# W3C WebRTC WG Meeting

February 25, 2016 1pm PST

Chairs: Harald Alvestrand

Stefan Hakansson

Erik Lagerway

The meeting is being recorded.

# 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 some outstanding issues before transition to CR
- Editor’s Draft update to follow meeting

# About this Virtual Meeting

Information on the meeting:

- Hangouts Meeting
  - Participatory Hangout Link
- Link to Slides has been published on WG wiki
- Scribe? IRC https://irc.w3.org/ Channel: #webrtc

# For Discussion Today

- **Pull Requests**
  - **None yet**
- **Issues**
  - **296: [Bernard] Debugging ICE problems needs more info**
  - **457: [Bernard] Non-normative ICE state transition diagram**
  - **332: [Adam] Timing of ICE gathering**
  - **483 : [Taylor & Justin] Signaling a=end-of-candidates**
  - **442: [Taylor & Justin] Impossible to know if ICE agent is "finished checking", for "failed" and "completed" states.**
  -

# Issue 296: Debugging ICE problems needs more info (BA)

In WebRTC 1.0, we have:

1. interface **RTCIceCandidate** (with object properties)
2. State attributes for RTCIceTransport objects and state change events, as well as the selected pair:

   **RTCIceConnectionState** state; State of an individual ICE transport
   **RTCIceGatheringState** gatheringState; State of gathering of an individual ICE transport

   **RTCIceCandidatePair**? getSelectedCandidatePair (); Retrieval of the selected candidate pair

   attribute EventHandler onstatechange;
   attribute EventHandler ongatheringstatechange;
3. ICE agent state attributes and state change events:

   **RTCIceGatheringState** iceGatheringState; State of gathering within the ICE agent
   **RTCIceConnectionState** iceConnectionState; State of the ICE agent
   attribute EventHandler oniceconnectionstatechange;
   attribute EventHandler onicegatheringstatechange;
4. `icecandidateerror` event:

```
attribute EventHandler          onicecandidateerror;
        dictionary RTCPeerConnectionIceErrorEventInit : EventInit {
        DOMString      hostCandidate;
        DOMString      url;
        unsigned short errorCode; //Carries STUN error codes defined in: http://www.iana.org/assignments/stun-parameters/stun-parameters.
xml
        USVString      statusText;
};
```

# Issue 296: What we have (cont’d)

In WebRTC statistics, we have:

```
dictionary RTCIceCandidateAttributes : RTCStats {
        DOMString                 ipAddress;
        long                      portNumber;
        DOMString                 transport;
        RTCStatsIceCandidateType candidateType;
        long                      priority;
        DOMString                 addressSourceUrl;
};
```

Whereas in WebRTC 1.0 we have:

```
interface RTCIceCandidate {
readonly attribute DOMString candidate;
readonly attribute DOMString? sdpMid;
readonly attribute unsigned short? sdpMLineIndex;
readonly attribute DOMString foundation;
readonly attribute unsigned long priority;
readonly attribute DOMString ip;
readonly attribute RTCIceProtocol protocol;
readonly attribute unsigned short port;
readonly attribute RTCIceCandidateType type;
readonly attribute RTCIceTcpCandidateType? tcpType;
readonly attribute DOMString? relatedAddress;
readonly attribute unsigned short? relatedPort;
serializer = {candidate, sdpMid, sdpMLineIndex};
};
```

Note differences in attribute names and types.

Should we clean this up?

# Issue 296: What we have (cont'd)

In WebRTC statistics, we have:

```
dictionary RTCIceCandidatePairStats : RTCStats {
        DOMString                  transportId;
        DOMString                  localCandidateId;
        DOMString                  remoteCandidateId;
        RTCStatsIceCandidatePairState state;
        unsigned long long           priority;
        boolean                    nominated;
        boolean                    writable;
        boolean                    readable;
        unsigned long long           bytesSent;
        unsigned long long           bytesReceived;
        double                     roundTripTime;
        double                     availableOutgoingBitrate;
        double                     availableIncomingBitrate;
};
```

Additional stats collected in Edge:

- roundtrip maximum
- Number of consent requests sent
- Number of consent requests received
- Number of consent responses sent
- Number of consent responses received

```
partial dictionary RTCIceCandidatePairStats : RTCStats {
        double                  roundTripTimeMax;
        unsigned long long      consentRequestsSent;
        unsigned long long      consentRequestsReceived;
        unsigned long long      consentResponsesSent;
        unsigned long long      consentResponsesReceived;
};
```

RTCIceCandidatePairStats.state permits tracking of consent failures (e.g. "failed")
Is it also useful to collect statistics on consent requests/responses?
What about errors during connectivity checks?

