W3C WebRTC WG Meeting

March 21, 2023
8 AM - 10 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 March 2023 interim meeting of the W3C WebRTC WG, at which we will cover:
  - WebRTC-Stats, WebRTC-PC, Mediacapture-extensions, Encoded Transform, ICE Controller API
- Future meetings:
  - April 18
  - May 16
  - June 27
  - July 18

About this Virtual Meeting

- Meeting info:
  - https://www.w3.org/2011/04/webrtc/wiki/March_21_2023
- Link to latest drafts:
  - https://w3c.github.io/mediacapture-main/
  - https://w3c.github.io/mediacapture-extensions/
  - https://w3c.github.io/mediacapture-image/
  - https://w3c.github.io/mediacapture-output/
  - https://w3c.github.io/mediacapture-screen-share/
  - https://w3c.github.io/mediacapture-record/
  - https://w3c.github.io/webrtc-pc/
  - https://w3c.github.io/webrtc-extensions/
  - https://w3c.github.io/webrtc-stats/
  - https://w3c.github.io/mst-content-hint/
  - https://w3c.github.io/webrtc-priority/
  - https://w3c.github.io/webrtc-nv-use-cases/
  - https://github.com/w3c/webrtc-encoded-transform
  - https://github.com/w3c/mediacapture-transform
  - https://github.com/w3c/webrtc-svc
  - https://github.com/w3c/webrtc-ice
- Link to Slides has been published on WG wiki
- Scribe? IRC http://irc.w3.org/ Channel: #webrtc
- The meeting is (still) being recorded. The recording will be public.
- Volunteers for note taking?

W3C Code of Conduct

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

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

Virtual Interim Meeting Tips

This session is (still) being recorded

- **Click**  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 - 08:50 AM WebRTC-Extensions, WebRTC-Stats, & WebRTC-PC
- 08:50 - 09:10 AM Encoded Transform - SDP (Harald)
- 09:10 - 09:50 AM Ice Controller API (Sameer Vijakar & 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.

WebRTC-Extensions, WebRTC-Stats, WebRTC-PC

Start Time: 08:10 AM

End Time: 08:50 AM

For Discussion Today

- WebRTC-Extensions
  - PR 148: Delete Removed Features Section (Florent)
  - PR 147: Add RTCRtpEncodingParameters.codec to change the active codec (Henrik & Florent)
- WebRTC-Stats
  - Issue 742: Assorted comments on RTCAudioPlayoutStats (Paul or Jan-Ivar)
  - Issue 730: The HW exposure check does not solve Cloud Gaming use cases (Henrik, Sun continued from last month discussion)
- WebRTC-PC
  - Issue 2820/PR 2829: setParameters/insertDtmf/replaceTrack should reject on [[Stopping]] as well as [[Stopped]]? (Jan-Ivar)
  - Issue 2827/PR 2828: Hard to tell if there are state gaps in connectionState algorithm (Jan-Ivar)
  - Issue 2835: Section 4.4.2: createOffer() and setLocalDescription() resource handling (Bernard)

PR 148: Delete Removed Features Section (Florent)

- Currently, Section 13 contains features removed from WebRTC-PC:

- There has been no progress on implementation.
- Can we remove Section 13?
    - If there is implementation progress, we can always add it back.

PR 147: Add RTCRtpEncodingParameters.codec to change the active codec (Henrik & Florent)

As previously decided and recently discussed (January Interim),
we want to be able to change codec with RTCRtpSender.setParameters() in order to…

- Allow different codecs on different encodings.
- Make it possible to change codec without re-negotiating.
- Allow specifying both codec and scalabilityMode with a single API call.

This is an FYI - Florent has now submitted PR 147 as promised:

```
partial dictionary RTCRtpEncodingParameters {
    // Parent dictionary to both RTCRtpCodecParameters and
    // RTCRtpCodecCapabilities as of PR 2834.
    RTCRtpCodec codec;
}
```

PR 147: Add RTCRtpEncodingParameters.codec to change the active codec (Henrik & Florent)

Example:

```
// Prior to negotiation
pc.addTransceiver('video', sendEncodings: [
  {codec: findCodec(RTCRtpSender.getCapabilities('video').codecs, 'VP8')},
  {codec: findCodec(RTCRtpSender.getCapabilities('video').codecs, 'VP9')},
]);

// After negotiation
const parameters = sender.getParameters();
parameters.encodings[0].codec = findCodec(parameters.codecs, 'H264');
parameters.encodings[1].codec = findCodec(parameters.codecs, 'AV1');
await sender.setParameters(parameters);
```

`codec` is a preference.

- If not specified, `getParameters().encodings[i].codec` is missing (backwards compat).
- If specified, we use `codec` rather than the first codec listed in the SDP.

Issue 742: Assorted comments on RTCAudioPlayoutStats (Paul or Jan-Ivar) (1/2)

PR 682 recently added `"media-playout"` stats that are *“Only applicable if the playout path represents an audio device.”* — but this appears to be a layer violation, gathering metrics downstream of an RTCRtpReceiver.track.

