



**San Jose State University**  
**Department of Electrical Engineering**

---

**EE-284 - Section - 2**  
**Voice and Data Networks**

**GROUP - 3**

*Course Project 2*

**Audio and Video Communication using WebRTC**

---

*Submitted to -*

**Prof. Balaji Venkatraman**

*Submitted by -*

**Shivang Singh**  
**Shivani Sinhmar**

## **Table of Contents**

<b><u>Contents</u></b>	<b><u>Page Numbers</u></b>
Project Overview	3
Steps for Implementation	4
About WebRTC Introduction	5-7
WebRTC Protocols	8-9
WebRTC Libraries	10
Project Simulation Results	11-12
Source Code	13-14
Conclusion	14
References	15

## **Table of Figures**

<b><u>Figures</u></b>	<b><u>Page Numbers</u></b>
1 : A general framework of WebRTC	6
2 : A simple WebRTC Communication	7
3 : WebRTC Protocol Stack	8

## **Project Overview**

- The main objective of this project is to implement WebRTC tool that is used to pervade the Real time communication (RTC) in mobile applications as well as in web browsers. It is used to share audio as well as video media between two distant browsers.
- This project aims at learning the API set which is responsible for communication between two browsers.
- We aim at creating an application using WebRTC libraries and also to leverage the capabilities of libraries like MediaStream, RTC Peer Connection and RTC Data Channel, for creating a peer-to-peer communication.
- The usage of MediaStream and RTC Peer Connection are shown in this project work.

## **Steps for Implementation**

The steps for implementing this project are as below -

- Firstly we install any text editor like ATOM.
- Install NPM compatible platform, io.js; based on node.js
- Write a program for HTML file in Atom and save it as index.html
- Write a program in javascript which defines getUserMedia.
- We get package.json file while installing .json
- Used command “npm install getusermedia” to install dependencies.
- Started the server by command “npm start”.
- Opened Web browser and entered “localhost:9966” in address bar.
- For peer, opened another browser and wrote “localhost:9966/#init”.
- The value of field id that is present in inti browser is shared with other web browser.
- Similarly, other web browser’s id field is also shared with the first browser.
- Now clicked on Connect to initiate the connection.
- This allows 2 web browsers to share media like audio and video.

## **About WebRTC - Introduction**

The full form of WebRTC is Web Real Time Communications. WebRTC uses an API set that is plug-in free.

- It is used in mobile browsers or desktop browsers.
- Earlier, an external plugin was also needed for implementation of WebRTC.
- WebRTC uses different protocols and standards which include data streams, TURN servers, ICE, SDP, NAT, TCP/UDP and network sockets.

The main use of WebRTC is real time communication between peers. It is obvious that an agreement is needed between two browsers, find a common path for connection, bypass firewall/security challenges, and transmit real time multimedia for the communication.

The main challenge and problem in implementing WebRTC is that in locating and establishing a network socket connection with other's browser; since only after this only we can transmit multimedia bidirectionally.

WebRTC has been standardized by the World Wide Web Consortium (W3C) and Internet Engineering Task Force (IETF).

- The later implication was released as a free software under terms of a BSD license.
- It also gives another free usage which is based on multimedia framework Gstreamer.

We know that there are basically two methods by which a website can be accessed.

→ The first one being that we directly enter the address of the webpage and other by clicking on some link so that it leads us to the website there. The client requests the server to provide the web page. Then the server response with an HTML response containing the webpage. The objective here we have is that the client initiates a HTTP request to the server that is already known or can be found via DNS queries. To this request the server will send a corresponding HTTP response that contains the details for the web page which has been requested by the client on the other side.

