

# **W3C WebRTC WG Meeting**

December 12, 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 second December 2023 interim meeting of the W3C WebRTC WG, at which we will cover:
  - Mediapture-screenshare, mediapture-extensions, keyframe API, dynamic switching, RtpSender encoded source
- [Future meetings:](#)
  - [January 16](#)
  - [February 20](#)
  - [March 19](#)

# About this Virtual Meeting



- Meeting info:
  - [https://www.w3.org/2011/04/webrtc/wiki/December\\_12\\_2023](https://www.w3.org/2011/04/webrtc/wiki/December_12_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>
- 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 [Mediacapture-Screenshare](#) (Elad)
- 08:30 - 08:50 AM [Mediacapture-Extensions](#) (Elad & Guido)
- 08:50 - 09:10 AM [Dynamic Switching](#) (Tove)
- 09:10 - 09:30 AM [RtpSender Encoded Source](#) (Guido)
- 09:30 - 09:50 AM [Keyframe API](#) (Harald)
- 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.



# **Mediacapture-screen-share (Elad)**

**Start Time: 08:10 AM**

**End Time: 08:30 AM**

# For Discussion Today

- MediaCapture-screen-share
  - [Issue 276](#): Handling of contradictory hints
  - [Issue 281](#): Distinguish cancellations from absent OS permissions
  - [Issue 219](#) (v2): Avoid user-confusion by not offering undesired audio sources #219

## Contradictory hints - example #1

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

## Contradictory hints - example #2

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

## Contradictory hints - example #3

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

# Handling of contradictory hints - proposal

---

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

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

## Proposal:

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

## Issue 281: Missing OS permissions - problem description

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

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

## Issue 281: Missing OS permissions - proposal

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

dictionary GetDisplayMediaNotAllowedErrorInit {
  required GetDisplayMediaNotAllowedReason reason;
};

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

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



## Issue 281: Possible objection

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

Answers:

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

## Issue 219: Don't prompt users to share unrequested audio sources

Recall:

- Some users have only partial understanding of the difference between the browser and the Web app.
- Absolutely not a single user can see the future.

We have previously specified systemAudio, which allows apps to say:

“I could use tab/window audio, but I would not use system audio.”

Result:

1. App doesn't want system audio.
2. Browser doesn't expose system audio toggle/checkbox/etc.
3. User doesn't end up ticking the checkbox to share system audio.
4. **User is not confused when system audio is not transmitted remotely.**

## Issue 219: What now?

“But Elad, this issue was closed 1.5 years ago. What now?”

Web applications have adopted `systemAudio` and are using it productively. There is now appetite for `windowAudio`.

```
enum WindowAudioPreferenceEnum {  
    "include",  
    "exclude"  
};  
  
dictionary DisplayMediaStreamConstraints {  
    WindowAudioPreferenceEnum windowAudio;  
};
```

## Issue 219: General Principle (Proposal)

Ask the user for the minimal set of permissions, training them to give permissions sparingly, and teaching them to view requests for excessive permissions as inherently suspect.

## Issue 219: Concern - nudges users towards sharing system audio?

### Concern:

Would users who are unable to share window audio be nudged to share their entire screen and system audio?

### Solution:

If `{windowAudio: "exclude"}` is specified, allow the call to `getDisplayMedia()` if `{systemAudio: "exclude"}`, and reject otherwise. (“Otherwise” being either “include” or unspecified.)

## Issue 219: Permitted permutations

<code>getDisplayMedia({video: true, <b>audio: true</b>});</code>	Browser offers {tab, window, screen} audio.
<code>getDisplayMedia({   video: true, audio: true,   "systemAudio": "exclude" });</code>	Browser offers {tab, window} audio.
<code>getDisplayMedia({   video: true, audio: true,   "<b>windowAudio</b>": "<b>exclude</b>",   "systemAudio": "exclude" });</code>	Browser offers {tab} audio.
<code>getDisplayMedia({video: true, <b>audio: false</b>});</code>	Browser offers {} audio.

