International Journal For Technological Research In Engineering
Volume 8, Issue 7, March-2021 *(Online): 2347 - 4718*

REAL-TIME COMMUNICATION AND MULTIPOINT VIDEO CONFERENCE USING WEBRTC AND MEDIA SERVER

Boobalan M

Department of Computer Science and Engineering
Sri Ramanujar Engineering College, Chennai-600127, India.

Abstract— Real-time communication (RTC) on the web is a new standard and industry-wide effort that enables users to access information in areas like chatting apps, social-media platforms, over-the-top media services, video conferencing apps and data streaming services. There are many proprietary protocols and codec(s) available that are not easily interoperable and scalable to implement a multipoint real-time video conference system. But with WebRTC (Web Real-Time Communication), you can add real-time communication capabilities to your application that works on top of an open standard. It supports video, voice, and generic data to be sent between peers, allowing developers to build powerful voice- and video-communication solutions. The technologies behind WebRTC (Web Real-Time Communication) are implemented as an open web standard and available as regular JavaScript APIs in all major browsers. This paper speaks about a multipoint real-time videoconferencing platform using WebRTC APIs. In this paper, I've proposed a web application for real-time communication using modern browser's WebRTC (Web Real-Time Communication) APIs that allows users to communicate with high speed data transmission over the communication channel using existing web technologies like HTML5, CSS3, JavaScript, WebRTC (Web Real-Time Communication) APIs and Coturn TURN server, NodeJs Signaling server and Kurento media server.

The outcome of this experiment shows that WebRTC (Web Real-Time Communication) is a capable solution for scalable multipoint real-time video conferencing and screen sharing app within a modern web browser.

Keywords— multipoint video conference, coturn, Kurento, webRTC, real-time, signaling server.

1. INTRODUCTION

Imagine a world where your phone, TV, and computer could communicate on a common platform. Imagine it was easy to add video chat and peer-to-peer data sharing to your web app. That's the vision of WebRTC. The WebRTC open standard enables users of these systems to view video content or record comments to stream it to achieve real-time communication between web browsers using its native APIs. One of the last major challenges for the web is to enable human communication through voice and video: real-time communication or RTC for short. RTC should be as natural in a web app as entering text in a text input. Without it, you're limited in your ability to innovate and develop new ways for people to interact. Historically, RTC has been corporate and complex, requiring expensive audio and video technologies to be licensed or developed in house. Integrating RTC technology with existing content, data, and services has been difficult and time-consuming, particularly on the web.

On the other hand WebRTC implemented open standards for real-time, plug-in free video, audio, and data communication. The need was real:

- Many web services used RTC, but needed downloads, native apps, or plug-ins. These included Skype, Facebook, and Hangouts.
- Downloading, installing, and updating plug-ins are complex, error prone, and annoying.
- Plug-ins are difficult to deploy, debug, troubleshoot, test, and maintain—and may require licensing and integration with complex, expensive technology. It's often difficult to persuade people to install plugins in the first place!

The guiding principles of the WebRTC project are that its APIs should be open source, free, standardized, built into web browsers, and more efficient than existing technologies.

In this project we are doing following things to make our app works:

- Get streaming audio, video, or other data.
- Get network information, such as IP addresses and ports, and exchange it with other WebRTC clients (known as peers) to enable connection, even though NATs and firewalls using TURN/STUN servers.
- Coordinate signaling communication to report errors and initiate or close sessions using WebSocket through our Signaling server.
- Exchange information about media and client capability, such as resolution and codec(s) using WebSocket through our Signaling server.
- Communicate streaming audio, video, or data through our Kurento media server.

To acquire and communicate streaming data, WebRTC implements the following APIs:

- MediaStream gets access to data streams, such as from the user's camera and microphone.
- RTCPeerConnection enables audio or video calling with facilities for encryption and bandwidth management.
- RTCDataChannel enables peer-to-peer communication of generic data.

