Journal of Multidisciplinary Engineering Science Studies (JMESS)
ISSN: 2458-925X
Vol. 3 Issue 4, April - 2017

A Peer-To-Peer Architecture For Real-Time Communication Using Webrtc

Edim Azom Emmanuel
Dept. of Computer Science
University of Calabar,
Calabar, Nigeria
edimemma@gmail.com

Bakwa Dunka Dirting
ICT Directorate
University of Jos, Nigeria
bakwadunka@gmail.com

***Abstract*—Developments in technology have provided potentials for better communication methods. New technologies have emerged to improve existing communication channels. Some technologies applied in Real-time communication come with several challenges such as the need for additional software plugins and downloads in order to establish real-time communication, as well as security issues. Web Real-time Communication (WebRTC) is a technology that can be harnessed to overcome these challenges. The aim of this study is to design a system architecture that applies WebRTC and other technologies to support peer-to-peer real-time communication solution for an e-health application. During the study, different technologies were identified and integrated with WebRTC to develop the system architecture. Qualitative and quantitative data were collected during the development process. The solution was able to eliminate the problems associated with plugins, downloads of third party applications, and minimise latency, and bandwidth usage and also established real-time peer-to-peer audio-visual communication. A WebRTC performance and load test was carried out on the application using Blazemeter. The results show that high quality real-time peer-to-peer communication was established. The system recorded 16.92 ms transaction processing time for 1000 Virtual users and a response time of 25.29 ms for 15 mins uptime. WebRTC is a viable technology for delivering real-time peer-to-peer interaction with high quality of service.**

Keywords—Web Real-Time Communication; Teleconferencing; Peer-To-Peer; MediaStream; RTCPeerConnection.

I. INTRODUCTION

A major challenge within the web has been the inability for two web browsers to communicate and share videos, audios and data without support from software such as Flash, Java applets, Silverlight or Flex which are collectively referred to as plugins [1]. As a result, there has been a progressive effort by Internet Engineering Task Force (IETF) to specify the protocols for networking. The Worldwide Web Consortium (W3C) implemented the JavaScript API in an attempt to bring the realities of a whole new communication experience to Internet users as well as the webservers [2]. This development led to the introduction of a technology known as Web Real-time Communications (WebRTC). It is meant to achieve Peer-to-Peer (P2P) communication within the dependable Web ecosystem. Prior to the establishment of a P2P communication, a signalling process was implemented using various technologies such as Xml HttpRequest (XHR), session initiation protocol (SIP), Extensible Messaging and Presence Protocol (XMPP) and WebSocket. Google in collaboration with IETF, W3C, Mozilla and other corporations are working on the open source project [3]. The open source project is meant to help developers to be able to design and built various applications that will allow users to communicate by establishing video conferencing, text chat in the browser in real-time [4, 5], or to be used with existing Over The Top (OTT) technologies such as Public Switch Telephone Network (PSTN) or Voice Over Internet Protocol (VOIP) [6]. WebRTC is closely similar to websocket, but websocket opens a pipe of connection with a server instead of another peer. In most cases these technologies are used together for signalling purposes. In chat applications for example, WebSocket clients first send messages to the server and the server send the messages to the recipients. WebRTC promise to provide secured direct P2P communication between users and free of plug-ins [7].

WebRTC assures a simplified, flexible and cost effective means of real-time communication for users without dependence on service providers. A critical challenge with plug-ins such as Flash, Silverlight, and Shockwave is the need for downloads each time a connection is to be established. Plugins can be problematic during execution, they increase bandwidth, latency, execution time and speed [8]. The implementation of video based communication without the need for plug-ins will eliminate the problems that are associated with plug-ins as well as impose greater coordination in communication. Technologies such as Network Address Translation (NAT), Session Traversal Utilities for NAT (STUN), Interactive Communication Establishment (ICE), Session Description Protocol (SDP), User Datagram Protocol (UDP) formed a significant part of the system architecture developed

during this study. The technologies were useful in supporting real-time media exchange over distances across platforms [9, 10]. The designed WebRTC architecture establishes end-to-end encrypted P2P communication with audio-visual content and data being transmitted directly. This implementation is designed to bypass intermediary hardware server and eliminate security challenges like interception of data by hackers. This unique feature makes the difference between WebRTC and other RTCs such as Skype.

Although Skype is a great communication service, it is proprietary and lacks the direct P2P ability, as well as a credible security feature found in WebRTC. Peer-to-Peer also enable user's data to be encrypted [11], safe, and cannot be compromised. A peer–to-peer connection can be credible because it is able to bypass all the problems associated with plug-ins. Factors such as latency, bandwidth and memory utilisation as well as support for anonymity are supported in WebRTC. In order to implement the designed system architecture in this study, a P2P e-health application was developed and evaluated. The application was deployed and tested on the Heroku platform. The open source Heroku platform as a service (PaaS) was also used to host the application and also for user evaluation. WebRTC Technology showed great potentials for providing peer-to-peer real-time communication between users [12].

WebRTC is not a service nor an application, rather, it is a technology without installation support for its components like media engine or codec. This factor may likely change the technology market in future due to the speed of adoption of WebRTC by large firms.

II. LITERATURE REVIEW

Most researchers describe WebRTC as "Skype-like" technology, but Skype has far existed before WebRTC and the two are completely incomparable. Despite the dominance of Skype in real-time communication industry, with over a billion subscribers, Skype still has certain drawbacks. It is proprietary and users must install supporting software or plug-ins such as Flash for it to work properly. The plug-ins used along with Skype can impose security concerns within the system. Skype also adopted the client server architecture which makes it resource intensive [13]. WebRTC also uses peer topology with less overhead. It is an open-standard / open-protocol descendant of FreeBSD media engine [14]. This means that all the real-time capabilities that existed in Flash plugin have been made available natively in WebRTC-compatible browsers so that developers can use the technology to develop various real-time solutions with ease [15, 3].

A. Communication Technology Issues and Remedies

One challenge with Internet communication before the emergence of WebRTC is the dependence on service providers. Before WebRTC was introduced for real-time video communications, vendors such as public switch telephone networks (PSTN) had used more intricate communication processes. Their clients must participate as members of a separate PSTN IP based community. This membership is maintained by a service provider who delivers basic rudimentary necessities. There must also be an assurance that each telephone number is unique for each physical location i.e. a mobile device must be associated with a unique user or service. These types of communication may also require participants to subscribe or buy a product, while others in addition may require the participants to download plugin before multimedia contents are delivered appropriately. For example WebEX and skype. These cases required third party applications, thereby limiting the scalability and diversity of communication. With WebRTC each website is essentially its own "service provider", without the need for any relationship with a party outside of itself and the user establishing the communication.

The issue of lack of trust during Installation of third party software or plugins is a big concern. This is because these technologies can surreptitiously introduce malicious software and malware. This may pose technical problems with the application. WebRTC is a technology that can be used to overcome the problems associated with plugins installation. Another problem is the lack of a standardised real-time media engine which can be access freely through a simple secured hypertext transfer protocol (HTTPS). In addition, security flaws or potential vulnerabilities in most browsers are programmed to be automatically updated when discovered and sometimes fixed in newer versions. Similarly, a WebRTC browser implementation can easily be fixed rapidly as compared to traditional applications such as Voice over Internet Protocol (VoIP) application. The complex VoIP addresses similar problems posed by malicious software by developing patches to address the security flaws. This often take much time. The importance placed on browsers are generally huge due to their ubiquitous nature and speed of information accessed [16].

