



# **WEBRTC based Broadcasting for Physical Surveillance**

Bih Fei Jong

Faculty of Engineering, Computing and Science  
Swinburne University of Technology  
Sarawak Campus, Malaysia

Submitted for the degree of Master of Engineering (by Research)

2017

Dedicated to  
*My parents and all my friends.*  
*Without them, none of my success would be possible.*

# Abstract

Camera monitoring systems have been highly demanded and wide deployed. The need of a comprehensive video recorder that is able to handle large amount of video data has greatly increased nowadays. Hence, this research was conducted to develop a real-time streaming solution dedicated for physical surveillance system. A Network Video Recorder which involves the latest streaming technology, WebRTC was designed and implemented.

WebRTC is a free, open source project that provides streaming service to browsers and mobile devices. As this research seeks for a streaming server that is light weighted and most importantly providing native web browser support, WebRTC is the best selection. The integrated design and implementation is discussed in this thesis.

In addition, a series of experiments were conducted to measure the QoE of this NVR system. Results of the real time streaming server are compared with FFserver. Besides having a normal PC as the end user, this thesis also included mobile platform (Android). The video streaming performance of both PC and mobile client are compared. Future research suggested in this thesis is to improve the video transcoding process through the use of hardware acceleration.

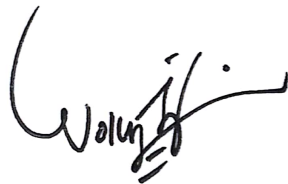
# Acknowledgements

Upon completion of this thesis, I would like to thank those who have guided me and lent me a hand throughout the course of my study. First of all, I would like to extend my heartfelt gratitude to my supervisor Dr. Dennis Wong for the continuous support of my research, for his patience, encouragement and useful advice. His guidance helped me in doing all research and experiment.

My sincere thanks also goes to Mr. Zhi Hao Chang and my fellow lab mates for their help and suggestions. Without their precious support it would not be possible to conduct this study. Last but not least, I would like to thank my family: my parents and sisters for motivating me mentally throughout the whole thesis writing period.

# Declaration

I declare that this thesis contains no material that has been accepted for the award of any other degree or diploma and to the best of my knowledge contains no material previously published or written by another person except where due reference is made in the text of this thesis.

A handwritten signature in black ink, appearing to read 'Bih Fei Jong', with a stylized flourish extending to the right.

Bih Fei Jong

# Publications Arising from this Thesis

[1] Z. H. Chang, B. F. Jong, W. J. Wong, M. L. D. Wong, "Distributed Video Transcoding on a Heterogeneous Computing Platform", IEEE APCCAS 2016, Jeju Island, Korea, October, 2016. pp. 444-447

[2] B. F. Jong, Z. H. Chang, W. J. Wong, M. L. D. Wong, and S. K. D. Tang, "A WebRTC based ONVIF Supported Network Video Recorder", ICISCA 2016, Chiang Mai, Thailand, July 2016. pp. 64-67

# Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
1.1	Research Objectives . . . . .	2
1.2	Thesis Structure . . . . .	3
<b>2</b>	<b>Background</b>	<b>4</b>
2.1	Analogue vs Digital Camera Surveillance Systems . . . . .	4
2.2	Streaming Technologies . . . . .	5
2.3	Web Real-Time Communication . . . . .	8
2.3.1	Structure . . . . .	9
2.3.2	WebRTC in the Web Browser . . . . .	11
2.3.3	WebRTC Communication Flow . . . . .	11
2.3.4	WebRTC APIs . . . . .	13
2.3.5	STUN, TURN and ICE . . . . .	14
2.3.6	Signaling . . . . .	17
2.4	Case Studies . . . . .	18
2.4.1	GSM Based Smart Home Security System . . . . .	18
2.4.2	IP based Surveillance System . . . . .	19
2.4.3	Cloud-based Multimedia Surveillance System . . . . .	21
2.4.4	Video Surveillance in Wireless Router . . . . .	23
2.4.5	Video Surveillance System and Facility to Access PC from Remote Areas Using Smart Phone . . . . .	24
<b>3</b>	<b>System Proposal and Experimental Setup</b>	<b>27</b>
3.1	NVR System Architecture . . . . .	27
3.1.1	Camera Discovery and Management . . . . .	28



## CONTENTS

3.1.2	Recording and Data Storing . . . . .	29
3.1.3	Web Server . . . . .	29
3.2	WebRTC Real-time Video Streaming . . . . .	30
3.2.1	Transcoder . . . . .	30
3.2.2	WebRTC Media Gateway . . . . .	32
3.3	WebRTC Web Server . . . . .	33
3.4	Methodology and Experimental Setup . . . . .	38
3.4.1	Performance Measures . . . . .	39
3.4.2	Network Topology . . . . .	41
3.4.3	Hardware Specifications . . . . .	41
3.4.4	Software Configuration . . . . .	44
3.4.5	Data Acquisition . . . . .	46
3.4.6	Video Quality Assessment . . . . .	50
<b>4</b>	<b>Results and Discussion</b>	<b>54</b>
4.1	General Real Time WEBRTC Streaming Performance . . . . .	54
4.2	WEBRTC versus FFserver . . . . .	61
4.2.1	CPU and Memory usage . . . . .	63
4.3	Result from Mobile Client . . . . .	66
4.3.1	Android . . . . .	66
4.4	Video Quality Assessment . . . . .	69
4.4.1	PSNR . . . . .	69
4.4.2	SSIM . . . . .	71
4.4.3	VQM . . . . .	71
4.5	Streaming from Standard Video File . . . . .	73
<b>5</b>	<b>Conclusions and Future Works</b>	<b>79</b>
5.1	Conclusions . . . . .	79
5.2	Hardware Acceleration Support . . . . .	80
	<b>References</b>	<b>81</b>

# List of Figures

2.1	Analog CCTV System [1] . . . . .	5
2.2	IP Camera System [1] . . . . .	6
2.3	WebRTC Triangle [2] . . . . .	9
2.4	WebRTC Trapezoid [2] . . . . .	10
2.5	WebRTC Communication Flow in Browser [2] . . . . .	11
2.6	General WebRTC call flow . . . . .	12
2.7	STUN P2P Communication [2] . . . . .	15
2.8	TURN P2P Communication [2] . . . . .	16
2.9	JSEP signaling and media flow [3] . . . . .	17
2.10	Schematic design of PC based IP surveillance security [4] . . . . .	19
2.11	(a): Flow chart of add/select view operation, (b): Flow chart of add/select camera operation [4] . . . . .	20
2.12	Cloud based multimedia surveillance framework [5] . . . . .	22
2.13	Video Surveillance Architecture [6] . . . . .	23
2.14	System Design of the Video Surveillance System and Facility to Access Pc from Remote Areas Using Smart Phone [7] . . . . .	24
3.1	Block diagram of the NVR system . . . . .	27
3.2	Block diagram of the live streaming module . . . . .	30
3.3	Block diagram of Transcoder . . . . .	31
3.4	WebRTC gateway and different technologies . . . . .	32
3.5	WebRTC Live Streaming Flow Chart . . . . .	34
3.6	Experiment topology . . . . .	41
3.7	General process flow of the topology . . . . .	43

## LIST OF FIGURES

3.8	Ping rate from client to server . . . . .	44
3.9	Traceroute result . . . . .	45
3.10	IPerf result . . . . .	45
3.11	A snapshot of webrtc-internals . . . . .	47
3.12	A snapshot of Firefox about:webrtc . . . . .	48
3.13	Wireshark GUI snapshot . . . . .	50
4.1	Graph of Bit rate (Axis Camera) . . . . .	55
4.2	Graph of Bit rate (Hikvision Camera) . . . . .	55
4.3	Graph of Packet rate (Axis Camera) . . . . .	56
4.4	Graph of Packet rate (Hikvision Camera) . . . . .	57
4.5	Graph of Packet loss (Axis Camera) . . . . .	57
4.6	Graph of Packet loss (Hikvision Camera) . . . . .	58
4.7	Received Frame rate (Axis Camera) . . . . .	59
4.8	Received Frame rate (Hikvision Camera) . . . . .	60
4.9	Jitter and Delay (Axis Camera) . . . . .	60
4.10	Jitter and Delay (Hikvision Camera) . . . . .	60
4.11	WebRTC Bitrate measured by Wireshark (Axis) . . . . .	61
4.12	WebRTC Bitrate measured by Wireshark (Hikvision) . . . . .	62
4.13	FFserver Bitrate measured by Wireshark (Axis) . . . . .	62
4.14	FFserver Bitrate measured by Wireshark (Hikvision) . . . . .	63
4.15	WebRTC CPU Usage . . . . .	64
4.16	FFserver CPU Usage . . . . .	64
4.17	Total Memory Usage . . . . .	65
4.18	Snapshot from Android device . . . . .	66
4.19	Bit Rate(Android) . . . . .	67
4.20	Packet Rate(Android) . . . . .	68
4.21	Frame Rate(Android) . . . . .	68
4.22	Jitter and Delay (Android) . . . . .	69
4.23	PSNR (Axis) . . . . .	70
4.24	Left: Original Frame versus Right: Streamed Frame . . . . .	70

## LIST OF FIGURES

4.25 SSIM (Axis) . . . . .	71
4.26 VQM (Axis) . . . . .	72
4.27 VQM of H.264 versus VP8 [8] . . . . .	72
4.28 Bitrate (Big Buck Bunny 480P) . . . . .	74
4.29 Jitter and Target Delay (Big Buck Bunny 480P) . . . . .	74
4.30 Bitrate (Big Buck Bunny 720P) . . . . .	75
4.31 Jitter and Target Delay (Big Buck Bunny 720P) . . . . .	75
4.32 Bitrate (Big Buck Bunny 1080P) . . . . .	75
4.33 Jitter and Target Delay (Big Buck Bunny 1080P) . . . . .	76
4.34 Bitrate (Big Buck Bunny) . . . . .	77
4.35 Packet Loss (Big Buck Bunny) . . . . .	77
4.36 Frame Rate (Big Buck Bunny) . . . . .	78

# List of Tables

3.1	Server Specifications . . . . .	42
3.2	IP Camera Specifications . . . . .	42
3.3	Client Specifications . . . . .	43
3.4	Switch Specifications . . . . .	43
3.5	Router Specification . . . . .	44
3.6	IP Camera Configurations . . . . .	46
3.7	WebRTC Internal Tool: Chrome versus Firefox . . . . .	49
4.1	WebRTC Frame rate . . . . .	58
4.2	WebRTC versus FFserver: Bit rate . . . . .	61
4.3	CPU Usage . . . . .	65
4.4	Stream Quality Measurement (Android) . . . . .	67
4.5	Mapping PSNR to MOS [9] . . . . .	70
4.6	Big Buck Bunny Video files Specifications . . . . .	73
4.7	Streaming Result from PC Client . . . . .	74
4.8	Streaming Result from Android Client . . . . .	76

# Listings

3.1	FFmpeg command . . . . .	31
3.2	Function to check WEBRTC availability . . . . .	35
3.3	Media Handler . . . . .	35
3.4	Stream Handler 1 . . . . .	37
3.5	Stream Handler 2 . . . . .	37

# Commonly Used Acronyms

**API** Application Programming Interface.

**CCTV** Closed-Circuit Television.

**CPU** Central Processing Unit.

**FPS** Frame Per Second.

**HTTP** Hypertext Transfer Protocol.

**ICE** Interactive Connectivity Establishment.

**ICMP** Internet Control Message Protocol.

**IP** Internet Protocol.

**JSON** JavaScript Object Notation.

**LAN** Local Area Network.

**NAT** Network Address Translation.

**NVR** Network Video Recorder.

**ONVIF** Open Network Video Interface Forum.

**PoE** Power over Ethernet.

**PTZ** Pan Tilt Zoom.

**QoE** Quality of Experience.

**RTCP** RTP Control Protocol.

**RTP** Real-time Transport Protocol.

## COMMONLY USED ACRONYMS

**RTT** Round Trip Time.

**SDP** Session Description Protocol.

**SIP** Session Initiation Protocol.

**SRTP** Secure Real-time Transport Protocol.

**TCP** Transmission Control Protocol.

**UDP** User Datagram Protocol.

**WebRTC** Web Real-Time Communication.

**XML** Extensible Markup Language.



# Introduction

The necessity of digital video surveillance system has increased due to alarming increasing crime rate around the globe. Researches have proven the effects of video surveillance on crime for both public and home used. In 2011, La Vigne [10] conducted a research in measuring the effects of video surveillance on crime. Experiments were carried out in three cities (Baltimore, Chicago, and Washington, D.C.) in United States. The result showed the importance of video surveillance as a useful tool for preventing crimes and supporting investigations. It was also proven that savings from the reduced crime rates are far greater than the surveillance costs.

