W3C WebRTC WG Meeting

May 16, 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 May 2023 interim meeting of the W3C WebRTC WG, at which we will cover:
  - Use Cases, WebRTC-Extensions, Mediacapture-Extensions, IceController, RtpTransport
- Future meetings:
  - June 27
  - July 18
  - September 19
  - October 17
  - November 21
  - December 12

About this Virtual Meeting

- Meeting info:
  - https://www.w3.org/2011/04/webrtc/wiki/May_16_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 

to get into the speaker queue.
- Click 

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 WebRTC Use Cases (Tim Panton)
- 08:30 - 09:10 AM Extensions (WebRTC & MediaCapture) (Henrik & Fippo)
- 09:10 - 09:30 AM IceController (Sameer & Peter)
- 09:30 - 09:50 AM RtpTransport (Peter)
- 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.

WebRTC-NV Use Cases (Tim Panton)

Start Time: 08:10 AM

End Time: 08:30 AM

Status of WebRTC-NV Use Cases

- New name: “WebRTC Extended Use Cases”
- 9 consensus use cases (Sections 2 and 3)
  - Some with non-consensus requirements
- 7 non-consensus use cases (Section 3)
- 31 open issues. Distribution:
  - 10 issues opened in 2023 (105+ days ago)
    - Opened during “Call for Consensus” (Jan - Feb 2023)
  - 3 issues opened in 2022
  - 18 issues open 2+ years

Questions:

- What do we do with non-consensus use cases that don’t make progress?
- What do we do with non-consensus requirements?
- What do we do with use cases with no requirements or proposals?

Use Cases with Consensus

- Section 2: Existing Use Cases (extensions to RFC 7478)
  - Section 2.1: Multiparty online game with voice communications
    - All requirements have consensus.
    - Relevant proposals: IceController, RtpTransport
  - Section 2.2: Mobile calling service
    - Relevant proposals: IceController, RtpTransport
    - Non-consensus requirements: N30, N31, N32 (re-establishment after a media interruption, parking of connections)
  - Section 2.3: Video Conferencing with a Central Server
    - All requirements have consensus.
    - Relevant proposals: WebRTC-SVC, RtpTransport

Use Cases with Consensus (cont'd)

- Section 3: New Use Cases
  - Section 3.1: File Sharing
    - All requirements have consensus.
    - Relevant proposals: RTCDataChannel in Workers
  - Section 3.3: Internet of Things
    - Non-consensus requirement: N33 (re-establishment of connections)
    - Relevant proposals: IceController
  - Section 3.5: Virtual Reality Gaming
    - All requirements have consensus.
    - Relevant proposals: RtpTransport
  - Section 3.6: Funny Hats
    - All requirements have consensus
    - Relevant proposals: RtpTransport, mediacapture-transform
  - Section 3.7: Machine learning
    - Same requirements as "Funny Hats" use case.
    - Text refers to proposals in other WGs (WebGPU, WebNN).
    - WEBRTC WG proposals focus on accelerating specific tasks (e.g. Face Detection, background blur), rather than "machine learning" in general.

Use Cases with Consensus (cont’d)

- Section 3: New Use Cases (cont’d)
  - Section 3.8: Don’t P0wn my Video Conferencing
    - Section 3.8.1: Untrusted Javascript Cloud Conferencing
      - Overlaps with Machine Learning use case requirements
      - Non-consensus requirements: N35 (group member re-forwarding)
      - Relevant proposals: Encoded Transform

Non-Consensus Use Cases (7)

- No consensus on Trusted Javascript Use Case (removed from Section 3.8)
- Section 3.2: Low Latency Streaming
  - Open Issue: 103
  - Section 3.2.1: Game Streaming
    - Non-consensus requirements: N37 (no copies), N38 (jitter buffer control).
    - Relevant proposals: WebRTC-Extensions (jitterbufferTarget), RtpTransport
    - Open Issue: 94 (game pad)
  - Section 3.2.2: Low Latency Broadcast with Fanout
    - Non-consensus requirements: N39 (media forwarding)
    - Relevant proposals: WebRTC-Extensions (jitterbufferTarget), RtpTransport
- Section 3.4: Decentralized Messaging
  - Non-consensus requirements: N34 (intercepting fetch API)
  - No relevant proposals