2. RELATED WORK

Before the existence of WebRTC, there were already many

video conferencing systems available on the market. The most popular example is Skype. Microsoft is the company that owns Skype. Skype uses a proprietary protocol for the transmission of multimedia streams, plus it requires the installation of a mobile application or desktop to access services such as phone calls, messages, and video conferences. Still, it is not possible to integrate a phone call with an active video conference [16].
As mentioned Skype uses proprietary protocol and lacks the direct P2P ability, as well as a credible security feature found in WebRTC.
The WebRTC architecture provides end-to-end encrypted P2P communication with audio-visual content and data being transmitted directly. The implementation is realized to bypass intermediary hardware servers and eliminate security challenges like hijackers. This particular feature makes the difference between WebRTC and other RTCs such as Skype.
WebRTC also avoids critical challenges with plug-ins such as Flash, Silverlight, and Shockwave is the need for downloads each time a connection is to be established. Plug-ins can be problematic during execution; they increase bandwidth, latency, execution time, and speed [12].
The current version of the WebRTC API was designed only to support browser-to-browser communication. WebRTC for "Multi-browser" communication is not inherently recommended, especially for conference models that spread the media load over participating peers/browsers [5].
An overview of WebRTC video conferencing architecture using MCU is shown in [17]. However, this scenario does not discuss any kind of signaling while the proposed test was relying on using MCU that can be applied using a single connection. In addition, [17] ran an application of WebRTC video conferencing using the Licode-Erizo (MCU) over LAN (Local Area Network). Licode offers a client API with -Erizo that handles connections for virtual rooms and a server API for communication. But, without using the third party (LicodeErizo) it would not be possible to run this application. On the other hand, as illustrated in [18] using MCU is very expensive, and [19] mentioned that MCU is costly and it can be rented from service providers just during a conference, although some video conferencing CODECs are able to support a specific number of multipoint (e.g. up to 4 users). Adding to that, [18] emphasizes that MCU consumes a significant amount of bandwidth.
Based on the various articles of the related work as shown above. It can be comprehended that signaling between browser-to-browser and server is not standardized in WebRTC [20][21].

2.1 ARCHITECTURE OF WEBRTC

Figure 1. shows the architecture of WebRTC. As you see there are 3 main layers in WebRTC as following

- API for web developers.
- API for browser makers.
- Customization service layer for browser makers.

Also the architecture consists of voice engine, video engine, transport and communication tools. Each has its own functionalities. C++ APIs are exposed to access by browser and native frameworks and these low level APIs are not accessible from the web application layer for security and compatibility reasons. So, browser makers have to provide another way to developers to use it.
To do that browser makers provide standard Javascript APIs to access functionality of WebRTC[2][4].

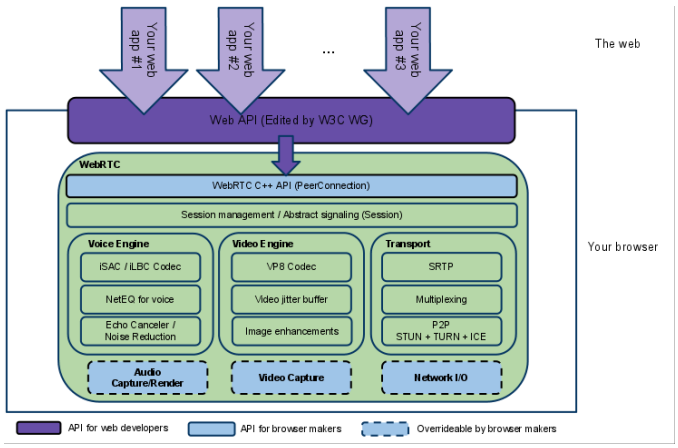

Fig. 1 Architecture of WebRTC

Voice and Video Engines - Enabling RTC requires that the browser be able to access the system hardware to capture both voice and video. Raw voice and video streams are not sufficient on their own. They have to be

- Processed for noise reduction and echo cancellation
- Automatically encoded with one of the optimized narrowband or wideband audio codecs
- Used with a special error-concealment algorithm to hide the negative effects of network jitter and packet loss.

