WEBRTC WORKSHOP

Adam Bergkvist

@MUM2013

WHAT IS WEBRTC?

- A set of APIs and protocols to enable real-time audio-visual communication and peer-to-peer data natively in a browser
  - No plug-ins
  - Standardized in W3C and IETF
- Main Components
  - getUserMedia() - Get access to the user's devices
  - MediaStream API - Control real-time media in JavaScript
  - RTCPeerConnection - Send real-time media (and data) to others

STANDARDIZATION

- IETF
  - Low level stuff: media codecs
  - On the wire protocols
- W3C
  - API layer

- Contribute
  - Finds bugs in specs
  - Find bugs in implementations

WEBRTC – OVERVIEW

- getUserMedia()
  - Get access to the user’s devices
- MediaStream API
  - Control real-time media in JavaScript
- RTCPeerConnection
  - Send real-time media (and data) to others

GETUSERMEDIA()

› Simple example without error handling

```
<script>
navigator.getUserMedia({ "audio": true, "video": true }, gotStream);

function gotStream(stream) {
    var selfView = document.getElementById("self_view");
    selfView.src = URL.createObjectURL(stream);
}
</script>

<video id=“self_view” autoplay></video>
```

GETUSERMEDIA()

› Simple example with error handling

```
<script>
navigator.getUserMedia({ "audio": true, "video": true }, gotStream, gotError);

function gotStream(stream) {
    var selfView = document.getElementById("self_view");
    selfView.src = URL.createObjectURL(stream);
}

function gotError() {
    notifyUser("No device for you!");
}
</script>

<video id="self_view" autoplay></video>
```

TASKS – INTRO & STEP 1

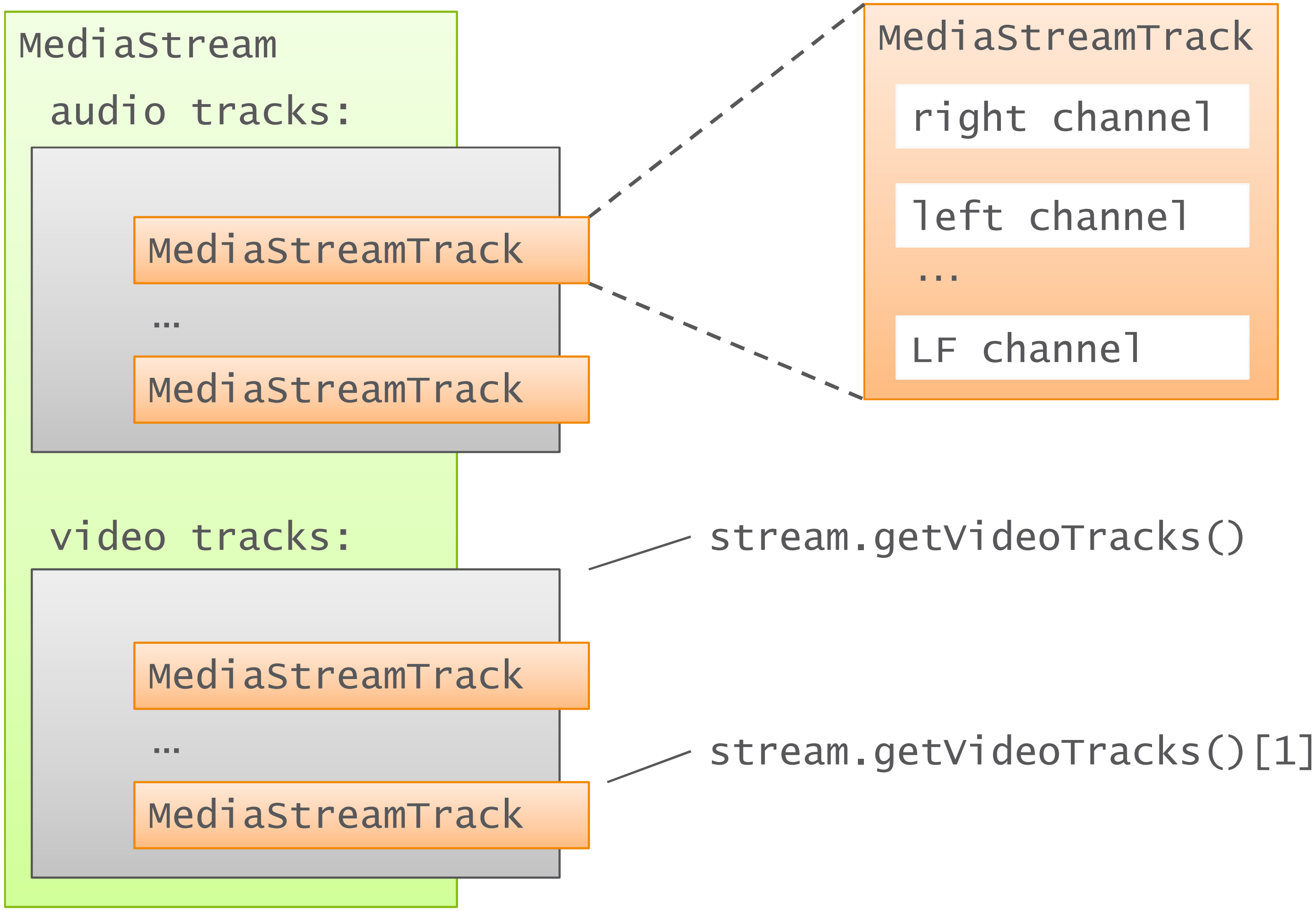
```
# Resources:

* channel_server.js: Combined signaling server and web server written for Node.js.
Web server hosts files from client/ dictionary (defaults to webrtc_example.html).
Signaling server is used together with the signaling_channel.js client library.

* signaling_channel.js : Client library to channel_server.js.

Type "node channel_server.js" to run the server.

getUserMedia() and RTCPeerConnection are still under development and are therefore
vendor prefixed in Chrome. This means that you need to use webkitGetUserMedia() and
webkitRTCPeerConnection().
```

```
# Step 1 - Hair check

The task is to write a simple hair check application. Use getUserMedia() to get
hold of a MediaStream with a video track and show it in a video element.

Start from scratch or use the hair_check.html template. To load hair_check.html,
point your browser to http://localhost:8080/hair_check.html.

You’ll find hints in the template file and the two previous slides.
```