A basic video surveillance system includes video cameras, sensors, terminals, and servers. Some general functions include remote monitoring, live viewing, device control, video storage, and input output control. Network Video Recorder (NVR) is a camera managing software that runs on a dedicated, standalone device and has improved performance and advanced features enabling us to handle a larger database with more comprehensive video processing. Henceforth, with improved technologies, the surveillance system has been equipped with even more functions such as face recognition, object identification and motion detection.

This thesis discovers possibilities of enhancing network surveillance video recording system with the latest media streaming technology. Since the development of communication technologies, networking, and computing, video surveillance system has evolved from the traditional analogue Closed-Circuit Television (CCTV) camera to the digitalized Internet Protocol (IP) based camera. It is crucial for a NVR system to manage enormous numbers of IP camera and deliver these video data to the end user with as much delay as possible. As such, this research introduces the concept and system architecture of integrating WebRTC and NVR system.

## 1.1 Research Objectives

The main focus of this thesis is to develop and implement a streaming module designed particularly for the network video recorder system. After reviewing several available streaming technologies, it is found that Web Real-Time Communication (WebRTC) has the space for further improvement and yet to mature and it also fulfilled the requirements below. There are several key requirements in choosing the desired streaming technology and server for this system:

- (i) Include native web browser support.
- (ii) Supports as much devices as possible, for example, PC, tablets, Android, and IOS phones.
- (iii) Ensure high quality and low delay output particularly for real-time streaming.
- (iv) Has to be open-source, meaning easy to be adapted.

Hence after making sure which streaming server to implement, research objectives are created:

- (i) To design a NVR system that is able to communicate with ONVIF compliant IP cameras.
- (ii) The NVR system has to include WebRTC streaming which streams the video feed obtained from IP cameras in a real-time manner.
- (iii) Since there is no standardized method in interoperating the transport protocol for IP cameras and WebRTC, a bridging method is needed.
- (iv) After developing and deploying the server, experiments are needed to measure the performance of WebRTC streaming. Besides, another streaming server, FF-server is used to compare with WebRTC.

With the above conditions in head, a NVR system is developed and implemented (details in chapter 3), integrating WebRTC and a surveillance NVR system which had yet be done. Moreover, to ensure quality of experience, various experiments are carried out.

## 1.2 Thesis Structure

The remaining sections of the thesis are as follows. Chapter 2 introduces the several streaming technologies and servers, the main technology used, WebRTC, and its background. Later of the chapter describes past related work. Chapter 3 includes the design of the entire system and how WebRTC works as a streaming protocol in my NVR. The methodology and experimental setup used to benchmark the system is also included in this chapter. Following on in Chapter 4 is the analysis of the general WebRTC performance by taking both camera input and standard video files. Comparisons are made between the proposed streaming server and FFserver. Besides, this thesis also included the result of 3 types of video quality assessment for Quality of Experience (QoE) test in Chapter 4. The thesis ends with drawing conclusions and suggestions for future work in Chapter 5.

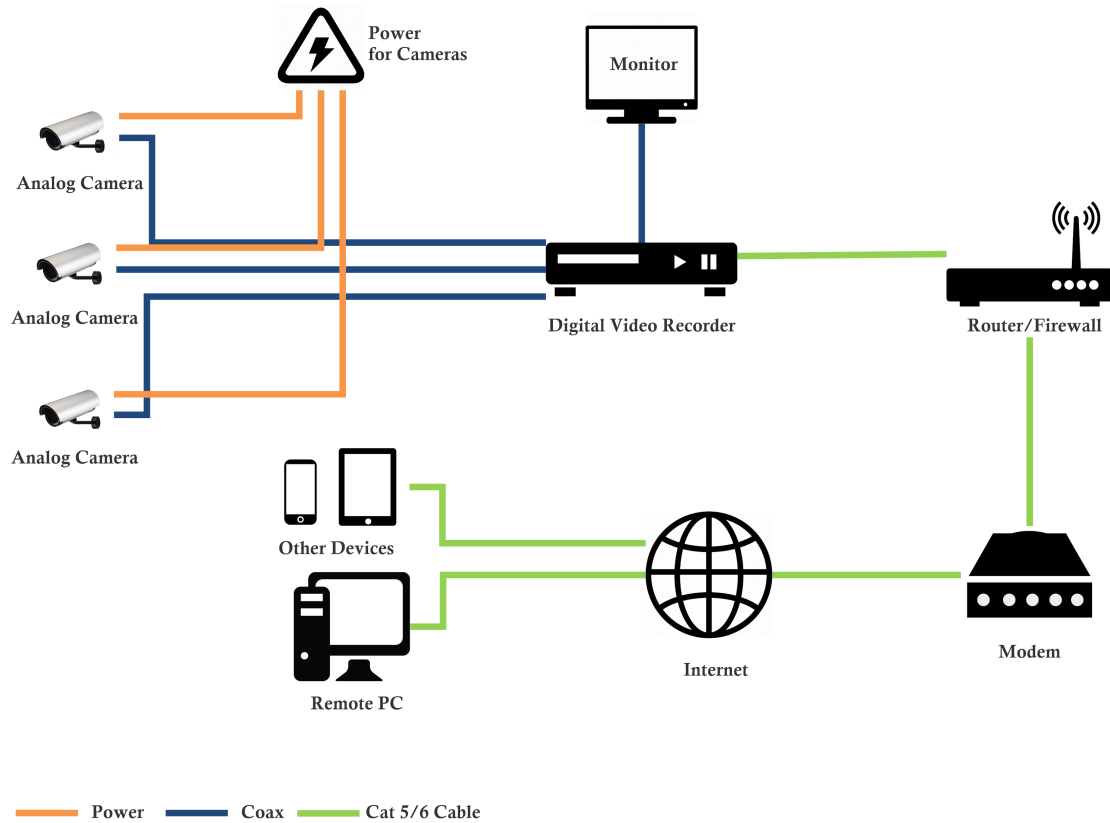
# Background

## 2.1 Analogue vs Digital Camera Surveillance Systems

IP based surveillance system has becoming the mainstream products in recent years. The definition of an IP based surveillance system is basically digitizing all video streams and transmit them using network [11]. Users are able to request information such as snapshot or video clips through the network. This enables users to conveniently control and manage the whole system remotely. As compared to the traditional analogue CCTV, IP based system provides:

- Better quality images without degradation.
- Greater ease of use.
- The ability to record and play simultaneously (live viewing).
- The ability to compress multimedia content (encoding) which eventually improve storage.
- Enhancement in search capability.
- Better scalability and flexibility.
- Quick alert response.

Figures 2.1 and 2.2 differentiates the system setup between an analogue CCTV system and the modern IP camera based system. As illustrated, analogue camera and IP camera uses different power source. Analogue camera uses the normal power source whereas IP camera is powered through a CAT 5/6 cable. This technology is known



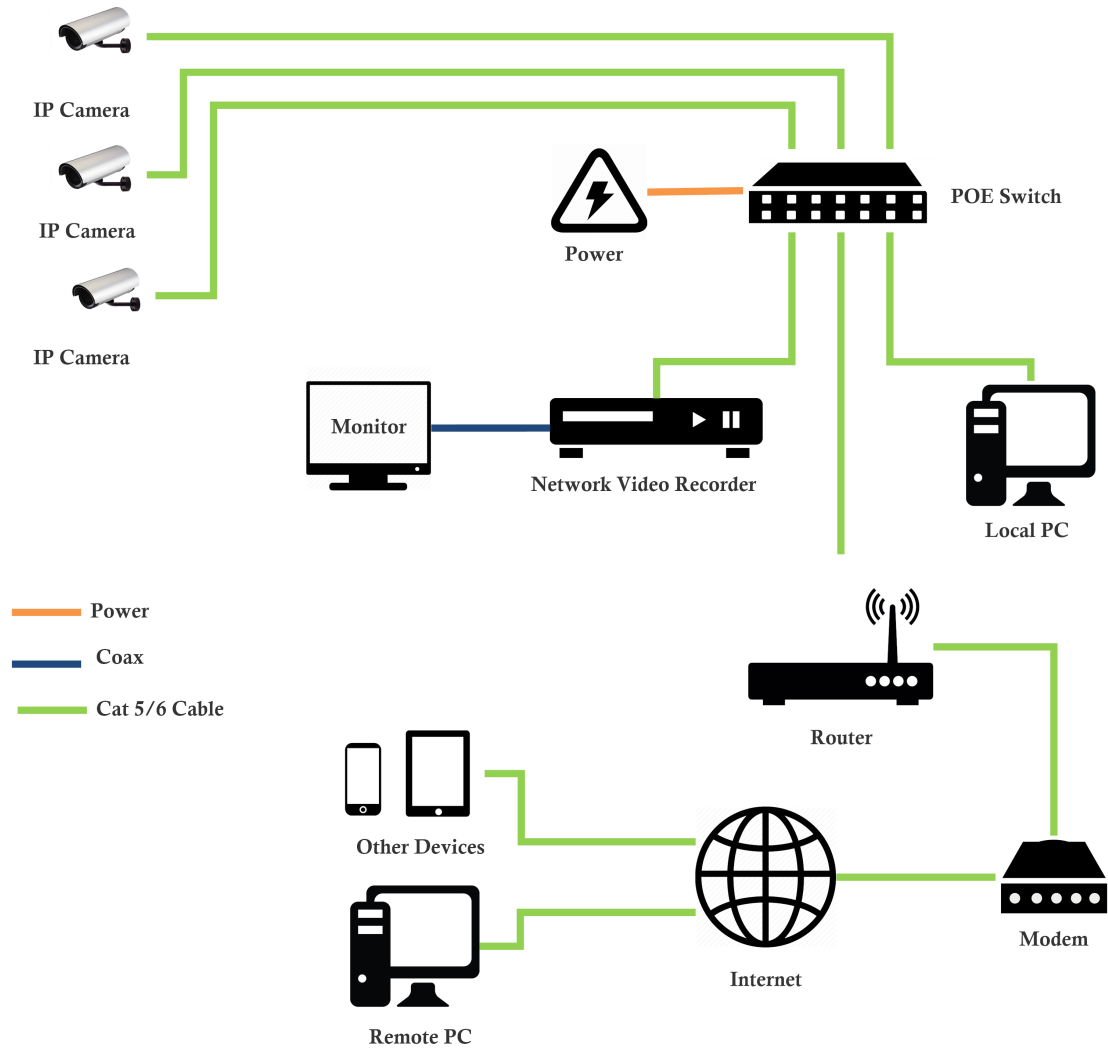
**Figure 2.1:** Analog CCTV System [1]

as Power over Ethernet (PoE) allowing electrical power and data being transmitted through Ethernet cable together. PoE switches also allow users to connect more cameras to the same NVR. In an analogue set-up, similar amount of cameras would require additional digital video recorders (DVRs) and cables.

Another main difference between the two mentioned systems is the use of NVR and DVR. According to Figure 2.1, a typical analogue CCTV setup is that cameras are connected to the DVR directly. DVR records, processes, and stores all the data collected from the cameras. In order to view the camera feeds, a monitor is linked to the DVR. Similarly, NVR serves the purposes of recording the footage, processing, and storing the data. However, NVR provides more freedom and flexibility in the whole system design. Since NVR works via Internet connection, camera placement is not an issue as long as it is within the range of the network.

