

W3C WebRTC WG Meeting

November 21, 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 November 2023 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/November 21 2023](https://www.w3.org/2011/04/webrtc/wiki/November_21_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-rtpttransport>
- 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.

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

- 08:05 - 08:40 AM [Grab Bag](#) (Henrik, Fippo, Sameer, Jan-Ivar, Florent)
- 08:40 - 09:00 AM [SDP Codec Negotiation \(Harald\)](#)
- 09:00 - 09:20 AM [RtpTransport \(Harald, Bernard & Peter\)](#)
- 09:20 - 09:40 AM [Three-Thumbs-Up](#) (Elad & Guido)
- 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.

Grab Bag

Start Time: 08:05 AM

End Time: 08:40 AM

For Discussion Today

- **Mediacapture-Extensions (Henrik & Harald)**
 - [Issue 121](#): Background blur: unprocessed video should be mandatory
 - [Issue 129](#): [Audio-Stats] Disagreement about audio dropped counters (Henrik)
- **WebRTC-Extensions (Fippo & Henrik)**
 - [Issue 146](#): Exposing decode errors/SW fallback as an event
- **WebRTC-SVC (Bernard)**
 - [Issue 92](#): Align exposing scalabilityMode with WebRTC “hardware capabilities” check
- **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'

Issue 121: Background Blur: Unprocessed video should be mandatory

- For background blur and similar platform effects, potential for confusion when effects can be applied both in the application and the platform.
 - Users don't know where to turn effects off
 - Combined app + platform processing may cause unintended effects. Examples:
 - Double blur
 - Competing face touchups and background replacements
 - Unintended activation of emoji reactions
- Proposal: such capabilities “MUST support constraint=false”
 - Ensures that the application can turn processing off and get an unprocessed video feed.

Issue 129: [Audio-Stats] Disagreement about audio dropped counters

The WG decided to add the following metrics ([April](#) and [October](#) Virtual Interims):

```
interface MediaStreamTrackAudioStats {  
  readonly attribute unsigned long long deliveredFrames;  
  readonly attribute DOMHighResTimeStamp deliveredFramesDuration;  
  readonly attribute unsigned long long totalFrames;  
  readonly attribute DOMHighResTimeStamp totalFramesDuration;  
  [Default] object toJSON();  
};
```

$\text{glitchRatio} = 1 - \text{deliveredFrames} / \text{droppedFrames}$

Allows calculating dropped frames and glitch ratios, a measure of audio quality.

Drops can happen by OS:

- Causes: device bugs, OS bugs, CPU starvation, UA underperforming, etc.
- UA can get info from OS (e.g. CrOS) or deduce based on timestamps (e.g. macOS).

Drops can happen by User Agent:

- IPC delays, processing delays, UA bugs.

BUT... Audio glitches are *rare* and *difficult* for an app to affect on a case-by-case basis.

Issue 129: [Audio-Stats] Disagreement about audio dropped counters

There's an **alleged problem**:

1. Frame drops are usually a sign of UA bugs.
2. If they happen (bug or not), the app can't save itself. "Catastrophic."
3. Conclusion: *We shouldn't expose this to JS.*

Google disagrees with both the premises and the conclusion. Apps needs to monitor quality:

- A/B testing.
 - Real-world data has shown that app features (e.g. affecting performance or triggering new JS or UA code paths) can improve or regress audio quality, including glitches.
- Understanding app-specific bug reports of individual users (e.g. upload quality graphs).
 - Is bad audio due to OS, UA, device, app bugs, capture stack, other?
 - Flag when something is wrong.
- An app being able to identify issues it is experiencing is a Good Thing.
- [Intent to Prototype](#) shows dev interest from Microsoft Teams, GoTo Meetings and Zoom.

Decision needed: *Should audio frame drops be exposed to JS?*

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

- Simplified proposal
 - Align with resolution of WebCodecs [Issue 669](#), ([Media WG](#))
 - `EncodingError` for input data errors
 - Omit `rid` until proven necessary
 - `OperationError` to indicate a resource problem
- Privacy seems fine, see [this comment](#)
- Ready for PR?

Issue 92: Align exposing scalabilityMode with WebRTC “hardware capabilities” check

- WebRTC-Stats Section 6.1 contains the following text:

§ 6.1 Limiting exposure of hardware capabilities

To avoid passive fingerprinting, hardware capabilities should only be exposed in capturing contexts. This is tested using the algorithm below.

To **check if hardware exposure is allowed**, run the following steps:

1. If the [context capturing state](#) is true, return true.
2. Otherwise return false.

- Issued filed by PING in w3cping/privacy-request#117
 - Concern about use of `scalabilityMode` discovery for hardware fingerprinting.

