W3C WebRTC WG Meeting

May 21, 2024
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 2024 interim meeting of the W3C WebRTC WG, at which we will cover:
  - Captured Surface Control, Encoded Transform, Mediacapture-main, P2P API, RtpTransport
- Future meetings:
  - June 18
  - July 16

About this Virtual Meeting

- Meeting info:
  - https://www.w3.org/2011/04/webrtc/wiki/May_21_2024
- Link to latest drafts:
  - https://w3c.github.io/mediacapture-main/
  - https://w3c.github.io/mediacapture-extensions/
  - https://w3c.github.io/mediacapture-image/
  - https://w3c.github.io/mediacapture-output/
  - https://w3c.github.io/mediacapture-screen-share/
  - https://w3c.github.io/mediacapture-record/
  - https://w3c.github.io/webrtc-pc/
  - https://w3c.github.io/webrtc-extensions/
  - https://w3c.github.io/webrtc-stats/
  - https://w3c.github.io/mst-content-hint/
  - https://w3c.github.io/webrtc-priority/
  - https://w3c.github.io/webrtc-nv-use-cases/
  - https://github.com/w3c/webrtc-encoded-transform
  - https://github.com/w3c/mediacapture-transform
  - https://github.com/w3c/webrtc-svc
  - https://github.com/w3c/webrtc-ice
- Link to Slides has been published on WG wiki
- Scribe? IRC http://irc.w3.org/ Channel: #webrtc
- The meeting is (still) being recorded. The recording will be public.
- Volunteers for note taking?

W3C Code of Conduct

- This meeting operates under W3C Code of Ethics and Professional Conduct

- We're all passionate about improving WebRTC and the Web, but let's all keep the conversations cordial and professional

Virtual Interim Meeting Tips

This session is (still) being recorded

- **Click Raise hand to get into the speaker queue.**
- **Click Lower hand to get out of the speaker queue.**
- **Please wait for microphone access to be granted before speaking.**
- **If you jump the speaker queue, you will be muted.**
- **Please use headphones when speaking to avoid echo.**
- **Please state your full name before speaking.**
- **Poll mechanism may be used to gauge the “sense of the room”.**

Understanding Document Status

- Hosting within the W3C repo does ***not*** imply adoption by the WG.
  - WG adoption requires a Call for Adoption (CfA) on the mailing list.
- Editor’s drafts do ***not*** represent WG consensus.
  - WG drafts ***do*** imply consensus, once they’re confirmed by a Call for Consensus (CfC) on the mailing list.
  - Possible to merge PRs that may lack consensus, if a note is attached indicating controversy.

Issues for Discussion Today

- 08:10 - 08:40 AM Captured Surface Control (Elad)
- 08:40 - 08:55 AM WebRTC-Encoded-Transform (Florent)
- 08:55 - 09:10 AM Mediacapture-main (Jan-Ivar)
- 09:10 - 09:30 AM Local P2P API (Anssi Kostiainen, Michiel De Backker)
- 09:30 - 09:50 AM RtpTransport (Peter Thatcher)
- 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.

Captured Surface Control (Elad Alon)

Start Time: 08:10 AM

End Time: 08:40 AM

Problem description

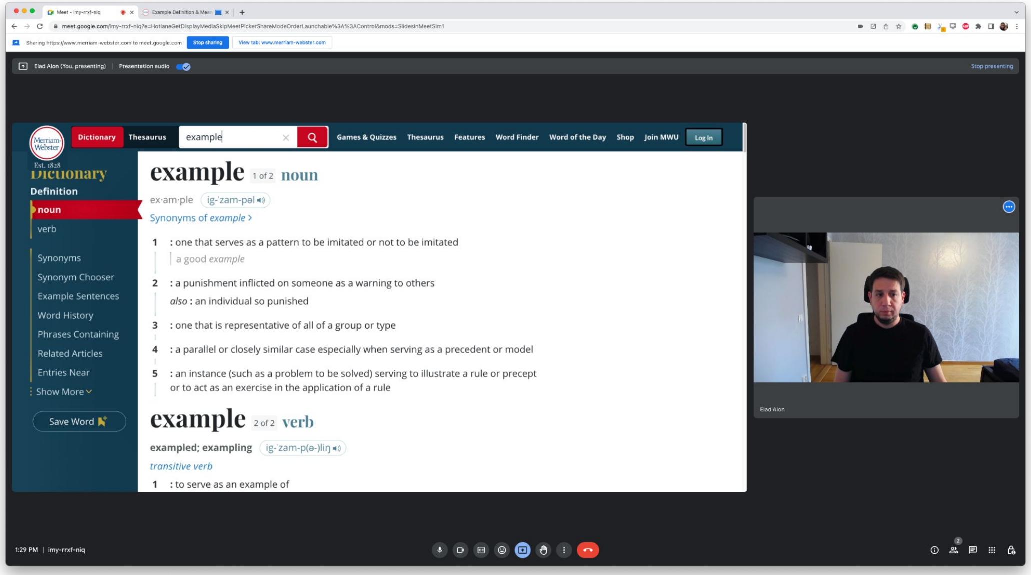
A user is in a video call and shares a tab.

How does the user…

- …scroll the captured tab?
- …change the zoom level?

If the user focused the captured tab…

- …how would the user see remote participants?
- …see annotations and additional content (e.g. a timer)?
- How would the user interact with VC app's controls?



example 1 of 2 noun

ex·am·ple ig-ˈzam-pəl

Synonyms of *example* >

1 : one that serves as a pattern to be imitated or not to be imitated

a good example

2 : a punishment inflicted on someone as a warning to others

also : an individual so punished

3 : one that is representative of all of a group or type

4 : a parallel or closely similar case especially when serving as a precedent or model

5 : an instance (such as a problem to be solved) serving to illustrate a rule or precept or to act as an exercise in the application of a rule

example 2 of 2 verb

exampled; exampling ig-ˈzam-p(ə-)liŋ

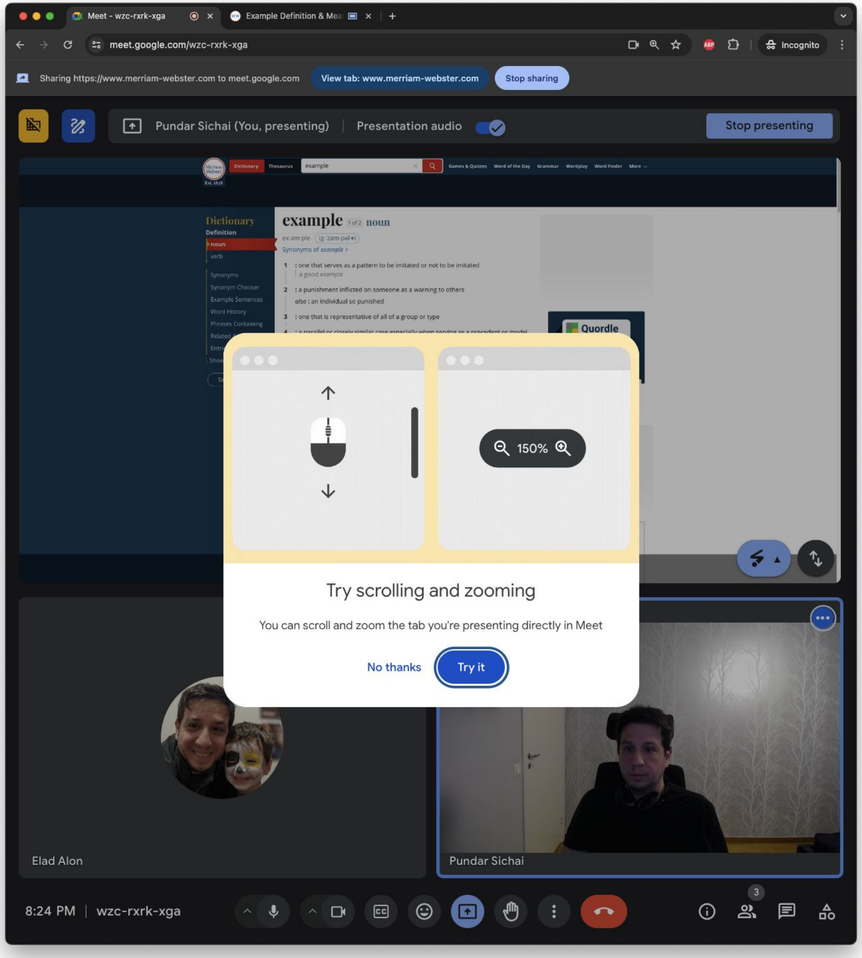
transitive verb

1 : to serve as an example of

Proposal

Produce an API allowing applications to scroll and zoom captured-tabs. (Possibly extend to captured-windows in the future.)

