REVIEW AND STUDY OF WEBRTC TECHNOLOGY AND IT'S USE CASES

[1]J.D. Paghadal, [2]N. M. Mochi, [3]B.H. Soni, [4]S. Bhaskar
[1]Student, [2]Student, [3]Student, [4]Assistant Professor
[1]B. TECH (Information Technology)
[1]Parul University, Vadodara 391760, Gujarat, India

Abstract: There are many ways for data communication and there are many medium with which data communication is possible. One of most communication ways in practice is audio and video communication. There are many types of audio and video communication also. For ex., one-way communication (audio or video broadcasting), two-way communication (audio or video calling), and multi-way communication (conference calling or group chat). WebRTC is one such technology that can be used to serve the purpose of data communication through Browser-to-Browser. WebRTC is open standard & secure peer-to-peer communication of audio, video or data. WebRTC operates and implements 3 major Application Programming Interfaces (APIs): 1.MediaStream (aka getUserMedia), 2.RTCPeerConnection, 3.RTCDataChannel. WebRTC also encodes and decodes data packets by itself. So, improving security for data communication. Also, WebRTC can handle Browser communication with PSTN calls.

Index Terms **- W3C, Real Time Communication, WebRTC, Media Engine, NAT, STUN, TURN, MCU.**

I. INTRODUCTION

Web Real-Time Communication (WebRTC) is free and opensource technology for Real-Time communication developed by Justin Uberti and Peter Thatcher, initial released 8 years ago in 2011 [14]. It is plugin free, low cost, high quality and secure peer-to-peer real-time communication of voice, video and data through simple Application Programming Interfaces (APIs) [15]. Some of the advantages of WebRTC apart from being free and open source is having effective peer-to-peer communication only through web browsers, it means no external plugins or application installation required to use WebRTC, unlike Skype which requires installation of external application for different platforms (Such as Desktop, IOS, Android, etc). Any application developed using WebRTC requires WebRTC enabled browsers. Initially it was not supported by all browsers. So, native application was required, and WebRTC has great support for native IOS & Android applications. But, as of now all the modern browsers support WebRTC except Internet Explorer. Still, if required Internet Explorer can run WebRTC through plugins.

Specific applications such as video chat, video conferencing, telephony, high speed file transfer, low latency data communication, online games playing, and many more can be implemented with WebRTC. WebRTC is collection of APIs and it implements 3 APIs. Apart from API implementation there is signaling mechanism required to identify clients and establishing effective peer-to-peer connection between them, these all thing can be handled and implemented using WebRTC. WebRTC handles APIs, Identification of peers, NAT Traversal and security for encryption of data [8] [9].

In this study we have discussed about WebRTC as Technology or media engine and survey for the same [3] *and brief description to implement simple WebRTC application, advantages and disadvantages of using WebRTC, signaling mechanism* [2], *Challenges and establishment of multiple-peer communication through WebRTC using various architectures* [15].

II. BACKGROUND

There exist many methods to implement and establish peer-to-peer Real-Time Communication and WebRTC is one such method as it plays an important role for browser-to-browser communication. WebRTC is of collection of APIs but it is simply HTML5 based technology and combined with few lines of JavaScript code, applications for Real-Time Communication can be developed easily. WebRTC mostly uses UDP based communication for browsers rather than TCP based HTTP communication [17]. So, WebRTC based applications can be developed and used where it is important to send data with high speed and low latency rather than to verify that data has been received by other side or not [15]. We can take an example of video chat in which WebRTC can be used, because even if there is loss of few frames in receiving side then also no huge problem can arise but if there are applications such as sending and receiving critically important files then if there is loss of even single data packet, the whole data can be corrupted. So, based on application requirements one can choose to use WebRTC or not[15].

III. METHODOLOGY

WebRTC is kind of media engine and it manages real-time transmission and receiving of audio or video. WebRTC application has several requirements such as [3] [15]:

- Getting streaming data (audio, video or any other data).
- Getting network information (which includes IP addresses, ports) and then exchange this information with other Peers (also known as WebRTC clients), which enables connection, and it can work even if there are NATs and firewalls.
- Signaling mechanism for initiate and close session.
- Exchange media and client information. Such as, Codecs, Resolution, etc.

