Implementation and performance analysis of a Video Streaming System using WEBRTC and Equal Cost Load Balancing.

T.P. Fowdur, N. Anooroop

Abstract— **Real-time Internet based video communication has become a simple and widely adopted means of communication. However, the exponentially rising numbers of internet users is proving to be a real challenge to guarantee a good Quality of Service (QoS) for applications such as video communication. This paper investigates how video streaming quality is impacted by the increasing number of users using a video streaming application developed with Web Real Time Communications (WebRTC). The video streaming application considers both live video streaming from of an event captured by the initiator's webcam and live streaming of a video file from the initiator's device. Tests were performed using both cabled Ethernet and Wi-Fi connections with different types of video sources. Moreover, the use of two servers at the initiator's site where each server with equal cost load balancing was used to assess the impact of using two servers on the video quality. Videos were recorded for five minutes on both the initiator and client sides using the application's record function and the Peak Signal to Noise Ratio (PSNR) was used to assess the quality of the videos. Results showed that the use of two servers can significantly improve the PSNR by an average of for both live streaming from a webcam and live file streaming.**

Keywords—**Quality of Service, Peak Signal to Noise Ratio**

I. Introduction

Web Real-Time Communication (WebRTC) is a recent web browsing model standard and industry effort that allows browsers to exchange media directly in peer-to-peer fashion with other browsers. WebRTC is used in various apps for instance, WhatsApp, Facebook Messenger, appear.in and platforms such as TokBox [1]. According to the Cisco Visual Networking Index published in June 2017 [2], live video traffic (streaming, video conferencing) is expected to grow dramatically from three percent of Internet video traffic in 2016 to 13 percent by 2021, which amounts to 1.5 exabytes (1 exabyte = 1 million terabytes) per month in 2016, up to 24 exabytes per month in 2021. Consequently, it is becoming increasingly important to assess the Quality of Service (QoS) and Quality of Experience (QoE) of video communication systems such as WebRTC. An overview of recent works dealing with the QoS in video communications is given next.

In [3], D. Vucic and L. Skorin-Kapov investigated QoE issues in the context of mobile multi-party tele meetings via WebRTC. A series of interactive, three-party audiovisual tele meeting tests with different smartphone and laptop configurations was conducted in a natural environment. The results indicated amongst others that especially for videoconferencing on smartphone, participants had lower expectations. The authors also argued that it might be necessary to shift the processing burden (or part of it) required for multi-party tele meetings to a centralized conference media server (as opposed to a full-mesh topology), since many smartphones may not be able to meet the high CPU requirements needed to ensure a smooth QoE. In [4], a first step has been proposed to assess the quality of the video aired by a WebRTC to many viewers. The authors used the Structural Similarity (SSIM) Index [5] as a measurement of video quality for this experiment and the test aimed to measure how many viewers could join in viewing the broadcast while maintaining an acceptable quality of the image. The SSIM measure remained surprisingly stable with values in the interval [0.96, 0.97] as the number of viewers joining the broadcast increased. When the number of customers reached approximately 175, SSIM suddenly fell to values close to 0. For an increase from 1 to 175 viewers, the user experience would probably remain acceptable without loss of quality. A recent experiment has been published in [6] to evaluate the quality of WebRTC video streaming experience in the broadband mobile network. Various videos in various resolutions (from 720×480 to 1920×1080) were used between the Chrome browser and the Kurento media server as input for a video call over WebRTC. Subjectively, 28 persons gave 1 (low quality) to 5 (excellent quality) for WebRTC videos. The authors then used a number of methods to assess objectively the quality of WebRTC videos, all based on errors calculated between the original video and the WebRTC video. In [7], the authors reported on a subjective user study involving three-party audiovisual conversations established via mobile devices and aimed to investigate the impact of different video resolutions on QoE. It was also investigated how different bandwidth limitations impacted QoE for different video resolutions. The calls were established using a Web-based Real-Time Communication (WebRTC) application running on the Licode Multipoint Control Unit [8]. One of its findings stated that unlimited bandwidth setting resulted in significantly reduced user perceived overall quality for all resolutions above 480x320, meaning that the capabilities of the tested mobile phones had trouble processing multiple real-time videos with high bitrates and resolutions. Moreover, in every test with unlimited bandwidth, participants reported picture freezing, although the speech was unimpaired and communication was not completely interrupted. Further simulations showed that the amount of generated traffic had a significant influence on the QoE, especially for bitrates higher than 1200kbps for each resolution tested, which may be attributed to high demands on smartphone processing power. However, despite the lack of smartphone processing power, 5.1" screen size remained as an argument that resolutions higher than 640x480 were unnecessary for three-party video conference calls. This authors in [9] focused on the QoE assessment of WebRTC-based applications and its contribution is threefold. First, an analysis of how WebRTC topologies affect the quality perceived by users was provided. Second, a group of Key Performance Indicators (KPIs) for estimating the QoE of WebRTC users was proposed. Finally, a systematic survey of the literature on QoE assessment in the WebRTC arena was presented. The results of this review have shown that there was a preference for assessing the quality experienced by end users using QoS parameters rather than directly with QoE methods Regarding QoS, the preferred parameters to evaluate WebRTC applications were: end-to end delay, jitter, packet loss, bandwidth, throughput, and bitrate. Regarding QoE, subjective MOS was the preferred assessment method, followed by several objective methods (PESQ, PEVQ, VQM, PSNR, and SSIM). Furthermore, the congestion control algorithm, implemented at the application layer in the WebRTC stack [10] adapt the WebRTC transmission bitrate when detecting network congestion thus, leading to play an important role in the total amount of end-to-end delay in the WebRTC. The authors in [11] proposed to create and implement a WebRTC hybrid signaling mechanism named (WebNSM) for video conferencing based on the Socket.io (API) mechanism and Firefox. WebNSM was designed over a combination of different topologies, such as simplex, star and mesh. Therefore, it offered several communications at the same time as one-to-one (unidirectional/bidirectional), one-to-many (unidirectional) and many-to-many (bi-directional) without any downloading or installation. WebRTC video conferencing was accomplished via LAN and WAN networks, including the evaluation of resources in WebRTC like bandwidth consumption, CPU performance, memory usage, Quality of Experience (QoE) and maximum links and RTPs calculation. Furthermore, a novel signaling mechanism was built among different users, devices and networks to offer multi-party video conferencing using various topologies at the same time, as well as other typical features such as using the same server, determining room initiator, keeping the communication active even if the initiator or another peer left, etc. Results showed that mesh topology [12] limited the number of peers as it requested high CPU and high bandwidth speed; as high as the CPU core, it would lead to allow more peers to join, better communication and encoding & decoding. It is clear from previous works such as [4] and [6] that the assessment of the QoE with regards to WebRTC is very important. However, previous works have not provided the actual PSNR of videos at their source and their destination with an elevated number of users. Moreover, research has not focused on the type of internet connection used; this can be taken a step further by evaluating video quality on Wi-Fi connection as well as Ethernet cable connection.

