WebRTC explained

WebRTC is a collection of standards and protocols that enable clients to establish a P2P connection that can be used to stream video and audio or send data without relying on plugins or other third-party software. The functionality is available through a high-level browser API.

Three different interfaces are being used when working with WebRTC:

- **RTCDataChannel** - Channel to transfer data. Sample (http://io13webrtc.appspot.com/#37)
- **MediaStream** - Channel to transfer audio and video. Sample (http://io13webrtc.appspot.com/#17)
- **RTCPeerConnection** - Used for linking of two peers. Both clients need to instantiate a `RTCPeerConnection` before being able to create a `DataStream` or `MediaStream`. Sample (http://io13webrtc.appspot.com/#32)

WebRTC use these concepts and technologies to establish a connection:

- **Session Description Protocol (SDP)** - Peers need to agree on a contract before establishing a connection. SDP is used to describe the details of this contract. ”Peer 1” sends a SDP offer to ”Peer 2” who generates an SDP answer that is returned to ”Peer 1”. ”Peer 2” either accept or rejects the SDP offer sent from ”Peer 1”.

- **Interactive Connectivity Establishment (ICE)** - To be able to reach each other, the two peers needs to exchange addresses and ports. Usually a peer is located behind a NAT, firewall or other barriers, which means that it doesn't expose a direct public IP address. The process of ICE makes it possible to traverse the NAT. The protocols used for this are STUN (https://en.wikipedia.org/wiki/STUN) and TURN (https://en.wikipedia.org/wiki/Traversal_Using_Relays_around_NAT).

- **SignalR** - SignalR on the server acts a middleman to help peers initiate a connection by exchanging SDP offer and SDP answer. Once a connection is established, the peers communicate P2P without any involvement of server. This reduce load on the server.

PoC

WebRTC proof of concept code: PoC (https://github.com/jimmybengtsson/grupp03-redriver/tree/videochat-poc)
WebRTC proof of concept URL: PoC (https://redrivervideocallpoc.azurewebsites.net/)

Video call

1. Peer 1 instantiates a RTCPeerConnection - A url to Googles STUN-server is added. This server is being used to get the identity of the peers.

```
const PC_CONFIG = { iceServers: [{ urls: ['stun:stun.l.google.com:19302'] }] };

this.pc = new RTCPeerConnection(PC_CONFIG);
```

2. Peer 1 creates a MediaStream - The API method `addStream()` is used to send a local video/audio stream from the computer.

```
pc.addStream(localStream);
```

3. Peer 1 creates a SDP offer - The API method `createOffer()` is used to create a SDP offer for Peer 2. After the off has been created, a local description for the peer is set by the API method `setLocalDescription()`. This description include whether it is a video/audio or data transfer, size of video, types of data and other useful information about the requested connection.

```
createOffer() {
            this.pc.createOffer()
                .then(this.pc.setLocalDescription(desc);)
                .catch(err => console.log(err));
            return this;
        }
```

4. Peer 1 gathers ICE candidates - `RTCIceCandidate()` gather information about peers identities and what server to use when establishing a RTCPeerConnection.

```
addIceCandidate(candidate) {
            if (candidate) {
                const iceCandidate = new RTCIceCandidate(candidate);
                this.pc.addIceCandidate(iceCandidate);
            }
            return this;
        }
```

5. Peer 1 sends offer to Peer 2 - To send the initial offer to the other peer, SignalR is used both on client and server. Client use HubConnection to invoke the method `Call` on server who then send the SDP offer to Peer 2.

```
import { HubConnection } from '@aspnet/signalr';
let userConnection = new HubConnection(serverUrl);

userConnection.invoke('Call', { to: this.friendID, sdp: desc });
```

6. Peer 2 instantiates a RTCPeerConnection - When Peer 2 receives a offer from Peer 1, a `RTCSessionDescription()` is created with information about both peers.

```
userConnection.on('call', (data) => {
                if (data.data.sdp) {
            const rtcSdp = new RTCSessionDescription(data.data.sdp);
                    this.pc.setRemoteDescription(rtcSdp);
                    if (data.data.sdp.type === 'offer') this.pc.createAnswer();
                } else this.pc.addIceCandidate(data.data.candidate);
            });
```

7. Peer 2 generates a SDP answer based on Peer 1's offer - `createAnswer()` creates an answer for Peer 1 and description for Peer 2 is set with `setLocalDescription()`.