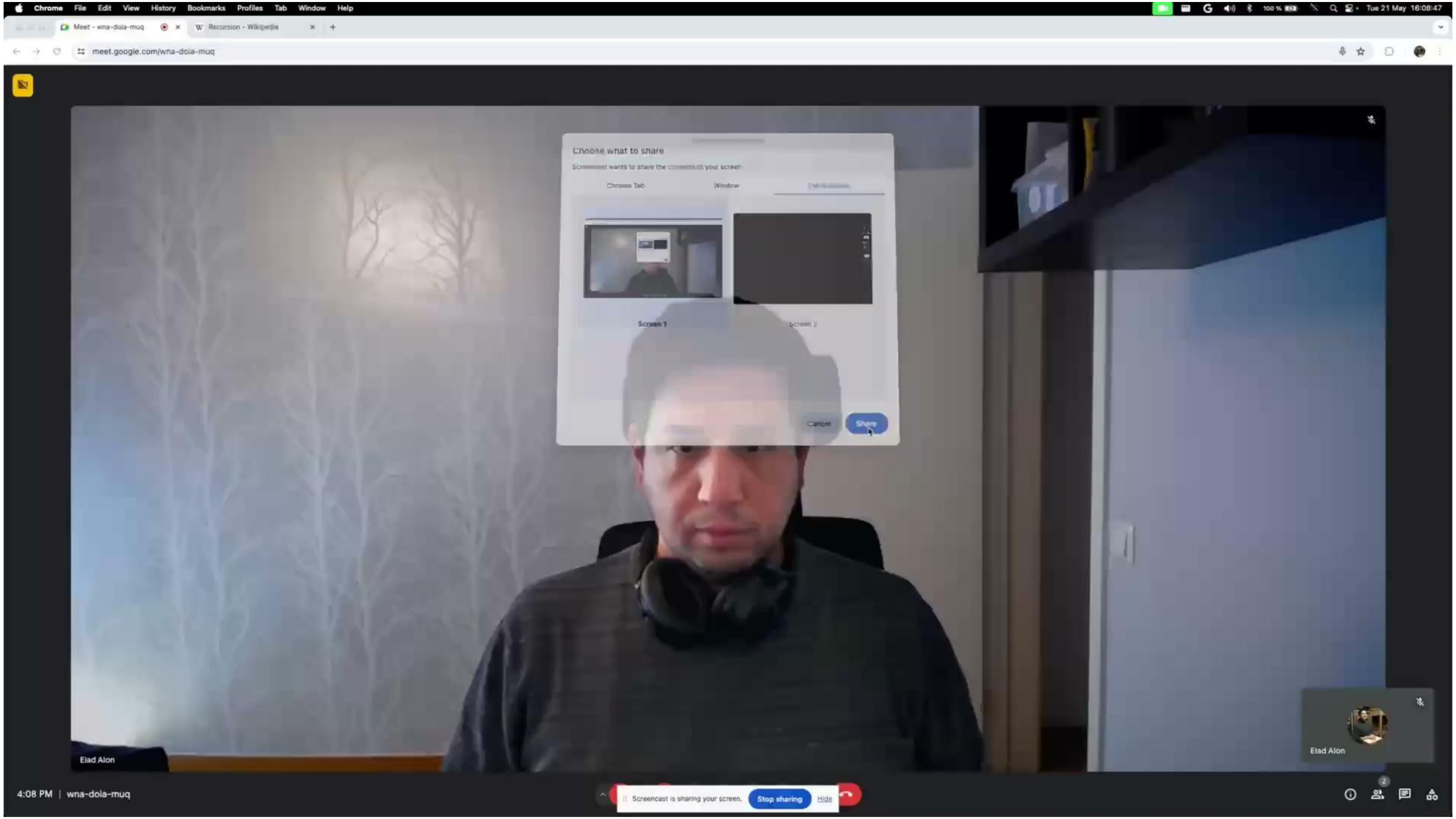
Applications can then communicate this to the user by building their own app-level user onboarding and controls.


Meet - wzc-rxrk-xga
Example Definition & Mea
meet.google.com/wzc-rxrk-xga
Incognito
Sharing https://www.merriam-webster.com to meet.google.com
View tab: www.merriam-webster.com
Stop sharing
Pundar Sichai (You, presenting)
Presentation audio
Stop presenting
150%
Elad Alon
Pundar Sichai
8:24 PM
wzc-rxrk-xga

Try scrolling and zooming

You can scroll and zoom the tab you're presenting directly in Meet

No thanks | Try it

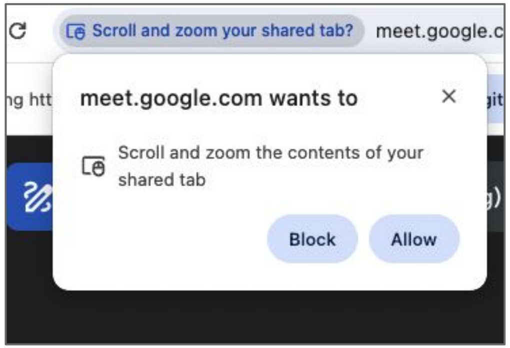


Screencast wants to share the contents of your screen

Screencast is sharing your screen.

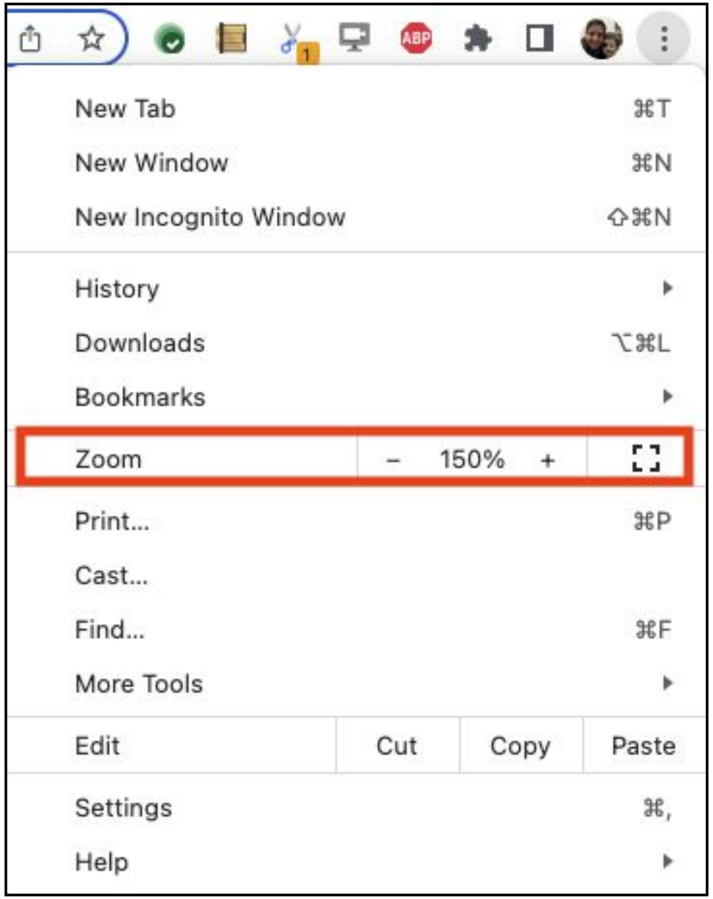
Sample usage 1: Obtain permission

```
const initialPermState = await navigator.permissions.query({
  name: 'captured-surface-control'
});

let hasCscPermission = (initialPermState.state === granted');
if (initialPermState.state === 'prompt') {
  startButton.hidden = false;
  startButton.addEventListener('click', async () => {
    const noOpWheelAction = {};
    await controller.sendWheel(noOpWheelAction);
    startButton.hidden = true;
    hasCscPermission = true;
  });
}

// Error handling - an exercise for the reader.
```


Scroll and zoom your shared tab?

