

(August 4th, 2015)



# Real-Time Communications with Microsoft Edge

Bernard Aboba  
Shijun Sun  
Microsoft









# Windows 10




# Edge Realtime Platform (Current Status)

<https://developer.microsoft.com/en-us/microsoft-edge/platform/status/>

 Media Capture and Streams	SUPPORTED	Build Number 10240+ 
 WebRTC – Object RTC API	SUPPORTED	Build Number 10547+ 
 WebRTC – WebRTC v1.0 API	SUPPORTED	Build Number 15019+ 
 H.264/AVC for RTC	SUPPORTED	Build Number 15019+ 
 VP8 for RTC	SUPPORTED	Build Number 15019+ 
 TLS 1.2	SUPPORTED	Build Number 10240+ 
 WebVR	SUPPORTED	Build Number 15002+ 
 Web Audio API	SUPPORTED	Build Number 10240+ 
 Web Speech API (synthesis)	SUPPORTED	Build Number 14316+ 

# Edge Realtime Platform (Roadmap)

<https://developer.microsoft.com/en-us/microsoft-edge/platform/status/>

 Capture from Canvas	UNDER CONSIDERATION		▼
 MediaRecorder	UNDER CONSIDERATION	★ 616 Votes	▼
 QUIC Support	IN DEVELOPMENT		▼
 Push API	IN DEVELOPMENT	★ 1756 Votes	▼
 Screen Capture	IN DEVELOPMENT		▼
 TLS 1.3	IN DEVELOPMENT	★ 129 Votes	▼
 Web Speech API (input)	IN DEVELOPMENT	★ 883 Votes	▼
 XBOX Support	IN DEVELOPMENT		▼

# Questions?

(August 4th, 2015)



# Developing RTC Applications with Microsoft Edge

Bernard Aboba  
Shijun Sun  
Microsoft

# What Can You Do With Edge?

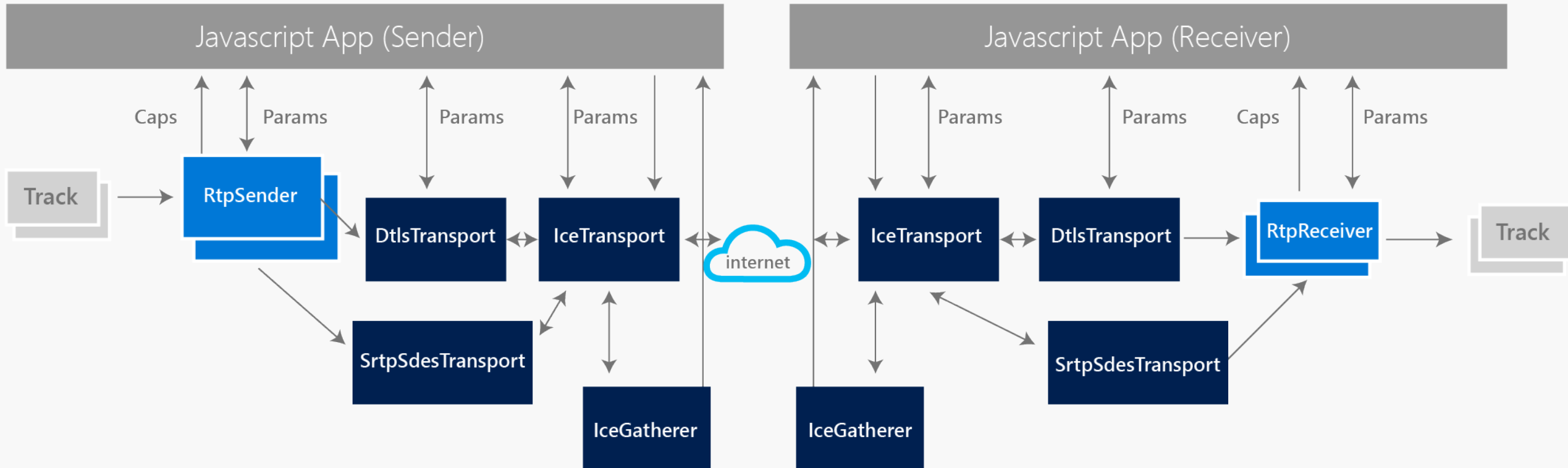
Build ORTC applications

Run WebRTC 1.0 Applications

- Using legacy native WebRTC 1.0 API

- Using the current WebRTC 1.0 API over ORTC (adapter.js)

