

Apéndice D. Código fuente del prototipo de aplicación de pruebas de AoIP mediante

WebRTC

En esta sección, se muestran el código fuente perteneciente a la página web de la aplicación en lenguaje HTML, CSS, y Javascript, y el código de la aplicación que se ejecuta de parte del servidor web / de señalización en Javascript y Node.JS.

1. Código Fuente en lenguaje HTML del prototipo de aplicación (index.html).

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>AoIP con WebRTC</title>
    <link rel="icon" type="image/x-icon" href="/source/icon/favicon.ico">
    <link rel="stylesheet" href="css/main.css"/>
    <link rel="stylesheet"
href="https://maxcdn.bootstrapcdn.com/bootstrap/4.0.0/css/bootstrap.min.css" integrity "sha384-
Gn5384xqQ1aoWXA+058RXPxPg6fy4IWvTNh0E263XmFcJlSAwiGgFAW/dAiS6JXm" crossorigin="anonymous">
  </head>
  <body>
    <header>
      <h3>Estudio e Implementación de aplicaciones soportadas en WebRTC que permitan mejorar la
Calidad de Experiencia (QoE) del Audio sobre IP (AoIP)</h3>
    </header>
    <div class="container-fluid">
      <div id="create_room_div" class="create_room">
        <fieldset class="fieldset_outline">
          <legend class="legend_outline">Crear una sala</legend>
          <div class="form-row align-items-end justify-content-center">
            <div class="col-auto">
              <label for="room_name" class="col-form-label">Nombre sala</label>
              <input type="text" id="room_name" class="form-control" required>
            </div>
            <div class="col-auto">
              <label for="codec_selector" class="col-form-label">Codec de preferencia a
usar</label>
              <select id="codec_selector" class="form-control">
                <option value="opus/48000" selected = "selected">OPUS variable bit
rate (6kbps-510kbps)</option>
                <option value="ISAC/16000">ISAC 16kbps bit rate</option>
                <option value="ISAC/32000">ISAC 32kbps bit rate</option>
                <option value="G722/8000">G.722</option>
                <option value="iLBC/8000" disabled>iLBC</option>
              </select>
            </div>
            <div class="col-auto">
              <button id="create_room_btn" class="btn btn-primary"
onClick="createRoom(document.getElementById('room_name').value)">Crear Sala</button>
            </div>
          </div>
        </fieldset>
      </div>
      <div id="room_list_div" class="room_list">
        <fieldset class="fieldset_outline">
          <legend class="legend_outline">Unirse a una sala</legend>
          <div id="room_list_btns_div" class="d-flex justify-content-center"></div>
        </fieldset>
      </div>
      <div id="audio_display_div" class="audio_display">
        <p id="room_name_label"></p>
        <div class="d-flex justify-content-center">
          <div class="audio_controls">
            Audio Local
            <br>
            <audio controls id="local_audio" autoplay muted></audio>
          </div>
          <div class="audio_controls">
            Audio Remoto
          </div>
        </div>
      </div>
    </div>
  </body>
</html>
```

Comentado [TRVH1]: Adjuntar un tar con todos los scripts de la APP.

Vamos a realizar registro de software ante la dirección general de derechos de autor?

Comentado [WU2R1]: Código fuente adjuntado en formato .zip

Lo más sencillo que se pueda hacer de registro del software, de preferencia software libre, no propietario.

Comentado [TRVH3R1]: Un producto interesante es el registro del software ante la dirección general de derechos de autor

Comentado [TRVH4]: Este apéndice ampliarlo un poco mas, Arquitectura y código

Comentado [WU5R4]: ¿A qué se refiere con arquitectura? La arquitectura está explicada en el libro, El código está completo.

Comentado [TRVH6R4]: La arquitectura extremo a extremo, el software y señalización y protocolos que se utilizan en cada extremo de la comunicación.

Se requiere una nube, lo que pasa en ella, que tipo de nube. Dos end-systems, (P2P o cliente servidor), servidores externos, mensajes entre los dispositivos intercambiados, tanto de datos como de control.

Comentamos este aspecto

Comentado [TRVH7]: En la sección de códigos utilizar un tamaño de letra menor!!! Para visualizar mejor las cosas!!

Comentado [WU8R7]: Letra tamaño 7 para todos los códigos