In line with the previous works on assessing the QoS of WebRTC, this paper first describes the implementation of a video conferencing system that can both stream live videos and videos from a file to multi-client. The QoS obtained at the client end from this system is then assessed using …

The organization of this paper is as follows. Section ….

II. COMPONENTS OF A WEBRTC SYSTEM

WebRTC is a new standard that extends the web browsing model so that browsers are able to directly exchange real-time media with other browsers in a peer-to-peer fashion. The goal is to define a WebRTC API that enables a web application to exchange real-time media and data with a remote party in a peer to-peer fashion.

WebRTC extends the client-server semantics by introducing a peer-to-peer communication paradigm between browsers. The most common WebRTC scenario is the one where both browsers are running the same web application, downloaded from the same web page. The signaling between the browser and server is not standardized in WebRTC, as it is considered to be part of the application as shown in Figure 1.

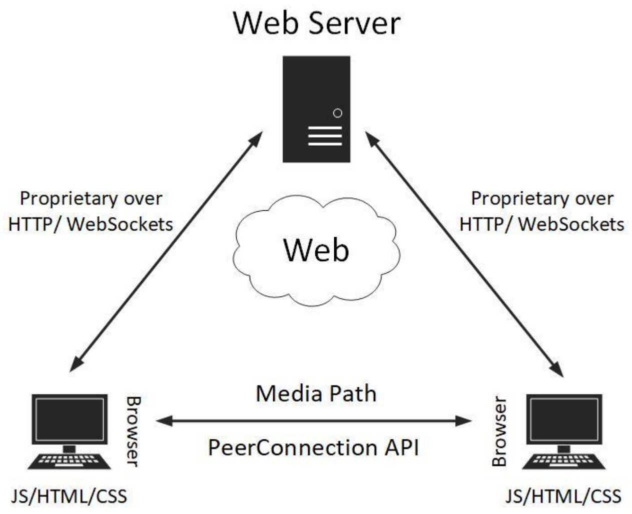


Figure 1: The WebRTC Triangle

WebRTC implements three APIs:

- MediaStream (aka getUserMedia)
- RTCPeerConnection
- RTCDataChannel

The MediaStream API enables users to have access to media streams, such as cameras and microphones, from local entry terminals. The getUserMedia API [13] was first focused on but has now been officially developed as the MediaStream API. However, the getUserMedia () method is still the main way to start access for local input devices.

MediaStream API has three key points of functionality:

- It gives a developer access to a stream object that represent video and audio streams
- It manages the selection of input user devices in case a user has multiple cameras or microphones on his device
- It provides a security level asking user all the time he wants to fetch the stream

The RTCPeerConnection API is the core of the peer-to-peer connection between each WebRTC enabled browser or peer. Since WebRTC is relatively new, peer connection has various implementation systems [14] for different browsers to make sure that it works in Firefox, Chrome and Opera, and when creating an object, it needs to take into consideration various function calls when creating an object; In Firefox it is called mozRTCPeerConnection, and in Chrome its webkitRTCPeerConnection. To create an object RTCPeerConnection, var peerconnection = RTCPeerConnection (configuration); constructor is used. The peerconnection object is then used in slightly different ways on each client, depending on the caller or the callee.

