W3C WebRTC WG Meeting

December 7, 2022
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 December 2022 interim meeting of the W3C WebRTC WG, at which we will cover:
    - Stuff
- Future meetings:
    - January 17
    - February 21
    - March 21
    - April 18
    - May 16
    - June 20

About this Virtual Meeting

- Meeting info:
  - https://www.w3.org/2011/04/webrtc/wiki/December_7_2022
- 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

- **Type +q and -q in the Google Meet chat to get into and out of the speaker queue.**
- **Please use headphones when speaking to avoid echo.**
- **Please wait for microphone access to be granted before speaking.**
- **Please state your full name before speaking.**
- **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 WebRTC Network of User’s Project (Tim Panton)
- 08:30 - 08:50 WebRTC-NV Use Cases (Bernard)
- 08:50 - 09:45 Encoded-Transform (Harald & Fippo)
- 09:45 - 10:00 WebRTC-PC (Jan-Ivar)

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.

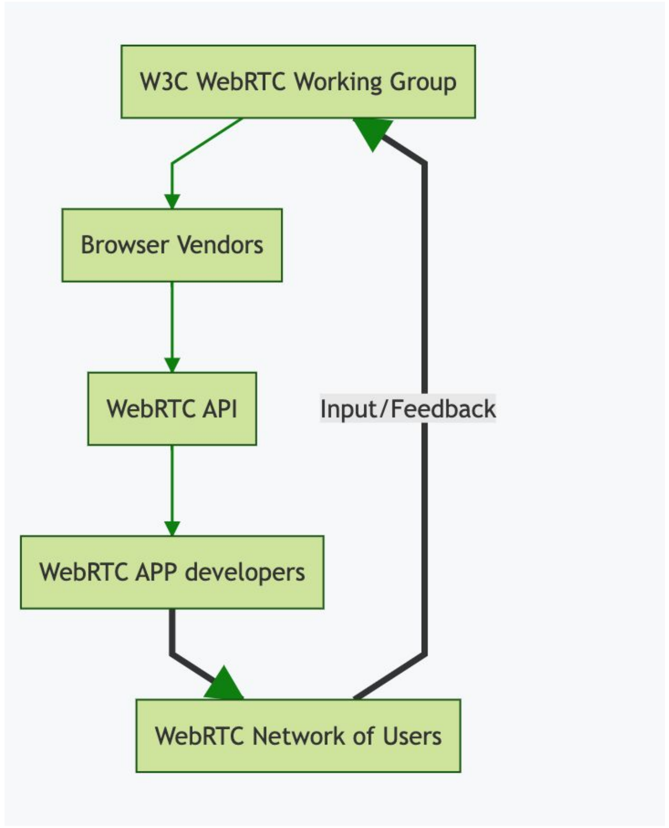
WebRTC Network of Users Project (Tim)

Start Time: 08:10 AM

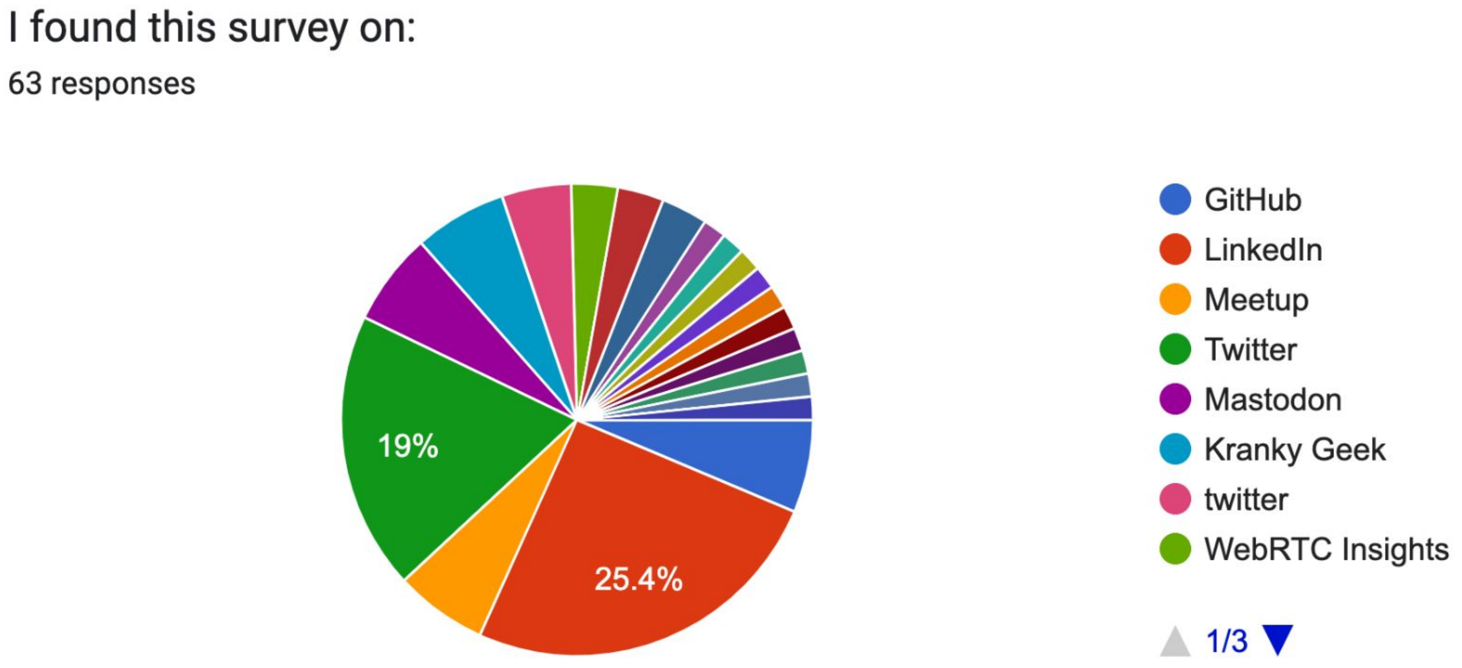
End Time: 08:30 AM

https://Webrtc.nu

- Feedback path from developers to WG
- 12 invited members to help guide it
- First activity is a survey of unresolved questions from here.



Survey results - who replied?



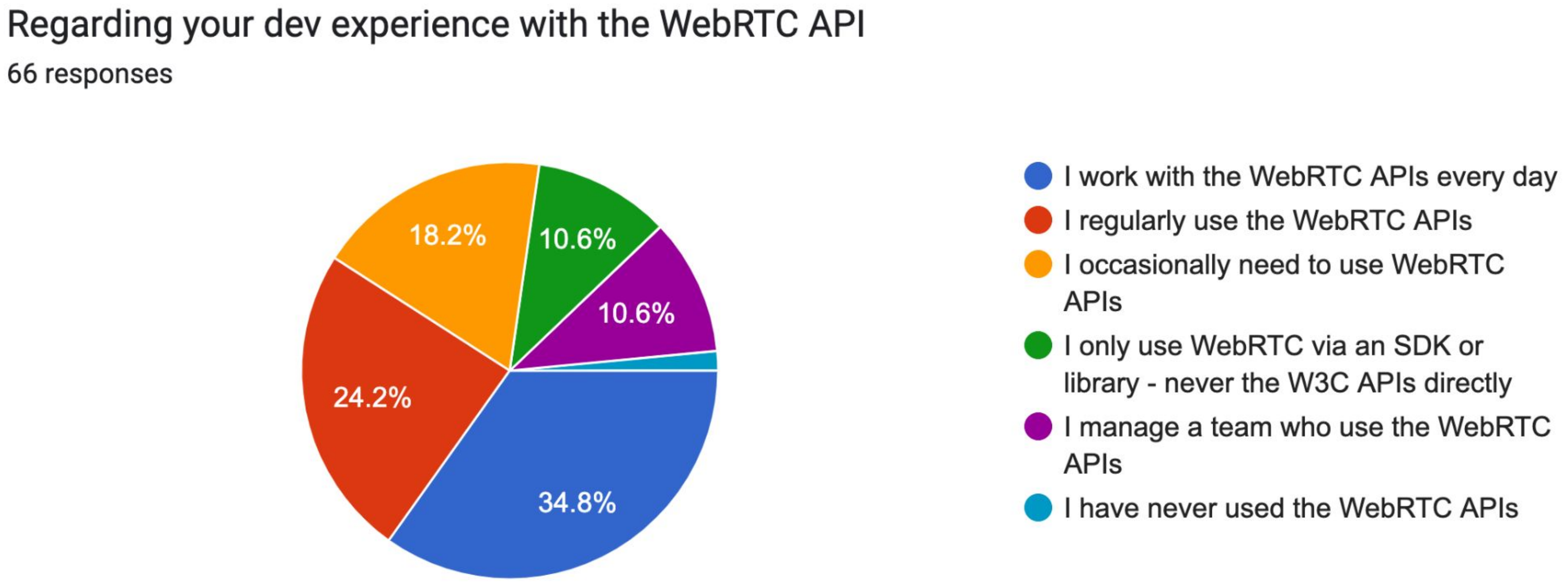
I found this survey on:

63 responses

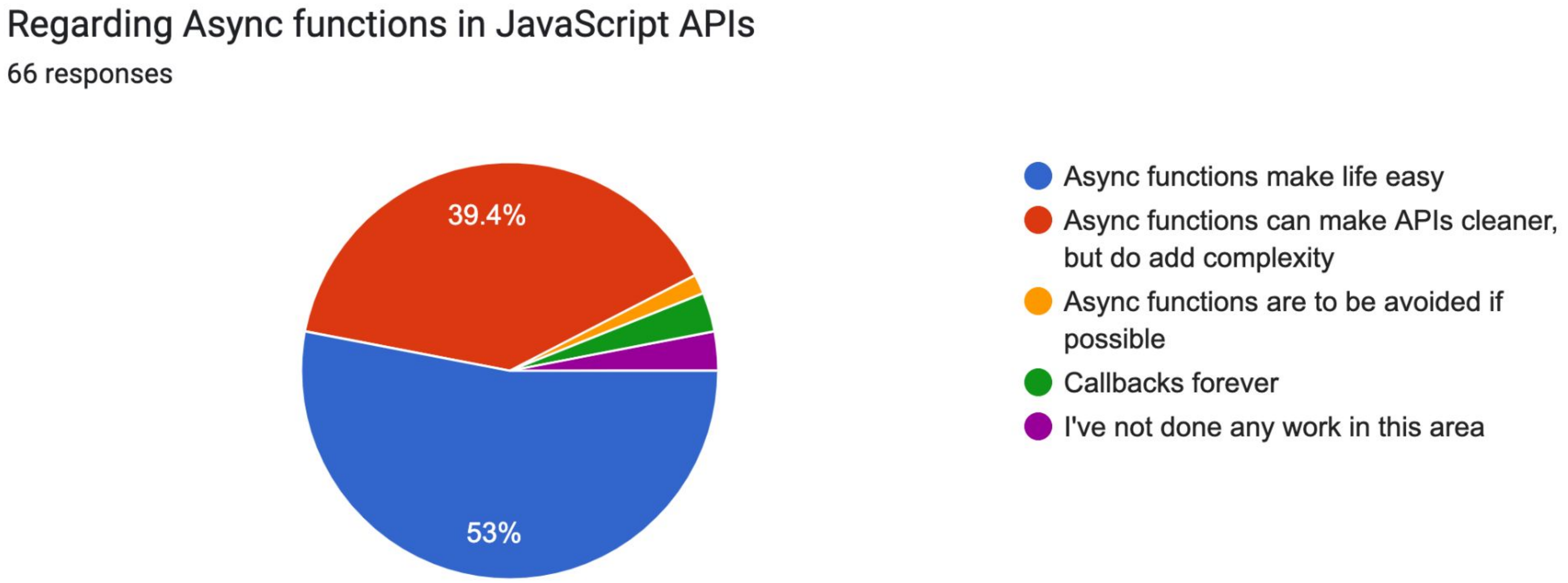
- ● GitHub
- ● LinkedIn
- ● Meetup
- ● Twitter
- ● Mastodon
- ● Kranky Geek
- ● twitter
- ● WebRTC Insights

▲ 1/3 ▼

Survey results - what do they do?



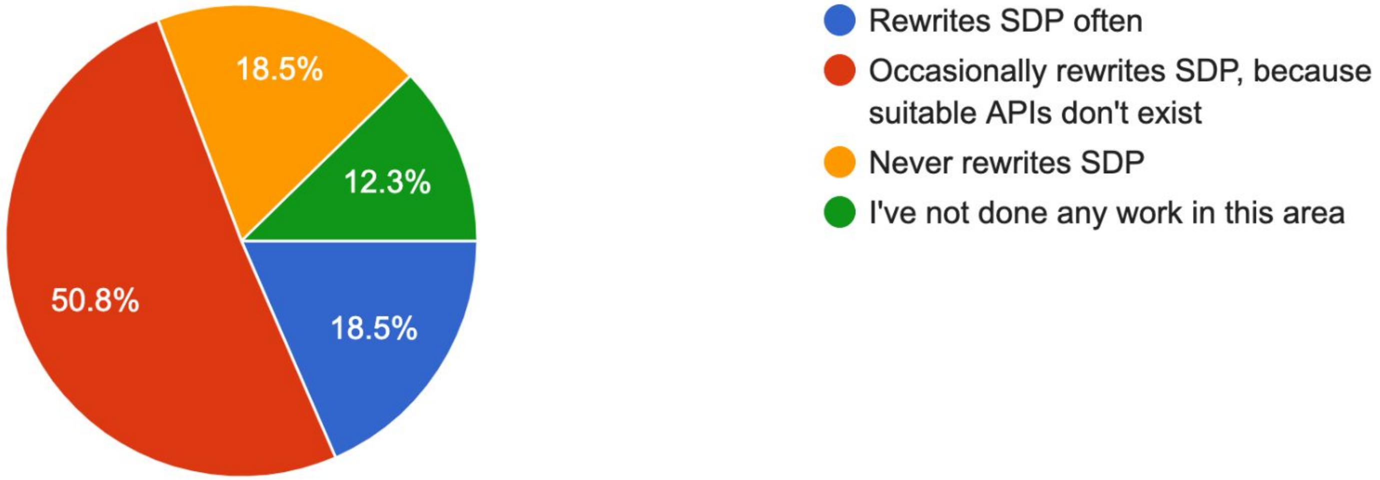
Survey results - feelings about async?



Survey results - do they even munge?

Regarding SDP munging, your WebRTC (in-browser) JavaScript:

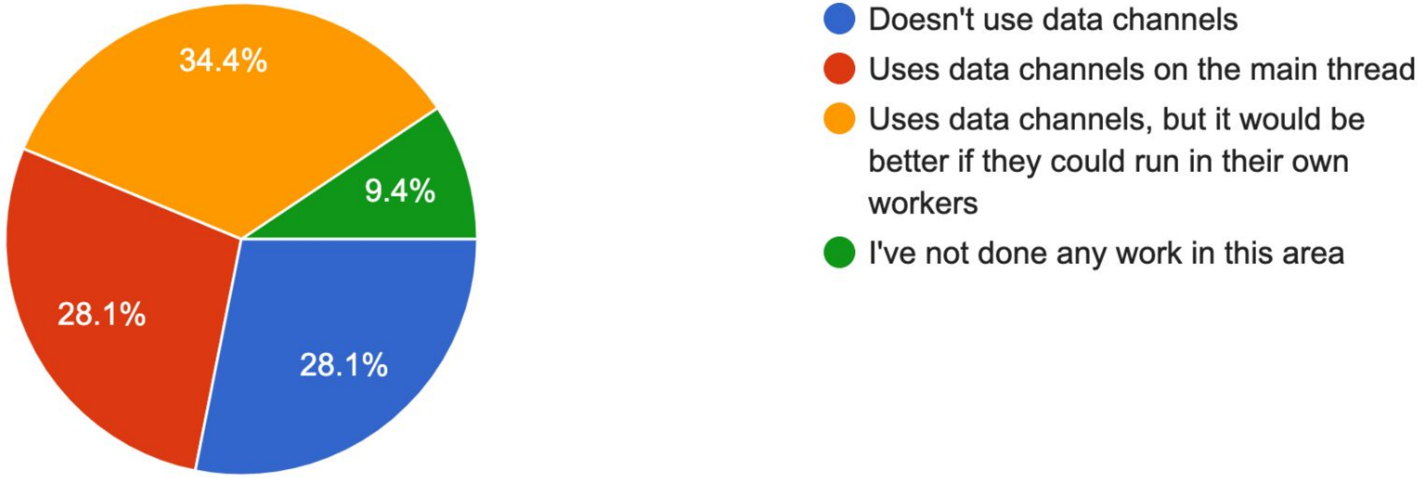
65 responses



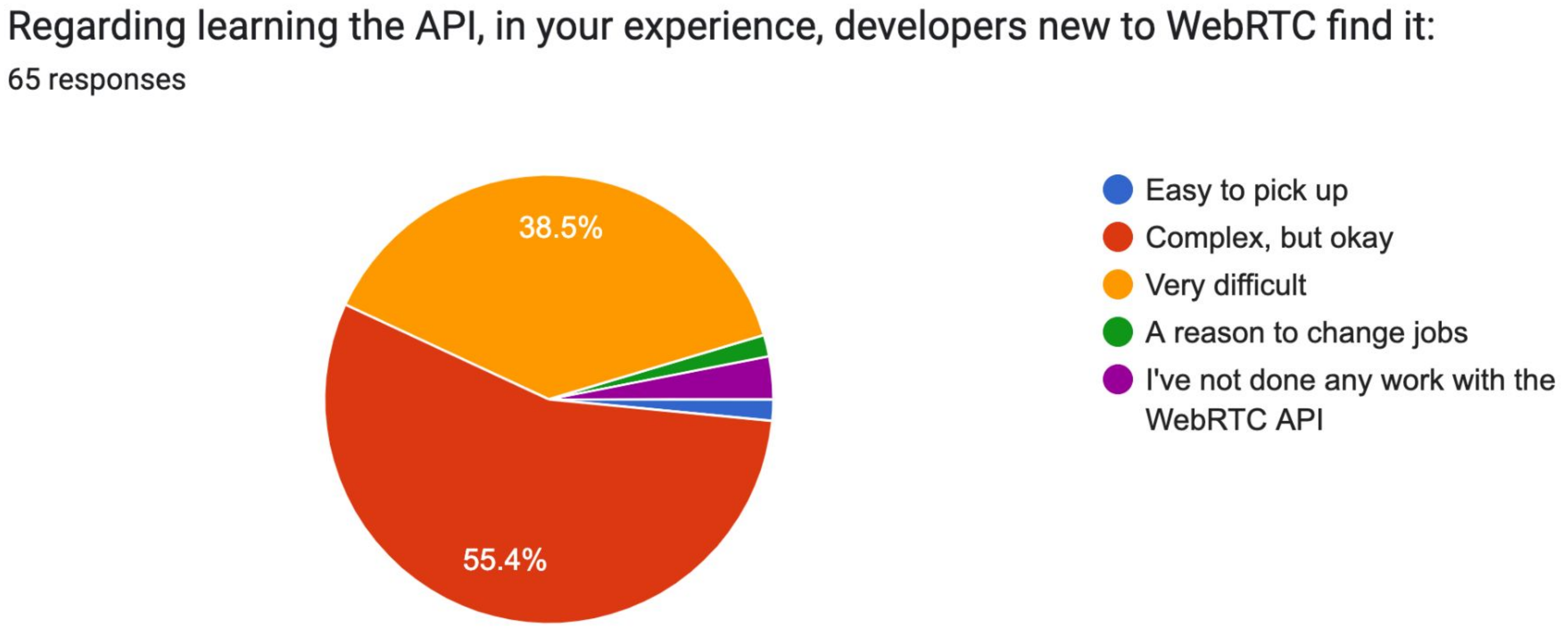
Survey results - datachannel usage

Regarding Data channels in Web Workers, your WebRTC application or SDK:

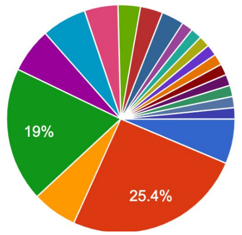
64 responses



Survey results - how hard is this to learn?