```

        <br>
        <audio controls id="remote_audio"></audio>
    </div>
</div>
<br>
<div class="d-flex justify-content-center">
    <button id="exit_room_btn" class="btn btn-secondary">Salir de la sala</button>
    <button id="cambiar_codec_prf_btn" type="button" class="btn btn-info" data-
toggle="modal" data-target="#cambiarCodecModal">Cambiar codec preferido</button>
    <button id="show_net_info_btn" class="btn btn-primary">Mostrar estadísticas de
red</button>
</div>
</div>
<br>
<!-- Modal -->
<div class="modal fade" id="cambiarCodecModal" tabindex="-1" role="dialog" aria-
labelledby="cambiarCodecModalLabel" aria-hidden="true">
    <div class="modal-dialog" role="document">
        <div class="modal-content">
            <div class="modal-header">
                <h5 class="modal-title" id="cambiarCodecModalLabel">Cambiar codec
preferido</h5>
                <button type="button" class="close" data-dismiss="modal" aria-
label="Close">
                    <span aria-hidden="true">&times;</span>
                </button>
            </div>
            <div class="modal-body">
                Codec de preferencia a usar:
                <br>
                <em>(Sólo se puede cambiar mientras la comunicación con otro par no esté
activa.)</em>
                <select id="codec_selector_2" class="form-control">
                    <option value="opus/48000" selected = "selected">OPUS variable bit
rate (6kbps-510kbps)</option>
                    <option value="ISAC/16000">ISAC 16kbps bit rate</option>
                    <option value="ISAC/32000">ISAC 32kbps bit rate</option>
                    <option value="G722/8000">G.722</option>
                    <option value="iLBC/8000" disabled>iLBC</option>
                </select>
            </div>
            <div class="modal-footer">
                <button type="button" class="btn btn-secondary" data-
dismiss="modal">Cerrar</button>
                <button id="guardar_codec_sel_btn" type="button" class="btn btn-primary"
data-dismiss="modal">Guardar</button>
            </div>
        </div>
    </div>
</div>
<!-- Modal End -->
<div id="net_statistics_div" class="net_statistics">
    <div class="d-flex flex-row justify-content-center">
        <label for="peer_info" class="p-2">Conectado a:</label>
        <input id="peer_info" class="form-control p-2" type="text" placeholder=""
readonly>
    </div>
    <div class="d-flex flex-row justify-content-center">
        <label for="active_codec" class="p-2">Codec activo:</label>
        <input id="active_codec" class="form-control p-2" type="text" placeholder=""
readonly>
    </div>
    <div id="peer_connection_stats_div"></div>
</div>
</div>
<a
href="https://docs.google.com/forms/d/e/1FAIpQLSetWlAx04f8eJkH527reHelVIZkfwNvxilqRJLOJaPESuIrQ/viewform
?usp=sf_link">Evaluación calidad de la experiencia</a>
<footer>2018</footer>
<script src="/socket.io/socket.io.js"></script>
<script src="https://webrtc.github.io/adapter/adapter-latest.js"></script>
<script src="js/main.js"></script>
<script src="https://code.jquery.com/jquery-3.2.1.slim.min.js" integrity="sha384-
KJ3o2DKtIkVYIK3UENzmM7KCKKr/rE9/Qpg6aAZGJwFDMVNA/GpGFF93hXpG5KkN" crossorigin="anonymous"></script>
<script src="https://cdnjs.cloudflare.com/ajax/libs/popper.js/1.12.9/umd/popper.min.js"
integrity="sha384-ApNbgh9B+Y1QKtv3Rn7W3mgPxhU9K/ScQsAP7hUibX39j7fakFFkvXusvfa0b4Q"
crossorigin="anonymous"></script>
<script src="https://maxcdn.bootstrapcdn.com/bootstrap/4.0.0/js/bootstrap.min.js"
integrity="sha384-JZR6Spejh4U02d8jOt6vLEHfe/JQGiRRSQQxSfFWpi1MquVdAyjUar5+76PVCmY1"
crossorigin="anonymous"></script>
</body>
</html>
```

2. Código fuente en lenguaje CSS del prototipo de aplicación (/css/main.css).

```
body {
    font-family: sans-serif;
}

.btn {
    margin: 0.5em;
}

.fieldset_outline {
    border: 1px solid gray;
    border-radius: 10px;
    padding: 0.5em;
}

.legend_outline {
    width: inherit;
    margin: 0px;
}

.audio_display {
    display: none;
}

.audio_controls {
    margin: 0.5em;
    border: 2px solid #a1a1a1;
    border-radius: 20px;
    padding: 0.8em;
}

.net_statistics{
    margin: auto;
    padding: 1em;
    width: 50%;
    justify-content: center;
    text-align: left;
    display: none;
}

#active_codec, #peer_info {
    width: auto;
}

#room_name_label {
    text-align: center;
    font-size: 120%;
    font-weight: bold;
}

header, footer {
    padding: 1em;
    color: Black;
    background-color: white;
    clear: left;
    text-align: center;
}
```

3. Código fuente en lenguaje Javascript del prototipo de aplicación (/js/main.js).

