RTCMultiConnection coding tricks

- Get First Peer Object
- Validate First User's Audio Tracks
- Adding screen at runtime?
- User is expected to be NOT having microphone or webcam?
- Data-only connection?
- Facing media capturing error?
- Worried about too many messages?
- Stereo audio?
- Voice Activity Detection
- How to handle who is typing?
- Share screen using custom methods
- Replace Tracks

Get First Peer Object

Useful for one-to-one connection.

```
var firstUser = connection.peers[connection.getAllParticipants()[0]];

var nativePeer = firstUser.peer;
```

Validate First User's Audio Tracks

You can use same code for video tracks as well.

```
var firstUserAudioTrack =
connection.peers[connection.getAllParticipants()[0]].peer.getRemoteStreams()[0].getAudioTracks()[0];
```

```
if(!firstUserAudioTrack) {
  alert('First user is NOT sending any audio track');
}

if(!firstUserAudioTrack.enabled) {
  alert('First user disabled his audio track');
}

if(firstUserAudioTrack.muted) {
  alert('First user muted his audio track');
}
```

Adding screen at runtime?

```
document.getElementById('share-screen').onclick = function() {
  this.disabled = true;

  if (connection.UA.isChrome) {
    connection.DetectRTC.screen.getChromeExtensionStatus(function(status) {
      if (status == 'installed-enabled') {
        connection.addStream({
          screen: true,
          oneway: true
        });
      }
```

```
        if (status == 'installed-disabled') {
            alert('Please enable chrome extension and reload the page. It seems that chrome extension is installed but disabled. Or maybe you are using localhost or non-allowed domain.');
        }

        if (status == 'not-installed') {
            alert('Please install chrome extension. It seems that chrome extension is not installed.');
        }

        if (status == 'not-chrome') {
            alert('Please either use Chrome or Firefox to share screen.');
        }
    });
  }

  if (connection.UA.isFirefox) {
    connection.addStream({
        screen: true,
        oneway: true
    });
  }
};
```

User is expected to be NOT having microphone or webcam?

```
if (connection.UA.isChrome) {
  connection.DetectRTC.load(function() {
```

```
        if (!connection.DetectRTC.hasMicrophone) {
            connection.mediaConstraints.audio = false;
            connection.session.audio = false;
        }

        if (!connection.DetectRTC.hasWebcam) {
            connection.mediaConstraints.video = false;
            connection.session.video = false;
        }
    });
}

document.getElementById('join').onclick = function() {
    connection.channel = prompt('Enter room-id');

    if (!connection.UA.isChrome) {
        connection.join({
            sessionid: connection.channel,
            userid: connection.channel,
            extra: {},
            session: connection.session
        });

        return;
    }
```

```
    connection.DetectRTC.load(function() {
        if (!connection.DetectRTC.hasMicrophone) {
            connection.session.audio = false;
            connection.mediaConstraints.audio = false;
        }

        if (!connection.DetectRTC.hasWebcam) {
            connection.session.video = false;
            connection.mediaConstraints.video = false;
        }

        connection.join({
            sessionid: connection.channel,
            userid: connection.channel,
            extra: {},
            session: connection.session
        });
    });
};
```

Data-only connection?

```
connection.sdpConstraints.mandatory = {
    OfferToReceiveAudio: false,
    OfferToReceiveVideo: false
```

```
};
```

Facing media capturing error?

```
connection.mediaConstraints = {
    video: true,
    audio: true
};
```

Worried about too many messages?

To reduce number of messages, you can call open method with dontTransmit:true. Also, you should use trickleIce=false:

```
connection.trickleIce = false;

// the person who'll moderator the room
connection.userid = connection.channel
connection.open({
    dontTransmit: true,
    sessionid: connection.channel,
    onMediaCaptured: function() { }
});

// the person who will join
connection.join({
    sessionid: connection.channel,
    userid: connection.channel,
    extra: {},
```

```
    session: connection.session
});
```

Stereo audio?

```
connection.mediaConstraints.audio = {
   mandatory: {},
   optional: [{
      echoCancellation: false
   }]
};
```

And:

http://www.rtcmulticonnection.org/docs/processSdp/#maxaveragebitrate

Voice Activity Detection

RTCMultiConnection-v2.2.4 (latest) updates disables onspeaking event for remote stream; which means that only local media stream's activity is detected using hark.js.

Below code-snippets are using onmute, onstream, onspeaking and onmessage events to implement following tasks:

1. Ask hark.js to detect activity of local stream(s).
2. As soon as onspeaking event is fired; send a custom message or datachannel message to remote peers to tell them sync the activity.
3. Remote peers can either receive message using onmessage event or onCustomMessage event.
4. onmute is used to ask hark.js pause watching the voice activity i.e. stop-watching until stream.resume method is invoked.
5.

stream.pause and stream.resume are NEW methods implemented in v2.2.4 and MUST be used using if-block.

