Abacus (Mathematics Science Series) Vol. 44, No 1, Aug. 2019

ON THE DESIGN AND IMPLEMENTATION OF PEER – TO PEER COMMUNICATION USING WebRTC

By

Tiamiyu O.A.[1] & Garuba A. O

Department of Telecommunication Science, University of Ilorin, Ilorin, PMB 1515, Nigeria Anasofy Communication Gadget, Ilorin, Nigeria

ozutiams@yahoo.com

Abstract:

The advancement in technology over the last two decades has made life more comfortable. These advancements has enabled users to communicate using web browsers instead of standalone applications. While these in turn increase user mobility, they also save cost. The introduction of WebRTC that supports communication between web browsers in a peer-to-peer fashion is eliminating some of the challenges of macromedias Flash communication server, some of which are issues related to traversing firewalls, initiating a handshake and lower latency for video and audio over the web. This study intends to build a web application using WebRTC technology in order to make communication among peer easier than before.

Keywords*: WebRTC, Flash communication server, VoIP, P2P communication,*

1. Introduction

Communication is essential for maintaining a good relationship and performing day to day activities properly. Without communication, it may be impossible for most inventions as we see of today. The web is since its incarnation in 1996 formed by classic client/server architectures using the HTTP (hypertext transfer protocol) protocol [1]. There are several used cases, where a peer-to-peer (P2P) approach is preferable, e.g. for video communication or secure file transfer between two parties. Additionally the reliance on servers that users have no control over poses a great security and privacy risk for sensitive data. A set of new web technologies is currently being developed to enable a real browser-to-browser communication channel [2]. The Web Real-Time Communications (WebRTC) standard defines these technology consists of an API defined by the W3C and a set of underlying protocols defined by the IETF Rtcweb WorkingGroup. This chapter reviews the different applications of real-time communications as researched by different researchers.

The comfort of communicating using web browser instead of a standalone application has save cost and maintenance issue [3]. WebRTC integrates real-time communication over web. WebRTC is a free and open source application whose inception can be traced back to the year 2010 when Google, Apple, Microsoft, Mozilla, Ericsson, started building a real time communication platform that would work across the browsers without plug-ins [4].

WebRTC helps in enabling any Web server to convey a unique real-time communications experience, with ease and consistency, without relying on service providers or other services. Users participating in WebRTC will experience communication, as delivered by any website without downloading, registering or incurring any cost. The benefits are as follow:

1**. Platform and device independence**: WebRTC-enabled browser on any operating system that is having a web services application can have the browser creating a real-time voice or video connection to another WebRTC device or media server. The browser operating system is not relevant. This is achieved by implementing standard APIs from the W3C and protocols from the IETF. HTML5 code that can work on desktop and mobile devices can be written by Developers [5].

2. **WebRTC implementation is free:** WebRTC was introduced by Google in 2011. It is an open-source application programming interface (API). Google's aim for WebRTC is to deliver a standard-based, real-time media engine that will not only be free but also resident in all available browsers [6].

3. **Secure voice and video:** WebRTC always has voice and video encryption. The Secure real-time protocol (SRTP) is used for encryption and authentication of both voice and video as it is beneficial, especially over WiFi networks. It prevents eavesdropping and recording of the voice and video.

4. **Reliable session establishment:** This is supported by WebRTC. Moreover, it is true for Network Address Translators (NAT). It hinders and may block other communications and collaboration protocols. The reliable operation avoids server-relayed media, thus, reducing latency and server loads while increasing quality [3].

5. **Interoperability with VoIP and video:** WebRTC is highly valued for its capacity for interoperability with existing voice and video systems. For example, devices using SIP, XMPP, Jingle, and PSTN. Only hindrance to its global interoperability will be the upgrades required in existing devices. Nevertheless, the solution to interoperability can be gateways.

The most general WebRTC architectural model is shown in Figure 1.

