W3C WebRTC/MediaCapture WG Meeting

January 11, 2018
8 AM PDT

Chairs: Stefan Hakansson
Bernard Aboba
Harald Alvestrand

W3C WG IPR Policy

- This group abides by the W3C patent policy https://www.w3.org/Consortium/Patent-Policy-20040205
- 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 interim meeting of the W3C WebRTC WG!
- During this meeting, we hope to:
  - Make progress on open issues in webrtc-pc, media capture and webrtc-stats
  - Introduce the webrtc-quic and webrtc-ice documents
  - Discuss updates to the WG Charter
- Editor’s Draft updates to follow meeting

Current Status of WebRTC-PC

- 96 open issues:
  - 23 relating to Identity
  - 19 editorial
  - 13 arising from test suite development
  - 11 PR exists
  - 4 questions
  - 4 pending IETF actions
  - 2 icebox

About this Virtual Meeting

Information on the meeting:

- Meeting info:
  - https://www.w3.org/2011/04/webrtc/wiki/January_11_2018
- Link to Slides has been published on WG wiki
- Scribe? IRC http://irc.w3.org/ Channel: #webrtc
- The meeting is being recorded.
- WebEx info here

February Virtual Interim: Focus on Screen Capture

Links to the latest drafts

- https://rawgit.com/w3c/mediacapture-main/master/getusermedia.html
- https://rawgit.com/w3c/webrtc-pc/master/webrtc.html
- https://w3c.github.io/mediacapture-screen-share/
- https://w3c.github.io/webrtc-stats/

New documents:

- https://w3c.github.io/webrtc-quic/
- https://w3c.github.io/webrtc-ice/
- https://w3c.github.io/webrtc-dscp-exp/

Agenda for Today

- 16:00 - 16:45 UTC: Issues
    - WebRTC-PC
    - Media-Capture
    - Statistics
- 16:45 - 17:15 UTC: Introduction to new documents
    - **webrtc-quic**
    - **webrtc-ice**
- 17:15 - 17:30 UTC: Charter Review

For Discussion Today

- WebRTC-PC Issues
  - Issue 1662: addTransceiver woes? (Stefan)
  - Issue 1689: Why is RTCRtpSynchronizationSource.voiceActivityFlag required-but-nullable? (Jan-Ivar)
  - Issue 1497: Not possible to tell how old `RTCRtpContributingSource.timestamp` is (Jan-Ivar)
  - Issue 1690: RTCRtpContributingSource.timestamp needs a clearer definition (Taylor)
  - Issue 1695: Effect of mute/disable on on-the-wire framerate is not described (AdamBe)

For Discussion Today (cont'd)

- **Media Capture Issues**
  - **Issue 472: How to implement web-compatible camera downscaling? (Jan-Ivar)**
- **Statistics Issues**
  - **Issue 235: Is keeping stats around a memory problem? (Jan-Ivar)**
- **webrtc-quic and webrtc-ice (Peter Thatcher)**
- **WebRTC WG re-charter (Bernard)**

WebRTC-PC Issues

- Issue 1662: addTransceiver woes? (Stefan)
- Issue 1689: Why is RTCRtpSynchronizationSource.voiceActivityFlag required-but-nullable? (Jan-Ivar)
- Issue 1497: Not possible to tell how old `RTCRtpContributingSource.timestamp` is (Jan-Ivar)
- Issue 1690: RTCRtpContributingSource.timestamp needs a clearer definition (Taylor)
- Issue 1695: Effect of mute/disable on on-the-wire framerate is not described (AdamBe)

Issue 1662: addTransceiver woes? (Stefan)