## 2.2 Streaming Technologies

Video streaming is a phrase that is used in this thesis uncountable times. To define media streaming; a process that is constantly providing media data to an end- use by



**Figure 2.2:** IP Camera System [1]

a provider. The word "stream" indicates the delivery process of the medium rather than the medium itself, it is similar to file downloading. However, when an end user is receiving a streaming video file, he or she will be able to view the video file before downloading the file entirely. There are generally two types of media streaming; live streaming and video on demand. Live streaming is the delivery of media content (audio, video or both) in real-time just like a live television broadcast. Live streaming requires additional hardware and software in order to provide services with least latency. The basic setup before any live streaming involves a form of media source, for example, a video capture device, an audio interface or a screen capture software, a transcoder to digitise analogue content, a media publisher and the Internet to distribute the content [12]. On the other hand, on-demand streaming refers to receiving a stored media. It is a more economical way and it offers higher quality and lesser bandwidth for both user and provider. However, in this thesis, one of the major design concerns is to provide

user an instant video feed. Hence, live streaming will be the focus of this thesis.

Nowadays there are countless technologies and media servers that are able to stream media flawlessly. This section of the thesis discussed several popular streaming technologies and define the reason why FFserver is chosen as a comparison to WebRTC.

### **HTTP Live Streaming (HLS)**

HTTP Live Streaming, previously known as MPEG-DASH, is an HTTP based media streaming protocol developed by Apple Inc. This protocol is included in part of QuickTime, OS X, iOS and Safari software. HLS basically sends audio and video through HTTP from a normal web server to iOS based client devices that include iPhone, iPad, iPod, Apple TV, and MacOS computers. HLS is claimed to be reliable and is able to dynamically adapt to network condition. It is able to optimize playback based on the available network bandwidth. Although HLS is designed to handle multiple streams efficiently, it introduces high latency during playback (at least 20s). This is due to segmentation in queue before allowing playback. HLS creates three segments in queue and each segment is divided by a keyframe and the segment duration would be approximately quarter of a second, adding additional load to the server [13].

### **Flash Video Streaming (RTMP)**

Flash video streaming is a technology to deliver digital media content over the Internet created by Adobe. Flash uses RTMP for streaming audio and video between a flash player and the server. There are two different video container formats: FLV and F4V, used in flash streaming. There are known as flash videos. Flash video supports most video codec, for example, H.264, VP6, MPEG4, and most of the Sorenson Spark codecs. Although flash video streaming is widely used throughout the Internet, its major downside is that it is short of reliable mobile compatibility. Mobile devices such as iOS and Android devices do not support Adobe Flash player natively. Besides, web browsers needed flash player plugin to support flash playback [14].

### **Streaming Servers**

There are many media streaming servers available on the market. Media servers such as Wowza and Dacast are commercialized, in order to obtain a fully functional sys-

tem, users have to pay, commercialized systems are not in consideration of this thesis. Therefore, WebRTC and FFmpeg are taken into account. A comprehensive WebRTC explanation is included next section onwards and an overview of FFmpeg/FFserver is provided below.

FFmpeg is a free and open source project that includes enormous software libraries for video, audio, and other media files management. FFmpeg libraries are a core portion of media players like VLC. Moreover, it has been included in core processing for iTunes and Youtube. The main functions that FFmpeg performs are media format transcoding, video scaling, trimming, concatenating, and post-production effects. FFmpeg includes almost all known audio and video file formats and codecs, making it highly useful for multimedia transcoding. FFserver is part of FFmpeg, serving as a multimedia streaming server. It is able to stream video real-time and on demand [15]. In this thesis, FFserver is chosen to compare with WebRTC as it is the most popular open source media streaming server that is able to deliver high quality live streams. In detail, FFserver will be configured to stream live flash videos in comparison to WebRTC WebM videos.

## 2.3 Web Real-Time Communication

Web Real-Time Communications (RTC), or WebRTC [16], is a standard and industry effort which enhances web browsing models. With this, web browsers are now able to exchange real-time media directly on a peer-to-peer mode. According to Schollmeier [17], peer-to-peer network is defined as distributed network architecture. A P2P network includes one or more peers and a server which provides service and content. The shared services are accessible by other peers directly without an intermediate mechanism.

In 2010, Google [18] acquired On2 [19], a company that specialized in developing video codecs. The most popular codec created by On2 is the VP series [20], with the latest being VP8. The VP series codec is created to be patent free and most importantly to replace patented H.26x series [21]. On2's technologies are revealed as open sources VP8 under the name of WebM [22].

Google acquired Global IP Solutions (GIPS) [23] in May 2010. GIPS is a company known for developing media frameworks, a piece of technology that enhances Voice over IP (VoIP) and video calling applications. After Google's acquisition, GIPS' assets



were made open source. All patented voice and video codecs are removed and an additional layer (JavaScript API) is added. This extra layer is used to integrate with web browsers enabling bidirectional media processing and communication. This technology is named WebRTC, standardized by The World Wide Web Consortium (W3C) [24] and the Internet Engineering Task Force (IETF) [25]. IETF focuses on developing the fundamental communications and data transfer protocols while W3C creates the Web Application Programming Interface (API).

### 2.3.1 Structure

The WebRTC architecture consists of Web servers and browser clients. Before the implementation of WebRTC, Web browsers are only able to communicate to Web servers. With WebRTC, all Web applications with WebRTC support are now able to communicate with each other through P2P model. P2P communication paradigm between browsers provides more flexibility and scalability to WebRTC's architecture [2]. Henceforth, several WebRTC architectural models were introduced.

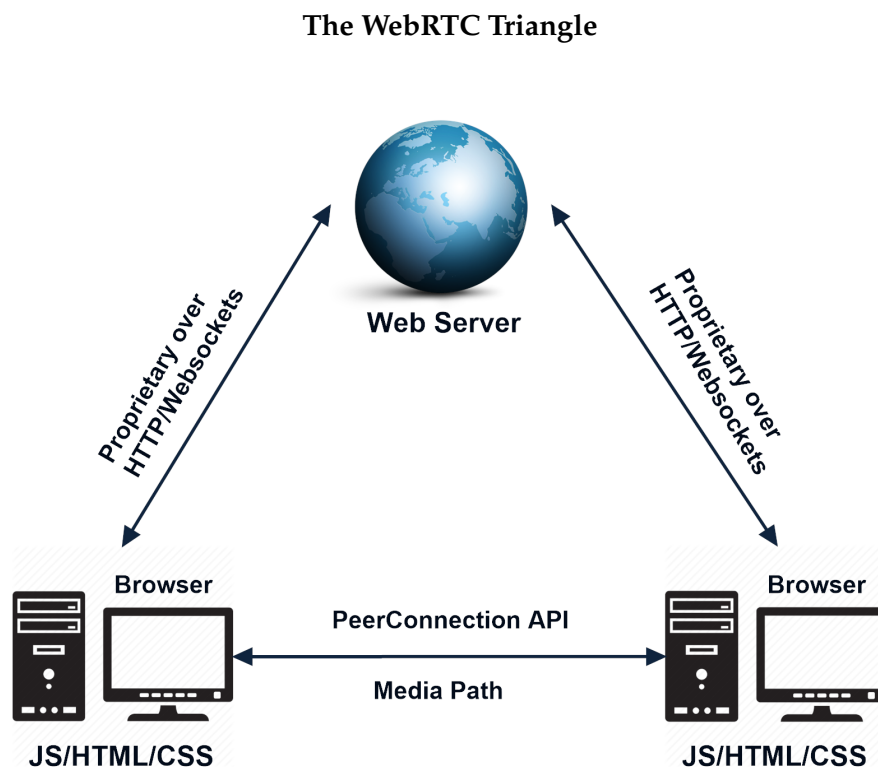


Figure 2.3: WebRTC Triangle [2]

Figure 2.3 shows the architecture of WebRTC triangle [2]. According to the figure, a Web server is used to serve Web applications with embedded Javascript [26]. Two or

more clients are interconnected to each other and the Web server. Clients such as PCs, smartphones, and tablets are able to access the Javascript application through browsers. The communication in between client's browser and the Web server controls data flows. Those data are called signaling messages, they are used to set up and terminate communications. Since the signaling mechanism between browser and server is not standardized in WebRTC, transport protocol such as Hypertext Transfer Protocol (HTTP) or WebSocket [27] can be used depending on the developer.

As for media path, the PeerConnection [28] API enables media data like audio and video streams to exchange directly between browsers without any master server. The direct media interchange between clients' browsers enhances the QoE (Quality of Experience) of voice and video as network latency, jitters are eliminated and most importantly, direct transmission reduces the additional path for packets to travel.

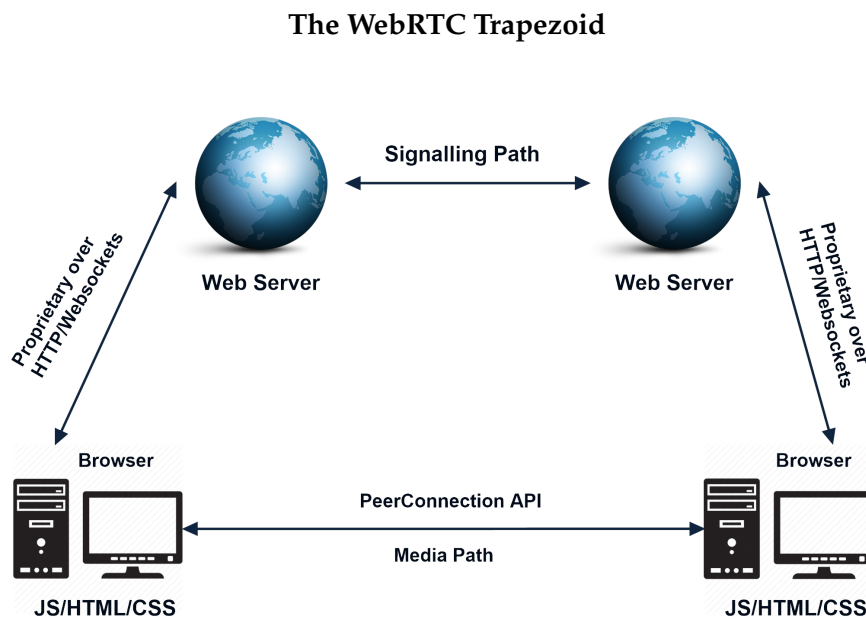


Figure 2.4: WebRTC Trapezoid [2]

The trapezoid model (Figure 2.4) is an extension of the triangle model. This model enables servers to federate with one another and eventually results in a more distributed network. Web servers use a signaling protocol such as Session Initiation Protocol (SIP) [29] or other proprietary protocol in handling the communications. In this model, there may be media relays or other elements set in between browsers, meaning that the media data may not flow directly between them[28].

### 2.3.2 WebRTC in the Web Browser

WebRTC is mainly used on web implementations which are written as a compilation of HTML [30] and JavaScript. The interactions between the web applications and browser use standardized WebRTC APIs, enabling the ability to exploit and control the real-time browser function. In details, the WebRTC APIs provide a wide set of functions such as P2P connection management, encoding/decoding necessity, media data exchange and control, firewall and Network Address Translation (NAT) [31] element traversal.

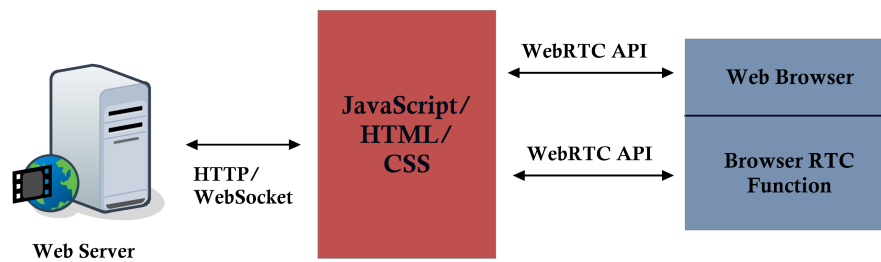


Figure 2.5: WebRTC Communication Flow in Browser [2]

Figure 2.5 demonstrates a real-time communication flow of WebRTC in the browser. In the real-time browser communication archetype, the design of WebRTC APIs are relatively important, a continuous, real-time flow of data communicating directly between browsers has to be ensured. The WebRTC basic flow of a video call that covers two web browsers summarizes as [32],

- Connect users
- Start signals
- Negotiate media sessions
- Start RTCPeerConnection streams

A more complex media communication flow and explanation will be discussed in the next section.