MEDIASTREAM API

- The “glue” of WebRTC
- MediaStreams can be
  - played in <video> & <audio>
  - cloned
  - recorded (with MediaStreamRecorder*)
  - sent to other browsers (with RTCPeerConnection)
  - …

MEDIASTREAM API



MEDIASTREAM API

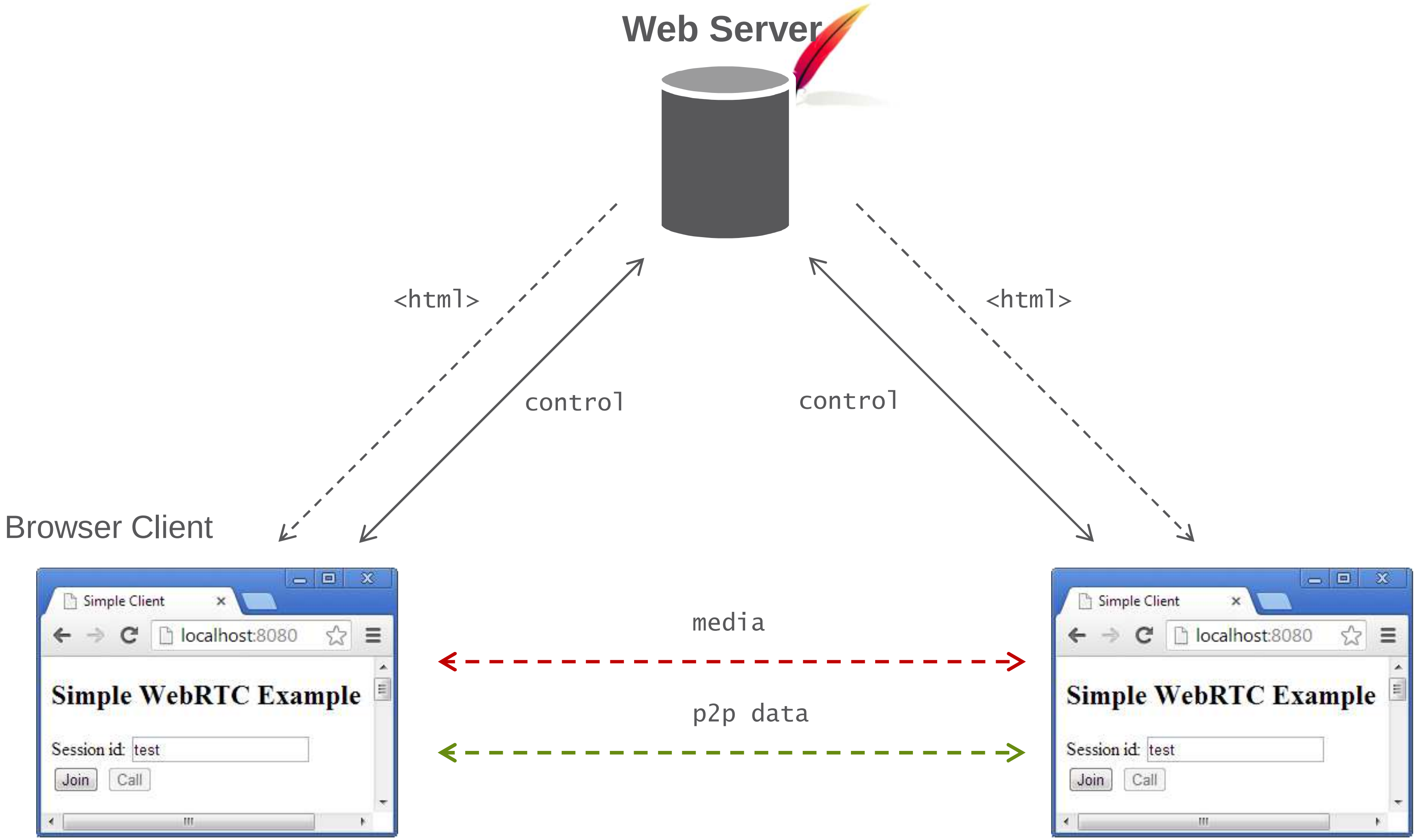
› Extended previous example – Toggle audio on/off

```
<script>
var localStream;
navigator.getUserMedia({ "audio": true, "video": true }, gotStream);

function gotStream(stream) {
    var localStream = stream;
    var selfView = document.getElementById("self_view");
    selfView.src = URL.createObjectURL(stream);
}

function toggleAudio() {
    var track = localStream.getAudioTracks()[0];
    if (track)
        track.enabled = !track.enabled;
}
</script>

<video id="self_view" autoplay></video>
```

