Peer-to-Peer Video Call with WebRTC

Shaikh Aejaz[1] Dr. Sachin Santoshkumar Agrawal[2]
[1,2]Department of Computer Science Engineering
[1,2]Shri Shivaji College of Engineering and Tech, Akola, MS, India

Abstract— Throughout the last decade we have witnessed a great evolution of the World Wide Web. Web pages have transitioned from static information to fully fledged applications with great interactivity and functionality. Videoconference has also been affected by this trend. Multimedia communication applications started to pop up in our web browsers enabled by proprietary plug-ins such as Adobe Flash. While these products in many cases provided complete videoconferencing and collaborative experiences, they frequently relied on proprietary solutions and protocols imposed by the developer of the browser plug-in instead of well-known standards. Lately, effort has been put to provide web browsers and pages with more functionality and interactivity without the need of relying on proprietary plug-ins. In this context, WebRTC is being defined and developed to offer real-time peer-to-peer communications to the web taking advantage of HTML5 and existent real-time protocols and codecs instead of defining new ones. WebRTC is a joint effort by the WebRTC working group of the World Wide Web Consortium (W3C) and the rtcweb group from the Internet Engineering Task Force (IETF) where the first provide the HTML and JavaScript API and the latter defines the protocols and codecs to be used in the communication. This paper describes the WebRTC technology and implementation of WebRTC client, server and signaling. Main parts of the WebRTC API are described and explained. This will leverage the browser with ability to host P2P multimedia streaming applications in a straightforward manner without having to install plugins or download the application.

Keywords: WebRTC, Video Conferencing, IETF, W3C, Proprietary Plugin

I. Introduction

Communicating with audio and video is a fairly common task with a history of technologies and tools. For a good example of audio communication, just take a look at a cell phone carrier. Large phone companies have established large networks of audio communication technology to bring audio communication to millions of people across the globe. These networks are a great example when it comes to showing widespread audio communication at its finest.

Video communication is also becoming just as prevalent as audio communication. With technologies such as Apple's FaceTime, Skype video calling and Zoom video calls, speaking to someone over a video stream is a simple task for an everyday user. A wide range of techniques have been developed in these applications to ensure that the quality of the video is an excellent experience for the user. There have been engineering solutions to problems, such as losing data packets, recovering from disconnections, and reacting to changes in a user's network.

The aim of WebRTC is to bring this technology into the browser. Many of these solutions require users to install plugins or applications on their PCs and mobile devices. They also require developers to pay for licensing, creating a huge barrier and deterring new companies to join this space. With WebRTC, the focus is on enabling this technology for every browser user without the need for plugins or hefty technology license fees for developers. The idea is to be able to simply open up a website and connect with another user right then and there.

II. WebRTC Overview

WebRTC allows you to set up peer-to-peer connections to other web browsers quickly and easily. To build such an application from scratch, you would need a wealth of frameworks and libraries dealing with typical issues like data loss, connection dropping, and NAT traversal. With WebRTC API, all of this comes built-in into the browser out-of-the-box. This technology doesn't need any plugins or third-party software. It is open-sourced and its source code is freely available at http://www.webrtc.org/.

The WebRTC API includes media capture, encoding and decoding audio and video, transportation layer, and session management.

A. Camera and Microphone Capture

The first step to using any communication platform is to gain access to the camera and microphone on the device that the user is using. This means detecting the types of devices available, getting permission from the user to access them, and obtaining a stream of data from the device itself.

B. Encoding and Decoding Audio and Video

Unfortunately, even with the improvements made in network speed, sending a stream of audio and video data over the Internet is too much to handle. This is where encoding and decoding comes into play. This is the process of breaking down video frames or audio waves into smaller chunks and compressing them into a smaller size. The smaller size makes it faster to send them across a network and decompress them on the other side. The algorithm behind this technique is typically called a codec.