# Microsoft Edge Object Model



Source: <https://rawgit.com/aboba/edgertc/master/msortc-rs3.html>



# Edge Limitations

- All APIs
  - No data channel support
  - No screen-sharing support (in development)
  - No support for stun URI, only turn
  - No support for certificate management (RSA certificates generated by default)
  - REMB implementation does not support RTX probes (under investigation)
  - Support for half-Trickle ICE only
  - RTP/RTCP routing issues
    - No support for “PT latching”
    - RTCP packet routing assumes all compound RTCP packets destined for same m-line

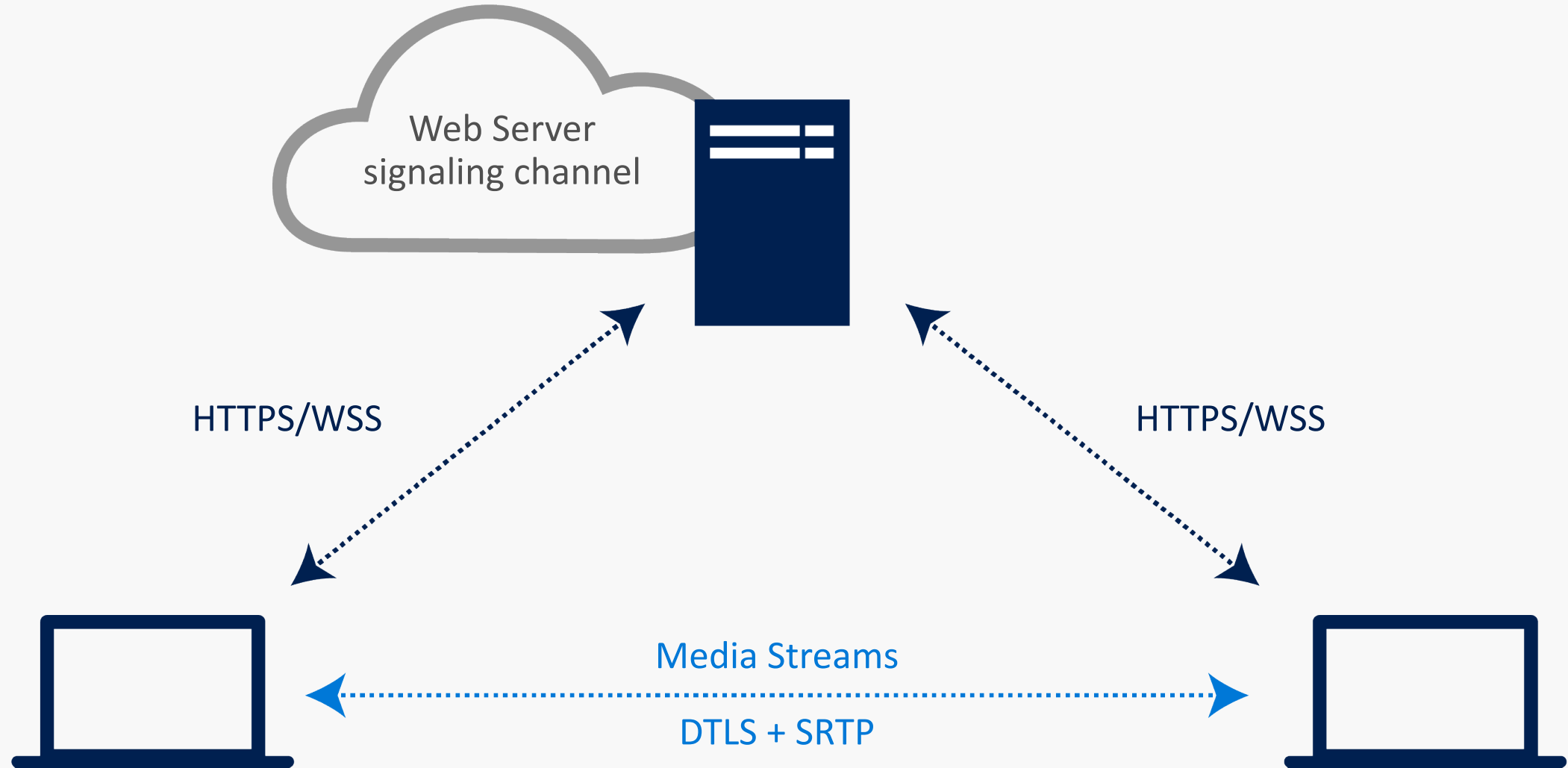
# Edge Limitations (cont'd)