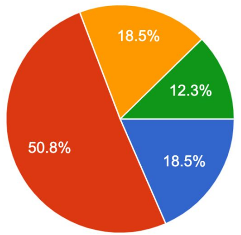
I found this survey on:
63 responses



- ● GitHub
- ● LinkedIn
- ● Meetup
- ● Twitter
- ● Mastodon
- ● Kranky Geek
- ● twitter
- ● WebRTC Insights

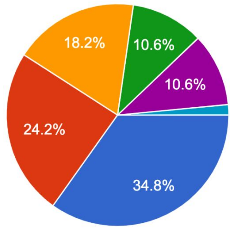
▲ 1/3 ▼

Regarding SDP munging, your WebRTC (in-browser) JavaScript:
65 responses



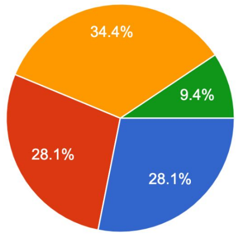
- ● Rewrites SDP often
- ● Occasionally rewrites SDP, because suitable APIs don't exist
- ● Never rewrites SDP
- ● I've not done any work in this area

Regarding your dev experience with the WebRTC API
66 responses



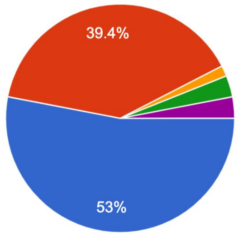
- ● I work with the WebRTC APIs every day
- ● I regularly use the WebRTC APIs
- ● I occasionally need to use WebRTC APIs
- ● I only use WebRTC via an SDK or library - never the W3C APIs directly
- ● I manage a team who use the WebRTC APIs
- ● I have never used the WebRTC APIs

Regarding Data channels in Web Workers, your WebRTC application or SDK:
64 responses



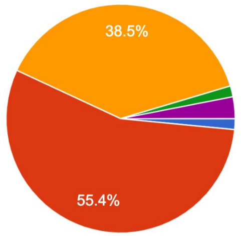
- ● Doesn't use data channels
- ● Uses data channels on the main thread
- ● Uses data channels, but it would be better if they could run in their own workers
- ● I've not done any work in this area

Regarding Async functions in JavaScript APIs
66 responses



- ● Async functions make life easy
- ● Async functions can make APIs cleaner, but do add complexity
- ● Async functions are to be avoided if possible
- ● Callbacks forever
- ● I've not done any work in this area

Regarding learning the API, in your experience, developers new to WebRTC find it:
65 responses



- ● Easy to pick up
- ● Complex, but okay
- ● Very difficult
- ● A reason to change jobs
- ● I've not done any work with the WebRTC API

Survey: was that useful?

- Did we learn anything?
- Do we have more questions?

Tip of the hat to **Patrick Rockhill**

Next (End Time: 08:30)

- More surveys
- More outreach
- Prototyping API ideas
- Finding good stuff from other WebRTC APIs

WebRTC-NV Use Cases (Bernard)

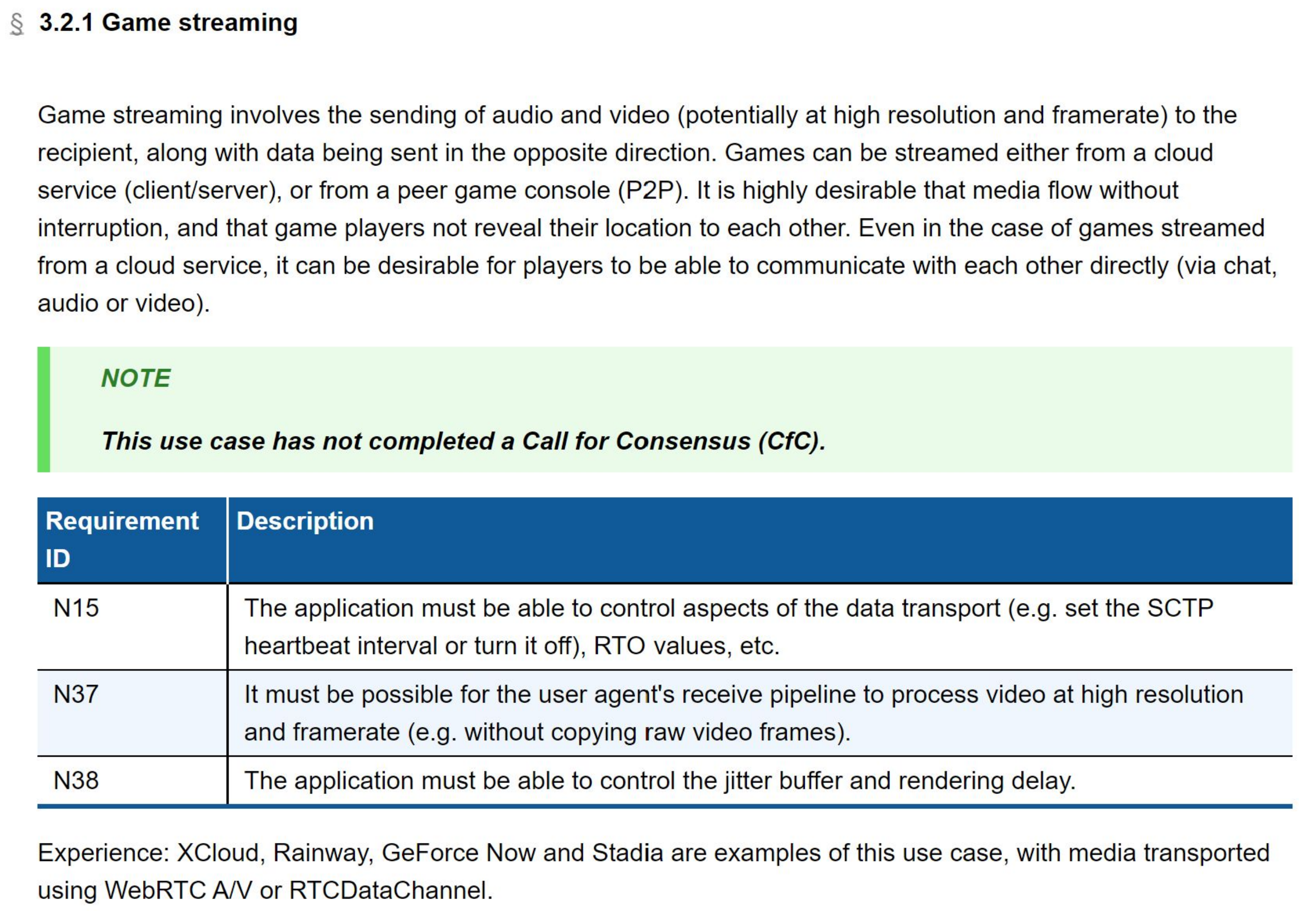
Start Time: 08:30 AM

End Time: 08:50 AM

For Discussion Today

- Use Cases (going to CfC in January)
  - Section 3.2.1: Game streaming
  - Section 3.2.2: Low latency Broadcast with Fanout
  - Section 3.5: Virtual reality gaming
- Issues (Harald)
  - Issue 81: Transport pre-encoded live content over RTP
  - Issue 82: Transmit stored pre-encoded content over RTP
  - Issue 106: “One-ended” Use Cases

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 not completed a Call for Consensus (CfC).

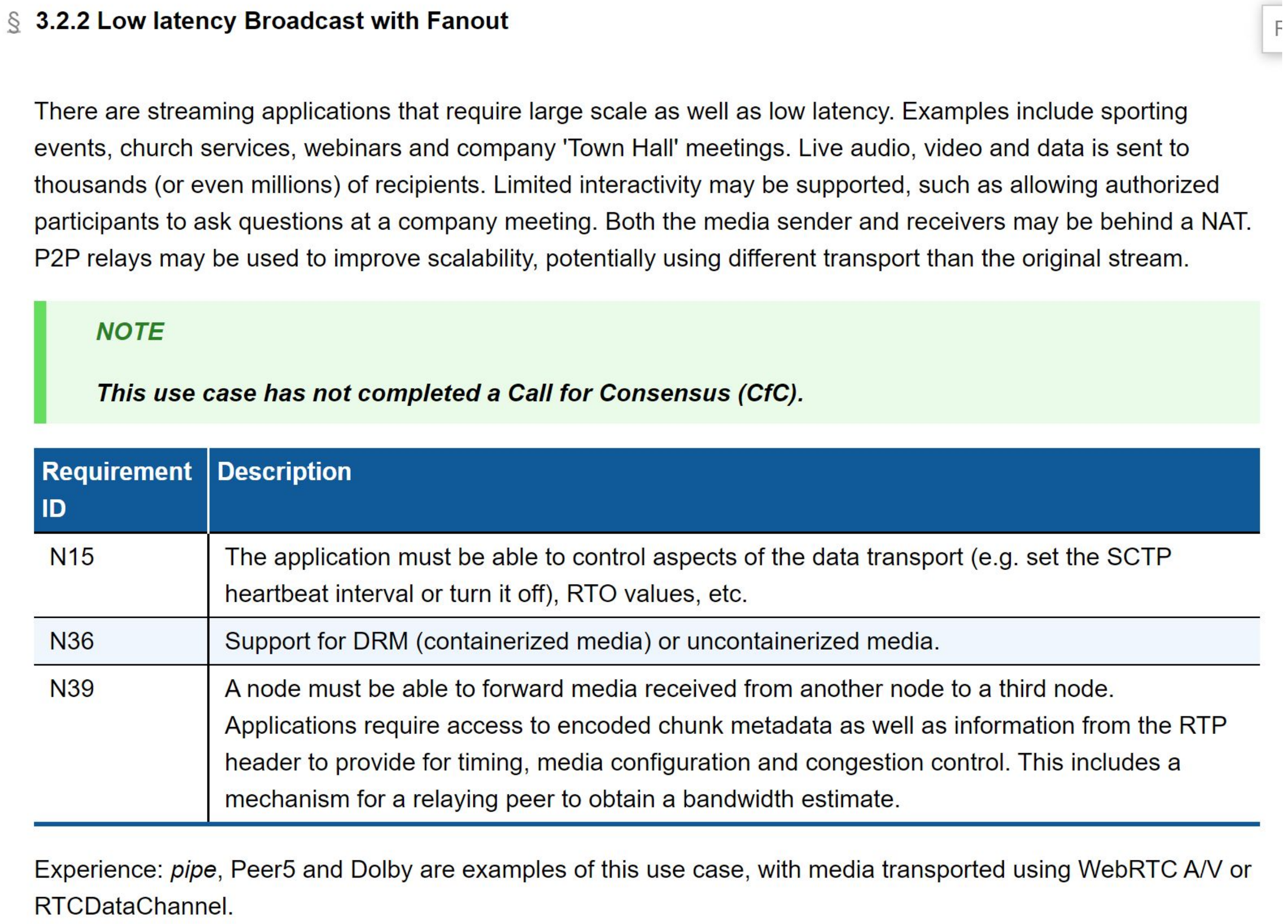
Requirement ID	Description
N15	The application must be able to control aspects of the data transport (e.g. set the SCTP heartbeat interval or turn it off), RTO values, etc.
N37	It must be possible for the user agent's receive pipeline to process video at high resolution and framerate (e.g. without copying raw video frames).
N38	The application must be able to control the jitter buffer and rendering delay.

