W3C WebRTC WG Meeting

February 22, 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 2nd interim meeting of 2021 of the W3C WebRTC WG!
  - During this meeting, we will discuss Testing, Media Capture, WebRTC Extensions and Insertable Streams.

About this Virtual Meeting

- Meeting info:
  - https://www.w3.org/2011/04/webrtc/wiki/February_22_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

- Testing
- Media Capture & Streams
    - Issue 529: Origin Isolation (Jan-Ivar)
- Media Capture & Streams Extensions
    - Issue 16: Transfer MediaStreamTrack (Youenn)
- 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 64: Transferable RTCDataChannel (Youenn)
- WebRTC Insertable Streams
    - Issue 48: RTC transforms in workers (Youenn)

WebRTC Testing Going Forward

- What will we need to test?
- How do we achieve better test coverage?

What Will We Need to Test?

- Work related to WebRTC PeerConnection (WebRTC-PC):
  - WebRTC-Stats
  - WebRTC-Priority
  - WebRTC DSCP
- Extensions to WebRTC PeerConnection:
  - WebRTC Extensions
  - WebRTC-SVC
  - Insertable Streams
- Extensions to Capture, Streams and Output-related specifications, including:
  - MediaStreamTrack Insertable Streams,
  - Media Capture and Streams Extensions
  - MediaCapture Depth Stream Extensions (recently revived)
  - Content-Hints
- Standalone specifications, not related to PeerConnection or Capture, such as:
  - WebRTC-ICE (in the W3C WebRTC WG)
  - WebTransport (in the W3C WebTransport WG)
  - WebRTC-QUIC (in the ORTC CG)
  - WebCodecs (being adopted by the W3C Media WG)

What Challenges Do We Face?

- WebRTC-Stats: Testing whether the stats are correct, not just whether they are retrievable.
- WebRTC-Priority, WebRTC DSCP, Content-Hints: Testing whether what is requested is provided, not just whether attributes can be set and retrieved.
  - Example: ‘text’ content-hint should activate AV1 content coding tools.
- WebRTC Extensions: Testing whether the requested RTP header extensions (and encryption) are delivered.
- WebRTC-SVC, Insertable Streams: Testing whether it works end-to-end (e.g. whether what is encoded/encrypted can be decrypted/decoded after SFU processing).
- MediaStreamTrack Insertable Streams: Performance testing.
- Media Capture and Streams Extensions: testing application backward compatibility.

A Harbinger: AV1/WebRTC Integration Testing

- Integration of AV1 with WebRTC required development of a custom end-to-end test suite.
  - Required to demonstrate correct operation of webrtclib endpoints communicating via an SFU (Medooze) implementing the Dependency Descriptor RTP header extension.
  - Many issues found in interactions between WebRTC RTP stack, AV1, DD header extension, SVC scalability modes, SFU behavior, E2E encryption, decoder filtering, etc.
  - Javascript tests in-progress, (complete JS API support not yet ready)
  - Test coverage: https://cosmosoftware.github.io/av1-rtp-spec-html/
- Will the AV1 E2E test suite be run on an ongoing basis, to guard against regressions?
  - The answer appears to be “no”. Uh oh...

Some Questions

- How do we ensure that recently added features do not regress?
- Should we add server support to WPT tests (as W3C WebTransport WG has done)?
- If so, what does the server code look like?
  - How would this improve coverage?

A Modest Proposal

- Make incremental progress
- Inter-browser tests are hard to get into browsers' submit-time checks
- Extensive server logic is hard to maintain inside shared services like the WPT tool
- Proposal: Build a content reflector that speaks the WebRTC stack - no more

Proposal: an aiortc-based server

- Server to act as WebRTC endpoint
  - as stupid^Wminimal as possible
- https://github.com/jlaine/aiortc-wpt-demo/
  - only 60 lines of code
  - terminates STUN + DTLS, decrypts SRTP, echoes RTP/RTCP packets via WebSockets
  - uses RTCPeerConnection from aiortc
  - WPT already uses aioquic for quic tests
- Simple tests!
  - create a peerconnection, connect WebSocket
  - get raw packets
    - parsing RTP/RTCP/SCTP...