Sender:

- Process the raw stream to enhance quality
- Synchronize and adjust the stream to match the continuously fluctuating bandwidth and latency between the clients.

Receiver:

- Decode the received stream in real-time.
- Adjust the decoded stream to network jitter and latency delays.

The fully featured audio and video engines of WebRTC take care of all the signal processing. While all of this processing is done directly by the browser, the web application receives the optimized media stream, which it can then forward to its peers using one of the JavaScript APIs!
VoiceEngine is a framework for the audio media chain, from sound card to the network.
VideoEngine is a framework for the video media chain, from camera to the network, and from network to the screen.

2.2 WEBRTC PROTOCOLS

Unlike all other browser communication which use Transmission Control Protocol (TCP), WebRTC transports its data over User Datagram Protocol (UDP).

TCP delivers a reliable, ordered stream of data. If an intermediate packet is lost, then TCP buffers all the packets

after it, waits for a retransmission, and then delivers the stream in order to the application.

UDP offers no promises on reliability or order of the data, and delivers each packet to the application the moment it arrives. In effect, it is a thin wrapper around the best-effort delivery model offered by the IP layer of our network stacks.

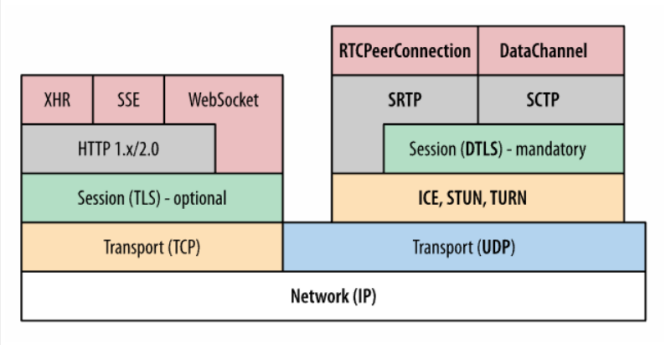

Fig. 2 WebRTC Network Protocol Stack

UDP is the foundation for real-time communication in the browser. In order to meet all the requirements of WebRTC, the browser needs a large supporting cast of protocols and services above it to traverse the many layers of NATs and firewalls, negotiate the parameters for each stream, provide encryption of user data, implement congestion and flow control, and more!

Real-time Transport Protocol (RTP) stack -

- ICE: Interactive Connectivity Establishment
- STUN: Session Traversal Utilities for Network Address Translation (NAT)
- TURN: Traversal Using Relays around NAT
- SDP: Session Description Protocol
- DTLS: Datagram Transport Layer Security
- SCTP: Stream Control Transport Protocol
- SRTP: Secure Real-Time Transport Protocol

ICE, STUN, and TURN are necessary to establish and maintain a peer-to-peer connection over UDP.

DTLS is used to secure all data transfers between peers; encryption is a mandatory feature of WebRTC.

SCTP and SRTP are the application protocols used to multiplex the different streams, provide congestion and flow control, and provide partially reliable delivery and other additional services on top of UDP.

Session Description Protocol (SDP) is a data format used to negotiate the parameters of the peer-to-peer connection. However, the SDP "offer" and "answer" are communicated out of band, which is why SDP is missing from the protocol diagram.

2.2.1 SECURITY

There are several ways a real-time communication app or plug-in might compromise security.
For example:

- Unencrypted media or data might be intercepted between browsers, or between a browser and a server.
- An app might record and distribute video or audio without the user knowing.
- Malware or viruses might be installed alongside an apparently innocuous plugin or app.

WebRTC has several features to avoid these problems:

- WebRTC implementations use secure protocols, such as DTLS and SRTP.
- Encryption is mandatory for all WebRTC components, including signaling mechanisms.
- WebRTC is not a plug-in. Its components run in the browser sandbox and not in a separate process. Components do not require separate installation and are updated whenever the browser is updated.
- Camera and microphone access must be granted explicitly and, when the camera or microphone are running, this is clearly shown by the user interface.

Figure 3. Shows the overall topology of WebRTC which explains call's flow.