- #1662 raises two questions for `addTransceiver(kind)`:
  - 1. What is the direction (in SDP offer, and `transceiver.direction`)
  - 2. What is the `mid`?
- Mid question resolved ('pending' `mid` at creation, may be overridden by SDP answer)
- Direction: What is the default (Note: can be overridden by app)
  - Spec now says: default is always "sendrecv"
  - Has been argued that if transceiver is created with "kind" as first argument the default direction should be "recvonly"
  - My recommendation: keep "sendrecv", simple and no strong reason for changing
- Follow up question: Should replaceTrack work if no track was ever attached?
  - Conclusion: yes (in fact, the 'warm-up' example (Example 13) in the spec requires it)
- Further conclusions:
  - Need to clarify "send"
  - Need to clarify that the `direction` should not be considered when determining if negotiation is needed

Issue 1689: Why is RTCRtpSynchronizationSource.voiceActivityFlag required-but-nullable? (Jan-Ivar)

- WebIDL team advice: prefer optional over `null`. (both only for feature detection).
- Let's allow implementations to not implement voiceActivityFlag yet (like Firefox):
  - `dictionary RTCRtpSynchronizationSource : RTCRtpContributingSource {`

    ~~`required`~~ `boolean? voiceActivityFlag; // unimplemented, null=no-header-ext, false=v, true=v`

    `};`
- Where implementation is required, use optional over `null`:
  - `dictionary RTCRtpContributingSource { ...`

    ~~`required`~~ `byte`~~`?`~~ `audioLevel; // undefined=no-header-ext, 0-255=level`

    `};`
- Internally inconsistent, but consistent with best practice.

Issue 1497: Not possible to tell how old `RTCRtpContributingSource.timestamp` is (Jan-Ivar) Issue 1690: RTCRtpContributingSource.timestamp needs a clearer definition (Taylor)

- Define timestamp in reference to context's global monotonic clock.
- Prefer to have something that gives "something like the wall clock" for logs, and for somewhat better backwards compatibility.
- Comparable `syncOrContribSource.timestamp - stats.timestamp`
- Sounds like we want a timestamp that's comparable to either:
    a. `performance.timing.navigationStart + performance.now() // page-load reset`
    b. `performance.timeOrigin + performance.now() // monotonic to browser-start`NEW
- Ignore footgun of "don't compare it to Date.now()!" (fiddle)

Issue 1695: Effect of mute/disable on on-the-wire framerate is not described (AdamBe)

- From the original mediacapture-main issue (#441)
  - At what framerate should the “blackness” from a muted or disabled MediaStreamTrack be transmitted?
  - Firefox and Chrome seem to be doing different things here
  - The effect may be observable with MediaRecorder and stats
- Discussion at TPAC
  - According to minutes [1], we decided to specify 1 fps as an advice.
- Currently in the spec:
  - “If track is ended, or if track.muted is set to true, the RTCRtpSender sends silence (audio) or **a black frame** (video).”

[1] https://www.w3.org/2017/11/06-webrtc-minutes.html#item19

For Discussion Today (cont'd)

- Media Capture Issues
  - Issue 472: How to implement web-compatible camera downscaling? (Jan-Ivar)
- Statistics Issues
  - Issue 235: Is keeping stats around a memory problem? (Jan-Ivar)
- webrtc-quic and webrtc-ice (Peter Thatcher)
- WebRTC WG re-charter (Bernard)

Media Capture

- Issue 472/PR 502: How to implement web-compatible camera downscaling? (Jan-Ivar)

Issue 472/PR 502: web-compatible camera downscaling (jib)

Peter's `{resizeMode: “crop-and-scale”}` constraint from TPAC, modulo `"box-and-scale"` *

resizeMode	ConstrainDOMString	This string (or each string, when a list) should be a member of VideoResizeModeEnum. The members describe the means by which the resolution can be derived by the UA. In other words, whether the UA is allowed to use cropping and downscaling on the camera output. The UA may disguise concurrent use of the camera, by cropping and/or downscaling to mimic native resolutions when "none" is used, but only when the camera is in use in another browsing context.

```
enum VideoResizeModeEnum {
    "none",
    "crop-and-scale"
};
```

none	This resolution is offered by the camera, its driver, or the OS. Note: The UA may report this value to disguise concurrent use, but only when the camera is in use in another browsing context.
crop-and-scale	This resolution is downscaled and/or cropped from a higher camera resolution by the user agent.