Here is how to implement voice activity detection in your applications:

```
// don't forget to link this:
// https://github.com/muaz-khan/RTCMultiConnection/blob/master/dev/hark.js

connection.onspeaking = function(e) {
    if (connection.numberOfConnectedUsers > 0) {
        // You can even use "sendCustomMessage"
        connection.send({
            streamid: e.streamid,
            speaking: true
        });
    }
    e.mediaElement.style.border = '3px dotted red';
};

connection.onsilence = function(e) {
    // "numberOfConnectedUsers" is checked because we are using data channels
    // you shouldn't use this block if you're using "sendCustomMessage"
    if (connection.numberOfConnectedUsers > 0) {
        // You can even use "sendCustomMessage"
        connection.send({
            streamid: e.streamid,
```

```
            silence: true
        });
    }

    e.mediaElement.style.border = '1px solid rgb(0, 0, 0)';
};

connection.onmute = function(event) {
    if (event.session.video) {
        event.mediaElement.src2 = event.mediaElement.src;
        event.mediaElement.src = '';
        event.mediaElement.style.background = 'transparent url(https://cdn.webrtc-
experiment.com/images/muted.png) no-repeat center center';
        return;
    }

    if (event.stream.pause) {
        // for audio-streams
        // ask hark.js to resume looping and checking for voice activity
        event.stream.pause();
    }
};

connection.onunmute = function(event) {
    if (event.session.video) {
        event.mediaElement.src = event.mediaElement.src2;
```

```
        event.mediaElement.play();
        event.mediaElement.style.background = '';
        return;
    }

    if (event.stream.resume) {
        // for audio-streams
        // ask hark.js to stop looping and checking for voice activity
        event.stream.resume();
    }
};

connection.session = {
    audio: true,
    video: true,
    data: true //------ data-channels
};

connection.onmessage = function(event) {
    if (event.data.speaking) {
        var mediaElement = document.getElementById(event.data.streamid);
        if (mediaElement) mediaElement.style.border = '3px dotted red';
        return;
    }
};
```

```
connection.onstream = function(e) {
   e.mediaElement.id = e.streamid; // ---------- set ID
   document.body.appendChild(e.mediaElement);
};
```

How to handle who is typing?

As soon as you'll press a key, a lastMessageUUID number will be generated; which will be used until you press ENTER key to send the message.

As soon as onmessage event is fired, remote users are using same lastMessageUUID object to create a DIV and display message e.g. "Remote peer is typing..."

Below code snippet is also having modify and remove methods; which allows you implement Skype-like behavior in your applications.

```
var numberOfKeys = 0;
var lastMessageUUID;

document.querySelector('input[type=text]').onkeyup = function(e) {
   numberOfKeys++;

   if (numberOfKeys > 3) {
      numberOfKeys = 0;
   }

   if (!numberOfKeys) {
      if (!lastMessageUUID) {
         lastMessageUUID = Math.round(Math.random() * 999999999) + 9995000;
```

```
    }

    connection.send({
        typing: true,
        lastMessageUUID: lastMessageUUID
    });
}

if (!this.value.replace(/^\s+|\s+$/g, '').length) {
    connection.send({
        stoppedTyping: true,
        lastMessageUUID: lastMessageUUID
    });
    return;
}

if (e.keyCode != 13) {
    return;
}

// removing trailing/leading whitespace
this.value = this.value.replace(/^\s+|\s+$/g, '');

connection.send({
    message: this.value,
```

```
        lastMessageUUID: lastMessageUUID
    });
    appendDIV(this.value, lastMessageUUID, true);
    lastMessageUUID = null;

    this.value = '';
};

connection.onmessage = function(event) {
    if (event.data.typing) {
        appendDIV(event.userid + ' is typing..', event.data.lastMessageUUID);
        return;
    }

    if (event.data.stoppedTyping) {
        var div = document.getElementById(event.data.lastMessageUUID);
        if (div) div.parentNode.removeChild(div);
        return;
    }

    if (event.data.modified) {
        var div = document.getElementById(event.data.lastMessageUUID);
        if (div) {
            div.innerHTML = event.data.message;
        }
```

```
        return;
    }

    if (event.data.removed) {
        var div = document.getElementById(event.data.lastMessageUUID);
        if (!div) return;
        div.parentNode.removeChild(div);
        return;
    }

    appendDIV(event.data.message, event.data.lastMessageUUID);
};

var chatContainer = document.querySelector('#chat-output');

// a custom method used to append a new DIV into DOM
function appendDIV(message, messageUUID) {
    var existing = false,
        div;

    if (document.getElementById(messageUUID)) {
        div = document.getElementById(messageUUID);
        existing = true;
    } else {
        div = document.createElement('div');
```

```
        if (messageUUID) {
            div.id = messageUUID;
        }
    }

    div.innerHTML = message;

    if (!existing) {
        chatContainer.insertBefore(div, chatContainer.firstChild);
    }

    div.tabIndex = 0;
    div.focus();

    document.querySelector('input[type=text]').focus();
}

function modify(lastMessageUUID, modifiedValue) {
    connection.send({
        message: modifiedValue,
        lastMessageUUID: lastMessageUUID,
        modified: true
    });
}
```

```
function remove(lastMessageUUID) {
    connection.send({
        lastMessageUUID: lastMessageUUID,
        removed: true
    });
}
```