```
'use strict';

var SIGNALING_SERVER = '190.254.213.124';
var CHROME = (navigator.userAgent.toLowerCase().indexOf("chrome") != -1);
var isChannelReady = false;
var isInitiator = false;
var isStarted = false;
var local_stream;
var remote_stream;
var pc;
var room;
var codec_selected;
var codec_last_used;
var net_stats_interval_id;
var toggle_net_statistics = false;
var media_recorder;
var socket = io.connect(SIGNALING_SERVER);
// Restricciones del stream a obtener por medio de getUserMedia().
var constraints = {
```

```

        video: false,
        audio: true
    });
    // Configuración de los servidores ICE/TURN a usar en la conexión.
    var iceTurnServers = { iceServers: [ { urls: 'stun:stun.l.google.com:19302' },
        { urls: 'stun:stun1.l.google.com:19302' },
        { urls: 'stun:stun2.l.google.com:19302' },
        { urls: 'stun:stun3.l.google.com:19302' },
        { urls: 'stun:stun4.l.google.com:19302' }
    ]};
    var createRoomsDiv = document.getElementById('create_room_div');
    var roomName = document.getElementById('room_name');
    var codecSelector = document.getElementById('codec_selector');
    var createRoomBtn = document.getElementById('create_room_btn');
    var roomListDiv = document.getElementById('room_list_div');
    var roomListBtnsDiv = document.getElementById('room_list_btns_div');
    var audioDisplayDiv = document.getElementById('audio_display_div');
    var roomNameLabel = document.getElementById('room_name_label');
    var localAudio = document.getElementById('local_audio');
    var remoteAudio = document.getElementById('remote_audio');
    var exitRoomBtn = document.getElementById('exit_room_btn');
    exitRoomBtn.onclick = exitRoom;
    var cambiarCodecPrfBtn = document.getElementById('cambiar_codec_prf_btn');
    var codecSelector2 = document.getElementById('codec_selector_2');
    var guardarCodecSelBtn = document.getElementById('guardar_codec_sel_btn');
    guardarCodecSelBtn.onclick = saveSelCodec;
    var showNetInfoBtn = document.getElementById('show_net_info_btn');
    showNetInfoBtn.onclick = showNetInfo;
    showNetInfoBtn.disabled = true;
    var netStatisticsDiv = document.getElementById('net_statistics_div');
    var activeCodec = document.getElementById('active_codec');
    var peerInfo = document.getElementById('peer_info');
    var peerConnectionStatsDiv = document.querySelector('div#peer_connection_stats_div');

    requestRoomsList(); // pedir la lista de cuartos creados al iniciar.

    /***** Configuración Websockets *****/

    /**
     * Función para manejar la petición de la lista de salas activas en el servidor. Si el par está
     * buscando salas a unirse se crean botones con los que permitirle unirse a las salas recibidas.
     * @param rooms lista de salas activas en el servidor.
     */
    socket.on('room list', function(rooms) {
        console.log('room list arrived');
        if(!isChannelReady && !isInitiator && !isStarted) {
            createRoomsListBtns(rooms);
        }
    });

    /**
     * Función para manejar el informe de que el par es el creador de la sala.
     * @param room nombre de la sala creada.
     */
    socket.on('created', function(room) {
        console.log('Created room ' + room);
        isInitiator = true;
    });

    /**
     * Función para manejar el informe de que la sala a la que intenta unirse el par está llena.
     * Desactiva la interfaz gráfica de la transmisión de datos y vuelve a pedir la lista de salas.
     * @param room nombre de la sala.
     */
    socket.on('full', function(room) {
        console.log('Room ' + room + ' is full');
        alert('La sala especificada está llena, intente con otra o espere a que se desocupe.');
```

```

* Función para manejar el informe de que el par se unió a una sala ya existente.
* @param room nombre de la sala.
*/
socket.on('joined', function(room) {
    console.log('joined: ' + room);
    isChannelReady = true;
});

/**
* Función para manejar el informe de que el par se salió de la sala especificada.
* @param room nombre de la sala.
*/
socket.on('left', function(room){
    console.log('left: ' + room);
    isChannelReady = false;
    isInitiator = false;
});

/**
* Función para registrar los mensajes del servidor en el cliente.
* @param array Registro enviado por el servidor.
*/
socket.on('log', function(array) {
    console.log.apply(console, array);
});

/**
* Función para manejar la desconexión de la conexión con el servidor.
*/
socket.on('disconnect', function () {
    console.log('disconnected to server');
} );

/***** Funciones Websockets *****/

/**
* Función para enviar un mensaje al servidor web.
* @param message
*/
function sendMessage(message) {
    console.log('Client sending message: ', message);
    socket.emit('message', message, room);
}

/**
* Función para recibir mensajes. Los mensajes pueden ser de 5 clases.
* 1.got user media: Marca la obtención del stream de datos del emisor del mensaje.
* 2.offer: Inicia la negociación de la comunicación entre pares al hacer la oferta.
* 3.answer: Establecen la comunicación con entre pares. Obtiene la información de
*           contacto del mensaje y la configura en la descripción remota.
* 4.candidate: Establece un candidato remoto ICE a partir del mensaje.
* 5.bye: Marca el final de la comunicación del par remoto.
* @param message
*/
socket.on('message', function(message) {
    console.log('Client received message:', message);
    if (message === 'got user media') {
        maybeStart();
    }
    else if (message.type === 'offer') {
        if (!isInitiator && !isStarted) {
            maybeStart();
        }
        pc.setRemoteDescription(message);
        doAnswer();
    }
    else if (message.type === 'answer' && isStarted) {
        pc.setRemoteDescription(message);
    }
    else if (message.type === 'candidate' && isStarted) {
        var candidate = new RTCIceCandidate({
            sdpMLineIndex: message.label,
            candidate: message.candidate
        });
        pc.addIceCandidate(candidate);
    }
    else if (message === 'bye' && isStarted) {
        handleRemoteHangup();
    }
});