3.1 Three Main Tasks of WebRTC

There are three main categories of API that exist in WebRTC, which represents three different tasks associated with it [3] [15]:

1. Acquiring audio and video
2. Communicating audio and video
3. Communicating arbitrary data

Three APIs associated with these tasks are respectively [3] [15]:

1. MediaStream (aka getUserMedia)
2. RTCPeerConncetion
3. RTCDataChannel

So, further we will brief about each of these APIs and Signaling mechanism.

3.1.1 MediaStream (getUserMedia)

MediaStream API represents synchronized stream of media. For example, stream taken from camera and mic input has synchronized video and audio tracks. Each MediaStream has an input, which is generated by getUserMedia(), and it has an output stream which can be either used to view video stream output on the browser or it can be passed to RTCPeerConnection for communication purpose.

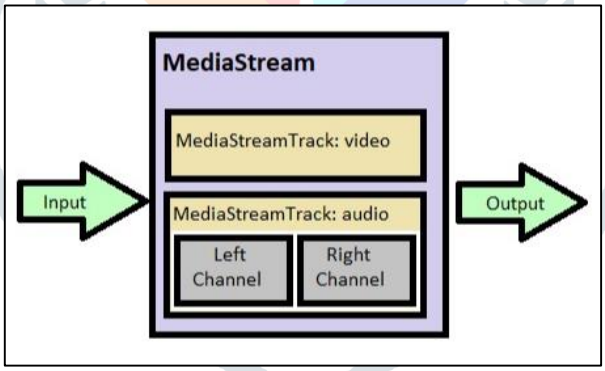

Figure 1-MediaStream [15]

In simple terms MediaStream is used to get access of media streams such as audio or video or both from user's mic or camera. This method explicitly asks for user's permission for camera or mic access.

navigator.getUserMedia(constraints, successCallback, errorCallBack)
It takes three argument and First one is constraints. Such as audio or video. And in successCallback, source of video is set using the stream returned by getUserMedia(). If access permission of camera or mic is denied by user, then errorCallBack function is invoked.

Sample code to access camera using getUserMedia():

```
var constraints = {video: true};
function successCallback (stream) {
        var video = document.querySelector("video");
        video.src = window.URL.createObjectURL(stream);
}
function errorCallback (error) {
        console.log ("navigator.getUserMedia error: ",error);
}
```

navigator.getUserMedia(constraints, successCallback, errorCallback); [16]

3.1.2 RTCPeerConnection

As the name applies, RTCPeerConnection is all about making connection to another peer and over this peer connection we can send audio and video. The way to do this is by taking media streams that we got from getUserMedia() and plug them into RTCPeerConncetion and send them off to the other side. When the other side receives them, they will pop out as new media stream on their peer connection and then they can plug these media streams to video element to display it on a web page. So, both sides of peer connections, they both gets streams from the getUserMedia() and then they plug these streams in with RTCPeerConncetion and then these streams pops out encoded and decoded on the other side.

Now, under the hood RTCPeerConncetion do a lot of work, which includes:

- Signal processing: To remove noise from Audio and Video.
- Codec handling: Codec selection, compression and decompression of the actual audio and video.
- Peer-to-Peer communication: Finding the actual peer-to-peer route through firewalls, through NATs, through relays.
- Security: Encrypting the data, so that user's data gets fully protected all time.
- Bandwidth Management: Managing bandwidth for communication.

Things mentioned above are completely done by WebRTC to hide the WebRTC complexity from user or developer. So, developer don't have to worry too much about internal things and can direct write code to develop WebRTC app.

