W3C WebRTC WG Meeting

January 19, 2021
8:00 AM - 9:30 AM Pacific Time

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 1st interim meeting of 2021 of the W3C WebRTC WG!
  - During this meeting, we will talk about Insertable Streams, a cropping control for MediaStreamTrack, and open issues from WebRTC Extensions.

A Request...

- Please fill out the Doodle Poll for the February WEBRTC WG virtual interim:
  - https://www.doodle.com/poll/gkuzpac6kdet5qng
- Poll closes today, results will be announced tomorrow.

About this Virtual Meeting

- Meeting info:
  - https://www.w3.org/2011/04/webrtc/wiki/January_19_2021
- Link to latest drafts:
  - https://w3c.github.io/mediacapture-main/
  - 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://w3c.github.io/webrtc-dscp-exp/
  - https://github.com/w3c/webrtc-insertable-streams
  - 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 being recorded. The recording will be public.

Issues for Discussion Today

- Raw Media Insertable Streams Status (Harald, 5 minutes)
- Beyond WebRTC Insertable Streams (Harald, 5 minutes)
- WebRTC Insertable Stream (Youenn, 10 minutes)
  - **Issue #48 - WebRTC insertable streams as transform (Youenn)**
- Add a cropping control to MediaStreamTrack (Elad Alon)
- WebRTC Extensions
  - Issue 52: Invalid TURN credentials: What Function Should Fail? (Henrik)
  - Issue 63: Enabling opus stereo audio without SDP munging (stereo=1) (Henrik)
  - Issue 230: Add support for WebRTC Data Channel in Worker (Youenn)

Issues for Discussion Today

- Raw Media Insertable Streams Status (Harald, 5 minutes)
- Beyond WebRTC Insertable Streams (Harald, 5 minutes)
- WebRTC Insertable Streams (Youenn, 10 minutes)
    - **Issue #48 - WebRTC insertable streams as transform (Youenn)**

Raw Media Insertable Streams Status Report (Harald)

- Video Breakout Box is available in Chrome M89 (behind a flag / origin trial)
- Performance is ~equivalent to production Canvas-based solutions
- Changes to spec since last meeting:
  - Stage 2 is removed. Only the separate TrackProcessor and TrackGenerator objects remain.
- Changes from spec to implementation (to be merged):
  - TrackProcessor and TrackGenerator prefixed with MediaStream
  - TrackProcessor takes a “bufferSize” argument
- Not yet implemented, not changed in spec:
  - Control signals (attribute name “writableControl” and “readableControl” suggested for the streams)

Beyond Insertable Streams

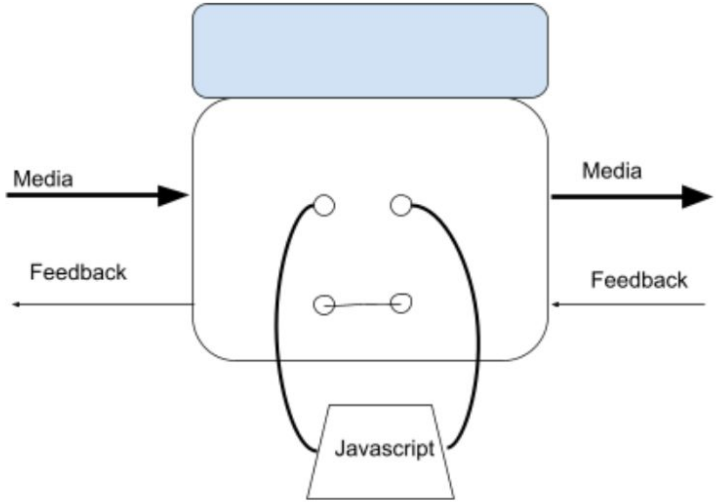
Encoded data - next steps towards the Lego Laser

Insertable Streams is a wires-out-the-side model

This means that in addition to the exposed stream, there is a reverse stream of feedback data that is not shown to the user

It is not arbitrary where to connect the wires - JS can't create frames, and has to send frames back to the sender that created them.