Skype uses a domain name server (DNS), and this may cause the decryption keys of media contents conveyed through their service to be intercepted. WebRTC ensures encryption and authentication of voice, video and data by default. This is achieved through Datagram Transport Layer Security (DTLS) and secure real-time protocol (SRTP). It is used to prevent eavesdropping and recording of the voice and video data in WiFi networks.

WebRTC does not hinder support for other communication and collaboration protocols. This is also a major consideration for a reliable session establishment using Network Address Translators (NAT). This is important because it avoids delays of responses from servers and drastically reduces factors such as server load, latency and intensifies quality. This feature is good because it helps in the development of customized applications that allow

others vendors to initiate communications session with WebRTC applications. This unique achievement realized with WebRTC is not the same with skype. Application developed using WebRTC allows participants to join the interactions from any site without having to undergo separate registrations or bear cost as a result of joining the interactions like it is experienced with linkedln or google for federation.

B. WebRTC Implementation API

The impact of WebRTC can be viewed from different angles. It can be viewed in terms of the protocol stack, Codec, SDP and signalling [17]. It can also be viewed in terms of the API [18]. WebRTC relies on three implementation APIs which performs different roles to enable real-time communication within any web application [3]. They include: media engine, RTCPeerConnection and RTCDataChannel.

Media Engine (getUserMedia)-The media engine enables the browser to access the user media such as microphone and camera. This API is also part of HTML5 used in accessing hardware directly. getUserMedia avoids the use of external codec to capture audio or video data.

RTCPeerConnection - An RTCPeerConnection makes the actual WebRTC connection possible, while WebRTC actually handles the efficient streaming of data between two peers. Hence, for a caller to initiate a connection with a remote party, the browser must begin by instantiating an RTCPeerConnection object. This API sends the real-time media data and it is responsible for managing the full life-cycle of each peer-to-peer connection, encapsulates all the connection setup and management, and its state within a single easy-to-use interface [19].

RTCDataChannel **-** The RTCDataChannel is an API that is offered as part of WebRTC designed to exchange arbitrary data between peers. RTCDataChannel acts like the well-known WebSocket, but offers a customizable transport protocol. It is useful in many applications such as game applications, file sharing and text chat applications.

C. Internal WebRTC Architecture and Standards

The inherent WebRTC architecture consist of a Web API for developers. It also contains a platform for developers to handle issues relating to capturing and rendering hooks [20]. These layers are mandated to work across browsers and also on different platforms. The Web API layer present the web developers with a RTCPeerConnection, RTCDataChannel, and MediaStrean objects.

The WebRTC native C++ API allows an easy implementation of the API for different browsers. It consist of a session management and signaling management modules that takes care of session establishment and signaling that enable developers to set up calls easily. WebRTC also handles the implementation of different transport mechanisms. The architecture also consist of a VoiceEngine and VideoEngine collectively referred to as the media engine [21, 22].

The VoiceEngine framework transmit audio contents from the sound card into the network. Examples of VoiceEngine include iSAC a wideband codec, iLBC a narrowband codec and OPUS. iSAC and iLBC were initially products of the Global IP Solutions but became part of WebRTC in 2011. These codecs basically manages audio streams [23]. The VoiceEngine provides features to keep voice latency and bandwidth in microphones at a low level, while retaining high quality. It include dynamic jitter buffer and error concealment algorithm used for concealing the negative effects of network and packet losses. It also handles the negative effect of echo cancellation, VAD, noise reduction, compression, encryption as well as the statistics [24, 25].

VideoEngine framework controls bi-directional movement of video contents from camera to the network and from network to the users screen. It includes features for camera image capturing, video processing, and video image enhancement. It also provide the Dynamic Jitter Buffer to help increase video quality and conceal any de-jitter, packet loss, and Bandwidth Management. The video codec include VP8 and H.264 [26]. Fig. 1 shows the internal WebRTC voice and video codecs being part of the internal architecture.

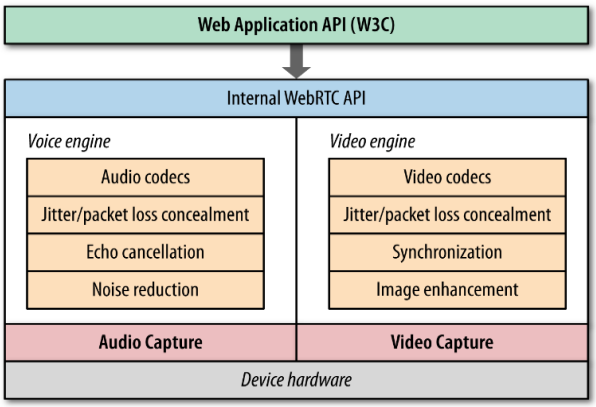

Fig. 1. Built in audio and video WebRTC engines [27](Ilya, 2013)

D. WebRTC Protocol Stack

The core working philosophies of the media engine cannot be achieved without the WebRTC protocol stack presented in Fig. 2. The components of the stack include ICE, STUN, and TURN used for network address translation (NAT). A NAT basically establishes and maintain a peer-to-peer connection over UDP transport protocol. UDP controls the delivery of packets of media streams also known as the transport channel. Data transfer is achieved through the DTLS component which secures media contents by mandatorily encrypting the data. DTLS is equivalent to TLS but

works only with UDP. The SCTP and SRTP are the application protocols used in multiplexing the different streams, data framing and to provide flow and congestion control.

SDP: Session Description Protocol - Participating peers needs a way to exchange call setup information. This is where SDP comes in. SDP holds basic metadata about a browser prior to peer connection. The new sessions are announced by initiation, invitation and exchange of information. The information to be exchanged include; Video and audio media capabilities, codecs information, user information which include IP address and port numbers, secured RTP P2P data transmission protocol, available bandwidth, session features such as name, identifier and active time.

ICE: Interactive Connectivity Establishment - According to [26] and [28], ICE is used to enable participating peers to understand how to exchange media data. ICE factors out the best path for such communication based on the information that is interchanged between peers via wired or wireless network interfaces. In trying to simplify the process of finding the best path through NAT, ICE chooses the most efficient algorithm for NAT when it attempts to create a connection using the host IP address and port obtained via the Operating System and the device network interface card. If this attempt fails due to the peer being behind NAT, in this case, ICE uses STUN server to generate an equivalent public or external IP address. And if this also fails, a TURN server will be used to route the public IP over the other peer's device. NAT dynamically converts private IP address into a public IP address when an outbound request is passed through. Similarly, inbound requests to a public IP are converted back into a private IP to ensure correct routing on the internal network. This implies that private IPs alone are often not enough to establish a connection to another peer.

STUN: Session Traversal Utilities for NAT **-** In order to perform P2P communication, both parties require at least the knowledge of their peer's IP address and the assigned UDP port. As a result, a certain amount of information exchange is necessary before WebRTC communication can be established. STUN server are used freely by each peer to determine their public IP address and ports. STUN uses stun.l.google.com to obtain an API key.

TURN: Traversal Using Relays around NAT - TURN servers are used as a fallback preference where STUN fails in establishing P2P communication. TURN server works by relaying traffic between peers. The WebRTC communication can be ensured, but can suffer degradations of media quality and latency. Though they guarantee better connection establishment in whatever user's environments. TURN impose more overhead on bandwidth especially for simultaneous calls routed through the server. TURN server is not completely free.