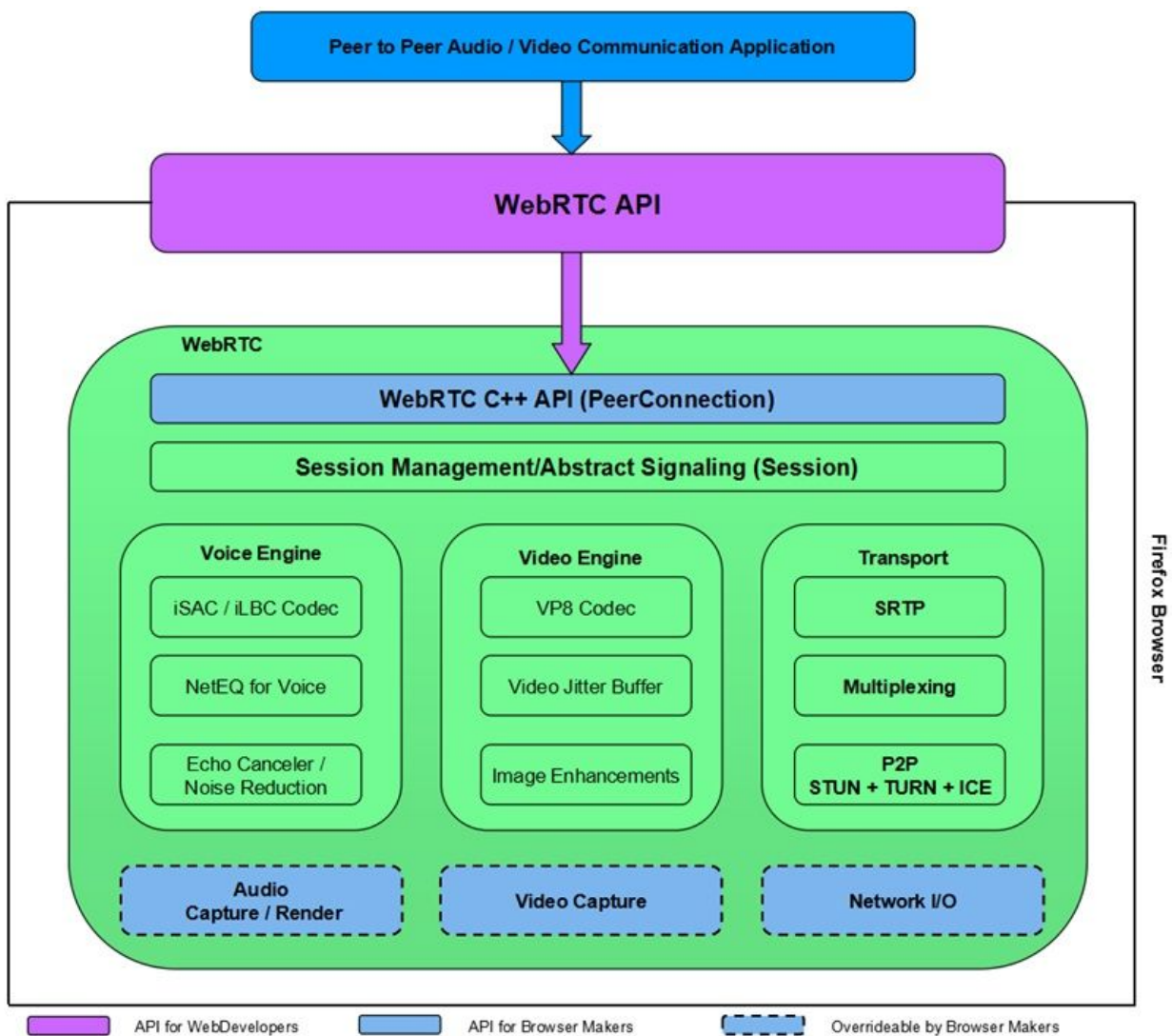


Figure 1: A general framework of WebRTC <sup>(\*)</sup>

As we can see in the above figure of Framework of WebRTC, there are 2 layers of architecture of WebRTC. They are WebRTC C++ API and Web API.