### 2.3.3 WebRTC Communication Flow

Figure 2.6 shows a full WebRTC call process which involves channel Initiator, channel Joiner and a signaling mechanism circulating messages between each other during channel setup. The whole initialization process involves the steps below:

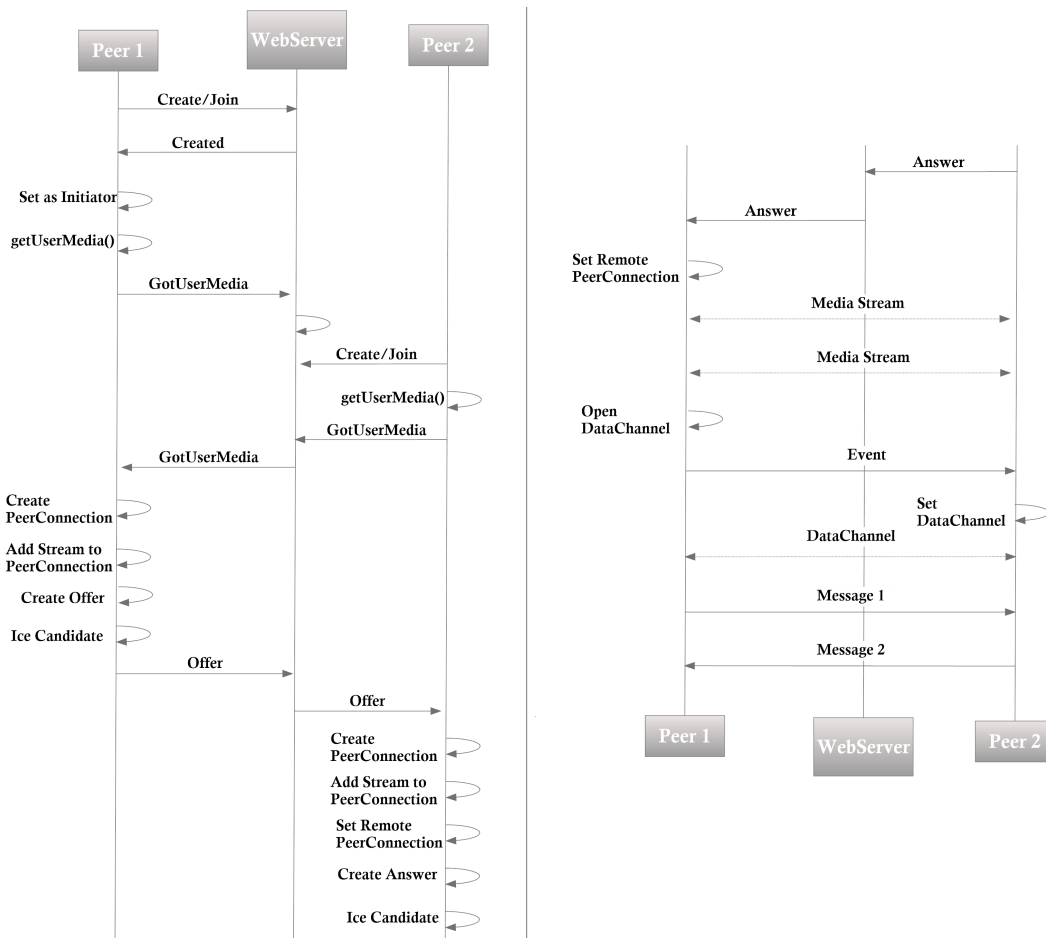


Figure 2.6: General WebRTC call flow

1. Once the Initiator attaches to the server, a signaling channel will be created.
2. The Initiator access own user media after acquiring user’s approval.
3. The Joiner will then connect to the server and join the channel.
4. After the Joiner is able to access the local user’s media, a message is then sent to the Initiator, a negotiation procedure is triggered:
  - (i) PeerConnection is generated by the Initiator and local streams are added to it. After that, an SDP offer is created and sent to the Joiner through the signaling server.
  - (ii) Once the SDP offer has been received, the Joiner cycles the operation of the Initiator such as creating a PeerConnection object, adding the local stream, and sending back an SDP answer to the remote peer via the server.
5. During the negotiation period between the Initiator and Joiner, network information is an exchange in the format of ICE candidate addresses.

6. As soon as the SDP answer is received, the negotiation process is over. Both parties switch to P2P communication by manipulating their respective PeerConnection objects. Media streams can be exchanged from this point onward. Additionally, data channel is able to exchange text messages directly for applications like chatting program.

### 2.3.4 WebRTC APIs

WebRTC enables data sharing, P2P teleconferencing, media exchange on browser without plugins or other third-party software through several interrelated APIs and protocols. Developer uses these functions and objects to communicate with the WebRTC layer and create peer connections. The APIs is designed based on three primary concepts:

- Media Stream
- RTCPeerConnection
- RTCDataChannel

#### MediaStream API

MediaStream API is designed to access streams of data (audio and/or video) from local input devices like webcams and microphones [32]. It handles actions on media stream mainly, displaying media content, recording or sending them to remote peers. MediaStream can be developed in a way that media may be obtained from remote stream.

A LocalMediaStream is a media stream from local devices like camera and microphone [2]. In order to create and use local streams, web application uses getUserMedia() function to request access from user. The media requested may be audio or video depending on the application.

MediaStream uses Secure Real-time Transport Protocol (SRTP) [33] to transport media data and the RTP Control Protocol (RTCP) [34] to monitor transmission statistics produced by data streams. For security concern, the Datagram Transport Layer Security (DTLS) [35] protocol is used to manage SRTP keys.

### **RTCPeerConnection**

The RTCPeerConnection is the main API that handles the peer-to-peer connection between each WebRTC enabled browser or peer. This API uses the same JavaScript application to represent local peer and remote peer. Communications between local and remote peers are coordinated via a signaling channel such as WebSocket or XMLHttpRequest. Media streams are able to exchange directly from browser to another once the peer connection is established.

There are two important mechanisms that are used in this API, Interactive Connectivity Establishment (ICE) [36] protocol and Session Traversal Utilities for NAT (STUN) [37] or Traversal Using Relays around NAT (TURN) [38] servers. ICE, STUN, and TURN are used in media applications to ensure UDP-based media streams traverse NAT boxes and firewalls.

### **RTCDataChannel**

The RTCDataChannel interface enabled Web Browsers to transfer arbitrary data in a bi-directional peer-to-peer fashion [39]. The reason why the RTCDataChannel is more preferable than other data channel like WebSocket, AJAX [40] and Server Sent Events is because for media applications it works with the RTCPeerConnection API enabling full-duplex P2P connectivity which results in lower latency and no intermediary server.

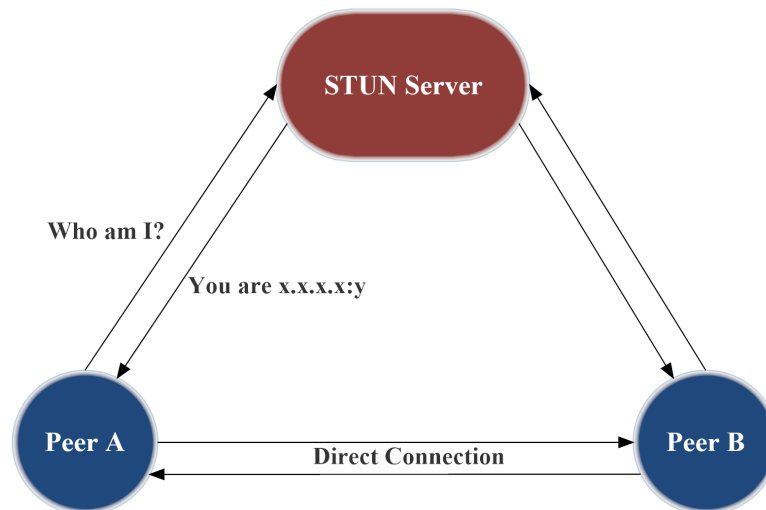
RTCDataChannel uses Stream Control Transmission Protocol (SCTP), a transport layer protocol which serves similar role as Transmission Control Protocol (TCP) [41] and User Datagram Protocol (UDP) [42]. SCTP is claimed to be a powerful improvement on TCP that is an ideal solution for transporting media [43]. By using SCTP, RTCDataChannel has the flexibility to send messages either ordered or unordered and the retransmit is configurable.

#### **2.3.5 STUN, TURN and ICE**

Computer networking often requires a host to exchange packets with another. In a situation when a host is behind a NAT and wish to exchange packets with other hosts which may also be behind NATs, a hole punching [44] technique is needed. It discovers a direct communication path through intervening NATs and routers but does not traverse relays [45]. However, according to state of P2P communication across NATs

[44] and NAT UDP unicast requirements [46], the hole punching technique will fail when both hosts behind NAT have mapping behaviour of "address-dependent mapping" or "address and port-dependent mapping" [47]. These mapping behavior causes the discovery of direct communication impossible, hence an intermediate host acting as a relay for the packets is needed. They have to sit in the public Internet and relay packets between the two hosts that are behind NATs.

The relaying service defines a protocol, named STUN/TURN. The functionality of STUN and TURN are relatively similar as TURN is an extension to the STUN. The situation and differences of using STUN and TURN are visualized in the diagrams below.



**Figure 2.7:** STUN P2P Communication [2]

Figure 2.7 and 2.8 illustrates a simple usage of STUN and TURN servers. In the situation shown in figure 2.8, both peers are sharing the same TURN server. They use the STUN server to discover their network information and use TURN server to exchange their data packets. However, in the real world application, every TURN server also serves as a STUN server, they are usually providing services as one single physical server. When the communication between two peers is established via a TURN server, the TURN server has to be always available. Once the TURN server is off, the connection will be dropped.

The selection of either STUN or TURN is decided by the type of NAT used. In the case that full cone, address-restricted or port-restricted NAT is used between peers, a direct link can be discovered by STUN. Additionally, when one of the router uses symmetric NAT, the other routers has to avoid symmetric or port-restricted NAT in

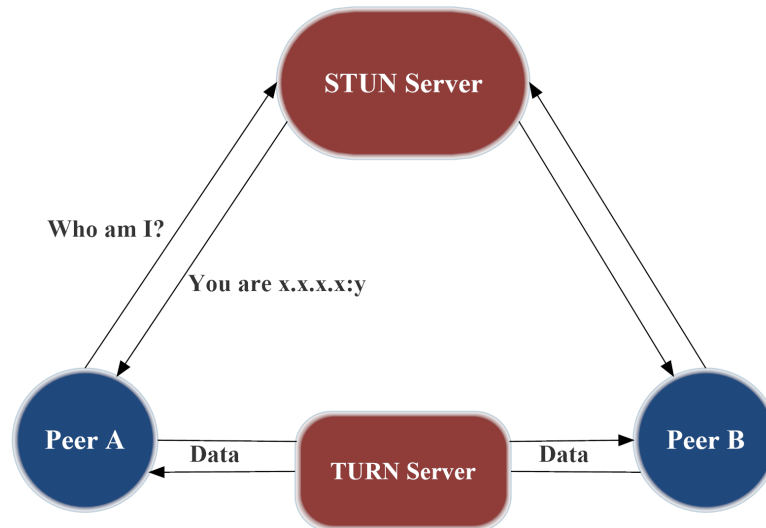


Figure 2.8: TURN P2P Communication [2]

order for STUN to discover the connection. Whereas if both peers are using symmetric NAT or one of them is behind a port-restricted NAT, TURN server is necessary.