This model was explicitly abandoned for Breakout Box (nonencoded data)

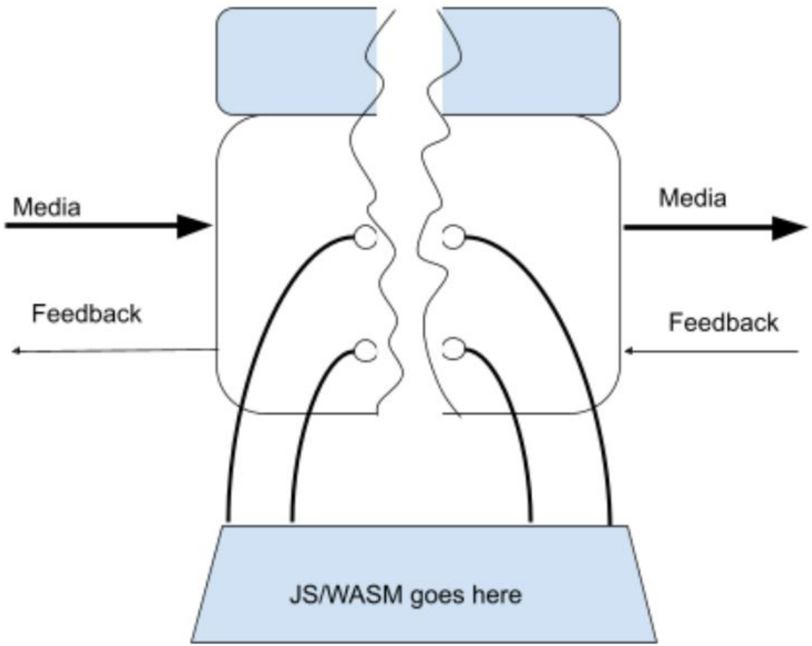


A Chained Boxes model is cleaner and more flexible

Requires the feedback channel to be explicit

In Encoded Data, a lot more feedback is needed than for non-encoded

We also should not delay dealing with the SDP issue any longer



First Sketch of An API

```
interface RTCRtpBaseSender {
   // Transport and stats attributes
}

interface RTCRtpSender : RTCRtpBaseSender {
  // everything to do with tracks goes here
}

interface RTCRtpFrameSender : RTCRtpBaseSender {

   Stream writable;  // Stream of EncodedFrame and control objects
   Stream readableControl;  // Stream of control objects
}
```

In RTCRtpTransceiver, the “sender” attribute has type (RTCRtpFrameSender or RTCRtpSender)

Requesting and Receiving Frame Streams

When sending, we can use AddFrameStream() rather than AddTrack.

When receiving, we can set a configuration attribute (per type) called whatSignalDoIWant with possible values “track” or “stream-of-frames”; one leads to firing ontrack; the other leads to firing onframestream, with appropriate parameters.

The SDP question

When the format of the data is defined by the application, the SDP negotiated has to be cared for by the application too. For instance, in a pipeline that goes

WebCodec(AV2) -> PeerConnection

Negotiating any codec but AV2 would be silly - but the codec negotiated also defines the encapsulation, which is different from the frame format.

Suggested:

- Control message JS->sender:
  ProposeCodec(codec description) (multiple times)
- Control message sender->JS:
  CodecAccepted(codec description)
  NoCodecAccepted

A possible application example: Jitter buffer in JS

- RTP receiver side only (sender remains unchanged)
- Takes encoded frames as input
    - Needs encoded data, codec ID and timing info
- Emits raw audio frames (may feed a MediaStreamTrackGenerator)
- May use a platform WebCodec or WASM module for decoding
- Control signals need review and specification
- Challenge: Ability to deliver data at exact 10 ms intervals on the audio clock
- Challenge: No-reassign buffer control - at both encoded and raw side

Issue #48 - WebRTC insertable streams as transform (Youenn)

Ongoing Safari experiment

- Support of SFrame transform, JS transforms and combos
  - JS transforms executed in background threads as a default
- Available in Safari nightly behind a feature flag

Experimental API allows attaching transforms to senders and receivers

```
typedef (SFrameTransform or RTCRtpScriptTransform) RTCRtpTransform;
partial interface RTCRtpSender {
    attribute RTCRtpTransform? transform;
};
partial interface RTCRtpReceiver {
    attribute RTCRtpTransform? transform;
};
```

Issue #48 - WebRTC insertable streams as transform (Youenn)

Summary from last meeting

- Interest in RTCRtpSender.transform/RTCRtpReceiver.transform
- Interest in having background-thread processing as a default