- Web Apps are applications which are created by increasing functionalities as given by WebAPI. Web RTC API has a set of extremely useful and powerful libraries such as MediaStream, RTCPeerConnection, RTCDataChannel.
- MediaStream API is used for streaming audio/video multimedias from one peer to another in real time.
- RTCPeerConnection API is used for setting up the connection between peers.
- RTCDataChannel API is used for transferring data between peers and also for building chat application in which peer can send messages as well as data through one web browser.

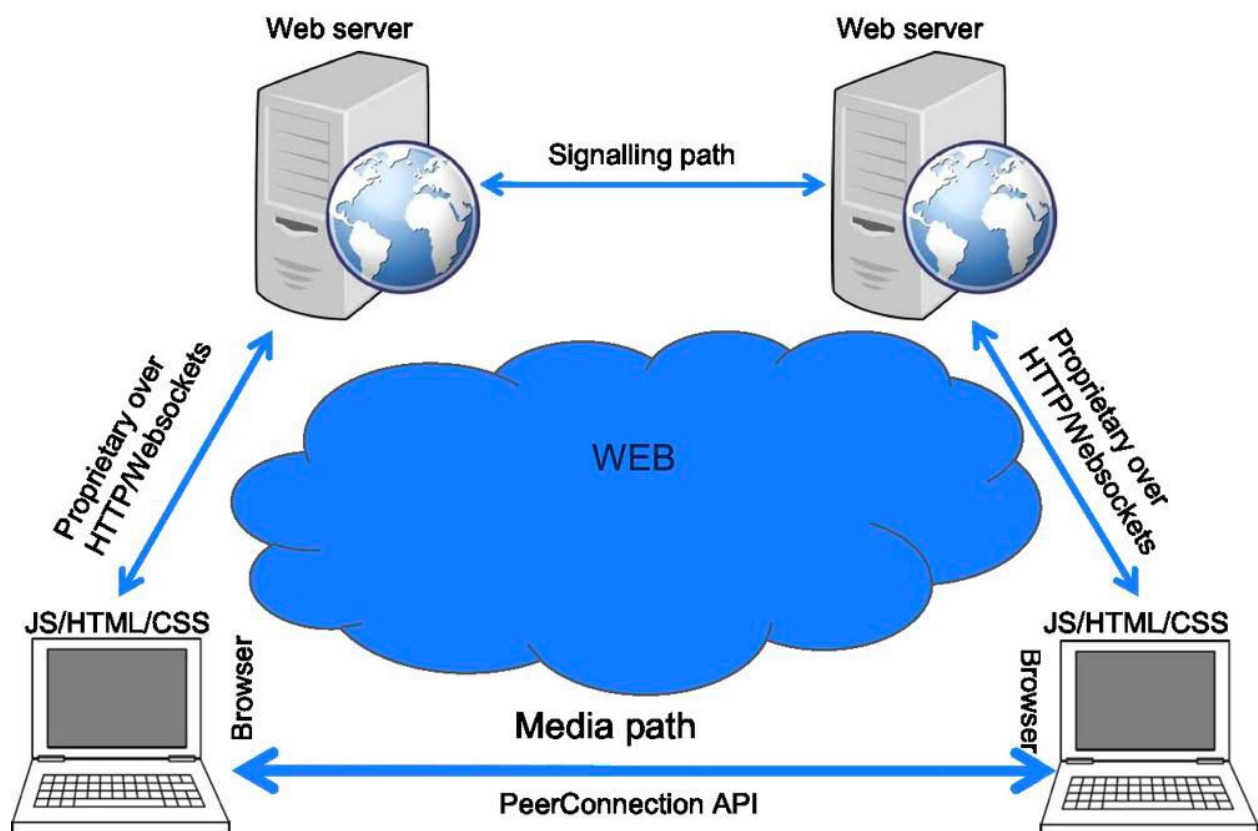
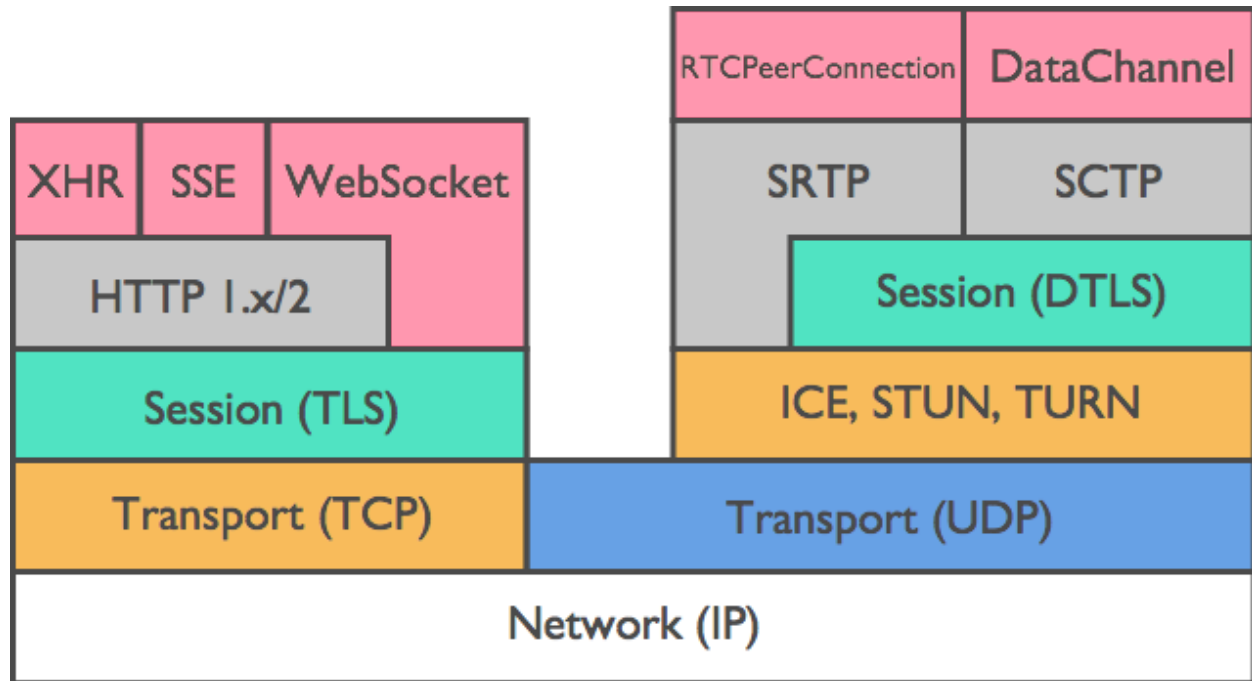