/**

```

```

* Función para pedir la lista de salas creadas en el servidor web.
*/
function requestRoomsList() {
    if(!isChannelReady && !isInitiator && !isStarted) {
        console.log('requesting room list');
        socket.emit('request room list');
    }
}

/***** Funciones *****/

/**
 * Función para detener intervalos creados con setInterval.
 * @param IntervalID Identificador del intervalo a detener.
 */
function stopInterval(IntervalID) {
    try {
        clearInterval(IntervalID);
    }
    catch(e) {
        console.log(e);
    }
}

/**
 * Función para crear botones que sirven para unirse las salas especificadas en la lista de salas.
 * @param rooms Lista de salas.
 */
function createRoomsListBtns(rooms) {
    var btns2create = '';
    rooms.forEach(function(room){
        btns2create += '<button class="btn btn-outline-primary'
onClick="createRoom(\''+room+'\')">'+room+'</button>';
    });
    room_list_btns_div.innerHTML = btns2create;
}

/**
 * Función para habilitar/deshabilitar mostrar la interfaz gráfica de creación de salas,
 * listado de salas, información transmisión de audio, y estadísticas de red.
 * @param show Valor booleano que representa la interfaz grafica disponible para creacion de
 * salas(false)/muestra de información durante la transmisión de audio(true).
 */
function toggleShowDivs(show) {
    if(show){
        create_room_btn.disabled = true;
        exit_room_btn.disabled = false;
        create_rooms_div.style.display = 'none';
        room_list_div.style.display = 'none';
        audio_display_div.style.display = 'block';
    }
    else{
        toggle_net_statistics = false;
        create_room_btn.disabled = false;
        exit_room_btn.disabled = true;
        create_rooms_div.style.display = 'block';
        room_list_div.style.display = 'block';
        audio_display_div.style.display = 'none';
        net_statistics_div.style.display = 'none';
    }
}

/**
 * Función para la creación de salas en el servidor.
 * @param room_n Nombre de la sala.
 */
function createRoom(room_n) {
    room = room_n;
    if (room !== '' || typeof room !== 'undefined') {
        socket.emit('create or join', room);
        console.log('Attempted to create or join room', room);
        room_name_label.innerHTML = 'Sala: ' + room;
        codec_selected = codec_selector.options[codec_selector.selectedIndex].value; // Obtener
el nombre del codec a usar.
        toggleShowDivs(true);

        // Obtiene el stream de datos local.
        navigator.mediaDevices.getUserMedia(constraints)
        .then(gotStream)
        .catch(function(e) {
            alert('getUserMedia() error: ' + e.name);
        });
    }
}

```

```

    }

    /**
    * Función para salir de la sala actual a la que se encuentra unido en el servidor.
    */
    function exitRoom() {
        if(pc) {
            sendMessage('bye');
        }
        stop();
        socket.emit('leave', room);
        console.log('Attempted to leave room', room);
        room = '';
        room_name.value = '';
        toggleShowDivs(false);
        //requestRoomsList();
        stopInterval(net_stats_interval_id);
        var track = local_stream.getTracks()[0];
        track.stop(); // detiene el stream local de audio.
    }

    /**
    * Función para guardar el codec seleccionado en el dialogo modal.
    */
    function saveSelCodec() {
        codec_selected = codec_selector_2.options[codec_selector_2.selectedIndex].value;
    }

    /**
    * Función para mostrar la información de red proveida por la interfaz RTCStatsReport,
    * la cual ofrece datos estadísticos sobre las conexiones RTCPeerConnection.
    * (RTCPeerConnection.getStats())
    * Además muestra el codec activo en RTCPeerConnection, y la dirección ip y puerto del par al que está
    * conectado.
    */
    function showNetInfo() {
        if(toggle_net_statistics) {
            toggle_net_statistics = false;
            clearInterval(net_stats_interval_id);
            net_statistics_div.style.display = 'none';
        }
        else {
            toggle_net_statistics = true;
            net_statistics_div.style.display = 'block';

            if(pc) {
                console.log('local sdp: ', pc.localDescription.sdp);
                console.log('remote sdp: ', pc.remoteDescription.sdp);

                // Visualización de datos estadísticos cada 1000 mseg.
                net_stats_interval_id = setInterval(function() {
                    if(isInitiator) {
                        var a_codec = getCodec(pc.remoteDescription.sdp); // Obtiene el
                        codec del sdp de RTCPeerConnection.
                    }
                    else {
                        var a_codec = getCodec(pc.localDescription.sdp); // Obtiene el
                        codec del sdp de RTCPeerConnection.
                    }
                    active_codec.value = a_codec;
                    pc.getStats(null)
                    .then(function(results) {
                        var statsString = dumpStats(results);
                        peer_connection_stats_div.innerHTML = '<h2>Peer connection
                        stats</h2>' + statsString;