“The UA SHOULD use the one with the smallest fitness distance, as calculated in step 3, *but MAY prefer ones with resizeMode set to "none" over "crop-and-scale"*.”

*) `"box-and-scale"` adds pixels and is incompatible with the constraints algorithm (these fitness distances would compete with "crop-and-scale" and "none" modes, with unpredictable and undesirable results).

Statistics

- Issue 235: Is keeping stats around a memory (really speed) problem? (Jan-Ivar)

Issue 235: Is keeping stats around a speed problem? (jib)

- getStats() gathers a non-trivial amount of live data, and is often called at a high frequency.
- Two uses of getStats: a) Live operational feedback, b) Accounting. Mirrors different impls:
  a. Firefox reports snapshots of live objects that exist at that time. Removed tracks disappear
  b. Chrome stores everything that ever happened.
- Uncapped accumulation of stats slows down getStats() over time + complicates result parsing:

```
[...(await sender.getStats()).values()].find(s => s.type == “track”) // latest track stats?
```

- Use-cases: “always-on” meeting rooms, security feeds, flipping between cameras (front/back):
  a. Repeated `removeTrack/addTrack` accumulates RTCRtpTransceivers + related stats.
  b. Repeated `sender.replaceTrack` accumulates “track” stats, even flipping same tracks.
  c. Frequent ICE restarts accumulate old ice candidates.
- Spec since 2012: *“The basic statistics model is that the browser maintains a set of statistics referenced by a selector. The selector may, for example, be a MediaStreamTrack. For a track to be a valid selector, it MUST be a MediaStreamTrack that **is sent or received** by the RTCPeerConnection object on which the stats request was issued.”*

[Issue 235](): Is keeping stats around a speed problem? (jib)

Proposal to fix this:

Omit `objectDeleted` stats from getStats, and add new methods to get them instead:

```
partial interface RTCRtpSender {
  RTCStatsReport getCompletedStats();
};
```

```
partial interface RTCRtpReceiver {
  RTCStatsReport getCompletedStats();
};
```

```
partial interface RTCPeerConnection {
  RTCStatsReport getCompletedStats(optional MediaStreamTrack? selector = null);
};
```

These return deleted stats (tracks + “stopped” senders/receivers) synchronously from a cache.

Benefits: Simpler parsing (no filtering on booleans). Isolates overhead (out of live-update hot path).

Trivial to combine results in JS:

```
[...(await sender.getStats()).values(), ...sender.getCompletedStats().values()] // All tracks
```

Issue 235: Is keeping stats around a speed problem? (jib)

If previous proposal is well received, a question: Do we need “track” stats in getStats()?

Users can compute live “track” stats from “sender” stats using JS:

```
let senderStat = [...(await sender.getStats()).values()].find(s => s.type == “sender”);
let oldTracks = [...sender.getCompletedStats().values()].filter(s => s.type == “track”);

let trackFramesSent = oldTracks.reduce((n, old) => n - old.framesSent, senderStat.framesSent);
```

WebRTC-QUIC Status

At TPAC, we decided to start an extension spec (to decide if we like it).

Now, here it is:

https://w3c.github.io/webrtc-quic/

https://github.com/w3c/webrtc-quic/issues

WebRTC-QUIC Approach

- `QuicTransport` constructed from `IceTransport` (and optional sequence of certificates)
- Outgoing `QuicStream` from `.createStream`()
- Incoming `QuicStream` from `.onstream`
- Read with `readInto(buffer)`
- Write with `write(data)`
- Controls for buffering/back-pressure, finish (clean end), reset (abrupt end)
- Access to state

WebRTC-QUIC Stream states (summarized)

As "sender"

createStream: -> opening

stream frame acked -> open

finish() or reset() -> closing

FIN/RST received -> closed

Write when opening/open

Read when open/closing

As "receiver"

.onstream: -> open

FIN/RST received -> closing

finish or reset() -> closed

Write when open/closing

Read when open

WebRTC-QUIC Issues Raised

- "QUIC" in the charter
- QUIC vs SCTP
- WHATWG streams
- "unadorned" QUIC
- ALPN
- Unidirectional/bidrectional streams
- 0-RTT

"QUIC" in the charter

Instead of mentioning "QUIC" in the charter, should we explain what we want to accomplish?