RTCDataChannel enables the arbitrary exchange of data between two peers with customized transport delivery characteristics. Each application using this channel can configure it to provide: a reliable or partially confident delivery of the messages transmitted and in-order or out-of-order delivery of sent messages. It is a bidirectional channel, which is part of the PeerConnection API. When using the channel, the PeerConnection object is called with createDataChannel ("channel name) "to create a dataChannel with a specific name or it is received in an existing PeerConnection channel event of the type RTCDataChannelEvent [15].

III. Video Streaming Application with WebRTC

A. Signaling Server

Signaling is the communication coordination process. To se t up a ' call ' for a WebRTC application, its customers exchang e information such as session control message for opening or closing communication, network data, such as the IP address and port of a host as seen by the outside world, key data used to create connection, media metadata such as codecs and codec settings, bandwidth and types of media and finally error messages.

The server is implemented using Node.js and is run by typing command ‘npm start’ on the project’s directory. After typing that command, the application is displayed, ready for use on ‘https://localhost:3000’

B. The Application HTML page

The application’s HTML page is the first thing displayed upon entering the appropriate URL. It offers two main options namely WebRTC Video Conferencing and WebRTC Video Streaming and consists of two input boxes and four action buttons. Almost all of its functions use basic JavaScript and HTML codes for their jobs are only to read the data into the input box and redirecting the user to the appropriate room. The only exception is the ‘Choose File’ option which uses a simple library [16] called ‘socket.io-file-client’ for its job is to browse a video file from the user’s device for broadcasting. Figure 2 shows how the homepage looks and specifies the uses of its components.

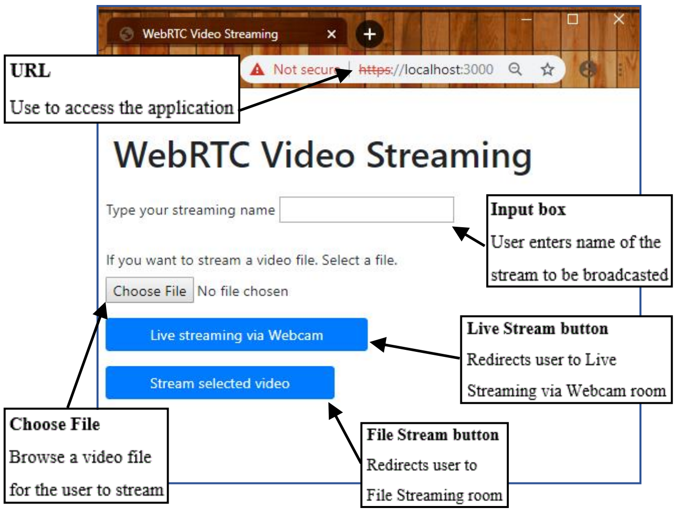

WebRTC Video Streaming
Not secure ~~https~~://localhost:3000
URL
Use to access the application
WebRTC Video Streaming
Type your streaming name
Input box
User enters name of the stream to be broadcasted
If you want to stream a video file. Select a file.
Choose File No file chosen
Live streaming via Webcam
Live Stream button
Redirects user to Live Streaming via Webcam room
Stream selected video
Choose File
Browse a video file for the user to stream
File Stream button
Redirects user to File Streaming room

Figure 2: Application’s HTML Page functions

C. Video Streaming Application

The general structure for the WebRTC Video Streaming application is shown in Figure 3.

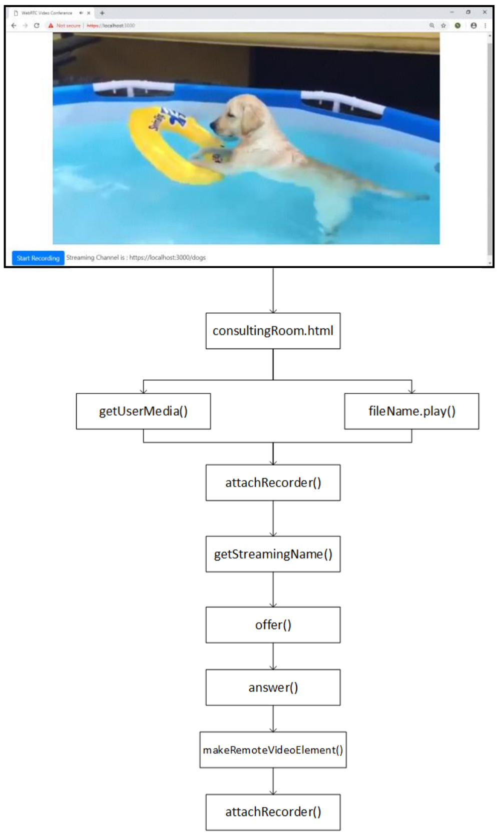


Figure 3: Structure of WebRTC Video Streaming

The steps in Figure 3 are as follows:

1. The initiator does a getUserMedia () request or selects a local video to access its video Mediastream and adds it to a new RTCPeerConnection object
2. The client creates an (SDP) offer using the generated RTCPeerConnection
3. The offer is set as the peer connection's localDescription and sent to the client via the signaling server
4. Upon receiving the offer, the initiator creates a RTCPeerConnection and adds it as a remoteDescription
5. The initiator creates an (SDP) answer from its RTCPeerConnection, adds this as its localDescription and sends this answer to the client via the signaling server
6. The client receives the answer and sets it as its remoteDescription and gets the video of the initiator

Now that both peers have exchanged each other's SDP's, a direct connection between both peers can be set up. Conversely to video conferencing, the media will be transmitted in only one way i.e. only from the initiator to the client.

The first thing to do before setting up a broadcast is to enter a stream name on the homepage. Afterwards, a broadcast can be set up by either Live Streaming or File Streaming. Live Streaming allows users to broadcast their live preview from their webcam and it can be done by simply clicking on the 'Live Streaming via Webcam' button. File Streaming on the other hand requires the user to browse a video file first via the 'Choose File' button and broadcast it by clicking on the 'Stream selected video' button. After selecting one of the two options, the user is presented with the streaming room which consists of the selected video or the webcam preview, a recorder and also a link which redirects clients to the broadcast directly.

As mentioned above, a specific link is required for clients to join the stream directly. This block, contained by getStreamingName () does the following:

1. Get the streaming name
2. Add a '/' at the end of the URL
3. Add the streaming name just after the '/' to form the link

A. Multi-Client connection

In order to set up communication among various users, a star topology [17], as shown in Figure 4 is used.

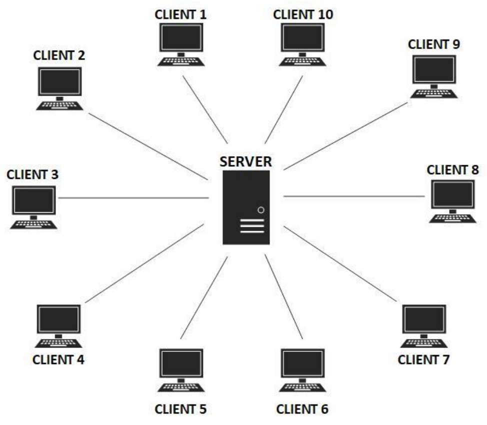
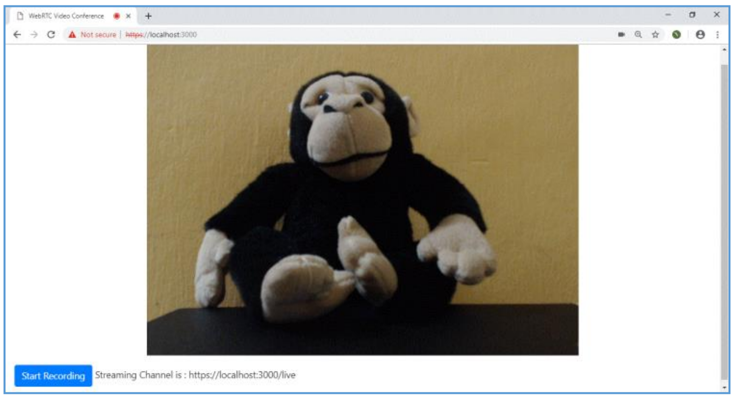


Figure 4: Star Topology

In the star topology, all clients are connected to the initiator who acts like a server in that case. After the offer/answer negotiations are completed, individual peer connections are created between the initiator and the clients.

B. Live Streaming

Live streaming provided users the facility to share the preview of their local video and it is very similar to video conferencing since both uses the local preview from the local camera. The only difference is that video transfer is unidirectional with video streaming and bi-directional with video conferencing. Figure 5 shows the streaming room of the initiator.



Streaming Channel is : https://localhost:3000/live

Figure 5: Streaming room of initiator (Live Stream)

The initiator shares to the clients the link at the bottom of the screen which is generated by adding '/streamingName' at the end of the URL of the initiator as mentioned by function getStreamingName (). However, the initiator must also share the IP address, which should replace 'localhost' thus altering the link to https://<IP_address>:3000/live. Figure 6 shows the streaming room of the client.