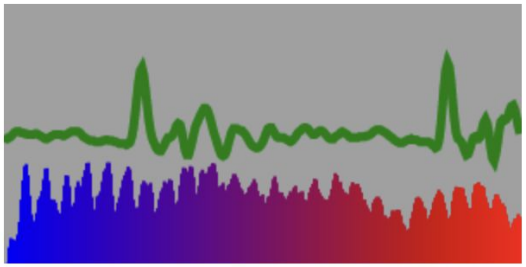
playoutId is frequently unimplementable, as there can be more than one playout path, or none, because a MediaStreamTrack can be sent to multiple AudioContexts and/or HTMLMediaElements for rendering.

An AudioContext can also be connected to another AudioContext, that are on different devices. The actual output can be on the first, second or both. This is another case where playoutId cannot be implemented.

All the other metrics are reimplementations of Web Audio API's Audio Render Capacity and latency metrics.

Issue 742: Assorted comments on RTCAudioPlayoutStats (Paul or Jan-Ivar) (2/2)

Things can look like playout paths that are not, e.g., an RTCRtpReceiver.track piped to an AudioContext with an AnalyzerNode or an AudioWorkletNode to compute some metrics or record the output (e.g. using a Web Worker and Web Codecs).



Only application authors know the shape of their audio output path, and only they can determine a quality metric, based on numbers provided by the object they actually use to render audio. Seems out of scope for WebRTC.

Proposal: Revert PR 682.

Issue 730: Recap: The HW exposure check does not solve Cloud Gaming use cases (Henrik, Sun)

`powerEfficient[Encoder/Decoder]` exposes HW capabilities and usage.

- To address privacy concerns, a HW exposure check was added:
  "*Only expose if context capturing state is true*" (= getUserMedia)

Problem:

- Does not work in the Cloud Gaming use case which does not capture.
- Specs are inconsistent!
  - Media Capabilities already exposes `powerEfficient`.
  - The MC privacy considerations section is vague.

Issue 730: The HW exposure check does not solve Cloud Gaming use cases (Henrik, Sun)

Proposal :

- There were two options. We decided to move forward with option 2 last month.
  - Option 2 (PR 732): Delegate the “am I allowed to expose HW information?” question to Media Capabilities (Discussed last month and continued on the issue)
  - We could not reach consensus because it could expose hardware capabilities.
- So we would like to go back to the option 1 (PR 725): defining a new metric for software decoder fallback events
  - Using the option #1, we could protect against fingerprinting by limiting the flag to when the decoder falls back when it started from a hardware decoder.

Issue 730: The HW exposure check does not solve Cloud Gaming use cases (Henrik, Sun)

Proposal - option 1(PR 725)

- `boolean decoderFallback` in `RTCInboundRtpStreamStats`. Advantages:
  - Decoder fallback can't be used to identify the User Agent since it only arises from system abnormalities and can be revoked when the system recovers. So it has a fingerprinting advantage compared power efficient.
    - In case of Software decoder(no HW Decoder): stays **false**
    - In case of HW Decoder: starts with **false** and become **true** on fallback to the software decoder. Changes to **false** again when the decoder recovers to HW.
    - In the Chromium browser implementation, fallback occurs for the following reasons:

```
enum class RTCVideoDecoderFallbackReason {
  kSpatialLayers = 0,
  kConsecutivePendingBufferOverflow = 1,
  kReinitializationFailed = 2,
  kPreviousErrorOnDecode = 3,
  kPreviousErrorOnRegisterCallback = 4,
  kConsecutivePendingBufferOverflowDuringInit = 5,
  kMaxValue = kConsecutivePendingBufferOverflowDuringInit,
};
```

Issue 2820/PR 2829: setParameters/insertDtmf/replaceTrack should reject on [[Stopping]] as well as [[Stopped]]?

Align spec with Chrome, Edge & WPT, which predate introduction of **[[Stopping]]**, preserving the following behavior:

```
tc1.stop();
await tc1.sender.setParameters(...); // InvalidStateError

tc2.stop();
await tc2.sender.track.replaceTrack(...); // InvalidStateError

tc3.stop();
await tc3.sender.dtmf.insertDtmf(...); // InvalidStateError
```

Matches precedent:

```
tc4.stop();
tc4.direction = "sendrecv"; // InvalidStateError
```

Issue 2827/PR 2828: Hard to tell if there are state gaps in connectionState algorithm

Editorial FYI:

Help show that

connectionState = iceConnectionState + DTLS state

RTCPeerConnectionState Enumeration description

Enum value	Description
`closed`	[[IceConnectionState]] is "closed".
`failed`	The previous state doesn't apply, and either [[IceConnectionState]] is "failed" or any RTCDtlsTransports are in the "failed" state.
`disconnected`	None of the previous states apply, and [[IceConnectionState]] is "disconnected".
`new`	None of the previous states apply, and either [[IceConnectionState]] is "new", and all RTCDtlsTransports are in the "new" or "closed" state, or there are no transports.
`connected`	None of the previous states apply, [[IceConnectionState]] is "connected", and all RTCDtlsTransports are in the "connected" or "closed" state.
`connecting`	None of the previous states apply.