Issue 92: Align exposing scalabilityMode with WebRTC “hardware capabilities” check

- Problem #1: WebRTC-SVC uses Media Capabilities API for discovery
 - Indicates if a configuration is “supported” “power efficient” or “smooth”.
 - Media Capabilities API not limited to capture context
- Problem #2: SVC rarely supported in hardware
 - Today, few devices support “powerEfficient” SVC.
 - Simulcast often preferred to SVC to save power.
 - Result: **scalabilityMode** support of little value for hw fingerprinting.
- Problem #3: WebRTC-SVC exposes less information than Media Capabilities
 - Calling `RTCRtpSender.setParameters()` or `addTransceiver()` with `RTCRtpEncodingParameters.codec` exposes whether configuration is “supported”, but not “powerEfficient” or “smooth”.

MediaCapabilities on MacBook Air M2 (Safari Tech Preview 183)

<https://webrtc.internaut.com/mc/>

chrome://gpu info

WebRTC-SVC Media Capabilities Discovery

info: DOM Content Loaded
info: Codec selected: AV1
info: Default (QVGA) selected
info: AV1 L1T1 is: supported, NOT smooth and NOT power efficient
info: AV1 L1T2 is: supported, NOT smooth and NOT power efficient
info: AV1 L1T3 is: supported, NOT smooth and NOT power efficient
info: AV1 L2T1 is: supported, NOT smooth and NOT power efficient
info: AV1 L2T1_KEY is: supported, NOT smooth and NOT power efficient
info: AV1 L2T1h is: supported, NOT smooth and NOT power efficient
info: AV1 L2T2 is: supported, NOT smooth and NOT power efficient
info: AV1 L2T2_KEY is: supported, NOT smooth and NOT power efficient
info: AV1 L2T2_KEY_SHIFT is: supported, NOT smooth and NOT power efficient
info: AV1 L2T2h is: supported, NOT smooth and NOT power efficient
info: AV1 L2T3 is: supported, NOT smooth and NOT power efficient
info: AV1 L2T3_KEY is: supported, NOT smooth and NOT power efficient
info: AV1 L2T3_KEY_SHIFT is: supported, NOT smooth and NOT power efficient
info: AV1 L2T3h is: supported, NOT smooth and NOT power efficient
info: AV1 L3T1 is: supported, NOT smooth and NOT power efficient
info: AV1 L3T1_KEY is: supported, NOT smooth and NOT power efficient
info: AV1 L3T1h is: supported, NOT smooth and NOT power efficient
info: AV1 L3T2 is: supported, NOT smooth and NOT power efficient
info: AV1 L3T2_KEY is: supported, NOT smooth and NOT power efficient
info: AV1 L3T2_KEY_SHIFT is: supported, NOT smooth and NOT power efficient
info: AV1 L3T2h is: supported, NOT smooth and NOT power efficient
info: AV1 L3T3 is: supported, NOT smooth and NOT power efficient
info: AV1 L3T3_KEY is: supported, NOT smooth and NOT power efficient
info: AV1 L3T3_KEY_SHIFT is: supported, NOT smooth and NOT power efficient
info: AV1 L3T3h is: supported, NOT smooth and NOT power efficient
info: AV1 S2T1 is: supported, NOT smooth and NOT power efficient
info: AV1 S2T1h is: supported, NOT smooth and NOT power efficient
info: AV1 S2T2 is: supported, NOT smooth and NOT power efficient
info: AV1 S2T2h is: supported, NOT smooth and NOT power efficient
info: AV1 S2T3 is: supported, NOT smooth and NOT power efficient
info: AV1 S2T3h is: supported, NOT smooth and NOT power efficient
info: AV1 S3T1 is: supported, NOT smooth and NOT power efficient
info: AV1 S3T1h is: supported, NOT smooth and NOT power efficient
info: AV1 S3T2 is: supported, NOT smooth and NOT power efficient
info: AV1 S3T2h is: supported, NOT smooth and NOT power efficient
info: AV1 S3T3 is: supported, NOT smooth and NOT power efficient
info: AV1 S3T3h is: supported, NOT smooth and NOT power efficient

Video Acceleration Information

Decoding	
Decode h264 baseline	16x16 to 4096x4096 pixels
Decode h264 main	16x16 to 4096x4096 pixels
Decode h264 extended	16x16 to 4096x4096 pixels
Decode h264 high	16x16 to 4096x4096 pixels
Decode vp9 profile0	16x16 to 4096x4096 pixels
Decode vp9 profile2	16x16 to 4096x4096 pixels
Decode hevc main	16x16 to 8192x8192 pixels
Decode hevc main 10	16x16 to 8192x8192 pixels
Decode hevc main still-picture	16x16 to 8192x8192 pixels
Decode hevc range extensions	16x16 to 8192x8192 pixels
Encoding	
Encode h264 baseline	0x0 to 4096x2304 pixels, and/or 120.000 fps.
Encode h264 baseline	2x2 to 4096x2304 pixels, and/or 120.000 fps (software codec).
Encode h264 main	0x0 to 4096x2304 pixels, and/or 120.000 fps.
Encode h264 main	2x2 to 4096x2304 pixels, and/or 120.000 fps (software codec).
Encode h264 high	0x0 to 4096x2304 pixels, and/or 120.000 fps.
Encode h264 high	2x2 to 4096x2304 pixels, and/or 120.000 fps (software codec).