Figure 2 : A simple WebRTC communication sample <sup>(\*)</sup>



## WebRTC Protocols

WebRTC has predefined protocol stack as shown in the following figure -



*Figure 3: WebRTC Protocol Stack <sup>(\*)</sup>*

In this stack;

- ICE, TURN and STUN WebRTC servers are used for maintaining peer to peer communication via UDO. It mandates data exchanged between peers must be encrypted.
- Datagram Transport Layer Security (DTLS) protocol is used for encryption of data which is being exchanged between the peers. For exchanges between Chrome and Firefox, SCTP AND SRTP are the protocols which are used, since they provide reliability, congestion control and flow control.

## **PROTOCOLS WITHIN STACK-**

1. **ICE** - Interactive Connectivity Establishment (*ICE*) is a framework to allow your web browser to connect with the peers. There are reasons why straight up connection from Peer A to B does not work; since it need to bypass firewalls that stops from opening the connection.
2. **STUN** - Session Traversal Utilities for NAT (*STUN*) is a protocol which is used to discover the public address of user and determine any restrictions in the router if that would prevent a direct connection with another peer.  
→ In this the client send a request to a STUN server on Web, who replies with the client's public address irrespective of the router's NAT.
3. **NAT** - Network Address Translation (NAT) is used for giving your personal device a public IP address. A router will have a public IP address and every device connected to the router gets a private IP address. Requests will be translated from the device's private IP to router's public IP with a unique port.  
→ Some routers will have restrictions on who can connect to devices on the network. This means that no one can create a connection even when we have a public IP address, found by the STUN server. So we switch to TURN .
4. **TURN** - Routers using NAT employment as a restriction are known as Symmetric NAT, which mean that those routers will only be accepting the connection from peers with whom you have connected previously. Traversal Using Relays around NAT (TURN) bypasses these symmetric NAT restriction by making a new connection with a TURN server and relaying all information through that server.

## **WebRTC Libraries**

WebRTC Libraries are differentiated into -

1. MediaStream
2. RTCPeerConnection
3. RTCDataChannel

1. **MediaStream** - The Media Capture and Streams API, often called the Media Stream API or Stream API, is an API related to WebRTC which supports streams of video and audio data, the constraints associated with that type of data.
2. **RTCPeerConnection** - The RTCPeerConnection interface gives a WebRTC connection between the local computer and remote peer. It gives methods for connecting, maintain and monitor the connection and then closing the connection once it is no longer needed, to a remote peer.
3. **RTCDataChannel** - The RTC Data Channel interface is a feature of the WebRTC API which lets you open a channel between two peers over which you send and receive data. The API is similar to WebSocket API, so that same model can be used for each other.

## **Simulation Results Obtained**


localhost:9966/#init    localhost:9966

localhost:9966/#init

Your ID:  
`{"type": "offer", "sd": "v=0\r\nno=-"}`

Other ID:  
`webrtc-datachannel1\r\n1024\r\n"`

Enter Message:



Windows taskbar: Type here to search, icons for File Explorer, Edge, etc., system tray shows 11:09 PM 12/8/2017.


localhost:9966/#init    localhost:9966    New Tab    Inbox (11,097) - binita.p...