aiortc-based server

```python
async def handle_rtp_data(websocket, data: bytes, arrival_time_ms: int) -> None:
    await websocket.send_bytes(data)


class Endpoint(WebSocketEndpoint):
    encoding = "json"

    async def on_connect(self, websocket):
        websocket.state.pc = RTCPeerConnection()

        await websocket.accept()

    async def on_receive(self, websocket, message):
        pc = websocket.state.pc
        offer = RTCSessionDescription(sdp=message["sdp"], type=message["type"])

        # handle offer
        await pc.setRemoteDescription(offer)

        # monkey-patch RTCDtlsTransport
        for transceiver in pc.getTransceivers():
            transport = transceiver.receiver.transport
            transport._handle_rtp_data = functools.partial(handle_rtp_data, websocket)

        # create answer
        answer = await pc.createAnswer()
        await pc.setLocalDescription(answer)

        # send answer
        await websocket.send_json(
            {"sdp": pc.localDescription.sdp, "type": pc.localDescription.type}
        )

    async def on_disconnect(self, websocket, close_code):
        await websocket.state.pc.close()

```

aiortc-based server: the test

https://github.com/jlaine/aiortc-wpt-demo/pull/1

```
const ws = await connect(pc);
t.add_cleanup(() => ws.close());

// Wait for a video frame. The last packet in the frame will have the marker bit set.
// Note that this doesn't deal well with conditions like missing last packets, however
// waiting for a timestamp change requires waiting a bit longer. This is just an example :-)
const frame = await (new Promise((resolve) => {
    const buffer = [];
    ws.addEventListener('message', function listener(message) {
        if (isRTCP(message.data)) {
            return;
        }
        const rtpData = new RTP(message.data);
        buffer.push(rtpData);
        if (rtpData.marker) {
            resolve(buffer);
            ws.removeEventListener('message', listener);
        }
    });
}));
// Run some assertions such as trying to reassemble a frame.
// Or just print a table.
console.table(frame);
```

aiortc-based server: the test

(index)	version	padding	extension	marker	payloadType	sequenceNum…	timestamp	synchroniza…	header	headerExten…	contributin…	payload
0	2	1	1	0	97	29577	903138129	512501542	DataView(20)	Array(1)	Array(0)	DataView(0)
1	2	0	1	1	96	963	903138129	1747254558	DataView(24)	Array(2)	Array(0)	DataView(55…

```
▼Array(2)
 ▼0:
  ▶contributingSources: []
   extension: 1
  ▶header: DataView(20) {}
  ▼headerExtensions: Array(1)
   ▶0: {id: 2, data: DataView(3)}
    length: 1
   ▶__proto__: Array(0)
   marker: 0
   padding: 1
  ▶payload: DataView(0) {}
   payloadType: 97
   sequenceNumber: 29577
   synchronizationSource: 512501542
   timestamp: 903138129
   version: 2
  ▶__proto__: Object
 ▼1:
  ▶contributingSources: []
   extension: 1
  ▶header: DataView(24) {}
  ▼headerExtensions: Array(2)
   ▶0: {id: 2, data: DataView(3)}
   ▶1: {id: 9, data: DataView(1)}
    length: 2
   ▶__proto__: Array(0)
   marker: 1
   padding: 0
  ▶payload: DataView(555) {}
   payloadType: 96
   sequenceNumber: 963
   synchronizationSource: 1747254558
   timestamp: 903138129
   version: 2
```

aiortc-based server: next steps

- O/A RTCPeerConnection from the server
  - Test might want to generate SDP themselves
  - reduce server to ORTC ICETransport + DTLSTransport
- Sending packets from the test
  - Send PLI, expect keyframe within X milliseconds.
- Needs to generate some RTCP
  - or bandwidth will be stuck at 300kbps

Issues for Discussion Today

- Media Capture & Streams
  - Issue 529: Origin Isolation (Jan-Ivar)
- Media Capture & Streams Extensions
  - Issue 16: Transfer MediaStreamTrack (Youenn)

Issue 529: Origin Isolation (Jan-Ivar)