meet.google.com wants to

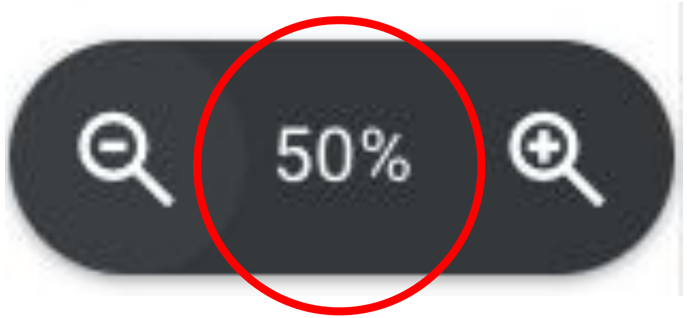
Scroll and zoom the contents of your shared tab


New Tab ⌘T
New Window ⌘N
New Incognito Window ⇧⌘N
History
Downloads ⌥⌘L
Bookmarks
Zoom – 150% +
Print... ⌘P
Cast...
Find... ⌘F
More Tools
Edit Cut Copy Paste
Settings ⌘,
Help

Sample usage 2: Zoom read-access

```
const zoomLabel = document.querySelector('#zoomLevelLabel');

// Read current zoom-level and expose it to the user.
zoomLabel.textContent = `${controller.getZoomLevel()}%`;

// Monitor changes to zoom-level, either through the app’s own
// control of the zoom, or through the user’s direct manipulation
// of the browser’s zoom-level for the captured tab.
controller.addEventListener('capturedzoomlevelchange', (event) => {
  zoomLabel.textContent = `${controller.getZoomLevel()}%`;
});
```

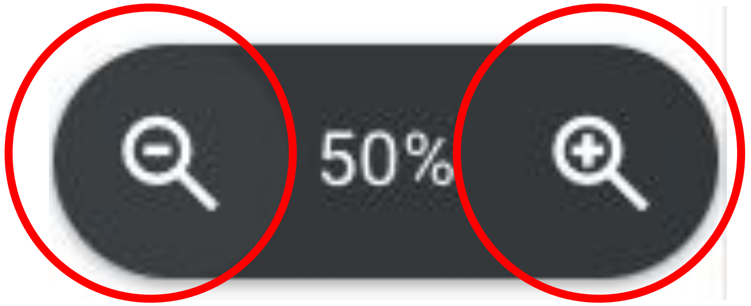
<u>Note:</u>
Read-access is synchronous and <u>not</u> permission-gated.

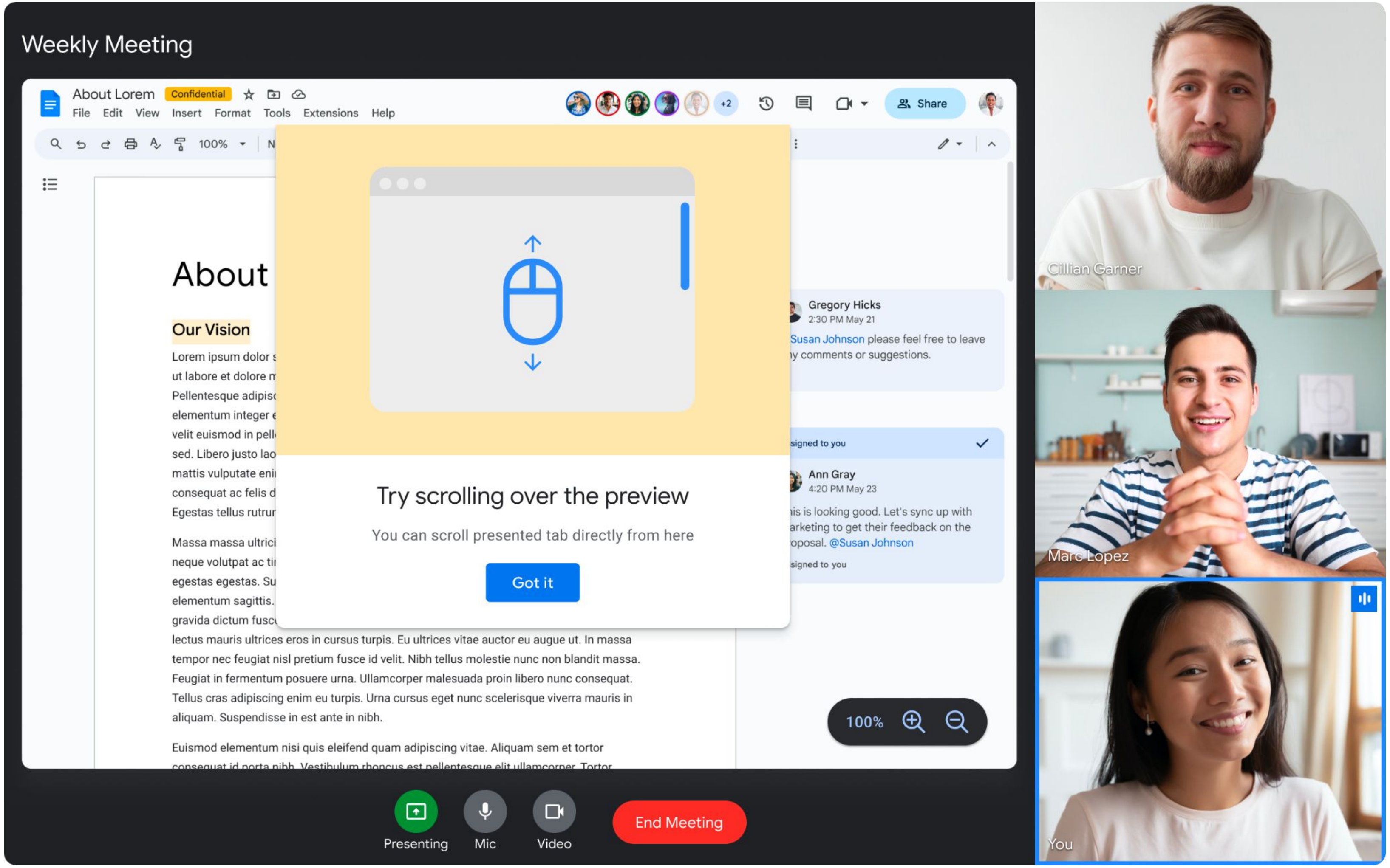


Sample usage 3: Zoom write-access

```
const zoomIncreaseButton = document.getElementById('zoomInButton');

zoomIncreaseButton.addEventListener('click', async (event) => {
  const levels = CaptureController.getSupportedZoomLevels();
  const index = levels.indexOf(controller.getZoomLevel());
  const newZoomLevel = levels[Math.min(index + 1, levels.length - 1)];

  try {
    await controller.setZoomLevel(newZoomLevel);
  } catch (error) {
    // Inspect the error.
    // ...
  }
});
```

zoomDecreaseButton() left as an exercise for the reader.





About

Our Vision

Lorem ipsum dolor s
ut labore et dolore m
Pellentesque adipisc
elementum integer e
velit euismod in pell
sed. Libero justo lao
mattis vulputate eni
consequat ac felis d
Egestas tellus rutrur

Massa massa ultrici
neque volutpat ac ti
egestas egestas. Su
elementum sagittis.
gravida dictum fusc
lectus mauris ultrices eros in cursus turpis. Eu ultrices vitae auctor eu augue ut. In massa tempor nec feugiat nisl pretium fusce id velit. Nibh tellus molestie nunc non blandit massa. Feugiat in fermentum posuere urna. Ullamcorper malesuada proin libero nunc consequat. Tellus cras adipiscing enim eu turpis. Urna cursus eget nunc scelerisque viverra mauris in aliquam. Suspendisse in est ante in nibh.