We should discuss in the charter discussion. But if it's not QUIC, what is it?

QUIC vs SCTP

A desire for "good reasons" for QUIC vs SCTP.

Some of my reasons:

- Fewer round trips to setup
- Ease of deployment/termination
- Robustness, maturity, variety of impls
- Direction of future innovation/protocols

WHATWG streams

Basically choose between these two things.

Proposal: Keep the left w/o dependencies

```
readInto(Uint8Array);
write(Uint8Array);
waitForReadable(amount);
waitForWritable(amount);
```

(And implement WHATWG streams on top, if you want them)

```
attribute ReadableStream readableStream;
attribute WritableStream writableStream;
```

(With a big dependency on WHATWG ReadableStream and WritableStream)

"unadorned" QUIC

To do multiple protocols on top of QUIC streams over the same 5-tuple at the same time, need either a new mechanism for segregating the QUIC streams, or multiple QUIC connections. This is an issue with QUIC in general.

Proposal: If a solution comes up for QUIC in general, use it. Maybe propose one in QUIC WG.

ALPN

You're supposed to put a value in the ALPN field in the client hello. It's supposed to identify the "application protocol". Who decides what that is?

Proposal: let the JS decide.

Unidirectional/bidrectional streams

The API is bidirectional because unidirectional streams didn't exist in QUIC until very recently. But it does make sense to support both.

Proposal: Update createStream/onstream to support both.

0-RTT

We don't have anything in the API for 0-RTT, which QUIC is capable of. Should we add support for it?

Proposal: Finish everything else we want and then come back to this later.

WebRTC-ICE Status

At TPAC, we decided to start an extension spec (to decide if we like it).

Now, here it is:

https://w3c.github.io/webrtc-ice/
https://github.com/w3c/webrtc-ice/issues

WebRTC-ICE Approach

- Extends existing IceTransport object with:
  - Constructor (no need for PeerConnection)
  - `gather(...)` to start gathering candidates
  - `start(...)` to start pairing/checking/selecting
  - `onlocalcandidate` to get local candidates
  - `addRemoteCandidate` to add remote candidates
- Does *not* have separate IceGatherer object
- Does *not* support parallel forking

Tricky Parts: What if you don't call gather()?

If you don't call gather(), you won't get local candidates.

So you have to call both gather() and start().

Is that too big of a foot gun?

The alternative is to make start() semi-magic. But then ICE restarts become less clear.

Proposal: Require calling both. You need to, anyway, for good use of trickle ICE on the caller side. So it encourages good use.

Tricky Parts: ICE Restarts

To initiate a full ICE restart, two things must happen:

1. New local parameters (ufrag/pwd) and candidates
2. New remote parameters (ufrag/pwd) and candidates

Proposal:
- New gathering done via call to `gather()`, which changes local parameters
- New remote parameters via `restart()`, which changes remote parameters

Like when you first start, you must call both. Again, on the restarter side, this is good trickle practice.

Tricky Parts: Changing gathering policy

If you want to add TURN servers without an ICE restart, like with PeerConnection.setConfiguration, what do you do?

Proposal: Have a variant of `gather()` which doesn't change the local ufrag/pwd, but just gathers new candidates.

Tricky Parts: Gathering done

Like with PeerConnection, it might be useful to signal "end of candidates" easily out of the box.

Proposal: Make it just like WebRTC: candidate.candidate == "" means "end of candidates". It's ugly but it works and we don't have to specify anything new.