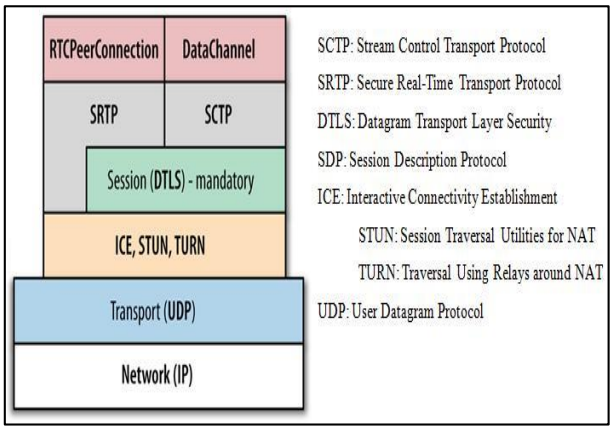

Fig. 2. WebRTC Protocol Stack [27]

UDP: WEBRTC Real-time transport protocols - Real-time communication is a time critical activity that may result in intermittent packet losses during video streaming. The WebRTC audio and video codecs has overcome this challenge by implementing various logic to recover from losses or packets delays. And at the same time, considers timeliness and low latency in data transmission as major factors. WebRTC considers these factors more important than reliability of data. This is the main reason why UDP protocol is the preferred option over TCP for delivering real-time data. For TCP, it delivers reliable, and ordered stream of data. For instance, if an intermediate packet is lost, then TCP will buffer all the packets after it, wait for a retransmission, and then delivers the stream in order to recover. While UDP offers No guarantee of message delivery or order of delivery, No acknowledgments, retransmissions, or timeouts, No packet sequence numbers, no head-of-line blocking, No connection state tracking, establishment or teardown state machines, congestion control, built-in client or network feedback mechanisms. As a result, UDP offers no reliability promise. UDP transport protocol therefore delivers each packet to the target application the moment it is sent. In effect, it is a thin wrapper around the best delivery model effort offered by the IP layer of the network stacks.

WebRTC uses UDP at the transport layer for transportation of video streams since latency, timeliness and bandwidth are critical. WebRTC requires encryption of all media, the gaps within UDP are filled by additional protocols implemented on top of UDP, such as SRTP and DTLS. SRTP is used to transport audio and video streams, while DTLS is needed for encryption of every data while on transit as required by WebRTC. TLS would have been the best protocol of choice, but because it cannot be used on UDP. It is rather used on a reliable protocol such as TCP. DTLS has been introduced to offer an equivalent security as TLS. It also offer more reliable delivery of handshake records to negotiate the tunnel and in order, and it is also fragmentation friendly. This is why it is able to fix the problems with TLS. DTLS negotiates the secret keys for encrypting media data and for secure

transport of application data. SCTP is used for application data transport [3].

E. WebRTC Communication Topology

There are two communication topology that are useful in setting up calls, and are used as the situation demands. These include the trapezoid Model and the triangle Model.

The trapezoid Model **-** This model allows browsers to run on separate web servers or gateways using any kind of signaling mechanism such as SIP, Jingle or other proprietary protocol. This communication model is important because different corporations will synchronize their communication and avoid duplication of functions for their various web applications, such as sharing the same address space for each of their client. This type of model are applied for mobile phone numbers subscribers such as MTN and GLO mobile networks. The signaling process is set up to use HTTP or WebSocket transport protocol for communications unlike WebRTC which allows direct communication between browsers. This model may incur more overhead in terms of bandwidth, latency and resources consumption.

Open triangle Model -. The Open triangle model is arranged in such a way that two clients can share the same web server [29]. The web server stands in the center to facilitate the connection between the Peers and to establish voice, video and data communication. Skype is an example of this type of model. Skype facilitates all the communication and as well coordinating all the addresses. The peers do not have to federate with individual webservers. The triangle model also has a closed triangle variant. WebRTC is another example of this model. The key points about this model is that, 1) the single server coordinates communication, that is, the signaling process, but the model do not participate in the communication, 2) any kind of server can stand in the middle to coordinate the communication, 3) the media contents are passed directly the advantages are for low latency, speed, low bandwidth, and security, 4) media contents can still pass through firewall installed between the peers and signaling layer.

III. RESEARCH METHOD

This study was conducted through an iterative process. The first step was to identify the required technologies including WebRTC technologies for the design of the system architecture. Also at this stage, an E-health application was considered for real-time communication and implementation of the WebRTC and other technologies. The potential users were also identified and data was collected from them through interviews. The second step involved the design of the architecture of the real-time communication system. The next step involved the development of the application and testing. The application development process was iterative. The application was also evaluated by the potential users. The final process in the study was to perform analysis of the test results to determine the system was carrying out real-time interactions.

A. Survey of Required Technologies

One important consideration in the design of the real-time application was to make it simple, testable and a reliable application. The study focussed on providing a technical solution using WebRTC technology to deliver real-time interactions to the users. Other technologies that were identified as suitable solution to the system requirements include MongoDB, ExpressJS, AngularJS, Node.JS (M.E.A.N) stack technology. The backend (server) was implemented with websockets server written in Node.JS. MongoDB was implemented to house the various users and meta-data documents in BSON format. A tool called MongoLab was useful for organising a readable JSON display for mongoDB data. The interfaces were developed using JADE view as a default frontend replacement for Angular.JS.

B. Design of the System Architecture

The design stage is a significant stage that provides a description guiding the process and method that were applied in this research. The research design for this study followed the planning / listening, design, coding, testing and implementation phases. The WebRTC communicating platform was also evaluated. In the planning and listening stage, the researchers followed the listening concept of the agile method to extract the basic requirements. It involved the identification of specifications typical for determining in very simple terms how the real-time application should work. The system design and coding of major components was achieved by creating a simple M-V-C architecture for the proposed system. This involved writing program and logic crucial in creating the quality application.

C. M-V-C Controller

The prototype for building the application was based on the MVC architecture. We may think of a Model as data or information store, the View as the layout of the user interfaces that interchanges data, and the Controller as the logic that handles the control flow including any business logic needed to build the Model and pass the model data to the view for presentation to its end users. In the design of the system, a route component was added between the controllers and the mandated user's browsers. The route was written with Express.JS framework for Node.JS necessary in order to coordinates interactions. One common strategy was also to implement a representational state transfer RESTful API which feeds a Single Page Application (SPA). The REST component provided good representation, visibility, and performance through the use of HTTP/S method while supporting simpler internal communication and control flow. In the design, an HTTP request was routed to the appropriate controller action which in turn processes the various information then returns the appropriate model and view for rendering to the other peer. Thus, in this implementation the M-V-C was instrumental in handling

the logic, visualization and data. Fig. 3 presents the MVC architecture and MEAN stack implementation of our prototype.

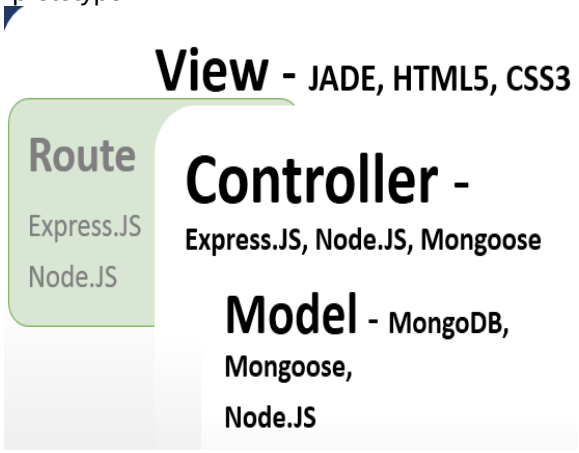

Fig. 3. MVC – MEAN architecture.

