W3C ORTC Community Group Meeting

May 13, 2015 10:00AM-11:30AM PDT

W3C CG IPR Policy

- See the Community License Agreement for details.
- Goals are
  - Enable rapid spec development
  - Safe to implement via royalty-free commitments from participants+employers
  - Comfort for committers by limiting scope to OWN contributions
  - Transparency about who is making commitments
- How it works in practice
  - Anyone can post to public-ortc@w3.org
  - CG members who have signed CLA can post to public-ortc-contrib
  - Editor should ensure that spec includes only "contributions", CC-ing public-ortc-contrib makes that easier on the editor.

Welcome!

- Welcome to the 8th meeting of the W3C ORTC Community Group! Now 102 members!
- During this meeting, we hope to:
    - Bring you up to date on the status of the ORTC specification
    - Make progress on some outstanding issues
    - Organize/plan for implementation feedback

About this Virtual Meeting

Information on the meeting:

- ~~Jitsi~~ Hangouts Meeting
- Link to Slides has been published on CG home page & ORTC.org
- Scribe? IRC http://irc.w3.org/ Channel: ORTC
- The meeting is being recorded.

CG Chair

Erik Lagerway, erik@hookflash.com

W3C ORTC Community Group Basics

- W3C ORTC CG website:
  - http://www.w3.org/community/ortc/
- Public mailing list: public-ortc@w3.org
  - Join Here - link on the right hand side
  - Non-members can post to this list.
  - Non-member contributions are problematic.
- Contributor's mailing list: public-ortc-contrib@w3.org
  - Join Here - link on the right hand side
  - Members only, preferred list for contributions to the specification.

Associated Sites

- ORTC developer website: http://ortc.org/
  - Editor’s drafts, pointers to github repos, etc.
- ORTC API Issues List: https://github.com/openpeer/ortc/issues?state=open

Editor's Draft Changes

- 07 May 2015 Editor's draft:
  - http://ortc.org/wp-content/uploads/2015/05/ortc.html

Changes from the 25 March Editor's Draft:

Editorial

1. Transport terminology (SRST/MRST) and reference to RTP Grouping Taxonomy draft added to Section 1.1 (Issue 193)

WebRTC 1.0 compatibility

1. `sender.setTrack()` now a Promise (Issue 148)
2. Added DTMF parameters and capabilities (events from RFC 4733 Section 2.4) (Issue 177)
3. Exception when `sender.insertDTMF()` is called before `sender.send()` (Issue 178)
4. Exceptions for contradictory RTP/RTCP mux settings (***parameters.rtcp.mux*** false but ***rtcpTransport*** not set) (Issue 185)

Editor's Draft Changes (cont'd)

DTLS

1. DTLS packets arrive before `dtlsTransport.start()` (or even before **RTCDtlsTransport** is constructed!) (Issue 173)
2. Error handling for **RTCDtlsTransport, RTCIceTransport**, and **RTCIceGatherer** objects in the `"closed"` state (Issue 186)
3. ***RTCDtlsTransportState*** definitions (Issue 194)

173: DTLS packets arrive before `dtlsTransport.start()`

- Incoming DTLS packets can arrive once a successful response has been sent to an ICE connectivity check (Issue 170).
- **RTCIceTransport** buffers incoming DTLS packets prior to **RTCDtlsTransport** construction.
- Prior to calling `dtlsTransport.start()`, **RTCDtlsTransport** responds to incoming DTLS packets, but cannot verify remote fingerprint (required to enter the “`connected`” state).

186: Error Handling for Objects in the “closed” state

- Reviewed situations where an object in the “`closed`” state could be passed as an argument in a constructor or method.
- Examples where an `InvalidStateError` Exception is proposed to be thrown:
    a. Attempting to construct an **RTCDtlsTransport** from an **RTCIceTransport** in the “`closed`” state.
    b. Calling `dtlsTransport.start()` if ***dtlsTransport.state*** is “`closed`”.
    c. Calling `iceTransport.start()` if ***gatherer.state*** is “`closed`”.
    d. Calling `iceTransport.addRemoteCandidate(), .setRemoteCandidates(), .createAssociatedTransport()` if ***iceTransport.state*** is “`closed`”.
- Examples where an Exception is not proposed:
    a. Calling `iceTransport.stop()` if ***iceTransport.state*** is “`closed`”.
    b. Calling `iceTransport.getRemoteParameters(), .getRemoteCandidates(), .getNominatedCandidatePair()` if ***iceTransport.state*** is “`closed`”.