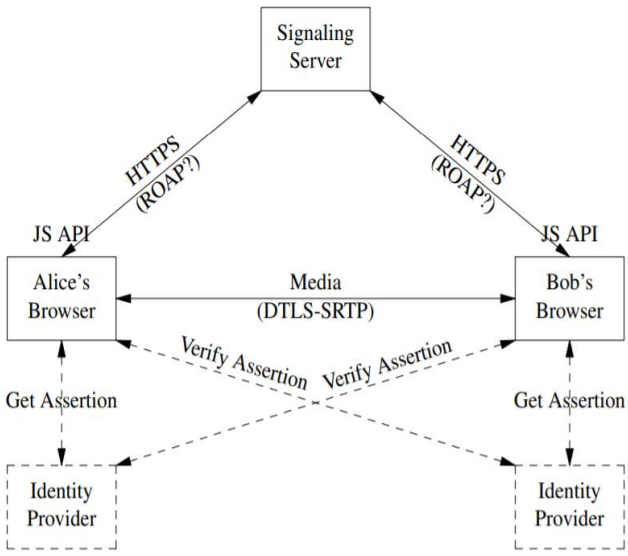

Fig. 3 Overall topology of WebRTC

- Bob knows Alice is calling [verified with IdP]
  - Browser can display trusted UI for Alice's identity
  - If in an address book, maybe name, picture, etc.
- If no IdP, Bob knows the signaling service claims Alice is calling.
- Alice knows Bob has answered
  - Verified with Bob's identity provider
- Alice and Bob know media is not flowing to innocent third parties (media consent)
- Alice and Bob know they have a secure call with each other
  - Security details displayed via trusted UI

3. MULTIPOINT VIDEO CONFERENCE

USING KURENTO MEDIA SERVER

WebRTC, as currently implemented, only supports one-to-one communication, but could be used in more complex network scenarios, such as with multiple peers each communicating with each other directly or through a Multipoint Control Unit (MCU), a server that can handle large numbers of participants and do selective stream forwarding, and mixing or recording of audio and video.

Why a WebRTC media server? - WebRTC is a set of protocols and APIs that provide web browsers and mobile applications with Real-Time Communications (RTC) capabilities over peer-to-peer connections. It was conceived to allow connecting browsers without intermediate helpers or services, but in practice this P2P model falls short when trying to create more complex applications. For this reason, in most cases a central media server is required.

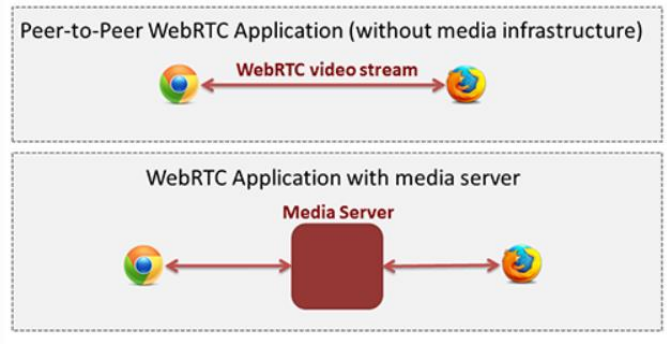

Fig. 4 Peer-to-peer WebRTC approach vs. WebRTC through a media server.

Kurento is a WebRTC media server and a set of client APIs making simple the development of advanced video applications for WWW and smartphone platforms. Kurento Media Server features include group communications, transcoding, recording, mixing, broadcasting and routing of audiovisual flows.

Fig. 5 Kurento Media Server capabilities

3.1 ARCHITECTURE

Figure 6. Shows the application architecture which consists of important components like coturn STUN/TURN server for NAT traversal, kurento media server for exchanging media, Signalling server for handshake connections.

In the real world, WebRTC needs those servers, however simple, so the following can happen:

- Users discover each other and exchange real-world details.
- WebRTC client apps (peers) exchange network information.
- Peers exchange data about media, such as video format and resolution.
- WebRTC client apps traverse NAT gateways and firewalls.