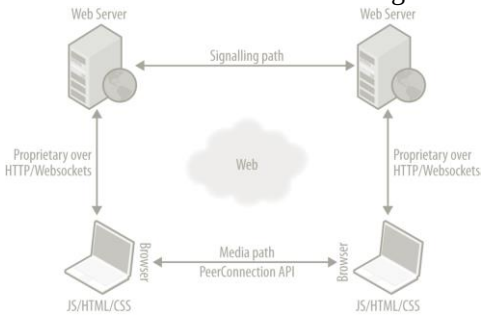

Figure 1: WebRTC Architecture [1]

In the model, both browsers are running a web application that was downloaded from a different Web Server. *A Peer Connection* configures the media path to flow directly between browsers without any intervening servers. Signaling goes over WebSockets or hypertext transfer protocol (HTTP), via Web Servers that can modify, translate or manage signals as required. It is noted that the signaling between browser and server is not standardized in WebRTC since it is considered to be part of the application. The two web servers can communicate using a standard signaling protocol such as Jingle or SIP. alternatively, a proprietary signaling protocol can be used. The most common WebRTC scenario could be the one where both browsers are running the same web application, downloaded from the same source. In such case, the WebRTC Trapezoid model becomes a Triangle [6].

WebRTC is a platform and standards effort to put real-time communications capabilities into all browsers and make these capabilities accessible to web developers via standard HTML5 (hypertext markup language) tags and JavaScript APIs (Application Program interface). For example, consider functionality similar to that offered by Skype or Whatsapp but without installing any software or plug-ins [3]. With the rapidly development of internet, more communication traffic is moving to web from the traditional telephony network. In addition, in the recent decade, VoIP network services are growing to the peak of the market capacity.

Solution to integrate WebRTC and existing VoIP network is the right approach considering the trend of the Internet communication requirement.

Prior to WEBRTC, macromedias Flash communication server (FCS) has been used for communication over IP network. FCS was acquired by adobe and had its name changed to Flash Media Server which is now popularly called Adobe Media Server. FCS allowed developers to capture webcam video and audio stream right from browser to the server and then to multiple viewers anywhere in the world [3].

WebRTC is a set of JavaScript Application Programming Interfaces (APIs) which enable web developers to develop Real-time Communication (RTC) features into their web-based application without bother of any plugins [1]. WebRTC comes as a solution for P2P file sharing, video streaming and audio calls by soliciting real-time communication solutions into the end user's Web browser. There are various applications currently in market like Skype, Google Hangout, FaceTime, and Whatsapp. However, these applications use client-server architecture [5].

P2P architecture is better as it is scalable and reliable than client-server architecture. In P2P architecture, a single node failure will not affect the whole system. WebRTC system consists of web servers, browsers with different operating system, workstations, tablets, mobile phone. WebRTC can interoperate with Session Initiation Protocol (SIP), and Public Switched Telephone Network (PSTN). Due to its interoperability with VoIP and other video communication system like SIP, Jingle, and PSTN; WebRTC is the best options in implementing P2P audio and video calling application [5].

2. Literature Review

Author in [7] presented an overview of real-time communication on the web by explaining its architecture and protocols. He clearly explained the relationship between WebRTC API and the network protocol suite that are bringing real time communication capabilities to web browsers. He further summarized the different components involved in the implementation of real-time communication, such as data framing, securing data, transporting data, managing connections and security considerations.

Implementing direct P2P connection often involves packet traversing networking equipment such as firewalls and other NAT devices. In the research by [4], an approach called Interactive Connectivity Establishment (ICE), which utilizes STUN (Session Traversal utilities for NAT) and TURN (Traversal Using Relay NAT) protocols for implementation was utilized. In their work, the different configuration procedures for implementing P2P communication with STUN and TURN servers to allow seamless communication between clients were explained.

Several other applications have been developed to meet various demands of different audiences. Author in [8] designed a web application that allowed for the integration of WebRTC and Session Initiation Protocol (SIP). The application allowed users to make VoIP calls using a web browser and in some cases, depending on the infrastructure, to even call a traditional landline and PSTN users.

Privacy is a great deal in any communication because information exchanged may be of sensitive in nature, thus there is need to protect the flow of information. WebRTC can be designed to include a series of security considerations, as done by author in [8]. The author defined a security threat model that can be used in WebRTC. He further went ahead to survey security threats for the implemented model. Ownership of personal data in web applications is a matter of ongoing passionate discussion. The main problem is that data resides on the providers' servers. A P2P architecture has the potential to mitigate the impacts of storing data on foreign servers since it can

be distributed and encrypted. The author investigated the possibilities of a censorship-resistant P2P collaboration architecture, but without focusing on web technologies.

A way to distribute the load and stream video content between browsers using WebRTC, thus reducing the bandwidth cost of content providers was examined by [1]. The author used a BitTorrent-like architecture involving a tracking server to discover content. However, most current implementations and demos leveraging WebRTC are currently focusing on audio/video communication using SIP, like sipML5.