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

-

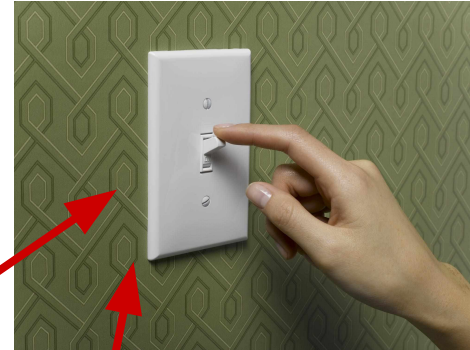
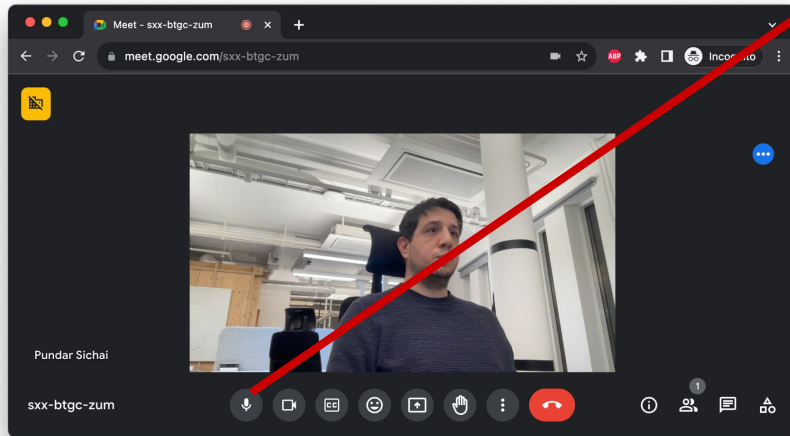
# **Mediacapture-Extensions (Elad & Guido)**

**Start Time: 08:30 AM**

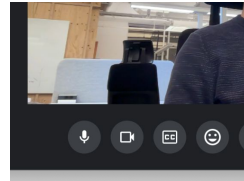
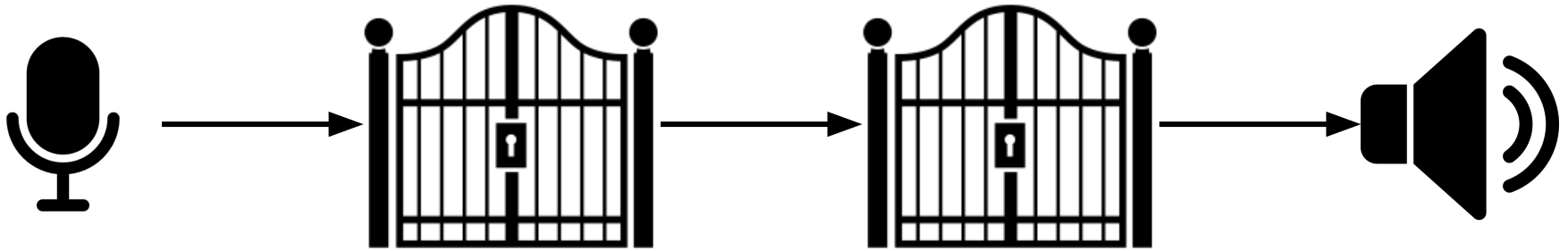
**End Time: 08:50 AM**



# Recap - Users' Mental Model



# Recap - Reality



## Of this issue's grave importance to users

- Video conferencing apps have hundreds of millions of daily active users.
- Often high-stake situation for these users.
  - Users with short time available to speak to distant loved ones
  - Users with social anxiety
  - Users in a job interview
  - Users pitching clients
  - Users presenting in WebRTC WG interims

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

- Paying user

# Of this issue's grave importance to developers

- We live in a competitive world.
- User experiences issues? Developer can't help? User changes provider!
  - Users don't care if the issue is with the hardware, the browser or the OS.
  - Problems with the browser? Users will use a native app.
  - Risk of loss of revenue - among the top developer concerns.
- Video conferencing providers have engineering teams.
  - Impacted by bug outside own codebase? Transparency needed!
  - Open source browser? Operating system? Ask own engineers to fix.
  - Closed source? File a clear bug on the correct entity (hardware, OS, browser).