                        // figure out the peer's ip
                        var activeCandidatePair = null;
                        var remoteCandidate = null;

                        // Search for the candidate pair, spec-way first.
                        results.forEach(function(report) {
                            if (report.type === 'transport') {
                                activeCandidatePair =
                                results.get(report.selectedCandidatePairId);
                            }
                        });
                        // Fallback for Firefox and Chrome legacy stats.
                        if (!activeCandidatePair) {

```

```

        results.forEach(function(report) {
            if (report.type === 'candidate-pair' &&
                report.type === 'googCandidatePair' &&
                report.googActiveConnection === 'true') {
                activeCandidatePair = report;
            }
        });
    }
    if (activeCandidatePair &&
        activeCandidatePair.remoteCandidateId) {
        results.get(activeCandidatePair.remoteCandidateId);
        remoteCandidate =
            results.get(activeCandidatePair.remoteCandidateId);
        if (remoteCandidate) {
            if (remoteCandidate.ip && remoteCandidate.port) {
                peer_info.value = remoteCandidate.ip + ':' +
                    remoteCandidate.port;
            } else if (remoteCandidate.ipAddress &&
                remoteCandidate.portNumber) {
                // Fall back to old names.
                peer_info.value = remoteCandidate.ipAddress +
                    ':' + remoteCandidate.portNumber;
            }
        }, function(err) {
            console.log(err);
        });
    }, 1000);
}
else {
    console.log('Not connected yet');
}
}

/**
 * Evento que ocurre cuando el documento está por ser cerrado. Se usa para enviar un mensaje de
 * despedida al servidor y cerrar el websocket.
 */
window.onbeforeunload = function() {
    hangup();
    return null;
};

/**
 * Función para obtener el stream de datos local y reproducirlo en la página.
 * @param stream stream local.
 */
function gotStream(stream) {
    console.log('Getting user media with constraints', constraints);
    console.log('Adding local stream. ');
    local_audio.srcObject = stream; // Mostrar el audio local en la página.
    local_stream = stream;
    sendMessage('got user media');
    if (isInitiator) {
        maybeStart();
    }
}

/**
 * Función que al comprobar los requisitos previos inicia el proceso de negociación para establecer la
 * transmisión de audio al crear la conexión entre pares y añadir el stream local.
 */
function maybeStart() {
    console.log('>>>>>> maybeStart() ', isStarted, local_stream, isChannelReady);
    if (!isStarted && typeof local_stream !== 'undefined' && isChannelReady) {
        console.log('>>>>> creating peer connection');
        createPeerConnection(ice_turn_servers);
        pc.addStream(local_stream);
        isStarted = true;
        cambiar_codec_prf_btn.disabled = true; // Deshabilitar la interfaz de selección de
        show_net_info_btn.disabled = false; // Habilitar el boton de muestra de información de
        console.log('isInitiator', isInitiator);
        if (isInitiator) {
            doCall();
        }
    }
}

```



```

/**
 * Función para crear objeto de la clase RTCPeerConnection e inicializarlo con los servidores de STUN/TURN
 * a usar en
 * la conexión, además de establecer los manejadores de eventos de la conexión.
 * @param ICE_Config JSON que contiene la información de la dirección y autenticación de los servidores
 * ICE/TURN a usar en la conexión.
 */
function createPeerConnection(ICE_Config) {
    try {
        pc = new RTCPeerConnection(ICE_Config);
        pc.onicecandidate = handleIceCandidate;
        pc.ontrack = handleRemoteStreamAdded;
        pc.onremovestream = handleRemoteStreamRemoved;
        console.log('Created RTCPeerConnection');
    } catch (e) {
        console.log('Failed to create PeerConnection, exception: ' + e.message);
        alert('Cannot create RTCPeerConnection object.');
```

```

* @param event De tipo RTCTrackEvent el cual es enviado cuando un MediaStreamTrack entrante ha sido
creado y asociado con un objeto RTCRtpReceiver el cual ha sido añadido al conjunto de * receptores en la
conexión.
*/
function handleRemoteStreamAdded(event) {
    console.log('Remote stream added. ');
    remote_stream = event.streams[0];
    remote_audio.srcObject = remote_stream;
    remote_audio.setAttribute('autoplay','1'); // Autoreproduce el stream de audio remoto.