In order to provide more secured means of communicating, an application called SaltyRTC which utilizes WebRTC technology and Network and Cryptography Library (NaCL) as a security mechanism was developed by [5]. SaltyRTC uses a box model of NaCL to encrypt and authenticate messages that is been exchanged between communicating parties. It is designed to provide high-speed and high-security, and it is easy to use. The fundamental operation behind NaCL is the use of public key authenticated encryption that ensures confidentiality, integrity and authentication of a message between sender and receiver.

The existing and common application that support real-time communication are numerous, some of those applications are Skype, Google hangout and Whatsapp among others. The weakness of those applications is that they have to be downloaded before using them. Most of this application are based on server/client architecture, which implies that when the server is down, client would not be able to communicate. Another weakness of the existing application is security issue. Information of the client is stored in database of the sever that may not be secured enough. The new WebRTC technology have come to eradicate or reduce the cost of downloading application as well as solve the security issue because all the content or chat of the client would not be saved or stored in any database.

3. METHODOLOGY

3.1. DESIGN PROCEDURE

This project was carried out using Visual Studio Code Editor. The Visual Studio Code Editor was installed on a Windows OS-based PC and the requirements for processing speed and memory are 2GHz and 1GB of RAM respectively.

The system architecture is based on the core concept of JavaScript (Express NodeJS Server, Peer JS Server and Peer JS Client) and real-time supported Network Address Translator (NAT).

WebRTC application is based on Secure Hyper Text Transmission Protocol (HTTPs) which provides more security than the widely used HTTP. HTTPs is a secure web based application protocol that resides in the application layer and is responsible for the proper transmission of information over the web (i.e. through web browsers).

Express NodeJS Server handled signaling between the two ends of the communication (peers) and also served the interface for relaying with the entire application. An html file named index.html was created to contain the homepage of peer2chat and styling was done using materialize CSS. The Express Node JS Server served this page to user once connection was established to the URL (peer2chat.herokuapp.com). The Express NodeJS Server also had peerJS Server script embedded within it. To install the dependencies for nodejs through a single NPM install command in the bootstrapserver's path, the NPM nodeJS Packet Manager was used. The PeerJS Server script embedded inside the Express NodeJS Server obtained information from the STUN server. The STUN Server used in this study was a Google public server that allows for transversal of communication media as a UDP stream through the firewall and NAT (NAT is a technique common with networking that allows for mapping of private address to a public address so as to allow for use of a private address on the Internet). STUN servers simply allowed for UDP media stream to transverse the NAT and firewall so that it was locally bounded to an

originating and receiving host despite the dynamism of the communication identifier used. The primary function of STUN servers is to check the IP address and port number of an incoming request from an application that was running behind NAT and set that address as response, thus enabling a peer to get its own publicly accessible address which was used to set up a direct link.

The PeerJS Client script allowed for the functionality of peer communication between two different hosts. Once communication was established between the two peers, PeerJS wrapped WebRTC technology and allowed the establishment of peer connections between two clients based on unique IDs. The PeerJS script called the functionality of a PeerServer that allowed a station to either host itself and form a peer or use one of the **free** cloud instance provided by peerjs.com. This script allowed for the parallel connectivity and sharing of data between peers without the need for a central server node that could be a form of relay agent.

All media and data streams exchanged between peers were encrypted using DTLS.

WebRTC supports different browsers which is essential since different devices might use different browsers to communicate. This cross-platform interoperability among web browsers is made possible because of a particular script function called within the appJS code. The code specified that the application should obtain information about the browser from the client and use the information to render the page appropriately. Apart from the functionality to get browser media from a user, Google Chrome and Mozilla browser were specified to be supported using the code in Figure 2:

```
navigator.getUserMedia = navigator.getUserMedia || navigator.webkitGetUserMedia || navigator.mozGetUserMedia;
```

Figure 2: Portion of the script that handled browser cross-platform interoperability

Communication commenced between two different users once a peer connection was established. The PeerConnection function allowed two users to communicate directly, browser to browser, and indcated an association with a remote peer, which implied that another instance of the same JavaScript application was running at the remote end. Although the Express NodeJS Server handled the signaling that led to the initiation of communication, however, once a peer connection was established, media streams (locally associated with ad hoc defined MediaStream objects) was sent directly to the remote browser without the need for the server. The PeerConnection mechanism used the STUN server to let UDP-based mediastreams traverse NAT boxes and firewalls.