In other words, WebRTC needs four types of server-side functionality:

- User discovery and communication using kurento server
- Signaling
- NAT/firewall traversal
- Relay servers in case peer-to-peer communication fails using coturn TURN server.

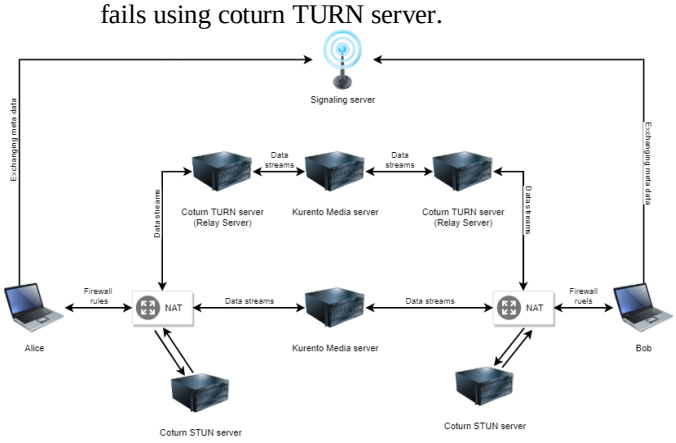

Fig. 6 Architecture diagram

3.2 MODULES

Part of this paper we will explore features including audio calling, video calling, text based chat, screen sharing, call recording, smart speech detection.

3.2.1 AUDIO CALL FEATURE

In this module we will be creating an audio calling facility via WebRTC audio streams. With this feature you will be able to make audio calls. And several users can do it simultaneously.

Our app allows users setting their specific session configuration before joining. Not only will you be able to check your audio and video devices before joining the session, but you will also be able to switch them between all the available ones.

Besides, you can capture your own avatar thumbnail by using the webcam and you can set your custom nickname. To join the session, we are going to do the following steps.

- Obtain an audio stream from a microphone.
- Create the RTCPeerConnection object and establish a connection.

Once the session is completed, we will be doing the following steps.

- We will send a “leave” message to the other users.
- We will close the RTCPeerConnection and destroy the connection locally.

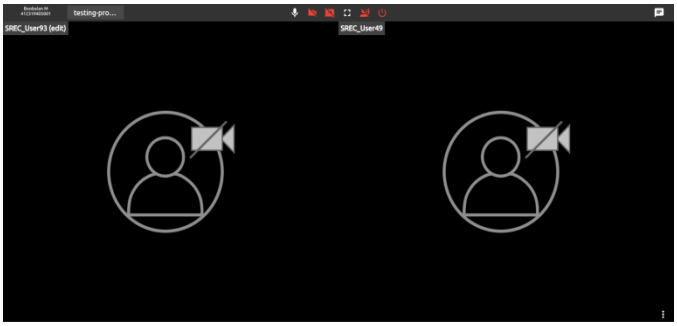

Fig. 7 Audio calling feature

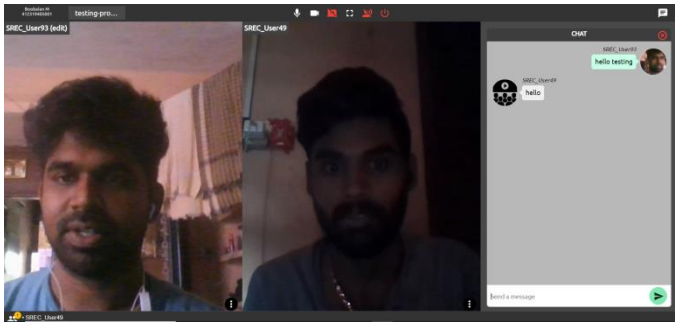

Fig. 8 Text based chat feature

3.2.2 VIDEO CALL FEATURE

In this module we will be creating a video calling facility. With this feature you can join into a multi-party video conference, displayed in a nice, intelligent layout. You will be able to zoom in and zoom out and full screen any video you want. An "active speaker" layout is provided to focus on the user currently speaking.