Experience: XCloud, Rainway, GeForce Now and Stadia are examples of this use case, with media transported using WebRTC A/V or RTCDataChannel.

Section 3.2.1: Game streaming (cont'd)

- Requirements N37 and N38 relate to performance
  - N37 may require support for hw-accelerated decode.
  - Lots of known issues w/hw-acceleration, including handling of decode errors.
  - Potential missing requirements
    - Support for custom FEC. May be appealing at high resolutions (4K) and frames rates (60 fpbs).
- Should requirement N15 be included?
  - Media typically flows from server->browser
    - Server can implement its own custom transport, including congestion control.
    - Same dynamic in P2P game streaming where game console (native application) streams to a mobile device (browser).
  - What about browser -> server flows?
    - Does N15 apply to input sent to server from a game console?

Section 3.2.2: Low latency Broadcast with Fanout



§ 3.2.2 Low latency Broadcast with Fanout

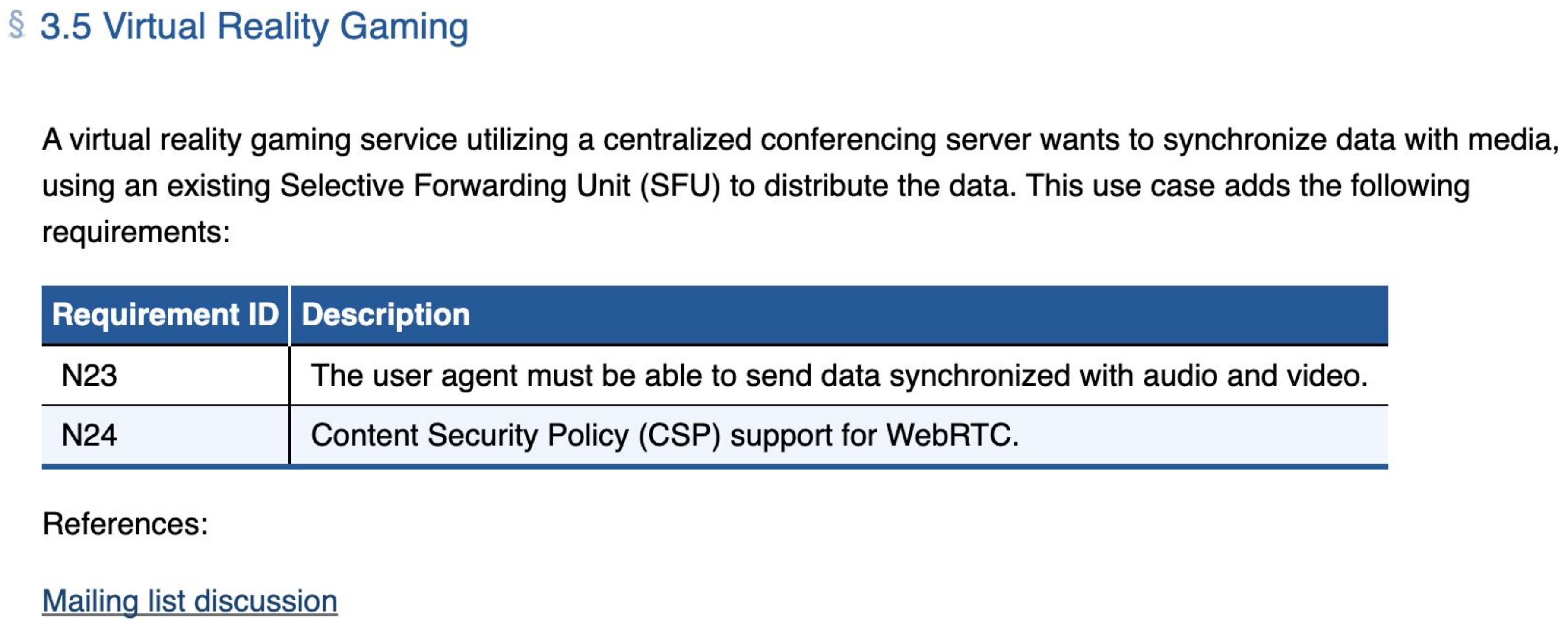
There are streaming applications that require large scale as well as low latency. Examples include sporting events, church services, webinars and company 'Town Hall' meetings. Live audio, video and data is sent to thousands (or even millions) of recipients. Limited interactivity may be supported, such 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 not completed a Call for Consensus (CfC).***

Requirement ID	Description
N15	The application must be able to control aspects of the data transport (e.g. set the SCTP heartbeat interval or turn it off), RTO values, etc.
N36	Support for DRM (containerized media) or uncontainerized media.
N39	A node must be able to forward media received from another node to a third node. 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 using WebRTC A/V or RTCDataChannel.

Section 3.5: Virtual reality gaming



§ 3.5 Virtual Reality Gaming

A virtual reality gaming service utilizing a centralized conferencing server wants to synchronize data with media, using an existing Selective Forwarding Unit (SFU) to distribute the data. This use case adds the following requirements:

Requirement ID	Description
N23	The user agent must be able to send data synchronized with audio and video.
N24	Content Security Policy (CSP) support for WebRTC.

References:

Mailing list discussion

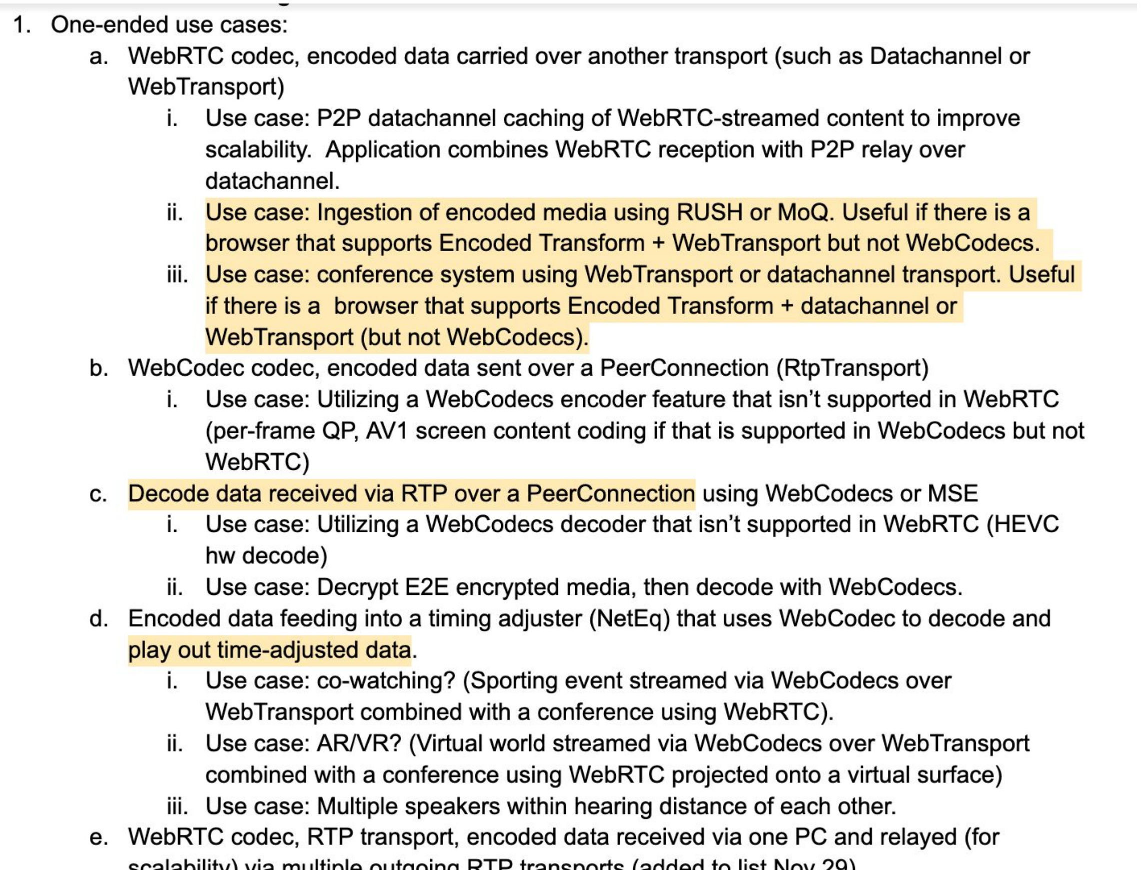
Section 3.5: Virtual reality gaming (cont’d)

- Are the requirements complete?
  - Virtual reality games often support spatial audio
    - Can be implemented via “bring your own codec”

Issue 106: “One-ended” Use Cases

- Post from Harald on August 27, 2022:
  - Encoded data access - Requirements for a new API from Harald Alvestrand on 2022-08-27 (public-webrtc@w3.org from August 2022)
- Envisioned applications:
  - End to end encryption (app-controlled) of video and audio streams
  - “SFU in the browser”: selective forwarding of encoded frames to other network entities (“Live broadcast with fanout”)
  - Alternative transport: moving frames over mechanisms other than RTP.
  - Alternative generators: Generating frames using other mechanisms such as WebCodecs rather than WebRTC (NV issue 81, 82)
  - Alternative consumers: Feeding frames to WebCodecs, MSE-type mechanisms or other destinations rather than WebRTC for decoding
  - Integration with MSE-type content protection mechanisms