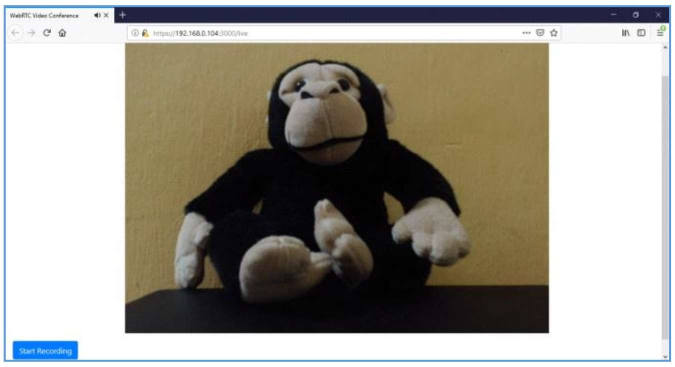
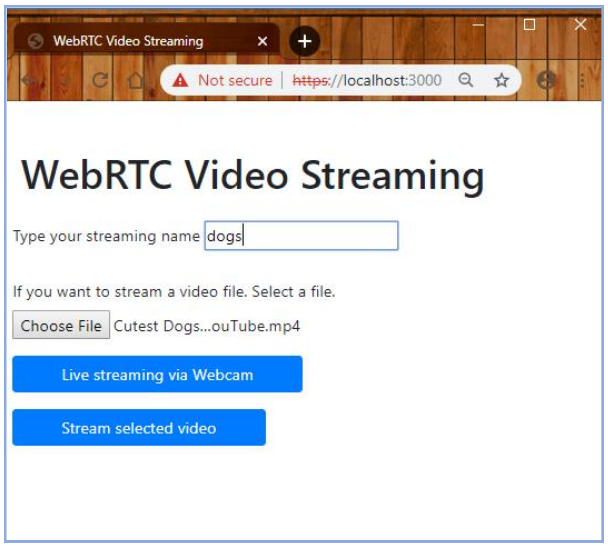


Figure 6: Streaming room of client (Live Stream)

C. File Streaming



WebRTC Video Streaming

~~https~~://localhost:3000

Not secure

WebRTC Video Streaming

Type your streaming name dogs

If you want to stream a video file. Select a file.

Choose File Cutest Dogs...ouTube.mp4

Live streaming via Webcam

Stream selected video

Figure 7: Selecting video to broadcast

File streaming includes broadcasting a video file to various client. File Streaming is almost identical to live streaming; the single difference comes from the type of video each one is broadcasting. Figure 7 shows a video file being uploaded, ready for broadcast.

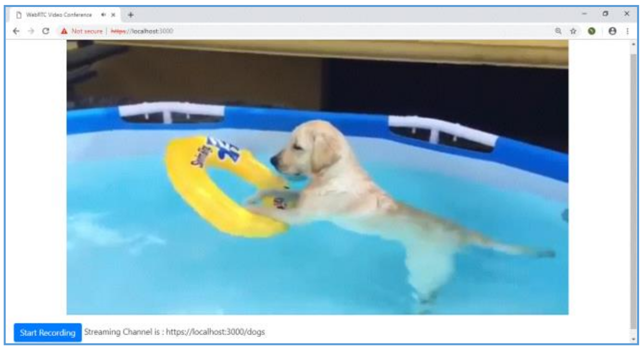


Figure 8: Streaming room of initiator (File Stream)

a) Getting the local media

This block requests the user's local video as soon as the latter enter the room. One of the main APIs in WebRTC namely, Mediastream is used for accessing the local camera. The function used namely, getUserMedia() can only be called from an HTTPS URL, localhost or a file://URL.

b) Making an offer

In order for the video of the initiator to appear on the page of the receiver, a negotiation process takes place which included sending on offer to the initiator. The receiver executes the next block namely, offer().

c) Making an answer

Similarly to the previous section, a negotiation process takes place for the local video of the receiver to appear on the initiator's page in the answer() block.

d) Recording the streams

This block is responsible for recording any video displayed in the conference room. Recording streams requires no help from the signalling server however, it does need a library namely 'RecordRTC.min' library and it is represented by the attachRecorder() function which is executed when the user:

```
1. Clicks on 'Start Recording' button
2. Recorder sets maximum recording time,
     maxRecordingTime  to 300000ms (5 minutes)
3. Recording starts
4. Hide 'Start Recording' button
5. Display 'Stop Recording' button
6. if maxRecordingTime == 300000
     a. Recording stops
     b. Hide 'Stop Recording' button
     c. Display 'Download Recording' button
     d. Click on 'Download Recording' button
     e. Downloading starts
     f. Hide 'Download Recording' button
     g. Display 'Start Recording' button
7. else
     a. Click on 'Stop Recording' button
     h. Recording stops
     b. Downloading starts
     c. Hide 'Stop Recording' button,
     d. Display 'Start Recording' button
```

e) Getting the remote video

Using the makeRemoteVideoElement() function, this next block helps in displaying the remote video i.e. the local video of the other peer. In addition to displaying this remote video, it also shapes the size in which the video will appear. These steps are described below:

```
1. Set width to zero
2. Set height to zero
3. if the video conference application is being used.
4.      width = (width of window / 2) -50
5.      height = (height of room /2) – 50
6. end if
7. Display remote video with the calculated attributes
```

Below is a complete algorithm of the video streaming application

```
1.  User input streaming Name, streamingName
2.  User uploads a video file
3.  if streamingName = “”
4.        alert “Please type a room number”
5.        User input streamingName
6.  else if file is not found
7.        alert "Please select a video file"
8.        User uploads a video file
9.        else
10.             Get streamingName
11.             Get video file
12.       end if
13.end if
14.Hide Homepage
15.Display Conference Room
16.Plays selected file
17.Attach recorder to local video
```