# The Problems We Seek to Solve

## Problems:

1. Expose upstream state (isMuted, MuteReason, MediaSessionInfo)
2. Control upstream state - with a prompt (requestUnmute)

## Sequencing:

- The first problem **must** be solved first.
- Before an app can solicit a user gesture, it must know that it is muted.
  - It must also know if requestUnmute() may be called.
  - Otherwise, users would be frustrated.

# State Exposure - MuteReason

- Solves the problem
- Is backwards compatible
- Acknowledges the possibility of multiple concurrent reasons
  - Good for telemetry (for the Web app)
  - Integrates well with any shape of requestUnmute()
- Only viable solution (see next slides)

# State Exposure non-solution: Change definition of mute attribute

- Not backwards compatible
- No non-destructive way to migrate developers to the new definition.
  - Want to remove func()?
    - Issue deprecation warnings for a few revisions.
  - Want to change the mute attribute?
    - How do you warn developers?
    - How do you measure how many of current users of the mute attribute would be negatively impacted by a change of its definition?



# State Exposure non-solution: MediaSession-based APIs

- Unclear at the moment what is being proposed.
- Requires major modifications of the MediaSession spec.
  - No current support for reading the current state; only modifying it.
  - No current support for multiple devices.
  - What if muted for multiple reasons? User agent AND operating system?
  - Asynchronicity issues when trying to the onmute event.
  - State split between MediaStreamTrack and MediaSession - complex.
- Needlessly low-resolution.
  - UA-based muting? OS-based muting? Mic-gain set to 0%?
  - Why shouldn't the app know? (Apps might wish to inform users.)

# State Exposure-Control Cross-over

We have often discussed `requestUnmute()`. But what is the exact shape?

MuteReason really helps here. It acknowledges a key point, that multiple concurrent reasons are possible.

- Can be muted in the UA and the OS at the same time.
- Integrates well with the possibility of separate prompts.
  - Would `requestUnmute()` throw up two prompts?
  - MuteReason allows such API shapes as:
    - `requestUnmute("user-agent")`
    - `requestUnmute("operating-system")`
  - Not a requirement; we could still use `requestUnmute()`.

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

-

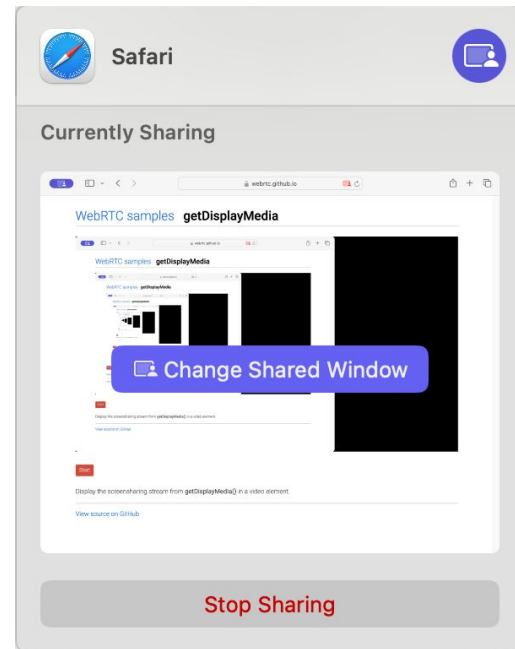
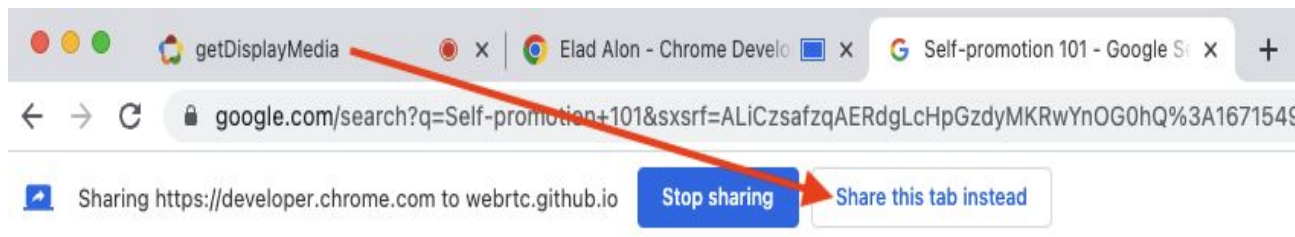
# **Dynamic Switching (Tove)**