Issue 92: Align exposing scalabilityMode with WebRTC “hardware capabilities” check

- Section 4.2.1: addTransceiver() validation
 - If *sendEncodings* contains any encoding with a [RTCRtpEncodingParameters.codec](#) value *codec* [exists](#) and where the same encoding's *scalabilityMode* value is not supported by *codec*, [throw](#) an [OperationError](#).
- Section 4.2.2: setParameters() validation
 - If *encodings* contains any encoding with an existing [RTCRtpEncodingParameters.codec](#) value *codec*, where the same encoding's *scalabilityMode* value is not supported by *codec*.
- Proposed resolution
 - [Issue 209](#) filed on Media Capabilities API.
 - To be discussed in MEDIA WG.
 - If MEDIA WG decides to limit MC to capture context, will bring a proposal to WEBRTC WG to provide equivalent support in WebRTC-PC.

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”.

Discussion (**End Time: 08:40**)

-

SDP Codec Negotiation (Harald)

Start Time: 08:40 AM

End Time: 09:00 AM

Issue 186: New API for SDP negotiation

After discussion with Jan-Ivar who thought the API was too complex, a revised proposal:

- Add transceiver function:

```
transceiver.addCodec(RTCRtpCodec codec, RTCRtpCodec packetizationMode)
```

- Can only be called while transceiver is being created (just like transform)

- Add utility helper:

```
pc.addCodecCapability(DOMString kind, RTCRtpCodec codec, RTCRtpCodec packetizationMode)
```

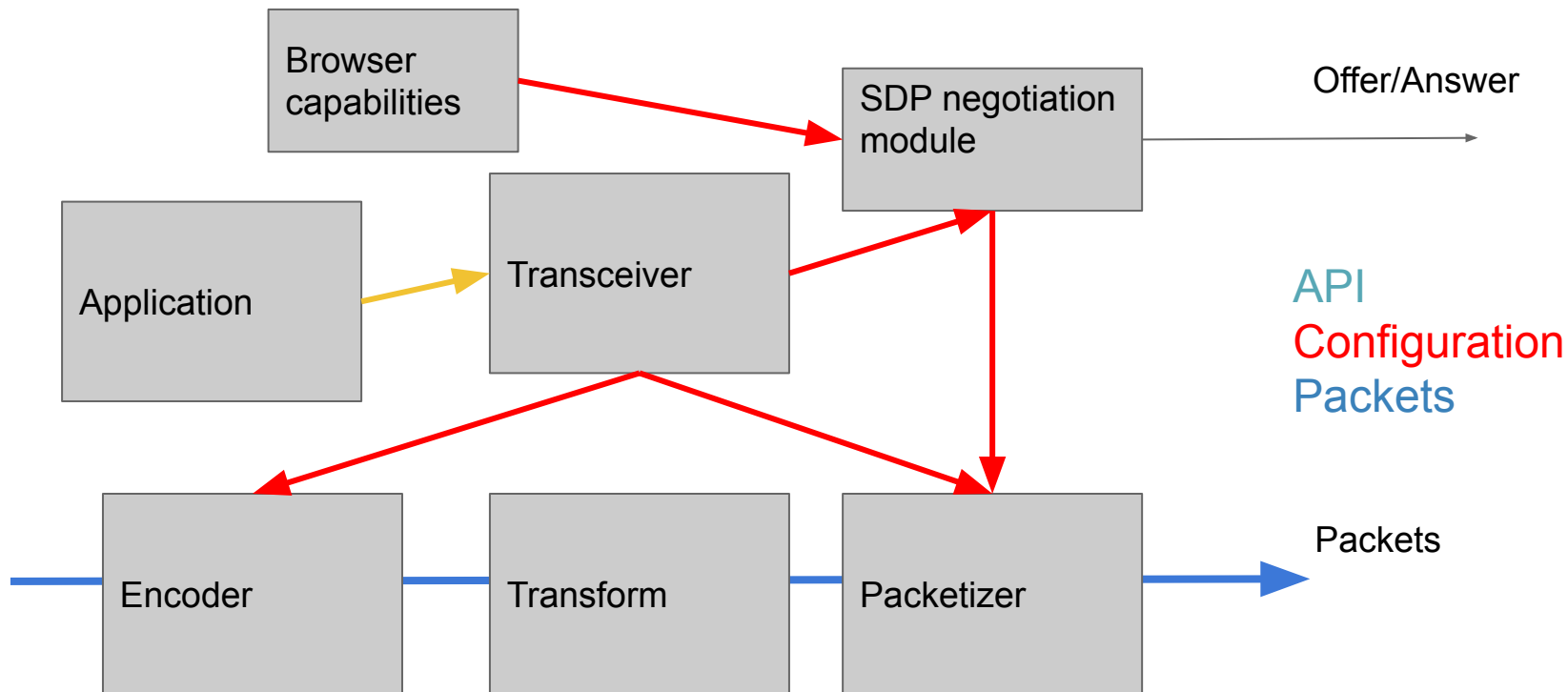
The utility helper merely executes the transceiver function for any newly created transceiver.