D. The system architecture

This section explains the implementation of the peer-to-peer technology. The WebRTC P2P architecture mentioned in the previous section was left out specifically for this section because it is part of the implementation of this study.

The generic architecture of the system adapts to the peer-to-peer architecture. Two peers will be able to communicate with each other after a signaling process has been completed. In Fig. 4, WebRTC is designed to sit within the browsers to ensure direct media communication in a peer-to-peer (P2P) fashion with no plugins. The accomplishment of communication between peers must begin with an attempt to initiate and facilitates familiarity between the callers. This is explained by the process of Offer and Answer.

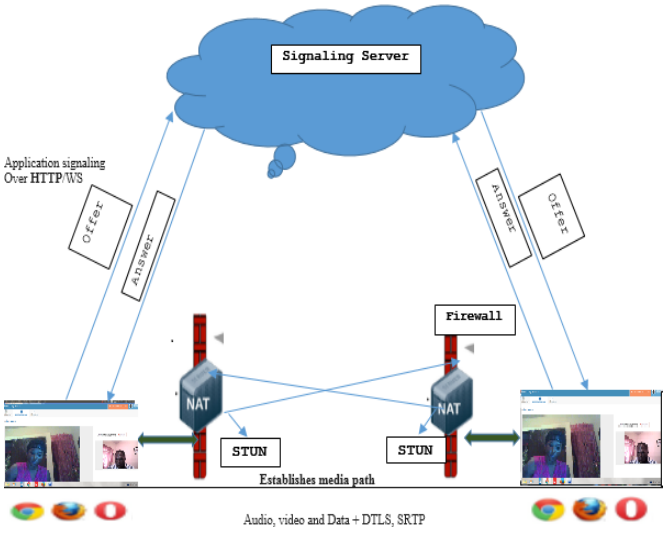

Fig. 4. The P2P system architecture

The minimum standard infrastructures necessary in set up the above WebRTC architecture include 1) the client's browsers, the HTTP signalling written in node.JS. It is required to introduce the peers together. The STUN server was used to find an optimal path to relay the media. Other infrastructures that could be used but beyond the scope of this study include Asterisk, SFU or Multipoint Control Unit (MCU). These facilities are used for large multiparty video conferencing, recording and gateways.

E. Description of the Generic Architecture

The few lines of the implemented codes snippet captured the part that describes the minimum functions for setting-up a successful video conferencing. Google Chrome and Mozilla Firefox browsers were used to enables access to media devices. The actual permission captured during the implementing are described in Figure. To establish the connection between two peers, a Websocket signalling layer was implemented using socket.io library. This layer was necessary for the signalling server to communicate with the peers freely. The server is written in Node.Js. The code snippet below captures Node.JS server and socket.IO initialization process.

```
var app = require('../app');
var debug = require('debug')('hrtc:server');
var http = require('http');
var port = normalizePort(process.env.PORT || '3000');
app.set('port', port);
var server = http.createServer(app);
var io = require('socket.io').listen(server);
require('../sockets/base')(io);
server.listen(port);
server.on('error', onError);
server.on('listening', onListening);
var socket = require('socket.io');

…
```

Again, the implementation of the STUN server based on this study is described using the codes snippet below.

```
var peer = new Peer('#{session.userId}',
key: 'he3gxx6jfoxyldi',
debug: 3,
config: {'iceServers' : [
{url: 'stun:stun.l.google.com:19302'}
]}
});
peer.on('open', function(){
$('#my-id').text(peer.id);
});
peer.on('call', function(call){
    …
```

The frontend is implemented using the Jade View Engine a default framework for AngularJS. This is initiated with the code snippet below

```
app.set ('views', path.join (__dirname, 'views'));
app.set ('view engine', 'jade');
```

F. The Offer, Answer, Handshake and Connection Process

When the signalling process first starts, an offer is created by say the first peer who is the initiator of the call, the Offer contains a session description called SDP, this needs to be delivered to the second peer i.e. the person receiving the call. The second peer

responds with an answer message, which contains an SDP description about the other end. This now signify that the two peers now know certain details regarding each other to be used for the call example of these details include video parameters, codec information, transport protocols, ports and codecs used. One problem is that the two peers need to also understand how to transmit the media data, this is realised through the use of Interactive Connectivity Establishment (ICE). ICE figures out the best path to communicate between the two peers based on the information gathered from each peer. For example: through wireless or wired network interfaces. ICE is a combination of IP address, port, and transport protocol. The handshake model enables exchange of networking information with participating peers. Fig. 5. describes the entire Offer, Answer, Handshake and Connection implemented.

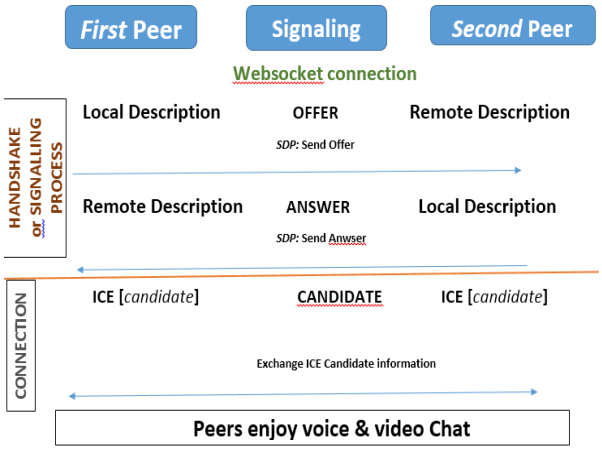

Fig. 5. Offer, Answer and connection process for the application

In Fig. 5, in order to connect to another peer, its location on the web need to be known. This is a logical process where both peers need to first create an *RTCPeerConnection* object and obtain a *Session Description*, an object that indicates what kind of data they want to send to the other peer through the peer-to-peer connection. They do this by calling the built-in methods of the RTCPeerConnection object. The initiator of the video call will obtain a self-session description called the *offer* and then set it as a local description by invoking the method *localDescription*, then sends the offer to the other peer through the signaling channel. The other client receives the offer and set it as their *remoteDescription*, they will also obtain their *own* session description called the *answer* and set it as their *localDescription*, and send it back to the initiator through the signaling channel. The initiator receives the answer and sets their own description using the *remoteDescription method.*

G. Application Prototyping and Testing

The actual structure describing the M-V-C directories and design prototype of the application is described in Fig. 6.

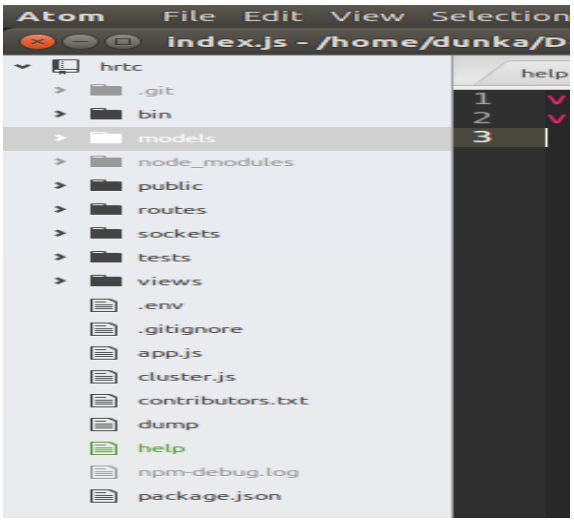

Fig. 6. The application directory in M-V-C

The technologies and design specifications that was put into consideration for this interactive system include a peer–to–peer network architecture, Heroku platform as a service (PaaS) was used for cloud hosting. Some operational and internal design specifications include the creation of an updated node package manager (NPM) modules, capturing the environment variables, organizing the models, views and controllers M-V-C logic as well as creating the built-in and user defined middleware, routes, ExpressJS, WebRTC integration logic by providing internal implementations for SDP, NAT traversal, ICE, STUN servers integration, codecs, User Datagram Protocol (UDP) transport protocol, support for full duplex communication using websockets. SDP signaling and other meta-data in the implementation of this study were transmitted using javaScript object notation (JSON) format. Finally the logic for including the mandated WebRTC browser options were created.