Euismod elementum nisi quis eleifend quam adipiscing vitae. Aliquam sem et tortor consequat id porta nibh. Vestibulum rhoncus est pellentesque elit ullamcorper. Tortor

Try scrolling over the preview

You can scroll presented tab directly from here

Got it

Gregory Hicks
2:30 PM May 21
Susan Johnson please feel free to leave
ny comments or suggestions.

signed to you ✓

Ann Gray
4:20 PM May 23
his is looking good. Let's sync up with
arketing to get their feedback on the
oposal. @Susan Johnson
signed to you

Sample usage 4: Scrolling

```
previewTile.addEventListener('wheel', async (event) => {
  const [x, y] = translateCoordinates(event.offsetX, event.offsetY);
  const [wheelDeltaX, wheelDeltaY] = [-event.deltaX, -event.deltaY];

  await controller.sendWheel({ x, y, wheelDeltaX, wheelDeltaY });
});


function translateCoordinates(offsetX, offsetY) {
  const previewDimensions = previewTile.getBoundingClientRect();
  const trackSettings = previewTile.srcObject.getVideoTracks()[0].getSettings();

  const x = trackSettings.width * offsetX / previewDimensions.width;
  const y = trackSettings.height * offsetY / previewDimensions.height;

  return [Math.floor(x), Math.floor(y)];
}
```

Discussion (End Time: 08:40)

-

WebRTC-Encoded-Transform (Florent)

Start Time: 08:40 AM

End Time: 08:55 AM

For Discussion Today

- Issue 225: Add captureTimestamp senderCaptureTimeOffset to the encoded frame metadata
- Issue 226: Expose RTCEncodedAudioFrame interface in Worklets

Issue 225: Add captureTimestamp senderCaptureTimeOffset to the encoded frame metadata

- RTCRtpContributingSource is extended in WebRTC-Extensions:
  - `captureTimestamp` **of type** **`DOMHighResTimeStamp`.**
  - `senderCaptureTimeOffset` **of type** **`DOMHighResTimeStamp`.**

- Proposing to add those fields to:
  - RTCEncodedAudioFrameMetadata
  - RTCEncodedVideoFrameMetadata

- Exposing value for locally captured frames
- Exposing value for received frames with abs-capture-time RTP header extension
- Tentative PR

Issue 226: Expose RTCEncodedAudioFrame interface in Worklets

- Goal is to receive audio frame data in an AudioWorklet for decoding using a WASM implementation.

- This could be done with an RTCRtpScriptTransformer in 2 ways:
  - Run a transformer on a Worker and transfer the frame data to an AudioWorklet.
    - Doesn't need any API change.
    - Leads to more JS invoked in a non real-time context first, bad for performance and reliability.
  - Run the transformer on an AudioWorklet
    - Requires API change
    - Less thread hopping

Issue [226](#): Expose RTCEncodedAudioFrame interface in Worklets

- API changes required to run a RTCRtpScriptTransformer in an AudioWorklet
  - Expose interfaces to AudioWorklet context
    - RTCRtpScriptTransformer
    - RTCEncodedAudioFrame
    - RTCTransformEvent
  - New constructor for RTCRtpScriptTransform that accepts a first argument of type AudioWorklet only if the sender is of kind “audio”
  - Add “onrtctransform” to AudioWorkletGlobalScope

- Possible extra-hardening of keyframe related functionality to throw when used on a sender of kind “audio” to prevent exposing video related interfaces.

Mediacapture-main (Jan-Ivar)

Start Time: 08:55 AM

End Time: 09:10 AM

For Discussion Today

- Issue 1003: repo name nit: it'd be nice if this were simply w3c/mediacapture

- Issue 966: Should devicechange fire when the device info changes?

Issue 1003: repo name nit: it'd be nice if this were simply w3c/mediacapture

Today we have:

- https://www.w3.org/TR/mediacapture-streams
- https://wpt.fyi/results/mediacapture-streams
- https://w3c.github.io/mediacapture-main

Proposal(s): change all (or just some) to:

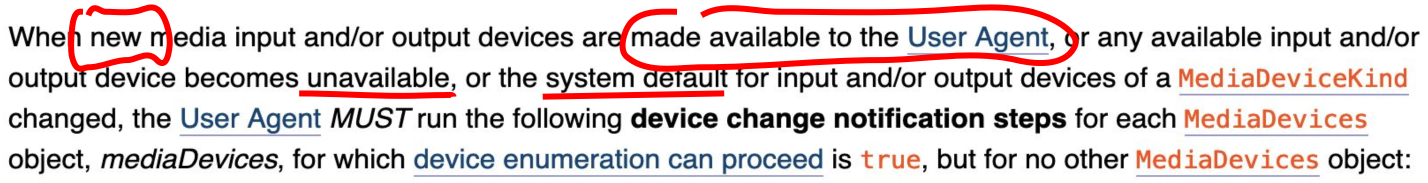
1. https://www.w3.org/TR/mediacapture
2. https://wpt.fyi/results/mediacapture
3. https://w3c.github.io/mediacapture

Related:

- How come https://w3c.github.io/webrtc-pc and https://www.w3.org/TR/webrtc ?

Issue 966: Should devicechange fire when the device info changes? (Jan-Ivar)

Recap from October: The spec normatively says:



When new media input and/or output devices are made available to the User Agent, or any available input and/or output device becomes unavailable, or the system default for input and/or output devices of a `MediaDeviceKind` changed, the User Agent *MUST* run the following **device change notification steps** for each `MediaDevices` object, *mediaDevices*, for which device enumeration can proceed is `true`, but for no other `MediaDevices` object:

This limits when the device change notification steps run to ***when OS changes happen***, limiting both when `devicechange` fires and when `mediaDevices.[[storedDeviceList]]` changes.

This is ***not*** the same as calling it every time enumerateDevices() would produce *different results*.

Safari violates this, firing it from set the device information exposure in getUserMedia(), IFF the application has previously called enumerateDevices() AND the user has >1 device/kind

Also, is an OS device label change covered under "new" media input? OP and others want this.

Competing uses cases: **Updating in-content UX** vs. **strong signal from user inserting a device**

Issue 966: Should devicechange fire when the device info changes? (Jan-Ivar)

Claim: A user inserting a media device at the start of a call is a strong signal.

But reliably detecting this signal is hard. A `devicechange` with a new secondary microphone or camera in the list immediately after getUserMedia() success no longer means it was inserted by the user

(We shouldn't rely on applications detecting known headset brands like AirPods: U see 'em; U use 'em!)

Do we need two new events?

- `deviceinfochange` fired for every delta, exposure or label change
- `deviceinserted` fired post-getUserMedia (even for pre-gUM insertion?)

Discuss (if 3 is too many, we can discuss altering the existing one to be one of the two)

Discussion (End Time: 09:10)

-

Local Peer-to-Peer API
(Anssi Kostiainen, Michiel De Backker)
Start Time: 09:10 AM
End Time: 09:30 AM

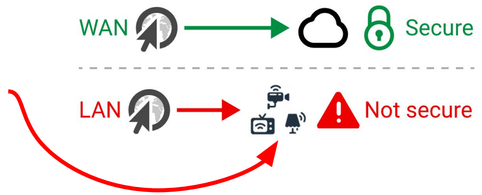
Background

Local Peer-to-Peer API?

New WICG incubation in prototyping
API to connect securely over a local communication medium, *without* the aid of a server in the middle

Motivation

⚠️ The local network is not a first-class citizen of the web



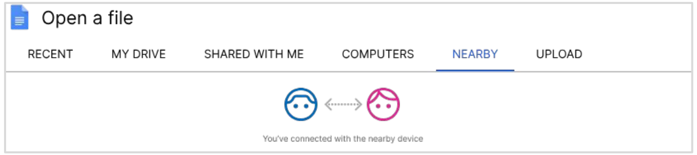
Goals

↔ Arbitrary bidirectional communication channel in the context of a ***local communication medium***:

🔎 Discover, request, and connect to peers
✉️ Send and receive data
🔒 Enable secure HTTPS connections

Use Cases

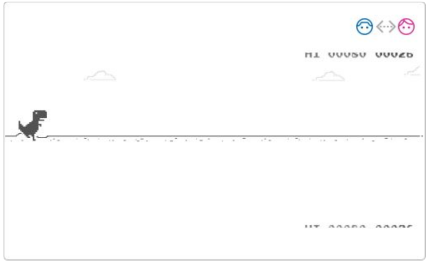
🧑↔🧑 Offline collaboration


Open a file

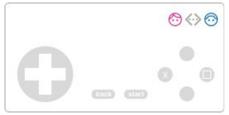
RECENT | MY DRIVE | SHARED WITH ME | COMPUTERS | NEARBY | UPLOAD

You've connected with the nearby device

🎮📱↔💻 Local multi-player
📁💻↔📱 Local in-app sharing

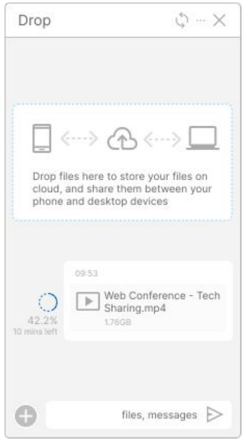


Device A: A web game running on the smart TV



Device B: A web game running on the smart phone, show game controller to play the game on nearby TV