Questions that were raised

- Expose streams or frames
- Expose a feedback channel
- Evaluate API complexity

Let's look at JS examples to answer these questions

#48 - WebRTC insertable streams - Example 1
RTCRtpScriptTransform in window
RTCRtpScriptTransformer in worker

```
// HTML page
const pc = new RTCPeerConnection();
const worker = new Worker("worker-module.js");

const sender = pc.addTrack(track, stream);
sender.transform = new RTCRtpScriptTransform(worker, "myEncryptFrameTransform");
sender.transform.port.postMessage("Hello Transformer");
sender.transform.onmessage = (e) => console.log(e.data);
```

```
// worker-module.js - stream based variant
function encryptFrame(frame)
{
    ...
}
onrtctransform = (e) => {
    const transformer = e.transformer;
    transformer.port.postMessage("Starting");
    transformer.readable.pipeThrough(createTransform())
        .pipeTo(transformer.writable)
}
function createTransform()
{
    return new TransformStream({ transform : encryptFrame });
}
```

```
// worker-module.js - frame based variant
function encryptFrame(frame)
{
    ...
}
onrtctransform = (e) => {
    const transformer = e.transformer;
    transformer.port.postMessage("Starting");
    transformer.onframe = async (e) => {
        transformer.write(await
encryptFrame(e.frame));
    }
}
```

#48 - WebRTC insertable streams - Example 2
RTCRtpScriptTransform in worker, stream transferable
Dedicated RTCRtpScriptTransform context

```
// HTML page
const pc = new RTCPeerConnection();
const worker = new Worker("worker-module.js");

const transformer = await new Promise(resolve => worker.onmessage = (e) => resolve(e.data);
const sender = pc.addTrack(track, stream);
sender.transform = transformer;


// worker-module.js - transform based variant
function encryptFrame(frame)
{
    ...
}
async function transformFrame(frame, context)
{
    context.enqueue(await encryptFrame(frame));
}
const transform = new RTCRtpScriptTransform({ transform });
self.postMessage(transform, [transform]);
```

#48 - WebRTC insertable streams - Example 3
Extending API for feedback control and more

```
// HTML page
const pc = new RTCPeerConnection();
const worker = new Worker("worker-module.js");

const sender = pc.addTrack(track, stream);
sender.transform = new RTCRtpScriptTransform(worker, "myEncryptFrameTransform");
sender.transform.port.postMessage("Hello Transformer");
sender.transform.onmessage = (e) => console.log(e.data);
```

```
// worker-module.js
onrtctransform = (e) => {
    const transformer = e.transformer;
    transformer.port.postMessage("Starting");
    transformer.readable.pipeThrough(createTransform(transformer))
        .pipeTo(transformer.writable);
    transformer.onbitratechange = (e) => {
        ...
    };
}
```

```
// worker-module.js
function encryptFrame(frame, transformer)
{
    if (needsKeyFrame && frame.type != 'key') {
        tranformer.requestKeyFrame();
        return;
    }
    ...
}
function createTransformer(transform)
{
    return new TransformStream({ transform :
        frame => encryptFrame(frame, transformer)
    });
}
```

#48 - WebRTC insertable streams - Evaluation

Expose streams or frames in workers

- API can expose one or the other, no change needed in window API

Expose a feedback channel

- API can be extended to support it: new API, stream…
- API can be extended to expose further knobs: requestKeyFrame...
    - Consistent to expose this API where the frame processing happens

Evaluate API complexity

- Proposed window interface is simple
    - Meaningful construct, easy to understand, extensible
- Worker interface can expose more or less JS constructs
    - Need to find the right tradeoff

#48 - WebRTC insertable streams - Proposal

Proposal 1

- Adopt transform based API in windows environment

Proposal 2

- Adopt SFrame native transform

Proposal 3

- Continue API design for script transforms
  - Core set of features for simple transforms
  - Feedback control handling

Cropping MediaStreamTracks

Elad Alon

Would introducing one specific video-editing capability in the browser actually *make sense*?

Previously on WEBRTC WG, 90210

The benefits and drawbacks of adding video-editing capabilities to the browser have been discussed in the past. A new argument needs to be made, in order for this discussion to be worth our time.

I believe I have such an argument, which applies to one particular video-editing capability - cropping.

<u>The Browser's Mandate [Citation Needed]</u>

- The browser ensures applications can accomplish reasonable tasks well.
- Good applications give the user good guarantees.
- When the application has no reasonable way of making good guarantees, it is the browser's responsibility to pave the way.
- (Citation: The Mozilla's Manifesto + my extrapolation.)