Take a look at the bottom footer of the session layout: you can check at a glance all the participants connected to the session. This is awesome information, especially when some user is publishing just audio and they do not appear in the video layout.

Our app allows users setting their specific session configuration before joining. Not only will you be able to check your audio and video devices before joining the session, but you will also be able to switch them between all the available ones.

Besides, you can capture your own avatar thumbnail by using the webcam and you can set your custom nickname.

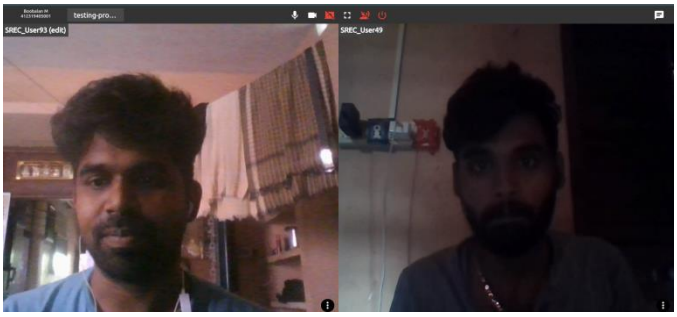

Fig. 8 Video calling feature

3.2.3 TEXT BASED CHAT FEATURE

In this module we will be creating text-based chat features. With this feature we will be including a nice chat already integrated in the intelligent layout. The chat provides you an alternative way to silently exchange messages with all your videoconference partners.

Everything works as you expect: messages are properly displayed with the author's name and avatar, links are automatically formatted to allow clicking on them, users will be subtly notified when a new message arrives, and mobile view is specifically adapted for ease of use.

A great and advanced chat out-of-the-box!

3.2.4 SCREEN SHARING FEATURE

In this module we will be creating a screen sharing feature. With this feature you will be able to share your screen and your webcam at the same time. And several users can do it simultaneously!

Thanks to our intelligent layout, your screen share video will take the lead and be shown at a larger size than the rest, and never being cropped so no information is lost.

Moreover, you can seamlessly switch your shared screen on the fly with the click of a button.

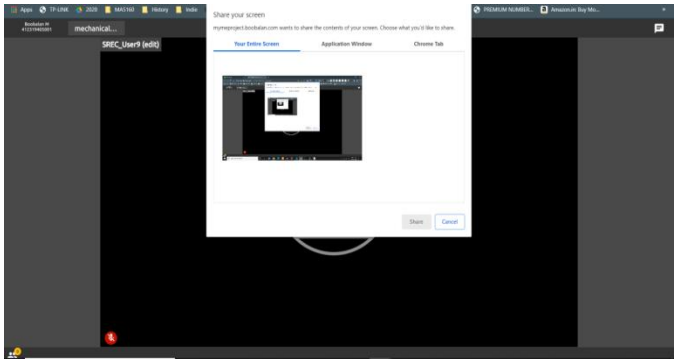

Fig. 9 Screen sharing

3.2.5 CALL RECORDING

In this module we will be implementing a recording feature. With this feature you will be able to record our call both audio and video at the same time.

Recording start and stop features will be provided to trigger recording action.

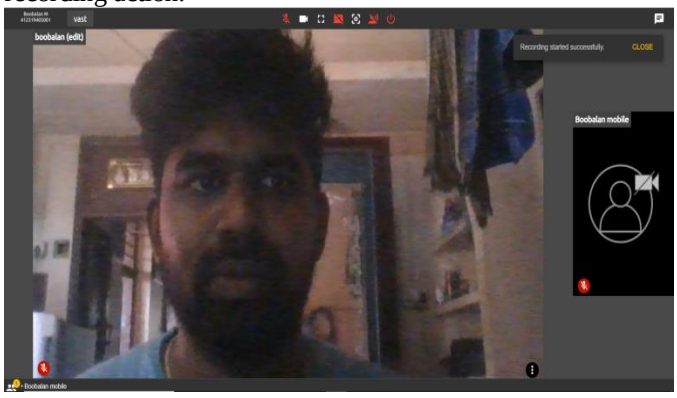

Fig. 10 Call recording feature

3.2.6 SMART SPEECH DETECTION

In this module we will be creating a smart layout feature using smart speech detection. With this feature you can join into a multi-party video conference, displayed in a nice, intelligent layout. You will be able to zoom in and zoom out and full screen any video you want.
An "active speaker" layout is provided to focus on the user currently speaking.