**Start Time: 08:50 AM**

**End Time: 09:10 AM**

# Background

- Switching what is presented is common in video conferences, but starting and stopping presentations is cumbersome.
- Direct surface switching added by some vendors:
  - Chrome: Between tabs
  - Safari on MacOS: Between windows/screens



# How can we expand on this on the web?

Types of switching:

- **Same-type surface switching** (existing):
  - Between surfaces of the same type.
  - Can be done behind the scenes using an “injection model”
- **Different-type surface switching** (what we want!)
  - Cannot be done transparently:
    - Properties/methods are affected (e.g., capabilities)
    - Setup may be required (e.g., adding audio)

# Proposal: Switch-track model

- Let apps opt-in to a switch-track model for surface switching
- If opted-in (switch-track model):
  - Same- and different-type surface switching.
  - When switching occurs:
    - Old tracks are stopped.
    - An event is posted with a new mediastream.
- If not opted-in (use injection model):
  - Same-type surface switching only.
  - When switching occurs:
    - Old tracks are migrated to the new source transparently.
    - An event is posted with a new mediastream (if the app has registered an event listener).

# Proposal: Switch-track model API

```
getDisplayMedia({appAssistedSurfaceSwitching: "include", ...})  
controller.onsourceswitch = event => {  
  video.srcObject = event.stream;  
};
```



# State of discussions

Goal: Allow users to switch between different surfaces and add/remove audio

- General agreement that this functionality is of value

Recent points of the discussions:

1. Events/Callback - notification always?
  - Status: Agreement on events always

2. Explicit opt-in?

```
getDisplayMedia({appAssistedSurfaceSwitching: "include", ...})
```

- Status: Consensus forming in favor

3. Early/late decision-point?

- Status: Main remaining point of disagreement

# Counter-proposal: Late decision

- Surface switching (late decision/cancelable events):
  - Pause frame delivery from old sources and connected tracks
  - Post onsourceswitch event with a new stream for the new source-set
  - If preventDefault: stop old tracks, deliver frames to new stream
  - Otherwise: stop new tracks, resume frame delivery to old tracks
- Setup:
  - ```
controller.onsourceswitch = event => {  
  if (InjectionDesired(event))  
    return; // Use injection model  
  video.srcObject = event.stream;  
  event.preventDefault(); // Use switch-track model  
};
```

# Problems with late decision

1. Track migration is inconsistent and unclear
  - Audio tracks may be added/removed - video tracks are injected
  - Are audio track ever migrated?
  - Are old tracks migrated to the new source before or after the event?
2. Injection model with different-type switching:
  - Properties expected to be constant change, e.g., capabilities, methods
  - Added tracks are not handled
3. Code is not self-evident.
  - What does `preventDefault()` mean?
4. The late-decision model is more complex (it's a superset of the early-decision switch-track model)
  - When is this complexity needed?