NOTE

In the "connecting" state, one or more RTCIceTransports are in the "new" or "checking" state, or one or more RTCDtlsTransports are in the "new" or "connecting" state.

…and that there are no gaps (see issue for proof — thanks @pthatcher!)

Issue 2835: Section 4.4.2: createOffer() and setLocalDescription() resource handling (Bernard)

- Section 4.4.2 says:

  "If a system has limited resources (e.g. a finite number of decoders), createOffer needs to return an offer that reflects the current state of the system, so that setLocalDescription will succeed when it attempts to acquire those resources. The session descriptions MUST remain usable by setLocalDescription without causing an error until at least the end of the fulfillment callback of the returned promise."

- Do existing implementations provide this guarantee?
  - Youenn: “My understanding is that implementations guarantee that setlocalDescription will succeed because UA will not really acquire these resources at this time.”
- Do we need to change the text?

Discussion (End Time: 08:50)

-

Encoded Tranform (Harald)

Start Time: 08:50 AM

End Time: 09:10 AM

Negotiating Custom Codecs

Transforming content and being truthful about it

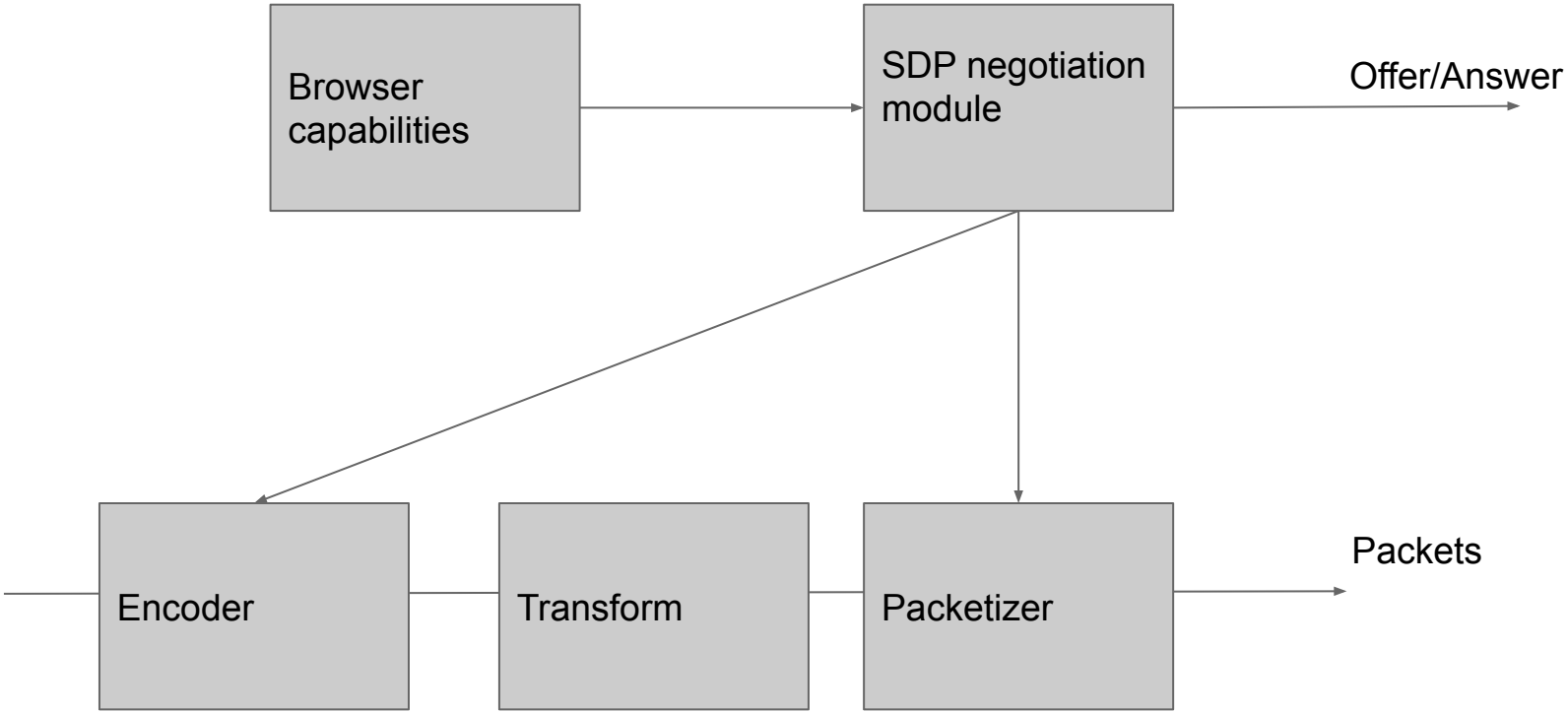
Issue #172

The Problem with Encoded Transform

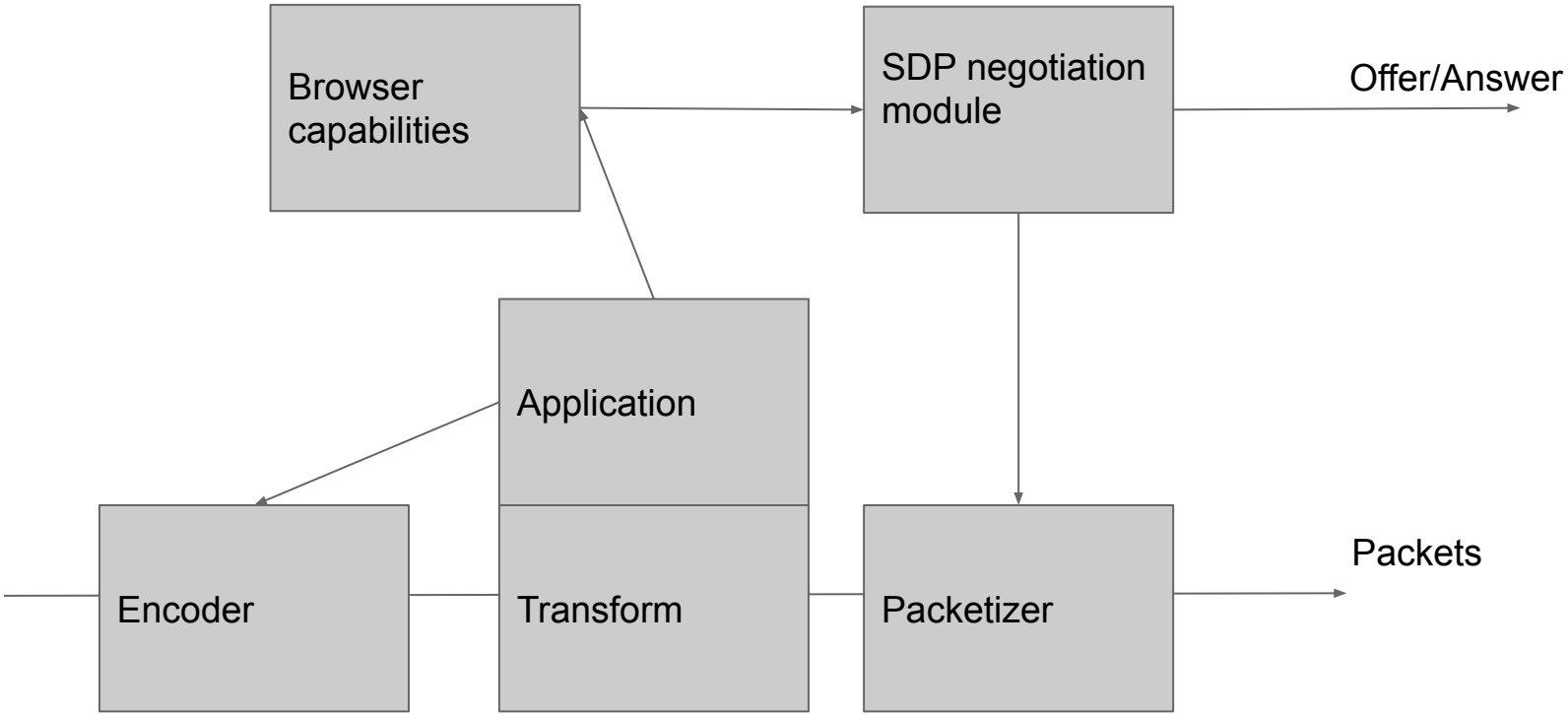
- An app sets up a connection, negotiating a set of codecs
- The app sender inserts a transform, changing the format on the wire
- The app receiver reverses the transform
- Problem: Elements on the way (SFUs, packetizers) expect the negotiated format, not the transformed format
- This complicates things. Complexity is bad.

Solution: Negotiate what you send.