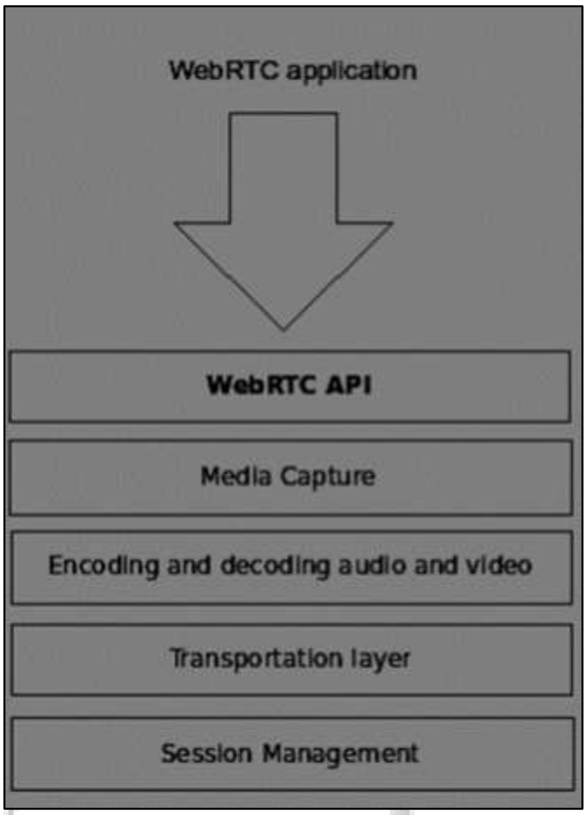

Fig. 1: WebRtc Layers

There are many codecs in use inside WebRTC. These include H.264, Opus, iSAC, and VP8. When two browsers speak to each other, they pick the most optimal supported codec between the two users. The browser vendors also meet regularly to decide which codecs should be supported in order for the technology to work. You can read more about the support for various codecs at https://developer.mozilla.org/en-US/docs/Web/Media/Formats/WebRTC_codecs.

C. *Transportation Layer*

This layer deals with packet loss, ordering of packets, and connecting to other users. The API makes it easy to deal with the fluctuations of a user's network and facilitates reacting to changes in connectivity.

The way WebRTC handles packet transport is very similar to how the browser handles other transport layers, such as AJAX or WebSocket. The browser gives an easy-to-access API with events that tells you when there are issues with the connection. In reality, the code to handle a simple WebRTC call could be thousands or tens of thousands of lines long. These can be used to handle all the different use cases, ranging from mobile devices, desktops, and more.

D. *Session Management*

Session management is the final piece of the WebRTC puzzle. This is simpler than managing network connectivity or dealing with codecs but still an important piece of the puzzle. This will deal with opening multiple connections in a browser, managing open connections, and organizing what goes to which person. This can most commonly be called signaling.

WebRTC today has many of the building blocks needed to build an extremely high-quality real-time communication experience. Google, Mozilla, Opera, and many others have invested a wealth of time and effort through some of their best video and audio engineers to bring this experience to the Web. WebRTC even has roots in the same technology used to bring Voice over Internet Protocol (VoIP) communication to users. It will change the future of how engineers think about building real-time communication applications.

III. Architecture

The overall WebRTC architecture has a great level of complexity.

If we look at the WebRTC architecture from the client-server side we can see that one of the most commonly used models is inspired by the SIP (Session Initiation Protocol) Trapezoid (RFC3261).

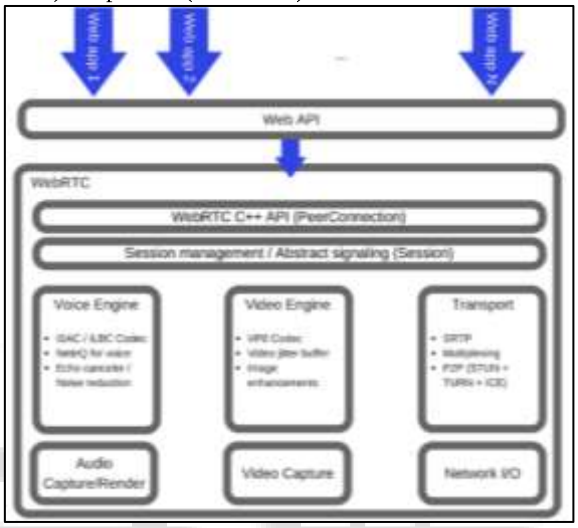

Figure 2: WebRtc Architecture

In the WebRTC Trapezoid model, both browsers are running a web application, which is downloaded from a different web server. Signaling messages are used to set up and terminate communications. They are transported by the HTTP or WebSocket protocol via web servers that can modify, translate, or manage them as needed. It is worth noting that the signaling between browser and server is not standardized in WebRTC, as it is considered to be part of the application. As to the data path, a PeerConnection allows media to flow directly between browsers without any intervening servers. The two web servers can communicate using a standard signaling protocol such as SIP or Jingle (XEP-0166). Otherwise, they can use a proprietary signaling protocol.