PEERCONNECTION

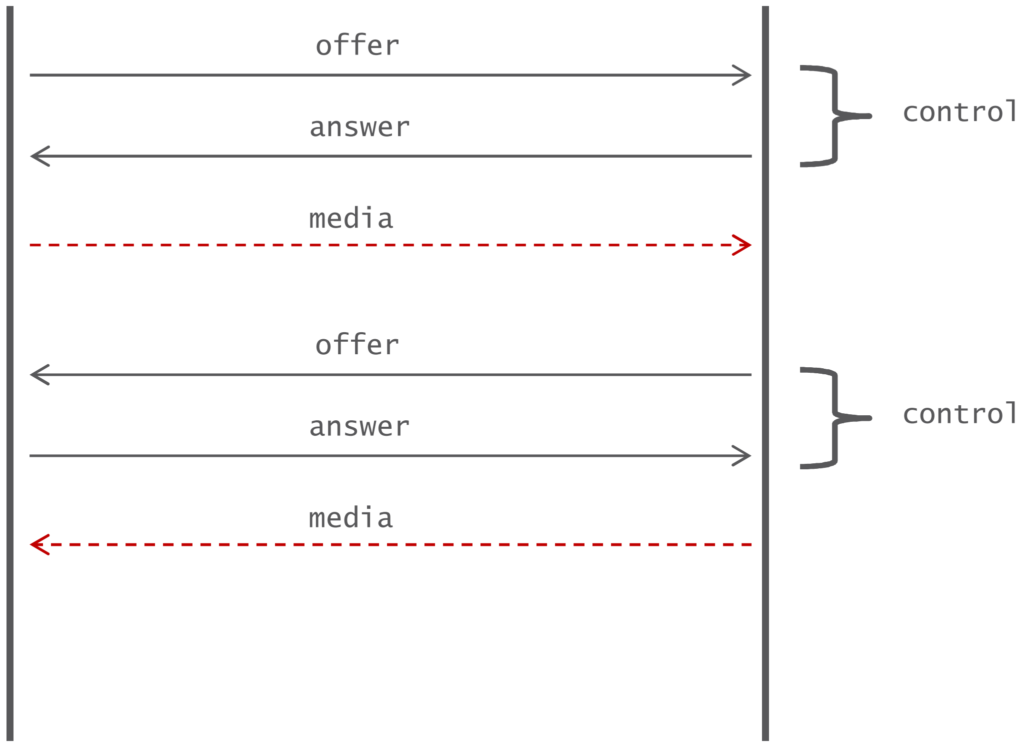
- Connect directly (peer-to-peer) to another browser
  - Through Firewalls/NATs (ICE)
- Share media streams with others
- Other features
  - P2P Data channels
  - Send DTMF
  - Statistics API
  - Identity API
- Out of band signaling channel needed (given for assignment)

WHAT GOES WHERE (P2P)



OFFER/ANSWER

› Offers and answers: describes the media session



ICE

- ICE – Interactive Connectivity Establishment
  - Procedure to do NAT traversal
- ICE Candidate
  - An “address” where you might be reachable
- ICE in WebRTC
  - ICE state machine built into RTCPeerConnection