The Encoded Transform model



The Enhanced Encoded Transform model



Operations in the Enhanced Encoded Transform

- The application **tells the SDP module about new formats**
- The application **tells the encoder what format to encode to**
- The other modules of the system **operate as before**

In particular:

- The SDP module negotiates over the set of known formats, with the normal controls over what format to select
- The encoder encodes to a supported format
- The packetizer is configured by the SDP module as before

(The receiving side functions similarly)

New APIs needed to achieve this functionality

New information needed about codecs - this allows SDP to configure the packetizer

```
partial dictionary RTCRtpCodecCapability {
  DOMString packetizationMode;
}
```

Pre-negotiation calls - these allow SDP to negotiate support of “custom” codecs

```
PeerConnection.AddSendCodecCapability(DOMString kind, CodecCapability capability)
PeerConnection.AddReceiverCodecCapability(DOMString kind, CodecCapability capability)
```

After creating senders and receivers - these allow the app to select the encoder and PT->decoder mapping

```
RTCRtpSender.SetEncodingCodec(RTCCodecParameters parameters)  // Alternatively, extensions PR #147
RTCRtpReceiver.AddDecodingCodec(CodecParameters parameters)
```

Example Code - Sender

```
customCodec = {
   mimeType: “video/acme-encrypted”,
   clockRate: 90000,
   sdpFmtpLine = “encapsulated-codec=vp8”,
   packetizationMode = “video/vp8”,
};
```

```
pc.addSenderCodecCapability(‘video’, customCodec);
sender = pc.AddTrack(videotrack);
// Negotiate as usual
for (codec in sender.getParameters().codecs) {
   if (codec.mimeType == “video/acme-encrypted”) {
      encryptedPT = codec.payloadType;
   }
}
if (!encryptedPT) { /* failure; don’t encrypt */ return; }
(readable, writable) = sender.getEncodedStreams();

readable.pipeThrough(new TransformStream(
   transform: (frame) => {
      metadata = frame.metadata();
      if (metadata.payloadType == expectedPT) {
        encryptBody(frame);
        metadata = frame.metadata();
        metadata.pt = encryptedPT;
        frame.setMetadata(metadata);
        writable.write(frame);
      }  // “Else” branch depends on application
   }
}).pipeTo(writable);
```

Example Code - Receiver

```
pc.AddReceiverCodecCapability(customCodec);
// Negotiation goes here
pc.ontrack = (receiver) => {

   for (codec in receiver.getParameters().codecs) {
      if (codec.mimeType == “video/acme-encrypted”) {
         encryptedPT = codec.payloadType;
      }
   }
   if (!encryptedPT) {  /* Failure, don’t decrypt */ return; }
   receiver.addDecodingCodec({mimeType: video/vp8, payloadType=208});
   (readable, writable) = receiver.getEncodedStreams();
   readable.pipeThrough(new TransformStream(
      transform: (frame) => {
         metadata = frame.metadata();
        if (metadata.payloadType == encryptedPT) {
           decryptBody(frame);
           metadata.payloadType = 208;
        }  // “Else” branch will depend on application
        writable.write(frame);
      }
   }).pipeTo(writable);
};
```

Next Steps

- This is not ready for adoption
- We need feedback, experimentation and thinking about whether this
  - implementable
  - useful
- We intend to host a spec branch on a public Git repository and take comments there
- In a month or two, it should be baked enough to propose to the WG for adoption in the form of merging a PR against the encoded-transform spec

Discussion (End Time: 09:10)

-

Ice Controller API
(Sameer Vijakar & Peter Thatcher)
Start Time: 09:10 AM
End Time: 09:50 AM

ICE Controller API

Allow applications to have greater visibility and control over the choice of connection used for transport.

Draft: https://sam-vi.github.io/webrtc-icecontroller

GitHub: https://github.com/sam-vi/webrtc-icecontroller

WebRTC NV Use Cases - Control

- N01 - The user agent can control candidate gathering and pruning, limiting the networks on which candidates are gathered, the types of candidates, etc.
- N04 - The ICE agent must be able to maintain multiple candidate pairs and move traffic between them.
- N05 - The ICE agent must be able to take the network cost into account when considering re-routing.
- N14 - The application must be able to minimize ICE connectivity checks.

WebRTC NV Use Cases - Reliability

- N15 - The application must be able to take steps to ensure a low and consistent latency for audio, video and data under varying network conditions.
- N30 - The user agent must provide the ability to re-establish media after an interruption.

WebRTC NV Use Cases - Forking

- N02 - The user agent must be capable of establishing multiple connections to peers without generating a separate configuration ("offer") for each connection prior to establishment.

API scope

Trigger candidate gathering with constraints	N01, N05
Fire events when candidates are added and removed	All
Status and RTT measurements of candidate pairs	N15
Send STUN pings, fire events when ping response is received or times out	N04, N14, N15, N30
Switch active candidate pair for sending and receiving	N04, N15, N30
Prevent or trigger pruning of candidate pairs	N04, N15, N30

API design principles

- Applications should have the necessary level of visibility and control over ICE
- User privacy and security must not suffer
- Applications should not need to implement a full ICE agent, just intervene when necessary

Incremental API - ICE Controller

- Browser performs ICE unless application says otherwise
- Established pattern for application to prevent default behaviour
  - Cancelable events
    - Form submission
    - UI events (pointer click, key press, wheel scroll)
  - Calling preventDefault() from a listener cancels the event
- Lower bar to use the API, only implement what you need
- Extensible and still allows full ICE takeover