H. WebRTC Video Conferencing Topology

The signaling process happens via a server which is separate from WebRTC, this is important philosophy because it gives room for implementers to choose whatever signalling protocol they so desire which in-turn gives room for use with any kind of signalling protocol and communication end points such as Session initiation protocols (SIP), extensible messaging and presence protocol (XMPP) Jingle and most other over the top (OTT) technologies such as Public Switch Telephone Network (PSTN) [4, 15, 30].

I. Implementation of Mean Stack Component

Since one aim of this study is to enhance better and timely interactions, the design of the WebRTC videoconferences application was implemented alongside MongoDB, ExpressJS, AngularJS, Node.JS (M.E.A.N) stack technologies. M.E.A.N stack improves on some limitations found in the current LAMP stack which existed about 2.5 decades ago. This components of the MEAN stack offers more convenience in this application in terms of speed and

real-time application. The MEAN stack components has many benefits which include: Same language of data transfer across every layer of MEAN in the JSON format, use of superfast V8 chromes engine, Node.js is Event-Driven and conforms to a non-blocking capability. These benefits can go on but finally narrows down to optimizing throughputs and ease coding effort to make for a better version of a real-time video conferencing application.

J. Deployment on the Heroku Platform and Evaluation

Heroku platform as a service (PaaS) was chosen for the deployment platform because it allow developers to focus on building, running, scaling, storage and better management of the application. It helps in reducing cost, better access to developer infrastructure and support for quality of service. We considered the benefits of allowing developers to concentrate on the sole responsibility of development rather than concentrating on both development and hosting infrastructures. This platform caters for hardware and software infrastructure, it uses tools that the developer is aware of, an example is the Git Bash command line Interface (CLI) for passing commands for development, manipulating, monitoring and hosting purposes. Effortless scaling is another benefit which allow developers to quickly scale applications dynamically through their interface matrix system or through passing commands. The Heroku infrastructure has dramatically reduce the time and cost for developers. Heroku supports various platforms and has less to do with incompatibility issues. It means application developed with various languages can enjoys the same services. The result of deployment on heroku PaaS result was achieved using the code snippet.

```
$ heroku login   // login into the heroku account
and install the toolbelt
git init
   heroku create app_name
   git remote -v
git add.
git commit –m"also create a file in the directory
with content www:node bin/www"
git push heroku master
```

K. Application Testing

In order to determine the practicality of the application on the local area network (LAN), a test was carried out with several peers (users) within the University of Calabar. Thereafter, Blazemeter "a tool for generating load testing and performance matrix" was used to test for performance of the application prototype. WebRTC internals and the open source Heroku dashboard were used to measure the performance of the application based on real live test. Parameters such as transaction processing time (TPS), Virtual users (VU), Errors, Response time / Latency, Hits, Bandwidth, and memory usage were measured and captured**.** Several of these tests indices were displayed graphically and reported in the result section. The initial configuration were as follows: Engine: console only (1 console, 0 engine), RAMP-up: 300-1200, users / peers 1-1000, Duration: 15-30 minutes, iteration: Test continues for ever. This simple setup is targeted towards meeting up with the performance goals.

IV. RESULTS AND ANALYSIS

Fig. 7 presents the e-health application user interface. The interface shows a video conferencing communication between two users. The users are engaged in real-time interactions. It is a direct connection between the users' browsers devoid of any conventional DNS server connection between the users. The users' browsers did not need the support of any third party plug-ins or downloaded software such as flash for the video to play on both browsers. The connection is possible because the *getUserMedia()* method establish access to the cameras and microphones. Once the video conferencing button is enabled by the users, WebRTC mandate a request for permission to use media devices. The users can then take any of the options to "allow" or "block" the request. Taking the "allow " option means that the system will have access to the users' camera and microphone for real-time interactions. The application was tested and ran on Firefox, Opera and Google Chrome browsers.

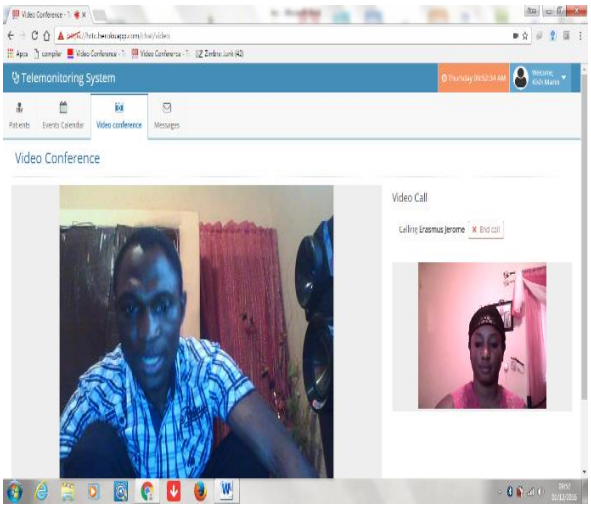

Fig. 7. Real-time video communication between peers.

A. System Performance Analysis

The Graph in Fig. 8, shows the performance of the system. The different curves in the graph represent different performance attributes from the real-time communication system. The result shows that real-time communication was established and the system performed adequately.

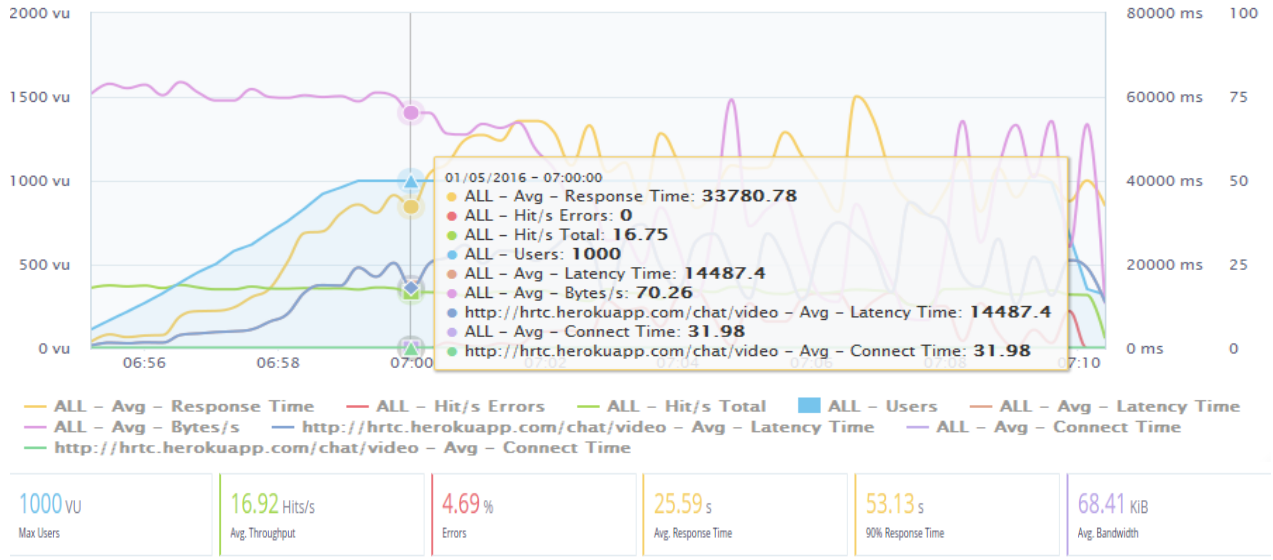

Fig. 8. Performance graph for 1000 VUs

The system recorded an average throughput or Transactions processing Time (TPS) of 16.92ms for 1000 VUs and a response time of 25.9ms for 15 minutes uptime of the load test.

