MŰEGYETEM 1782

Budapest University of Technology and Economics
Faculty of Electrical Engineering and Informatics
Department of Telecommunications and Media Informatics

WebRTC Laboratory Report

Student: Bat-Erdene Tushig **Neptun#:** QBI3JH

Field: Computer Engineering **Specialization:** Infocommunication
Github: https://github.com/tushig0826/MMS

Date: 2023 October 20

Task 1.1

The following code snippet demonstrates the fixed bugs in chat.js file:

```
callButton.onclick = call;
setBandwidthButton.onclick = setBandwidth;
pc.onicecandidate = onLocalICECandidateGenerated;
pc.oniceconnectionstatechange = onIceConnectionStateChange;
```

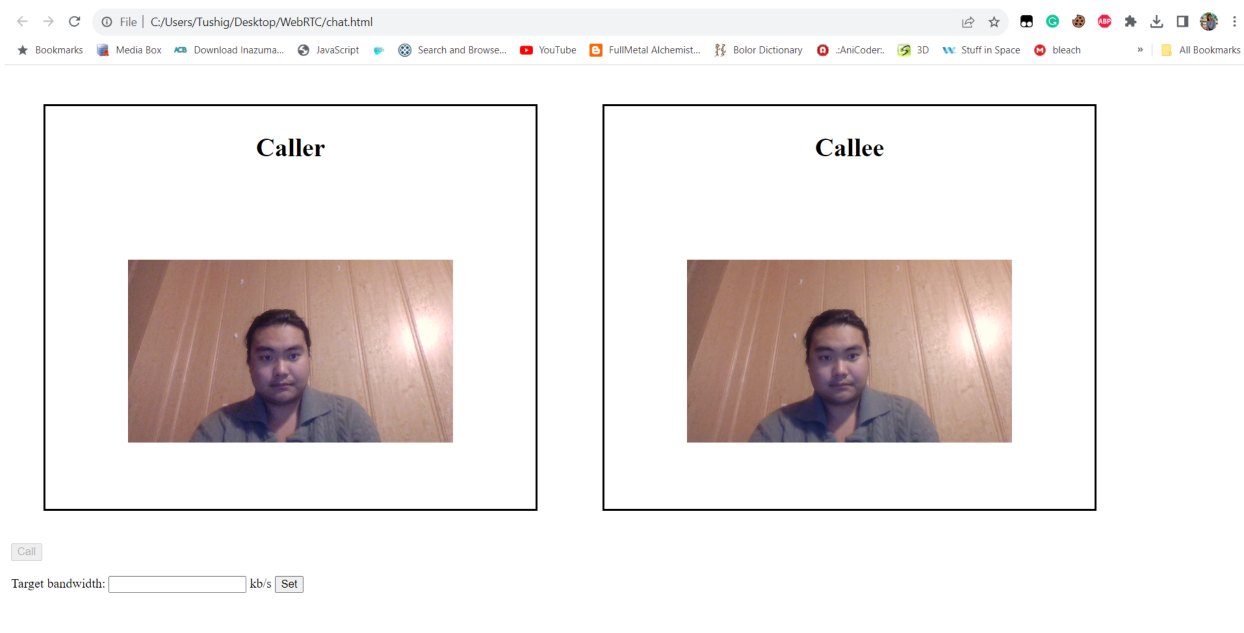

Figure 1. Result after fixing the bug

Task 1.2:

1. WebRTC connection it has to be defined:

```
var pc = new RTCPeerConnection();
```

2. To establish a connection, caller has to call which means that has to create an offer. In other words, when the user clicks on a call button, the call function will be called and it sets the calling variable to true. Then it creates an offer based on the pc.createOffer() function:

```
function call() {
  callButton.disabled = true;
  calling = true;

  pc.createOffer()
   .then(onCreateOfferSuccess)
   .catch(onError);
}
```

3. setBandwidth function will be called if the user clicks on a button and it will set up the parameters for making it the maximum bitrate of the video stream.
4. onSignallingMessage will work if a signaling message is received from a remote peer. If OnSignallingMessage is a messaging offer it has to set the description on the RTC session.
   If ice_candidate triggered, new ice candidate peer will be received from the remote caller. Msg object also contains media description, stream identification and candidate attributes.

```
function onSignallingMessage(msg) {
 switch(msg.type) {
  case 'offer':
   callButton.disabled = true;

   pc.setRemoteDescription(new RTCSessionDescription(JSON.parse(msg.data)))
    .then(onSetRemoteDescriptionSuccess)
    .catch(onError);
   break;

  case 'ice_candidate':
   var candidate = new RTCIceCandidate({
    sdpMLineIndex: msg.sdpMLineIndex,
    sdpMid: msg.sdpMid,
    candidate: msg.candidate});

   pc.addIceCandidate(candidate)
    .then(onAddIceCandidateSuccess)
    .catch(onError);
   break;
 }
}
```

setRemoteDescription() will set up the RTC session description.