ICE Controller - webrtc-icecontroller

WebIDL

```
[Exposed=Window]
interface RTCIceController : EventTarget {
  constructor();

  readonly attribute RTCIceRole role;
  RTCIceCandidatePair? getSelectedCandidatePair();

  attribute EventHandler oncandidatepairadded;
  attribute EventHandler oncandidatepairupdated;
  attribute EventHandler oncandidatepairswitched;
  attribute EventHandler oncandidatepairdestroyed;

  attribute EventHandler onicepingproposed;
  attribute EventHandler oniceswitchproposed;
  attribute EventHandler onicepruneproposed;

  Promise<undefined> sendIcePing(RTCIceCandidatePair candidatePair);
  Promise<undefined> switchToCandidatePair(RTCIceCandidatePair candidatePair);
  Promise<undefined> pruneCandidatePairs(sequence<RTCIceCandidatePair> candidatePairs);
};
```

WebIDL

```
partial dictionary RTCConfiguration {
  RTCIceController iceController;
};
```

WebIDL — cancelable

```
[Exposed=Window]
interface RTCIcePingProposalEvent : Event {
  constructor(DOMString type, RTCIcePingProposalEventInit eventInitDict);
  readonly attribute RTCIceCandidatePair candidatePair;
  readonly attribute DOMHighResTimeStamp? lastPingSentTimestamp;
  readonly attribute DOMHighResTimeStamp? earliestNextPingTimestamp;
};
```

WebIDL — conditionally cancelable

```
[Exposed=Window]
interface RTCIceSwitchProposalEvent : Event {
  constructor(DOMString type, RTCIceSwitchProposalEventInit eventInitDict);
  readonly attribute RTCIceCandidatePair candidatePair;
  readonly attribute RTCIceSwitchReason reason;
  readonly attribute RTCIceRecheck? recheck;
};
```

WebIDL — cancelable

```
[Exposed=Window]
interface RTCIcePruneProposalEvent : Event {
  constructor(DOMString type, RTCIcePruneProposalEventInit eventInitDict);
  sequence<RTCIceCandidatePair> getCandidatePairs();
};
```

A review on how ICE works….

And the difference between web and native

ICE steps

1. Gather local candidates
2. Signal candidates
3. Pair local and remote candidates
4. Send and receive checks and resposnes
5. Select a candidate pair
6. Send and receive “media”

Basic/Classic ICE

- Do these steps once
- One after another
- With one remote endpoint
- If the connection fails, completely start over

WebRTC ICE

- Do some steps more than once
  - Send checks forever for renewing consent
  - Add ICE servers
- Do some steps in parallel or ahead of time
  - Local candidate pooling
  - Trickle ICE

libwebrtc ICE

- Speed things up even more
  - Prioritize relayed candidates for checks, but not selection
  - Connect to many TURN servers, but then “prune” to one
- Do lots of steps more than once
  - Gather local candidates as network interfaces come up
  - Send checks very frequently to verify connectivity
  - Re-select and re-nominate candidate pairs regularly
- Be smarter about connection failures
  - Detect “failure” aggressively by watching incoming media
  - Regather candidates for “failed networks” regularly
  - Keep a backup candidate pair to fail over to

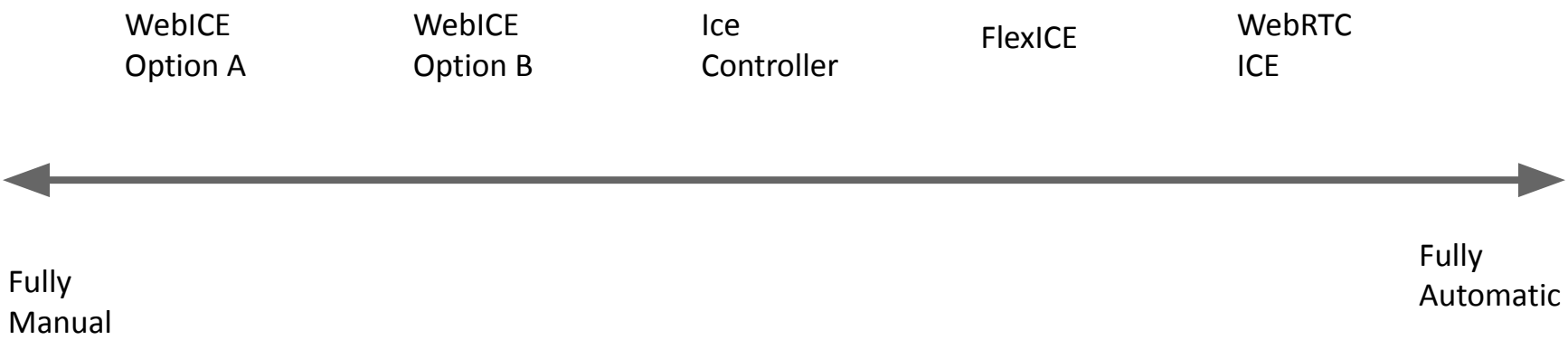
Even more advanced ICE

- Connect to many remote endpoints
  - ICE forking
- Use with QUIC
- Do optimizations that we haven’t thought of yet (or someone is doing in a native app)

Web apps vs. native apps

- All of this is available to native apps
- Some of this is available on some web browsers (but not very controllable)
- There is a gap
- Developers have been asking about it for years
- We already have WebRTC NV use cases for many
- Can we close the gap?