3.2. IMPLEMENTATION AND TESTING

3.2.1. ACCESSING THE WebRTC APPLICATION ENVIRONMENT

The domain peer2chat.herokuapp.com was hosted on the herokuapp and thus it was accessed via the Internet.

To initiate conversation with a remote user, the user must join via web (peer2chat.herokuapp.com) using, preferably, Google Chrome or Mozilla Firefox (Figure 4.1), and signing in appropriately as requested using an active Google account and valid password (Figure 3).

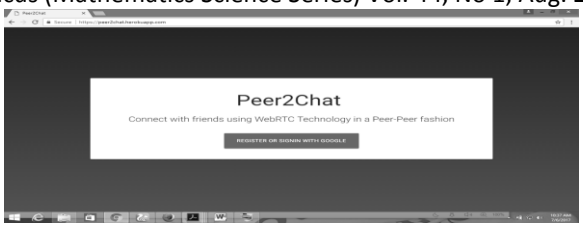

*Figure 3: Home (login in) interface of https://peer2chat.herokuapp.com*mmediately logged in, the user could invite others (via email or phone number) for calls or instant messaging.

Figure 4: Login into the application with Google account

After initiation of a call session with a remote party, a popup requesting access to the web camera and microphone of the device is displayed. The user could either allow or block the application access to the web camera and microphone device for the entire duration of the use of the application (Figure 5).

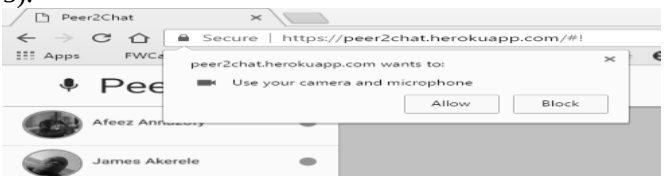

Figure 5: Requesting user consent for access to camera and microphone.

3.2.2. CALL INITIATION AND CALL REJECT

A session was being created using https, thus, the session cannot be easily highjacked or sniffed as https encrypts packet flow between communicating ends.

On clicking the invite friend tab, there would be a popup menu requesting for the name and email address of the intended party. Having provided correct details, the party would receive a link for joining the application (Figure 6).

For those successful requests and positive response from the recipient, user presence and all active friends in the address book were displayed at left corner of the application. Clicking on any of the active users, a popup requesting if a call should be placed was displayed (Figure 7).

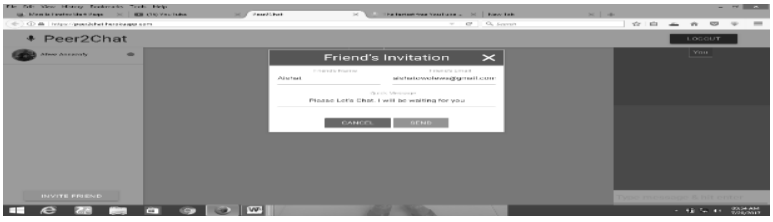

Figure 6: The menu for inviting friends.

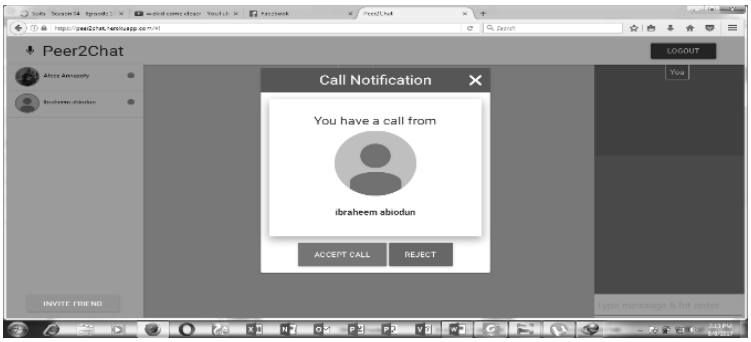

Figure 7: Pop-up confirming user intention to make call.

By clicking OK, a secure channel to handle signaling between the two parties was created and this channel carried the request for call access to the remote party. A pop-up was displayed on the remote party's device requesting the remote user to either accept or reject the call (Figure 8). Accepting the call, a notification was forwarded to the originating user and a full duplex connection handled by DTLS was created to handle media stream exchanges between the parties.