localhost:9966

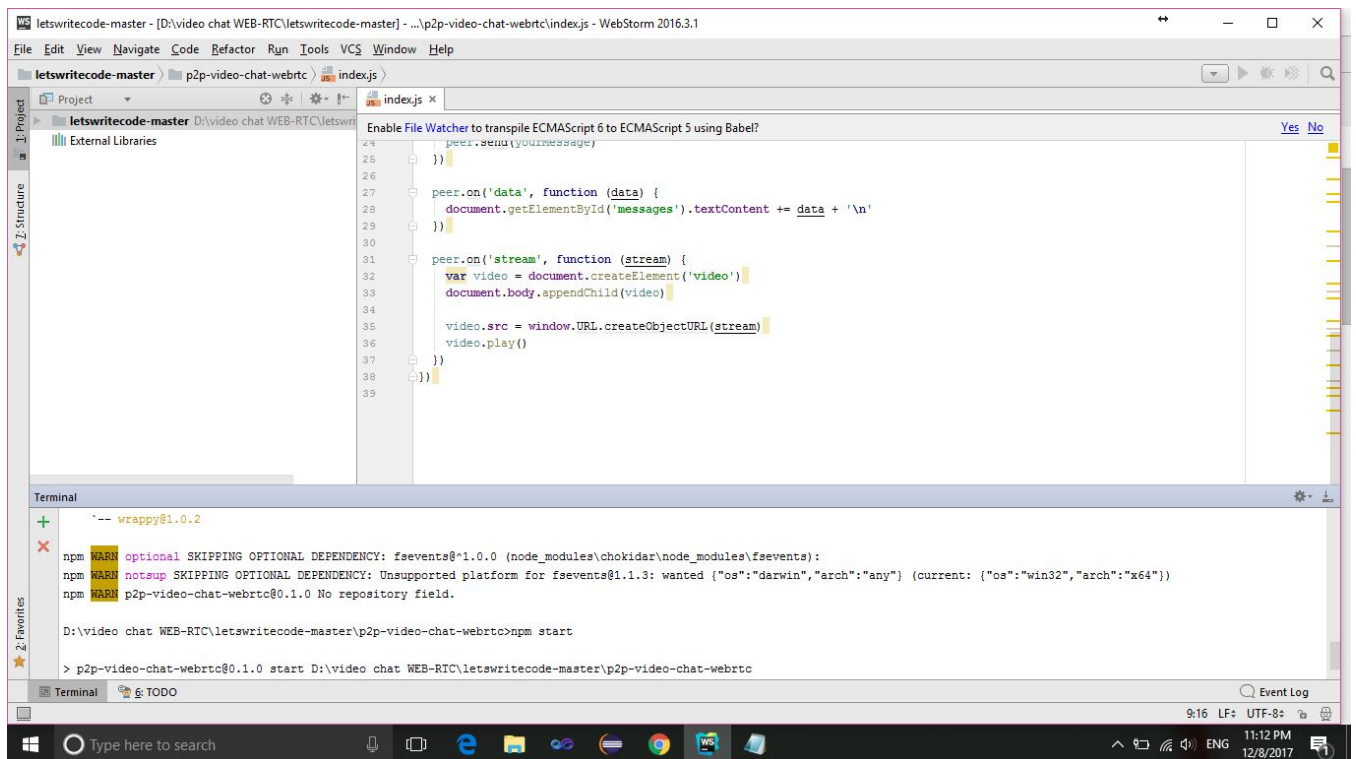