Functionality is the same as before.

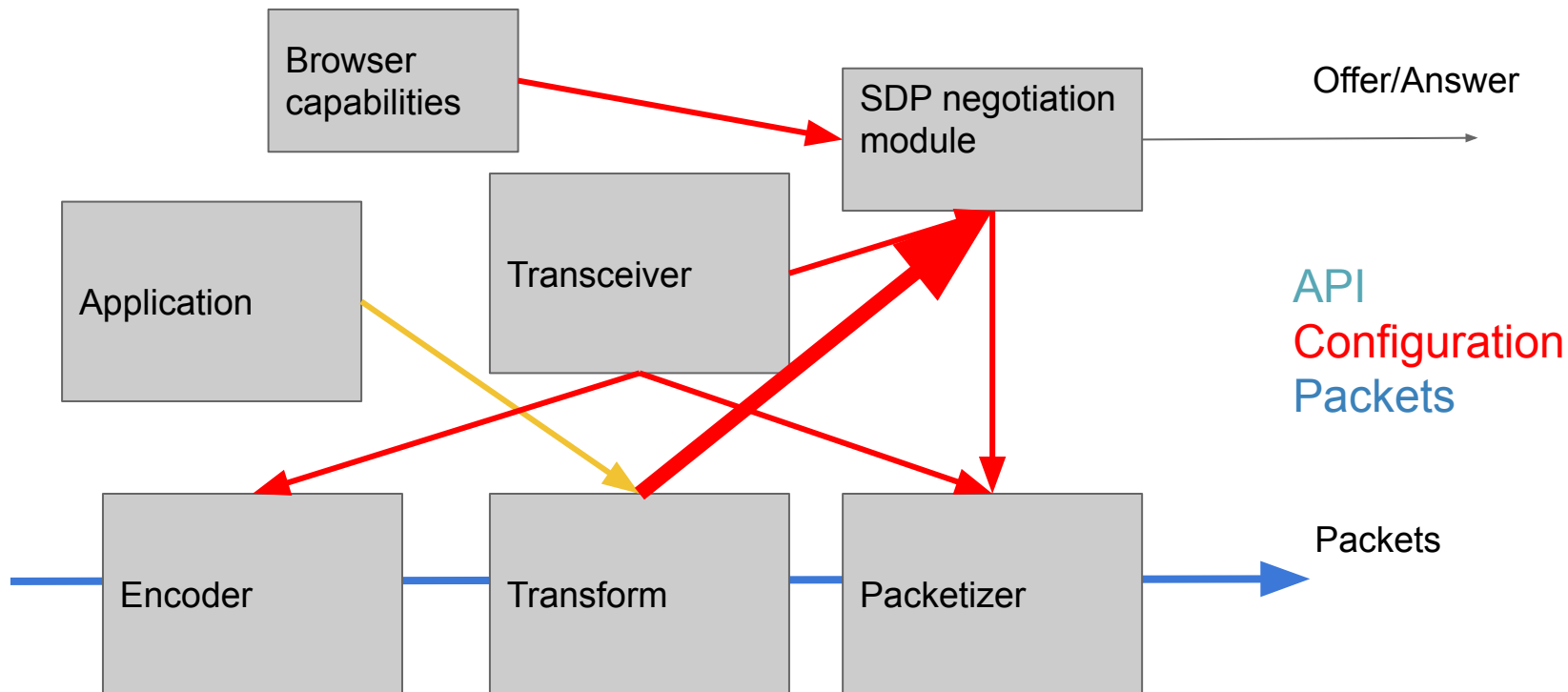
Debated point: API on transceiver or on transform?