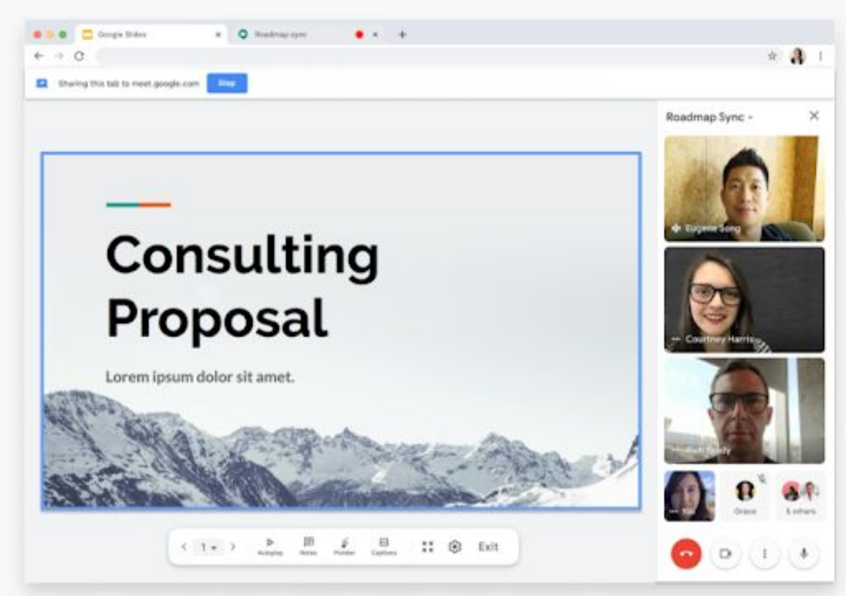
Interesting Use-Case

Consider this application for editing and displaying slides. It shares a cropped version of itself with remote users.

It may also display to the local user some private content, such as speaker notes.

The application's promise to the user:
The user's private content (e.g. speaker notes) will be cropped away. Consistently and robustly. Even if the user resizes the window or changes the zoom-level. Not a single frame containing private information will ever be shared remotely.

The browser must ensure the application has the means to deliver on such a reasonable promise.



This Particular Promise is Hard to Keep

- Whenever the window size changes, DOM elements get shifted around, and the application must change the video-cropping parameters.
- The application's update of the video's crop-parameters must never be late. Not even a single incorrectly-cropped frame should ever be shared remotely.
- PeerConnections are not synchronized with the JavaScript event-loop.

Insufficient “Solution” - OnResize Event Handlers

Registering OnResize event handlers - not a solution.

- The media flow (capture -> PeerConnection) is not synchronized with the JavaScript event loop.
- The problem is further complicated when content on the screen spans multiple origins, requiring postMessage to be used in order to get elements’ coordinates.

Insufficient "Solution" - Buffering

The application could buffer the frames. It can wait to ensure no resize-event is received shortly after a frame is captured, then consume the frame by forwarding it to a PeerConnection.

This is **not** a good solution. It would introduce delay. (Noteworthy: the shorter the delay - the less robust the guarantee by the application.)

Insufficient “Solution” - requestVideoFrameCallback

We cannot rely on requestVideoFrameCallback() to robustly provide privacy-sensitive functionality. Quote:

> `Since requestVideoFrameCallback() runs on the main thread, but, under the hood, video compositing happens on the compositor thread, everything from this API is a best effort, and we do not offer any strict guarantees.`

Additionally, calculating and re-calculating the crop-coordinates of content is problematic if the content in question is fully/partially from a cross-origin frame, in which case postMessage() needs to be used to communicate these coordinates across frame boundaries.

Conclusion: The Browser Must Intervene

The browser should provide the application either of the following:

1. pause-capture-on-resize-or-zoom-or-scroll-or-...
   - Overly complex API required to be useful. (Too many complications to mention here. Surmountable, but inelegant and unwieldy.)
2. crop-to-div
   - Browser continuously crops to a given DIV element, whatever its current coordinates happen to be in any given frame.
   - Similar to the much-requested HTML-element-capture, but obscured content is not captured, and overlaid content is captured, resulting in more user-friendly, privacy-respecting behavior.

Shape of APIs to Come

Avoiding particulars, the general shape we envision for the API is:

- Start with a MediaStream from an ongoing capture of the current tab.
- The MediaStreamTrack can have a crop-region applied to it.
- The region is specified by referencing a DIV (or other HTML element).
- For each frame capture, the browser will compute the DIV's coordinates, and crop to those coordinates.
- Note that pre-crop, the content spanned the entire tab. We therefore wish to allow an arbitrary crop, meaning we'd like to be able to crop to a DIV on any frame, from any frame. To achieve this, we'll reference the DIV using a name given in an HTML attribute. Bikeshedding notwithstanding, something along the lines of <div ... crop_id='some_name'>. This way, 'some_name' can be passed around using postMessage(), and any frame can crop to any DIV, if both frames cooperate.