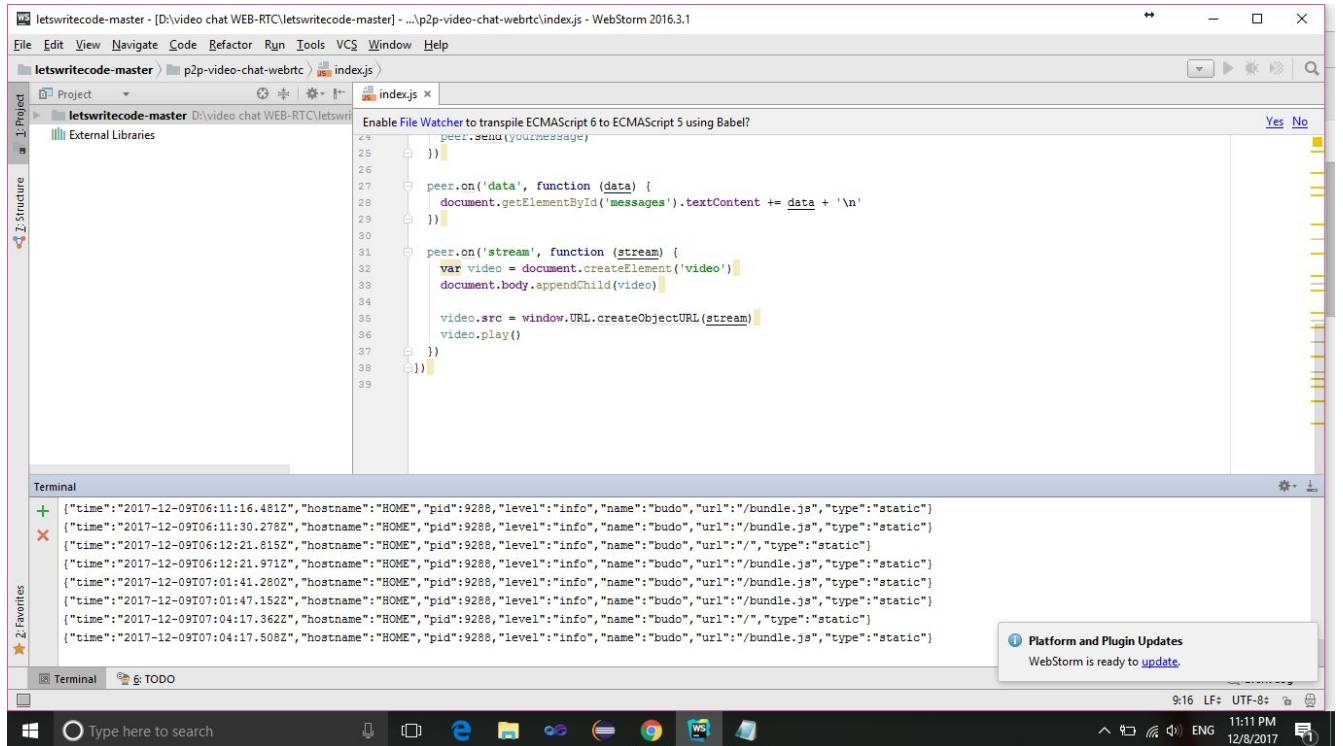
Your ID:  
`{"type": "answer", "sd": "v=0\r\nno=-"}`