In order for the video of the initiator to appear on the client’s page, the following processes occurred:

1. Client
   a. Creates a peer connection object,for initiator
   b. Adds onicecandidate event handler to peer connection object
   c. Sends ice candidates to initiator,
   d. Creates offer
   e. Ties local description to peer connection object
   f. Sends offer to initiator
2. Initiator
   a. Receives offer from client
   b. Creates a peer connection object for client
   c. Sets remote description of receiver to peer connection
   d. Adds onicecandidate event handler to peer connection object
   e. Sends ice candidates to initiator
   f. Adds local stream to peer connection object
   g. Creates answer
   h. Ties local description to peer connection object
   i. Sends answer to client,
3. Client
   a. Receives answer and sets remote description of initiator to peer connection object
   b. Stores ice candidates in peer connection object
   c. Adds video of initiator to peer connection object by triggering the onaddstream event handler
   d. Displays video of initiator
   e. Attaches recorder to video of initiator

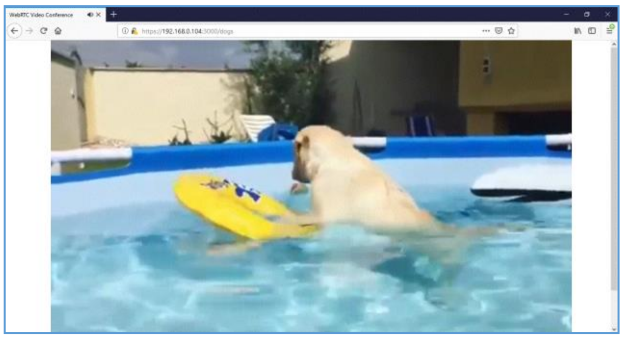


Figure 9: Streaming room of client (File Stream)

IV. System Testing and Performance Analysis

The video streaming application, which includes live stream and file stream, was assessed with up to ten clients, again using Wi-Fi connection first and then using Ethernet cable connection. Finally, the variation in video quality was evaluated using two servers with five clients each compares to using only one server with ten clients. All tests were done on the same LAN and the PCs, the router as well as the internet speed, which was 10Mbps were kept constant during the experiment. The videos of all users were recorded for five minutes and their quality was evaluated using the PSNR metric [18]. With video streaming, one user had set up the broadcast and the stream were compared to the other streams obtained by the other users. The downloaded videos were analyzed and compared using MATLAB.

Experimental Setup with 2 Servers:
Two servers were used simultaneously on two different PCs assigned with five clients each. Such that the first server takes the first five clients from client 1 to client 5 and the second server takes the next five clients from client 6 to client 10.

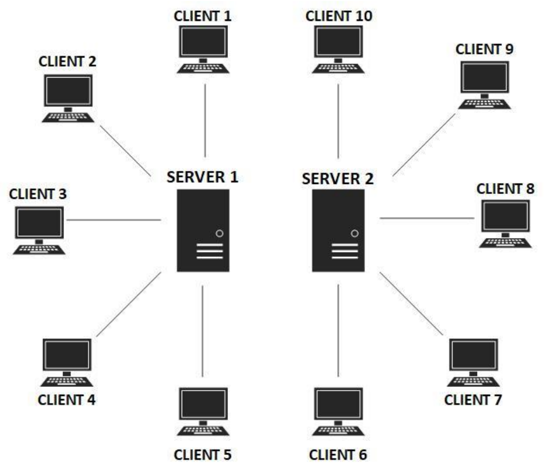
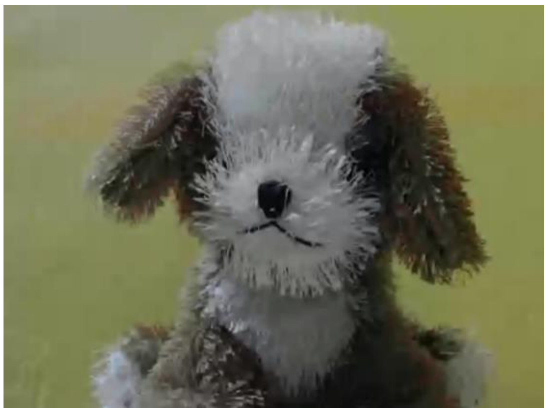


Figure 10: Star Topology with two servers

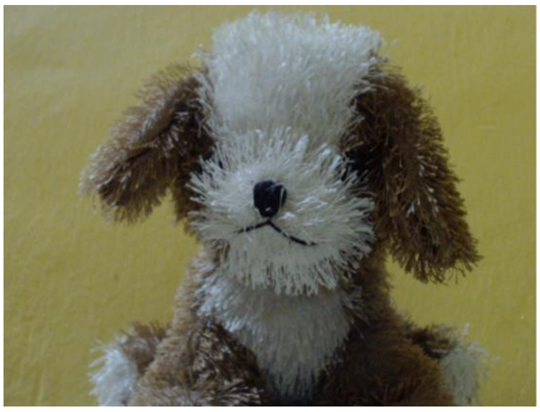
A. Result of video streaming with 2 servers using Wi-Fi

i. Live Stream quality with two servers

Figure 11 shows the comparison between the tenth client using one server and the tenth client using two servers