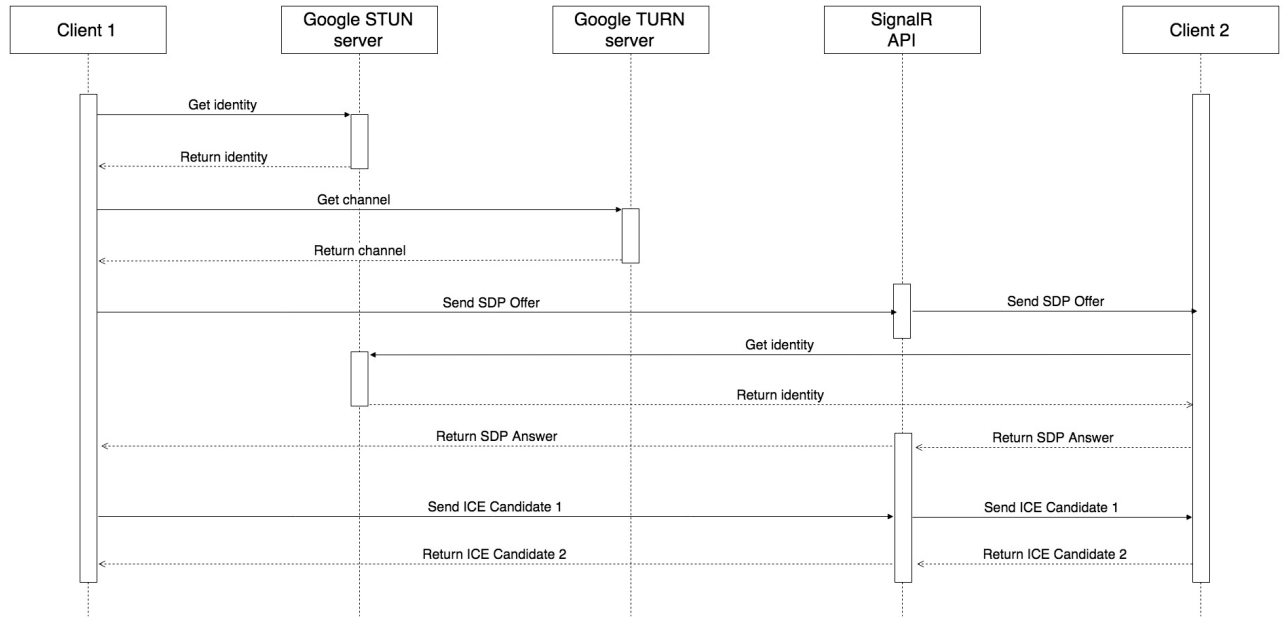
```
createAnswer() {
            this.pc.createAnswer()
                .then(this.pc.setLocalDescription(desc))
                .catch(err => console.log(err));
            return this;
        }
```

8. Peer 2 gathers ICE candidates – The method `RTCIceCandidate()` is also used to get information about Peer 2.

9. Peer 2 sends answer to Peer1 – SignalR is used to send back an answer to Peer 2.

```
this.connection.invoke('Call', { to: this.friendID, sdp: desc });
```

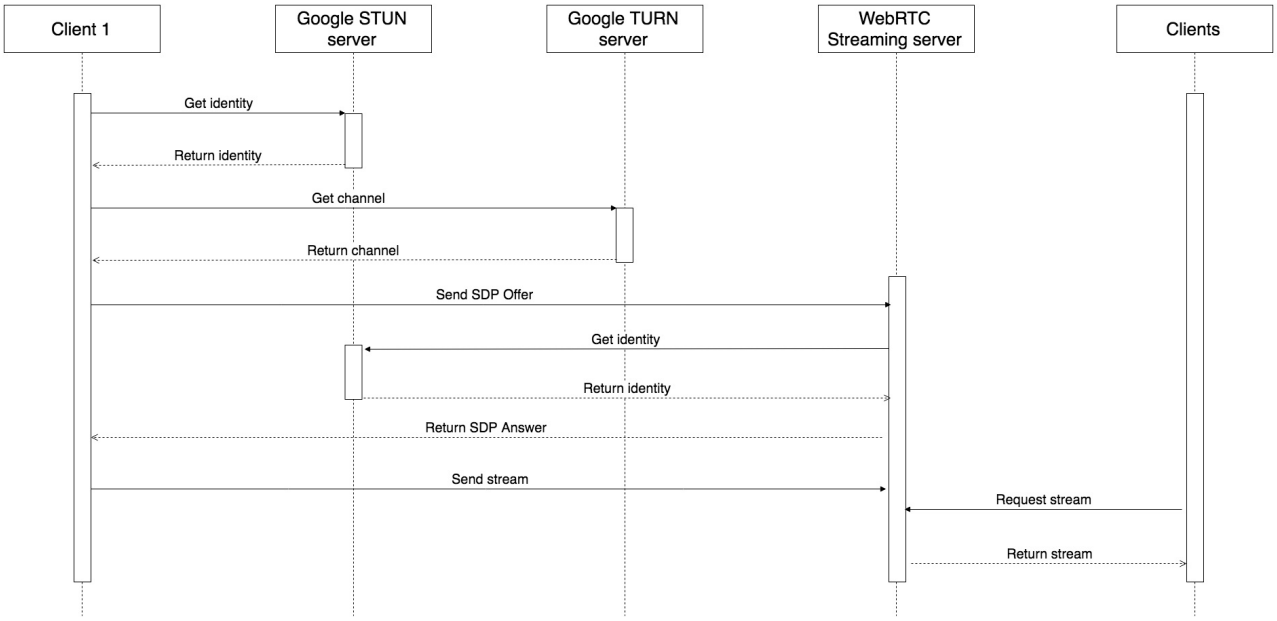
Diagram



Live streaming

Same process as video calls but instead of creating a connection between peers, it is established between one peer and the server. Once a MediaStream is established, other peers can make a request access to it from server.

Diagram


Client 1
Google STUN server
Google TURN server
WebRTC Streaming server
Clients
Get identity
Return identity
Get channel
Return channel
Send SDP Offer
Get identity
Return identity
Return SDP Answer
Send stream
Request stream
Return stream