▶️📱↔💻Cross-device workflows


Drop

Drop files here to store your files on cloud, and share them between your phone and desktop devices

09:53

Web Conference - Tech Sharing.mp4

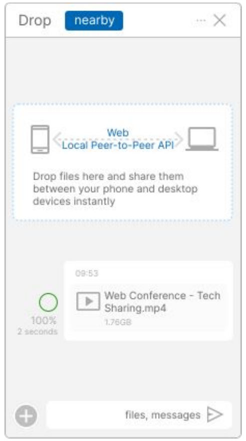
1.76GB

42.2%

10 mins left

files, messages

File sharing feature in web pages, same cloud account ID


Drop nearby

Web Local Peer-to-Peer API

Drop files here and share them between your phone and desktop devices instantly

09:53

Web Conference - Tech Sharing.mp4

1.76GB

100%

2 seconds

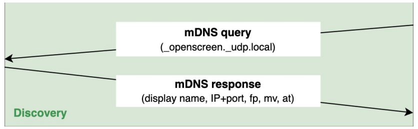
files, messages

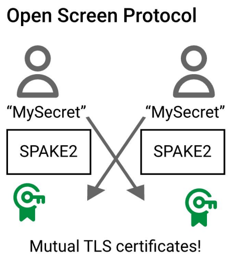
File sharing feature in offline web app without internet, between phone and desktop devices

📵Disaster relief
🏠Home services & IoT and more

Proposal

- **Discovery**
  - API inspired by `PresentationRequest` & `PresentationReceiver`
- **Authentication**
  - Open Screen Protocol to establish mutual TLS certificates between peers
- **Data exchange provided by two APIs:**
  - `LP2PDataChannel` API inspired by `RTCDataChannel`
    - simple message passing & WebRTC familiarity
  - `LP2PQuicTransport` API inspired by `WebTransport`





Comparison

	WebRTC	Local Peer-to-Peer
Discovery	SDP Signaling	DNS-SD / mDNS
NAT	ICE, STUN, TURN	Out-of-scope (local)
Authentication	SDP Signaling	SPAKE2
Encryption	DTLS	TLS
Transport	SRTP & SCTP	QUIC
Data APIs	RTCDataChannel	LP2PDataChannel LP2PWebTransport
Media APIs	RTCRtpSender & RTCRtpReceiver	Remote Playback API

$\left.\begin{array}{l}\text{Authentication}\\\text{Encryption}\\\text{Transport}\end{array}\right\}$ Provided by the OpenScreen protocol

→ Purposeful re-use of browser APIs for developer familiarity.

Feedback wanted

Considerations that in particular benefit from WebRTC WG's expertise:

1. The interplay between the DataChannel and WebTransport APIs
2. The potential opportunity to unify DataChannel & WebTransport across HTTP, LP2P, WebRTC transports
3. The interplay between QUIC and WebRTC

Spec draft: https://wicg.github.io/local-peer-to-peer/

Explainer: https://github.com/WICG/local-peer-to-peer/blob/main/EXPLAINER.md

Feedback: https://github.com/WICG/local-peer-to-peer

Beyond WebSockets (and Data Channel) APIs

- Problem: WebSockets API circumvents transport flow control
    - TCP recv window advertises remaining buffer space
    - When data is removed from the buffer, recv window expands
    - In a browser-based WebSockets API implementation, received data is moved to the event queue, is delivered to applications via the ***onmessage*** event handler. Problems:
        - recv window expands even if application does not process incoming messages.
        - Applications cannot exert back pressure without implementing app layer flow control
- Data Channel: copied the (broken) WebSockets API

WHATWG Streams and Back Pressure

- WHATWG Streams API supports backpressure
  - Application data consumption directly coupled to transport flow control
- APIs based on WHATWG Streams
  - WebTransport API (client/server)
    - Protocols: WebTransport over HTTP/3 (QUIC) or HTTP/2 (TCP)
  - P2P WebTransport API (P2P extension to WebTransport API)
  - WebSocketStream API (stream-based API for the WebSockets protocol)
    - Protocol: WebSockets over TCP (RFC 6455)
    - Explainer
    - Tracking bug

Top 5 Things to Know about P2P QUIC

1. It has already been done (WebTransport started life as p2p)
   a. P2P WebTransport API: https://w3c.github.io/p2p-webtransport/
   b. P2P Origin Trial: https://developer.chrome.com/blog/rtcquictransport-api
2. Is better than DTLS+SCTP (WebRTC data channels) in some ways:
   a. More things being built on QUIC (MoQ, RoQ, …)
   b. More implementations of QUIC (for mobile/native apps)
   c. P2P QUIC more likely to get real-time congestion control (port of googcc)
3. Is better than WebTransport in some ways:
   a. p2p :)
   b. supports self-signed certs
   c. Is "raw QUIC" underneath (more impls)
4. On the web, it doesn't exist yet (needs public demand)
5. For use in native apps, P2P QUIC has been implemented in:
   a. Pion over QUIC
   b. RTP over QUIC
   c. P2P QUIC (Rust) (no ICE; NAT traversal a "TODO")

What Can You Do With P2P QUIC?

P2P QUIC is potentially useful in scenarios that require peer-to-peer operation, such as:

- Remote desktop or mobile device
- Peer-to-peer caching
- Console to mobile device game streaming

While these scenarios can be implemented using the WebRTC data channel, P2P QUIC provides better performance:

- Faster setup time (0RTT)
- More efficient loss recovery
- Fewer false RTOs under widely varying RTT
- Improved congestion control (farewell NewReno!)

Bidirectional Communication on the Web

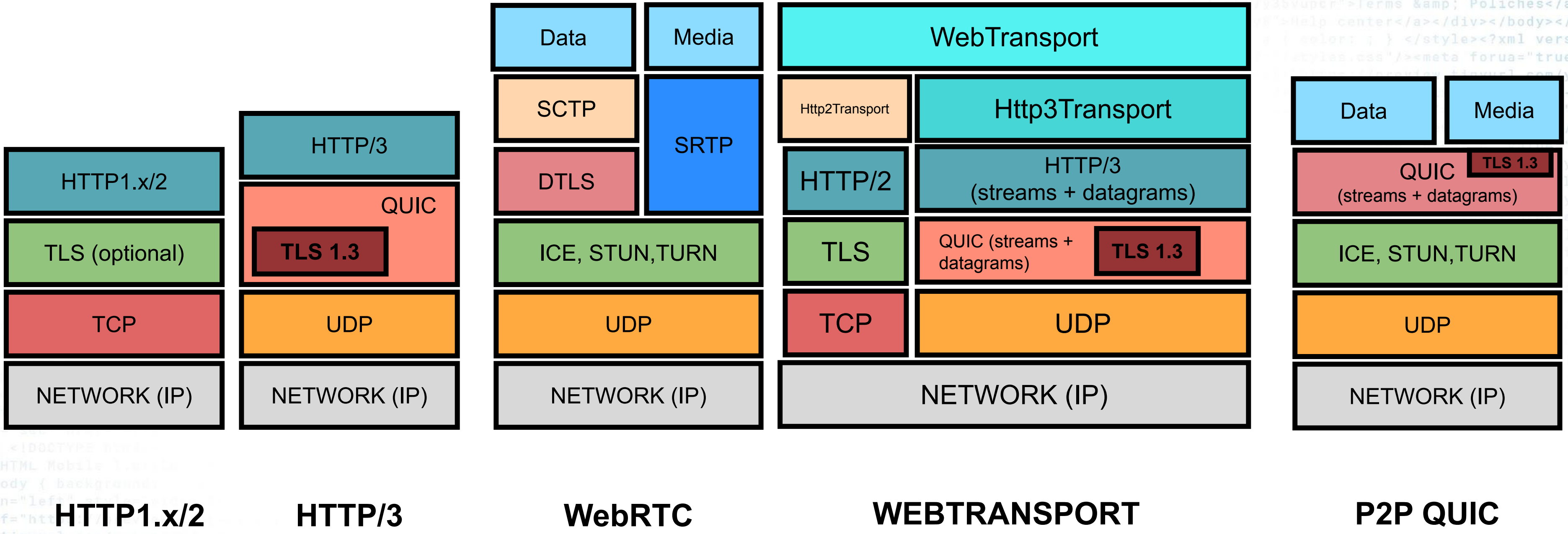
	Client-Server	**Peer-to-peer**
Reliable and ordered	WebSocket (also WebTransport)	RTCDataChannel or P2P QUIC
Reliable but unordered	WebTransport	
Unreliable and unordered		