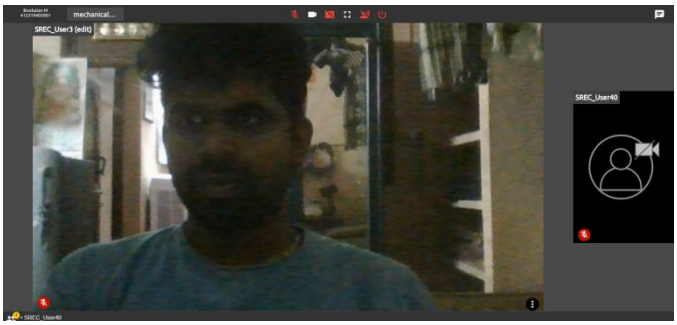

Fig. 11 Smart layout

4. CONCLUSIONS

The main goal of this paper is to implement a multipoint video conferencing system using Kurento media server through the WebRTC APIs, where each user is connected to all other users, and the same video stream must be delivered to all connections. We also focus on the description of the video conference processing system. We have realized a video conference system using a media server, which is a push messaging service, where you create a channel; you will find a unique ID that will be utilized in the app. This is programmed in HTML, and the core of the system is a JavaScript API. To do this, we applied protocol ICE (Interactive Connectivity Establishment) together with the Session Traversal Utilities for NAT (STUN). An endpoint is aware only of its private address, and a parameter from another LAN (Local Area Network) will be unable to use this address for a connection. So, the STUN server is used by each endpoint to ask the public address that stands in front of the NAT (Network Address Translator). Now, the connections between public addresses are more comfortable to access.

WebRTC technology will be available through user's browsers to minimize installation and use of plug-in in supporting communication. It will also improve the security of multimedia content and help developers to create better real-time video communication solutions.
The technology for video conference regarding the COVID19 crisis will increase the use of the online meeting applications, together with an improvement in the domain of thermal imaging, which provides a healthy and secure life. The technology for video conference regarding the COVID-19 crisis will increase the use of the online meeting applications, together with an improvement in the domain of thermal imaging, which provides a healthy and secure life.

ACKNOWLEDGMENT

I am extremely happy to express my gratitude in thanking our beloved correspondent Thiru Nithya Sundar, our secretary Thiru G. Kamaraj, principal Prof. Dr. A. Dhanapal, Ph.D, HOD Dr. B. Gowri Sankaran, M.Tech., MBA, Ph.D. and all department faculty members for giving an opportunity with all kinds of excellent infrastructure, encouragement, guidance and motivation for the successful completion of this paper.