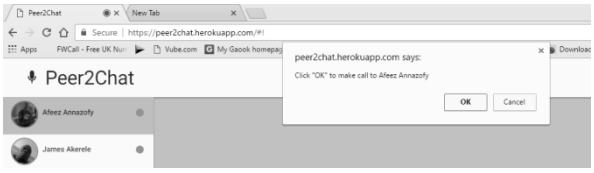

Figure 8: Pop-up requesting remote user to accept or reject incoming call

3.2.3. MESSAGING AND END CALL

Another functionality embedded in the application was the messaging functionality that allowed the connected parties to send instant messages. After initiation of call session, connected parties, instead of voice calls, could be sending instant messages using messaging bar at the right corner of the application (Figure 9).

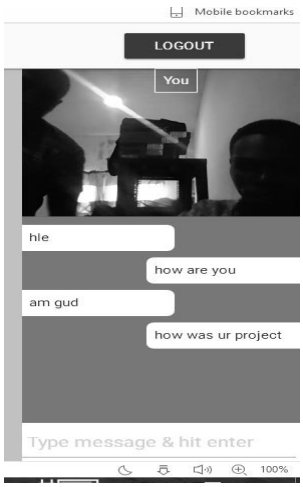

Figure 9: Instant Messaging among connected parties.

For every connection-oriented based protocols and applications, there is need to tear down the connection in order to free resources for other services. Peer2chat application made provision for this, allowing either of the calling party or called party to end an active call. On this peer2chat, clicking the red button named EndCall that was placed at the bottom center of the video display put an end to the communication session (Figure 10).

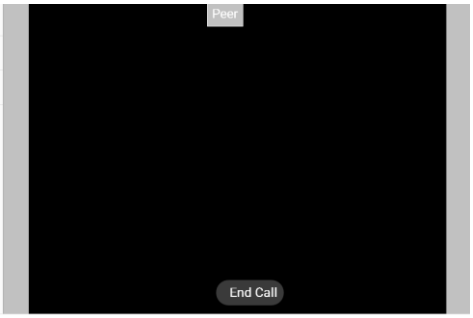

Figure 10. Ending call on peer2chat application

3.2.4. LOGGING OUT OF THE APPLICATION

Every web-based application that is associated with the use of email account is expected to give a user the option to log out. On the peer2chat application, a button named LOGOUT that was positioned at the right-top corner could be clicked at any point in time to logout from the application (Figure 11). Clicking on the button, a pop-up showing that sign out was successful was displayed and the application returned to the home screen.

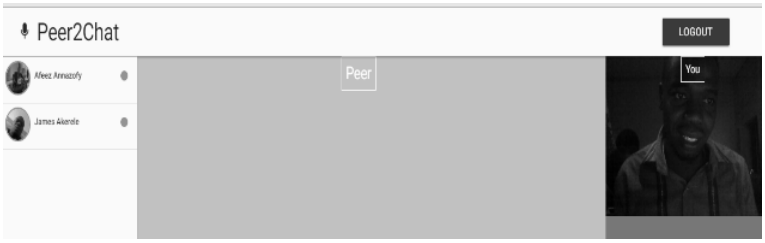

Figure 11: Logging out of the application

3.3. Results Analysis

WebRTC utilizes media transport and RTP in the communication process between two parties [8]. The Real-time Transport Protocol components such as payload formats, RTP sessions and header extensions were used for improving data transport robustness. RTP made encryption mandatory for WebRTC implementation and SRTP was the protocol used for transporting media such as audio and video.

Another protocol that is used in WebRTC implementation was SCTP. It was mainly used for data channel, precisely, for exchanging generic data from peer to peer.

Since WebRTC handles sensitive data transfer such as voice or video, security mechanism was put in place to secure vital or confidential information that were being transmitted or sent over web application. WebRTC security measures were in three categories: Client, Server, and Network Traffic.

i. **Server Security**: Since WebRTC was designed as a P2P communication interface, the roles performed by the server were implemented using pre-WebRTC software components, such as STUN and IdP (Identity Provider), as done also by [1].

ii. **Client – Access Control**: Since WebRTC was primarily designed to facilitate P2P audio and video chats, thus, the facilitating browser should be able to access the microphone and video camera. As such, the requirement for the user's consent was taken into consideration (Figure 5).