5. gotLocalStream is called when local stream is obtained, and gotRemoteStream when remote stream is obtained.

```
function gotLocalStream(stream) {
 localVideo.srcObject = stream;
 stream.getTracks().forEach(track => pc.addTrack(track, stream));
 callButton.disabled = false;
}

function gotRemoteStream(event) {
 remoteVideo.srcObject = event.streams[0];

 pc.getSenders().forEach(sender => {console.log(sender);})
```

```
  setBandwidthButton.disabled = false;
}
```

6. onSetLocalDescriptionSuccess doesn't to anything but it is called when local session description has been set. onSetRemoteDescription behaves the same but in remote and it creates an answer to the offer that has been called by remote caller using pc.createAnswer().

```
function onSetLocalDescriptionSuccess() {}

function onSetRemoteDescriptionSuccess() {
  if(!calling) {
    pc.createAnswer()
      .then(onCreateAnswerSuccess)
      .catch(onError);
  }
}
```

7. It is called when creating an offer is successful. Offer object sets local session description.

```
function onCreateOfferSuccess(offer) {
  pc.setLocalDescription(offer)
    .then(onSetLocalDescriptionSuccess)
    .catch(onError);

  signalling.send({
    type: 'offer',
    data: JSON.stringify(offer)
  });
}
```

8. Answer is called when answer to the offer is created successfully. Answer object also sets local description and using signalling,send() to send object to the caller.

```
function onCreateAnswerSuccess(answer) {
  pc.setLocalDescription(answer)
    .then(onSetLocalDescriptionSuccess)
    .catch(onError);

  signalling.send({
    type: 'offer',
    data: JSON.stringify(answer)
  });
}
```

9. New ICE candidate is generated. It sends a candidate to a remote caller or peer.

```
function onLocalICECandidateGenerated(event) {
  if (event.candidate) {
    console.log('ICE candidate generated: ', event.candidate);
    signalling.send({
      type: 'ice_candidate',
      sdpMLineIndex: event.candidate.sdpMLineIndex,
      sdpMid: event.candidate.sdpMid,
      candidate: event.candidate.candidate
    });
  }
}
```

10. It is triggered when a new candidate has been added.

```
function onAddIceCandidateSuccess() {
  console.log('Ice candidate added successfully');
}
```

11. When there is a change in the current state of connection of the candidate, this function will be called.

```
function onIceConnectionStateChange(event) {
  console.log(pc.iceConnectionState);
}
```

Task 1.3:

Session Description Protocol is for describing multimedia sessions such as session announcement, session invitation, and so on.