P2P QUIC as a Media Transport

P2P QUIC isn't just about transport of data. Currently, it is under consideration as the next generation media transport within the IETF:

- RTP over QUIC (RoQ, AVTCORE WG): Transport of realtime media (RTP/RTCP)
  - Protocol specification
  - Github repo
- Media over QUIC (MoQ, MoQ WG): Transport of streaming media (e.g. CMAF)
  - MoQ Transport specification

The Stack


HTTP1.x/2
TLS (optional)
TCP
NETWORK (IP)
HTTP1.x/2
HTTP/3
QUIC
TLS 1.3
UDP
NETWORK (IP)
HTTP/3
Data
Media
SCTP
SRTP
DTLS
ICE, STUN,TURN
UDP
NETWORK (IP)
WebRTC
WebTransport
Http2Transport
Http3Transport
HTTP/2
HTTP/3
(streams + datagrams)
TLS
QUIC (streams + datagrams)
TLS 1.3
TCP
UDP
NETWORK (IP)
WEBTRANSPORT
Data
Media
QUIC
TLS 1.3
(streams + datagrams)
ICE, STUN,TURN
UDP
NETWORK (IP)
P2P QUIC

Multiplexing and P2P QUIC

- Data-only
    - P2P QUIC used for data exchange, with STUN/TURN for NAT traversal and RTP/RTCP used for media.
        - Example: P2P QUIC substituting for WebRTC data channel.
        - Requires multiplexing STUN/TURN/DTLS/RTP/RTCP/QUIC on the same socket (defined in RFC 7983bis, now in RFC Editor Queue)
- Data + Media
    - P2P QUIC used to carry media as well as data.
        - Example: RoQ or MoQ + data (+ signaling?)
        - Requires multiplexing of STUN/TURN/QUIC on the same socket (defined in RFC 7983bis)
        - Requires a mechanism for multiplexing of data and media ***within*** QUIC.

RFC 9443: Multiplexing QUIC/STUN/TURN/RTP/RTCP/DTLS

```
            +----------------+
            |        [0..3] -+--> forward to STUN
            |                |
            |       [4..15] -+--> DROP
            |                |
            |      [16..19] -+--> forward to ZRTP
            |                |
packet -->  |      [20..63] -+--> forward to DTLS
            |                |
            |      [64..79] -+--> forward to TURN Channel
            |                | (if from TURN server), else QUIC
            |     [80..127] -+--> forward to QUIC
            |                |
            |    [128..191] -+--> forward to RTP/RTCP
            |                |
            |    [192..255] -+--> forward to QUIC
            +----------------+
```

Figure 3: The receiver's packet demultiplexing algorithm.

Establishing a P2P QUIC Connection

1. STUN/TURN/ICE
2. QUIC connection establishment
   a. Exchange of self-signed certificates, ALPN ("q2q") and transport settings.
      i. Support for QUIC datagrams required (`max_datagram_frame_size parameter, value=0x20`)
      ii. Endpoints that wish to demultiplex QUIC MUST NOT send the `grease_quic_bit transport` parameter, described in [RFC9287].
      iii. Adjusting the frequency of QUIC ACKs:
         1. draft-ietf-quic-ack-frequency
      iv. Support for arrival timestamps, as proposed in:
         1. Draft-smith-quic-receive-ts
         2. draft-huitema-quic-ts
      v. Support for QUIC timestamps highly desirable.
      vi. ICE used for connection migration, so no need for QUIC connection migration.

A bit of history…

- Peer-to-peer QUIC began life as an extension to ORTC API (a decomposed, SDP-free version of WebRTC)
  - Has "standalone" RTCIceTransport for ICE
- RTCQuicTransport+RTCIceTransport were implemented in a Chrome Origin Trial in 2019
  - Video: P2P QUIC Origin Trial
  - Developer blog
- After the Origin Trial, P2P QUIC morphed into WebTransport, a client/server API now shipping in Chromium.

RTCQuicTransport Object Model

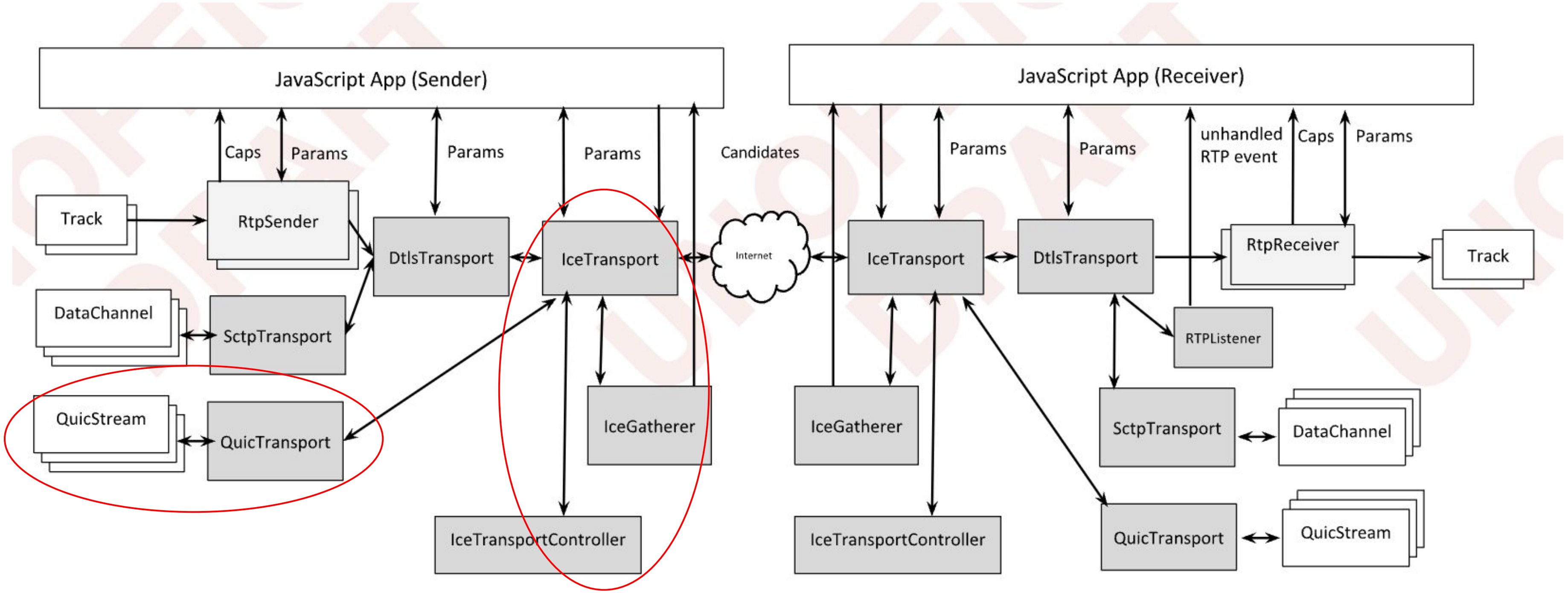
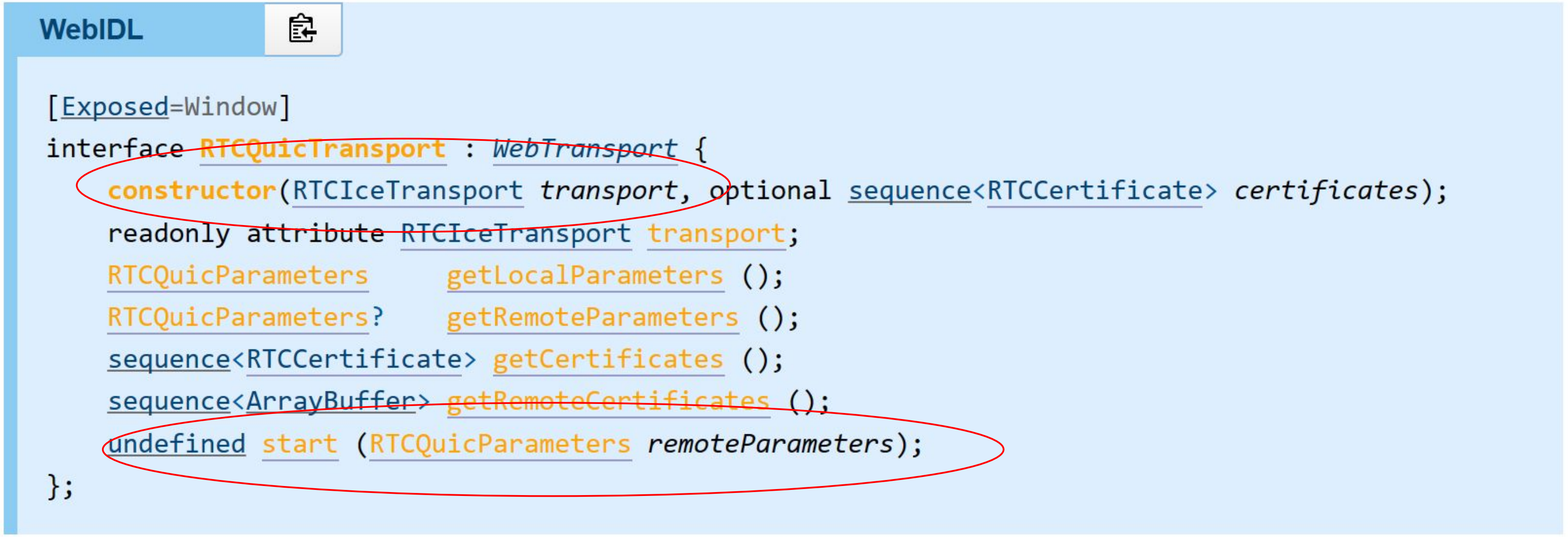