Non-Consensus Use Cases (cont'd)

- Section 3.9: Low Complexity Signaling
  - No requirements
  - No relevant proposals
- Section 3.10: One-Way Media
  - Open Issues: 93, 96, 102
  - Section 3.10.1: Live encoded non-WebRTC media
    - Non-consensus requirements: N40, N41, N42
    - Open Issues: 100, 104
  - Section 3.10.2: Transmitted stored encoded media
    - Non-consensus requirements: N41, N42, N43, N44
    - Open Issues: 101, 105
  - Section 3.10.3: Decoding pre-encoded media
    - Non-consensus requirements: N45, N46, N47
    - Open Issues: 106

WebRTC Extended Use Case Document

- Is it even useful?

  Probably yes since it has been cited recently
- Can it be improved?

  Almost certainly
- Looking for guidance+agreement on how

So I re-read it (and RFC 7478)

It is a bit unsatisfactory - some examples…

- RFC 7478 is oddly dated - it talks about telephony terminals ;-)
- Section 3.6 is a case in-point:
  “Funny-hats” isn’t a use-case, it is a new feature in an existing use-case

  But the requirements are valid/useful

  The resulting API point is popular …

  Far beyond the described use-case

- Section 3.9 is (arguably) already met by WISH/WHIP
  But has no consensus
- Overtaken by other standards and events

Use-case for the use-cases document

- What is this document for?
- Who are the intended readership?
- What will they do with it?
- How should it evolve?

So I put 2 (boring) hats on.

Hat 1:

WG member

I Want

- A to-do list
- A yard stick to gauge suitability of changes
- A progress meter
- A way to decouple scenarios from requirements
- A place to define direction/aspirations

Other ?

Hat 2:

Developer building on WebRTC

I Want

- Confidence that my API usage will continue to be supported
- A guide to the future direction of the standards (planning)
- A place to ask for new API features
- A way to know what is possible now

Other ?

Proposals

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

- Focus on things that can only be done by WebRTC (p2p etc)
- Remove use cases that are now met by other standards
- Include use cases that have no requirements but extend RFC 7478
- Remove use cases that don’t get consensus within a few months
- Remove requirements that don’t get consensus within a few months
- Remove use cases that don’t add new requirements
- Proposed API changes should include changes to the use-case doc
- Broaden the input somehow - perhaps from webrtc.nu ?

Discussion (End Time: 08:30)

[WebRTC/MediaCapture]-Extensions (Henrik & Fippo)

Start Time: 08:30 AM

End Time: 09:10 AM

For Discussion Today

- WebRTC-Extensions
  - Issue 134/PR 164: Remove JSEP Modifications (fippo)
  - Issue 158/PR 167: Requesting a key frame via setParameters (fippo)
  - Issue 159: RTCRtpEncodingParameters: scaleResolutionDownTo (Henrik)
- Mediacapture-Extensions
  - mediacapture-extensions#98 track.getFrameStats() allocates memory, adding to the GC pile (Henrik)

[Issue 134]/[PR 164]: Remove JSEP Modifications (fippo)

- Header extension API (Section 5.2) modifies RFC 8829 (JSEP)
  - Attempts to introduce JSEP references to WebRTC-PC internal slots
  - Endrun around the RFC Editor errata process
  - PR 164: Removes the JSEP modifications in Section 5.2.
- Attempting to make changes to draft-uberti-rtcweb-rfc8829bis instead
  - https://github.com/rtcweb-wg/jsep/pull/1033
    - Changes language from “for each *supported* RTP header extension” to “for each *enabled* RTP header extension”.
  - Discussion in progress on the IETF RTCWEB WG mailing list:
    - [rtcweb] JSEP-bis and WebRTC-Extensions Section 5.2 (ietf.org)
  - Changes may enable other potential extensions:
    - Issue 143: API for RTCP feedback mechanisms
- Thanks to Justin Uberti.