REFERENCES

[1] Nayyef, Zinah & Amer, Sarah & Hussain, (2019). Peer to Peer Multimedia Real-Time Communication System based on WebRTC Technology. International Journal for the History of Engineering & Technology. 2.9. 125-130.
[2] Suciu G., Anwar M., Mihalcioiu R.,Virtualized Video and Cloud Computing for Efficient e-Learning, 13th International Scientific Conference eLearning and Software for Education, April 27-28, 2017.
[3] Suciu G., Anwar M., Virtualized Video conferencing for eLearning, 14th International Scientific Conference eLearning and Software for Education Bucharest, April 19-20, 2018.
[4] Vasilescu C., Beceanu C., Collaborative object recognition for parking management, 15th International Scientific Conference eLearning and Software for Education Bucharest, April 11-12, 2019.
[5] Elleuch, Wajdi. (2013). Models for multimedia conference between browsers based on WebRTC. 279-284. 10.1109/WiMOB.2013.6673373.
[6] Rodríguez P, Cerviño J, Trajkovska I, Salvachúa J (2013) Advanced Video Conferencing Services Based on WebRTC. In: Proceeding of IADIS multi conference on computer science and information systems.
[7] C. Cola and H. Valean, "On multi-user web conference using WebRTC," in 18th International Conference on System Theory, Control and Computing, ICSTCC, pp. 430–433, 2014
[8] M. Phankokkruad and P. Jaturawat, "An Evaluation of Technical Study and Performance for Real-Time Face Detection Using Web Real-Time Communication,", no. I4ct, pp. 162–166, 2015.
[9] T. Sandholm, B. Magnusson, and B. A. Johnsson, "An on-demand WebRTC and IoT device tunneling service for hospitals," in Proceedings - International Conference on Future Internet of Things and Cloud, FiCloud, pp. 53–60, 2014.
[10] M. Deshpande, "Integration of WebRTC with SIP – Current Trends," Int. J. Innov. Eng. Technol. Integr., vol. 6, no. 2, pp. 92– 96, 2015
[11] Julius Flohr; Ekaterina Volodina; Erwin P. Rathgeb(2018) FSE-NG for managing real time media flows and SCTP data channel in WebRTC.
[12] Edim Azom Emmanuel; Bakwa Dunka Dirting (2017) A Peer-To-Peer Architecture For Real-Time Communication Using WebRTC.
[13] Vamis Xhagjika, Oscar Divorra Escoda, Leandro Navarro, Vladimir Vlassov(2017) Media Streams Allocation

and Load Patterns for a WebRTC Cloud Architecture.
[14] Jansen, Bart & Goodwin, Timothy & Gupta, Varun & Kuipers, Fernando & Zussman, Gil. (2018). Performance Evaluation of WebRTC-based Video Conferencing. ACM SIGMETRICS Performance Evaluation Review. 45. 56- 68. 10.1145/3199524.3199534.
[15] Sodhro Ali Hassan & Giancarlo Fortino Energy Management during Video Transmission in WBSNs", 14th IEEE International Conference on Networking, Sensing and Control (ICNSC), Calabria, Southern Italy, May 16-18, 2017.
[16] Sodhro Ali Hassan. Power Control Algorithms for Media Transmission in Remote Healthcare Systems, IEEE Access,Vol.6, July, 2018.
[17] M. S. D. Vuþiü, L. Skorin-Kapov, "The impact of bandwidth limitations and video resolution size on QoE for WebRTC-based mobile multi-party video conferencing Faculty of Electrical Engineering and Computing , University of Zagreb," in 5th ISCA/DEGA Workshop on Perceptual Quality of Systems, pp. 59– 63, 2016.
[18] K. Fai Ng, M. Yan Ching, Y. Liu, T. Cai, L. Li, and W. Chou, "A P2P-MCU Approach to Multi-Party Video Conference with WebRTC," Int. J. Futur. Comput. Commun., vol. 3, no. 5, pp. 319– 324, 2014.
[19] S. Potthast, "Point to Point and Multipoint," Jisc community, 2016. [Online]. Available: https://community.jisc.ac.uk/library/janetservices-documentation/point-point-and-multipoint. [Accessed: 23-Aug-2017].
[20] J. Jang-Jaccard, S. Nepal, B. Celler, and B. Yan, "WebRTC-based video conferencing service for telehealth," Computing, vol. 98, no. 1–2, pp. 169–193, 2016.
[21] G. Carullo, M. Tambasco, M. Di Mauro, and M. Longo, "A Performance Evaluation of WebRTC over LTE," in 12th Annual Conference on Wireless On-demand Network Systems and Services (WONS), pp. 170–175, 2016.
[22] G. Carullo, M. Tambasco, M. Di Mauro, and M. Longo, "A Performance Evaluation of WebRTC over LTE," in 12th Annual Conference on Wireless On-demand Network Systems and Services (WONS), pp. 170–175, 2016.
[23] L. O. D. Nedberg, "Quality of Experience of WebRTC based video communication Eirik Fosser," Norwegian University of Science and Technology, 2016.