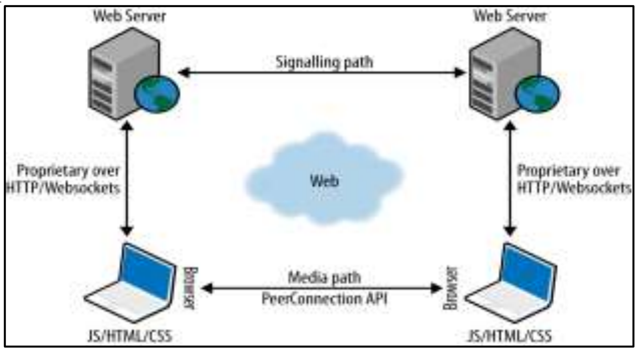

Fig. 3: WebRTC Trapezoid

The most common WebRTC scenario is likely to be the one where both browsers are running the same web application, downloaded from the same web page. It gives web developer more flexibility when managing user connections. In this case the Trapezoid becomes a Triangle.

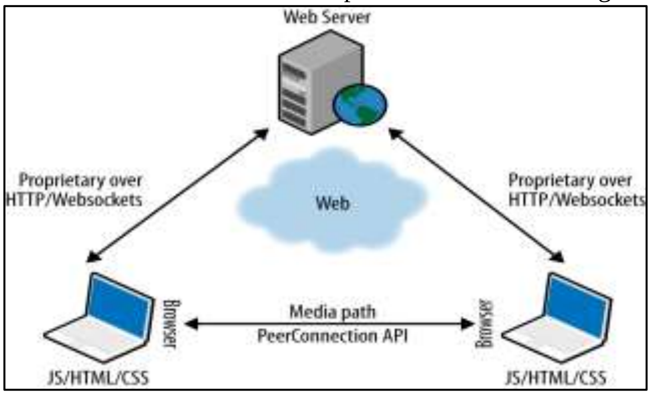

Fig. 4: WebRTC Triangle

A. *Signaling*

Signaling is a process of exchanging system control information like session control messages, media metadata, error messages and network data - information important to set up a call. This signaling process needs a two-way (bi-directional) messaging.

The general idea behind the design of WebRTC has been to fully specify how to control the media plane, while leaving the signaling plane as much as possible to the application layer. The rationale is that different applications may prefer to use different standardized signaling protocols (e.g., SIP or eXtensible Messaging and Presence Protocol [XMPP]) or even something custom.

Session description represents the most important information that needs to be exchanged. It specifies the transport and Interactive Connectivity Establishment [ICE] information, as well as the media type, format, and all associated media configuration parameters needed to establish the media path.

Since the original idea to exchange session description information in the form of Session Description Protocol (SDP) "blobs" presented several shortcomings, some of which turned out to be really hard to address, the IETF is now standardizing the JavaScript Session Establishment Protocol (JSEP). JSEP provides the interface needed by an application to deal with the negotiated local and remote session descriptions (with the negotiation carried out through whatever signaling mechanism might be desired), together with a standardized way of interacting with the ICE state machine.

The JSEP approach delegates entirely to the application the responsibility for driving the signaling state machine: the application must call the right APIs at the right times, and convert the session descriptions and related ICE information into the defined messages of its chosen signaling protocol, instead of simply forwarding to the remote side the messages emitted from the browser.

IV. WEBRTC API

The W3C WebRTC 1.0 API allows a JavaScript application to take advantage of the novel browser's real-time capabilities. The real-time browser function implemented in the browser core provides the functionality needed to establish the necessary audio, video, and data channels. All media and data streams are encrypted using DTLS.

The DTLS (Datagram Transport Layer Security) protocol (RFC6347) is designed to prevent eavesdropping, tampering, or message forgery to the datagram transport offered by the User Datagram Protocol (UDP). The DTLS protocol is based on the stream-oriented Transport Layer Security (TLS) protocol and is intended to provide similar security guarantees.

To ensure a baseline level of interoperability between different real-time browser function implementations, the IETF is working on selecting a minimum set of mandatory to support audio and video codecs. Opus (RFC6716) and G.711 have been selected as the mandatory to implement audio codecs. However, at the time of this writing, IETF has not yet reached a consensus on the mandatory to implement video codecs.

The API is being designed around three main concepts:

1) MediaStream: allows a web browser to access the camera and microphone;
2) PeerConnection: sets up audio or video calls;
3) DataChannel: allows browsers to send data through peer-to-peer connections;

It is important to know that WebRTC is not just a single API, but it is a collection of APIs and protocols defined by the various working groups such as W3C (World Wide Web Consortium) and IETF (Internet Engineering Task Force). Support for each of them develops in different browsers and operating systems.