TURN server uses public intermediary in relaying packets. It enables hosts that are behind NATs (TURN clients) to request the TURN server to act as a relay. Clients are able to arrange and control the relaying flow to and from other peers. TURN client first obtained the IP address and port on the server and the relayed transport address is called. After a peer sends a packet to the relayed transport address, the packet will be relayed to the client. On the other hand, when the client sends a packet to the server, it uses the relayed transport address as the source to deliver the packet. By using the TURN server, more resources are needed and it also increases latency between peers. Hence, it would be last option for using TURN when no other way is available [45].

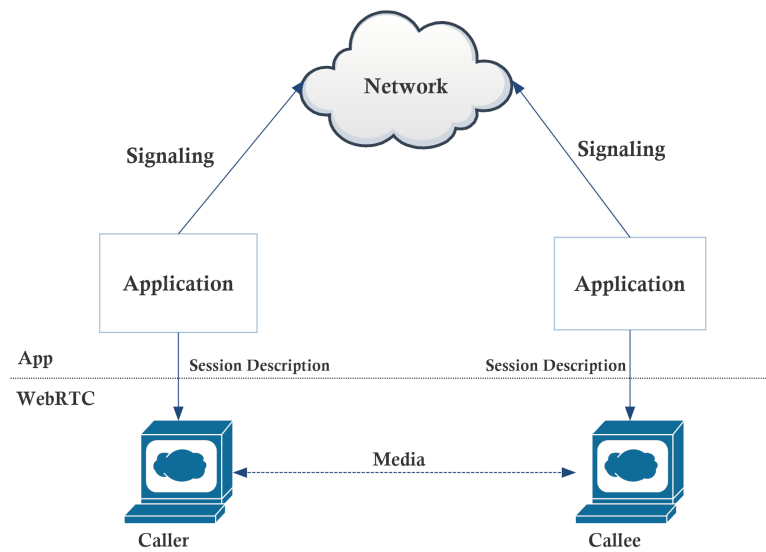
Interactive Connectivity Establishment (ICE) is another important mechanism that is used with STUN/TURN. ICE defines a set of standard that handles the multimedia communication for NAT traversal [48]. ICE's architecture is based on the offer/answer [49] model that deals with session negotiation. According to different situation, ICE creates numerous transport addresses (IP:port) which are typically called ICE candidates [50]. These ICE candidates are provided to media stream which each media stream may contain multiple candidate addresses. The transport addresses are then validated with P2P connectivity checks by STUN/TURN.

A more detail operation of ICE is arranged into seven steps based on Rosenberg [51], gathering, prioritizing, encoding, Offering and answering, pairing, checking and completing.



### 2.3.6 Signaling

Signaling and negotiation is the process of exchanging contact information with another and coordinating the communications. Information such as session control messages (start/terminate communication), error messages, key data (establish secure connections), media metadata (codec, bandwidth, and media type), network data (IP address and port) are exchanged during the setup of a call. Session control messages known as Session Description Protocol (SDP) [52] is the most essential information that needs to be exchanged. It contains transport (ICE) information, media type, format, and all media configuration parameters that are necessary for the initial handshake. The communication model differentiates the signaling and media transaction into different layers based on the JavaScript Session Establishment Protocol (JSEP) [53]. According to figure 2.9, the media communication is established between the browsers of caller and callee whereas the signaling happens between applications through the network.



**Figure 2.9:** JSEP signaling and media flow [3]

The general operation of signaling consists of steps below [3]:

1. A list of potential candidates are generated.
2. A user will be selected by either the user or algorithm to connect with.
3. The signaling mechanism will notify the recipient that someone would like to communicate with him/her, and he/she can choose to accept or decline.

4. If the request is accepted, the host will initiate `RTCPeerConnection` with the recipient.
5. Hardware and software information will be exchanged over the signaling channel.
6. Location information will also be exchanged over the signaling channel.
7. The connection will either succeed or fail.

Since the idea behind the design of WebRTC is to fully specify and control the media plane, while leaving the signaling plane to the application layer, applications are provided the flexibility to use different protocols [53]. The chosen protocol can be any among Session Initiation Protocol (SIP) over websockets [29], XMPP [54], HTTP/REST [55], JavaScript Object Notation (JSON) via XMLHttpRequest (XHR) [56], or any custom to the particular application. As outlined by JSEP [53], WebRTC's signaling methods and protocols are not standardized due to the fact that with customized protocols, compatibility of applications will be maximized and redundancy will be reduced.

## 2.4 Case Studies

Physical surveillance and security systems have been advancing in past decades. Different approaches are made not only to improve the performance of the system itself but also to users quality of experience. In this section, several security systems are studied and the cross analysis between each technology used will be discussed.

### 2.4.1 GSM Based Smart Home Security System

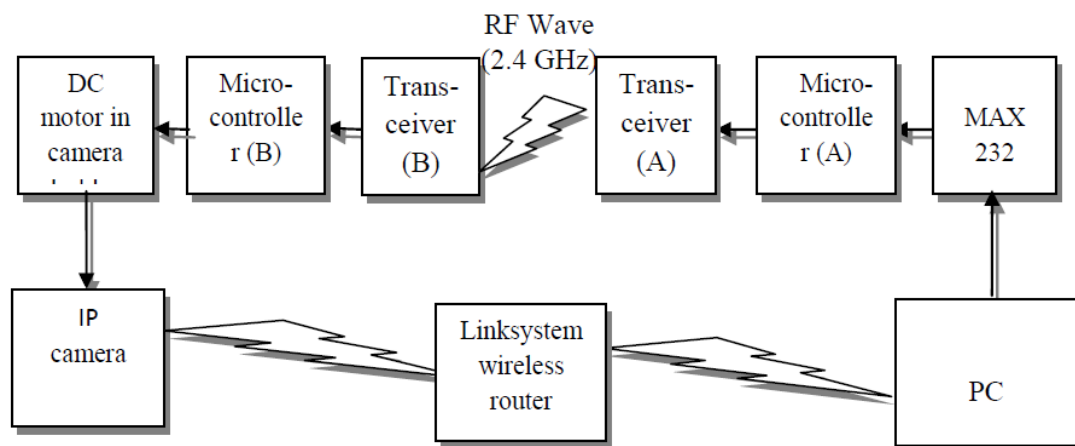
Bangali and Shaligram [57] proposed a smart home security system that includes two prototypes. The first design uses web cameras, they are installed in-house premises. All the management and controls of the cameras are performed on a PC using Internet as communication method. Bangali and Shaligram further stated that the camera management software they used is a free Java based tool called Yawcam (Yet Another WebCAM Software). It is able to view the web camera's video feed live, take snapshots and also stream the videos through Internet. The implementation and operation of

this system are simple and it requires only a PC as the central server and several web cameras.

The second system that Bangali and Shaligram suggested is a GSM based home security system. This system is designed particularly against intruders and fire. It consists of three main modules: sensors, GSM-GPS module and microcontroller. The microcontroller collects information from the sensors, if alarm is needed, it will send SMS to a corresponding number through the GSM modem. In the case of sensor interruption, an alert SMS will be sent to the homeowner. Furthermore, house temperature is also monitored. If the temperature breaches certain level, the house owner will also be notified. Since the system is specially designed for home use, all the components used are low cost and the SMS alert is nearly instant. However, Bangali and Shaligram's approaches are more to a real-time alert system. There is no specific way to monitor the real-time video feeds efficiently. Besides, the system also lacks a way to manage more web cameras.

#### 2.4.2 IP based Surveillance System

In 2012, Ohaneme [4] developed an IP based surveillance system. The system uses a simple PC as the server, an IP camera, a pair of wireless transceivers and microcontrollers as shown in Figure 2.10.

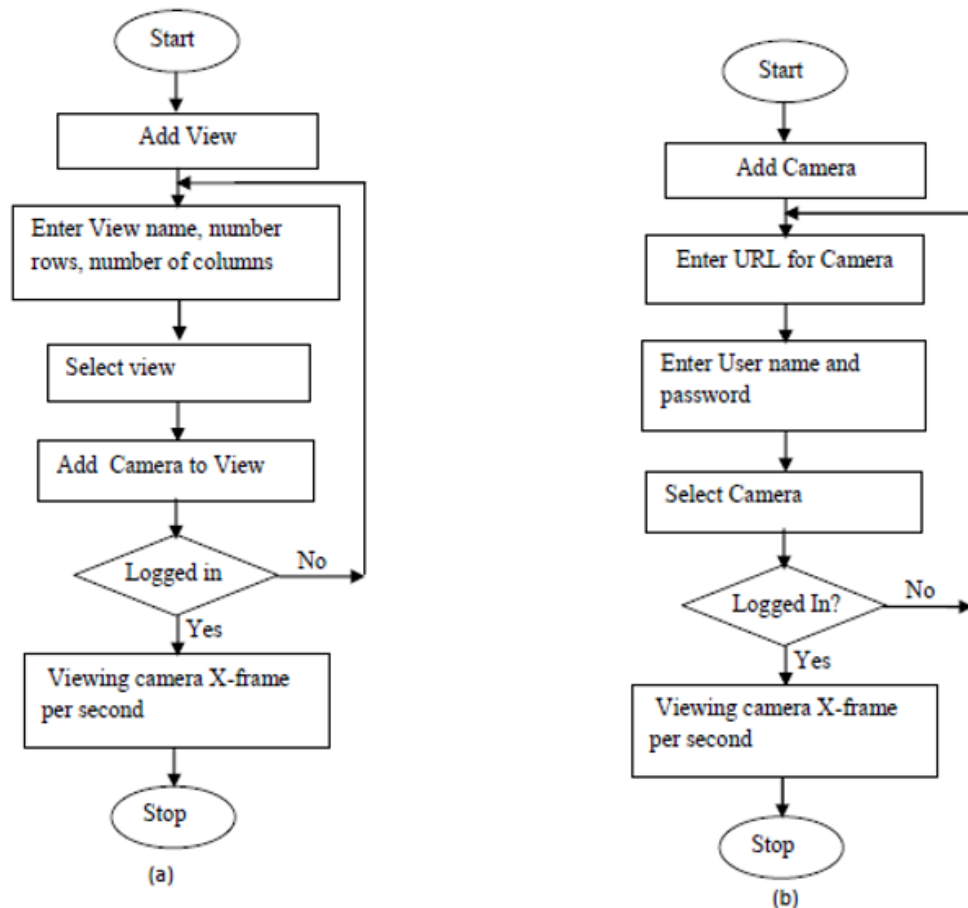


**Figure 2.10:** Schematic design of PC based IP surveillance security [4]

The wireless transceivers and microcontrollers are for the rotation purpose. The authors further explained, the user interface on the PC is a comprehensive application that not only provides camera video feeds, record and save the video footages but it also controls the rotation of the camera to various directions. Whenever the direction

of the camera is changed, information is sent from the PC through the serial port, MAX 232 to microcontroller (A). The communication between microcontroller (A) and (B) uses a pair of RF wireless transceivers. After microcontroller (B) received the signal, it then drives the DC motor mounted on the camera.

The output video of the system is streamed in compressed form over the network and displayed to the viewer in a real-time manner. In order to view, users need a standalone player that deals with the decoding of received media data and display the video and audio data. Additionally, if users prefer to view the camera feed on a Web browser, a plug-in is needed. A dedicated Web server is made to multicast compressed media data to users Web browser at the same time. User interface applications of this system are created using Visual C# (Visual C-Sharp) programming.



**Figure 2.11:** (a): Flow chart of add/select view operation, (b): Flow chart of add/select camera operation [4]

The design and implementation of this system are similar to the proposed surveillance system. However, the streaming method used only supports standalone player and Web browser's plugin. Users are not able to view the live streams using other devices

such as smartphone and tablet. Besides, referring to the flowchart, management of the cameras requires users to insert the URL manually. Hence, it is tedious and time consuming for users to handle large numbers of camera.