Open Question: stats

Option A: iceTransport.getStats()

Option B: statCollector.getStats(iceTransport)

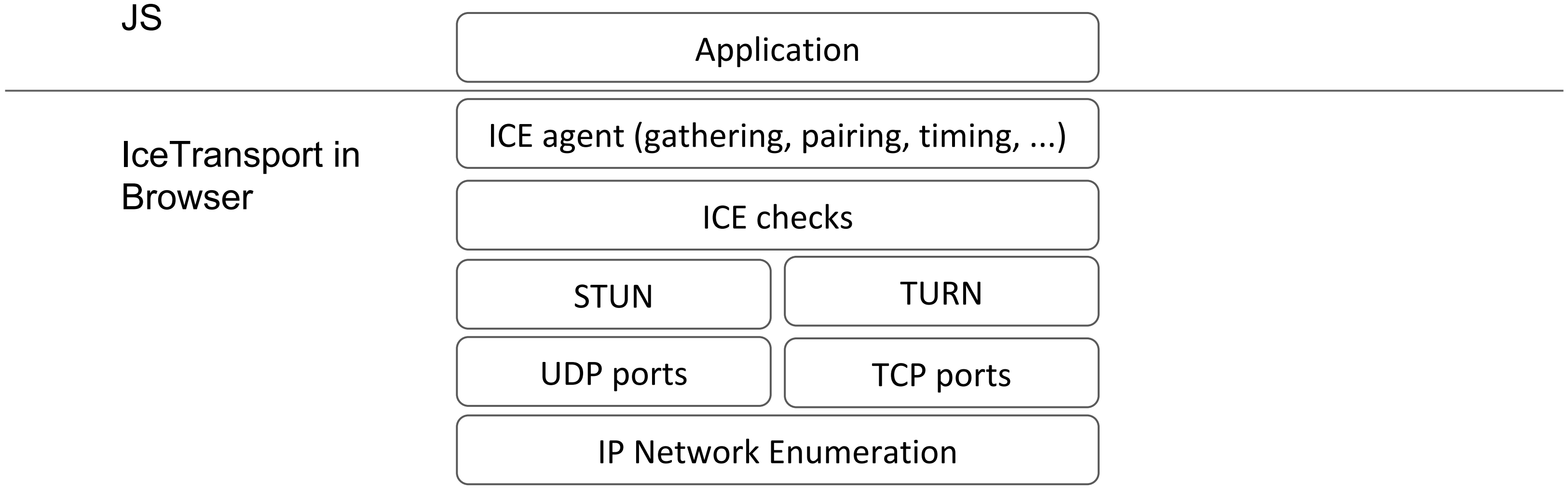
Which one?