`video.src = "https://cross-orig.in/v.mp4"`

`drawImage()`

`fillText()`

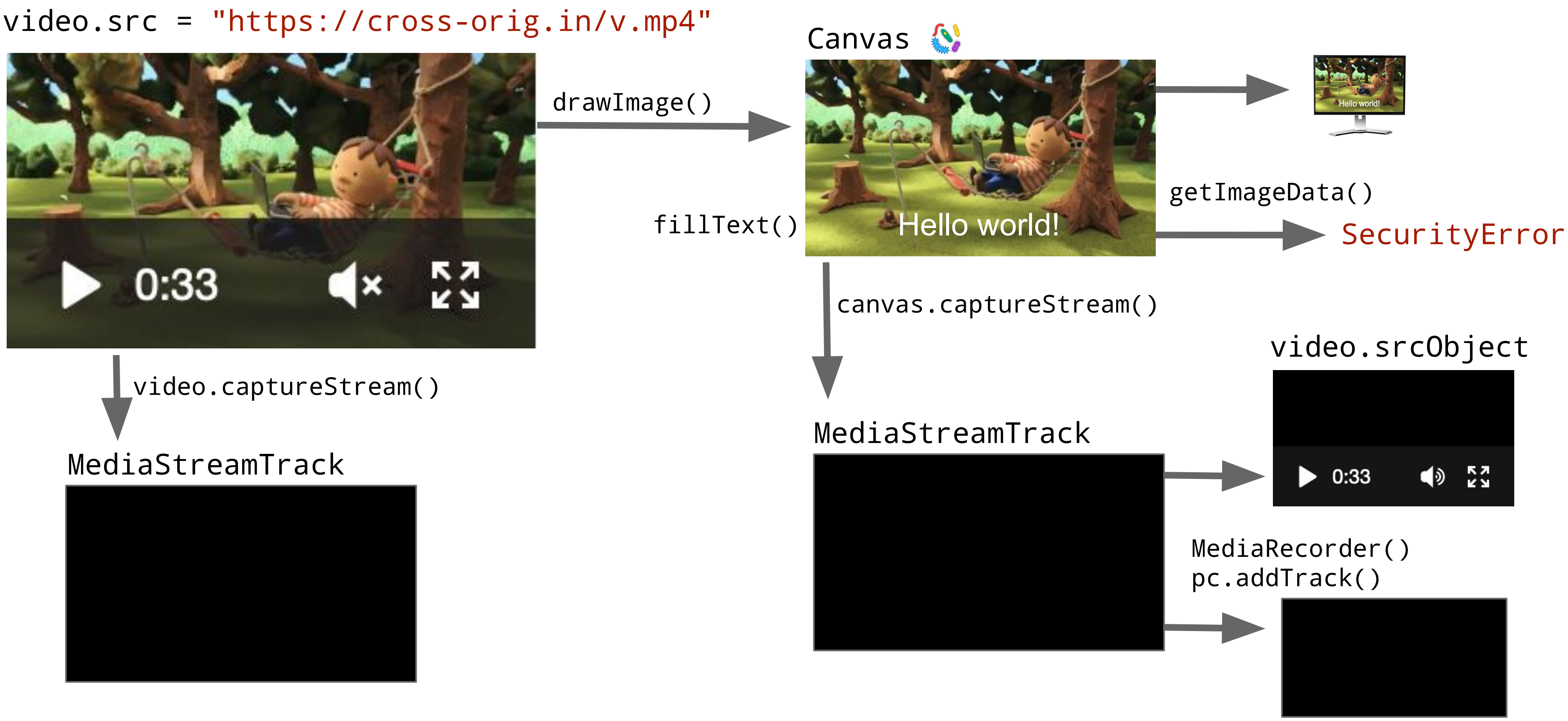
Canvas

`getImageData()` → `SecurityError`

Tainted Canvas

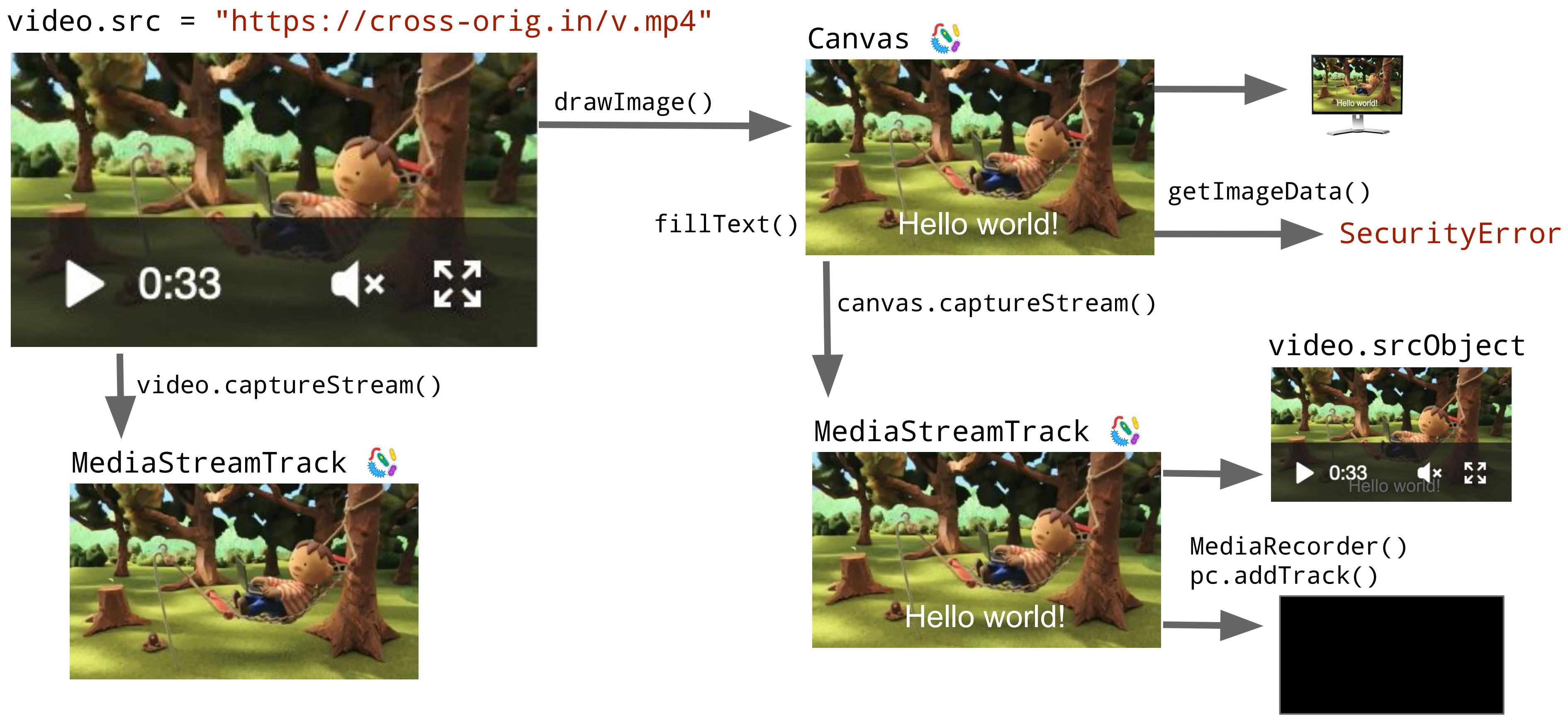
From https://jsfiddle.net/jib1/cwzdtpgk/

Issue 529: Origin Isolation (Jan-Ivar)



From https://jsfiddle.net/jib1/cwzdtpgk/

Issue 529: Origin Isolation (Jan-Ivar)



Issue 529: Origin Isolation (Jan-Ivar)

Open issue w3c/mediacapture-fromelement#83 Taint, not mute cross-origin tracks

Why? Symmetry with tainted canvas. Authorization model; track origin for security.

Tainted MediaStreamTrack use cases:

- Captioning/compositing of *cross-origin* videos in canvas using mediaelement playback features like picture-in-picture, chromecast, airplay. (performance?)
- peerIdentity or a future e2ee replacement?
- Future MediaStreamTrack manipulation features (cropping)?
- Transferable MediaStreamTracks to cross origins?