Below are some of the things which are also handled by WebRTC:

- Packet loss concealment (PLC): minimize the effect of packet loss during communication.
- Echo cancellation: this is as a part of signal processing, to improve quality of voice quality and remove noise.
- Bandwidth Adaptivity: using available bandwidth and manipulating it for effective real-time communication.
- Dynamic jitter buffering: it helps to adapt changes in network delay.
- Automatic gain control (AGC): it uses several methods to adjust mic signals, so it helps to improve quality of voice by making it loud and clear to the other side of peer communication.
- Noise reduction and suppression: Reducing or removing unwanted audio, which is mixed up with actual audio.
- Image cleaning: Improve the quality of image and video frames, at most possible extent.

RTCPeerConncetion can be established in two ways:

(I) RTCPeerConncetion without servers:

This method has a local and remote RTCPeerConncection on the same web page. This might not look like something very useful. Because, caller and receiver are on same page. Still it does make working of RTCPeerConncetion API clearer, because here RTCPeerConncetion objects are on same page and can exchange data and messages directly without using intermediate Signaling mechanism.

(II) RTCPeerConncetion with servers:

In real-time remote communication WebRTC needs servers to do following things:

- Users can find each other and can exchange details such as name.
- WebRTC clients (peers) can exchange network details.
- Peers exchanges data about video resolution and format.
- WebRTC client application can pass through NAT and firewalls.

The core server-side functionalities required by WebRTC are:

- User discovery and User communication.
- Signaling (this is the key requirement and basis to establish RTCPeerConnection).
- NAT and firewall traversal.
- Requirement of relay server in case of failure of peer-to-peer communication.

Sample code for RTCPeerConnection:

```
pc = new RTCPeerConncetion(null);
pc.onaddstream = gotRemoteStream;
pc.addstream(localStream);
pc.createOffer(gotOffer);
function gotOffer(desc) {
        pc.setLocalDescription(desc);
        endOffer(desc);
```

```
}
function gotAnswer(desc) {
        pc.setRemoteDescription(desc);
}
function gotRemoteStream(e) {
        attachMediaStream(remoteVideo, e.stream);
} [15]
```

Here, in the code new RTCPeerConncetion is created and when the stream is received the call back for that in gotRemoteStream there attaches the media we are getting from video element to the stream. It is also creating an offer giving information about media, and setting that as local description, then sending that to a receiver side, so that receiver will able to set remote description that is shown in gotAnswer() function.

3.1.3 RTCDataChannel

As WebRTC main use case may look like audio and video communication. But it is not limited to audio and video. It is also possible to do communication of any kind of data using WebRTC and RTCDataChannel is having one such use case. RTCDataChannel can be used for bidirectional communication of arbitrary data between peers with low-latency and high throughput. RTCDataChannel is like WebSocket. But still it is much faster than WebSocket because of direct communication between browsers, even if Relay (TURN) server is required when passing through NATs and firewall fails [15].

Some of the RTCDataChannel characteristics:

- Same as WebSocket: It is intentionally designed in such a way that people familiar with WebSocket can also use similar APIs for RTCDataChannel.
- Ultra-low latency
- Unreliable or reliable: These are delivery semantics and can be kind of TCP vs UDP.
- Secure: It uses standard DTLS encryption technique to make sure that the packages send across data channel are fully encrypted on their way to destination.

Here, is the sample code and it has very similar syntax to WebSocket, with **send()** method and a **message** event [11]:

```
const localConnection = new RTCPeerConnection(servers);
const remoteConnection = new RTCPeerConnection(servers);
const sendChannel = localConnection.createDataChannel('sendDataChannel');
remoteConnection.ondatachannel = (event) => {
        receiveChannel = event.channel;
        receiveChannel.onmessage = onReceiveMessage;
        receiveChannel.onopen = onReceiveChannelStateChange;
        receiveChannel.onclose = onReceiveChannelStateChange;
};
function onReceiveMessage(event) {
        document.querySelector("textarea#send").value = event.data;
}
document.querySelector("button#send"). onclick = () => {
        var data = document.querySelector("textarea#send").value;
        sendChannel.send(data);
} [15]
```

3.2 Signalling

The process with which both the peers gets agree to conduct session is known as Signaling. It is not part of RTCPeerConncetion. Signaling and protocols are not specified by WebRTC. In fact, WebRTC application developer can choose protocols based on their needs and requirements. Such as, Session Initiation Protocol (SIP), XMPP or any appropriate two-way (full duplex) communication channel.