# Discussion (End Time: 09:10)

-

# **RtpSender Encoded Source (Guido)**

**Start Time: 09:10 AM**

**End Time: 09:30 AM**

# Use Case

## § 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.

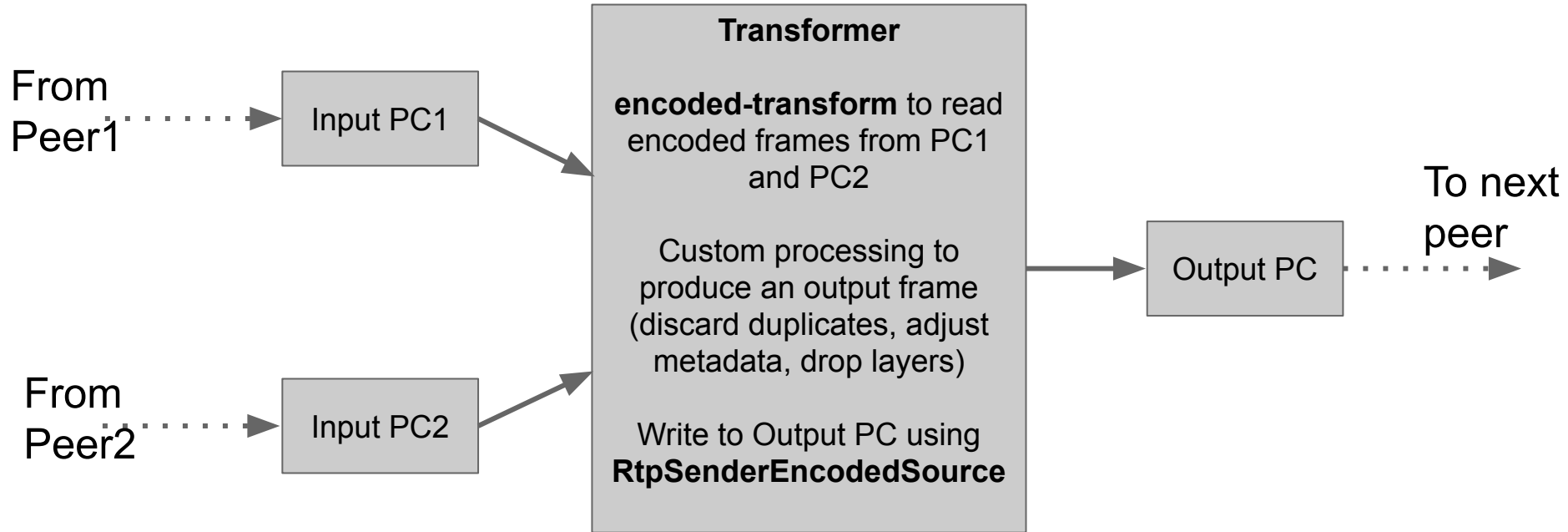
### NOTE

***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.

# Architecture for a node in a P2P network



- PC1 and PC2 provide the same media (might have different metadata)
- Output masks failures from either PC1 or PC2
- There might be more than 2 input peers and more 1 output peer

# Youenn's proposal (in [ET issue 211](#))

```
interface RTCRtpSenderEncodedSource {
  constructor()
  readonly attribute RTCRtpSenderEncodedSourceHandle handle;
  // Need congestion API, error API and enqueue API
  undefined enqueue((EncodedAudioChunk or EncodedVideoChunk) chunk, DOMString mimeType)
};
```

[Exposed=(DedicatedWorker, Window), Transferable]

```
interface RTCRtpSenderEncodedSourceHandle {}
```

```
partial interface RTCRtpSender {
  undefined replaceTrack(RTCRtpSenderEncodedSourceHandle handle);
  readonly attribute RTCRtpSenderEncodedSourceHandle encodedSourceHandle;
}
```



# Issues

- EncodedAudioChunk and EncodedVideoChunk seem appropriate for non-forwarding use cases
  - We also want to support those cases
  - Separate discussion
- We want to forward RTP Metadata too, not just payload
  - Applications need the metadata
  - Forwarders need the metadata
    - For example, drop layers to save bandwidth
- We want to adjust RTP Metadata in the output
  - For the same payloads, input PCs might have different metadata

# Extended proposal

- Allow the enqueue method to accept `RTCRtpEncodedVideoFrame` and `RTCRtpEncodedAudioFrame`
  - Allows forwarding metadata
- Allow sending a frame with updated metadata to the output PC
  - Provides a consistent metadata to the output PC
  - Makes it possible to mask failures of input PCs
- Not all metadata fields need to be updatable. For the forwarding case, the following should suffice:
  - `rtpTimestamp` / `timestamp`
  - `frameId`
  - `dependencies`