Other ID:  
`webrtc-datachannel1\r\n1024\r\n"`

Enter Message:



Windows taskbar: Type here to search, icons for File Explorer, Edge, etc., system tray shows 11:10 PM 12/8/2017.



# Source Code

HTML Code -

```
1  <html>
2  <head>
3  <base target="_blank">
4  <title>Media Stream</title>
5  </head>
6  <body>
7      <video id="localVideo" autoplay muted></video>
8      <video id="remoteVideo" autoplay muted></video>
9
10     <div>
11         <button id="start">Start</button>
12         <button id="call">Call</button>
13         <button id="hangup">Hang Up</button>
14     </div>
15
16     <script src="284.js"></script>
17 </body>
18 </html>
19
20
```

JS Code -

```
284.js
1  var localStream, localPeerConnection, remotePeerConnection;
2
3  var localVideo = document.getElementById("localVideo");
4  var remoteVideo = document.getElementById("remoteVideo");
5
6  var startButton = document.getElementById("start");
7  var callButton = document.getElementById("call");
8  var hangupButton = document.getElementById("hangup");
9  startButton.disabled = false;
10 callButton.disabled = true;
11 hangupButton.disabled = true;
12 startButton.onclick = start;
13 callButton.onclick = call;
14 hangupButton.onclick = hangup;
15 navigator.getUserMedia = navigator.getUserMedia || navigator.webkitGetUserMedia || navigator.mozGetUserMedia;
16
17 function logger(text) {
18     console.log(text);
19 }
20
21 function getStream(stream){
22     localVideo.src = URL.createObjectURL(stream);
23     localStream = stream;
24     var audioContext = new AudioContext();
25     var mediaStreamSource = audioContext.createMediaStreamSource(stream);
26     mediaStreamSource.connect(audioContext.destination);
27     callButton.disabled = false;
28 }
29
30 function start() {
31     logger("Requesting local stream");
32     startButton.disabled = true;
33     navigator.getUserMedia({audio:true, video:true}, getStream,
34         function(error) {
35             logger("Error in getUserMedia function: ", error);
36         });
37 }
38
39 function call() {
40     callButton.disabled = true;
41     hangupButton.disabled = false;
42
43     var servers = null;
44
45     localPeerConnection = new RTCPeerConnection(servers);
46     logger("Created local peer connection object..");
47     localPeerConnection.onicecandidate = getIceCandidates;
48 }
49
```

```
284.js
47 localPeerConnection.onicecandidate = getIceCandidates;
48
49 remotePeerConnection = new RTCPeerConnection(servers);
50 logger("Created remote peer connection object.");
51 remotePeerConnection.onicecandidate = getIceCandidateRemote;
52 remotePeerConnection.onaddstream = getRemoteStream;
53
54 localPeerConnection.addStream(localStream);
55 logger("Added localStream to localPeerConnection");
56 localPeerConnection.createOffer(getDescription,errorHandler);
57 }
58
59 function getDescription(description){
60 localPeerConnection.setLocalDescription(description);
61 logger("Offer from localPeerConnection: \n" + description.sdp);
62 remotePeerConnection.setRemoteDescription(description);
63 remotePeerConnection.createAnswer(getRemoteDescription,errorHandler);
64 }
65
66 function getRemoteDescription(description){
67 remotePeerConnection.setLocalDescription(description);
68 logger("Answer from remotePeerConnection: \n" + description.sdp);
69 localPeerConnection.setRemoteDescription(description);
70 }
71
72 function hangup() {
73 logger("Call will be ended now...");
74 localPeerConnection.close();
75 remotePeerConnection.close();
76 localPeerConnection = null;
77 remotePeerConnection = null;
78 hangupButton.disabled = true;
79 callButton.disabled = false;
80 }
81
82 function getRemoteStream(event){
83 remoteVideo.src = URL.createObjectURL(event.stream);
84 logger("Received remote stream");
85 }
86
87 function getIceCandidates(event){
88 if (event.candidate) {
89 remotePeerConnection.addIceCandidate(new RTCIceCandidate(event.candidate));
90 logger("Local ICE candidate: \n" + event.candidate.candidate);
91 }
92 }
93
Line 12, Column 13 Tab Size: 4 JavaScript
```

```
284.js
56 localPeerConnection.createOffer(getDescription,errorHandler);
57 }
58
59 function getDescription(description){
60 localPeerConnection.setLocalDescription(description);
61 logger("Offer from localPeerConnection: \n" + description.sdp);
62 remotePeerConnection.setRemoteDescription(description);
63 remotePeerConnection.createAnswer(getRemoteDescription,errorHandler);
64 }
65
66 function getRemoteDescription(description){
67 remotePeerConnection.setLocalDescription(description);
68 logger("Answer from remotePeerConnection: \n" + description.sdp);
69 localPeerConnection.setRemoteDescription(description);
70 }
71
72 function hangup() {
73 logger("Call will be ended now...");
74 localPeerConnection.close();
75 remotePeerConnection.close();
76 localPeerConnection = null;
77 remotePeerConnection = null;
78 hangupButton.disabled = true;
79 callButton.disabled = false;
80 }
81
82 function getRemoteStream(event){
83 remoteVideo.src = URL.createObjectURL(event.stream);
84 logger("Received remote stream");
85 }
86
87 function getIceCandidates(event){
88 if (event.candidate) {
89 remotePeerConnection.addIceCandidate(new RTCIceCandidate(event.candidate));
90 logger("Local ICE candidate: \n" + event.candidate.candidate);
91 }
92 }
93
94 function getIceCandidateRemote(event){
95 if (event.candidate) {
96 localPeerConnection.addIceCandidate(new RTCIceCandidate(event.candidate));
97 logger("Remote ICE candidate: \n" + event.candidate.candidate);
98 }
99 }
100
101 function errorHandler(){
102
Line 12, Column 13 Tab Size: 4 JavaScript
```

## **Conclusion**

- By doing this project, we got an insight of the working of WebRTC and learnt how everything works in our daily life, which looks simple but are complex and have multiple steps and hops.
- We have successfully implemented and learnt MediaStream library and RTCPeerConnection WebRTC libraries.
- Our code is written in coding languages of HTML5 and JavaScript.



## **References**

1. <https://www.html5rocks.com/en/tutorials/webrtc/basics/>
2. <http://webrtc-security.github.io/>
3. <https://www.webrtc-experiment.com/docs/WebRTC-Signaling-Concepts.html>
4. <https://www.html5rocks.com/en/tutorials/webrtc/infrastructure/>
5. <https://github.com/muaz-khan/WebRTC-Experiment/wiki/RTCPeerConnection-Documentation>