1) *MediaStream*

A MediaStream is an abstract representation of an actual stream of data of audio and/or video. It serves as a handle for managing actions on the media stream, such as displaying the stream's content, recording it, or sending it to a remote peer. A MediaStream may be extended to represent a stream that either comes from (remote stream) or is sent to (local stream) a remote node.

A LocalMediaStream represents a media stream from a local media-capture device (e.g., webcam, microphone, etc.). To create and use a local stream, the web application must request access from the user through the getUserMedia() function. The application specifies the type of media—audio or video—to which it requires access. The devices selector in the browser interface serves as the mechanism for granting or denying access. Once the application is done, it may revoke its own access by calling the stop() function on the LocalMediaStream.

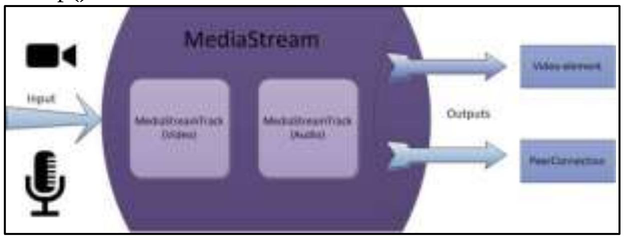

Figure 5: MediaStream

The two main components in the MediaStream API are the MediaStream interface and MediaStream Track. The MediaStream interface represents a stream of media content.

That media stream can have several parts. Those parts are called tracks and they have explicit type, such as audio and video. MediaStreamTrack represents a type of media that have been obtained from the input source. For example, if it is a microphone, we will get MediaStreamTrack (Audio).

B. *PeerConnection*

A PeerConnection allows two users to communicate directly, browser to browser. It then represents an association with a remote peer, which is usually another instance of the same JavaScript application running at the remote end. Communications are coordinated via a signaling channel provided by scripting code in the page via the web server, e.g., using XMLHttpRequest or WebSocket. Once a peer connection is established, media streams (locally associated with ad hoc defined MediaStream objects) can be sent directly to the remote browser.

The PeerConnection mechanism uses the ICE protocol together with the STUN and TURN servers to let UDP-based media streams traverse NAT boxes and firewalls. ICE allows the browsers to discover enough information about the topology of the network where they are deployed to find the best exploitable communication path. Using ICE also provides a security measure, as it prevents untrusted web pages and applications from sending data to hosts that are not expecting to receive them.

Each signaling message is fed into the receiving PeerConnection upon arrival. The APIs send signaling messages that most applications will treat as opaque blobs, but which must be transferred securely and efficiently to the other peer by the web application via the web server.

C. *DataChannel*

The DataChannel API is designed to provide a generic transport service allowing web browsers to exchange generic data in a bidirectional peer-to-peer fashion. Each DataChannel can be configured to provide:

- reliable or unreliable delivery of messages;
- in-order or out-of-order delivery of messages.

The reliable and in-order delivery is equivalent to TCP (Transmission Control Protocol). On the other hand, the unreliable and out-of-order delivery is equivalent to UDP (User Datagram Protocol).

The Internet addressing system is still using IPv4 (Internet Protocol version 4) and because of that, most of our devices are behind one or more layers of NAT (Network Address Translation). NAT is a mechanism of mapping private addresses to the public address changing address information within IP packets while it is in transit through the device for routing.

WebRTC technology developers can use ICE which will simplify complexity of the Internet addressing system. Developers need to pass the ICE server URLs to the peer connection. ICE will try to find the best path to connect two peers. ICE is using two types of servers, STUN (Session Traversal Utilities for NAT) and TURN (Traversal Using Relays around NAT). ICE is using a STUN server to find out what external address is assigned to a specific peer. If that fails, the traffic is routed via a TURN server. Every TURN server supports STUN, a TURN server is a STUN server with added relaying functionality built in.

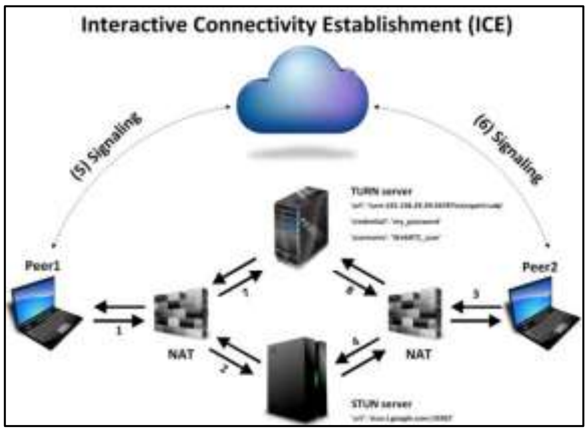

Fig. 6: Internet Connectivity Establishment (ICE)

The standardization work within the IETF has reached a general consensus on the usage of the Stream Control Transmission Protocol (SCTP) encapsulated in DTLS to handle nonmedia data types.