### 2.4.3 Cloud-based Multimedia Surveillance System

Another approach discussed by Hossain [5] is a surveillance system that uses cloud infrastructure to process and store the multimedia data. The paper included different design choices and also issued the difficulties of designing and implementing a cloud-based multimedia surveillance system. As stated by the author, when designing a cloud-based surveillance system there are several distinctive issues:

- (i) Deployment architecture: public cloud, private cloud or hybrid.
- (ii) Media acquisition: push-only, pull-only, push-pull, or event-driven.
- (iii) Cloud storage: vendor lock-in, recovery capability, elasticity, security and privacy policy and pricing structure.
- (iv) Media processing: different media processing tasks, for example, motion detection, face recognition and people tracking.
- (v) Resource allocation: balance needed between resource utilization, response time and cost.
- (vi) Notification and sharing: ways to propagate information such as live feed and alerts from publishers to subscribers.
- (vii) Crowd-based surveillance and big data analytics: efficient algorithm in analyzing and manipulating big data.
- (viii) Security and privacy: access control policy and security impact.
- (ix) Performance issues: accuracy of event detection, response time, CPU and storage utilization.

Hence, by considering all the above issues, Hossain came up with a design as shown in figure 2.12. Two types of content providers are supported in this architecture, one is the sensor devices such as fixed cameras and IP cameras, whereas the other is the crowd that reports security incidents. Contents are then transmitted to the cloud according to user's configuration. This framework consists of several core components:

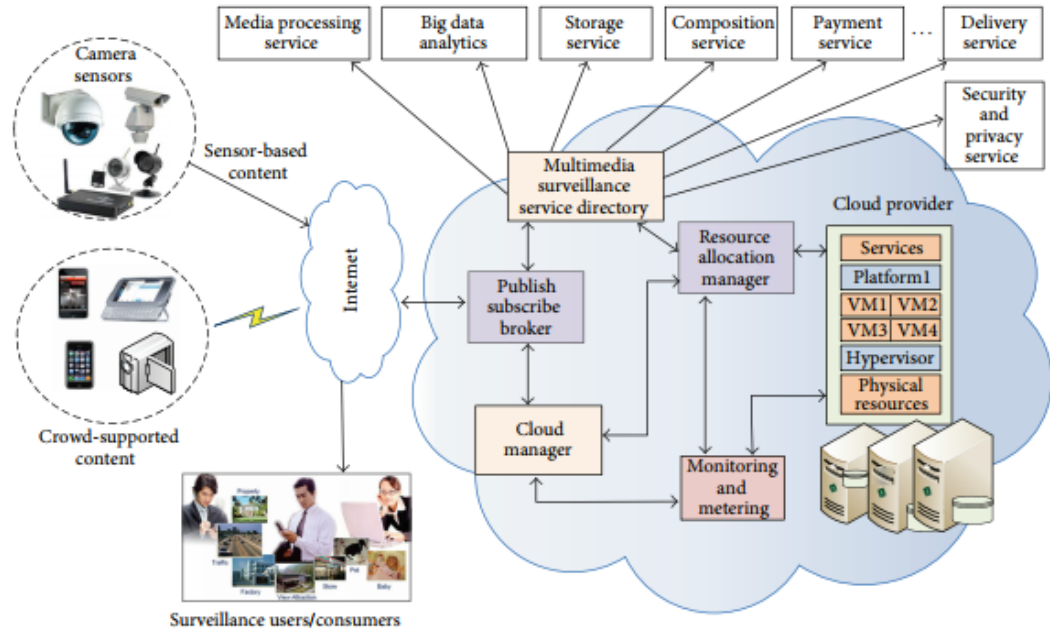


Figure 2.12: Cloud based multimedia surveillance framework [5]

- (i) Publish-subscribe broker: facilitates in publishing and subscribing media streams. It delivers video streams to content providers and users by using multicasting approach.
- (ii) Multimedia surveillance service directory: where services are registered and are accessible over the Internet.
- (iii) Cloud manager: controls the publish-subscribe broker, multimedia surveillance service directory, the monitoring and metering component and the resource allocation manager.
- (iv) Monitoring and metering: computing and tracking resource usage, monitoring performance, and provides statistics of usage and billing.
- (v) Resource allocation manager: manages and allocates resources for the surveillance system and related services.

Hossain concluded that although the implemented prototype shows the suitability of this particular framework, issues like cost, security, and privacy still remain as a decisive factor.

#### 2.4.4 Video Surveillance in Wireless Router

The fourth case study found is a video surveillance in wireless router system designed by Kao et al.[6]. A comparison of CCTV, Web camera and IP camera is made in this paper. Referring to the comparisons, IP cameras are easier to manage and able to cover a wider area, it is used in this design. Besides, the authors stated that the reason for using wireless communication is to reduce deployment costs and to save energy consumption.

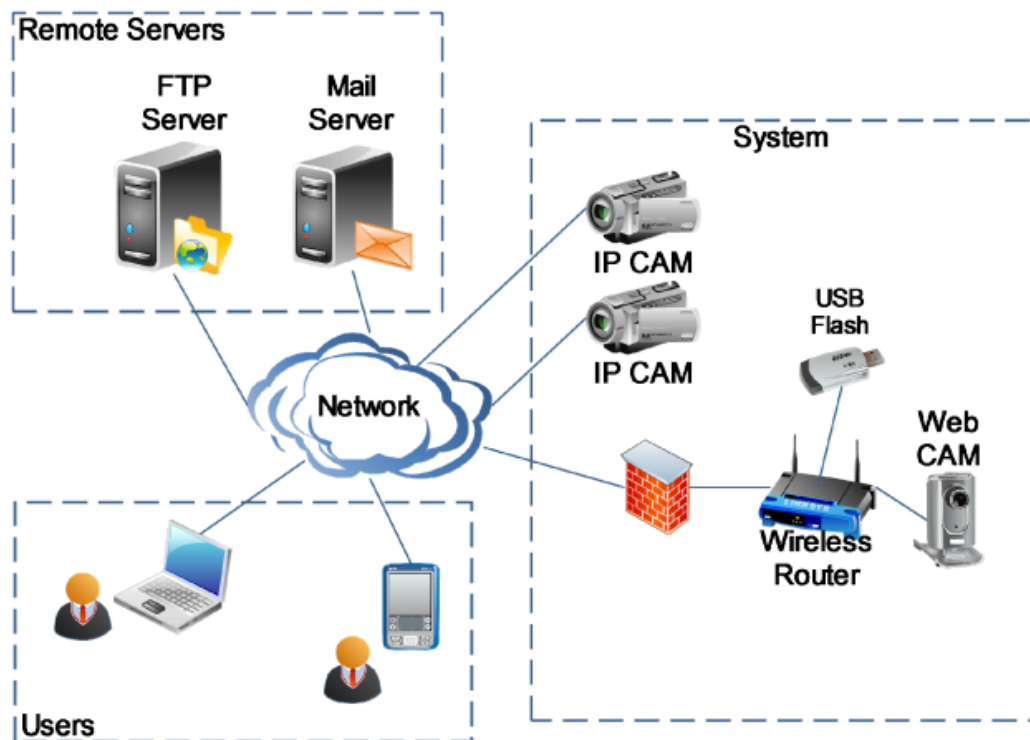


Figure 2.13: Video Surveillance Architecture [6]

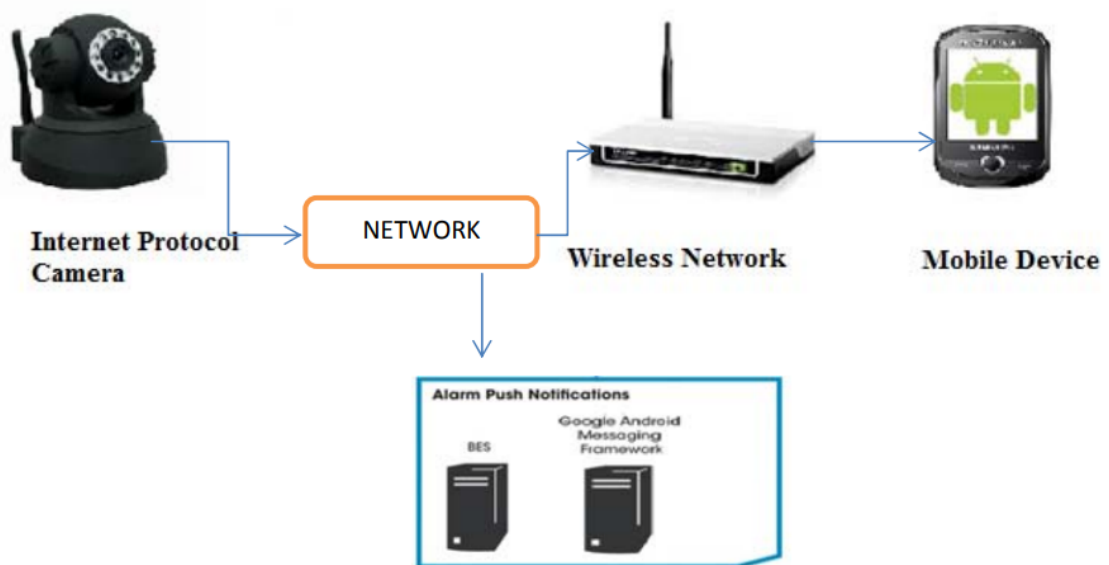
The diagram above shows the architecture of the complete system. It includes several IP cameras, a wireless router connecting a Web camera and a USB drive for media storage. The wireless router used has a powerful Central Processing Unit (CPU) with 400MHz, 32MB RAM and 4 or 8 MB serial flash. The OpenWRT/Gargoyle which is a Linux based OS is installed on the wireless router. A software named Motion is also built in the wireless router to provide video surveillance functions such as real-time image monitor, motion detection, image upload, and email alert. Whenever an event occurs, the motion detector will record the IP camera feed into the USB flash and concurrently either upload the image to the FTP server or send an alert email to the user.

The CPU and memory usage of the wireless router is tested and graphed. Based on their result, the CPU usage will be maximised when 4 IP cameras are detected to have events and at the same time storing video into USB flash.

This video surveillance system demonstrates a rather simple design and implementation with multi-function. It is able to reduce cost and power consumption due to the fact that only a wireless router is used in managing all the cameras. However, the limitation of only 4 IP cameras can be used is restricting the scaling of this system meaning that it is only suitable for home and small area use. Besides, the wireless router is not able to handle a more complex function such as video processing and video streaming.

#### 2.4.5 Video Surveillance System and Facility to Access PC from Remote Areas Using Smart Phone

According to Rashmi and Latha [7], a video surveillance system dedicated to smart-phone is suggested. The whole system basically involves IP cameras, a wireless network and Android-based smartphones as shown in the figure below.



**Figure 2.14:** System Design of the Video Surveillance System and Facility to Access Pc from Remote Areas Using Smart Phone [7]

The complete system uses a wireless network in transmitting video feeds and control commands. Whenever there is an intruder or certain unauthorized action, an alert will be sent through messaging the user. At the same time, the image of the unauthorized action is stored in the mobile SD card. System configurations can be changed by send-



ing requests through smartphones. The live and recorded videos are able to be accessed by more than one user from different locations.

The proposed system includes several important technologies:

- (i) Motion detection: used in detecting changes in position relative to its surroundings or vice versa. The most common approach mentioned by Rashmi and Latha is to compare the current frame of the video with the previous one in order to detect the differences. Rashmi and Latha further explained the method developed named Motion Detection Extraction. It enables motion detection capabilities on Android smartphones. Motion Detection Extraction separates the foreground and the background precisely through the Basic Background Subtraction method. Since there are factors such as lighting and shadows which will affect the process of foreground background separation, a highly accurate algorithm is needed.
- (ii) Data transmission: mainly transmitted via email and SMS. Hence, Internet access is needed at the smartphone end in order to receive the alert emails. When users want to retrieve a file from the system, they have to provide the specific file name with extension. The system will then look through the directories and sent it back to users.
- (iii) Face recognition: used in attendance system that identifies employees through contactless optical face recognition technology. The recognition rate is claimed to be 99.9%. Some extra security features are included, for example when a user is being scanned, he may be asked to make facial expressions like smile, blink or nod their head. Besides, the facial recognition function calculates and analyses the characteristics of a person's face, measurements such as distance between nose, mouth, eyes and jaw edges are stored in the database. These stored information are used as comparisons in the scanning processes.
- (iv) Object identification: a function that is used to identify objects. This allows smartphones to capture the image from the IP cameras and the image is sent to Google where the image is compared. If there is any missing object detected, the application will immediately launch an alarm.