Issue 158/PR 167: When are Keyframes Generated? (Fippo)

- In WebRTC-PC, keyframe generation is an undocumented side effect. Examples:
  - `setParameters()` may cause key frames to be generated when:
    - Changing `scaleResolutionDownBy`
      - May not be necessary if the receiver supports resolution scaling!
    - Re-setting `active=true` after `active=false`.
  - `setParameters()` may ***not*** cause keyframes to be generated when:
    - Turning off the highest simulcast layer (e.g. setting `active=false`)
      - If SFU switches participants to a lower layer, without a keyframe they won't be able to decode it.
        - So when SFU notices that a layer is turned off, it sends an FIR to the encoder, causing a keyframe to be generated on ***all*** layers.
      - Bandwidth spike could be avoided if keyframe generation and layer activation could be controlled in a single (atomic) operation.

PR 167: Requesting a key frame via setParameters (Fippo)

- Allows the application to explicitly control keyframe generation (and layer activation) in a single `setParameters()` call.
- Semantics similar to RTCP FIR
  - “at the earliest opportunity. The evaluation of such an opportunity includes the current encoder coding strategy and the current available network resources”
- Supersedes encoded-transform#165
  - Make it more explicit and integrated
  - rids are checked automatically
  - Standalone API would often be called together with setParameters but could race.

Issue 159: RTCRtpEncodingParameters: scaleResolutionDownTo (Henrik)

- We all know and love `scaleResolutionDownBy`... it lets you do this:
  - Capture 720p and apply expensive video effects on the track.
    `Send {active, scaleResolutionDownBy:1 (720p)}`
    `+ {active, scaleResolutionDownBy:2 (360p)}` simulcast.
  - But then server tells us 720p is not needed, so we
    `Send {inactive}`
    `+ {active:360p}`.
- But… why are we applying expensive video effects on a 720p track if we're only sending it in 360p?
  - So we `track.applyConstraints()` to 1/2 downscale the track and `setParameters()` to 2x the scaling factors to counter smaller frame:
    `Send {inactive}`
    `+ {active, scaleResolutionDownBy:1 (still 360p!)}`.

Issue 159: RTCRtpEncodingParameters: scaleResolutionDownTo (cont'd)

- Problem: This is racy.
  - The track changing size and the parameters updating the scaling factors are not in-sync, so you might send 720p on the VGA layers for a few frames.
  - You're wasting a keyframe sending the wrong resolution and then you're wasting another keyframe to adjust back to the desired 360p resolution.
  - Working around this by temporarily inactivating all encodings while the track is resizing will create a glitch on the receiver and there would be more keyframes when the encodings are re-activated.
  - The API was not designed for this.
- Proposal
  - Add "`scaleResolutionDownTo: 360`" API.
  - Equivalent to always having the correct scaling factor.
  - In the above example, changing the track's size from 720p to 360p would not result in reconfiguring the encoder and there would be no key frames or races.
  - Something like this already exists in libwebrtc (RtpEncodingParameters::requested_resolution) so it might be fairly straight-forward to experiment with this in JavaScript.

mediacapture-extensions#98: track.getFrameStats() allocates memory, adding to the GC pile (Henrik)

Track stats API = expose metrics from **capture process** to **JS thread**.

The metrics are updated for every audio frame or video frame.

Original problem: In real-time JS apps, excessive dictionary creation adding to GC pile may be bad for performance.

What should be the API shape? Async or sync? Dictionary or interface?

1. Promise returning dictionary.
   `await track.getStats()`
2. Synchronous getter of an interface.
   `track.audioStats` / `track.videoStats`

mediacapture-extensions#98: track.getFrameStats() allocates memory, adding to the GC pile (cont'd)

Is GC a problem?

- The intended use case polls stats at 1 Hz.
- MediaStreamTrack is not accessible from real-time threads, so real-time pushes shouldn't be a requirement.
- GC nurseries can deal with temporary objects.

Main question: to `await` or not to `await`?

- Ergonomics
- Performance

Dictionary or interface can be done either way.

mediacapture-extensions#98: track.getFrameStats() allocates memory, adding to the GC pile (cont'd)

Async API queues a task when the app requests stats.

Sync API forces UA to continuously queue tasks to update internal slots - even if app never requests stats.

- We're talking 30-100 times per second *per track**.

