```
var sdpConstraints = {'mandatory': {'OfferToReceiveAudio':true, 'OfferToReceiveVideo':true }};

var channelReady;
var channel;

var pc = null;

var pc_config =
  { iceServers: [
    {urls:'stun:stun.l.google.com:19302'},
    {
      urls: "turn:webrtc.yangchaho.com",
      username: "yangcha",
      credential: "SprPswrd!"
    }
   ]
  };

var config = {
  wssHost: 'wss://webrtc.yangchaho.com:16443'
};

channel = new WebSocket(config.wssHost);
channel.onmessage = onChannelMessage;
channel.onclose = onChannelClosed;

function onChannelMessage(message) {
       processSignalingMessage(message.data);
   };

function onChannelClosed() {
       channelReady = false;
   };

function sendMessage(message) {
       var msgString = JSON.stringify(message);
       channel.send(msgString);
   };

function processSignalingMessage(message) {
    console.log("received message: " + message);
    var msg = JSON.parse(message);
    if (msg.type === 'offer') {
          pc.setRemoteDescription(new RTCSessionDescription(msg));
          doAnswer();
          console.log("send answer");
        } else if (msg.type === 'answer') {
            pc.setRemoteDescription(new RTCSessionDescription(msg));
            console.log("received answer");
        } else if (msg.type === 'candidate') {
```

```
            var candidate = new RTCIceCandidate({sdpMLineIndex:msg.label,
candidate:msg.candidate});
            pc.addIceCandidate(candidate);
            console.log("added candidate");
        } else if (msg.type === 'GETROOM') {
            room = msg.value;
        } else if (msg.type === 'WRONGROOM') {
            window.location.href = "/";
        }
    };

const startButton = document.getElementById('startButton');
const callButton = document.getElementById('callButton');
const endButton = document.getElementById('endButton');
callButton.disabled = true;
endButton.disabled = true;
startButton.addEventListener('click', start);
callButton.addEventListener('click', call);
endButton.addEventListener('click', hangup);

let startTime;
const localVideo = document.getElementById('localVideo');
const remoteVideo = document.getElementById('remoteVideo');

localVideo.addEventListener('loadedmetadata', function() {
  console.log(`Local video videoWidth: ${this.videoWidth}px,  videoHeight:
${this.videoHeight}px`);
});

remoteVideo.addEventListener('loadedmetadata', function() {
  console.log(`Remote video videoWidth: ${this.videoWidth}px,  videoHeight:
${this.videoHeight}px`);
});

remoteVideo.addEventListener('resize', () => {
  console.log(`Remote video size changed to
${remoteVideo.videoWidth}x${remoteVideo.videoHeight}`);
  if (startTime) {
    const elapsedTime = window.performance.now() - startTime;
    console.log('Setup time: ' + elapsedTime.toFixed(3) + 'ms');
    startTime = null;
  }
});

let localStream;

async function start() {
  console.log('Requesting local stream');
  startButton.disabled = true;
  try {
    const stream = await navigator.mediaDevices.getUserMedia({audio: true, video: true});
```

```
    console.log('Received local stream');
    localVideo.srcObject = stream;
    localStream = stream;
    callButton.disabled = false;

    var pc_constraints = {"optional": [{"DtlsSrtpKeyAgreement": true}]};
    pc = new RTCPeerConnection(pc_config, pc_constraints);
    console.log('Created local peer connection object pc');
    pc.addEventListener('icecandidate', e => onIceCandidate(e));

    pc.addEventListener('track', gotRemoteStream);

    stream.getTracks().forEach(track => pc.addTrack(track, stream));
    console.log('Added local stream to pc');
    }
  catch (e) {
    alert(`getUserMedia() error: ${e}`);
  }
}

async function call() {
  callButton.disabled = true;
  endButton.disabled = false;
  console.log('Starting call');
  startTime = window.performance.now();
  const videoTracks = localStream.getVideoTracks();
  const audioTracks = localStream.getAudioTracks();
  if (videoTracks.length > 0) {
    console.log(`Using video device: ${videoTracks[0].label}`);
  }
  if (audioTracks.length > 0) {
    console.log(`Using audio device: ${audioTracks[0].label}`);
  }

  try {
    console.log('pc createOffer start');
    var constraints = {"optional": [], "mandatory": {"MozDontOfferDataChannel": true}};
    constraints = mergeConstraints(constraints, sdpConstraints);
    const offer = await pc.createOffer(constraints);
    await setLocalAndSendMessage(offer);
  } catch (e) {
    onCreateSessionDescriptionError(e);
  }
}

function setLocalAndSendMessage(sessionDescription) {
        pc.setLocalDescription(sessionDescription);
        sendMessage(sessionDescription);
    };

function onCreateSessionDescriptionError(error) {
```

```
  console.log(`Failed to create session description: ${error.toString()}`);
}

async function doAnswer() {
  try {
    const answer = await pc.createAnswer(sdpConstraints);
    setLocalAndSendMessage(answer);
  } catch (e) {
    onCreateSessionDescriptionError(e);
  }
};

function onSetLocalSuccess(pc) {
  console.log(`${getName(pc)} setLocalDescription complete`);
}

function gotRemoteStream(e) {
  if (remoteVideo.srcObject !== e.streams[0]) {
    remoteVideo.srcObject = e.streams[0];
    remoteVideo.play(0);
    console.log('channel received remote stream');
  }
}

async function onIceCandidate(event) {
if (event.candidate)
     sendMessage({type: 'candidate', label: event.candidate.sdpMLineIndex, id:
event.candidate.sdpMid,
        candidate: event.candidate.candidate});
}

function hangup() {
  console.log('Ending call');
  pc.close();
  pc = null;
  channel.close();
  channel = null;
  endButton.disabled = true;
  callButton.disabled = false;
}

function mergeConstraints(cons1, cons2) {
    var merged = cons1;
       for (var name in cons2.mandatory)
           merged.mandatory[name] = cons2.mandatory[name];
           merged.optional.concat(cons2.optional);
           return merged;
    };
```