After researching and interpreting all the past similar designs and systems. It is found that each system has their own limitations such as unable to manage large amount of

## CHAPTER 2: BACKGROUND

cameras efficiently, only supports single client platform: normally PC, not available for offline or LAN only system and also no media processing supports, specifically video transcoding function. Our system resolves the stated problems through integrating different modules. The system design and implementation will be further discussed in the next chapter.

# System Proposal and Experimental Setup

This chapter elaborates the details of the proposed NVR system. Discussion includes the overall system architecture, function, and features of each module and most importantly the in-depth explanation of WebRTC live streaming implementation.

## 3.1 NVR System Architecture

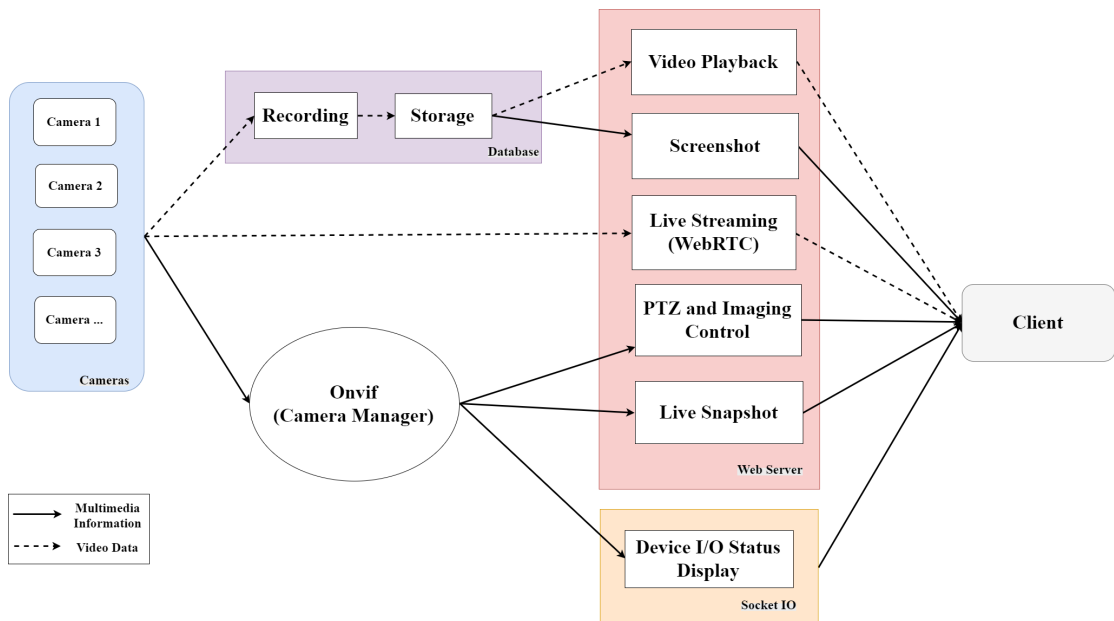


Figure 3.1: Block diagram of the NVR system

An overview of the complete NVR system is presented in diagram 3.1. The proposed idea of this architecture is to classify the entire NVR system based on different mod-

ules. Each module consists different functionality. WebRTC live streaming is one of the functionality under the web server block. It is the main focus of this research and involves several process from abstracting video feed from the IP camera to the end user. These processes will be further explained in the later sections.

There are 3 primary sections according to the block diagram: i) Camera discovery and management ii) Recording and data storing and iii) Web applications. As you can observe from the diagram, the communications in between each block are classified into solid arrows indicating multimedia information such as camera details, image data, and control information and dotted arrow specifying video data. From figure 3.1, the streaming process started from left to right. Video data is obtained from cameras and piped into the recording module and WebRTC as input. For database used, video is recorded, stamped with specific time stamp and stored in desired location. Stored videos can be viewed from the web interface. Besides, user is able to take screenshot from the web interface. Where for the live WebRTC streaming module decode the H.264 stream, encode it to VP8 format, send it to the WebRTC media server and finally deliver the content to end user through the Internet. Another big portion of the system is the Onvif module. It obtained camera information, stored it and escalate these information to other modules. It is basically the camera manager of this system.

### 3.1.1 Camera Discovery and Management

The key standard used in IP camera discovery and management is known as Open Network Video Interface Forum (ONVIF) [58]. It is a global standardization that allows network devices such as IP camera to communicate with interoperability between all ONVIF supported devices. ONVIF defines a set of network interface functions known as ONVIF services. These ONVIF services use Simple Object Access Protocol (SOAP) message in exchanging information. SOAP is a lightweight, Extensible Markup Language (XML) based protocol. Its principal function is to encode web service request and response before transmission. In this NVR framework, ONVIF is used in camera discovery, media and image configuration, camera detail management, pan tilt zoom (PTZ) controls, event handling and also alarm I/O monitoring.

During the initial stage of camera data acquisition, ONVIF sends its web service discovery message periodically to check the availability of ONVIF compliant cameras. Whenever an IP camera is detected, ONVIF will send a series of request to retrieve

camera information and features such as video stream address, video resolution, frame rate, IP address, hardware address, PTZ, relay I/O status, snapshot and imaging capabilities.

All the received responses from camera are arranged and stored in XML format. The XML file is then parsed to other NVR modules like a web server, video storing, socket communication and live snapshot. Besides, ONVIF enables our system to detect the failure or disconnection of a specific camera. Once that camera is reconnected back, all terminated processes will be recovered.

### **3.1.2 Recording and Data Storing**

This module records camera feeds and stores them in hourly basis. During the recording process, each received frame is stamped with the system's time and date. Based on the XML file produced by ONVIF camera manager, each video file is named accordingly. If there is any occurrence camera failure or connection lost, this module automatically generates blank frames in order to replace all lost time intervals.

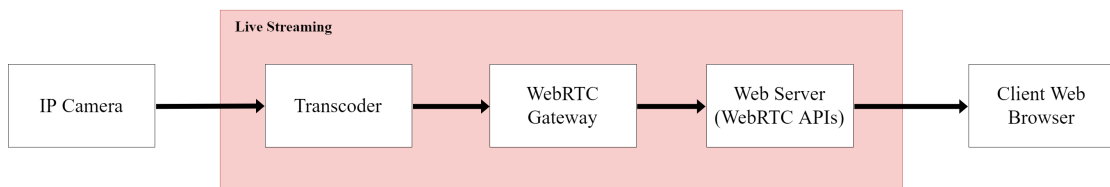
### **3.1.3 Web Server**

As for our web applications, we developed servlets in handling HTTP requests and various applications. Servlet is basically a class written in Java that is used to extend the capabilities of servers. Servlets are able to respond to any type of HTTP request and it is equipped with more comprehensive functions. They are normally used in extending applications hosted by web servers. Apart from that, the application server used for our web server is known as GlassFish [glassfish]. It is an open source Java EE reference implementation that supports servlets JavaServer Faces, Enterprise JavaBeans, JPA, JMS, RMI, JavaServer Pages and more.

In our case, servlets are implemented not only to handle HTTP client requests but also camera naming, imaging control, PTZ, live snapshot and the video processing for playback. In order to have a better camera management system, administrator of the web server is able to rename each camera specifically, the recorded video file naming will also be changed accordingly. Besides, they are also able to control Pan Tilt Zoom (PTZ) of a supported camera from the web interface. This allows clients to move the direction of the camera and save the current viewing angle.

The imaging function allows clients to adjust the focal length of an IP camera to have a better focus on near object as well as far. Some ONVIF compliant IP camera has a functionality of improving their viewing details by opening or closing the iris. In the video presentation stage, viewing of live stream and recording's playback are included. In order to retrieve the exact video for playback, client is required to select the desired start date, end date, start time and end time. Since the primary focus of the thesis is the real-time video streaming, the details and framework will be discussed in the following section.

## 3.2 WebRTC Real-time Video Streaming

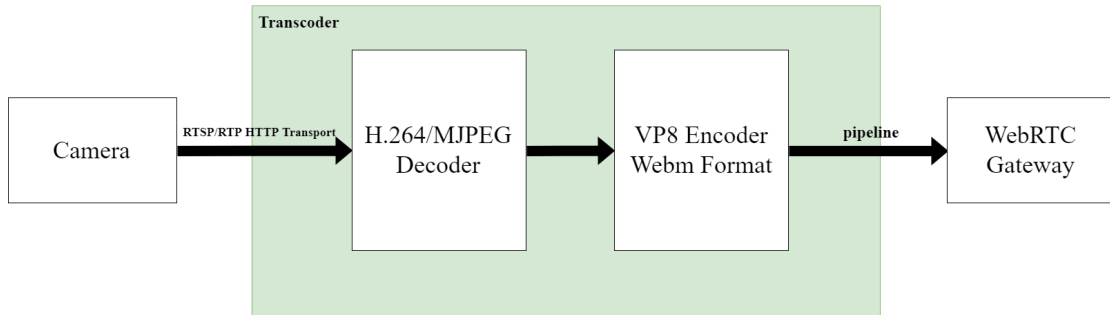


**Figure 3.2:** Block diagram of the live streaming module

Figure 3.2 shows the process of generating a real-time video stream from the IP camera to the end client. As observed from the block diagram, live streaming passes through 3 important processes, transcoding, gateway and finally the web server. An RTSP video feed is retrieved by the transcoder from IP camera, the video feed is then decoded and encoded with specific codec and format based on the media server requirement. The transcoded video will be piped into the WebRTC gateway. The WebRTC gateway basically acts as a bridge in between, relaying RTSP and WebRTC. Since there is no direct and standardized communication protocol for IP camera (RTSP) and WebRTC, a media gateway is important to interoperate. After the WebRTC server received the video data, the APIs will handle all the request and response between itself and the client.

### 3.2.1 Transcoder

Referring to Figure 3.3, the camera video feed is encoded with H.264 codec and is transported by RTSP/Real-time Transport Protocol (RTP) transport protocol. The feed is processed by a H.264/MJPEG decoder and next, re-encoded by the VP8 encoder with WebM container. The process involving encoding and decoding is known as transcod-



**Figure 3.3:** Block diagram of Transcoder

ing. It refers to direct digital to digital conversion of, in our case, video data files like MJPEG and Webm.

### FFMPEG

The transcoding tool used in this system is called FFmpeg [59]. FFmpeg is a prime multimedia framework that includes comprehensive utilities. It is able to decode, encode, transcode, mux, demux, filter, play, and stream nearly every multimedia content created by machines and humans. Besides, FFmpeg supports most known digital format and codec, from the most ancient and obscure to the cutting edge. FFmpeg toolset is highly portable, it compiles and runs on most operating systems such as Linux, Mac OS X, Microsoft Windows, BSD, and Solaris.

After compiling FFmpeg, it can be easily accessed through the operating system's command line utility. An example of the command used to transcode the IP camera video data is shown below.

See the following command :

#### Listing 3.1: FFmpeg command

```

$ ffmpeg -rtsp_transport tcp -i rtsp://admin:admin@192.168.191.20/axis
  -media/media.amp -c:v libvpx -deadline realtime -f rtp rtp://
  localhost:8000
  
```

The above command defines the transcoding process of video source (rtsp://admin:admin@192.168.191.20/axis-media/media.amp) which is obtained from the IP camera with TCP protocol. In order to retain the original video source quality, for instance, the resolution, frame rate, and bit rate, no extra command is inserted. The only change made to the stream is the codec, -c:v libvpx, which converts the original H.264 to VP8. FFmpeg uses CPU cores in processing all the transcoding processes. In order to increase

the delay as low as possible -deadline realtime is used. This function speeds up the transcoding process by using more processing power, occupying more CPU usage at the same time. The CPU and memory usage experiment and result will be included in the later chapter. Lastly, `rtp://localhost:8000` indicates the address of the WebRTC media gateway.