The throughput is directly proportional to the load or number of users and the type of activity performed on the system (e.g. video chart). This means that the system can reasonably sustain increasing load of 1000VUs with very high throughput. The high and low movement of some of the curves indicates fluctuations with server response during loading of data. The average response time based on the number of users is an indication that the system has the capacity to serve these number of users with the required quality of service. The response time measured in seconds is a very critical factor in user experience because it indicates the user waiting time. This is the time taken by the client to connect to the real-time communication system with a request and also to receive the desired media response as the load increases. Based on the recorded system response time, the users will experience high quality real-time interactions with their peers. At the time of consideration, the system recorded minimum errors (4.69%). This points to the fact that the technologies applied in the design and implementation of the system are reliable and will be able to provide the expected service.

The resource utilization graph was obtained from Heroku matrix dashboard. This shows the utilization of resources such as the CPU, memory and the network. The level of utilization of these resources can adversely affect the performance of the system. From the results, it is evident that the network, memory, CPU and VU connection shows that these resources are adequately being used during the real-time interactions.

It was found that the WebRTC and other technologies put together to design the architecture of the system successfully enabled real-time communication between peers without the need for plugins and other downloaded applications needed to support the establishment of communication. This will greatly increase data security from eavesdropping, unauthorized access and other server related security issues. Also, the overall cost of communication is greatly reduces for the users. The STUN server within the WebRTC is freely used by each user to establish the communication between them. This reduces the cost of communication.

B. WebRTC Internal Output

The overall benefit of implementing WebRTC on recent browsers is for adaption and to produce better quality media. When there is an established communication between two peers, WebRTC records media information during the session. These are output that shows the statistics of the processes and performance of the system during the communication session. These media information is presented next.

Fig. 9 shows the WebRTC STUN server parameters that indicate the actual processes performed during the communication session between peers. These include request and responses sent and received information, the timestamp, the amount of packets sent and received, the bytes sent and received and others. STUN servers also display the round trip time (RTT) obtained from the last STUN request displayed as googRtt report. ICE server displays various parameters such as localCandidateId, remoteCandidateId. This indicates the l local and remote ICE candidates during the real-time interactions. The information from the STUN and ICE server in the WebRTC shows that communication was established between peers. One important point about this information from webrtc is that, when there is a problem with media transfer, this is easily reported here. Information regarding Jitter

received, jitter buffer, frame size and network are also captured. The information in Fig. 9 also presents the amount of data packets that was sent and received through the video channel in WebRTC. And so, the information showed that the STUN server enabled the remote ICE candidates to interact in real-time.

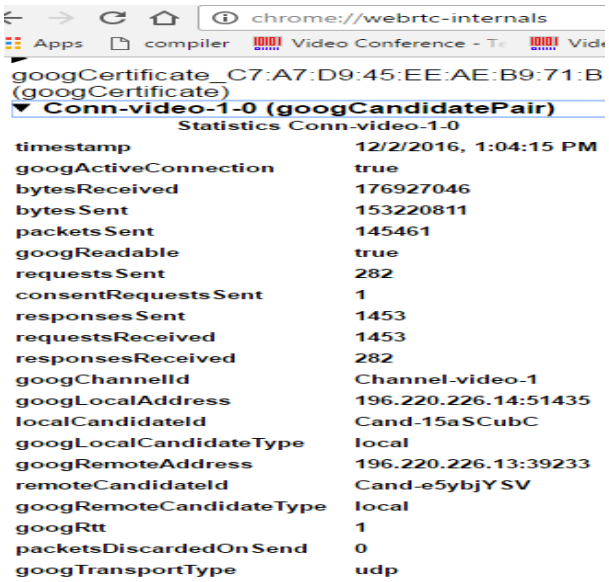

Fig. 9. WebRTC STUN server performance Output

Fig. 10 also presents statistics from the WebRTC STUN and ICE server for the communication session between the two peers with one of the peer using Google Chrome and the other Firefox web browser. Different WebRTC statistics can be provided for successful and unsuccessful communication. In this case successful communication took place as indicated by the createOffer, CreateOfferOnSuccess, setLocalDescription, and addIceCandidate parameters in Fig. 10. The results show an optimized outcome of the getUserMedia and RTCpeerConnection events in WebRTC and breaks down the complex process of ICE, SDP and STUN processes. The core of RTCpeerConnection include ICE keep-alives to guarantee that UDP do not easily expires. Once authorized, they will ensure that user agents are actively working to send and receive media. Each candidate involved in the communication process is presented as “a=candidate”. This is similar to a container comprising major information that need to be known to other peers to facilitate the connection. It includes the IP address, port, priority, transport protocol and component id.

As soon as the Offer and Answer processes are completed, ICE accomplishes its task, while also verifying and implementing connectivity. These processes are indicated with symbols (Fig. 10) such as: 1) v=0 – : defines the SDP version that is used. 2) s=- : indicates the session name 3) t=0 0 :Refers to the session start and end times. The second 0 describes the ending time that the session is valid at but not necessary limited to a specific time 4) a=group:BUNDLE video: This shows that the communicating browsers support each other for video communication and that they are capable of multiplexing the video at hand with the same RTP session. The BUNDLE grouping displayed here is a video line associated with the SDP, it can also be an audio. 5) a=msid-semantic: WMS which is a unique identifier for the WebRTC Media Stream (WMS) during the PeerConnection’s life. The digits 100 101 116 117 96 97 99 88 indicates the video format that was sent by the browser to its communicating peer 6) c=IN IP4 0.0.0.0., indicates the IP of the target source and destination used during the interaction session. It also specified the use of RTCP for multiplexing. Fig.10 proved that there was exchange of data between two users in different locations. The information proved that real-time video based interactions between peers have taken place. Also, the system architecture developed using WebRTC and other technologies was able to deliver real-time video based communication between two users.