The encapsulation of SCTP over DTLS over UDP together with ICE provides a NAT traversal solution, as well as confidentiality, source authentication, and integrity protected transfers. Moreover, this solution allows the data transport to interwork smoothly with the parallel media transports, and both can potentially also share a single transport layer port number. SCTP has been chosen since it natively supports multiple streams with either reliable or partially reliable delivery modes. It provides the possibility of opening several independent streams within an SCTP association towards a peering SCTP endpoint. Each stream actually represents a unidirectional logical channel providing the notion of in-sequence delivery. A message sequence can be sent either ordered or unordered. The message delivery order is preserved only for all ordered messages sent on the same stream. However, the DataChannel API has been designed to be bidirectional, which means that each DataChannel is composed as a bundle of an incoming and an outgoing SCTP stream.

The DataChannel setup is carried out (i.e., the SCTP association is created) when the createDataChannel() function is called for the first time on an instantiated PeerConnection object. Each subsequent call to the createDataChannel() function just creates a new DataChannel within the existing SCTP association.

V. WEBRTC A SIMPLE CALL FLOW

The call flow we are going to build here will help us connect two users together located on remote distance. The goal is to enable the signaling mechanism with something that travels over a network. The flow will be straight forward and simple, supporting only the most basic WebRTC connections.

The diagram in the picture involves three different actors:

- A channel initiator, such as the peer that first takes the initiative of creating a dedicated communication channel with a remote party
- A signaling server, managing channel creation and acting as a message relaying node
- A channel joiner, for instance, a remote party joining an already existing channel

The idea is that the channel is created on demand by the server after receiving a specific request issued by the initiator. As soon as the second peer joins the channel, conversation can start. Message exchanging always happens through the server, which basically acts as a transparent relay node. When one of the peers decides to quit an ongoing conversation, it issues an ad hoc message (called leave) towards the server, before disconnecting. This message is dispatched by the server to the remote party, which also disconnects, after having sent an acknowledgment back to the server. The receipt of the acknowledgment eventually triggers the channel reset procedure on the server side, thus bringing the overall scenario back to its original configuration.

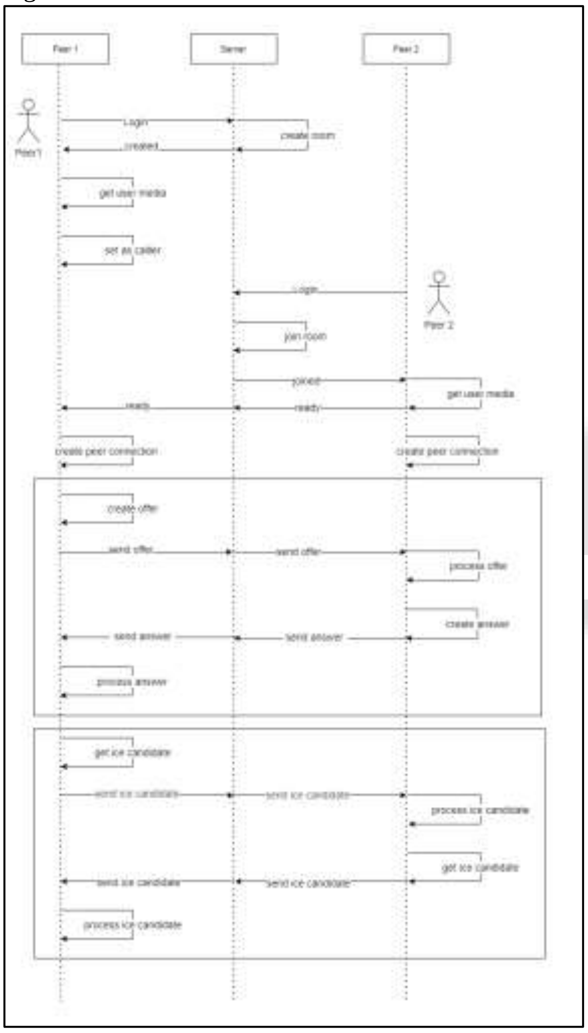

Fig. 7: WEB-RTC Simple Call Flow End-to-End

VI. Challenges

The implementation of a known technology in a new environment brings a host of challenges. Previous content provided us with enough background and knowledge to foresee challenges that are going to be solved before video conferencing through WebRTC becomes widely used in advanced applications that go beyond one on one conversation. These challenges rise from the web environment, where, ideally, different kinds of devices have to interact in an efficient way and from providing higher level services, needed in complex scenarios like education, corporate, etc.