An alternative to IceTransport: SliceTransport

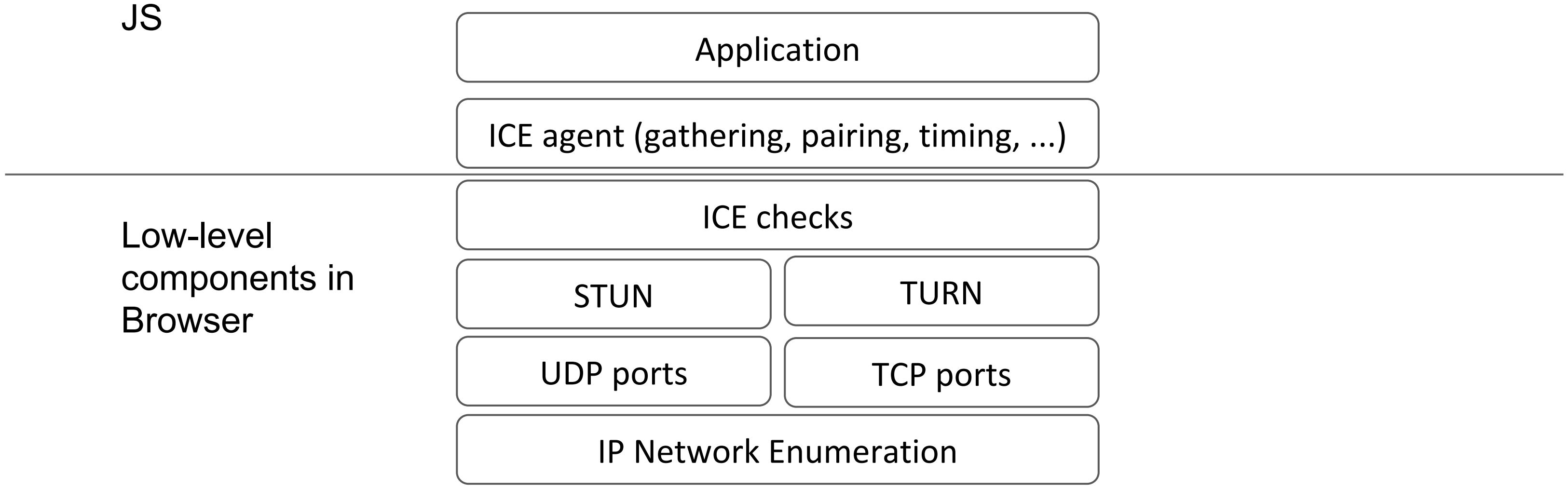
SLICE: Simple, Low-level ICE

Basically, Cullen's idea from TPAC of a low-level ICE where the JS gets to control (almost) everything.

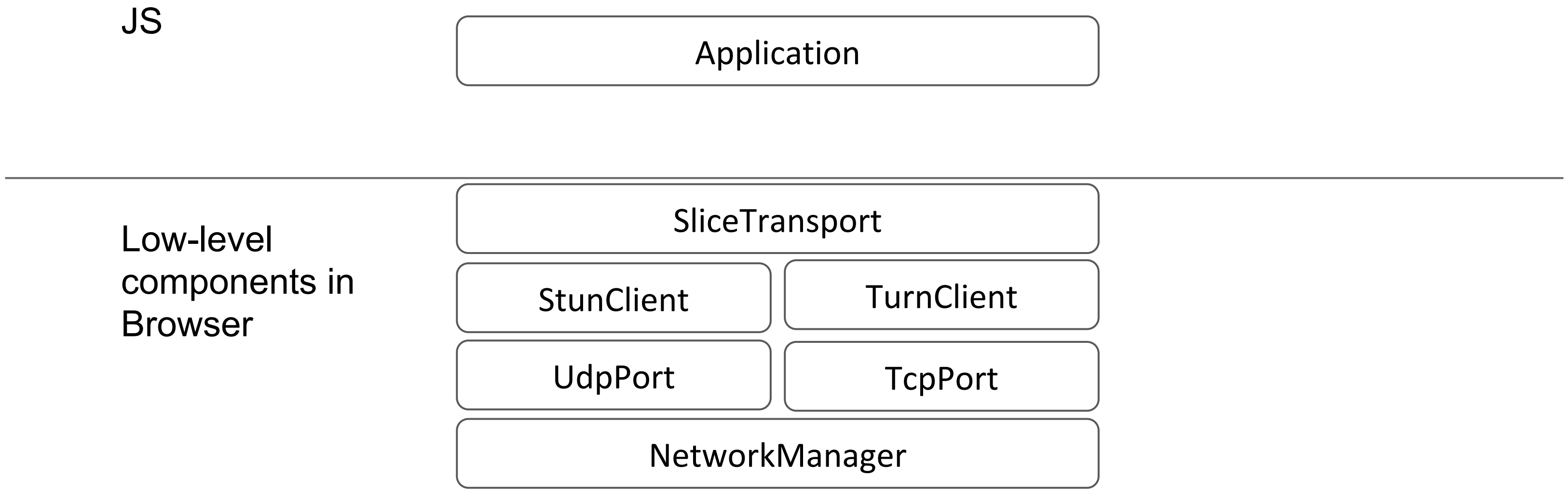
Today


JS
Application
IceTransport in Browser
ICE agent (gathering, pairing, timing, ...)
ICE checks
STUN
TURN
UDP ports
TCP ports
IP Network Enumeration