* So far **only local capture tracks** are being considered, limiting overhead to:

- E.g. 1 audio (100 Hz) + 1 video (30 fps) track = 130 tasks/s

mediacapture-extensions#98: track.getFrameStats() allocates memory, adding to the GC pile (cont'd)

Async API queues a task when the app requests stats.

Sync API forces UA to continuously queue tasks to update internal slots - even if app never requests stats.

- We're talking 30-100 times per second *per track**.

* So far **only local capture tracks** are being considered, limiting overhead to:

- E.g. 1 audio (100 Hz) + 1 video (30 fps) track = 130 tasks/s

Not being proposed today, but what we if want **remote track stats** in the future?

- 56 tracks are common
    - 1 local audio, 1 local video, 4 remote audio, 50 remote video.
    - Fearing 5*100+51*30 = 2030 tasks/s.
    - Fearing 2030 IPC/s if grabbing stats from other process.

mediacapture-extensions#98: track.getFrameStats() (cont’d)

Problems with sync API

Problem 1: Excessive tasks posted for apps only occasionally interested in stats.

Problem 2: Cross-process metric collection => Unnecessary IPC?

mediacapture-extensions#98: track.getFrameStats() (cont'd)

Problems with sync API

Problem 1: Excessive tasks posted for apps only occasionally interested in stats.

Ways to avoid: Mutex, only grab lock when requesting stats.

Should work, but we'll have to implement a cache cleared in the next task.

Problem 2: Cross-process metric collection => Unnecessary IPC?

mediacapture-extensions#98: track.getFrameStats() (cont'd)

Problems with sync API

Problem 1: Excessive tasks posted for apps only occasionally interested in stats.

> **Ways to avoid:** Mutex, only grab lock when requesting stats.
>
> Should work, but we'll have to implement a cache cleared in the next task.

Problem 2: Cross-process metric collection => Unnecessary IPC?

> **Ways to avoid:** Piggyback on existing IPC messages.
>
> **Problem:** Assumes IPC happens anyway. That's an implementation detail.
>
> Lost optimization potential: what if neither source or sink lives in the renderer?

mediacapture-extensions#98: track.getFrameStats() (cont'd)

Async or sync?

There can only be one!

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

mediacapture-extensions#98: track.getFrameStats() (cont’d)

Proposal A: Async + dictionary

```
dictionary MediaStreamTrackStats { … }
partial interface MediaStreamTrack {
  Promise<MediaStreamTrackStats> getStats();
}
```

Proposal B: Sync + interface

```
interface MediaStreamAudioTrackStats { … }
interface MediaStreamVideoTrackStats { … }
partial interface MediaStreamTrack {
  [SameObject] readonly attribute MediaStreamAudioTrackStats audioStats;
  [SameObject] readonly attribute MediaStreamVideoTrackStats videoStats;
}
```

Proposal C: Sync + interface + snapshots

- Same as B, but return latest snapshot that SHOULD be recent.
- E.g. update at 10 Hz or batch-update all tracks with a single 1 IPC.

Discussion (End Time: 09:10)

-

IceController (Sameer & Peter)

Start Time: 09:10 AM

End Time: 09:30 AM

WebRTC ICE incremental improvements

- Prevent removal of candidate pairs - **Issue 166**
- Remove candidate pairs
- Control selection of candidate pair
- (?) Observe candidate pair states
- Observe result/RTT of outgoing checks
- Control frequency of outgoing checks of particular candidate pairs
- Prevent outgoing checks of particular candidate pairs
- Control order and timing of outgoing checks
- Observe presence of not of incoming checks or media for particular candidate pairs
- Gather local candidates for new network interfaces
- Re-gather local candidates of previously failed network interfaces
- Prevent removal of local candidates
- Remove local candidates
- Construct IceTransport without PeerConnection
- Support forking

Issue 166 - prevent candidate pair removal

Use case: connection redundancy

Keep one or more backup connections around for if/when the active connection deteriorates

Extend with further improvements - switch to a backup connection without an ICE restart or waiting for an ICE disconnect

Issue 166 - prevent candidate pair removal

Options

- Cancelable event before candidate pair removal
- Change automatic behaviour with a candidate pair attribute

With either option, user agent keeps existing behaviour unless the application requests otherwise. Difference is in the *how*.