iii. **Network Traffic – Media Path Encryption:** Considering WebRTC specification, a secure encrypted version of the RTP, SRTP was used as did author in [8]. To achieve this, there was a distinction between the signaling plane (HTTP, SIP) and the media plane (RTP). Though each of these could be secured independently, yet it was difficult. However, securing each plane independently did not guarantee that the signaling user was the same as the media messaging user. As a result, DTLS-SRTP (Datagram Transport Real-time protocol Stream Real-time protocol) was used for both channels as enforced in WebRTC specifications. This made establishment of the media stream without the need to reveal the SDP (Session Descriptor Protocol) in the message exchange. Author in [1] did the same. Furthermore, DTLS-SRTP protocol is a variation of TLS, commonly used in an HTTP based session to achieve secure sessions between a client and a server (HTTPS). Since TLS operations were considered to be expensive, SRTP was used as a lightweight encryption model for communications. DTLS-SRTP utilized DTLS to bootstrap the SRTP key exchange over a higher encryption level channel to prevent Man in The Middle (MiTM) attacks [1].

Another point considered was how the browser would react to a proxy that tampers with encrypted certificates. Though there is normally a clear warning in each browser that the channel is compromised when using HTTPS applications, however, in WebRTC this choice was not given, and as such conversations were rejected by the browser between the signaling plane (HTTP, SIP) and the media plane (RTP).

Identity Provider (IdP) handled the third security issue of end-to-end authentication. IdP do normally provides identities to the web browsers whenever required via process known as Identity Generation. The Id generated was sent to the recipient browser prior to the communication. The recipient browser verified the Id from IdP via a process known as Identity Verification. Thereafter, a communication session was initiated for direct exchange of media. In Figure 12 is shown WebRTC security architecture.

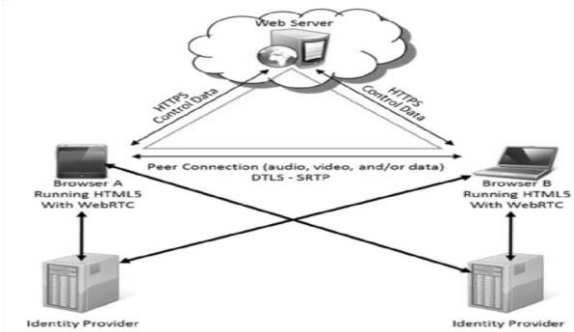

Figure 12: WebRTC security Architecture [9]

4. CONCLUSION

The main objective of this project was achieved. The peer2chat application was able to make a video call, send message, which are termed real-time communication. The application allowed multiple numbers of users to log in, provided there was Internet connection. Peer2chat application allowed to initiate a user peer, invite friend, make and reject calls and end call session. Peer2chat application reduced the stress of downloading a real-time communication application like Skype Google hangout among others. It also provided security measure which was a great issue in a server based application, and it did not save user information any form. Once session in Peer2chat was terminated, all the related information was deleted. Thus, Peer2chat provided web real-time secure communication among peer at no additional cost.

4.1. RECCOMENDATION

Further study could be to integrate Peer2chat to traditional telephone system using the SIP gateway. This would allow user to call mobile number as well, though at extra cost, i.e. mobile network operator's charges would apply.

REFERENCE

Ben F. et al., "The Security of WebRTC," 26 July 2015.

M. J. Werner, ""WebRTC Security in the context of a DHT implementation," ," 2013.

Kwok-Fai Ng, Man -Yan Ching, Yang Liu, Tao Cai, Li Li, and Wu Chou, "A P2P-MCU Approach to Multi-Party Video Conference," *International Journal of Future Computer and Communication*, vol. vol 3, October 2014.

Pooja Birajdar, Sagar Soni, Nandkishor Surashe, Vishal Telsang, "Restriction System for Communication of," *International Journal of Computing and Technology*, vol. Volume 3, no. Issue 3, march 2016.

Lieven Desmet,Martin Johns, "Real-Time Communications security on the web ," may 2014.

Salvatore loreto, simon pietro Ramano, *Real Time communication with webRTC* , Simon St.Laurent and Allyson MacDonald, Ed. united state , united state of America : O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472., may 2014.

M. G. a. B. G. Luis López-Fernández, ", "Authentication, Authorization, and Accounting in WebRTC PaaS Infrastructures," ," *IEEE INTERNET COMPUTING*, vol. volume 1089, 2014.

Kedar G. Pathare1, Pushpanjali M. Chouragade2, "WebRTC Implementation and Architecture Analysis," *International Journal of Scientific & Engineering Research*, vol. volume 7, no. issue 2, February 2016.

E. Rescorla, ""WebRTC Security Architecture," ," *Internet-Draft – work in progress*, march 7 2015