- **ORTC API**
  - RTP/RTCP mux required with DtlsTransport (non-mux possible with SrtpSdeSTransport)
  - No forking support (one IceTransport per IceGatherer)
  - No support for RTCIceTransportController or RTCRtpListener
  - No ssrconflict or rtpunhandled events
  - send() and receive() methods can only be called once
  - Filed bugs: <https://developer.microsoft.com/en-us/microsoft-edge/platform/issues/?page=1&q=ortc>
- **Native WebRTC 1.0 API**
  - RTP/RTCP mux required
  - No support for multi-stream or “Unified Plan”
  - No support for WebRTC 1.0 API object model
  - Filed bugs: <https://developer.microsoft.com/en-us/microsoft-edge/platform/issues/?page=1&q=webrtc>

# Step-by-Step: Building an ORTC Application

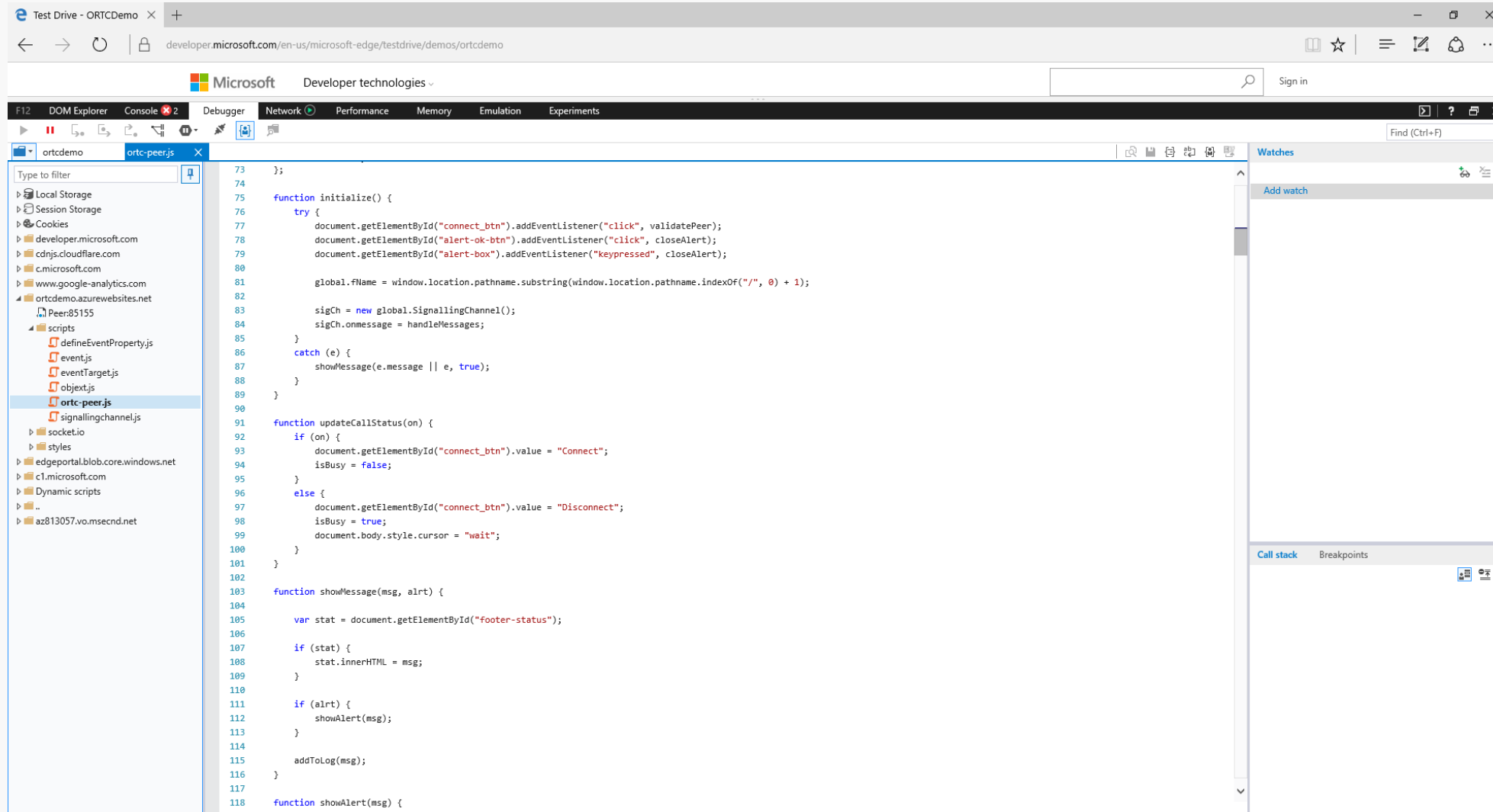
1. Establish a signaling channel
2. Create `MediaStream` object with audio and video tracks
3. Create transports and exchange parameters.
4. Create sender and receiver objects and extract and exchange capabilities
5. Start the ICE and DTLS transports, call `send()` and `receive()` methods
6. Connect incoming media tracks from `RtpReceiver` to audio and video tags

