Bern University of Applied Sciences

BTI7311p - PROJECT 1

State of WebRTC and its pitfalls

Authors:
Frédéric N. Lehmann

Tutor:
Dr. Simon Kramer

March 8, 2020

Abstract

In this paper we will explore the state of WebRTC and its architectural pitfalls. WebRTC is a web API for real-time communication on a peer to peer basis. Goal of this work will be to document the state of WebRTC and its real world applications. We will explore those possibilities on the basis of example applications and their implementations and limitations.

Contents

List of Figures

1 Introduction

WebRTC is short for web real-time communication, it is an API that modern browser support and can be used by web developers to implement a peer to peer communication. It can be used to capture and stream audio and/or video data, as well as to exchange arbitrary data between browser without requiring an intermediary.

1.1 Support

All major browser support WebRTC in its newest release. Older versions might not, or only partially, implement this API so the Adapter.js [1] project should be considered for productive solutions. For detailed information on supported browsers use caniuse [2].

1.2 Session Establishment

The session establishment uses different network methods to create connection to a peer. This also includes substitutions for situations where the default connection can not be established.

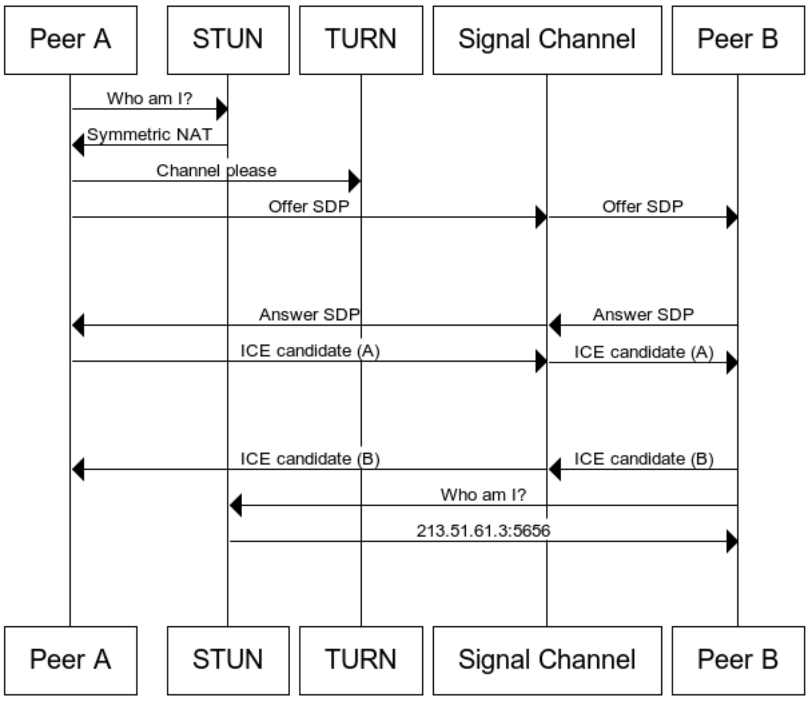

Figure 1.1: WebRTC Complete communication schema, `https://developer.mozilla.org/en-US/docs/Web/API/WebRTC_API/Connectivity`

1.2.1 Network Address Translation (NAT)

Is used to give devices in a network a public IP address. This is achieved by translating requests from the device's private IP to the router's public IP with a unique port. The goal is to not need a unique public IP for each device.

1.2.2 Session Traversal Utilities for NAT (STUN)

This protocol is used to discover the public address of the peer. It also will determine any restrictions that would prevent a direct connection with a peer.

The peer sends a 'who am i' request to a STUN server which responds with the public address of the peer.

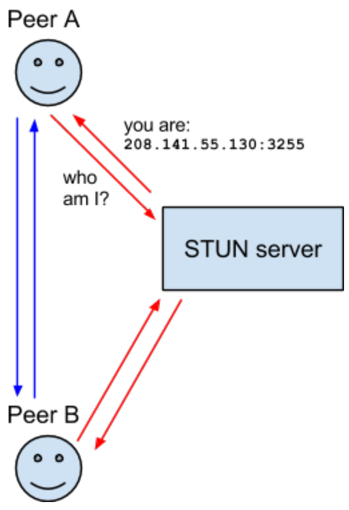

Figure 1.2: STUN communication schema, `https://developer.mozilla.org/en-US/docs/Web/API/WebRTC_API/Protocols`

There are some open STUN servers available:

- stun.l.google.com:19302
- stun[1-4].l.google.com:19302
- stunserver.org
- stun.schlund.de
- stun.voipstunt.com

There can be found more online.

1.2.3 Traversal Using Relays around NAT (TURN)

If STUN can't be used, because for example 'Symmetric NAT' is employed in the network, TURN will be used as fallback. This is achieved by opening a connection with a TURN server, this server then will relaying all information through that server.

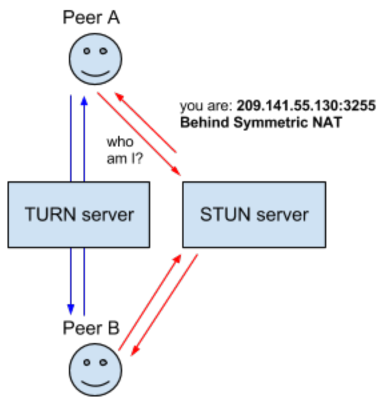

Figure 1.3: TURN communication schema, `https://developer.mozilla.org/en-US/docs/Web/API/WebRTC_API/Protocols`

There are open TURN servers, for example provided by google. But this will mean all communication is going threw a foreign server which might not be acceptable.