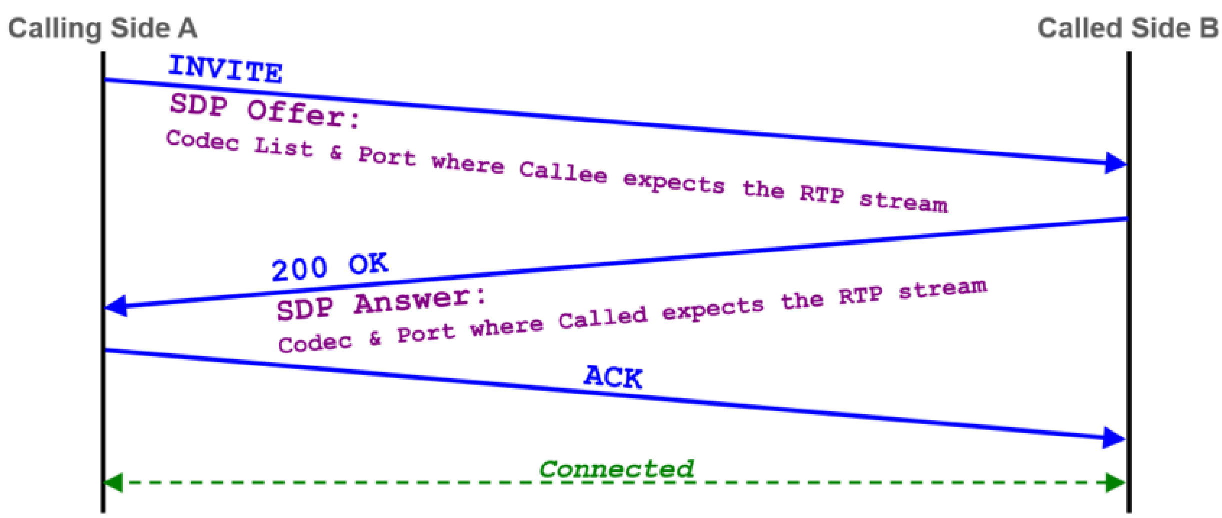

Figure 2: Overview of SDP with the minimal needed messages for a connection setup source: [http://help.aarenet.com/wiki/Support_voip_protocol]

As you can see from the figure 2, the SDP answer/offer model is used by two devices or users to reach acknowledgement on the description of the session.

The offering side indicates the desired session description in the offer with codec list and port.

The answering side replies to the offer with the desired session description from its own viewpoint. If the offering side receives the answer with expected codec and port or information then acknowledgement will be sent and connection will be established.

Task 2.1:

```
function onLocalICECandidateGenerated(event) {
  if (event.candidate) {
    console.log('ICE candidate generated: ', event.candidate);
    signalling.send({
      type: 'ice_candidate',
      sdpMLineIndex: event.candidate.sdpMLineIndex,
      sdpMid: event.candidate.sdpMid,
      candidate: event.candidate.candidate
    });
  }
}

function onAddIceCandidateSuccess() {
  console.log('Ice candidate added successfully');
}

function onIceConnectionStateChange(event)f {
  console.log(pc.iceConnectionState);
}
```