GET STARTED

- Create and configure RTCPeerConnection
- Handle ICE candidates (ICE trickling)
- Add media to be sent
- Creating offers and answers
- Handling incoming offers and answers
- Render remote streams

CONFIGURE

```
var configuration = { "iceServers": [{ "url": "stun:stun.l.google.com:19302" }] };

var pc = new webkitRTCPeerConnection(configuration);
```

ICE CANDIDATES

```
// send any ice candidates to the other peer
pc.onicecandidate = function (evt) {
    if (evt.candidate)
        signalingChannel.send(JSON.stringify({ "candidate": evt.candidate }));
};

// in message handler (incoming candidates)
var message = JSON.parse(evt.data);
if (message.candidate)
    pc.addIceCandidate(new RTCIceCandidate(message.candidate));
```

ADD MEDIA

```
navigator.getUserMedia({ "audio": true, "video": true }, function (stream) {
    pc.addStream(stream);
});
```

CREATING OFFERS/ANSWERS

```
// let the "negotiationneeded" event trigger offer generation
pc.onnegotiationneeded = function () {
    pc.createOffer(localDescCreated, logError);
}

// create an answer
pc.createAnswer(localDescCreated, logError);

function localDescCreated(desc) {
    pc.setLocalDescription(desc, function () {
        // send pc.localDescription on signaling channel
    }, logError);
}
```

INCOMING OFFERS/ANSWERS

```
// in message handler
var message = JSON.parse(evt.data);
if (message.sdp) {
    var desc = new RTCSessionDescription(message.sdp);
    pc.setRemoteDescription(desc, function () {
        // if we received an offer, we need to create an answer with
        // a call to pc.createAnswer(success, logError);
        // Tip: check pc.remoteDescription.type
    }, logError);
}
```

RENDER REMOTE STREAMS

```
// once remote stream arrives, show it in the remote video element
pc.onaddstream = function (evt) {
    remoteVideo.src = URL.createObjectURL(evt.stream);
};
```

TASKS –STEP 2

```
# Step 2 - Simple video chat

The task is to write a simple video conferencing application that connects two
users that join the same room (or session). WebRTC relies on a signaling channel
between the communicating peers to exchange setup messages (offers and answers).
The webrtc_example.html template uses the provided signaling_channel.js for this
purpose.

To test your solution, run the page in two browser windows or tabs. Once you've got
your solution working, let someone else connect and test it between two computers.

You’ll find hints in the template file and the slides that walks through the
RTCPeerConnection setup procedure.
```

PEER-TO-PEER DATA

- Classic approach
  - Sending client > web server > receiving client
  - HTTP, WebSocket (all TCP under the hood)
- DataChannel
  - P2P – no server involved
  - Configurable
    - Reliable and unreliable
    - Priority
  - Low latency (gaming)

PEER-TO-PEER DATA

```
// initiating side
dataChannel = pc.createDataChannel("chat");
dataChannel.onopen = function () {
    // dataChannel is ready to be used
    enableChat();
};
```

```
// receiving side
pc.ondatachannel = function (evt) {
    dataChannel = evt.channel;
    enableChat();
};
```

```
// used by both sides
function enableChat(() {
    // send messages
    sendButton.onclick = function () {
        dataChannel.send(sendText.value);
    };
    // receive messages
    dataChannel.onmessage = function (evt) {
        printInChat("peer: " + evt.data);
    };
}
```

TASKS –STEP 3 (EXTRA)

```
# Step 3. Peer-to-Peer text chat with DataChannel (Extra)

This extra task is about sending arbitrary data peer to peer. You could actually
use the signaling channel to send the chat messages via the web server, but in some
situations, such as in games, it's important with low latency.

There's no template for this task, but use your code from step 2 and rip out the
getUserMedia(), pc.addStream() and media element parts. Let one side create a
DataChannel and the other wait for the resulting event. When both sides have open
channels, enable the chat UI.

Modify the RTCPeerConstructor as below to enable RTP data channels in chrome.
pc = new webkitRTCPeerConnection(configuration, { "optional": [{ "RtpDataChannels":
true }] });

Review the two lasts slides for hits.

If you get this far we always come up with more extra tasks. :)

Good luck!
```

LINKS

- W3C
  - WebRTC Working Group
    - http://www.w3.org/2011/04/webrtc/
  - Specifications
    - http://dev.w3.org/2011/webrtc/editor/getusermedia.html
    - http://dev.w3.org/2011/webrtc/editor/webrtc.html
- IETF
  - RTCWeb Working group
    - http://tools.ietf.org/wg/rtcweb/

ERICSSON