(a) Tenth client using one server



(b) Tenth client using two servers

Figure 11: Snapshot of Live Stream using Wi-Fi

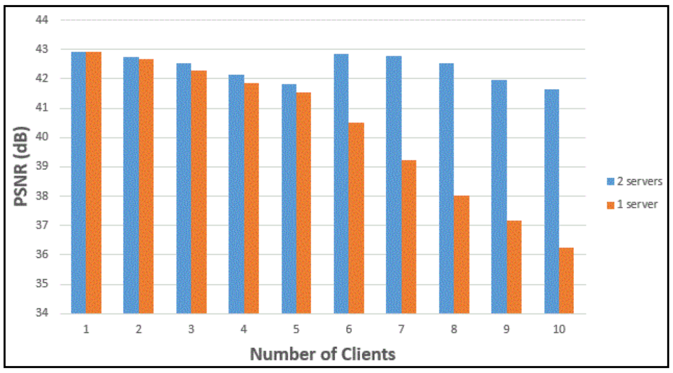
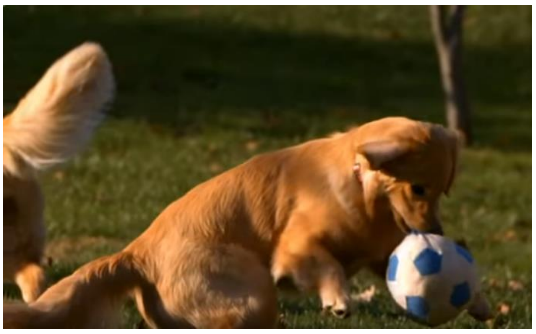


Figure 12: Live Stream Quality with two servers (Wi-Fi)

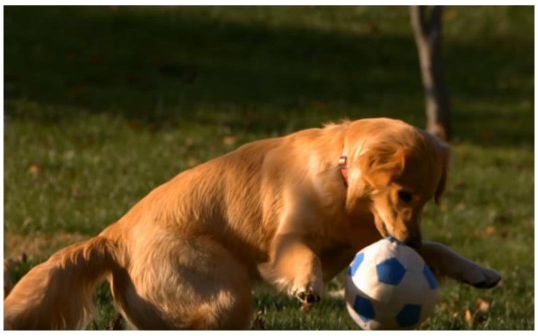
Figure 12 shows the comparison between using one server and two servers for a total of ten clients. As expected, the general trend shows a degradation of PSNR with increasing number of clients. For the first five clients, there is no major difference between the PSNR whether one or two servers are used because in this scheme the second server starts accepting clients only after the first server has accepted the first five clients. Hence as from the sixth client, the use of two servers starts to impact the PSNR. As such for the tenth client the PSNR is 36.2482dB with one server compared to 41.6325dB with two servers.

ii. File Stream quality with two servers (Regular Video)

Figure 4.31 shows the comparison between the tenth client when using one server and two servers respectively.



(a) Tenth client using one server



(b) Tenth client using two servers

Figure 13: Snapshots of File streaming using Wi-Fi

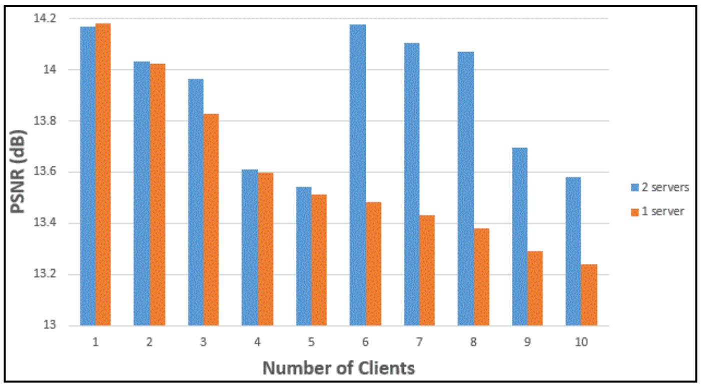


Figure 14: File Stream Quality with two servers (Wi-Fi)

Figure 14 shows the comparison between using one server and two servers for a total of ten clients. In this case for the tenth client the PSNR is 13.2406 dB with one server compared to 13.5823 dB with two servers.

B. Result of video streaming with 2 servers using Ethernet Cable

i. Live Stream quality with two servers

Figure 15 shows the comparison between the first and the tenth client for when using two servers with five clients each.