# Issue 296: Error stats

`icecandidateerror` *`event.errorcode`* `includes the following errors:`

| Value | Name | Reference |
|---|---|---|
| 300 | Try Alternate | [RFC5389] |
| 400 | Bad Request | [RFC5389] |
| 401 | Unauthorized | [RFC5389] |
| 403 | Forbidden | [RFC5766] |
| 420 | Unknown Attribute | [RFC5389] |
| 437 | Allocation Mismatch | [RFC5766] |
| 438 | Stale Nonce | [RFC5389] |
| 440 | Address Family not Supported | [RFC6156] |
| 441 | Wrong Credentials | [RFC5766] |
| 442 | Unsupported Transport Protocol | [RFC5766] |
| 443 | Peer Address Family Mismatch | [RFC6156] |
| 446 | Connection Already Exists | [RFC6062] |
| 447 | Connection Timeout or Failure | [RFC6062] |
| 486 | Allocation Quota Reached | [RFC5766] |
| 487 | Role Conflict | [RFC5245] |
| 500 | Server Error | [RFC5389] |
| 508 | Insufficient Capacity | [RFC5766] |

1. These errors are only for gathering. Should we have connectivity check errors as well?
2. Is there value in having error counters in stats, or should we just let app developers handle it?

# Issue 296: Checklist State

1. Currently there is no info in either WebRTC 1.0 or statistics specs on the state of the check list.
2. In Trickle-ICE one cannot deduce the state of the check list from the state of each of the candidate pairs (since there could be candidates outstanding).
3. Should we introduce check list state?

# Issue 457: Non-normative ICE state transitions

**Introduction**

In Section 4.4.4, WebRTC 1.0 defines `RTCIceConnectionState` for the state of the ICE agent and includes a non-normative state transition diagram for the ICE agent.

`RTCIceConnectionState` is reused for RTCIceTransport.state (even though it refers to the ICE agent), and there is no equivalent state transition diagram for the individual ICE transports.

# Issue 332: Adam B

## Timing of ICE gathering

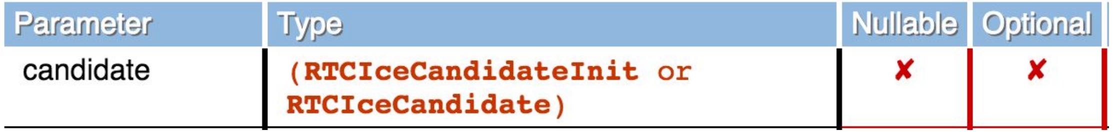
- Issues
  - When does gathering start? Conflicting text in the spec
  - “When ICE events occur" seems ill-defined
  - Assumption of two candidates for pre-gathering
- Proposed solution
  - PR #510 (merged) collects text about the ICE Agent in a section that directly follows its definition
  - PR #510 says that when the ICE Agent is initialized, it should start gathering if candidate pool size is non-zero.
  - Minor fixes in PR #515 (not merged)

# Issue 442: Taylor

# Impossible to know if ICE agent is “finished” checking

**Background**:

The “completed” and “failed” states only occur when the ICE agent is “finished checking”.



| Parameter | Type | Nullable | Optional |
|---|---|---|---|
| candidate | (RTCIceCandidateInit or RTCIceCandidate) | ✘ | ✘ |

However, *candidate* in addIceCandidate() is not nullable, so there is no way to “trickle” the fact that the remote peer is finished gathering candidates. A new remote candidate could therefore be added at any time, causing checking to resume.

# Issue 483: Is there inherent value to trickling end-of-candidates?

Trickle ICE says:

- “Sending the indication is necessary in order to avoid ambiguities and speed up ICE conclusion.”
- “Receiving an end-of-candidates notification allows an agent to update check list states and, in case valid pairs do not exist for every component in every media stream, determine that ICE processing has failed. It also allows agents to speed ICE conclusion in cases where a candidate pair has been validated but it involves the use of lower-preference transports such as TURN.”

JSEP says:

- If candidate gathering for the section has completed, an "a=end-of-candidates" attribute MUST be added, as described in [I-D.ietf-mmusic-trickle-ice], Section 9.3.

Are these reasons *alone* enough to lead us to trickling end-of-candidates?