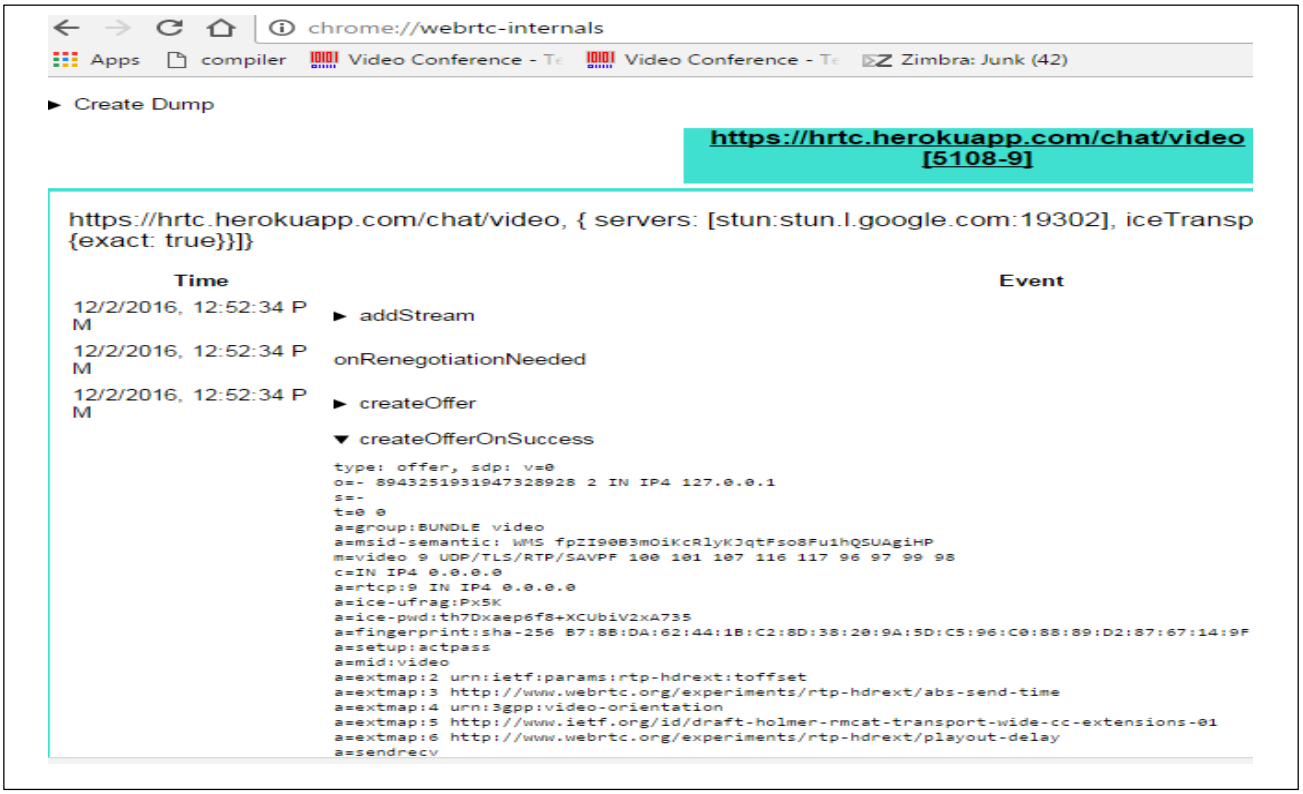

Fig. 10. WebRTC Internal Parameters for Actual Video Communication

Fig. 11 presents the graphical details of the actual amount of data in bits that were sent and received per second and the data packets sent and received per second during the interactions between the peers. The graphs also shows the time of interactions. The WebRTC and other technologies used to develop the system architecture provided the users video based real-time interactions at reduced data cost and with quality of service.

This was done without browser plugins and additional plug-in software like flash and downloads as necessary requirements to enhance audio and video based interactions. The elimination of these requirements means that the conversations and data are better secured from attacks and eave dropping.

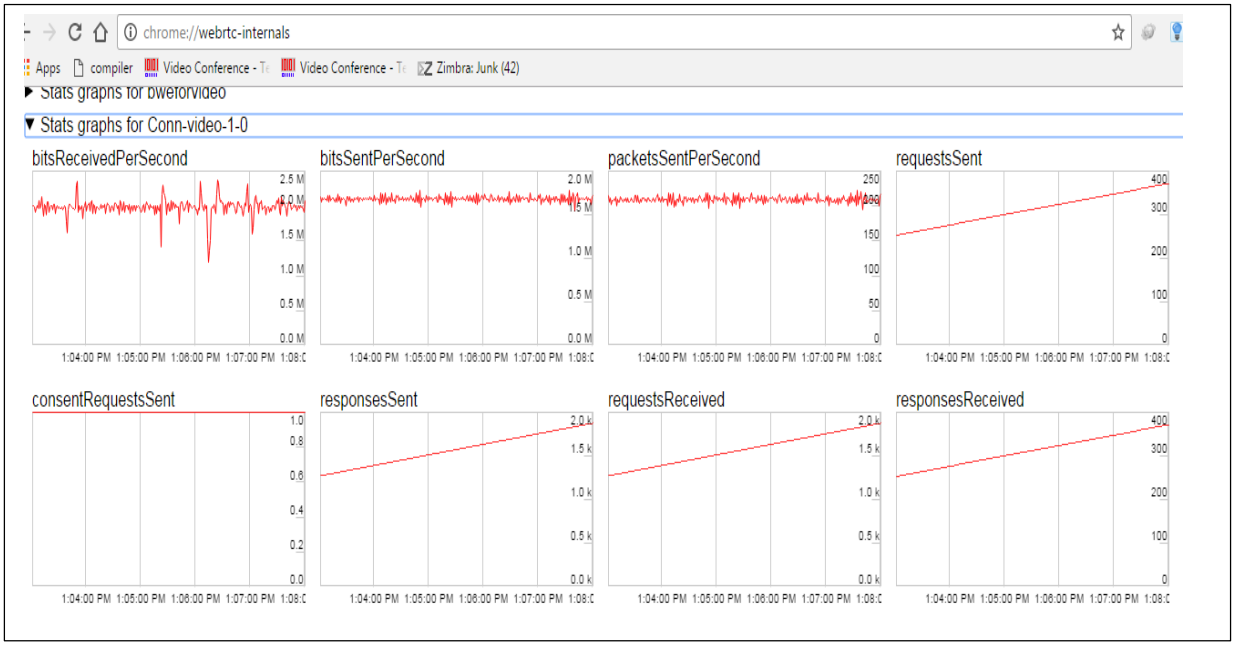

Fig. 11. Statistical Graphs showing exchange of data between peers.

V. DISCUSSION

WebRTC is a new technology that is still under development. It is a technology that has potentials for providing quality of service and enhance user experience on browsers. In this study, a system architecture for real-time audio and video based interactions was developed. The architecture combines WebRTC and M.E.A.N stack components. An e-health application was developed to evaluate the architecture [12]. These technologies were useful in building a functional cross platform solution that implemented minimum standard architecture. From the real-world perspective, this project did not cover everything about the capabilities of the technologies. The system that was developed was adequately tested and proved to work as expected across different browsers, although it is possible to experience unexpected errors or changes as browsers are modified or as WebRTC implements new features.

The system performed its function of establishing real-time video conferencing between peers in remote locations using WebRTC supported browsers. The peer-to-peer communication was made possible after a signalling process that introduces the peers together has been established with Node.JS through HTTP unlike the client-server architecture. The advantages of this implementation include speed of delivery, secured from interception or intruders, reduced latency and bandwidth. Also, the communication was achieved using end-to-end encryption. An e-health application was created using WebRTC and M.E.A.N stack without installation of custom drivers, without plug-ins and software downloads.

The application was hosted on the Heroku PaaS cloud platform as part of the deployment process. A flexible deployment platform was the target in this study because of its benefits. A load test was carried out using blazemeter. The results shows that there was real-time interactions between remote peers. Audio and video based data was sent and received at both ends. Factors that could affect the speed of any user request include the CPU speed and the network strength or quality of the network. The user interface for interaction was design with simplicity and ease of use. This will enhance users' experience.