1) Heterogeneous access networks and devices: The diversity of devices accessing the Web has increased considerably over the last decade. Where once the traditional personal computer was the only significant access point to the World Wide Web (WWW) there are now a number of devices with different capabilities and, in many cases, access networks. Such is the case of smartphones, which still have limited processing power compared to desktop computers and usually access the web via cellular networks.
2) Screen size: smaller screens cannot show the same amount of information as big ones. In video conferencing that means, for instance, sending a really high-quality video to a small screen device is inefficient, as users cannot tell the difference. A more complicated scenario is multi-conferencing where the screen size limits the count of participants that can be shown at the same time.
3) CPU: Video conferencing requires processing power to decode, encode and distribute video and audio in real time. This CPU stress depends on several factors such as the codecs used, the quality of both audio and video and video size. In mobile systems CPU power not only imposes a limitation in the things that can be done but also, stressing the device too much can lead to perceivable decrease in battery life.
4) Bandwidth availability and latency: As we mentioned above, the variety of devices goes in hand with an array of different access networks. For instance, we have typical wired Ethernet access from desktop computers and 4G networks for mobile devices. These differences have to be taken into account if we want to optimize the communication. For instance, 4G connections can vary the available bandwidth without any notice. That will result in an interruption in the conference if the system is not able to react accordingly.
5) Session recording: In some scenarios such as enterprise meetings and e-learning, recording the entire information generated in a videoconference session is an essential feature. While the current WebRTC definition is explicit when it comes to self-video recording it does not provide a great mechanism to gather the streams from all the participants and store them.
6) Gateways: The interoperability of one communication platform with another is always a challenge. For instance, communication between SIP Phones and traditional Public Switched Telephone Networks (PSTN) networks is achieved via gateways that translate signaling and media streams.

VII. Conclusion

WebRTC is a powerful technology that benefits all areas of communication accessibility and feature rich tool. This technology has a potential to improve enterprise communications. It can provide a lot of benefits for enterprises. WebRTC technology reduces the costs of using paid software to connect remote branches and real-time communication is supported without the need for additional third-party software or plugins. Secure communication with peers in within an enterprise and its remote branches is enabled by state-of-the-art encryption standards.

In this paper we described the WebRTC technology and implementation of WebRTC client, server and signaling. Main parts of the WebRTC API are described and explained. Signaling methods and protocols are not specified by the WebRTC standards, therefore in this study we designed a novel signaling mechanism. The corresponding message sequence chart of the WebRTC communication behavior describes a communication flow between peers and the server. The server application is considered as a WebSocket server. The client application demonstrates the use of the WebRTC API for achieving real-time communication. Benefits and challenges of the WebRTC technology are mentioned.

References

[1] R. Manson, “Getting Started with WebRTC, Explore WebRTC for real-time peer-to-peer communication,” in Birmingham, Packt Publishing, ISBN 978-1-78216-630-6, September 2013.
[2] Salvatore Loreto and Simon Pietro Romano “Real-Time Communication with WebRTC” O’Reilly Media, Inc. Publishing, ISBN: 978-1-449-37187-6, May 2014
[3] Dan Ristic, “Learning WebRTC-Develop interactive real-time communication applications with WebRTC” in Birmingham, Packt Publishing, ISBN 978-1-78398-366-7, June 2015
[4] IETF - WebSocket, https://tools.ietf.org/html/rfc6455
[5] IETF - SDP, https://tools.ietf.org/html/rfc4566
[6] Branislav Sredojev, Dragan Samardzija and Dragan Posarac “WebRTC technology overview and signaling solution design and implementation”, IEEE Electronic ISBN:978-9-5323-3082-3, 2015
[7] Pedro Rodríguez, Javier Cerviño, Irena Trajkovska and Joaquín Salvachúa “ADVANCED VIDEOCONFERENCING SERVICES BASED ON WEBRTC”, IEEE ISBN: 978-972-8939-72-4 , 2012