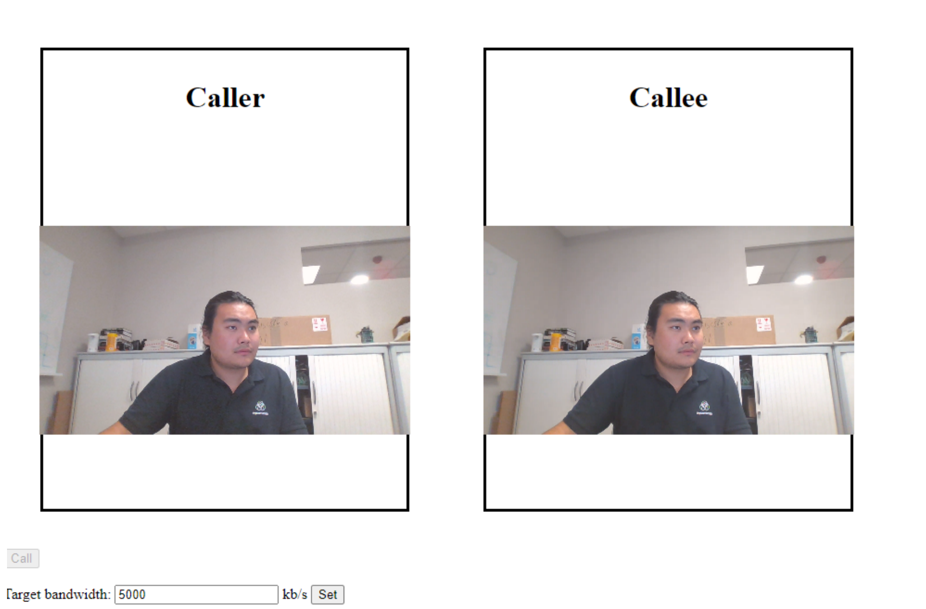

Figure 3. Setted bandwidth 5000

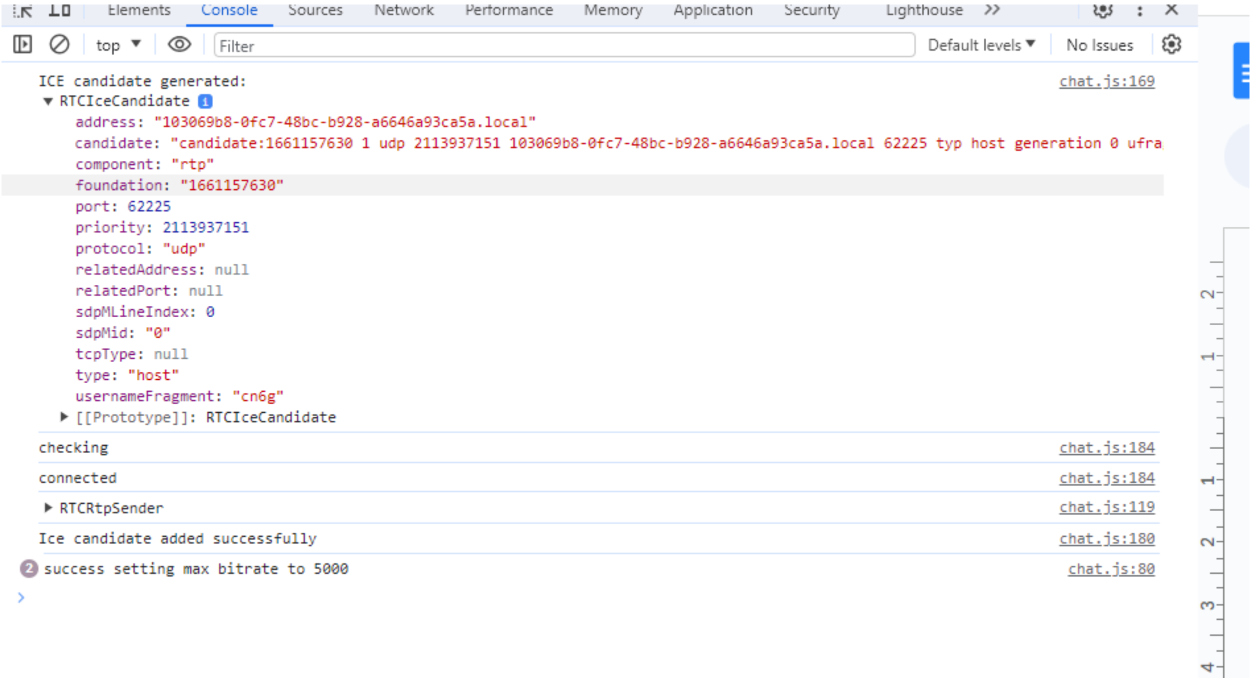

Figure 4. Console output

In the above figure, It demonstrates if ICE candidate generated (***console.log('ICE candidate generated: ', event.candidate);***) and adding ICE candidate was successful (***console.log('Ice candidate added successfully');***).

Task 2.2:

Both resolutions written below were supported.

```
navigator.mediaDevices.getUserMedia({
  audio: false,
  video: {
    width: { min: 640 },
    height: { min: 480 }
  }
}).then(gotLocalStream)
  .catch(onError);
```

```
navigator.mediaDevices.getUserMedia({
  audio: false,
  video: {
    width: { min: 800 },
    height: { min: 600 }
```

```
 }
}).then(gotLocalStream)
 .catch(onError);
```

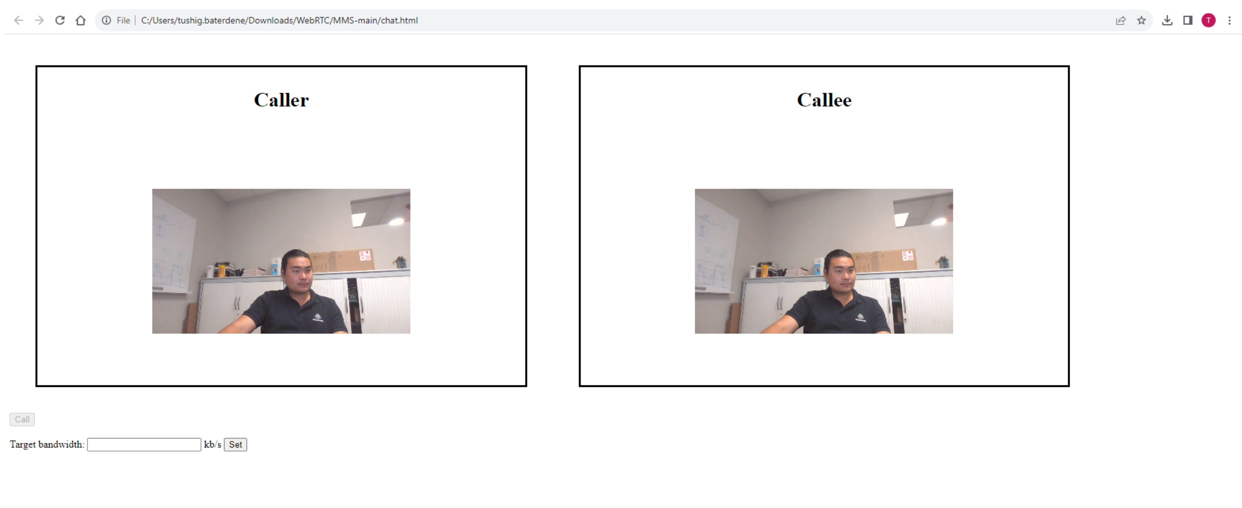

Figure 5: Result

We can also modify HTML files highlighted sections to change height and width:

video {

position: relative;

left: 50%;

transform: translateX(-50%);

width: 400px;

height: 400px; **}**

Task 2.3:

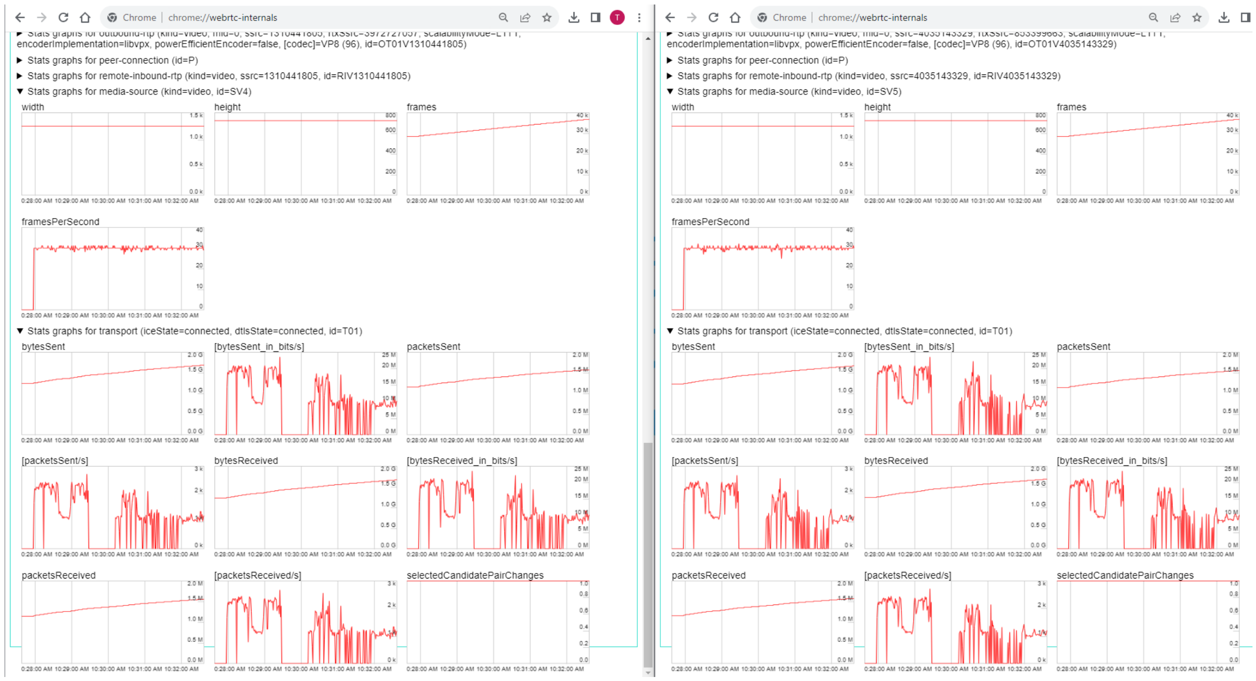

Figure 6. Webrtc-internals tools to check degradation

Left side is our first caller where the bandwidth set to 50000 kb/s (file:///C:/Users/tushig.baterdene/Downloads/WebRTC/MMS-main/chat.html [rid: 106, lid: 4, pid: 14284]) and right side is second caller where the bandwidth set to 100000 kb/s(file:///C:/Users/tushig.baterdene/Downloads/WebRTC/MMS-main/chat.html [rid: 103, lid: 5, pid: 17396])

In the following figure 7, the values of the callers where bandwidth has been set to 50000 kb/s is 12.089, and 100000 kb/s is 20.15 in quality limitation duration.