194: Clarification of ***RTCDtlsTransportState*** definitions

- No outgoing media before `“connected”` state (which requires remote fingerprint verification).
- Possible to receive incoming media in `“connecting”` state
  a. Remote peer has verified the local fingerprint (allowing it to send media), but local peer has not yet verified the remote fingerprint (because media arrives before the Answer).
- Calling `stop()`, `transport.stop()` or failing to verify the remote fingerprint causes transition to the `“closed”` state.

Editor’s Draft Changes (cont’d)

ICE

1. Added `close()` method to the ***RTCIceGatherer*** object (Issue 189)
    a. Calling `RTCIceGatherer.close()` prunes all local candidates.
    b. Associated ***RTCIceTransport*** objects transition to the "`disconnected`" state.
2. Behavior of TCP candidate types (Issue 190)
    a. Browsers MUST gather active TCP candidates and only active TCP candidates.
    b. Servers and other endpoints MAY gather active, passive or so candidates.
3. Behavior of `RTCIceGatherer.onlocalcandidate` (Issue 191)
    a. When the final candidate is gathered, an `onlocalcandidate` event occurs with ***RTCIceCandidateComplete*** emitted.
    b. When `RTCIceTransport.addRemoteCandidate()` is called with an ***RTCIceCandidateComplete*** dictionary as an argument, the ***RTCIceTransport*** can enter the “`completed`” state.

Questions for the CG

- Is the CG generally OK with the direction in which the Editor’s draft is headed?
- Do you have questions about general aspects of the spec?

For Discussion Today

- Additional ICE issues
  - ICE gatherer pruning (174, 197)
  - Incoming connectivity checks prior to calling `RTCIceTransport.start()` (170)
  - Priority calculation not possible without component (188)

Coming Attractions

- Behavior of `sender.send(parameters)` and `receiver.receive(parameters)` when ***parameters.encodings*** not set.
- Statistics API (updates to support simulcast and scalable video coding)
- Updates for compatibility with WebRTC 1.0 objects
- DTMF (if updated in WebRTC 1.0)
- IdP (if updated in WebRTC 1.0)
- Data Channel (if updated in WebRTC 1.0)

Ice Gatherer Pruning (Issues 174, 197)
(Problem Statement)

- An ***RTCIceGatherer*** is responsible for gathering potential host, server reflexive, and relay candidates.
  - However, it is not clear when those candidates can (or should) be pruned.
  - The number of outstanding "offers" awaiting an "answer" is uncertain.
- If candidates are prematurely pruned, incoming ICE connectivity checks might be ignored on pruned candidates.
- If candidates are never pruned, this can waste firewall ports and TURN resources, and potentially keep unused wireless WAN interfaces "warm" which consumes battery life on mobile.

Ice Gatherer Pruning (Issues 174, 197)
(Possible Solution "Incoming IceTransports keeps alive")

```
[Constructor(optional RTCIceGatherer gatherer)]
partial interface RTCIceTransport {
  // The RTCIceGatherer will not prune candidates if associated
  // RTCIceTransports are not in "failed" or "completed" state...
  readonly attribute RTCIceTransportState state;
};

// Example:
var iceGatherer = new RTCIceGatherer(..);  // will not prune until iceTransport is done
var iceTransport = new RTCIceTransport(iceGatherer,..);
var dtlsTransport = new RTCDtlsTransport(iceTransport);
```

Ice Gatherer Pruning (Issues 174, 197)
(Possible Solution "Incoming IceTransports keeps alive")

If an ***RTCIceTransport*** is constructed from an ***RTCIceGatherer*** and `RTCIceTransport.state` is not "`completed`" or "`failed`", then do not prune ***RTCIceGatherer*** host, reflexive, or relay candidates.

This requires that an ***RTCIceTransport*** accept an optional ***RTCIceGatherer*** in the constructor prior to calling `RTCIceTransport.start()` so that an extra transport can be used to keep candidates alive.

Benefits: Automatic pruning based on `iceTransport.state` with extra transport methodology to keep candidates alive for pending "answers"

Drawbacks: Optional ***RTCIceGatherer*** in the ***RTCIceTransport*** constructor.