    /***** Grabador de Audio 1 *****/

    /**
     * Funciones para grabar el stream de audio remoto mediante la API de MediaRecorder (Sólo Google
Chrome).
     * El audio grabado se muestra en el elemento html de audio remoto permitiendo descargarlo.
     * (Si se conecta otro par a la sala antes de descargar el archivo de la transmisión de audio
previa
     * el link de la grabación se pierde y se empieza una nueva grabación.)
     */
    if(CHROME)
    {
        media_recorder = new MediaRecorder(remote_stream);
        media_recorder.start();
        console.log("recording of remote stream started");
        var chunks = [];
        media_recorder.ondataavailable = function(e) {
            chunks.push(e.data);
        }
        media_recorder.onstop = function(e) {
            console.log("recording of remote stream stopped");
            var mtype;
            if (codec_last_used !== null)
            {
                switch(codec_last_used){
                    case "opus/48000":
                        mtype = { 'type' : 'audio/opus; codecs=opus' };
                        break;
                    case "ISAC/16000":
                        mtype = { 'type' : 'audio/isac; codecs=isac' };
                        break;
                    case "ISAC/32000":
                        mtype = { 'type' : 'audio/isac; codecs=isac' };
                        break;
                    case "iLBC/8000":
                        mtype = { 'type' : 'audio/iLBC; codecs=iLBC' };
                        break;
                    case "G722/8000":
                        mtype = { 'type' : 'audio/G722; codecs=G722' };
                        break;
                    default:
                        mtype = { 'type' : 'audio/opus; codecs=opus' };
                        break;
                }
            }
            var blob = new Blob(chunks, mtype);
            chunks = [];
            var audioURL = window.URL.createObjectURL(blob);
            remote_audio.removeAttribute('autoplay'); // Detiene la autoreproduccion del
archivo de audio grabado.
            remote_audio.src = audioURL;
        }
    }
    /***** Fin Grabador de Audio 1 *****/
}

/**
 * Función a ser llamada cuando el evento removestream ocurre en la interfaz RTCPeerConnection.
 * @param event De tipo RTCTrackEvent el cual es enviado cuando un MediaStream se elimina de esta
conexión.
 */
function handleRemoteStreamRemoved(event) {
    console.log('Remote stream removed. Event: ', event);
}

/**
 * Función para cerrar la conexion entre pares en el objeto RTCPeerConnection.
 */
function stop() {
    if(pc) {
        isStarted = false;
    }
}

```

```

        pc.close();
        pc = null;
    }

    /**
     * Función para cerrar la conexión entre pares y el websocket e informar al servidor del cierre al enviar
     * un
     * mensaje de terminación de la conexión.
     */
    function hangup() {
        console.log('Hanging up.');
```

stop();

sendMessage('bye');

socket.disconnect();

}

/**

* Función que maneja la desconexión por parte del otro par de la transmisión de audio al cerrar la

conexión actual

* y dejar al par listo para recibir otra conexión en la misma sala.

* (se activa al recibir un mensaje de 'bye' retransmitido por el servidor de señalización)

*/

function handleRemoteHangup() {

console.log('Session terminated.');

if(CHROME) { // Detiene el grabador de audio y obtiene el codec para establecer el MimeType

(sólo para Google Chrome).

if(isInitiator) {

var codec_last_used = getCodec(pc.remoteDescription.sdp); // Obtiene el codec

del sdp de RTCPeerConnection.

} else {

var codec_last_used = getCodec(pc.localDescription.sdp); // Obtiene el codec

del sdp de RTCPeerConnection.

}

media_recorder.stop();

}

stop();

isInitiator = true; // El par que queda luego de la desconexión del otro es el nuevo iniciador.

if (toggle_net_statistics == true) {

clearInterval(net_stats_interval_id); // Detiene la obtención de datos estadísticos de

la conexión.

}

cambiar_codec_prf_btn.disabled = false; // Habilitar la interfaz de selección de codec.

alert('La conexión se ha terminado');

}

/****** Funciones Codec *****/

/**

* Función para obtener el nombre del codec de audio en uso. (se obtiene del sdp local)

* @param sdp

* @return codec nombre del codec segun el sdp.

*/

function getCodec(sdp) {

var codec;

var sdpLines = sdp.split('\r\n');

var mLineIndex; // Almacena el índice de la línea m del sdp donde se encuentra la definición de

los codec de audio disponibles.

for (var i = 0; i < sdpLines.length; i++) {

if (sdpLines[i].search('m=audio') !== -1) {

mLineIndex = i;

break;

}

}

var codecPayload = (sdpLines[mLineIndex].split(' '))[3]; // obtiene la descripción del formato

de medios (media format description) del codec en uso (el cual se encuentra en la 4ta posición de la

línea m).

for (i = 0; i < sdpLines.length; i++) { // Busca según la descripción del formato de medios

(media format description) obtenido la definición del codec en uso.

if (sdpLines[i].search('a=rtpmap:'+codecPayload) !== -1) {

codec = (sdpLines[i].split(' '))[1];

break;

}

}

return codec;

}

/**

* Función para seleccionar el codec de audio a usar por defecto si está disponible.

* @param sdp

* @param codec nombre del codec segun el sdp.