# Question 2: Can we remove “completed”?

We requested feedback from application developers (10 responded), and no one used “completed” for anything but analytics.

# Question 3: Can we remove “failed”?

Out of the 10 application developers that provided feedback:

- Some use “failed” to show a message to the user. Others rely on “disconnected” or other criteria.
  - Messages are often different (e.g. “disconnected” is transient, while “failed” is not).
  - User may be able to do something to respond to “failed” indication (e.g. bring up new interface)
- Some don’t like the idea of “failed” (if it exists) being recoverable.
  - Currently, “Failed” is not recoverable in Trickle-ICE (or RFC 5245).
- Most are optimistic about “continuous gathering”, and agree that “failed” doesn’t make sense in that context.
- Everyone said they’d rather have continuous gathering without the “failed” state than to have no continual gathering.
- Everyone expressed willingness to change their application to handle new state definitions, if there’s a clear migration path.

# Option A: Trickle end-of-candidates (if an answer to any of the previous questions was “yes”)

State definitions would change as follows. Note that this almost matches the ORTC definitions.

**Old definitions (paraphrased):**

- checking: ICE agent is checking candidate pairs, and has never been connected.
- connected: ICE agent is connected, and checking other pairs.
- completed: ICE agent is connected, and not checking other pairs.
- disconnected: ICE agent is not currently connected, but was previously connected.
- failed: ICE agent is not checking, and has never been connected.

**New definitions:**

- checking: ICE agent is checking candidate pairs, and has never been connected.
- connected: ICE agent is connected, and either checking other pairs, or waiting for local/remote gathering to finish.
- completed: ICE agent is connected, not checking other pairs, and local/remote gathering is done.
- disconnected: ICE agent is not currently connected, and either was previously connected, or is not checking and is waiting for local/remote gathering to finish.
- failed: ICE agent is not checking, has never been connected, and local/remote gathering is done.

# Option A - Another way of looking at it

Old state matrix:

| | **Never connected** | **Connected** | **Liveness check failed** |
|---|---|---|---|
| **Checking** | Checking | Connected | Disconnected |
| **Not checking** | Failed | Completed | Disconnected |

New state matrix:

| | **Never connected** | **Connected** | **Liveness check failed** |
|---|---|---|---|
| **Checking** | Checking | Connected | Disconnected |
| **Not checking** | Disconnected | Connected | Disconnected |
| **Not checking + gathering done** | Failed | Completed | Disconnected |

# Option A - How would the API look?

One possibility:

```
pc.addIceCandidate(null);
```

This mirrors how onicecandidate signals a null candidate when gathering is done, and means we don’t need to add another API point.

However, we can discuss other options and work out the specifics out on the mailing list.

# Option B: Remove the “completed” and “failed” states.

State definitions would change as follows.

**Old definitions (paraphrased):**

- checking: ICE agent is checking candidate pairs, and has never been connected.
- connected: ICE agent is connected, and checking other pairs.
- completed: ICE agent is connected, and not checking other pairs.
- disconnected: ICE agent is not currently connected, but was previously connected.
- failed: ICE agent is not checking, and has never been connected.

**New definitions:**

- checking: ICE agent is checking candidate pairs, and has never been connected.
- connected: ICE agent is connected (may or may not be checking other pairs).
- ~~completed~~
- disconnected: ICE agent is not currently connected, and was either previously connected or is not checking.
- ~~failed~~

# Option B - Another way of looking at it

Old state matrix:

| | **Never connected** | **Connected** | **Liveness check failed** |
|---|---|---|---|
| **Checking** | Checking | Connected | Disconnected |
| **Not Checking** | Failed | Completed | Disconnected |

New state matrix:

| | **Never connected** | **Connected** | **Liveness check failed** |
|---|---|---|---|
| **Checking** | Checking | Connected | Disconnected |
| **Not Checking** | Disconnected | Connected | Disconnected |

# Option B - Migration path

<table>
<tr><th>Original code</th><th>New code</th></tr>
<tr><td>state == failed</td><td>!was_connected && state == disconnected</td></tr>
<tr><td>state == completed || state == connected</td><td>state == connected</td></tr>
<tr><td>state == disconnected</td><td>was_connected && (state == disconnected || state == checking)</td></tr>
<tr><td>state == checking</td><td>!was_connected && state == checking</td></tr>
</table>

# Thank you

Special thanks to:

Google - for the Hangout

WG Participants, Editors & Chairs