Code Sample

```
<div crop_id='b9eb9d62'> content… </div>

let ms = navigator.mediaDevices.thisTabMedia();
let mst = await ms.getVideoTracks();
mst.cropTo('b9eb9d62');  // Or programmatically.
```

Note that the crop ID ‘b9eb9d62’ is either defined in the context in which cropTo() is called, or it can be transmitted there via postMessage(), etc.

It is up to debate whether the crop_id should be (a) an unguessable token or (b) a user-defined string, with any simple string being fair game.

Issues for Discussion Today

- WebRTC Extensions
    - Issue 52: Invalid TURN credentials: What Function Should Fail? (Henrik)
    - Issue 63: Enabling opus stereo audio without SDP munging (stereo=1) (Henrik)
    - Issue 230: Add support for WebRTC Data Channel in Worker (Youenn)

Issue 52: Invalid TURN credentials: What Function Should Fail? (Henrik)

TURN credentials are set with pc.setConfiguration(). For non-parse errors like invalid credentials or unable to reach host, errors would only be discovered later.

Problem: Not clear if/where invalid TURN credential failures are surfaced.

pc.onicecandidateerror already covers unable to reach server:

> `If no host candidate can reach the server, errorCode will be set to the value 701 which is outside the STUN error code range. This error is only fired once per server URL while in the RTCIceGatheringState of "gathering".`

Proposal:

- Parse-error: throw at setConfiguration(). Non-parse errors:
  Fire pc.onicecandidateerror with errorCode:701 for invalid TURN credentials.
  - Alternative: new errorCode 702?

Issue 63: Enabling opus stereo audio without SDP munging (stereo=1) (Henrik)

SDP munging is currently required to send stereo audio.

- In SDP, “stereo=1” means “I am OK with *receiving* stereo”.
  No stereo line or “stereo=0” means “I prefer to *receive* mono”.
- Regardless of stereo attribute, opus decoders MUST support stereo.

Problem:

- We currently don’t specify stereo, meaning we default to mono, and there is no API to control this.
- I think Chromium’s SDP munging turns on stereo for *sending* at setLocalDescription() when SDP munging to say “I am OK with *receiving* stereo”? This is backwards!
  (The encoder also does no care about MediaStreamTrack’s channelCount?)

Issue 63: Enabling opus stereo audio without SDP munging (stereo=1) (Henrik)

Proposal:

- Make stereo=1 the default.
- Channels to send: `min(track's channelCount, stereo attribute)`

Q: What if I want mono? Do I have to SDP munge stereo=0?

A: No, use getUserMedia({audio:{channelCount:1}}), WebAudio, etc.

Follow-up:

- Do we need to specify which channelCount to default to? If the default is 1, stereo would become opt-in using channelCount:2

Issue 230: Add support for WebRTC Data Channel in Worker

Web sites do use data channel to transmit data but process the data in workers

- Conferencing: Zoom
- Game streaming: parsec, XCloud
- Remote desktop: parsec
- Audio/video low latency transmission, receiving and sending

Potential solution: **make data channels transferable**

- Create data channels as done today
- Transfer data channel to audio worklet/video worker

Reduced problem scope

- No solution to the persistent data channel in shared worker use cases
- Cannot easily share the same data channel object between workers

Issue 230: Add support for WebRTC Data Channel in Worker

What is needed to make data channels transferable?

- A transfer algorithm (Limit to 'opened' channels?)
- A 'neutered' data channel behavior (similar to closed?)

Specification check

- No changes to creation/closing algorithms, methods definitions
- Minor changes to other algorithms (6.2.4 to 6.2.7)
- Garbage collection handled as part of transfer algorithm

Implementation check (based on webrtc.org code base)

- No change needed to allow processing data without hitting main thread
- Feasible to directly go from network thread to worker thread

Issue 230: Add support for WebRTC Data Channel in Worker

Conclusion

- Transferring data channels can help existing web applications
- Reduced complexity compared to creating data channels in workers

Alternative

- Apply WebSocketStream to RTCDataChannel
    - ReadableStream/WritableStream getters
- Piggy back on transferable streams to transfer data processing to workers

Is there interest in any of these possibilities?

For extra credit

Name this bird!

Thank you

Special thanks to:

WG Participants, Editors & Chairs

The bird