- This has been debated in the webrtc-editors meeting, but is really a WG-level issue
- The right decision is to place it on the transceiver.
 - The transceiver is already closely entangled with the SDP negotiation
 - It's where we set other functions that affect SDP negotiations
 - The transform is only concerned with the movement of frames
 - If we ever get senders/receivers that are detached from SDP, they may still have transforms in them. They should not have SDP linkage.
 - If we ever get “one-ended transforms” (which are unlikely to be transforms), several use cases will still need the SDP negotiation API. Separation of concerns argues against requiring a transform in order to set SDP.

Present proposal - API on transceiver



With API on transform



Open issue: MIME type or PT?

- The format of a frame may be indicated by its PT (a number, looked up in the PC's mapping table)
- It may also be indicated by its MIME type (a string), which is PC independent - but more complex to match
- Which of those should a transform change?

Discussion (**End Time: 09:00**)

-

RtpTransport

Start Time: 09:00 AM

End Time: 09:20 AM

RtpTransport

1. Review current state
2. Related API discussions
 - a. BWE
 - b. Forwarding
3. Responding to feedback from last time
 - a. Using existing m-lines
 - b. Workers
 - c. WHATWG streams

RtpTransport

- 1. Review current state**
2. Related API discussions
 - a. BWE
 - b. Forwarding
3. Responding to feedback from last time
 - a. Using existing m-lines
 - b. Workers
 - c. WHATWG streams

Review Current State

- There was general agreement on adding an API for sending and receiving RTP/RTCP packets
- For purposes of many use cases like custom payloads, packetization, protection, metadata...
- By incrementally building on PeerConnection, Encoded Streams, and WebCodecs
- So we made a [repository](#) with an [explainer](#)
- And now we're iterating on it

RtpTransport

1. Review current state
2. Related API discussions
 - a. **BWE**
 - b. Forwarding
3. Responding to feedback from last time
 - a. Using existing m-lines
 - b. Workers
 - c. WHATWG streams

BWE API

My proposal: Build BWE API on RtpTransport
(such as this simplified version of Harald's proposal)

```
partial interface RtpTransport {
    attribute BandwidthInfo bandwidthInfo;
    attribute EventHandler onbandwidthestimate;
};
partial interface BandwidthInfo {
    readonly attribute long availableOutgoingBitrate;
};
```

BWE API

You may also want a way to get to it with existing m-lines:

```
partial interface RtpSender {  
    readonly attribute RtpTransport rtpTransport;  
};  
partial interface RtpReceiver {  
    readonly attribute RtpTransport rtpTransport;  
};
```

RtpTransport

1. Review and current state
2. Related API discussions
 - a. BWE
 - b. Forwarding**
3. Responding to feedback from last time
 - a. Using existing m-lines
 - b. Workers
 - c. WHATWG streams

Forwarding API

My proposal: Use RtpTransport for forwarding

- Get packets from remote senders S1, S2, S3
- Send packets to remote receivers R1, R2, R3
- Modify RTP/RTCP packets: PT, SSRC, seqnum, payload, ...
- Use BWE on RtpTransport for rate control

RtpTransport

1. Review current state
2. Related API discussions
 - a. BWE
 - b. Forwarding
3. Responding to feedback from last time
 - a. **Using existing m-lines**
 - b. Workers
 - c. WHATWG streams

Using existing m-lines

- Add RtpReceiver/RtpSender.**rtpTransport**
(from a few slides ago)
- But big questions:
 - What are you prevented from sending?
 - Is built-in demux worth it?
 - How do we "mix" traffic?

What are you prevented from sending?

- Unnegotiated PT? Easy to check
- Invalid Payload? Not easy to check
- Unnegotiated Header Extension ID? Easy
- Invalid Header Extension Value? Not Easy
- Unnegotiated SSRC? MIDs allow any
- Unnegotiated MID/RID? Not too hard
- Unnegotiated RTCP type
- All the m-lines are recvonly/inactive? Easy

What are you prevented from sending?

- So... "bumper lanes" are unnegotiated PTs, Header Extension IDs, MIDs, RIDs, directions, and RTCP types
- But if you really want to send them, you just have to munge the SDP to make them "negotiated", and then you can
- In other words, the web app can turn off the bumper lanes via SDP munging
- So why not make it easy?
`rtpTransport.bumperLanesEnabled = false`

What are you prevented from sending?

Proposal: allow disabling bumper lanes easily, and keep the bumper lanes simple and easy to implement

Is built-in demux worth it?

- RTP demux is easy (RFC 8843, Section 9.2)
- Some RTCP is easy (PLI, FIR, NACK)
- Some RTCP doesn't make sense (CC feedback)
- Some RTCP is impossible (custom)
- And you have to deal with compound RTCP
 - Break it up before demux?