VI. CONCLUSION

In this study, a practical experience in the analysis and design of a system architecture for real-time communication has been presented. The architecture was implemented using WebRTC with its inherent features and other technologies in bringing the benefit and experience of a more flexible, speedy and cost effective real-time communication to all Internet users. WebRTC technology will be available through user's browsers to minimize installation and use of plugins in supporting communication, it will also improve the security of multimedia content and help developers to create better real-time video communication solutions. Apart from improving user experience, quality of service, it will also reduce cost of communication, and provide better security of user data and information.

The implementation of this new technology will help in breaking the monopoly that has existed with under the control of most OTT corporations and bring innovative opportunities to synchronize with existing and future applications. WebRTC is still in its infancy stage, more development is in progress to decide on certain standardization policies that will improve the technology and provide quality of service.

REFERENCES

[1] C. Chuang-Yen, C. Yen-Lin, T. Pei-Shiun, and Y. Shyan-Ming, "A Video Conferenceing System Based on WebRTC for Seniors", In: *Proc. of the 2014 International Conference on Trustworthy Systems and their Applications*, 51-56. 2014.

[2] R. Manson, "*Getting started with WebRTC: Explore WebRTC for real-time peer-to-peer communication",* Birmingham, UK: Packt Pub. 2013.

[3] A. B. Johnston, & D. C. Burnett, "*WebRTC: APIs and RTCWEB protocols of the HTML5 real-time web".* Publ: Digital Codex LLC, ISBN-13: 978985978808 2014.

[4] S. Loreto, S. P. Romano, and L. Miniero, "*Real-Time Communication with WebRTC: Peer-to-Peer in the Browser".* O'Reilly Media, UK. ISBN 1491938080 2016.

[5] D. Ristic, "*Learning WebRTC: Develop interactive real-time communication applications with WebRTC",*. Packt Publishing ISBN: 9781783983667. 2015.

[6] S. Hudson, "Video-to-Video Using WebRTC. *JavaScript Creativity: Exploring the Modern Capabilities of JavaScript and HTML5".* Apress Berkely, CA, USA.ISBN:1430259442 9781430259442. 2014.

[7] Altanai, "*WebRTC integrator's guide: Successfully build your very own scalable WebRTC infrastructure quickly and efficiently"* Birmingham, UK: Packt Pub. 2014.

[8] S. Dutton, "Real-time communication without plugins", Available from: https://www.html5rocks.com/en/tutorials/webrtc/basics/ (Accessed: March, 2016). 2012.

[9] B. Patrik, and T. Alexandra, "The Online Paris Café", A Master thesis in Department of Computer science, Electrical and Space engineering. Luleå University of Technolog. Available from: http://www.diva-portal.org/smash/get/diva2:1031094/FULLTEXT02. 2013.

[10] A. Bergkvist, D. C. Burnett, C. Jennings, and A. Narayanan, "WebRTC 1.0: Real-time Communication Between Browsers", Available from: https://www.w3.org/TR/2012/WD-webrtc-20120821/ (Accessed: March, 2016). 2012.

[11] M. Di Mauro, & M. Longo, "Revealing Encrypted WebRTC Traffic via Machine Learning

Tools". In: *Proc. 12th International Joint Conference on e-Business and Telecommunications (ICETE) IEEE Conference Publications* 4, 259-266. 2015.

[12] D. D. Bakwa and A. E. Edim, "Application Of M.E.A.N Stack Restive API And WEBRTC In The Design Of A Real-Time Telemedicine Service" *International Journal of Innovative Research and Advanced Studies (IJIRAS)* 4(3), 311-322. 2017.

[13] B. John, "4 video chat alternatives that beat Skype", Available from: http://www.foxnews.com/tech/2015/10/28/4-video-chat-alternatives-that-beat-skype.html. (Accessed June, 2016). 2015.

[14] K. Shuang, X. Cai, P. Xu, and Q. Jia, "WebCDN: A Peer-To-Peer Web Browser CDN Based WebRTC", In: Yao L., Xie X., Zhang Q., Zomaya A., Jin H. (eds) *Advances in Services Computing. Lecture Notes in Computer Science*, Vol. 9464. Springer, Chan. 2015.

[15] S. Loreto, S. P. Romano, "*Real-time Communication with WebRTC: Peer-to-Peer in the Browser",* Pub: O'Reilly Media. UK. ISBN: 978-1-4493-7187-6. 2014a.

[16] A. Gary, "*Network Computing"* Available from: http://www.networkcomputing.com/unified-communications/9-advantages-webrtc/195325984. (Accessed May, 2016). 2014.

[17] H. W. Barz, H. W. and G. A. Bassett, "*WebRTC, in Multimedia Networks: Protocols, Design, and Applications"*, John Wiley & Sons, Ltd, Chichester, UK. doi: 10.1002/9781119090151. 2016.

[18] J. C. Zhang, W. Barnes, D. King, "Getting Started with WebRTC and Test Driven Development", Available from https://medium.com/@coldbrewtesting/getting-started-with-webrtc-and-test-driven-development-1cc6eb36ffd#.yswz9omvt (Accessed: Jan. 2017). 2016.

[19] A. W. Sime, "WebRTC: Delivering Telehealth in the Browser", *Journal of mHealth*, 1-*2* doi:10.21037/mhealth.2016.03.08. 2016.

[20] A. Arrichiello, "*Learning WebRTC application development*. Birmingham", UK: Packt Pub. 2014.

[21] M. Maruschke, O. Jokisch, M. Meszaros, M., & V. Iaroshenko, "Review of the Opus Codec in a WebRTC Scenario for Audio and Speech Communication", In: *Proc. Speech and Computer International Conference, SPECOM, Athens,* 348-355. 2015.

[22] D. Odell, Using WebRTC for Video Chat. *Pro JavaScript Development*, Apress. ISBN: 978-1-4302-6269-5. 321-339. 2014.

[23] S. Branislav, S. Dragan, & P. Dragan, "WebRTC technology overview and signaling solution design and implementation", In: *38th International Convention on Information and CommunicationTechnology, Electronics and Microelectronics (MIPRO), IEEE Conference Publications. PP.* 1006 – 1009. 2015.

[24] A. Sergiienko, "*WebRTC blueprints: Develop your very own media applications and services using WebRTC",* Birmingham, UK: Packt Pub. 2014.

[25] A. Sergiienko, *WebRTC cookbook: Get to grips with advanced real-time communication applications and services on WebRTC with practical, hands-on recipes*. UK: Packt Pub. ISBN: 9781783284450. 2015.

[26] A. Roach, "WebRTC Video Processing and Codec Requirements", Available from: https://tools.ietf.org/html/draft-ietf-rtcweb-overview. doi:10.17487/rfc7742 (Accessed June, 2016). 2016.

[27] G. Ilya, "*High Performance Browser Networking"*, First Edition. Publisher: O'Reilly Media, Inc. ISBN: 9781449344757. 2013.

[28] Y. Helsingin, T. Matemaattis-luonnontieteellinen, L. Tietojenkäsittelytieteen, & A. Hussain, "WebRTC in presence of NAT, firewalls and HTTP proxies" Thesis / Dissertation ETD. Available from: http://hdl.handle.net/10138/153590. 2015.

[29] G. P. Kedar, M. C. Pushpanjali, "WebRTC Implementation and Architecture Analysis", *International Journal of Scientific & Engineering Research*, 7(2), 2229-5518. 2016.

[30] S. Loreto, S. P. Romano, "*Real-time communication with WebRTC",* O'Reilly Media, Sebastopol, CA. 2014b.