

WEBRTC

- 2 PEERS NEED TO ESTABLISH A DIRECT CONNECTION TO COMMUNICATE.
⇒ DIFFICULT DUE TO FIREWALLS AND NATs.

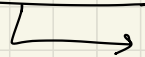


ICE = "INTERACTIVE CONNECTIVITY ESTABLISHMENT"

WEBRTC HAS APIS TO COMMUNICATE WITH AN ICE SERVER.

- EACH PEER CONNECTION IS HANDLED BY RTCPeerConnection OBJECT.

RTCPeerConnection (RTCConfiguration)



CONTAINS INFORMATION ON
SERVER ICE TO USE.

⇒ ONCE WE'VE THE RTCPeerConnection,

WE NEED TO CREATE AN SDP OFFER / ANSWER DEPENDING IF WE'RE
THE CALLING OR RECEIVING PEER.

ONCE THE SDP OFFER/ANSWER IS CREATED, IT MUST BE SENT TO THE REMOTE PEER THROUGH A DIFFERENT CHANNEL "SIGNALING"

```
async function makeCall() {
  const configuration = {'iceServers': [{'urls': 'stun:stun.l.google.com:19302'}]}
  const peerConnection = new RTCPeerConnection(configuration);
  signalingChannel.addEventListener('message', async message => {
    if (message.answer) {
      const remoteDesc = new RTCSessionDescription(message.answer);
      await peerConnection.setRemoteDescription(remoteDesc);
    }
  });
  const offer = await peerConnection.createOffer();
  await peerConnection.setLocalDescription(offer);
  signalingChannel.send({'offer': offer});
}
```

COMMUNICATION CHANNEL THAT RELAYS MESSAGES BETWEEN PEERS (WEBSOCKET)

ANSWER TO THE CONNECTION OFFER

EVENT TYPE

CREATES A RTCSessionDescription

THE RTCSessionDescription IS SET AS LOCAL DESCRIPTION

CALLING
SIDE

```
const peerConnection = new RTCPeerConnection(configuration);
signalingChannel.addEventListener('message', async message => {
  if (message.offer) {
    peerConnection.setRemoteDescription(new RTCSessionDescription(message.offer));
    const answer = await peerConnection.createAnswer();
    await peerConnection.setLocalDescription(answer);
    signalingChannel.send({'answer': answer});
  }
});
```

CREATE AN ANSWER TO THE RECEIVED OFFER

RECEIVING
SIDE

→ NOW BOTH CLIENT AND RECEIVER KNOW THE CAPABILITIES OF THE REMOTE PEER.
⇒ CONNECTION STILL NOT READY!

2 PEERS NEED TO EXCHANGE CONNECTIVITY INFORMATION THROUGH

ICE

→ TURN

↓
STUN

```
// Listen for local ICE candidates on the local RTCPeerConnection
peerConnection.addEventListener('icecandidate', event => {
  if (event.candidate) {
    signalingChannel.send({'new-ice-candidate': event.candidate});
  }
});

// Listen for remote ICE candidates and add them to the local RTCPeerConnection
signalingChannel.addEventListener('message', async message => {
  if (message.iceCandidate) {
    try {
      await peerConnection.addIceCandidate(message.iceCandidate);
    } catch (e) {
      console.error('Error adding received ice candidate', e);
    }
  }
});
```

```
// Listen for connectionstatechange on the local RTCPeerConnection
peerConnection.addEventListener('connectionstatechange', event => {
  if (peerConnection.connectionState === 'connected') {
    // Peers connected!
  }
});
```

REMOTE STREAMS

ONCE A RTCPeerConnection IS CONNECTED TO A REMOTE PEER, IT IS POSSIBLE TO STREAM AUDIO AND VIDEO BETWEEN THEM

A MEDIA STREAM CONSISTS OF AT LEAST 1 MEDIA TRACK, INDIVIDUALLY ADDED TO THE RTCPeerConnection.

```
const localStream = await getUserMedia({video: true, audio: true});
const peerConnection = new RTCPeerConnection(iceConfig);
localStream.getTracks().forEach(track => {
  peerConnection.addTrack(track, localStream);
});
```


TRACKS CAN BE ADDED TO A RTC PeerConnection BEFORE IT HAS CONNECTED TO A REMOTE PEER => **PERFORM THIS SETUP AS EARLY AS POSSIBLE!**

TO RECEIVE THE REMOTE TRACKS THAT WERE ADDED BY THE OTHER PEER, REGISTER A LISTENER ON RTCPeerConnection LISTENING FOR "track" EVENT.

RTCTrackEvent = [...] MediaStream objects

```
const remoteVideo = document.querySelector('#remoteVideo');

peerConnection.addEventListener('track', async (event) => {
  const [remoteStream] = event.streams;
  remoteVideo.srcObject = remoteStream;
});
```

HANDSHAKE

JS

CREA CONFIG

```
pc = new RTCPeerConnection(config)
```

SETTA CONSTRAINTS

```
pc.addTrack(track, stream);
```

```
pc.createOffer();
```

```
pc.setLocalDescription(offer);
```

→ ASPETTA FINO A CHE...

```
pc.iceGatheringState === "complete"
```

```
offer = pc.localDescription
```

Python

```
relay = MediaRelay()
```

WEBSOCKETS

offer.sdp offer.type

offer = RTCSessionDescription(sdp, type)

pc = RTCPeerConnection()

pc.on("track")

def on-track(track):

pc.addTrack(

USER
DEFINED

VideoStreamTrack(

relay.subscribe(track)

)

)

await pc.setRemoteDescription(offer)

answer = await pc.createAnswer()

pc.setRemoteDescription(answer)

answer.sdp answer.type

await pc.setLocalDescription(answer)

WEBSOCKETS