How can we close the gap?



WebICE Option A: Almost fully manual

- Some things automatic
  - Sending check responses
  - Gathering local candidates as network interfaces added
- Some things manual
  - When to start (re)gathering local candidates and
  - Adding and removing candidate pairs
  - Sending checks
  - Selecting a candidate pair
- Supports everything mentioned
  - ICE forking
  - Future optimizations (or novel native optimizations)

WebICE Steps

1. Gather local candidates with **`IceGatherer`**
2. Signal candidates with **`IceLocalCandidate`** and **`IceRemoteCandidate`**
3. Pair candidates with **`IceTransport.addCandidatePair`**
4. Send checks with **`IceCandidatePair.sendCheck`**
5. Select a candidate pair with **`IceTransport.selectedCandidatePair`**
6. Send and receive “media” with an **`IceTransport`**

WebICE Option A example (part 1)

```
let iceGatherer = new IceGatherer();
let iceTransport = new IceTransport();
let remoteCandidates = [];
iceGatherer.onlocalcandidategathered = (localCandidate) => {
  signaling.sendLocalIceCandidate(localCandidate);  // Step 2
  for (remoteCandidate in remoteCandidates) {
    iceTransport.addCandidatePair(localCandidate, remoteCandidate);  // Step 3
  }
};
signaling.onremoteicecandidatereceived = (remoteCandidate) => {  // Step 2
  remoteCandidates.push(remoteCandidate);
  for (localCandidate in iceGatherer.localCandidates) {
    iceTransport.addCandidatePair(localCandidate, remoteCandidate);  // Step 3
  }
};
iceGatherer.gather({iceServers: …}); // Step 1
```

WebICE Option A example (part 2)

```
let peerConnection = new PeerConnection({iceTransport: iceTransport});  // Step 6
sendChecks();  // Steps 4 and 5

function sendChecks() {
  let candidatePair = chooseNextCandidatePairToCheck(iceTransport.candidatePairs);
  if (candidatePair) {
    (async() => {
      let response = await candidatePair.sendCheck({priority: …, nominate: …}).getResponse(1.0);
      if (response && response.success &&
          betterThanSelectedCandidatePair(candidatePair, iceTransport.selectedCandidatePair) {
        iceTransport.selectedCandidatePair = candidatePair;
      }
    })();
  }
  setTimeout(sendChecks, 100);
}
```

WebICE Option B: optimally automatic

- A superset of Option A
- Can do everything Option A can
- Can optionally keep automatic behavior

WebICE Option B example

```
let iceGatherer = new IceGatherer();
let iceTransport = new IceTransport({
  automaticallyPairCandidates: iceGatherer,
  automaticallySendChecks: “controlling”,
  automaticallySelectCandidatePair: true,
});
iceGatherer.onlocalcandidategathered = (localCandidate) => {
  signaling.sendLocalIceCandidate(localCandidate);  // Step 2
};
signaling.onremoteicecandidatereceived = (remoteCandidate) => {  // Step 2
  iceTransport.addRemoteCandidate(remoteCandidate);  // Step 3, step 4, and step 5 are automatic
};
iceGatherer.gather({iceServers: …}); // Step 1
let peerConnection = new PeerConnection({iceTransport: iceTransport});  // Step 6
```

WebICE Example (N01; control gathering)

```
let iceGatherer = new IceGatherer();
iceGatherer.gather({
  // Control types of candidates
  excludeHost: true,
  excludeServerReflexive: true,
  excludeIpv6: true,
  // To exclude TURN, give no TURN servers
  iceServers: …,
  // Only useful for re-gathering after learning about network IDs.
  // No way to enumerate networks separately.
  excludeNetworkIds: …,
});
// “prune” a local candidate
iceGatherer.removeLocalCandidate(iceGatherer.localCandidates[0]);
// 5 minutes later
iceGatherer.gather(...);
```

WebICE Example (N02; ICE forking)

```
let iceGatherer = new IceGatherer();
let endpointById = {};
iceGatherer.onlocalcandidategathered = (localCandidate) => {
  signaling.sendLocalIceCandidate(localCandidate);  // Broadcast to many
};
signaling.onremoteendpoint = (endpointId) => {
  let iceTransport = new IceTransport({ automaticallyPairCandidates: iceGatherer, …});
  endpointById[endpointId] = {
    iceTransport: iceTransport,
    peerConnection: new PeerConnection({iceTransport: iceTransport}),
  };
};
signaling.onremoteicecandidate = (endpointId, remoteCandidate) => {
  iceTransportByEndpointId[endpointId].iceTransport.addRemoteCandidate(remoteCandidate);
};
iceGatherer.gather({iceServers: …});
```