Issue 166 - prevent candidate pair removal

Cancelable event

- Fire an event when ICE agent has determined a pair to remove, but before removal
- Application can prevent removal by cancelling the event
- Similar to touch and form submit events

```
partial interface RTCIceTransport  {
    attribute EventHandler oncandidatepairremoval;
};

interface RTCIceCandidatePairEvent : Event {
    readonly attribute RTCIceCandidatePair candidatePair;
};
```

```
transceiver.sender.transport.iceTransport
  .oncandidatepairremoval = (e) => {
    if (e.candidatePair.local.type === 'relay') {
        e.preventDefault();
    }
    // else e.candidatePair gets removed
};
```

Issue 166 - prevent candidate pair removal

Change automatic behaviour - `removable` attribute

- Application can prevent removable by setting the attribute when candidate pair first added or later

```
partial interface RTCIceTransport  {
    attribute EventHandler oncandidatepairadded;
};

interface RTCIceCandidatePairEvent : Event {
    readonly attribute RTCIceCandidatePair candidatePair;
};

partial interface RTCIceCandidatePair {
    attribute boolean removable;
};
```

```
transceiver.sender.transport.iceTransport
  .oncandidatepairadded = (e) => {
    if (e.candidatePair.local.type === 'relay') {
        e.candidatePair.removable = false;
    }
};
```

Issue 166 - prevent candidate pair removal

Change automatic behaviour - `idleTimeout` attribute

- Application can delay or prevent removal *(or even hasten it)* by setting the attribute when candidate pair first added or later
- ICE agent can remove pair after `idleTimeout` duration has elapsed without an ICE check or data sent / received

```
partial interface RTCIceTransport  {
    attribute EventHandler oncandidatepairadded;
};

interface RTCIceCandidatePairEvent : Event {
    readonly attribute RTCIceCandidatePair candidatePair;
};

partial interface RTCIceCandidatePair {
    attribute unsigned long idleTimeout;
};
```

```
transceiver.sender.transport.iceTransport
  .oncandidatepairadded = (e) => {
    if (e.candidatePair.local.type === 'relay') {
        e.candidatePair.idleTimeout =
            Number.MAX_SAFE_INTEGER;
    }
};
```

Discussion

- Another approach?
- Cancelable event or settable attribute
  - possible to have different strategies for different events
  - eg. `idleTimeout` may be more suitable for candidate pair removal, leaving lifecycle management up to the ICE agent
  - cancelable event for ICE checks and switching pairs
- If attribute, which one - `removable` or `idleTimeout`
- Setter method that may fail instead of attribute
- Skip to another improvement altogether?

Discussion (End Time: 09:30)

-

RtpTransport Follow-up (Peter)