# A Simple Topology



# ORTC Demo Application

<https://developer.microsoft.com/en-us/microsoft-edge/testdrive/demos/ortcdemo/>



# App level code flow

// 2. Create MediaStream object (i.e. Media Capture API) with one audio track and one video track

```
navigator.mediaDevices.getUserMedia({  
  "audio": true,  
  "video": {  
    width: 640,  
    height: 480,  
    facingMode: "user"  
  }  
}).then(  
  gotMedia  
) .catch(  
  gotMediaError  
);
```

// 3. Create transports (ICE gatherer, ICE transport, DTLS transports, etc.) and exchange parameters.

```
function initiateConnection() {
    updateCallStatus();

    var iceOptions = { "gatherPolicy": "all", "iceServers": [{ "urls": "turn:turn-
        testdrive.cloudapp.net:3478?transport=udp", "username": "redmond",
        "credential": "redmond123" }] };

    iceGathr = new RTCIceGatherer(iceOptions);
    iceTr = new RTCIceTransport();
    dtlsTr = new RTCDtlsTransport(iceTr);
    ...
    signalMessage(JSON.stringify({
        params: {
            "ice": iceGathr.getLocalParameters(),
            "dtls": dtlsTr.getLocalParameters()
        }
    }));
    ...
}
```

// 4. Create sender and receiver objects and extract and exchange capabilities

```
function gotMedia(stream) {  
    var audioTracks = stream.getAudioTracks();  
  
    if (audioTracks.length > 0) {  
        var audioTrack = audioTracks[0];  
        local_audio_MST = audioTrack;  
  
        audioSender = new RTCRtpSender(audioTrack, dtlsTr);  
        sendAudioCaps = RTCRtpSender.getCapabilities("audio");  
  
        signalMessage(JSON.stringify({  
            params: {  
                "sendAudioCaps": sendAudioCaps  
            }  
        }));  
    }  
    ...  
}
```



// 5. Start the ICE and DTLS transports

```
if (message.params) {  
    var remote = message.params;  
  
    if (remote.ice) {  
        remoteIceParams = remote.ice;  
        remoteDtlsParams = remote.dtls;  
  
        if(localCandidatesCreated){  
            iceTr.start(iceGathr, remoteIceParams, (selfInfo.dtlsRole && selfInfo.dtlsRol  
=== "client" ? "controlled" : "controlling" ));  
            dtlsTr.start(remoteDtlsParams);  
        }  
    }  
  
    ...  
}
```

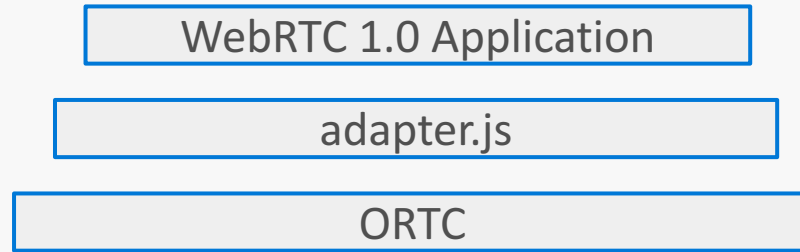
// 5. Call the send() and receive() methods