Raw media access is limited to same-origin media
If we expose MSTs to other origins, then s/taint/mark track with origin(s)/ regardless

Issue 529: Origin Isolation (Jan-Ivar)

Few enticing short-term use cases. Allow more exploration?

Proposal A: *“If the User Agent supports tainted MediaStreamTracks, then all sinks of MediaStreamTrack MUST protect the data of cross-origin media in said tracks from being exposed to the application, e.g. by replacing the data with muted output.”*

Proposal B: Band-aid this in mediacapture-fromelement somehow. *“Tainted tracks can only be consumed in HTMLMediaElement sinks. ...”*

Proposal C: Say nothing.

Issue 16: Transfer MediaStreamTrack (Youenn)

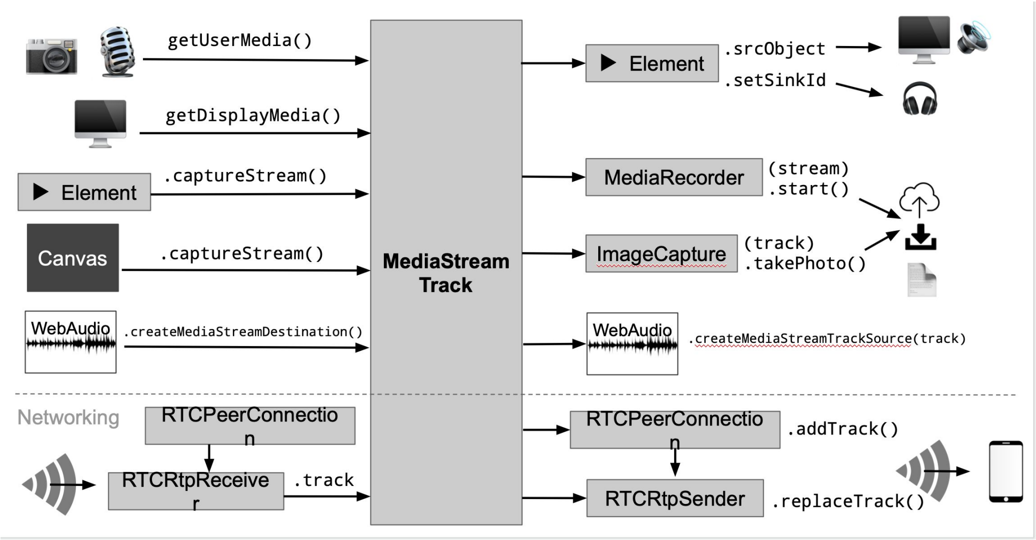
Why should we do it?

- Create a track in the IFrame that makes sense
  - Transport the track in a different IFrame that makes more sense
- Do processing in a worker
  - MediaStreamTrack -> WebCodec -> RTCDataChannel
  - RTCDataChannel -> WebCodec -> MediaStreamTrack

Issue 16: Transfer MediaStreamTrack (Youenn)

Can we do it? Cross process?

- MediaStreamTrack is already flowing out of process
  - Content is often produced out of process
  - Content is often processed out of process
- Browsers already do this, and efficiently



Issue 16: Transfer MediaStreamTrack (Youenn)

How can it be done?

What is needed is very similar to RTCDataChannel

- Transfer algorithm
- 'Neutered' behavior
- Lifetime of transferred MediaStreamTrack tied to creation context
  - Like streams, like data channel

Issue 16: Transfer MediaStreamTrack (Youenn)

Alternative

- Use mediacapture insertable streams to shim transferable MediaStreamTracks
  - MediaStreamTrack -> Streams
  - Transfer streams
  - Streams -> MediaStreamTrack

Potential downsides

- More difficult to optimize this code path than transferring a MediaStreamTrack
- Not the same support as transferable streams out of the box
  - Especially for capture tracks: getSettings, applyConstraints…

Should we do it?

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 64: Transferable RTCDataChannel (Youenn)
- WebRTC Insertable Streams
  - Issue 48: RTC transforms in workers (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)

In Chromium, 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.