[Issue 106](): “One-ended” Use Cases



1. One-ended use cases:
    a. WebRTC codec, encoded data carried over another transport (such as Datachannel or WebTransport)
        i. Use case: P2P datachannel caching of WebRTC-streamed content to improve scalability. Application combines WebRTC reception with P2P relay over datachannel.
        ii. Use case: Ingestion of encoded media using RUSH or MoQ. Useful if there is a browser that supports Encoded Transform + WebTransport but not WebCodecs.
        iii. Use case: conference system using WebTransport or datachannel transport. Useful if there is a browser that supports Encoded Transform + datachannel or WebTransport (but not WebCodecs).
    b. WebCodec codec, encoded data sent over a PeerConnection (RtpTransport)
        i. Use case: Utilizing a WebCodecs encoder feature that isn’t supported in WebRTC (per-frame QP, AV1 screen content coding if that is supported in WebCodecs but not WebRTC)
    c. Decode data received via RTP over a PeerConnection using WebCodecs or MSE
        i. Use case: Utilizing a WebCodecs decoder that isn’t supported in WebRTC (HEVC hw decode)
        ii. Use case: Decrypt E2E encrypted media, then decode with WebCodecs.
    d. Encoded data feeding into a timing adjuster (NetEq) that uses WebCodec to decode and play out time-adjusted data.
        i. Use case: co-watching? (Sporting event streamed via WebCodecs over WebTransport combined with a conference using WebRTC).
        ii. Use case: AR/VR? (Virtual world streamed via WebCodecs over WebTransport combined with a conference using WebRTC projected onto a virtual surface)
        iii. Use case: Multiple speakers within hearing distance of each other.
    e. WebRTC codec, RTP transport, encoded data received via one PC and relayed (for scalability) via multiple outgoing RTP transports (added to list Nov 29)

Issue 81: Transport pre-encoded live content over RTP

- This use case has come up in a couple of contexts, including one where the requester wanted to:
  a. Install a video camera that delivered pre-encoded H.264 data (on an interface independent of WebRTC)
  b. Send the resulting video stream out over a WebRTC RTP connection (for instance, as part of a video surveillance service that otherwise used RTP transmission)
- I (hta) think this can be satisfied with the following interfaces:
  a. Create encoded video frames based on existing encoded data + metadata
  b. Enqueue the encoded video frames on an outgoing RTCRtpSender
  c. Take signals from the RTCRtpSender to reconfigure the camera to provide encoding of the appropriate bandwidth

Issue 82: Transmit stored pre-encoded content over RTP

This has come up in a couple of contexts, including the provision of "wait signals" and the insertion of pre-recorded segments into an otherwise live conference application.

The important points are:

- The media is pre-recorded (but the media may be available in multiple formats/qualities)
- The desired transmission mechanism is RTP

I (hta) think this can be achieved by:

- Providing a means to create frames based on existing encoded video + metadata
- Providing a means to enqueue those frames on an existing RTCRtpSender
- Providing a means to take signals from the RTCRtpSender about available bandwidth and requests for new keyframes and have them processed in an application-specific manner

Responses to congestion signals may involve switching the source of frames to a lower quality source (much like DASH does), or it may involve switching the source to a video showing "wait a bit", or it may involve frame decimation of some kind (assuming the signal is encoded in a decimation-compatible format such as an SVC encoding). These decisions don't need to be part of the WebRTC component.

Use Cases That Might Not Be Covered (Peter)

- Bring your own (web) codec
  - Example: HEVC over RTP
  - Needs a packetization API? (can’t assume it is already in WebRTC)
- Bring your own FEC
  - Example: My FEC is better than what’s built into the browser
  - Needs a packetization API?
    - But is it enough to just inject many small “video frames” that are 1 MTU each?
- SFU Between “RTP over QUIC” and “RTP over UDP”
  - You might need to control RTP packetization…

Discussion (End Time: 08:50)

-

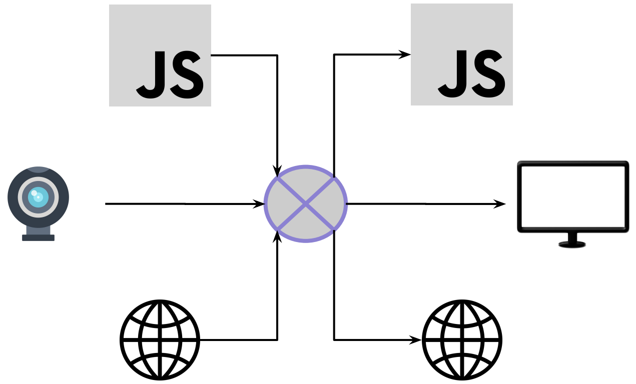
Encoded Transform (Harald)

Start Time: 08:50 AM

End Time: 09:10 AM

Encoded Media Manipulation - beyond Bump-In-Stack

- Unlocks several interesting use cases
  - Relay without decoding/decrypting
  - Send over non-RTP transports
  - Send pre-encoded media
- Issues / PR on nv-use-cases
- PR for explainer to encoded-media



API design - known requirements

- Must be usable within the PeerConnection/RTP ecosystem
  - Not contemplating greenfield designs
- Must allow frames that are not created by PeerConnection
- Must allow both sending and decoding of such frames

API design - uncertain requirements

- Must allow congestion control to work
  - Take signals from sender about how much to produce
- Must allow stream repair to work
  - Treat “request keyframe” properly
- May need to allow resolution negotiation
  - Destination may have opinions on what it can consume

Incremental API design

- Address known requirements first
  - Create frame from data + metadata
  - Modify frame metadata (we can already adjust data)
  - Add a clone() method for simplicity (PR)
- Take time to firm up uncertain requirements
  - Sketch in the IETF hackathon git repository
  - No formal proposal yet

WG decisions requested (aspirational)

- Agree that addressing these use cases is within the scope of the WG
- Agree that incremental API development is an OK approach
- CfC on Low Latency Streaming Use Cases (3.2.1 and 3.2.2)
- Agree that creating encoded frames is a required first step, and can be done ~now

Discussion (End Time: 09:10)

-

Encoded Transform Issues (fippo)

Start Time: 09:10 AM

End Time: 09:45 AM

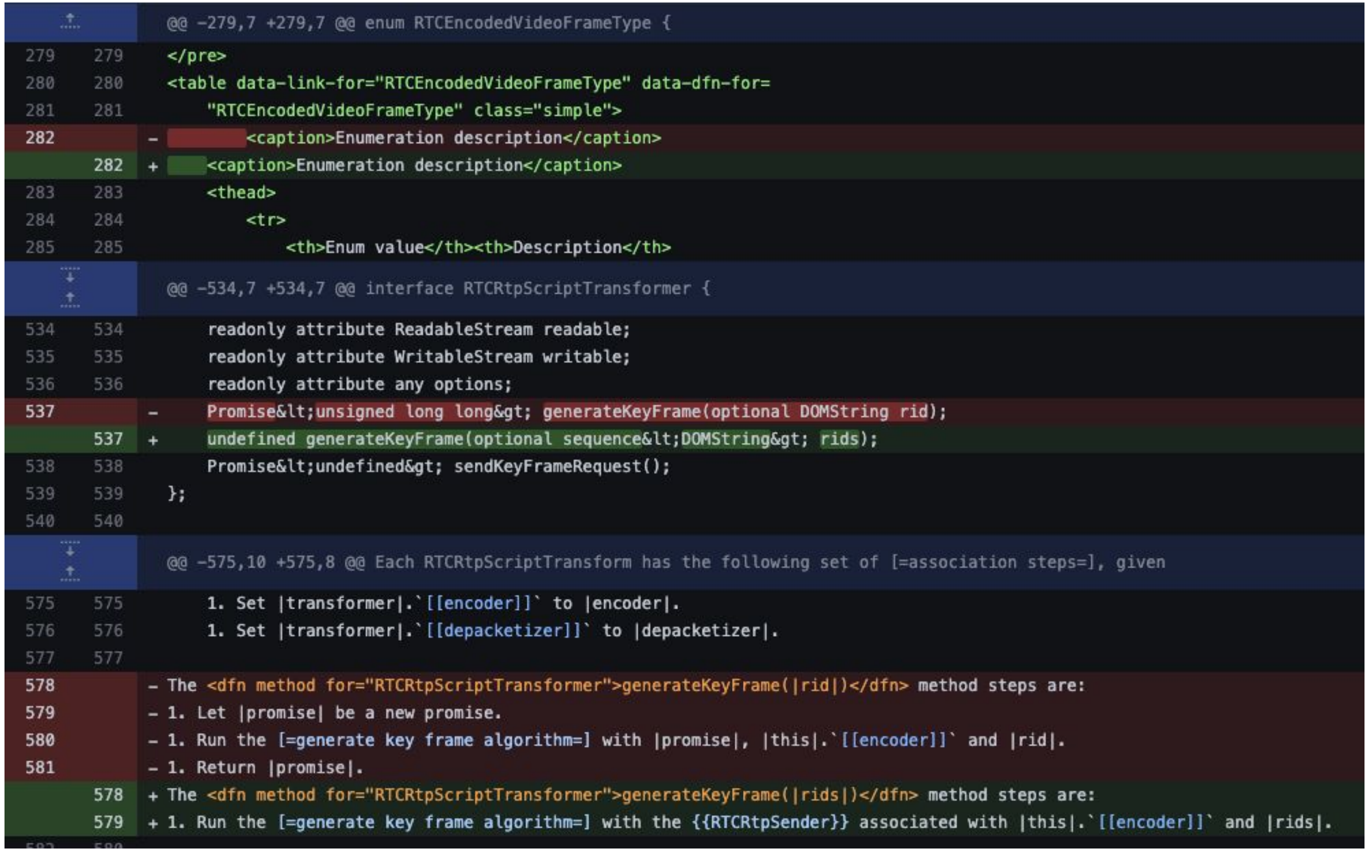
For Discussion Today

- Issue 143/PR 165: make generateKeyFrame() take a list of rids and return undefined
- Issue 167: Timing Metadata
- Issue 168: RTP metadata
- Issue 166/PR 154: RTP sequence number
- Issue 147: RID/MID
- Issue 169: add RTP timestamp to metadata
- Issue 158/PR 140: mimeType metadata
- Issue 170: Incompatible SVC metadata