(a) Tenth client using one server

(b) Tenth client using two servers

Figure 15: Snapshots of Live streaming with Ethernet Cable

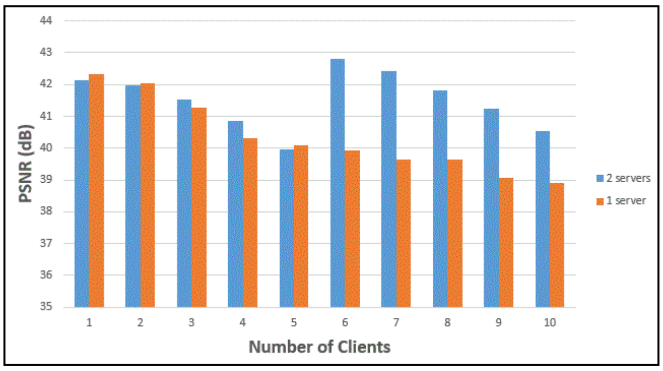


Figure 16: Live Stream Quality with two servers (Ethernet Cable)

Figure 16 shows the comparison between using one server and two servers for a total of ten clients. In this case for the tenth client the PSNR is 38.9105 dB with one server compared to 40.5422 dB with two servers.

ii. File Stream quality with two servers (Regular Video)

Figure 4.37 shows the comparison between the first and the tenth client for when using two servers with five clients each.

(a) Tenth client using one server

(b) Tenth client using two servers

Figure 17: Snapshots of File streaming with Regular Video (Ethernet Cable)

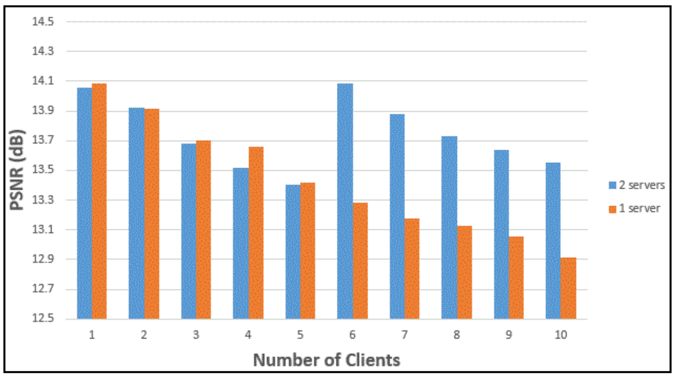


Figure 18: Live Stream Quality with two servers (Ethernet Cable)

Figure 18 shows the comparison between using one server and two servers for a total of ten clients. In this case the value of PSNR with two servers was 13.5562 dB compared to 12.9112 dB with one server.

V. CONCLUSION

ACKNOWLEDGMENT

REFERENCES

[1] "TokBox," [Online]. Available: https://en.wikipedia.org/wiki/TokBox. [Accessed 30 March 2019].

[2] "VNI Global Fixed and Mobile Internet Traffic Forecasts," Visual Networking Index, Cisco, 2017. [Online]. Available: https://www.cisco.com/c/en/us/solutions/service-provider/visual-networking-index-vni/index.html.

[3] D. Vucic and L. Skorin-Kapov, "The impact of mobile device factors on QoE for multi-party video conferencing via WebRTC," Telecommunications (ConTEL), 2015.

[4] B. Garcia et al., "WebRTC Testing: State of Art," WebRTC Testing: Challenges and Practical Solutions, 2017.

[5] Wang et al., "Image Quality Assessment: From Error Visibility to Structural Similarity," 2004.

[6] Yevgeniya Sulema et al., "Quality of Experience Estimation for WebRTC-based Video Streaming," 2018.

[7] Lea Skorin-Kapov, Mirko Suznijevi, "The impact of bandwidth limitations and video resolution size on QoE for WebRTC-based mobile multi-party video conferencing," 2016.

[8] T. Team, "Multipoint Control Unit (MCU)," [Online]. Available: https://trueconf.com/blog/wiki/multipoint-control-unit. [Accessed 2019 January 2019].

[9] Boni Garc__a _ Micael Gallego _ Francisco, "Understanding and Estimating Quality of Experience in WebRTC Applications," 2018.

[10] "Architecture," [Online]. Available: https://webrtc.org/architecture/. [Accessed 26 March 2019].

[11] Naktal Moaid Edan, A. Al-Sherbaz, Scott John Turner, "Design and implement a hybrid WebRTC Signalling mechanism for unidirectional & bi-directional video conferencing," 2018.

[12] "Mesh," [Online]. Available: https://webrtcglossary.com/mesh/. [Accessed 30 January 2019].

[13] A. D. Rosa, "An Introduction to the getUserMedia API," 02 January 2014. [Online]. Available: https://www.sitepoint.com/introduction-getusermedia-api/.

[14] "RTCPeerConnection," 18 March 2019. [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/API/RTCPeerConnection.

[15] K.T.Tokeshu, Moppers, "RTCDataChannel," May 2015. [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/API/RTCDataChannel.

[16] "JavaScript libraries," [Online]. Available: https://webplatform.github.io/docs/concepts/programming/javascript/libraries/. [Accessed 1 March 2019].

[17] "Star topology," Telecom ABC, [Online]. Available: http://www.telecomabc.com/s/star.html. [Accessed 6 January 2019].

[18] "PSNR," MathWorks, [Online]. Available: https://www.mathworks.com/help/vision/ref/psnr.html. [Accessed 5 January 2019].