Cullen's idea


JS
Application
ICE agent (gathering, pairing, timing, ...)
Low-level components in Browser
ICE checks
STUN
TURN
UDP ports
TCP ports
IP Network Enumeration

New APIs


JS
Application
Low-level components in Browser
SliceTransport
StunClient
TurnClient
UdpPort
TcpPort
NetworkManager

How to use them

- Use `NetworkManager` to enumerate network interfaces and IPs
- Use `NetworkManager` to open `UdpPorts` and `TcpPorts`.
- Use `UdpPort/TcpPort` to connect to STUN/TURN servers
- Use `StunClient` to obtain server reflexive addresses
- Use `TurnClient` to allocation `TurnAllocations`
- Signal addresses to remote side and obtain remote addresses
- Pair addresses together
- Use `UdpPort/TcpPort/TurnAllocation` + remote address to get PacketTransports
- Add `PacketTransports` to `SliceTransport` as `IceNetworkRoutes`
- Send checks over `IceNetworkRoutes`
- Select an `IceNetworkRoute`

SliceTransport: checking and selection

```
interface IceTransport {
  // Can receive any network route and send on selected one
  IceNetworkRoute addNetworkRoute(PacketTransport transport);
  void selectNetworkRoute(IceNetworkRoute networkRoute);
  void removeNetworkRoute(IceNetworkRoute networkRoute);
}
interface IceNetworkRoute {
  // Sends an ICE check.  Resolves when the response is received
  Promise sendCheck(...);
  // Checks responses automatically sent.
  eventhandler oncheckreceived;
}
```

UDP PacketTransports

```
interface UdpPort {
  readonly attribute IpPort localAddress;
  UdpTransport connect(IpPort remoteAddress);
  void close();
  // Fired once for each new remote address
  eventhandler onnewremoteaddress;
}
// Only usable within SliceTransport, not by itself.
interface UdpTransport : PacketTransport {
  readonly attribute IpPort localAddress;
  readonly attribute IpPort remoteAddress;
  void close();
}
```

TCP PacketTransports

```
interface ClientTcpPort {
  readonly attribute IpPort localAddress;
  Promise<TcpTransport> connect(IpPort remoteAddress,
                                TlsMode tlsMode);
  void close();
}
// Only usable within SliceTransport, not by itself.
interface TcpTransport : PacketTransport {
  readonly attribute IpPort localAddress;
  readonly attribute IpPort remoteAddress;
  void close();
}
```

How to get IP, UDP, TCP (with permission)

```
interface NetworkManager {
  Promise<sequence<NetworkInfo>> EnumerateNetworks();
  Promise<UdpPort> OpenUdpPort(IPAddress localIp);
  Promise<ClientTcpPort> OpenClientTcpPort(
    IPAddress localIp, bool tls);
  attribute eventhandler onnetworkschanged;
}

interface NetworkInfo {
  readonly attribute sequence<IPAddress> ips;
  readonly attribute NetworkAdapterType type;
}
```

Getting STUN addresses

```
interface StunClient {
  // Resolves when response is received
  Promise<IpPort> sendBindingRequest(
    PacketTransport transport, ...);
}
```

Getting TURN addresses

```
interface TurnClient {
  Promise<TurnAllocation> sendAllocationRequest(PacketTransport transport, ...);
}
interface TurnAllocation {
  // TODO: Are refreshes automatic or controlled by JS?
  readonly attribute IpPort serverAddress;
  Promise sendCreatePermissionRequest(IpPort remoteAddress);
  TurnTransport connect(IpPort remoteAddress);
  void close();
  eventhandler onnewremoteaddress;  // IpPort
}
interface TurnTransport : PacketTransport {
  // TODO: Are channels automatic or controlled by JS?
  readonly attribute IpPort serverAddress;
  readonly attribute IpPort remoteAddress;
  void close();
}
```