// 6. Connect incoming media tracks from RtpReceiver to audio and video tags

```
if(remote_audioRecvParams){
    var remote = remote_audioRecvParams;
    var audioRecvParams = util.myCapsToRecvParams(receiveAudioCaps, remote.sendAudioCaps);
    audioRecvParams.encodings.push(util.RTCRtpEncodingParameters(1001, 0, 0, 0, 1.0));
    audioReceiver.receive(audioRecvParams);
    trackCount++;
    if ( trackCount == 2) {
        videoRenderer.srcObject = renderStream;
    }
}
...
if( remote_audioSendParams ){
    var remote = remote_audioSendParams;
    var audioSendParams = util.myCapsToSendParams(sendAudioCaps, remote.receiveAudioCaps);
    audioSendParams.encodings.push(util.RTCRtpEncodingParameters(1001, 0, 0, 0, 1.0));
    audioSender.send(audioSendParams);
}
```

# WebRTC 1.0 Support on ORTC

Using the adapter.js library, you can build audio and video applications that run on Edge, Chrome and Firefox using the current WebRTC 1.0 API.



Latest release of adapter.js:

<https://github.com/webrtc/adapter/blob/master/release/adapter.js>

WebRTC 1.0 sample applications:

<https://github.com/webrtc/samples>

WebRTC 1.0 testbed:

<https://github.com/fippo/testbed>

# What Applications Can Use adapter.js?

- adapter.js supports getUserMedia as well as P2P use of RTCPeerConnection, including:
  - Single-stream video applications supporting H.264/AVC and VP8
    - P2P audio or audio/video chat
  - Conferencing applications using an MCU
    - MCU receives multiple video streams from participants and outputs a composite video stream.
  - addTrack/ontrack
  - Re-negotiation
- Work-in-progress
  - removeTrack
  - Multi-stream audio/video (e.g. Unified Plan)

# RTCPeerConnection Demo (adapter.js)

- In November 2013, Sam Dutton of Google published a “WebRTC In the Real World” tutorial: <https://www.html5rocks.com/en/tutorials/webrtc/infrastructure/>
  - Tutorial example uses sockets.io and a node.js server to support a 1:1 audio/video session.
  - Due to changes in sockets.io and the https: requirement in Chrome getUserMedia, the original code no longer runs, but is available here:
    - <http://simpl.info/rtcpeerconnection/>
    - <https://bitbucket.org/webrtc/codelab/src/master/complete/step5/>
- Simon Pietro Romano updated the code and added data channel support, available here:
  - [https://github.com/spromano/WebRTC\\_Book](https://github.com/spromano/WebRTC_Book)
- With a little “renovation” and addition of adapter.js, code now runs (and interoperates) on Chrome, Firefox and Edge.

# client renovations

- Provided a TURN server URI in IceServers
  - Edge does not support STUN URIs.
- New promise-based APIs
  - navigator.mediaDevices.getUserMedia
  - createOffer/createAnswer
  - setLocalDescription/setRemoteDescription
  - addIceCandidate
- End-of-candidates handling
  - Edge: calling addRemoteCandidate on a null event.candidate required for ICE processing to start.
- Not fixed yet: removal of deprecated APIs (generates warning in Firefox)
  - addStream (in favor of addTrack)
  - onaddstream (in favor of ontrack)

server code (runs in node.js):

<http://internaut.com:8080/~baboba/cluecon-tutorial/adapter/server.js>

client code

<http://internaut.com:8080/~baboba/cluecon-tutorial/adapter/js/main.js>

# For More Information

- 2016 Edge Web Summit talk on Edge RTC platform
  - <https://channel9.msdn.com/Events/WebPlatformSummit/edgesummit2016/ES1608>
- Edge Platform Status
  - <https://developer.microsoft.com/en-us/microsoft-edge/platform/status/>
- Edge RTC FAQ
  - <https://github.com/aboba/edgertc/blob/master/MicrosoftEdgeRTCQA.pdf>
- Edge RTC Roadmap blog
  - <https://blogs.windows.com/msedgedev/2017/01/31/introducing-webrtc-microsoft-edge/#4TFIxBvFftXFc8sJ.97>
- ORTC API
  - <http://draft.ortc.org/>
- Edge ORTC Developer documentation
  - <https://github.com/aboba/edgertc/blob/master/README.md>



# Questions?