Issue 143/PR 165: make generateKeyFrame() take a list of rids and return undefined

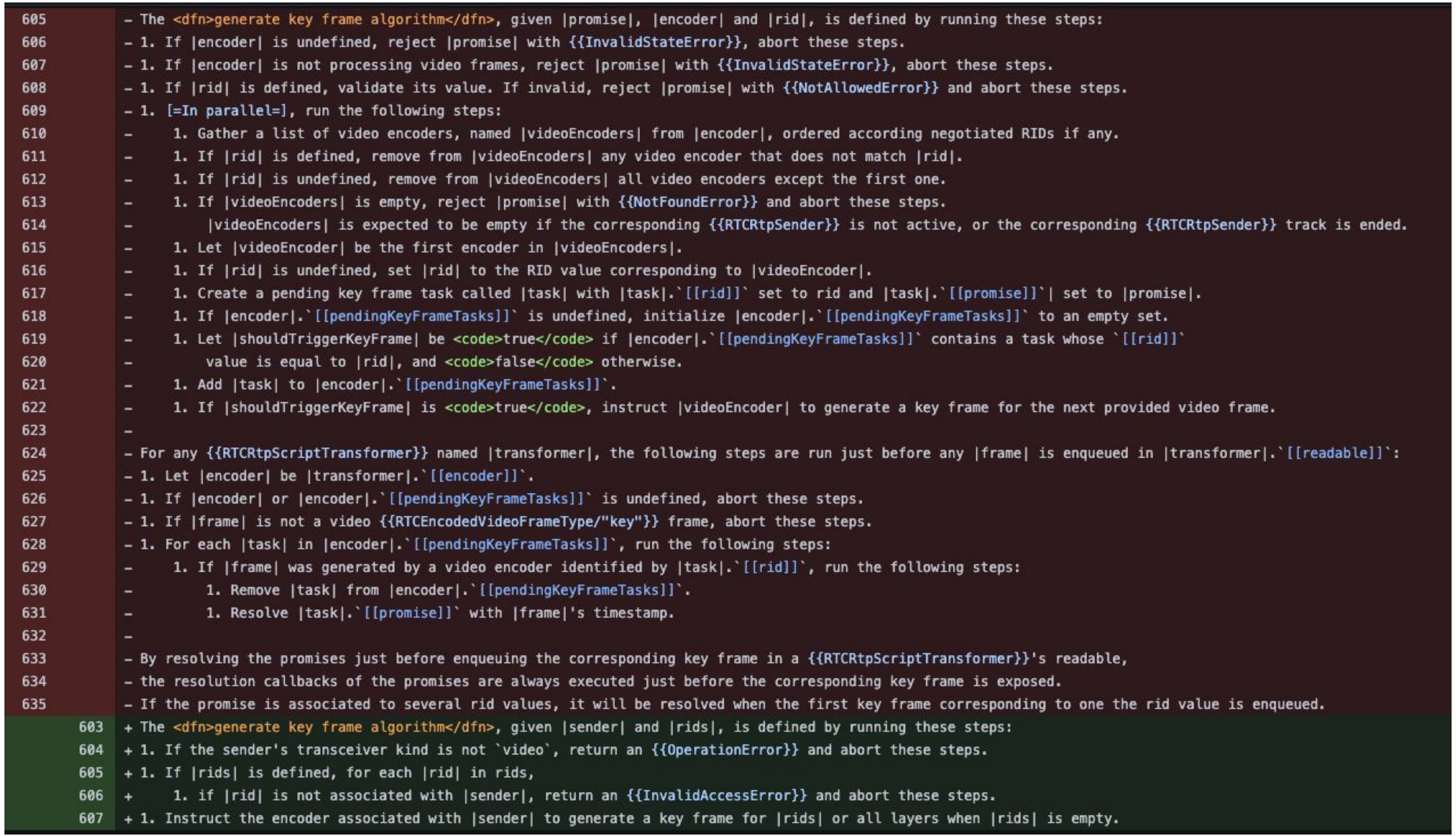
- Issue 143 discussed during October interim
  - Resolution: “pass an array arguments to generateKeyframes”
- PR 165 implements the resolution
  - Takes a list of rids
    - Must be negotiated rids
    - Empty lists means “all of them”
  - No return value
    - Since the encoder might be currently unable to generate a key frame
- Proposal:
  - Merge PR 165

Issue 143/PR 165: make generateKeyFrame() take a list of rids and return undefined (cont'd)



```
@@ -279,7 +279,7 @@ enum RTCEncodedVideoFrameType {
  </pre>
  <table data-link-for="RTCEncodedVideoFrameType" data-dfn-for=
      "RTCEncodedVideoFrameType" class="simple">
-         <caption>Enumeration description</caption>
+     <caption>Enumeration description</caption>
      <thead>
          <tr>
              <th>Enum value</th><th>Description</th>
@@ -534,7 +534,7 @@ interface RTCRtpScriptTransformer {
      readonly attribute ReadableStream readable;
      readonly attribute WritableStream writable;
      readonly attribute any options;
-     Promise&lt;unsigned long long&gt; generateKeyFrame(optional DOMString rid);
+     undefined generateKeyFrame(optional sequence&lt;DOMString&gt; rids);
      Promise&lt;undefined&gt; sendKeyFrameRequest();
  };

@@ -575,10 +575,8 @@ Each RTCRtpScriptTransform has the following set of [=association steps=], given
      1. Set |transformer|.`[[encoder]]` to |encoder|.
      1. Set |transformer|.`[[depacketizer]]` to |depacketizer|.

- The <dfn method for="RTCRtpScriptTransformer">generateKeyFrame(|rid|)</dfn> method steps are:
- 1. Let |promise| be a new promise.
- 1. Run the [=generate key frame algorithm=] with |promise|, |this|.`[[encoder]]` and |rid|.
- 1. Return |promise|.
+ The <dfn method for="RTCRtpScriptTransformer">generateKeyFrame(|rids|)</dfn> method steps are:
+ 1. Run the [=generate key frame algorithm=] with the {{RTCRtpSender}} associated with |this|.`[[encoder]]` and |rids|.
```

Issue 143/PR 165: make generateKeyFrame() take a list of rids and return undefined (cont'd)



```
- The <dfn>generate key frame algorithm</dfn>, given |promise|, |encoder| and |rid|, is defined by running these steps:
- 1. If |encoder| is undefined, reject |promise| with {{InvalidStateError}}, abort these steps.
- 1. If |encoder| is not processing video frames, reject |promise| with {{InvalidStateError}}, abort these steps.
- 1. If |rid| is defined, validate its value. If invalid, reject |promise| with {{NotAllowedError}} and abort these steps.
- 1. [=In parallel=], run the following steps:
-     1. Gather a list of video encoders, named |videoEncoders| from |encoder|, ordered according negotiated RIDs if any.
-     1. If |rid| is defined, remove from |videoEncoders| any video encoder that does not match |rid|.
-     1. If |rid| is undefined, remove from |videoEncoders| all video encoders except the first one.
-     1. If |videoEncoders| is empty, reject |promise| with {{NotFoundError}} and abort these steps.
-         |videoEncoders| is expected to be empty if the corresponding {{RTCRtpSender}} is not active, or the corresponding {{RTCRtpSender}} track is ended.
-     1. Let |videoEncoder| be the first encoder in |videoEncoders|.
-     1. If |rid| is undefined, set |rid| to the RID value corresponding to |videoEncoder|.
-     1. Create a pending key frame task called |task| with |task|.`[[rid]]` set to rid and |task|.`[[promise]]`| set to |promise|.
-     1. If |encoder|.`[[pendingKeyFrameTasks]]` is undefined, initialize |encoder|.`[[pendingKeyFrameTasks]]` to an empty set.
-     1. Let |shouldTriggerKeyFrame| be <code>true</code> if |encoder|.`[[pendingKeyFrameTasks]]` contains a task whose `[[rid]]`
-         value is equal to |rid|, and <code>false</code> otherwise.
-     1. Add |task| to |encoder|.`[[pendingKeyFrameTasks]]`.
-     1. If |shouldTriggerKeyFrame| is <code>true</code>, instruct |videoEncoder| to generate a key frame for the next provided video frame.
-
- For any {{RTCRtpScriptTransformer}} named |transformer|, the following steps are run just before any |frame| is enqueued in |transformer|.`[[readable]]`:
- 1. Let |encoder| be |transformer|.`[[encoder]]`.
- 1. If |encoder| or |encoder|.`[[pendingKeyFrameTasks]]` is undefined, abort these steps.
- 1. If |frame| is not a video {{RTCEncodedVideoFrameType/"key"}} frame, abort these steps.
- 1. For each |task| in |encoder|.`[[pendingKeyFrameTasks]]`, run the following steps:
-     1. If |frame| was generated by a video encoder identified by |task|.`[[rid]]`, run the following steps:
-         1. Remove |task| from |encoder|.`[[pendingKeyFrameTasks]]`.
-         1. Resolve |task|.`[[promise]]` with |frame|'s timestamp.
-
- By resolving the promises just before enqueuing the corresponding key frame in a {{RTCRtpScriptTransformer}}'s readable,
- the resolution callbacks of the promises are always executed just before the corresponding key frame is exposed.
- If the promise is associated to several rid values, it will be resolved when the first key frame corresponding to one the rid value is enqueued.
+ The <dfn>generate key frame algorithm</dfn>, given |sender| and |rids|, is defined by running these steps:
+ 1. If the sender's transceiver kind is not `video`, return an {{OperationError}} and abort these steps.
+ 1. If |rids| is defined, for each |rid| in rids,
+     1. if |rid| is not associated with |sender|, return an {{InvalidAccessError}} and abort these steps.
+ 1. Instruct the encoder associated with |sender| to generate a key frame for |rids| or all layers when |rids| is empty.
```