1.2.4 Session Description Protocol (SDP)

This standard describes the multimedia content of a connection. This includes a resolution, formats, codecs, encryption, etc. basically it is the metadata describing the content not the content itself.

1.2.5 ICE Candidates

Peers have to exchange information about the network connection, this is known as an ICE candidate. Each peer proposes its best candidate, and will work down to the worst candidate until they agree on a common candidate. This wil, be UDP ideally but can be TCP as well.

1.3 Security

Generally WebRTC traffic is encrypted using DTLS. Traffic that is relayed over a TURN server is not necessarily end-to-end encrypted.

Confidentiality for the application data relayed by TURN is best provided by the application protocol itself, since running TURN over TLS does not protect application data between the server and the peer. If confidentiality of application data is important, then the application should encrypt or otherwise protect its data. For example, for real-time media, confidentiality can be provided by using SRTP. [4]

1.4 Related topics

There are multiple related topics to WebRTC. In this section we'll try to give a quick overview over the most important ones.

Media Capture and Streams API

This API is heavily related to WebRTC, it provides support for streaming audio and video data. Provided are interfaces and methods for working with the success and error callbacks when using the data asynchronously and the events that are fired during the process, as well as the constraints associated with data formats.

1.4.1 Web media technologies

This topic represents all available API's for creating, presenting and managing audio, video and other media. Contained are HTML elements, DOM interfaces and other features.

2 WebRTC example applications

In this chapter we'll show some example applications and implementations. These are in a minified version and do not necessarily represent the best practices that should be applied. Nevertheless, there will be sources provided where those best practices can be deducted.

2.1 Video chat

2.1.1 Accessing client media

Feature check

First we should check if the current environment supports the needed API's.

```
function hasUserMedia() {
  return !!(navigator.mediaDevices
    && navigator.mediaDevices.getUserMedia);
}
```

https://codesandbox.io/s/feature-check-xqv8t

Access media

Then when we have checked if the API's is present we can access client medias, given the user has given his consent.

First we'll add a video channel so the user can see the input from the camera.

```
<video autoplay></video>
```

For more details on best practices with video medias https://developers.google.com/web/fundamentals/med

Then we use this code to access the media and present it with the video tag.

```
const constraints = {
  audio: true,
  video: true
};

const video = document.querySelector("video");

if (hasUserMedia()) {
  navigator
    .mediaDevices
    .getUserMedia(constraints)
    .then(stream => {
      video.srcObject = stream;
    });
}
```

https://developer.mozilla.org/en-US/docs/Web/API/MediaDevices/getUserMedia

https://codesandbox.io/s/access-device-uwvby

2.1.2 Establish connection

```
import { hasUserMedia, startStream } from "./media";

async function start() {
  // Check if the client supports the needed API
  if (!hasUserMedia()) return;

  // Start the video / audio stream
  const stream = await startStream();

  // Create sender / reciever connection
  const sender = new RTCPeerConnection();
  const reciever = new RTCPeerConnection();

  // Set ICE candidate
  sender.onicecandidate = e =>
    !e.candidate || reciever.addIceCandidate(e.candidate);
  reciever.onicecandidate = e =>
    !e.candidate || sender.addIceCandidate(e.candidate);

  // Listen on incoming stream
  reciever.ontrack = e => {
    document.querySelector("#remoteVideo").srcObject = e.streams[0];
  };

  // Bind stream to sender
  stream.getTracks().forEach(track => sender.addTrack(track, stream));

  // Create offer and set description
  const offer = await sender.createOffer({
    offerToReceiveAudio: 1,
    offerToReceiveVideo: 1
  });
  await sender.setLocalDescription(offer);
  await reciever.setRemoteDescription(offer);

  // Create answer from reciever and set description
  const answer = await reciever.createAnswer();
  await reciever.setLocalDescription(answer);
  await sender.setRemoteDescription(answer);
}

(function() {
  document.querySelector("#connectButton").addEventListener("click",
      start);
})();
```

2.1.3 Send data

2.1.4 Disconnect

3 Learnings

3.0.1 STUN / TURN Server

To not be dependent from open STUN/TURN servers one can deploy his own servers. There are open source projects covering this case. For example coturn [3].

3.0.2 Jitsi Videobridge

Video conferencing can be a resource intensive task for a browser. Which leads it to be a solution that is not really scalable. This is where Jitsi comes to the rescue. Jitsi Videobridge is an open source lightweight video conferencing server. It is used to implement scalable video conferencing platforms. It is an Selective Forwarding Unit (SFU) which relays video streams between peers.

Bibliography

[1] *Adapter.js*. URL: `https://github.com/webrtc/adapter` (visited on 03/03/2020).

[2] *Can I Use*. 2020. URL: `https://caniuse.com/`.

[3] *Coturn*. URL: `https://github.com/coturn/coturn` (visited on 03/03/2020).

[4] *TURN security*. URL: `https://tools.ietf.org/html/rfc5766#section-17.1.6` (visited on 03/03/2020).