Is built-in demux worth it?

Proposal: build in RTP demux, but not RTCP demux
(most of the benefit for a fraction of the complexity)

How do we "mix" traffic?

- The PC is sending
- And now you use RtpTransport to send
- What could go wrong?
 - You reuse an (SSRC, ROC, seqnum)
 - We might need to be more constrained about what SSRCs you can send with.
 - You send too much
 - We might need API for not just BWE, but also *allocation* (like what Harald has been proposing)

RtpTransport

1. Review current state
2. Related API discussions
 - a. BWE
 - b. Forwarding
3. Responding to feedback from last time
 - a. Using existing m-lines
 - b. Workers**
 - c. WHATWG streams

Workers

- Realistically a "send worker" will want a way to send RTP, receive RTCP (feedback), and get BWE updates; maybe more
- Realistically a "receive worker" will want a way to receive RTP, send RTCP (feedback), receive RTP, and maybe more
- Option A: pull off "parts" of RtpTransport and transfer those
- Option B: transfer the whole of RtpTransport to several workers
- Proposal: transfer the whole of RtpTransport to several workers

RtpTransport

1. Review current state
2. Related API discussions
 - a. BWE
 - b. Forwarding
3. Responding to feedback from last time
 - a. Using existing m-lines
 - b. Workers
 - c. **WHATWG streams**

WHATWG Streams

- Good when you want back pressure
- RTCP does not have back pressure
- RTP has no back pressure on the receive side
- RTP could have back pressure on the send side, but do you really want it?

WHATWG Streams

- Good when you want a filtered view
- `RtpTransport.receiveRtp({pt: 99})` or
`RtpTransport.receiveRtp({mid: "a"})` or
`RtpTransport.receiveRtcp({type: 206})`

WHATWG Streams

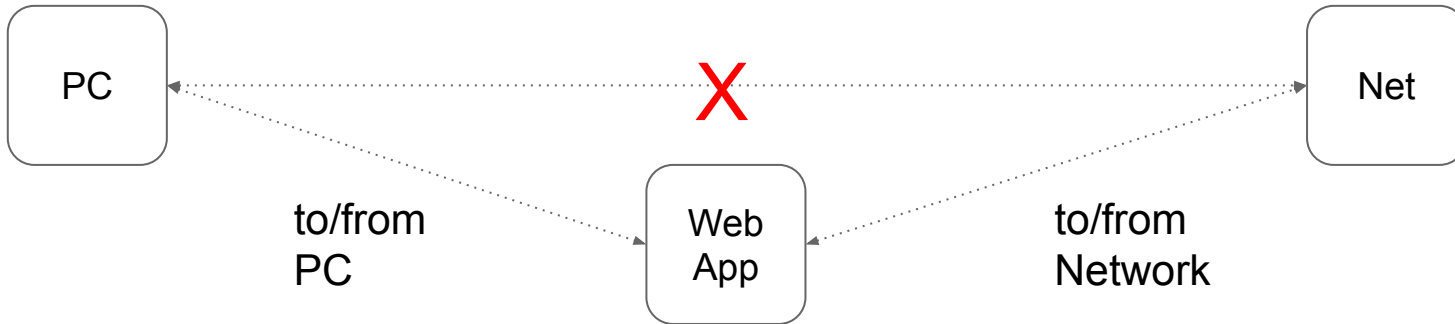
Proposal: Don't use WHATWG streams right now, but maybe use it if/when have somewhere they make sense

RtpTransport

1. Review current state
2. Related API discussions
 - a. BWE
 - b. Forwarding
3. Responding to feedback from last time
 - a. Using existing m-lines
 - b. Workers
 - c. WHATWG streams
4. **Bonus (if time allows)**

Bonus #1: "Modify" Packets

- Problem
 - PeerConnections sends/receives RTP/RTCP to/from the network.
 - You want to modify RTP/RTCP between PeerConnection and the network. (Custom FEC, for example; Not the same as modifying individual packets!)
 - The easy part: you can send/receive to/from network using RtpTransport.
 - The hard part: intercepting/injecting RTP/RTCP from/to PeerConnection
- Solution: an "inverted" RtpTransport
 - Instead of RTP/RTCP to/from network, they are to/from the PeerConnection
 - It's like you're forwarding between the local PC and the remote endpoint



Bonus #2: RTP over WebTransport

```
const wt = new WebTransport(url);
const writer = wt.datagrams.writable.getWriter();
let rtpPackets = myPacketizer.packetize(frame, mtu);
await sendRtpPackets(writer, rtpPackets);

async function sendRtpPackets(writer, rtpPackets) {
  for (const rtpPacket of rtpPackets) {
    await writer.ready;
    writer.write(rtpPacket.toArrayBuffer()).catch(() => {});
  }
}
```