qualityLimitationDurations	{bandwidth:12.089,cpu:434.744,none:1398.9,other:0}
qualityLimitationResolutionChanges	0
encoderImplementation	libvpx
firCount	0
pliCount	1
nackCount	39
qpSum	237686
[qpSum/framesEncoded]	2
active	true
powerEfficientEncoder	false
scalabilityMode	L1T1
rtxSsrc	3972727057

qualityLimitationDurations	{bandwidth:20.15,cpu:434.898,none:1390.72,other:0}
qualityLimitationResolutionChanges	0
encoderImplementation	libvpx
firCount	0
pliCount	1
nackCount	18
qpSum	240830
[qpSum/framesEncoded]	2
active	true
powerEfficientEncoder	false
scalabilityMode	L1T1
rtxSsrc	853399663

Figure 7. Quality Limitation duration with bandwidth

Task 2.4:

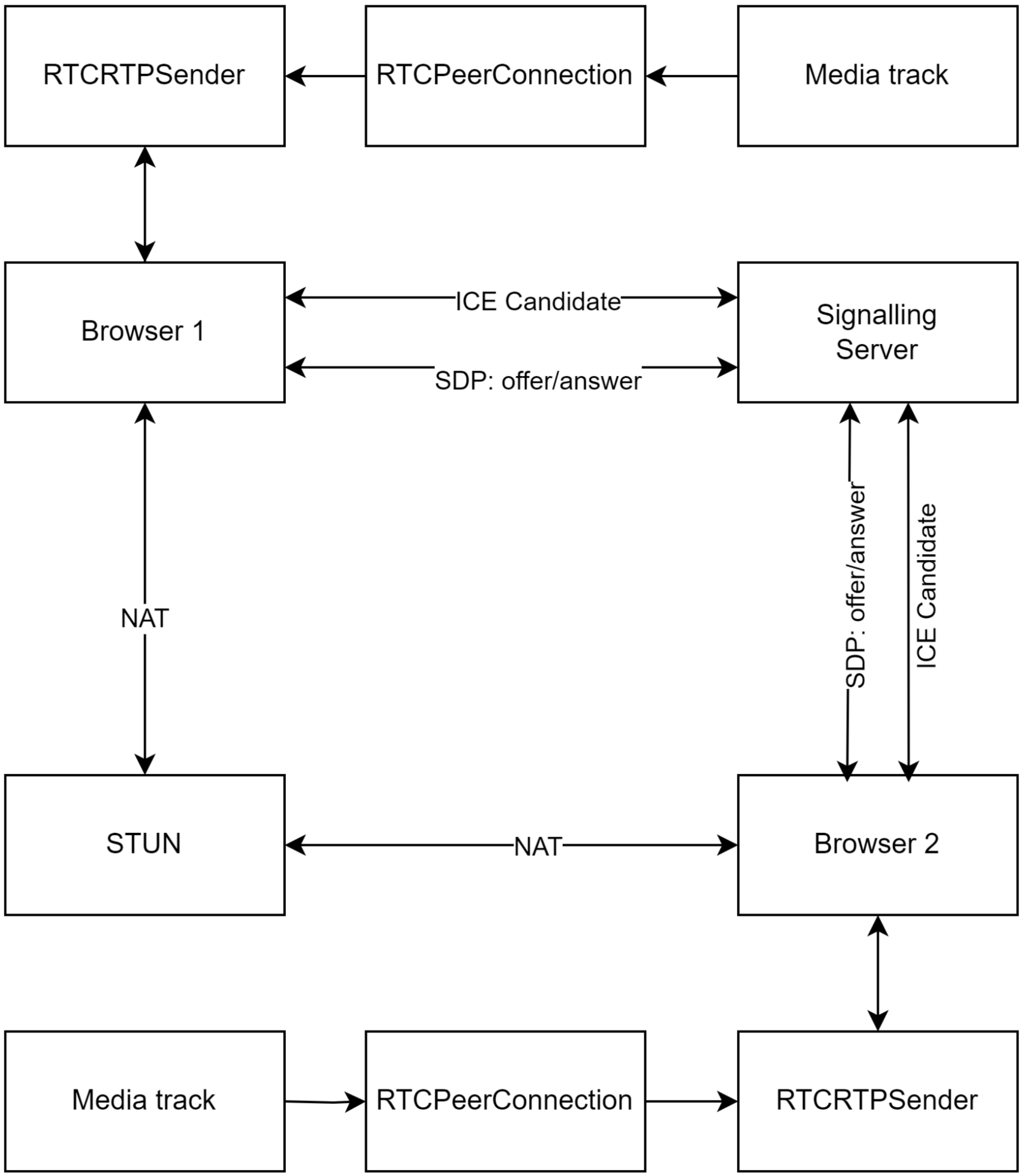

Figure 8. Architecture diagram