```

* @return regexp
*/
function setCodec(sdp,codec) {
    var sdplines = sdp.split('\r\n');
    var mlineindex; // Almacena el índice de la línea m del sdp donde se encuentra la definición de
    los codec de audio disponibles.
    var regexp_codec = getRegExprCod(codec); // Usa una expresión regular dependiendo del codec
    seleccionado.
    console.log('selected codec:', codec);
    // Busca el índice de la línea m del sdp donde se encuentra la definición de los codec de audio
    disponibles.
    for (var i = 0; i < sdplines.length; i++) {
        if (sdplines[i].search('m=audio') !== -1) {
            mlineindex = i;
            break;
        }
    }
    if (mlineindex === null) {
        return sdp;
    }

    // Si el codec seleccionado está disponible lo configura como por defecto en la línea m.
    for (i = 0; i < sdplines.length; i++) {
        if (sdplines[i].search(codec) !== -1) {
            var codecPayload = extractSdp(sdplines[i], regexp_codec);
            console.log('selected codec payload:', codecPayload);
            if (codecPayload) {
                sdplines[mlineindex] = setDefaultCodecSdp(sdplines[mlineindex],
                codecPayload);
            }
            break;
        }
    }

    // Elimina CN en la línea m y el sdp.
    sdplines = removeCN(sdplines, mlineindex);

    sdp = sdplines.join('\r\n');
    return sdp;
}

/**
 * Función para obtener una extraer del sdp el codec seleccionado mediante
 * una expresión regular. Retorna por defecto la expresión regular de Opus.
 * @param codec nombre del codec según el sdp.
 * @return regexp expresión regular del codec seleccionado.
 */
function getRegExprCod(codec) {
    var regexp
    switch(codec){
        case "opus/48000":
            regexp = /:(\d+) opus\/48000/i;
            break;
        case "ISAC/16000":
            regexp = /:(\d+) ISAC\/16000/i;
            break;
        case "ISAC/32000":
            regexp = /:(\d+) ISAC\/32000/i;
            break;
        case "iLBC/8000":
            regexp = /:(\d+) iLBC\/8000/i;
            break;
        case "G722/8000":
            regexp = /:(\d+) G722\/8000/i;
            break;
        default:
            regexp = /:(\d+) opus\/48000/i;
            break;
    }
    return regexp;
}

/**
 * Función para extraer de la línea del sdp con el codec seleccionado la
 * descripción del formato de medios (media format description), al usar una
 * expresión regular.
 * @param sdpLine línea del sdp con el codec seleccionado.
 * @param regexprcodec expresión regular del codec.
 * @return result descripción del formato de medios del codec seleccionado.
 */
function extractSdp(sdpLine, regexprcodec) {
    var result = sdpLine.match(regexprcodec);

```

```

        return result && result.length === 2 ? result[1] : null;
    }

    /**
    * Función para colocar el codec seleccionado por defecto en el sdp al
    * colocar su descripción del formato de medios (media format description)
    * de primera en la línea m.
    * @param mLine línea m del sdp.
    * @param payload descripción del formato de medios del codec seleccionado.
    * @return newLine línea m modificada.
    */
    function setDefaultCodecSdp(mLine, payload) {
        var elements = mLine.split(' ');
        var newLine = [];
        var index = 0;
        for (var i = 0; i < elements.length; i++) {
            if (index === 3) { // El formato de medios empieza de la 4ta posición.
                newLine[index++] = payload; // Coloca la descripción del formato de medios del
                codec seleccionado de primera.
            }
            if (!(index >= 3) || elements[i] !== payload) { // Condición para evitar duplicar la
                descripción del formato de medios del codec seleccionado al tener en cuenta que el número de puerto en la
                segunda posición puede ser igual.
                newLine[index++] = elements[i];
            }
        }
        return newLine.join(' ');
    }

    // Strip CN from sdp before CN constraints is ready.
    function removeCN(sdpLines, mLineIndex) {
        var mLineElements = sdpLines[mLineIndex].split(' ');
        // Scan from end for the convenience of removing an item.
        for (var i = sdpLines.length - 1; i >= 0; i--) {
            var payload = extractSdp(sdpLines[i], /a=rtpmap:(\d+) CN\/(\d+)/i);
            if (payload) {
                var cnPos = mLineElements.indexOf(payload);
                if (cnPos !== -1) {
                    // Remove CN payload from m line.
                    mLineElements.splice(cnPos, 1);
                }
                // Remove CN line in sdp
                sdpLines.splice(i, 1);
            }
        }
        sdpLines[mLineIndex] = mLineElements.join(' ');
        return sdpLines;
    }

    /***** Funciones de visualización de la información de red *****/

    /**
    * Función para mostrar como caracteres los datos estadísticos de la conexión (RTCPeerConnection)
    * indicados en la interfaz RTCStatsReport.
    * @param results Datos estadísticos de la conexión RTCPeerConnection.
    * @return statsString Texto con los datos estadísticos de la conexión.
    */
    function dumpStats(results) {
        var statsString = '';
        results.forEach(function(res) {
            statsString += '<h3>Report type=';
            statsString += res.type;
            statsString += '</h3>\n';
            statsString += 'id ' + res.id + '<br>\n';
            statsString += 'time ' + res.timestamp + '<br>\n';
            Object.keys(res).forEach(function(k) {
                if (k !== 'timestamp' && k !== 'type' && k !== 'id') {
                    statsString += k + ': ' + res[k] + '<br>\n';
                }
            });
        });
        return statsString;
    }
}

```

4. Código fuente en lenguaje Javascript y Node.JS de la aplicación que se ejecuta de parte del servidor web / de señalización (index.js).