Start Time: 09:30 AM

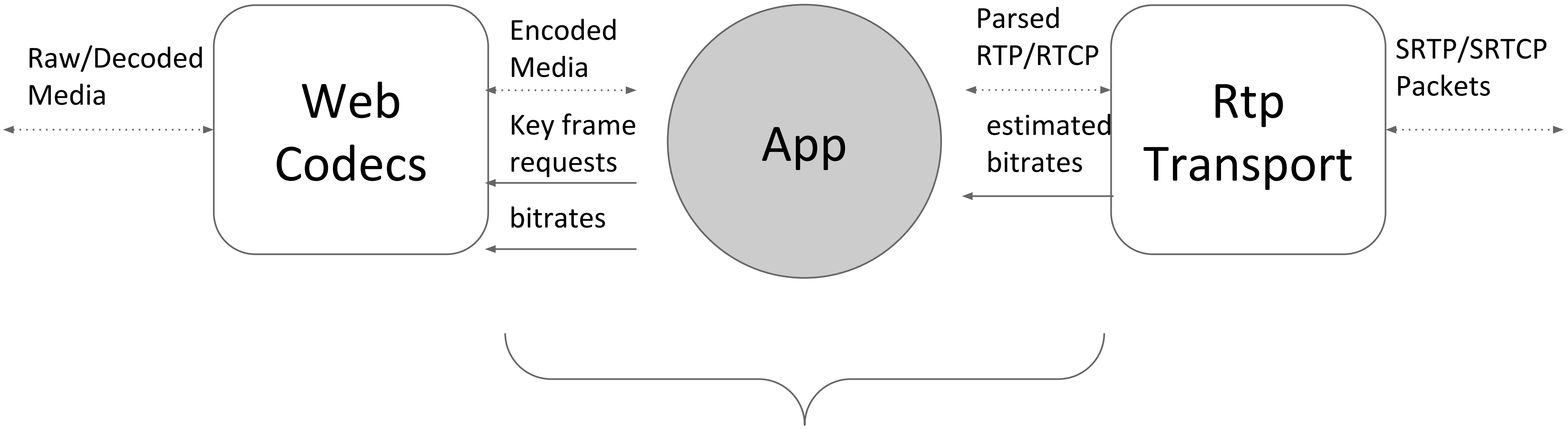
End Time: 09:50 AM

RtpTransport Follow-up

- What are the use cases?
- Is the gap between WebCodecs and RtpTransport too big for the app developer?
- Can you provide examples instead of WebIDL?

Use cases

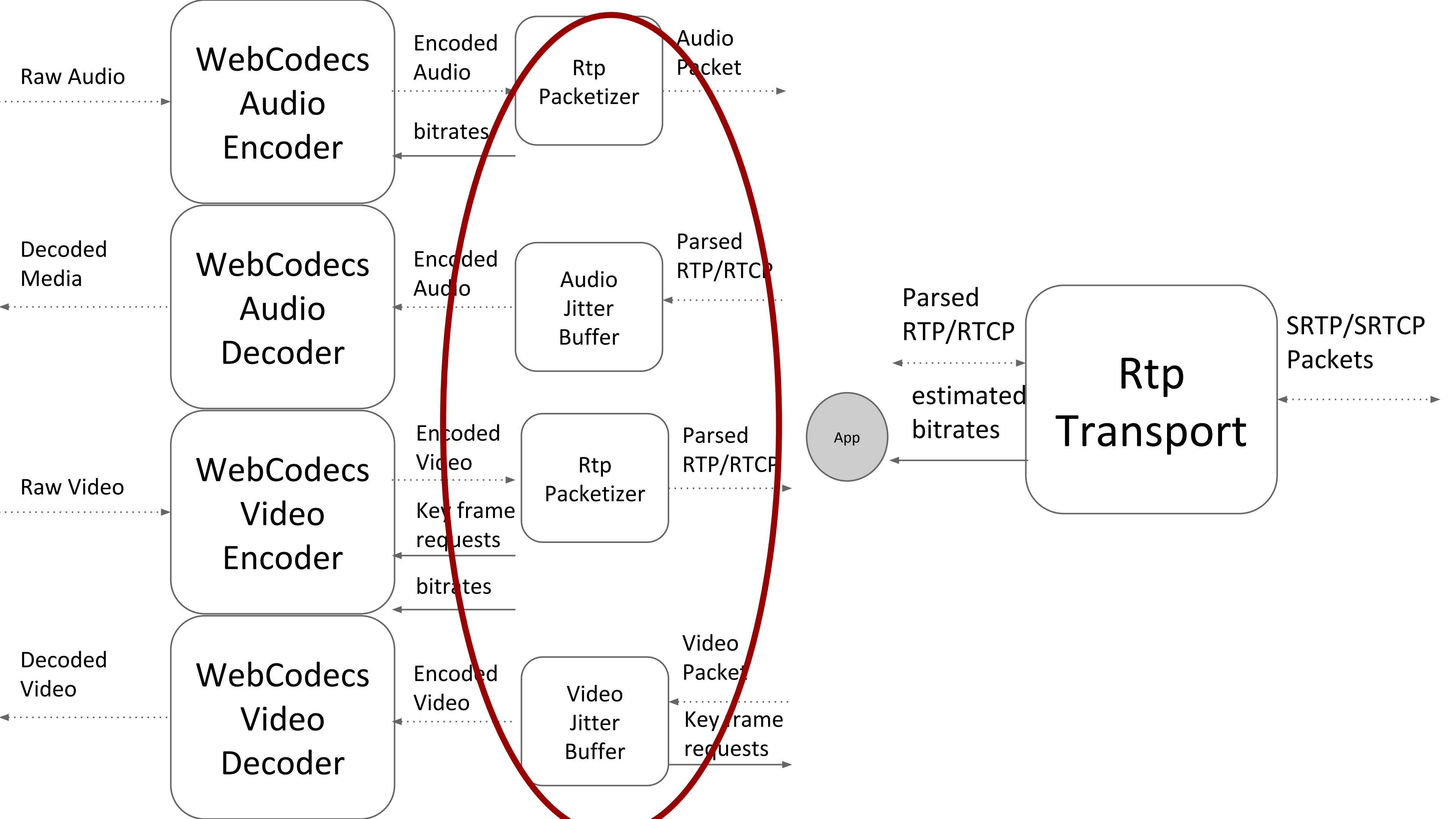
- Custom data along with audio and video (eg. 3D avatar)
- BYO packetization with new codecs (eg. WebCodecs HEVC)
- BYO packetization with existing codecs (eg. WebCodecs H264)
- Low-level control of codecs (eg. like WebCodecs)
- Control when key frames and layer refreshes are generated
- Support new RTCP messages (LRR)
- BYO codec (audio w/ WASM or WebGPU)
- BYO Bitrate Allocation + FEC + RED + RTX + Jitter Buffer
- Better interop with existing endpoints (custom RTCP; RTP data)