Issue 167: Timing Metadata

- Timing Model discussed at November VI
  - VideoFrameCallbackMetaData in rVFC specification
    - Includes receiveTime, captureTime, rtpTimestamp, mediaTime
    - Issue 601: Expose in VideoFrame
  - Resolution: “file specific issues on specific specs”
    - Issue 167 filed on encoded transform
    - Issue 88 filed on mediacapture transform
- Proposal: add timing metadata to RTCEncoded*Metadata
  - receiveTime as defined in rVFC
  - captureTime (#137, #159, defined in rVFC)

Issue 168: RTP metadata

- Issue 160: overall requirements arising from the “Low Latency Broadcast with Fanout” use case.
  - Issue 161: Add a clone operator
  - Issue 162: Need to modify PT, SSRC, CSRC in metadata
- Issue 168 tracks need for more complete RTP header metadata
  - Not just *PT*, *SSRC*, *[CSRC]*, but also:
    - Issue 166: sequence number
    - Issue 147: MID/RID
    - Issue 169: RTP timestamp (in metadata)
    - Marker bit
    - [header extensions]

Miscellaneous metadata

- mimeType (**Issue 158** / **PR 140**)
  - resolved from payload type
  - Presented at October VI
  - Details in Slide 46
- Codec-specific metadata like
  - width, height (issue 138: only incoming frames?)
  - Audio level, voice activity bit

Issue 166/PR 154: RTP sequence number

- Use case: Low Latency Broadcast with Fanout
- PR 154: incoming audio RTP sequence number
  - Incoming audio only, more complex for video
- Proposal:
  - Merge PR 154

Issue 147: MID/RID

- mid
  - transform needs to know from which transceiver it is receiving things from
  - Can use SSRC but…
- rid
  - transform needs to know from which rid/layer it is sending things for in simulcast
  - Can use SSRC but…
- Proposal:
  - add mid and rid to metadata

Issue 169: add RTP timestamp to metadata

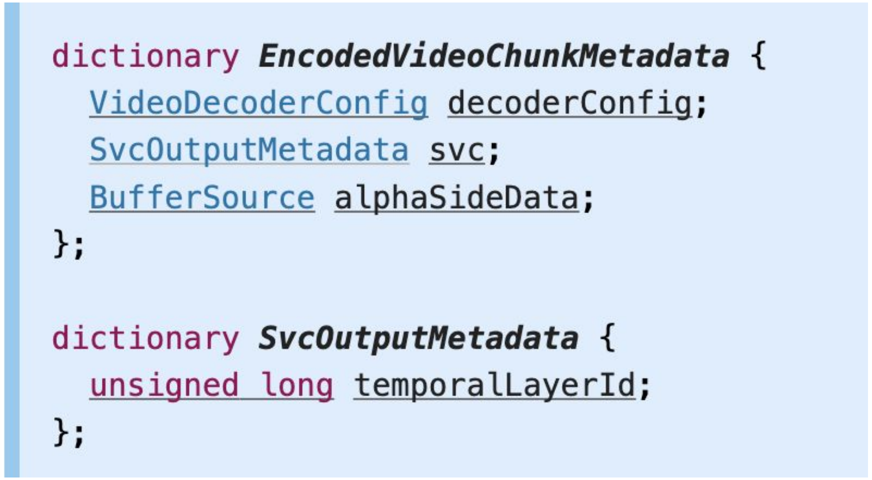
- RTCEncodedAudioFrame/VideoFrame
    - readonly attribute unsigned long timestamp
    - This is the RTP timestamp!
    - Problem: timestamp can’t be modified
- Proposal: add to respective metadata
    - Deprecate on main object
    - Remove from implementations for 1-2 releases
    - Re-add as defined in WebCodecs
      readonly attribute long long timestamp

Issue 158/PR 140: mimeType metadata

- October resolution: add mimeType
- Additional questions
  - Raw mimetype
    - Sufficient to tell VP8, H264 apart
    - Not sufficient to tell H264 profile levels apart
  - Do we need fmtp?
    - We have that in codec stats
- Proposal:
  - merge mimetype **PR 140**
  - add fmtp once someone commits to implement

Issue 170: Incompatible SVC metadata

- WebCodecs defines EncodedChunkMetadata as follows:



```
dictionary EncodedVideoChunkMetadata {
  VideoDecoderConfig decoderConfig;
  SvcOutputMetadata svc;
  BufferSource alphaSideData;
};

dictionary SvcOutputMetadata {
  unsigned long temporalLayerId;
};
```

- Dictionary has structure to allow for future expansion of SvcOutputMetadata dictionary.

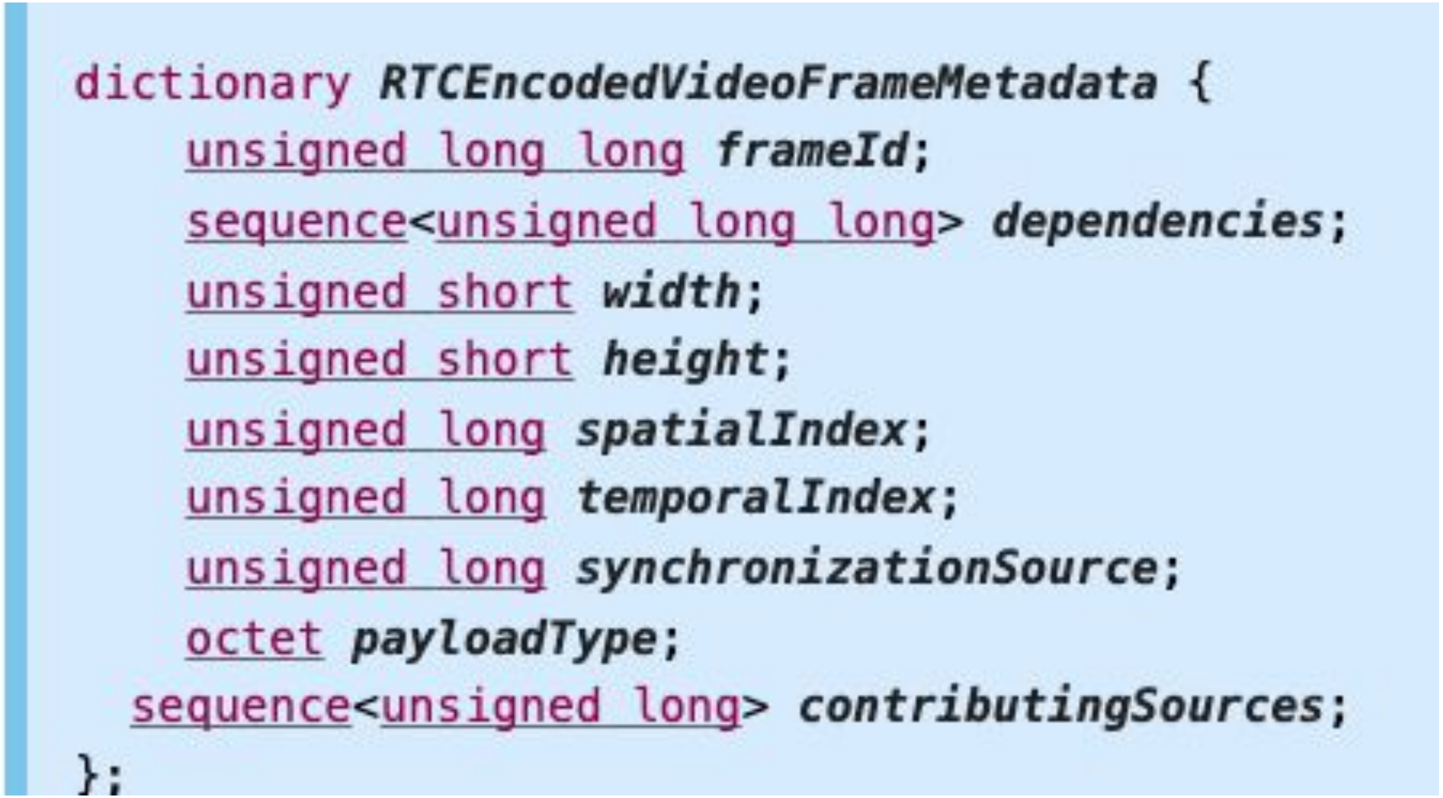
Issue 170: Incompatible SVC metadata (cont'd)

- Complete WebCodecs SVC metadata proposal is based on the information included within the Dependency Descriptor RTP header extension:

```
dictionary EncodedVideoChunkMetadata {
// Number for identifying this frame in |dependsOnIds| and |chainLinks| (for other chunks).
unsigned short frameNumber;

// List of frameNumbers that this chunk depends on. Used to detect/handle network loss. Decoding out of order is an error.
list<unsigned long> dependsOnIds;

// IDs of the spatial layer and temporal layer this chunk belongs to.
unsigned long spatialLayerId;
unsigned long temporalLayerId;

// List of decoder targets this frame participates in. Used to know whether this frame should be sent (forwarded) to a given
receiver  depending on what decode targets the receiver is expecting. Decode target is a numerical index determined by the
encoder. No commitment that a particular number implies a given layer.
list<unsigned long> decodeTargets;

// Mapping of decode target -> the last important frame to decode prior to "this"  frame for the given decode target.
// Used to ensure we preserve decode order for the desired decode target. It is insufficient to simply satisfy the
dependencies for the current frame. See example.
map<unsigned long, unsigned long> chainLinks;
};
```

Issue 170: Incompatible SVC metadata (cont'd)

- Comparison with RTCEncodedVideoFrameMetadata:



```
dictionary RTCEncodedVideoFrameMetadata {
    unsigned long long frameId;
    sequence<unsigned long long> dependencies;
    unsigned short width;
    unsigned short height;
    unsigned long spatialIndex;
    unsigned long temporalIndex;
    unsigned long synchronizationSource;
    octet payloadType;
  sequence<unsigned long> contributingSources;
};
```

Issue 170: Incompatible SVC metadata (cont'd)