There are three types of information exchanged using signaling and exchange of these information must be completed prior to peer-to-peer streaming can begin:

1. Session control messages: for initialization and closing communication and to report errors
2. Network configuration: Such as, IP address and Port number
3. Media Capabilities: codecs and resolution supported by our browser as well as other side peer's browser.

The above information is also called as session description and it need to be exchanged. It can use any messaging mechanism or any messaging protocol. [2] [6] [15] [17]

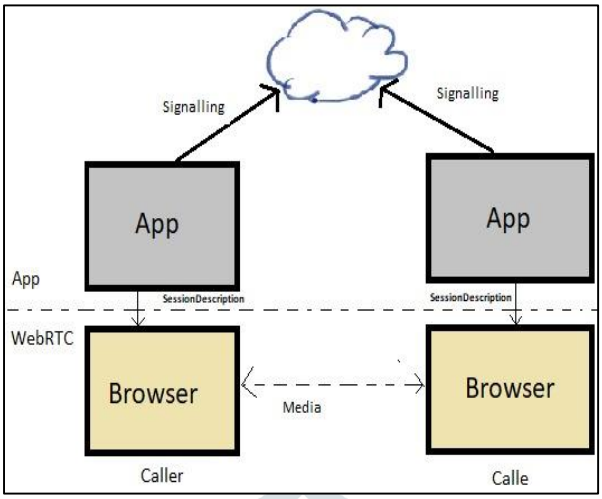

Figure 2-Sigaling Diagram [15]

3.2.1 STUN and TURN

(I) STUN

NATs consist private IP address of a computer and private IP addresses are not useful for communication. So, there is no way with which we make the link peer-to-peer, unless we have public IP address. So, this is the place where STUN server technology is used. We can contact STUN servers from WebRTC and ask for public IP address. So, request for public IP comes into the STUN server, then it finds the address from which that request came, then puts that address into the packet and sends it back. So, now WebRTC knows public IP address and now STUN server doesn't need to be in the communication anymore. STUN servers are simple and super cheap to run and capable of handling data flow peer-to-peer. But STUN server may not work in every case.

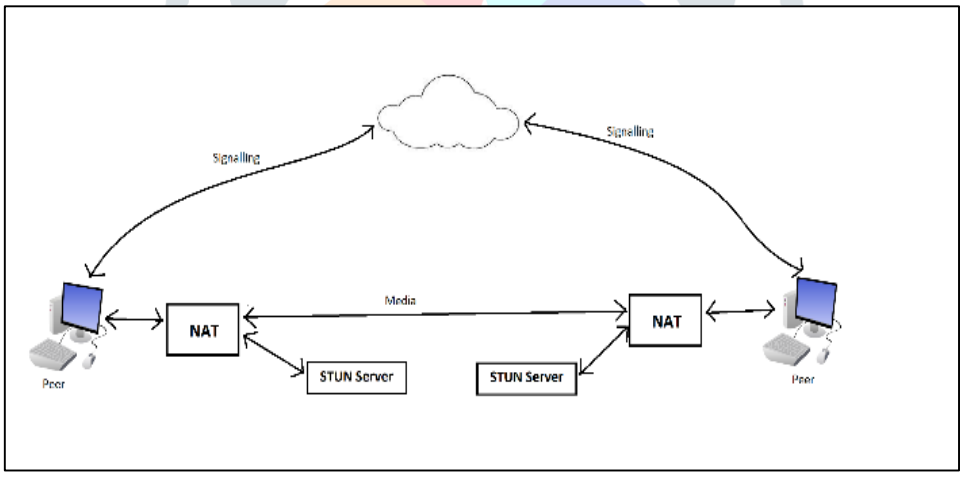

Figure 3-STUN Server in action [15]