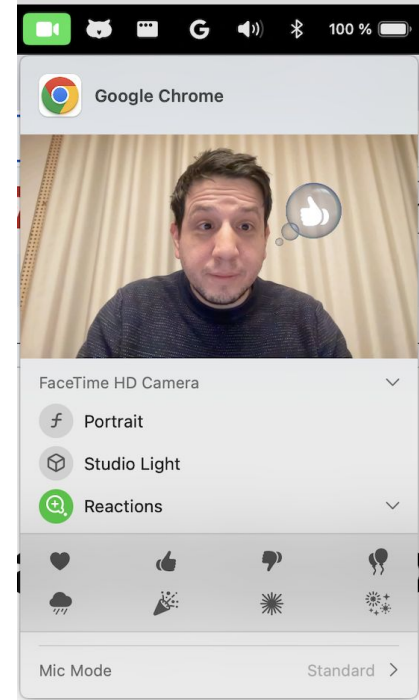
Discussion (**End Time: 09:20**)

-

Three Thumbs Up - Mute (Elad & Guido)

Start Time: 09:20 AM

End Time: 09:40 AM

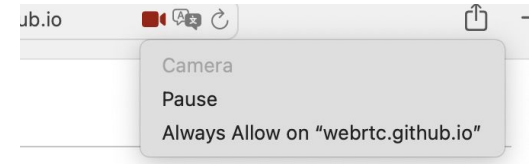
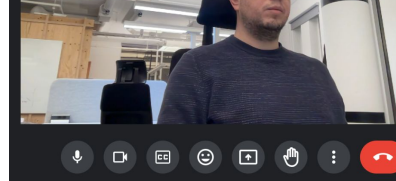


Three Thumbs Up - Setup

What happens when multiple entities offer similar functionality to the user?

Entities:

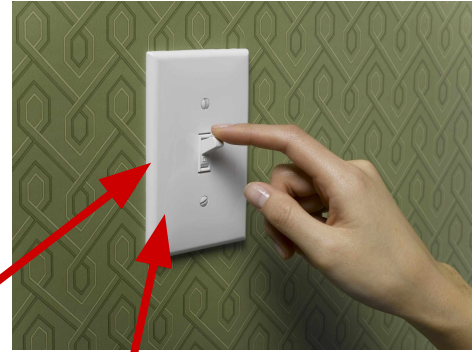
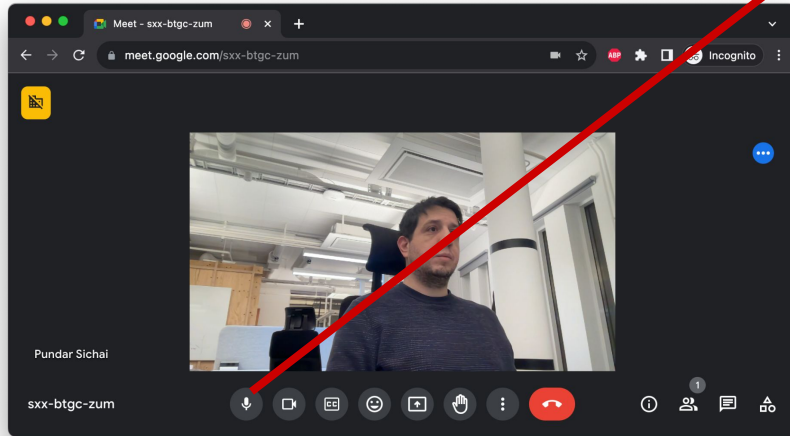
- Web application
- User agent
- Operating system



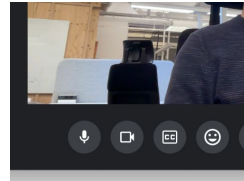
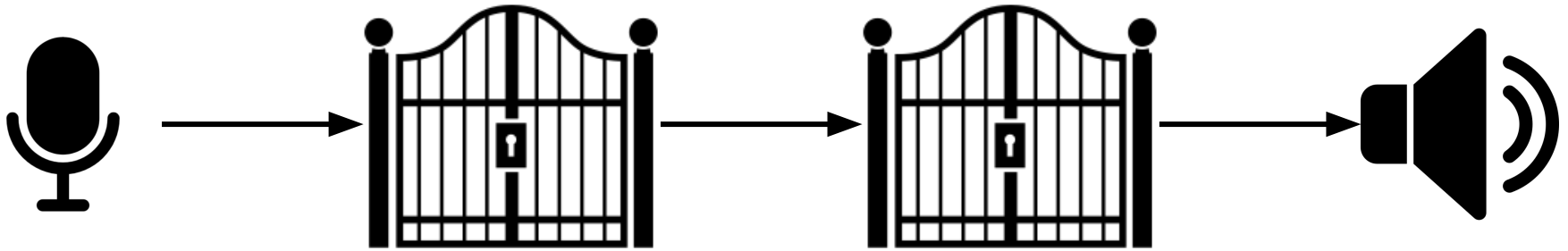
Functionality:

- Control of peripheral devices (mic/camera volume control, mute)
- Effects (background-blur, lighting adjustments, reactions)

Users Mental Model



Reality



I spoke but nobody could hear me.
I was not muted.