Advantages

- Apps can optimize for their use case:
    - TURN-first
    - Control over wifi/cell usage
    - Add non-relay candidates without an ICE restart
    - Continual gathering or ICE half-restarts
    - Long-lived candidates
    - Backup candidates pairs
    - Variable check rate
- New capabilities automatically/naturally:
    - Connect to TLS host candidates
    - Parallel forking
    - TURN within TURN

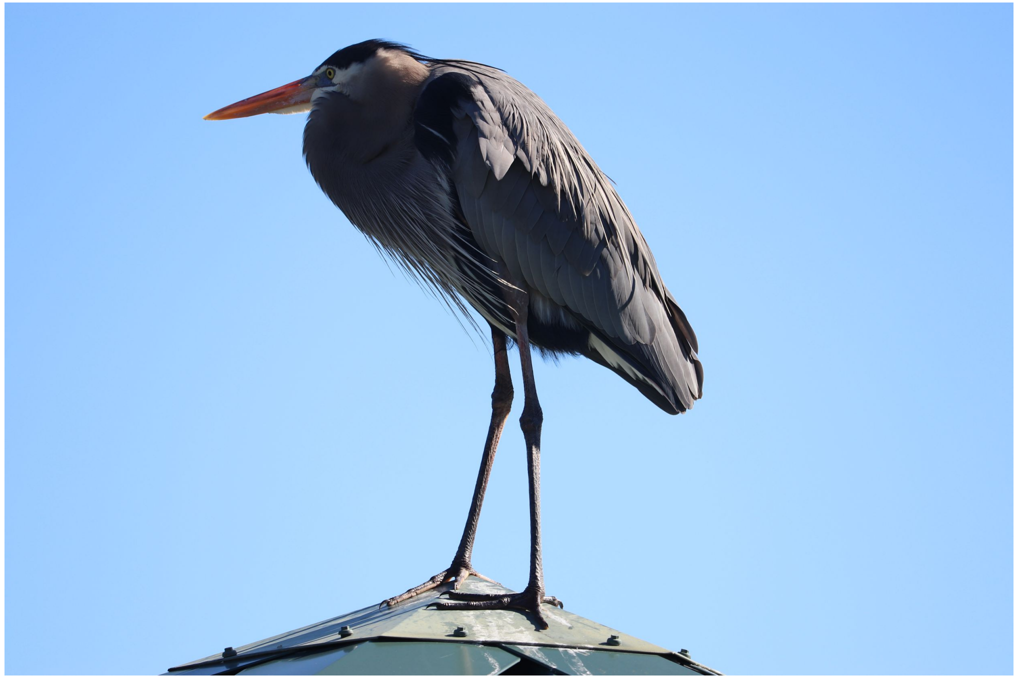
WebRTC WG Charter (Bernard)

- Charter proposal:
  - https://w3c.github.io/webrtc-charter/webrtc-charter.html
- Changes:
  - End date: 31 March 2020
  - Teleconferences: approximately 1 per month
  - Additional specifications:
    - QUIC in WebRTC
    - IceTransport extensions
    - DSCP Control API
  - Licensing: W3C Software and Document license

WebRTC WG Charter Discussion (cont’d)

- Issues raised on the list:
  - General stream API (for SCTP as well QUIC?)
  - Use cases
  - Protocol issues (for IETF)
    - Multiplexing (e.g. single QuicTransport limitation)
    - Pluggable congestion control for data channels
    - Definition of QUIC data channel protocol
    - ALPN usage with QUIC
  - Collaboration with IETF beyond RTCWEB WG:
    - QUIC WG (reliable and unreliable QUIC transport)
    - Congestion control WGs (RMCAT, TSVWG) and RGs (ICCRG)
- Please file Issues (and PRs)!

For extra credit



Name that bird!

Thank you

Special thanks to:

W3C/MIT for WebEx

WG Participants, Editors & Chairs

The bird