(II) TURN

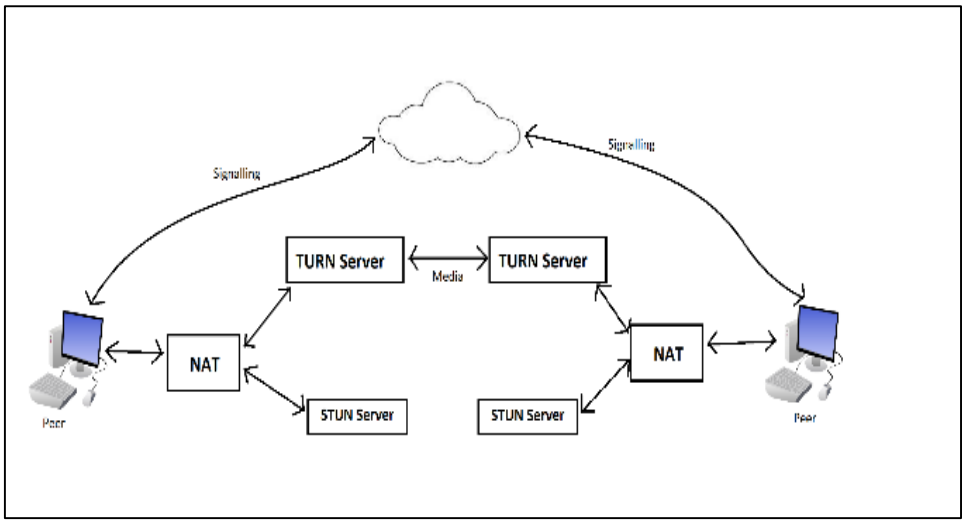

Figure 4-TURN Server in action [15]

STUN may fail especially if there is presence of restrictive NAT, proxy or firewall. So, to work with this kind of environment there is TURN server. It provides cloud fallback when there is peer-to-peer link is impossible. It asks for relay in cloud for public IP address and since public IP is in the cloud means anybody can contact it, which means communication always sets up and it works in every case where STUN was failing. But the disadvantage is that data is relayed through the servers and it uses server bandwidth. So, there is operational cost for it.

(III) ICE

Now, STUN is simple and cheap. But it may not work every time and on the other hand we have TURN which works every time, but it is very expensive. So, how to decide which technology and servers to choose. For that we have Interactive Connectivity Establishment (ICE). It tries to find best path possible for each call and using ICE results says that in 86% cases communication happens through STUN and in remaining cases there might be need of TURN [15].

3.3. Security [8] [9] [15]

When it comes to real-time communication, there exists several ways with which RTC applications or plugins may compromise security:

- Unencrypted audio, video or data might be intercepted between browser or between browser and server.
- Application might record or share user's audio, video or data without user's permission.
- Malware or virus might be installed along with RTC applications or plugins.

WebRTC includes some features which removes security issues:

- WebRTC has requirement to use https over http.
- WebRTC uses secure protocols such as Datagram Transport Layer Security (DTLS) protocol for data security and Secure Real-time Transport Protocol (SRTP) for audio and video.
- WebRTC is not any kind of plugin and it runs under browser's sandbox and gets updates with browser, which means no external malware or virus get installed.
- Camera and mic access permission are also asked to user explicitly and whenever camera and mic access permission is granted by users it is clearly shown by User Interface.

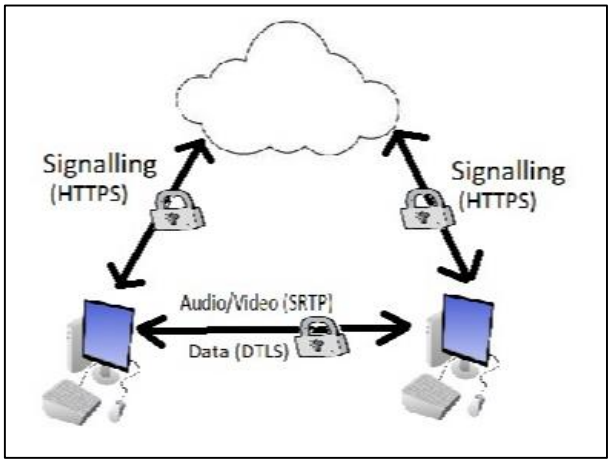

Figure 5-Security in WebRTC peer-to-peer [9] [15]