# Draft extended proposals

```
interface RTCRtpSenderEncodedSource {
    undefined enqueue((EncodedAudioChunk or EncodedVideoChunk) chunk, DOMString mimeType);
    undefined enqueue((RTCEncodedVideoFrame or RTCEncodedAudioFrame) frame);
};

// A: new constructor (only video shown, but also needed for audio)
interface RTCEncodedVideoFrame {
    constructor(RTCEncodedVideoFrame originalFrame,
                RTCEncodedVideoFrameMetadata newMetadata);
};

// B: new method, possibly used in combination with structured clone
interface RTCEncodedVideoFrame {
    // Updates eligible fields present in newMetadata, unset fields in newMetadata
    // leave existing fields unmodified
    setMetadata(RTCEncodedVideoFrameMetadata newMetadata);
};
```

# Discussion (End Time: 09:30)

- Do we have rough consensus on:
  - Introducing RTCRtpSenderEncodedSource
  - Support for feeding RTCEncodedVideoFrames and RTCEncodedAudioFrames
  - Support for some form of updating metadata for RTCEncodedVideoFrames/RTCEncodedAudioFrames

# Keyframe API (Harald)

**Start Time: 09:30 AM**

**End Time: 09:50 AM**

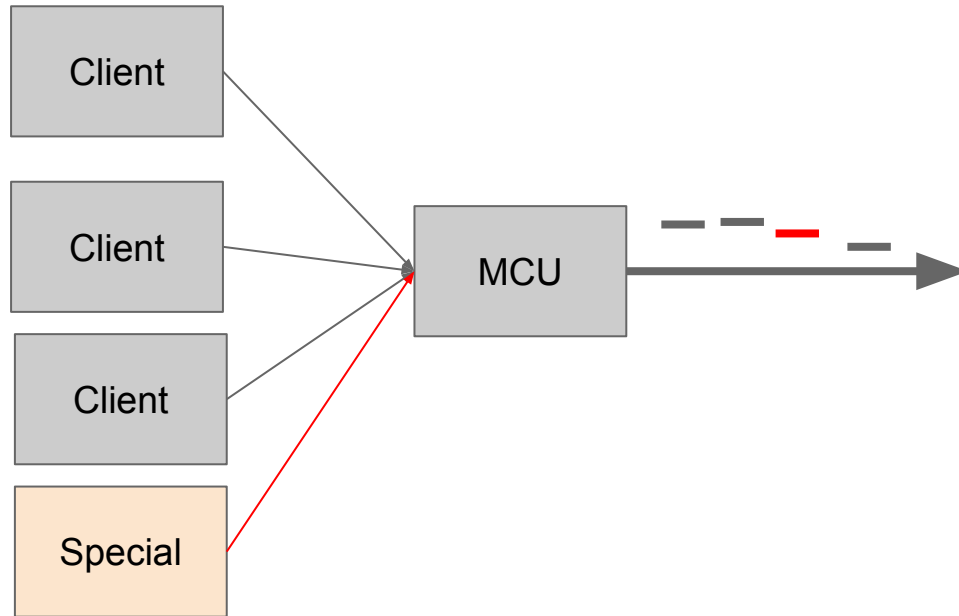
# Status of keyframe event proposal

- Still agreement that it is useful
- [#215](#) isolated keyframe event, but kept the mixins upstream and downstream
- Editors meeting agreed that keyframe event was good, but wanted to only use mixins when we have a second usage
- The edits were completed, and #215 is merged.

# SDP: Jan-Ivar and Harald have been talking

- Points of agreement:
  - Packetization mode is a new field inside codec description dictionary
  - Javascript code should deal with codec names, not PT numbers
  - There's a need for declaring what codecs a transform uses and produces (see next slides)
- Still discussing:
  - Codecs declared by app on transform, on transceiver, or both

# The Fan-In Use Case



- MCU forwards frames on a single SSRC
- Some are old format, some are new format
- Only new format needs transform



# Input filter on Transform

- Let RTCRtpTransform have an “input filter” parameter (a codec list)
- Only frames matching “input filter” get transformed
- Other frames are forwarded directly to sink
- Typical saving observed: 1 watt on a representative call.

# Discussion (**End Time: 09:50**)

-

# **Wrapup and Next Steps**

**Start Time: 09:50 AM**

**End Time: 10:00 AM**

# Next Steps

- Content goes here

# Thank you

Special thanks to:

WG Participants, Editors & Chairs