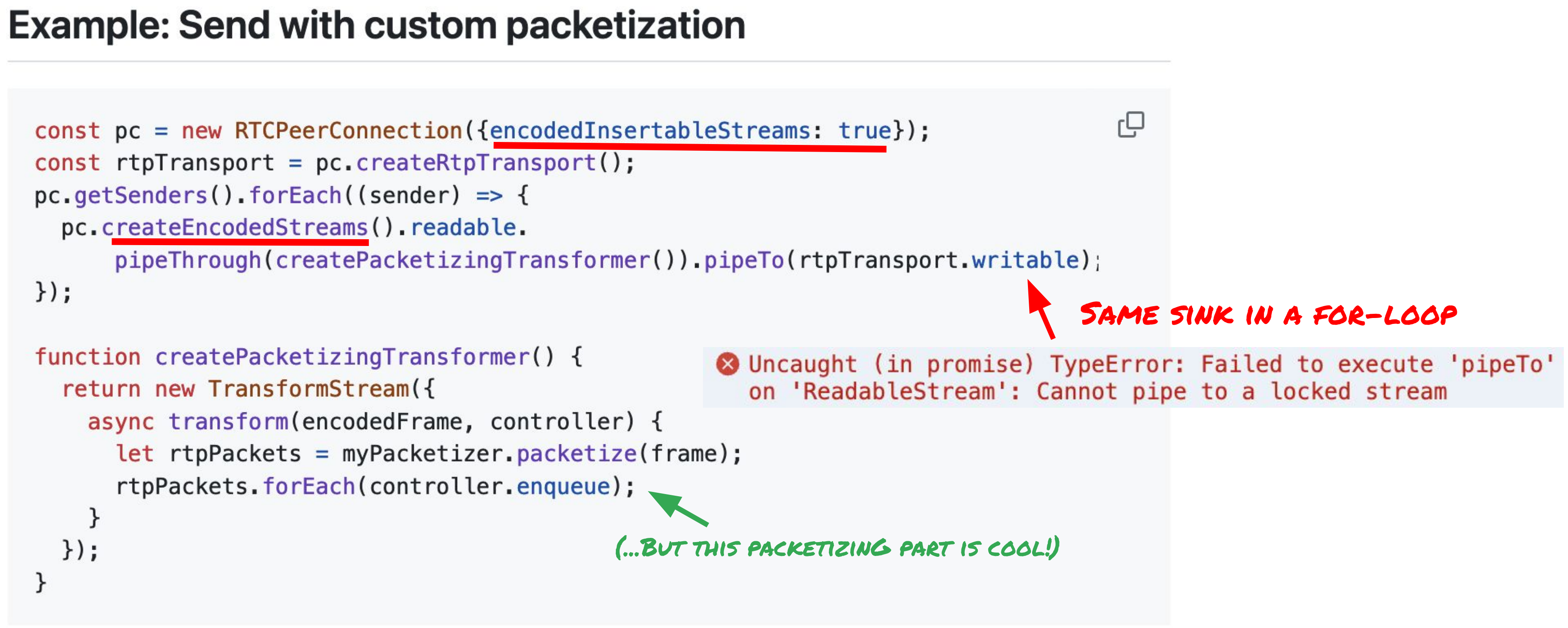
Yes, for the same reason
WebTransport datagrams have it:

1. Lets apps hook up pull-based sources
2. Lets browser keep its off-process send buffer filled for throughput

3. Our transform APIs already use it

(Jan-Ivar) Issue 14/15: Examples pipe multiple streams into writable

The explainer examples won't work. They use non-standard APIs and pipe multiple streams into one writable:



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);
    }
  });
}
```

SAME SINK IN A FOR-LOOP

```
Uncaught (in promise) TypeError: Failed to execute 'pipeTo'
on 'ReadableStream': Cannot pipe to a locked stream
```

(...BUT THIS PACKETIZING PART IS COOL!)

They also fail to address "worker-first" feedback from @jesup and myself, as well as Issue #13 **Workers**.

(Jan-Ivar) Issue 13/14/15: Update examples to use standard APIs

Updating them to spec solves workers & lets us move rtpTransport off main-thread to become a packetizer sink:

```
// main.js
const pc = new RTCPeerConnection();
for (const sender of pc.getSenders()) {
  sender.transform = new RTCRtpScriptTransform(worker);
}

// worker.js
onrtctransform = async ({transformer}) => {
  const {readable} = transformer;
  const {writable} = transformer.createRtpTransport();
  await readable.pipeThrough(new TransformStream({transform})).pipeTo(writable);

  function transform(frame, controller) {
    for (const packet of myPacketizer.packetize(frame)) {
      controller.enqueue(packet);
  }
};
```

(Jan-Ivar) Issue 13/14/15: Media pipeline in Workers

Conceptually, all we've done is switch the output type from encodedFrame to packet. Much like @alvestrand's JS codec/Lyra use case required input switched to unencoded Frame.

What if we let JS change the expected input and output types? (sender-side):

1.	**encodedFrame → encodedFrame**	(e2ee) *(default)*
2.	**encodedFrame → packet**	(add metadata)
3.	**Frame → encodedFrame**	(JS encoder)
4.	**Frame → packet**	(hold my beer!)

This closes the gap on Frame vs packet APIs, all while preserving a worker-first API.

Why transforms?

We built a pipeline into the browser, and the idea that it would be accessible was not really a central idea when building it. This lets the entire pipeline be accessible at different points, off main-thread.

(Jan-Ivar) Issue 13/14/15: WebCodecs + WebRTC

Two more use cases (sender-side):

1. **JS → encodedFrame** (WebCodecs)
2. **JS → packet** (WebCodecs)

Why? Because people will (probably) ask for it.

Does this break with transforms?

Yes, but short of transferring RTCPeerConnection, any API is going to be a retrofit. E.g.

```
// main.js
sender.source = new RTCRtpScriptEncoder(worker); // mutually exclusive w/sender.strack

// worker.js
onrtcencode = async ({transformer: {writable}}) => {
  await myData().pipeThrough(new EncodeVideoStream({codec: "vp8"})).pipeTo(writable);
```

Or is just overloading `sender.transform` simpler? What about SFrameTransform?

Discussion (End Time: 09:50)

-

Wrapup and Next Steps

Start Time: 09:50 AM

End Time: 10:00 AM

Title Goes Here

- Content goes here

Thank you

Special thanks to:

WG Participants, Editors & Chairs

PR 123: Use Cases Removed

- Church services, Webinars and Town Hall meetings removed
  - These use cases typically do not require ultra low latency
    - Broadcast often handled by conventional streaming technology (e.g. LL-HLS)
    - Fanout can be addressed using RTCDataChannel (e.g.Peer5).
    - RTCDataChannel requirements (e.g. worker support) covered in Section 3.1: File Sharing.
- Sporting events also removed.
  - While this requires low latency and feedback, it also needs content protection.
    - CMAF streaming can be addressed by MoQ or other ULL streaming protocol.
    - Fanout can be addressed using unreliable/unordered RTCDataChannel with custom FEC.
    - RTCDataChannel requirements (e.g. worker support) covered in Section 3.1: File Sharing.
    - Participant feedback (cheers) handled via WebRTC?
    - Content protection requires CMAF, absent DRM support for encodedChunks:
      - Issue 41: Support for content protection
      - https://rawgit.com/wolenetz/media-source/mse-for-webcodecs-draft/media-source-respec.html#webcodecs-based