Share screen using custom methods

RTCMultiConnection is having attachExternalStream that can be used to attach multiple external screen streams.

First of all, please enable screen sharing command-line flags:

https://github.com/muaz-khan/RTCMultiConnection/wiki/Screen-Sharing

Or deploy your own chrome extension. Then you can use Screen-Capturing.js to capture screen.

There is Firefox-Extension as well.

Here is how to use attachExternalStream method:

```
var screen_constraints = {
    chromeMediaSource: 'screen'
};

var video_constraints = {
    video: screen_constraints
```

```
};

navigator.webkitGetUserMedia(video_constraints, onScreenSuccess, onScreenFailure);

function onScreenSuccess(screen_stream) {
    connection.dontCaptureUserMedia = true;
    connection.attachExternalStream(screen_stream, true);
}

function onScreenFailure(error) {
    alert(JSON.stringify(error));
}
```

dontCaptureUserMedia is merely used to ask RTCMultiConnection that, don't try to capture any stream itself. It isn't mandatory though.

It is suggested to also set sdpConstraints.

Detect Who is Speaking

"onspeaking" and "onsilence" events has been removed since v3 however you can easily re-implement such events.

This wiki explains how to re-implement both events in RTCMultiConnection v3.

First of all, link hark.js:

```
<script src="https://cdn.webrtc-experiment.com/hark.js"></script>
```

Now paste following code in your HTML/PHP files:

```
connection.onstream = function(event) {
    initHark({
        stream: event.stream,
        streamedObject: event,
        connection: connection
    });
};

connection.onspeaking = function (e) {
    // e.streamid, e.userid, e.stream, etc.
    e.mediaElement.style.border = '1px solid red';
};

connection.onsilence = function (e) {
    // e.streamid, e.userid, e.stream, etc.
    e.mediaElement.style.border = '';
};

connection.onvolumechange = function(event) {
    event.mediaElement.style.borderWidth = event.volume;
};

function initHark(args) {
```

```
if (!window.hark) {
    throw 'Please link hark.js';
    return;
}

var connection = args.connection;
var streamedObject = args.streamedObject;
var stream = args.stream;

var options = {};
var speechEvents = hark(stream, options);

speechEvents.on('speaking', function() {
    connection.onspeaking(streamedObject);
});

speechEvents.on('stopped_speaking', function() {
    connection.onsilence(streamedObject);
});

speechEvents.on('volume_change', function(volume, threshold) {
    streamedObject.volume = volume;
    streamedObject.threshold = threshold;
    connection.onvolumechange(streamedObject);
});
```

```
}
```

Replace Tracks

```
setTrack(track, type) {
  return new Promise((resolve, reject) => {
    // Check track and type are valid
    if (!track || !type) reject(new Error('You don\'t set track or type track.'));
    // Check that track is not ended
    if (track.readyState === 'ended') reject(new Error('You don\'t can\'t replace with an "ended" track.'));
    // Each all participants for send screen track
    const foo = this.connection.getAllParticipants().forEach((pid) => {
      // Get peer connection of every participant
      const { peer } = this.connection.peers[pid];
      // Check that peer have senders or next peer
      if (!peer.getSenders) return;
      // Get all
      peer.getSenders().forEach((sender) => {
        // Check the sender of peer and track, or next sender
        if (!sender || !sender.track) return;
        //  Get the track type and replace track with the new stream if match
        if (sender.track.kind === type && track) sender.replaceTrack(track);
      });
    });
    // Wait for replace to all peers and ready
    Promise.all(foo).then(() => resolve()).catch(() => reject());
```

```
  });
}
```

via: https://github.com/muaz-khan/RTCMultiConnection/issues/652#issuecomment-508601407