Issue 64: Transferable RTCDataChannel (1/3)

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

- Conferencing: Zoom
- Game streaming/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 64: Transferable RTCDataChannel (2/3)

What is needed to make data channels transferable?

- Transfer algorithm
- 'Neutered' data channel behavior

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
- Transfer algorithm similar to streams 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 64: Transferable RTCDataChannel (3/3)

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?

[Issue 48](#): Expose RTC transforms in workers

Expose counterpart of RTCRtpScriptTransform in workers

- Named RTCRtpScriptTransformer

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

const sender = pc.addTrack(track, stream)
sender.transform = new RTCRtpScriptTransform(worker, "myTransform")
```

```
// worker-module.js - event based variant
onrtctransform = (e) => {
  const transformer = e.transformer
  if (transformer.options === "myTransform") {
    ...
    return
  }
}
```

```
// worker-module.js - callback based variant
class MyTransform extends RTCRtpScriptTransformer {
  constructor() {
    ...
  }
}
self.registerRTCRtpScriptTransformer("myTransform",
  MyTransform);
```

- **Proposal**
  - Adopt 'rtctransform' event

Issue 48: RTCRtpScriptTransformer API - option 1

RTCRtpScriptTransformer as a dictionary with Readable/Writable stream members

```
// worker-module.js - event based variant
onrtctransform = (e) => {
  const transformer = e.transformer;
  if (transformer.options === "myTransform") {
    transformer.readable
        .pipeThrough(createTransform(transformer))
        .pipeTo(transformer.writable);
    return
  }
}
```

```
// WebDL
dictionary RTCTransformEventData {
  ReadableStream readable;
  WritableStream writable;
  any options;
}
```

```
[Exposed=Worker]
interface RTCTransformerEvent : Event {
  readonly attribute RTCTransformEventData transformer;
}
```

Pros: small API surface

Issue 48: RTCRtpScriptTransformer API - option 2

RTCRtpScriptTransformer as an object with Readable/Writable stream attributes

```
// worker-module.js - event based variant
onrtctransform = (e) => {
  const transformer = e.transformer;
  if (transformer.options === "myTransform") {
    transformer.readable
        .pipeThrough(createTransform(transformer))
        .pipeTo(transformer.writable);
    return
  }
}
```

```
// WebDL
[Exposed=Worker]
class RTCScriptTransformer {
  readonly attribute ReadableStream readable;
  readonly attribute WritableStream writable;
  readonly attribute any options;
}
```

```
[Exposed=Worker]
interface RTCTransformerEvent : Event {
  readonly attribute RTCScriptTransformer transformer;
}
```

Pros: easy to expose additional API surface

- State getters: sender/receiver, bitrate...
- Mutators: request a key frame

Issue 48: RTCRtpScriptTransformer API - option 3

RTCRtpScriptTransformer as an object with events or callbacks

```
// worker-module.js - event based variant
onrtctransform = (e) => {
  const transformer = e.transformer;
  if (transformer.options === "myTransform") {
    transformer.onframe = (event) =>
transformer.enqueue(encrypFrame(event.data))
    return
  }
}
```

```
// WebDL
[Exposed=Worker]
class RTCScriptTransformer {
  attribute EventHandler onframe;
  void enqueue(RTCEncodedFrame frame);
  readonly attribute any options;
}
```

```
[Exposed=Worker]
interface RTCTransformerEvent : Event {
  readonly attribute RTCScriptTransformer transformer;
}
```

Close to option 2

- Option 2 and 3 are shimable one with the other
- Other similar options available (TransformerTransformCallback)

Issue 48: RTCRtpScriptTransformer API - tentative conclusion

Expose dictionary or interface

- Interface is more extensible
- **Proposal**
  - Expose a RTCRtpScriptTransformer interface

Expose ReadableStream+WritableStream or event+method

- Existing transform ecosystem (SFrameTransform, TransformStream)
- Existing methods that are not useful (dangerous?)
  - ReadableStream.cancel, ReadableStreamWriter.close
- **Proposal**
  - Stick with ReadableStream/WritableStream for now
  - Continue investigating pros and cons

For extra credit

Name the bird!

Thank you

Special thanks to:

WG Participants, Editors & Chairs

The bird