Figure 1 Non-normative ORTC Big Picture Diagram

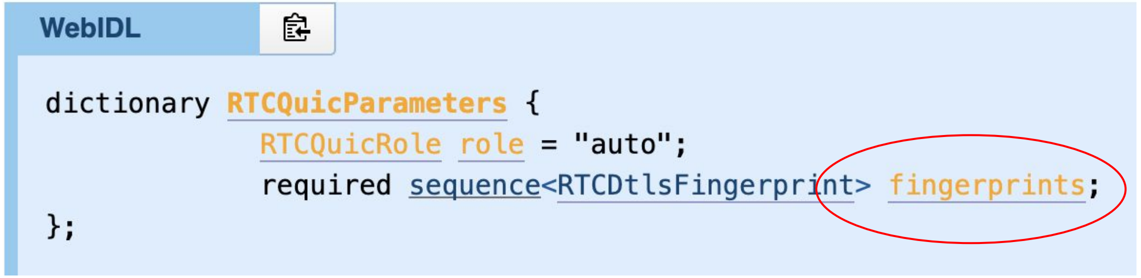
P2P QUIC Interface
https://w3c.github.io/p2p-webtransport/



WebIDL

```
[Exposed=Window]
interface RTCQuicTransport : WebTransport {
   constructor(RTCIceTransport transport, optional sequence<RTCCertificate> certificates);
   readonly attribute RTCIceTransport transport;
   RTCQuicParameters      getLocalParameters ();
   RTCQuicParameters?     getRemoteParameters ();
   sequence<RTCCertificate> getCertificates ();
   sequence<ArrayBuffer> getRemoteCertificates ();
   undefined start (RTCQuicParameters remoteParameters);
};
```

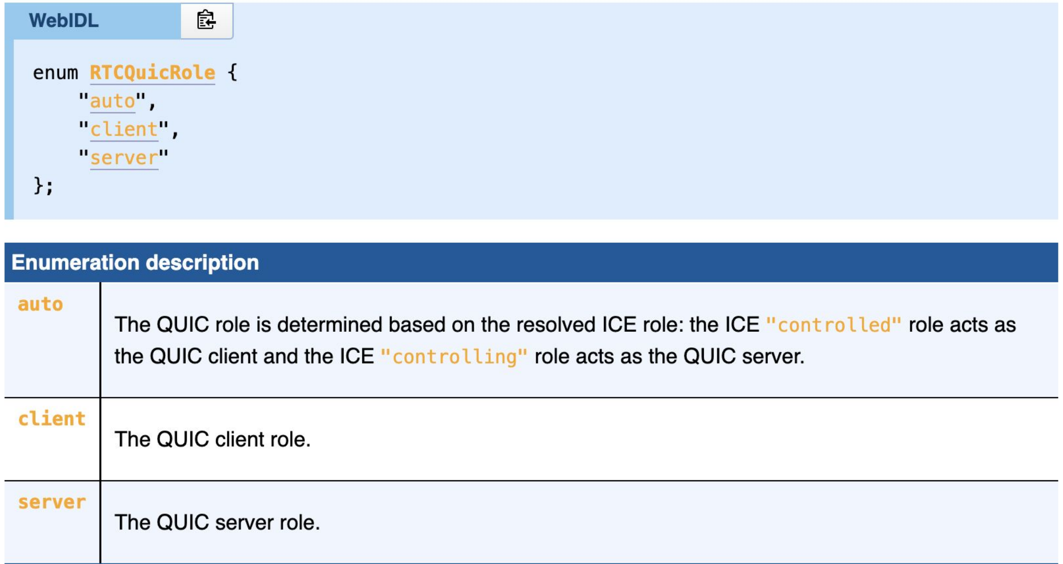
P2P QUIC API (cont'd)



WebIDL

```
dictionary RTCQuicParameters {
            RTCQuicRole role = "auto";
            required sequence<RTCDtlsFingerprint> fingerprints;
};
```

RTCQuicRole indicates the role of the QUIC transport.



WebIDL

```
enum RTCQuicRole {
    "auto",
    "client",
    "server"
};
```

Enumeration description	
auto	The QUIC role is determined based on the resolved ICE role: the ICE "controlled" role acts as the QUIC client and the ICE "controlling" role acts as the QUIC server.
client	The QUIC client role.
server	The QUIC server role.

Initiator Example

```
mySignaller.mySendInitiate({
   ice: iceGatherer.getLocalParameters(),
   quic: quicParameters,
 }, function(remote) {
   // Create the ICE and QUIC transports
   var iceTransport = new RTCIceTransport(iceGatherer);
   iceTransport.start(iceGatherer, remote.ice, RTCIceRole.controlling);
   iceTransports.push(iceTransport);
   // Construct a RTCQuicTransport object with the same certificate and fingerprint
   // as in the Offer so that the remote peer can verify it.
   var quicTransport = new RTCQuicTransport(iceTransport, certs);
   quicTransport.start(remote.quic);
   quicTransports.push(quicTransport);

   // ... Use WebCodecs to encode/decode media sent over the RTCQuicTransport
 });

  }
```

Responder Example

```
 / Prepare to handle remote candidates
  mySignaller.onRemoteCandidate = function(remote) {
    ice.addRemoteCandidate(remote.candidate);
  };

  mySignaller.mySendAccept({
    ice: iceGatherer.getLocalParameters(),
    quic: quic.getLocalParameters()
  });

  // Start the ICE transport with an implicit gather policy of "all"
  ice.start(iceGatherer, remote.ice, RTCIceRole.controlled);

  // Start the QUIC transport
  quic.start(remote.quic);

  // Use WebCodecs to encode/decode media sent over partially reliable QUIC streams or datagr
}
```

P2P QUIC Resources

- Videos
  - Introduction to WebTransport
  - P2P QUIC Origin Trial (2019)
- Protocol specifications
  - QUIC multiplexing with STUN/TURN/RTP/RTCP/DTLS
  - RTP over QUIC (RoQ)
- API specifications
  - ORTC API
  - P2P QUIC API
  - WebRTC-ICE API
- Blogs
  - P2P QUIC Origin trial announcement (2019)
- Implementations
  - Pion over QUIC
  - RTP over QUIC
  - P2P QUIC (RUST)

Discussion (End Time: 09:30)

•

RtpTransport (Peter Thatcher)

Start Time: 09:30 AM

End Time: 09:50 AM

What was RtpTransport, again?