WebICE Example (N04; select pair)

```
let iceTransport = …;
iceTransport.selectedCandidatePair = iceTransport.candidatePairs[0];
iceTransport.selectedCandidatePair = iceTransport.candidatePairs[1];

// Only for option B with automaticallyPairCandidates=true
// Ensures the candidate pair will not be removed and will and that checks
// will be sent with the given interval.
iceTransport.candidatePairs[0].sendCheckInterval = 25.0;
```

WebICE Example (N05; network cost)

```
let iceTransport = …;

if (iceTransport.candidatePairs[0].networkCost > iceTransport.candidatePairs[1].networkCost) {
  iceTransport.selectedCandidatePair = iceTransport.candidatePairs[1];
} else {
  iceTransport.selectedCandidatePair = iceTransport.candidatePairs[0];
}
```

WebICE Example (N14; infrequent checks)

```
let iceTransport = …;

for (let candidatePair of iceTransport.candidatePairs) {
  // Only for Option B.
  // Option A: just decide when to call candidatePair.sendCheck().
  candidatePair.sendCheckInterval = 25.0;
}
```

WebICE Example (N15; measure RTT)

```
// Option A
let iceCandidatePair = …;
let check = await iceCandidatePaircandidatePair.sendCheck();
let response = await check.getResponse(1.0);
let rtt = response.receivedTime - check.sentTime;

// Option B
iceCandidatePair.onchecksent = (check) => {
  let response = await check.getResponse(1.0);
  let rtt = response.receivedTime - check.sentTime;
}
```

WebICE WebIDL (IceGatherer)

```
interface IceGatherer {
 Promise<void> gather(IceGatherParameters);
 readonly sequence<IceLocalCandidate> localCandidates;
 attribute eventhandler onlocalcandidateadded;
 … more …
}

dictionary IceGatherParameters {
 sequence<RTCIceServer> servers;
 … more …
}
```

WebICE WebIDL (IceTransport)

```
interface IceTransport {
 readonly attribute sequence<IceCandidatePair> candidatePairs;
 attribute IceCandidatePair? selectedCandidatePair;

 Promise<IceCandidatePair?> addCandidatePair(IceLocalCandidate, IceRemoteCandidate);
 void removeCandidatePair(IceCandidatePair);

 // Option B only
 void addRemoteCandidate(IceRemoteCandidate);
 … more …
}
```

WebICE WebIDL (IceCandidatePair)

```
interface IceCandidatePair {
  readonly attribute LocalIceCandidate localCandidate;
  readonly attribute RemoteIceCandidate remoteCandidate;
  Promise<IceCheckSent> sendCheck(IceCheckParameters);
  // Option B only
  attribute double sendCheckInterval;
  attribute eventhandler onicechecksent;
  … more …
}

dictionary IceCheckParameters {
 unsigned long? priority;
 bool useCandidate = false;
 unsigned long? nomination;  // draft-thatcher-ice-renomination
 … more …
}
```

WebICE WebIDL (IceCheckSent)

```
interface IceCandidatePair {
  readonly attribute DOMHighResTimeStamp sentTime;
  readonly attribute Promise<IceCheckResponse?> getResponse(double timeout);
}

dictionary IceCheckParameters {
  readonly attribute success;
  readonly attribute unsigned short errorCode?;
  readonly attribute DOMHighResTimeStamp receivedTime;
  // … more …
}
```

WebICE WebIDL (IceLocalCandidate)

```
// Similar to RTCIceCandidate, but can have events and methods
interface IceLocalCandidate {
 readonly attribute DOMString address;
 readonly attribute unsigned short port;
 readonly attribute RTCIceProtocol protocol;
 readonly attribute RTCIceCandidateType type;
 readonly attribute RTCIceTcpCandidateType? tcpType;
 readonly attribute DOMString? relatedAddress;
 readonly attribute unsigned short? relatedPort;
 readonly attribute RTCIceServerTransportProtocol? relayProtocol;
 readonly attribute DOMString? Url;
 … more …
}
```

WebICE WebIDL (IceRemoteCandidate)

```
// More can be signaled, but this is all that’s needed by ICE
dictionary IceRemoteCandidate {
 DOMString usernameFragment;
 DOMString password;
 DOMString address;
 unsigned short port;
 RTCIceProtocol protocol;
 RTCIceTcpCandidateType? tcpType;
}
```

Discussion (End Time: 09:50)

-

Wrapup and Next Steps

Start Time: 09:50 AM

End Time: 10:00 AM

Name these Birds

Thank you

Special thanks to:

WG Participants, Editors & Chairs