```
'use strict';
```

```

var express = require('express');
var app = express();
var server = require('http').Server(app);
var io = require('socket.io')(server);

server.listen(8080);

app.use(express.static(__dirname + '/'));
app.get('/', function (req, res) {
    res.sendFile(__dirname + "/" + "index.html" );
});

var rooms_list = []; // lista de salas creadas.

/**
 * Función para comparar la lista de salas creadas con las que contiene el adaptador y depurar la lista
 * para obtener las salas actualmente activas en el servidor.
 * @param roomList lista de salas creadas.
 * @return roomList lista de salas creadas actualmente activas en el servidor.
 */
function findRooms(roomList) {
    var adapterRooms = [];
    var rooms = io.of('/').adapter.rooms;
    if (rooms) { // busca en el adaptador las salas existentes y las guarda en adapterRooms.
        for (var room in rooms) {
            if (!rooms[room].hasOwnProperty(room)) {
                adapterRooms.push(room);
            }
        }
        for (var i=0; i<roomList.length; i++) { // compara la lista de salas creadas con las que se
            encuentran en la lista de salas del adaptador y depura la lista de salas creadas.
                var ind = adapterRooms.indexOf(roomList[i]);
                if (ind == -1) {
                    var ind2 = roomList.indexOf(roomList[i]);
                    roomList.splice(ind2,1);
                }
            }
        }
        return roomList;
    }
}

io.sockets.on('connection', function(socket) {

    /**
     * Función para registrar los mensajes del servidor en el cliente.
     */
    function log() {
        var array = ['Message from server:'];
        array.push.apply(array, arguments);
        socket.emit('log', array);
    }

    /**
     * Función para retransmitir los mensajes del cliente a los demás miembros de la sala.
     * @param room nombre de la sala.
     * @param message
     */
    socket.on('message', function(message, room) {
        log('Client said: ', message);
        socket.broadcast.to(room).emit('message', message); // emite un mensaje sólo para los
        miembros de la sala (a excepción del emisor).
    });

    /**
     * Función para manejar la petición de la lista de salas activas en el servidor y enviarla al
     cliente.
     */
    socket.on('request room list', function() {
        rooms_list = findRooms(rooms_list);
        socket.emit('room list', rooms_list);
    });

    /**
     * Función para manejar la petición de la creación de una sala, o de unirse a ella. Cada sala
     tiene un límite de dos clientes.
     * @param room nombre de la sala.
     */
    socket.on('create or join', function(room) {
        log('Received request to create or join room ' + room);
    });
});

```

```

        var clientsInRoom = io.nsp['/'].adapter.rooms[room]; // mira en el namespace por
defecto ("/") el número de clientes en la sala especificada.
        var numClients = clientsInRoom === undefined ? 0 :
Object.keys(clientsInRoom.sockets).length; // socket.io 1.4.8
        log('Room ' + room + ' now has ' + (numClients + 1) + ' client(s)');

        if (numClients === 0) { // si es el creador de la sala.
            socket.join(room);
            rooms_list.push(room);
            log('Client ID ' + socket.id + ' created room ' + room);
            socket.emit('created', room, socket.id);

            rooms_list = findRooms(rooms_list);
            socket.broadcast.emit('room list', rooms_list); // envia un mensaje de
actualización de la lista de salas a todos menos al emisor.

            } else if (numClients === 1) { // si se une a una sala ya existente.
            log('Client ID ' + socket.id + ' joined room ' + room);
            io.sockets.in(room).emit('join', room);
            socket.join(room);
            socket.emit('joined', room, socket.id);
            io.sockets.in(room).emit('ready');
            } else { // si se intenta a unir a una sala llena (máximo 2 clientes).
            socket.emit('full', room);
        }

    });

    /**
     * Función para manejar la petición de salirse de una sala de un cliente.
     * @param room nombre de la sala.
     */
    socket.on('leave', function(room) {
        log('Received request from Client ID ' + socket.id + ' to leave room ' + room);
        socket.leave(room);
        socket.emit('left', room);
        var clientsInRoom = io.nsp['/'].adapter.rooms[room]; // mira en el namespace por
defecto ("/") el número de clientes en la sala especificada.
        var numClients = clientsInRoom === undefined ? 0 :
Object.keys(clientsInRoom.sockets).length; // socket.io 1.4.8
        log('Room ' + room + ' now has ' + (numClients + 1) + ' client(s)');

        rooms_list = findRooms(rooms_list);
        io.emit('room list', rooms_list); // envia un mensaje de actualización de la lista de
salas a TODOS los clientes.
    });

    /**
     * Función para manejar la petición de desconexión de un cliente.
     * @param room nombre de la sala.
     */
    socket.on('bye', function(room){
        console.log('received bye');
    });

    /**
     * Función para manejar la desconexión de un cliente.
     */
    socket.on('disconnect', (reason) => {
        rooms_list = findRooms(rooms_list);
        socket.broadcast.emit('room list', rooms_list); // envia un mensaje de actualización de
la lista de salas a todos menos al emisor.
    });
});

```