Ice Gatherer Pruning (Issues 174, 197) (Possible Solution "setKeepAlive()")

```
partial interface RTCIceGatherer {
  // Based on the existing state, the RTCIceGatherer will keep candidates warm if
  // not in "failed" or "completed" state...
  void setKeepAlive(bool alive);
};

// Example:
var iceGatherer = new RTCIceGatherer(..);
var iceTransport = new RTCIceTransport(..);
iceGatherer.setKeepAlive(true);
//... do stuff but not pruning will happen in here...
iceGatherer.setKeepAlive(false); // okay to prune now
```

Ice Gatherer Pruning (Issues 174, 197)
(Possible Solution "setKeepAlive()")

Add an `RTCIceGatherer.setKeepAlive()` method with explicit control over when ***RTCIceGatherer*** host, reflexive, and relay candidates can be pruned or not.

Benefits: Very explicit pruning control;

Drawbacks: No automatic pruning; requires developer to prune candidates or candidates may never be pruned.

IceGatherer Pruning (Issues 174, 197) (Questions for ORTC CG)

1) Is either solution "acceptable"?
2) Are the solutions worth the effort?
3) Which solution is preferred?
4) Any other suggestions?

Responding to connectivity checks (Issue 170) (IMPORTANCE)

Responding to incoming ICE connectivity checks immediately is desirable to provide faster media setup:

1) Avoid buffering incoming connectivity checks arriving to the ***RTCIceGatherer*** (to deliver to an ***RTCIceTransport*** later once `RTCIceTransport.start()` is called after the "answer" arrives).
2) Avoid setup delays by responding to incoming connectivity checks, allowing DTLS to be negotiated and setup earlier than full round trip signalling normally requires.
3) Correctly calculate the round trip time for connectivity checks (for better metrics).
4) Avoid loss of media due to buffer overflows:
    a) Incoming media can arrive once DTLS has entered the “connecting” state (after a successful connectivity check response is sent).
    b) If `receiver.receive()` is called prior to receipt of the Answer, for video, we can avoid having an initial I-frame overflow the buffer, which could result in multiple packet losses/frame loss despite robustness measures (RTX, FEC).

Responding to connectivity checks (Issue 170)

- Remote peer can send ICE connectivity checks as soon it calls `RTCIceTransport.start()`.
    - This requires the local ***RTCIceGatherer*** ICE usernameFragment and password.
- `RTCIceGatherer.getLocalParameters()` returns the local usernameFragment and password.
    - While incoming ICE connectivity checks can be validated, responding is NOT possible unless the local ICE role is known, to be able to resolve a potential role conflict (i.e. controlling versus controlled).
- The ***RTCIceGatherer*** object does not have a role attribute, and ***RTCIceTransport.role*** is not set prior to calling `RTCIceTransport.start()`.

Responding to connectivity checks (Issue 170) (Possible Solution "RTCIceTransport constructor")

```
[Constructor(RTCIceGatherer)]
partial interface RTCIceTransport {
  readonly attribute DOMString remoteUsernameFragment; // can get auto-filled
  readonly attribute RTCIceRole role;                  // can get auto-filled

  attribute EventHandler? onremotetransportlatched;    // needed? use case?
};

// Example:
var iceGatherer = new RTCIceGatherer(..);
var iceTransport = new RTCIceTransport(iceGatherer,..);
var dtlsTransport = new RTCDtlsTransport(iceTransport,..);
```

Responding to connectivity checks (Issue 170) (Possible Solution "RTCIceTransport constructor")

1) Construct ***RTCIceTransport*** from an ***RTCIceGatherer*** without specifying the remote usernameFragment, password or role prior to receiving remote signalling;
    a) remote usernameFragment and local role are filled in based on the incoming connectivity check.
    b) In a forking scenario, can construct an array of ***RTCIceTransport*** objects.

2) Optional event to indicate the remote usernameFragment/role ***RTCIceTransport*** information is partially filled so the values can be queried;

Responding to connectivity checks (Issue 170) (Possible Solution "RTCIceTransport constructor")

1) Very small window where ***RTCIceTransport*** constructed from an ***RTCIceGatherer*** (and ready to respond to connectivity checks), but ***RTCDtlsTransport*** has not yet been constructed from the ***RTCIceTransport*** (so buffering is still needed):

```
var iceGatherer = new RTCIceGatherer(..);
var iceTransport = new RTCIceTransport(iceGatherer,..);
// tiny race window where ICE response can be sent but no DTLS transport wired yet
var dtlsTransport = new RTCDtlsTransport(iceGatherer,..);
```