- Issues:
  - Name differences
    - `temporalLayerId` vs. `temporalIndex`
    - `spatialLayerId` vs. `spatialIndex`
  - Type mismatches:
    - `unsigned short frameNumber` vs. `unsigned long long frameId`
    - `sequence <unsigned long> dependsOnIds` vs. `sequence <unsigned long long> dependencies`
  - Missing information
    - `sequence <unsigned long> decodeTargets`
      - List of decode targets this frame participates in. Used to determine whether this frame should be forwarded to a receiver based on what decode targets the receiver is expecting.
    - `Map <unsigned long, unsigned long> chainLinks`
      - Used to ensure we preserve decode order for the desired decode target. It is insufficient to satisfy the dependencies for the current frame.
  - Proposal: submit PR to harmonize SVC metadata between Encoded Transform and WebCodecs

Discussion (End Time: 09:45)

-

WebRTC-PC

Start Time: 09:45 AM

End Time: 10:00 AM

For Discussion Today

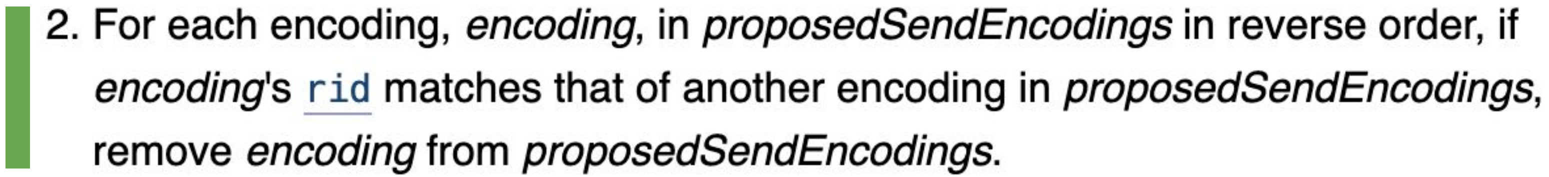
- Issue 2795: Missing URL in RTCIceCandidateInit
- Issue 2780: duplicate rids in sRD underspecified
- PR 2801: Prune createAnswer()'s encodings and [[SendEncodings]] in sLD(answer).

Issue 2795: Missing URL in RTCIceCandidateInit

- Added url and relayProtocol to RTCIceCandidate
  - These are not possible to reconstruct and only available for local candidates
  - Not serialized by toJSON, not to be signaled
- 4.8.1: “...the remaining attributes are derived from parsing the candidate”
  - Not updated when adding the new properties
- Proposal:
  - update description in 4.8.1
  - Write more tests!

Issue 2780 / PR 2800: duplicate rids in sRD underspecified

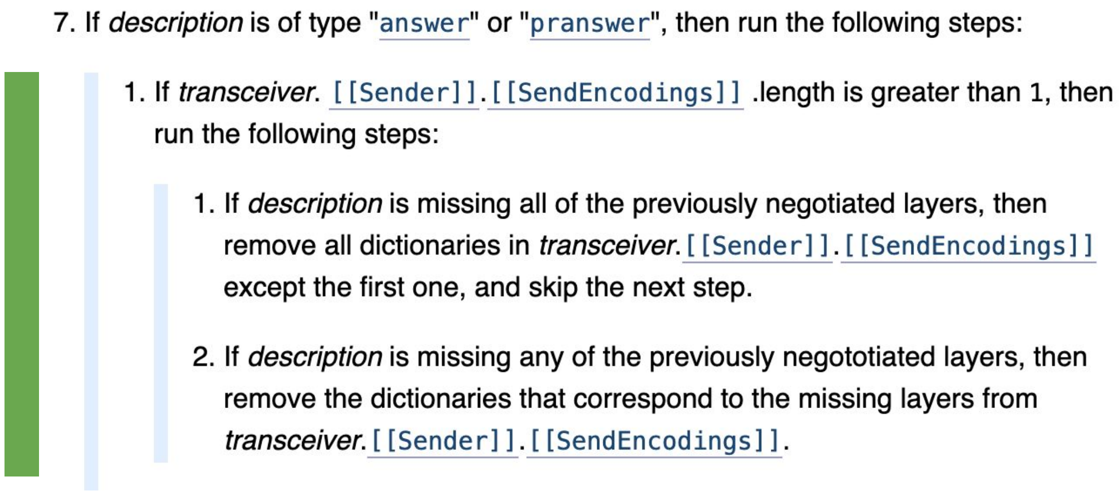
Proposal: Remove duplicate rids in proposedSendEncodings:


2. For each encoding, *encoding*, in *proposedSendEncodings* in reverse order, if *encoding*'s `rid` matches that of another encoding in *proposedSendEncodings*, remove *encoding* from *proposedSendEncodings*.

PR 2801: Prune createAnswer()'s encodings and [[SendEncodings]] in sLD(answer).

A follow-up to #2758 whose intent was to defer pruning of [[SendEncodings]] to sLD(answer), but mistakenly relied on the spec's existing pruning language which only applies to sRD(answer).

Add similar language to sLD(answer):

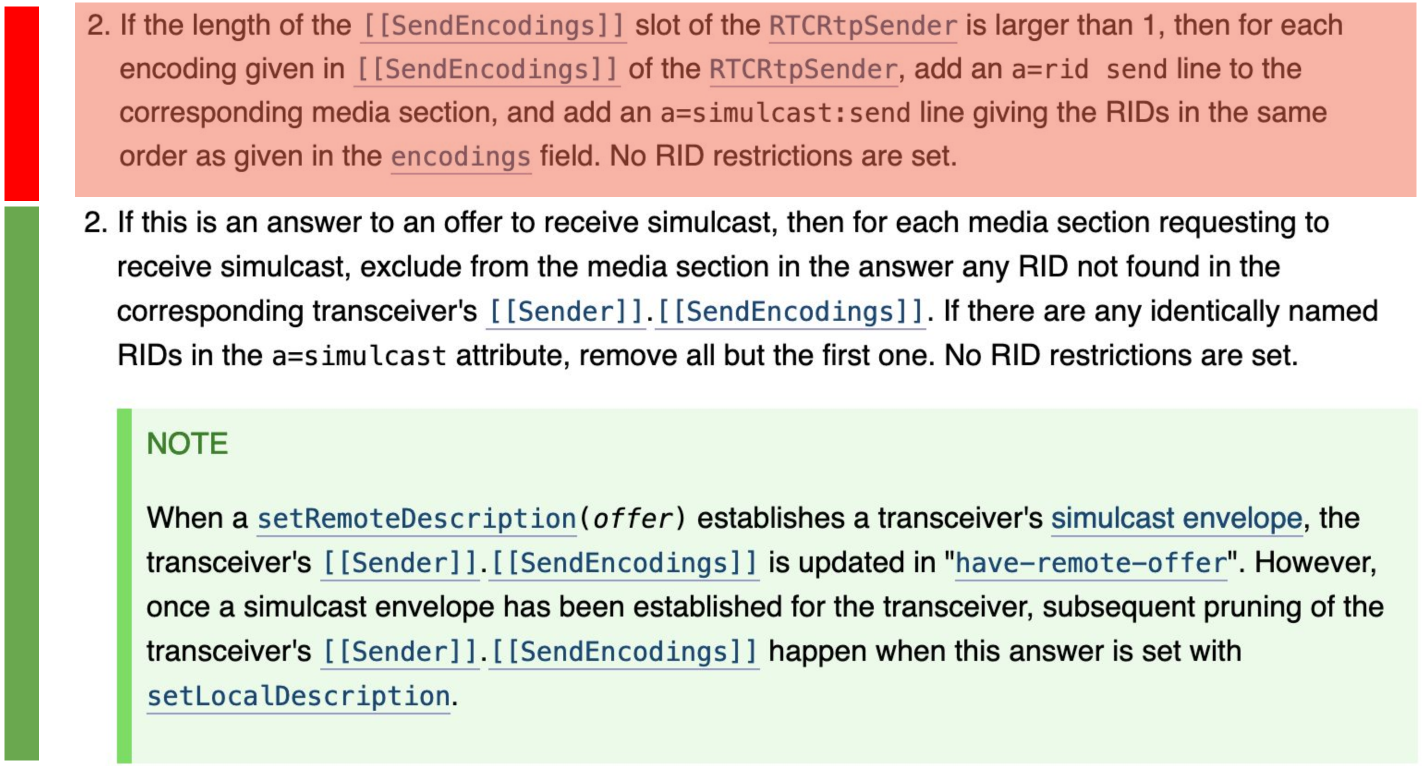


7. If *description* is of type "`answer`" or "`pranswer`", then run the following steps:
    1. If *transceiver*. `[[Sender]]`.`[[SendEncodings]]` .length is greater than 1, then run the following steps:
        1. If *description* is missing all of the previously negotiated layers, then remove all dictionaries in *transceiver*.`[[Sender]]`.`[[SendEncodings]]` except the first one, and skip the next step.
        2. If *description* is missing any of the previously negotiated layers, then remove the dictionaries that correspond to the missing layers from *transceiver*.`[[Sender]]`.`[[SendEncodings]]`.

Next, we need to touch where this *description* comes from (next slide: createAnswer)

PR 2801: Prune createAnswer()'s encodings and [[SendEncodings]] in sLD(answer).

Fix **final steps to create an answer** to prune based on JSEP’s answer ∩ [[Sender]].[[SendEncodings]], instead of parroting create offer:



2. If the length of the `[[SendEncodings]]` slot of the `RTCRtpSender` is larger than 1, then for each encoding given in `[[SendEncodings]]` of the `RTCRtpSender`, add an `a=rid send` line to the corresponding media section, and add an `a=simulcast:send` line giving the RIDs in the same order as given in the `encodings` field. No RID restrictions are set.

2. If this is an answer to an offer to receive simulcast, then for each media section requesting to receive simulcast, exclude from the media section in the answer any RID not found in the corresponding transceiver's `[[Sender]]`.`[[SendEncodings]]`. If there are any identically named RIDs in the `a=simulcast` attribute, remove all but the first one. No RID restrictions are set.

NOTE

When a `setRemoteDescription`(*offer*) establishes a transceiver's simulcast envelope, the transceiver's `[[Sender]]`.`[[SendEncodings]]` is updated in "`have-remote-offer`". However, once a simulcast envelope has been established for the transceiver, subsequent pruning of the transceiver's `[[Sender]]`.`[[SendEncodings]]` happen when this answer is set with `setLocalDescription`.

Discussion (End Time: 09:50)

-

Thank you

<u>Special thanks to:</u>

WG Participants, Editors & Chairs