Is this gap too big?

Some things in the gap

- RTP Packetization
- RTP Depacketization
- Jitter Buffer


Raw Audio
WebCodecs Audio Encoder
Encoded Audio
bitrates
Rtp Packetizer
Audio Packet
Decoded Media
WebCodecs Audio Decoder
Encoded Audio
Audio Jitter Buffer
Parsed RTP/RTCP
Raw Video
WebCodecs Video Encoder
Encoded Video
Key frame requests
bitrates
Rtp Packetizer
Parsed RTP/RTCP
Decoded Video
WebCodecs Video Decoder
Encoded Video
Video Jitter Buffer
Video Packet
Key frame requests
App
Parsed RTP/RTCP
estimated bitrates
Rtp Transport
SRTP/SRTCP Packets

RtpPacketizer

```
let transport = new RtpTransport(...);
let packetizer = new RtpPacketizer("vp8", …);
let frame = … encoded using WebCodecs …;
let packets = packetizer.packetize(frame);
for (let rtpPacket of packets) {
  transport.sendRtpPacket(rtpPacket);
}
```

VideoJitterBuffer

```
let transport = new RtpTransport(...);
let vBuffer = new VideoJitterBuffer(...);
transport.onrtppacket = (rtpPacket) => {
  vBuffer.insertPacket(rtpPacket);
}
vBuffer.onframe = (frame) => {
  … decode using web codecs …
}
```

AudioJitterBuffer

```
let transport = new RtpTransport(...);
let aBuffer = new AudioJitterBuffer(...);
transport.onrtppacket = (rtpPacket) => {
  aBuffer.insertPacket(rtpPacket);
}
// TODO: Support cross-worker mechanism
render(aBuffer.track);
```

Other things in the gap

- RTX
- FEX
- RED

Other examples (custom data)

```
let transport = new RtpTransport(...);
let customDataPayloadType = 126;
let customDataSsrc = …;
let customDataPayload = …;
transport.sendRtpPacket({
  payloadType: customDataPayloadType = 126,
  ssrc: customDataSsrc,
  …
  payload: customDataPayload,
});
```

Other examples (BYO packetization)

```
let transport = new RtpTransport(...);
let hevcDepacketizer = … custom thing …;
let hevcDecoder = … from WebCodecs …;
transport.onrtppacket = (rtpPacket) => {
  let frame = h264Depacketizer.insertPacket(rtpPacket);
  if (frame) {
    hevcDecoder.decode(frame);
  }
};
```

Discussion (End Time: 09:50)

- Do we need more use cases?
- Do we need more examples? Of what?
- Should we fill the gap or leave it to a JS library?
- Where do we go from here? Write an explainer?
-

Name that Bird

Thank you

Special thanks to:

WG Participants, Editors & Chairs