2) Potential race condition between programmer calling `RTCIceTransport.start()` with a usernameFragment / role different that an auto-latched incoming remote ICE usernameFragment / role due to asynchronous API;

Responding to connectivity checks (Issue 170) (Possible Solution "Event on new incoming check")

```
partial interface RTCIceGatherer {
  attribute EventHandler? onnewincomingcheck;
};

partial interface RTCIceTransport {
  void respondToIncomingChecks(DOMString remoteUsernameFragment, RTCIceRole localRole);
};

// Example:
var iceGatherer = new RTCIceGatherer(..);
iceGatherer.onnewincomingcheck = function(remoteUsernameFragment, remoteRole) {
   var iceTransport = new RTCIceTransport(..);
   var localRole; (remoteRole == "controlling") ? localRole="controlled" : localRole="controlling";
   iceTranport.respondToIncomingChecks(remoteUsernameFragment, localRole);
};
```

Responding to connectivity checks (Issue 170) (Possible Solution "Event on new incoming check")

1) Add a new `RTCIceGatherer.onnewincomingcheck` - an event that fires when a new remote usernameFragment / role is seen for the first time;

2) Add a new method that can be called prior to `RTCIceTransport.start()`, i.e. `RTCIceTransport.respondToIncomingChecks(remoteUsernameFragment, localIceRole);`

Benefits: Possible to respond to incoming ICE checks prior to signalling (thus DTLS / media can be setup prior to full round trip signalling); No race condition between event / signaling;

Drawbacks: Still requires some small packet buffering;

Responding to connectivity checks (Issue 170) (Questions for ORTC CG)

1) Is either solution "acceptable"?
2) Are the solutions worth the effort?

   NOTE: Many ICE scenarios cannot be set up until full round trip signalling happens anyway due to firewall pinholes not being opened until outgoing checks are issued.

3) Which solution is preferred? Any other options?

Priority Calculation (Issue 188)

https://tools.ietf.org/html/rfc5245#section-4.1.2.1

```
  priority = (2^24)*(type preference) +
             (2^8)*(local preference) +
             (2^0)*(256 - component ID)
```

An ***RTCIceGatherer*** will gather ICE candidates immediately upon construction. However, it cannot calculate the ICE candidate priority to RFC 5245 recommended value before `iceTransport.start()` is called, since the component ID will not be known.

Proposed solution:

Similar to the ***RTCIceTransport***'s `createAssociatedTransport()`, add a `createAssociatedGatherer()` method so that `RTCIeGatherer.component` (RTP vs RTCP) is always known.

```
partial interface RTCIceGatherer {
    readonly    attribute RTCIceComponent component;
    RTCIceGatherer createAssociatedGatherer ();
};
```

Question for Community Group:

Are there any comments regarding known deficiencies of the ORTC API at this time?

Good time to speak before implementation is too far along!

Request from WebRTC WG

Stefan requested we comment on this...

The WebRTC and Device APIs Working Groups request feedback on the Last Call Working Draft of **Media Capture and Streams**, a JavaScript API that enables access to cameras and microphones from Web browsers as well as control of the use of the data generated (e.g. rendering what a camera captures in a html video element):

http://www.w3.org/TR/2015/WD-mediacapture-streams-20150414/

Organization / Call for implementation feedback

Mobile C++ ORTC implementation:
https://github.com/openpeer/ortc-lib-sdk

ORTC JS "shims" (i.e. downshim and upshim to / from WebRTC 1.0)
https://github.com/openpeer/ortc-js-shim (vacant repo)

ORTC specification:
https://github.com/openpeer/ortc

ORTC Node JS implementations:
https://github.com/openpeer/ortc-node

Browser Implementations:
Requested at this time (status.modern.ie lists ORTC as "**In Development**")

ORTC-lib Update

Work on ORTC-lib has resumed.

If you would like to participate in coding any of the ORTC open source projects then:

1. Join ORTC Community Group
2. Join developer mailing list / group
   http://ortc.org/dev
3. Start helping!

Thank you

Special thanks to:

Bernard Aboba - Microsoft

Michael Champion - Microsoft

Justin Uberti - Google

Peter Thatcher - Google

Robin Raymond - Hookflash

Erik Lagerway - Hookflash

Also, thanks to **Emil Ivov** for use of Jitsi

For More Information

ORTC Community Group

http://www.w3.org/community/ortc/

ORTC Developers & API Drafts

http://ortc.org