3.4 Multiple-Peer Communication

As there might be need of more than two devices to communicate. For ex., Conference call, group video chat or broadcasting, in these cases there might be need for different architectures to implement WebRTC application. So, for above purpose various methodologies and network topologies can be used:

3.4.1 Mesh Topology

Mesh topology is something with which multiple-peer communication can be implemented. But there are some of the drawbacks which can be occur:

- When we start adding more peers, things get bit complicated, as every peer must connect with every other peer.
- If there is data communication happens then every peer must send and copy data. So, it has a corresponding CPU and bandwidth cost.

There can be limited number of peers connected. For audio communication, number of peers can be high and for video communication, this number are less, especially if one of the devices is mobile device.

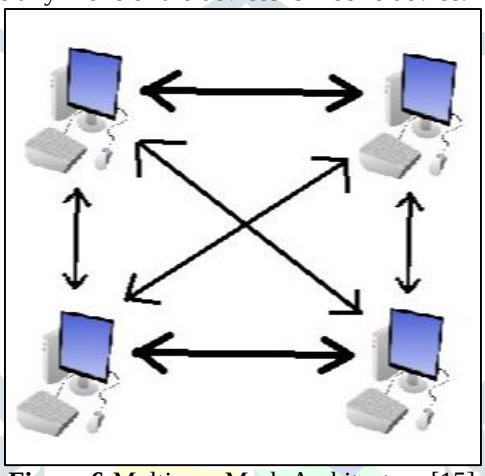

Figure 6-Multipeer Mesh Architecture [15]

3.4.2 Star Topology

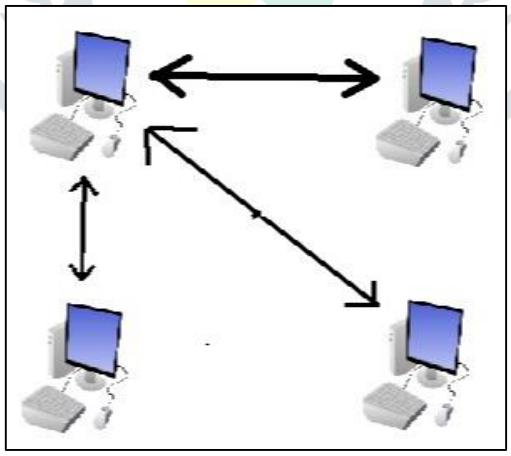

Figure 7-Multipeer Star architecture [15]

To deal with multiple peer, another architecture that can be propose is star topology. We can pick one of the devices which is called as focus for the call. It is responsible for receiving the data and sending copy of data to each other endpoints. But when it terms to be streaming high definition video data. Job of focus becomes very difficult.

3.4.3 Multipoint Control Unit (MCU)

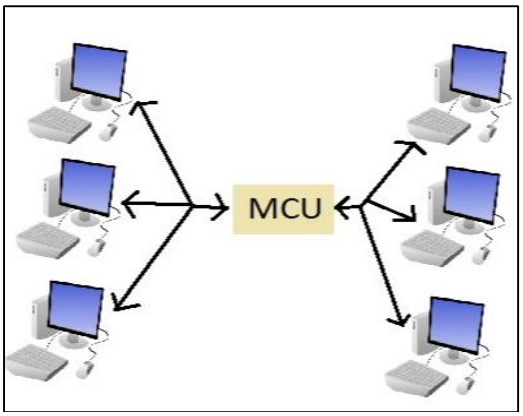

Figure 8-Multipeer Mesh Architecture [15]

Most robust architecture for conferencing is by using MCU or multipoint control unit. This is the server which is custom made for relaying large amount audio and video and it can do selective stream forwarding, mix audio or video data, recording and if one peer drops out, then it doesn't interrupt whole conference.