Today, we'll focus on "Use Case 1": Custom Packetization (https://github.com/w3c/webrtc-rtptransport/blob/main/explainer-use-case-1.md)

- (Reminder: we were all on board with a "piecemeal" approach)

Relevant WebRTC-Extended Use Cases:

- Section 2.3: Video Conferencing with a Central Server
- Section 3.2.1: Game streaming
- Section 3.2.2: Low latency Broadcast with Fanout
- Section 3.5: Virtual Reality Gaming

What You Can Accomplish With RtpTransport (Use Case 1)

- Customize RTP header extensions
  - Send and receive customized RTP header extensions
  - Send and receive custom RTP header extensions
- Customize codecs
  - Encode and packetize with a custom codec (WASM)
  - Depacketize and decode with a custom codec (WASM)
  - Encode and packetize with WebCodecs
  - Depacketize and decode with WebCodecs
- Implement a custom jitter buffer
- Implement custom FEC
- Do custom bitrate allocation

Example: Send customized RTP header extension (audio level)

```
const [pc, rtpSender] = await customPeerConnectionWithRtpSender();
const levelGenerator = new CustomAudioLevelCalculator();
const rtpSendStream = await rtpSender.replaceSendStreams()[0];
rtpSendStream.onpacketizedrtp = () => {
  const rtpPacket = rtpSendStream.readPacketizedRtp();
  rtpPacket.audioLevel = levelGenerator.generate(rtpPacket);
  rtpSendStream.sendRtp(rtpPacket);
};
```

Example: Send custom RTP header extension

```
// TODO: Negotiate headerExtensionCalculator.uri in SDP
const [pc, rtpSender] = await customPeerConnectionWithRtpSender();
const headerExtensionGenerator = new CustomHeaderExtensionGenerator();
const rtpSendStream = await rtpSender.replaceSendStreams()[0];
rtpSendStream.onpacketizedrtp = () => {
  for (const rtpPacket of rtpSendStream.readPacketizedRtp()) {
    rtpPacket.setHeaderExtension({
      uri: headerExtensionGenerator.uri,
      value: headerExtensionGenerator.generate(rtpPacket),
    });
    rtpSendStream.sendRtp(rtpPacket)
  }
};
```

Example: Receive custom RTP header extension

```
// TODO: Negotiate headerExtensionProcessor.uri in SDP
const [pc, rtpReceiver] = await customPeerConnectionWithRtpReceiver();
const headerExtensionGenerator = new CustomHeaderExtensionGenerator();
const rtpReceiveStream = await videoRtpReceiver.replaceReceiveStreams()[0];
rtpReceiveStream.onreceivedrtp = () => {
  for (const rtpPacket of rtpReceiveStream.readReceivedRtp()) {
    for (const headerExtension of rtpPacket.headerExtensions) {
      if (headerExtension.uri == headerExtensionProcessor.uri) {
        headerExtensionProcessor.process(headerExtension.value);
      }
    }
  }
  rtpReceiveStream.receiveRtp(rtpPacket);
}
```

Example: Send and packetize with custom codec (WASM)

```
const [pc, rtpSender] = await customPeerConnectionWithRtpSender();
const source = new CustomSource();
const encoder = new CustomEncoder();
const packetizer = new CustomPacketizer();
const rtpSendStream = await rtpSender.replaceSendStreams()[0];
for await (const rawFame in source.frames()) {
  encoder.setTargetBitrate(rtpSendStream.allocatedBandwidth);
  const encodedFrame = encoder.encode(rawFrame);
  const rtpPackets = packetizer.packetize(encodedFrame);
  for (const rtpPacket of rtpPackets) {
    rtpSendStream.sendRtp(rtpPackets);
  }
}
```

Example: Receive with custom codec (WASM) and jitter buffer

```
const [pc, rtpReceiver] = await customPeerConnectionWithRtpReceiver();
const jitterBuffer = new CustomJitterBuffer();
const renderer = new CustomRenderer();
const rtpReceiveStream = await videoRtpReceiver.replaceReceiveStreams()[0];
rtpReceiveStream.onreceivedrtp = () => {
  const rtpPackets = rtpReceiveStream.readReceivedRtp();
  jitterBuffer.injectRtpPackets(rtpPackets);
}
for await (decodedFrame in jitterBuffer.decodedFrames()) {
  renderer.render(decodedFrame)
}
```

Example: Receive audio with custom codec (WASM)

```
const [pc, rtpReceiver] = await customPeerConnectionWithRtpReceiver();
const depacketizer = new CustomDepacketizer();
const decoder = new CustomDecoder();
const packetizer = new CustomL16Packetizer();
const rtpReceiveStream = await videoRtpReceiver.replaceReceiveStreams()[0];
rtpReceiveStream.onrtpreceived = () => {
  const rtpPackets = rtpReceiveStream.readReceivedRtp();
  const encodedFrames = depacketizer.depacketize(rtpPackets);
  const decodedFrames = decoder.decode(encodedFrames);
  for (rtpPackets of packetizer.toL16(decodedFrames)) {
    rtpReceiveStream.receiveRtp(rtpPackets);
  }
}
```

Example: Send and packetize with WebCodecs

```
const [pc, rtpSender] = await customPeerConnectionWithRtpSender();
const source = new CustomSource();
const packetizer = new CustomPacketizer();
const rtpSendStream = await rtpSender.replaceSendStreams()[0];
const encoder = new VideoEncoder({
  output: (chunk) => {
    let rtpPackets = packetizer.packetize(chunk);
    for packet in rtpPackets {
      rtpSendStream.sendRtp(rtpPackets);
    }
  },
  …
});
```

Example: Send and packetize with WebCodecs (cont’d)

```
for await (const rawFrame of source.frames()) {
  encoder.configure({
    ...
    tuning: {
      bitrate: rtpSendStream.allocatedBandwidth;
      ...
    }
  });
  encoder.encode(rawFrame);
}
```

Example: Receive video with WebCodecs and custom jitter buffer

```
const [pc, rtpReceiver] = await customPeerConnectionWithRtpReceiver();
const jitterBuffer = new CustomJitterBuffer();
const renderer = new CustomRenderer();
const rtpReceiveStream = await rtpReceiver.replaceReceiveStreams()[0];
const decoder = new VideoDecoder({
  output: (chunk) => {
    renderer.render(chunk);
  }, …
});

rtpReceiveStream.onrtpreceived = () => {
  const rtpPackets = rtpReceiveStream.readReceivedRtp();
  jitterBuffer.injectRtpPackets(rtpPackets);
}
for await (encodedFrame in jitterBuffer.encodedFrames()) {
  decoder.decode(endcodedFrame)
}
```

Example: Receive audio with WebCodecs

```
const rtpReceiveStream = …;
const depacketizer = new CustomDepacketizer();
const packetizer = new CustomL16Packetizer();
const decoder = new AudioDecoder({
  output: (chunk) => {
    const rtpPackets = packetizer.toL16(chunk);
    for packet in rtpPackets {
      rtpRecieveStream.receiveRtp(rtpPackets);
    }
  }, …
});

rtpReceiveStream.onrtpreceived = () => {
  const rtp = rtpReceiveStream.readReceivedRtp();
  const encodedFrames = depacketizer.depacketize(rtp);
  decoder.decode(encodedFrames);
}
```

Big Remaining Question: What about Workers?

- We want sending and receiving of packets to be able to happen on a worker
  - But is that "can be on a worker" or "must be on a worker"?
  - In other words, is it "Transferable" or "Dedicated Worker"

More Stuff

- More examples in Use Case 1: (https://github.com/w3c/webrtc-rtptransport/blob/main/explainer-use-case-1.md)
  - Custom FEC
  - Custom bitrate allocation
- More great stuff in Use Case 2:

  (https://github.com/w3c/webrtc-rtptransport/blob/main/explainer-use-case-2.md)

  - Custom congestion control
  - Custom bandwidth estimation
  - Custom probing and pacing

Feedback

(Reminder: we were all on board with this "piecemeal" approach)

- Are all of your (relevant) use cases covered?
  - If not, file an issue at https://github.com/w3c/webrtc-rtptransport/issues
- Would you like to comment on the question of Workers?
  - Go to https://github.com/w3c/webrtc-rtptransport/issues/33
- Would you like to comment on the API shape while maintaining perf
  - Go to https://github.com/w3c/webrtc-rtptransport/issues/20
- Have any other thoughts/ideas?
  - File an issue at https://github.com/w3c/webrtc-rtptransport/issues

Discussion (End Time: 09:50)

-

Wrapup and Next Steps

Start Time: 09:50 AM

End Time: 10:00 AM

Next Steps

- Content goes here

Thank you

<u>Special thanks to:</u>

WG Participants, Editors & Chairs