- Paying user

Definition: Upstream

For media:

- The HW is upstream of the OS.
- The OS is upstream of the UA.
- The UA is upstream of the Web app.

Or:

HW -> OS -> UA -> Web app

Ideally...

1. An upstream-mute is exposed to downstream-entities.
 - Downstream entities choose whether and how to reflect this to the user.
 - If some state is private, it's not exposed.
2. Downstream-entities can request-unmute from an upstream entity.
 - Subject to reasonable gating:
 - User gesture
 - Prompt by upstream entity
 - Rejection

Prioritization

Exposing upstream-state comes before changing it.

Generally speaking, if you don't know what the upstream state is, when will you ask to change it?

Dear user, please click here occassionally so we could try to unmute.

Concretely for mic-mute, if the Web app doesn't know the user muted the mic in the OS, how will it solicit a user's gesture to unmute?

The `mute` event

Would the existing `mute` event suffice? No.

Quote:

Muted refers to the input to the [MediaStreamTrack](#). If live samples are not made available to the [MediaStreamTrack](#) it is muted.

[Muted](#) is outside the control of web applications, but can be observed by the application by reading the [muted](#) attribute and listening to the associated events [mute](#) and [unmute](#). There can be several reasons for a [MediaStreamTrack](#) to be muted: the user pushing a physical mute button on the microphone, the user closing a laptop lid with an embedded camera, the user toggling a control in the operating system, the user clicking a mute button in the [User Agent](#) chrome, the [User Agent](#) (on behalf of the user) mutes, etc.

Proposal

```
enum MuteSource {"unspecified", "user-agent", "operating-system"};
```

```
interface MuteReason {  
    readonly MuteSource source;  
    readonly boolean potentiallyActionable; // Explanation follows.  
};
```

```
partial interface MediaStreamTrack {  
    sequence<MuteReason> getMuteReasons();  
};
```

Multiple Reasons

Note that the previous slide anticipated the possibility of multiple concurrent mute-reasons.

- A new `mute` event will be fired whenever the set of reasons changes, but is non-empty.
- A new `unmute` event will be fired whenever the set of reasons becomes empty.

Privacy Concerns

Would exposing any of this information be problematic privacy-wise?

The one concrete concern mentioned thus far - incoming calls.

Solution - provide little information:

```
source == "operating-system".
```

Worst-case scenario, the app shows suboptimal app-UX while the user is not looking, offering to unmute what cannot be muted.

Any other concerns?

requestUnmute()

Leaving the topic of state-exposure, let's briefly discuss state-control.

```
partial interface MediaStreamTrack {  
    Promise<undefined> requestUnmute();  
};
```

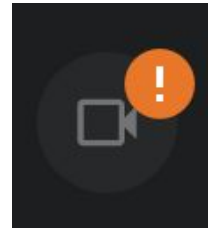
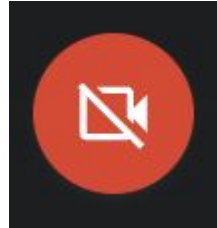
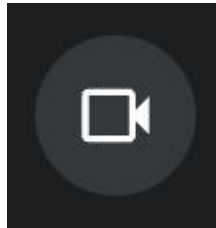
Requires user-gesture.

May result in a user-agent prompt, an operating-system prompt, or both.

What is potentially Actionable?

Web applications don't want to offer users impossible actions[*].

- If the device is unmuted - App-level UX #1.
- If calling requestUnmute() could work (with/without a prompt) - App-level UX #2.
- If it's guaranteed to fail - App-level UX #3.



[*] But it's acceptable, albeit suboptimal, to do so while the users aren't looking, as is likely the case when the users receive a phone call.

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 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 RTCDatChannel.

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 <code>jitterBufferTarget</code> , 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

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

```
1004     <tr id="N37">
1005         <td>N37</td>
1006         <td>It must be possible for the user agent's receive pipeline to process
1007 +       video at high resolution and framerate (e.g. by controlling hardware
1008 +       acceleration).</td>
1009     </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: 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

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/wolenez/media-source/mse-for-webcodecs-draft/media-source-respec.html#webcodecs-based>