IV. CONCLUSION

In this review paper we have represented introduction and overview of WebRTC, how WebRTC can be used for effective and efficient browser-to-browser communication of audio video or data, we also briefed about three WebRTC APIs: getUserMedia(), RTCPeerConnection, RTCDataChannel with signaling mechanism and also shown various architectures through various network topologies using which it is easy to implement effective Multiple peers conferencing applications. Here, we also discussed about some of the security methods which are followed by WebRTC. Still this technology is actively developing and getting adapted by some of the big tech giants to serve their communication purpose.

REFERENCES

[1] J.K. Nurminen, J.R. Meyn, E. Jalonen, Y. Raivio, R.G. Marrero, 2013, *"P2P Media Streaming with HTML5 and WebRTC"*, IEEE, Vol.13, pp. 63-64.

[2] B. Sredojev, D. Samardzija, D. Posarac, 2015, *"WebRTC technology overview and signaling solution design implementation"*, ACM**,** pp. 1006-1009.

[3] H. Patil, A. Buchade, 2014, *"Review and Study of Real Time Video Collaboration FrameWork WEBRTC"*, International Journel of Computer Science & Information Technology, Vol.5, Issue.1, pp. 108-111.

[4] Zepeng zhang, Xiangming Wen and Wei Zheng, 2009, "NAT Traversal mechanism for peer-to-peer networks", IEEE, Vol. 09, pp.129-132.

[5] Samuel Ouya, Cheikhane Seyed, Ahmath Bamba Mbacke, 2015, Gervais Mendy,Ibrahima Niang, "WebRTC platform proposition as a support to the educational system of universitites in a limited internet connection context", IEEE, Vol. 15, pp.47-52.

[6] Natkal Moaid Edan,Ali Al-Sherbaz, 2017, "Design and evaluation of Browser-to-Browser video conferencing in WebRTC", IEEE, Vol. 17, pp.75-78.

[7] Alfonso Sandoval Rosas, Jose Luis Alejos Martinez, 2015, "Videoconference system based on WebRTC with access to the PSTN", Science Direct, pp.105-121.

[8] Victoria Beltran, Emmanuel Bertin, Noel Crespi, 2014, "User Identity of WebRTC Services", IEEE, Vol. 14, pp.18-25.

[9] Richard L. Barnes, Martin Thomson, 2014, "Browser-to-Browser Security Assurance for WebRTC", IEEE,Vol. 14, pp.11-17.

[10] Xiaoqun Yuan, Hao Yin, Xuening Liu, Changlai Du, Geyong Min, 2010, "Server Placement for peer-to-peer Live Streaming systems", IEEE, Vol. 10, pp.349-355.

[11] Quanfeng Duan, Zhenghe Liang, 2016, "File Sharing Strategy based on WebRTC", IEEE, Vol. 16, pp.3-6.

[12] Junnosuke Kuroda, Yasuichi Nakayama, 2013, "STUN-based connection sequence through symmetric NATs for TCP connection", IEEE, pp.51-55.

[13] Tran Thi Thu ha, Jaehyung Park, Yonggwan Won, Jinual Kim, 2014, "Combining STUN protocol and UDP hole punching technique for peer-to-peer communication across network address Translation", IEEE, pp.500-504.

[14] J. Uberti, P. Thatcher, (2011), *"WebRTC"*, Retrieved from https://webrtc.org/

[15] S. Dutton, (2014, February 21), *"Getting Started with WebRTC"*, Retrieved from https://www.html5rocks.com/en/tutorials/webrtc/basics/

[16] Mozilla Developer Network, *"WebRTC API"*, Retrieved March 20, 2019 from https://developer.mozilla.org/en-US/docs/Web/API/WebRTC_API

[17] Fullstack Academy, (2017, May 05), "WebRTC Tutorial – How does WebRTC work?*"* [video file], Retrieved from https://www.youtube.com/watch?v=2Z2PDsqgJP8&t=341s