### 3.2.2 WebRTC Media Gateway

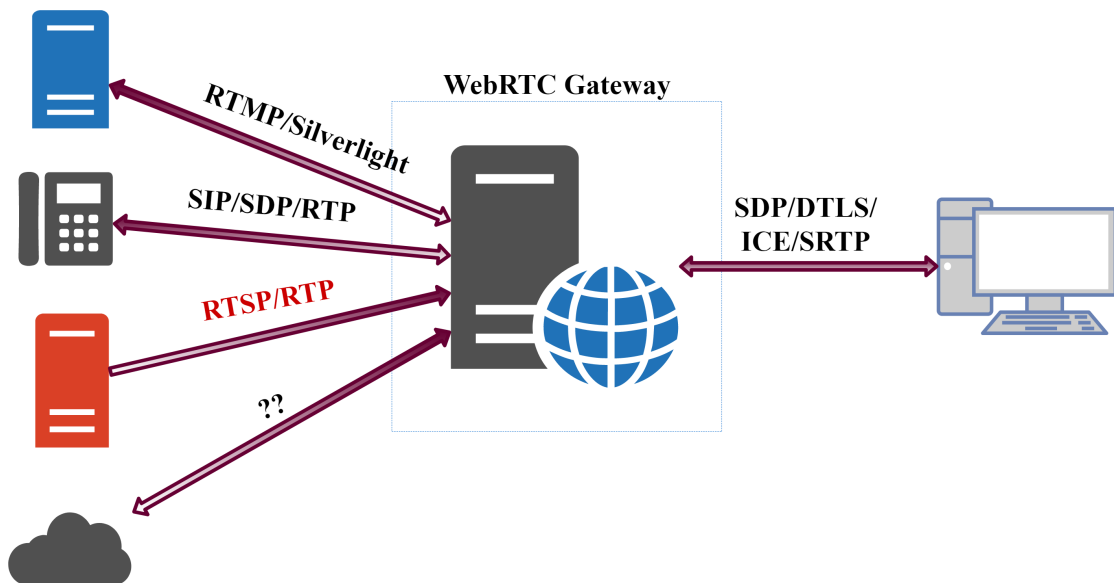


Figure 3.4: WebRTC gateway and different technologies

As mentioned earlier, a media gateway is very crucial in interoperating the communication between IP camera and WebRTC. Referring to Figure 3.4, it illustrates the communication process involving WebRTC and different technologies. The main reason for using a media gateway is due to the so-called legacy infrastructures out there. Legacy infrastructure is basically the system that uses older method, technology or application. This legacy system uses protocols like SDP, RTP, and others. For instance, if an existing SIP infrastructure is to communicate with WebRTC, the standard differences between SIP and those implemented by WebRTC endpoints causing difficulties in implementing communication.

Besides, media encryption is also an important factor causing compatibility issue. Most legacy components lack of media encryption supports. Whereas WebRTC uses DTLS to establish a secure media connection and DTLS is a protocol that is not popular in existing communication framework [60]. Incompatibility issue occurs in other aspects as well, WebRTC endpoint uses ICE for NAT traversal, RTCP feedback messages for



relaying connection status and RTP/RTCP muxing. However, the existing system usually depends on simpler approaches such as Hosted NAT Traversal (HNT) in Session Border Controller (SBC).

In this research, the use of media gateway is to solve the compatibility issue of RTP/RTSP stream and also the media feed provided by the camera has to be in WebRTC codec and browser compatible format. Therefore, Janus WebRTC gateway [61] is the gateway used in this system.

### **Janus WebRTC Gateway**

Janus WebRTC Gateway served as a general purpose gateway. Its primary means is to set up a WebRTC media communication with a browser. During the communication, JSON messages are exchanged and RTP/RTSP messages are relayed between browsers and server. Extra features and applications need to be implemented at the server side. These features can be in plugins formats like media recorders, SIP gateways and more. The main core of the gateway handles the gateway initialization (command line/ configuration file), setup and uses available transport plugins, for instance, HTTP (default), WebSockets, RabbitMQ and Janus protocol itself to communicate with applications, in this case, the web server. Besides, it also deals with peers and plugins in terms of both messaging and live media transfer through WebRTC.

### **3.3 WebRTC Web Server**

According to figure 2.6, WebRTC is basically used for conference call that initiate a room enabling users to enter and exchange multimedia contents. It is a multiple peer to a single web server design. However, for this research, it is a single P2P communication. Henceforth, the WebRTC web server is created based on figure 3.5. Details of the process flow is as below.

The WebRTC streaming for this system at the web server end was implemented by using 2 JavaScripts. The main API includes all primary WebRTC functionality like plugin initializations, peer connection set up, WebSocket handler, session setup, and multimedia player. Besides, it is also able to communicate with Janus WebRTC gateway to retrieve the multimedia data packages. The second JavaScript is the implementation of these functions. It uses the APIs to setup a real WebRTC live streaming process for

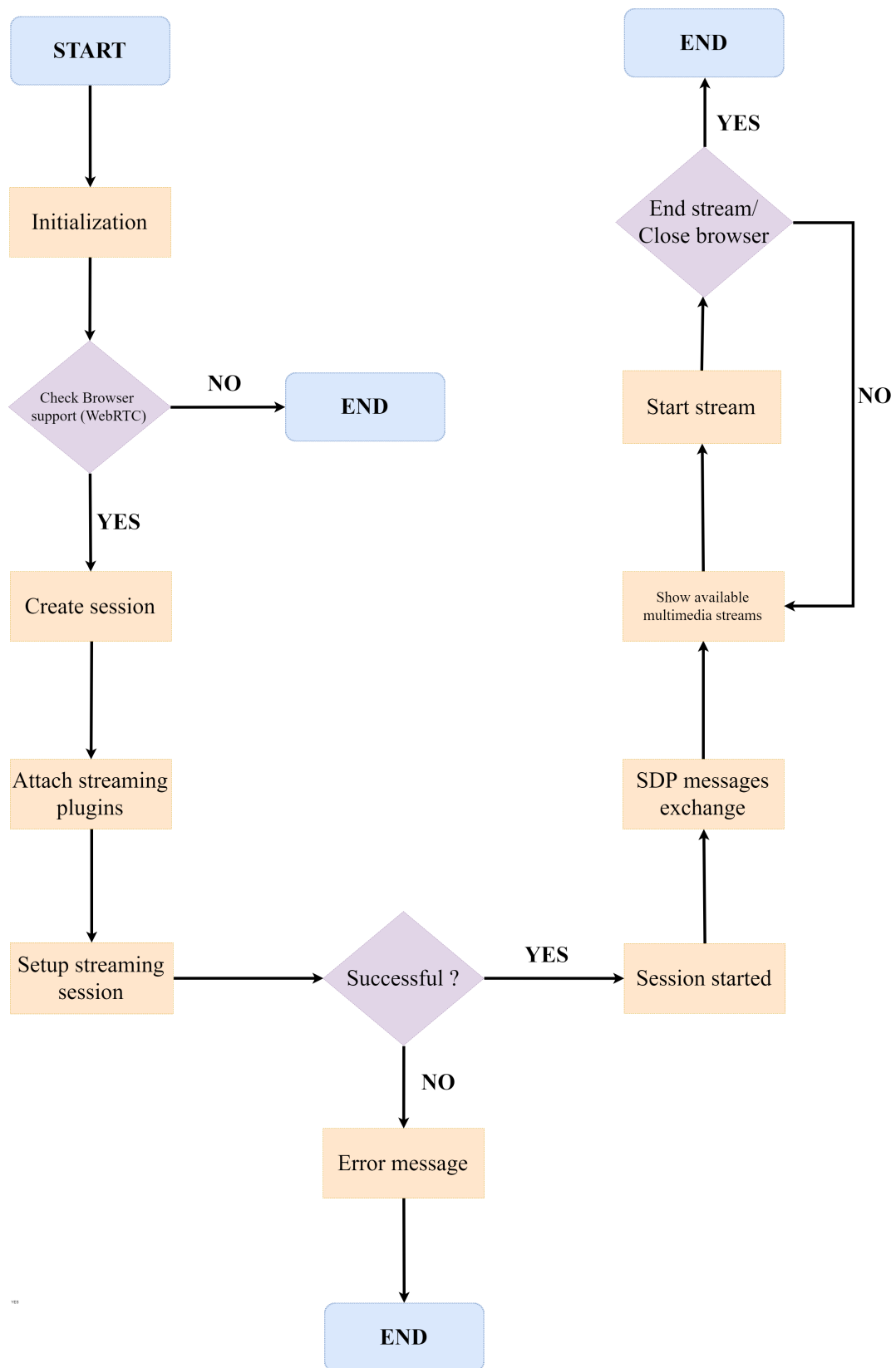


Figure 3.5: WebRTC Live Streaming Flow Chart

this web server. Figure 3.5 shows the process flow of the WebRTC streaming process based on our implemented server. Details of each step will be discussed as follow.

The first step of the WebRTC streaming process is to initialise several important libraries, enumerate devices and setting up console debugging and logging details. Next up, a specific function will check whether the client's browser supports WebRTC streaming. If the web browser do not support the browser, an error code will appear, else the process continues. A WebRTC session will be created that includes the session ID. Session ID carries important details for the server to identify what and which media to retrieve. Next, streaming plugins are attached. Streaming session is setup, it contains the stream list of the event. If the process is successful, the session is started. SDP messages containing the video stream details are exchanged. A list of available multimedia stream will be sent to client. Client is able to choose the available stream and video data transmission is started. The whole streaming process will end after the browser is closed.

Below listing shows the partial code that handles different task during the WebRTC streaming process.

**Listing 3.2:** Function to check WEBRTC availability

```
return window.RTCPeerConnection !== undefined && window.
    RTCPeerConnection !== null &&
        navigator.getUserMedia !== undefined && navigator.
            getUserMedia !== null;
```

The above code checks the availability of RTCPeerConnection and getUserMedia in order to know whether a WebRTC communication is being established successfully. For the case that a not compatible browser is used, an error message will appear, the whole streaming process will be terminated. Otherwise, after checking the compatibility, a session is created. This session attached Janus streaming plugin to the browser.

Following on, a streaming session is created. This part of the code deals with the stream list. It will retrieve the stream list from the Janus Gateway and display it on the web browser. The stream list can also be updated anytime. The communication between the web server and gateway uses JSON. Below shows a partial code handling the update process of a stream list.

**Listing 3.3:** Media Handler

```
function updateStreamsList() {
```

```

$('#update-streams').unbind('click').addClass('fa-spin');
var body = { "request": "list" };
Janus.debug("Sending message (" + JSON.stringify(body) + ")");
streaming.send({"message": body, success: function(result) {
    setTimeout(function() {
        $('#update-streams').removeClass('fa-spin').
            click(updateStreamsList);
    }, 500);
    if(result === null || result === undefined) {
        bootbox.alert("No response to our query for
            available streams");
        return;
    }
    if(result["list"] !== undefined && result["list"] !==
        null) {
        $('#streams').removeClass('hide').show();
        $('#streamslist').empty();
        $('#watch').attr('disabled', true).unbind('click
            ');
        var list = result["list"];
        Janus.debug(list);
        for(var mp in list) {
            Janus.debug(" >> [" + list[mp]["id"] +
                "]" + list[mp]["description"] + "
                (" + list[mp]["type"] + ")");
            $('#streamslist').append("<li><a href
                ='#' id='" + list[mp]["id"] + "'>" +
                list[mp]["description"] + " (" +
                list[mp]["type"] + ")" + "</a></li
                >");
        }
        $('#streamslist a').unbind('click').click(
            function() {
                selectedStream = $(this).attr("id");
                $('#streamset').html($(this).html()).
                    parent().removeClass('open');
                return false;
            });
        $('#watch').removeAttr('disabled').click(
            startStream);
    }
}

```

```

        });
    }
}

```

After the stream is up to date and displayed properly, SDP messages containing multi-media information are exchanged and we are able to start the real-time video streaming. The whole streaming process will end after the client closes the particular web page. An example of the start and stop function for playing the stream is as follow:

**Listing 3.4: Stream Handler 1**

```

function startStream() {
    if(selectedStream === undefined || selectedStream === null) {
        bootbox.alert("Choose a stream from list");
        return;
    }
    $('#streamset').attr('disabled', true);
    $('#streamslist').attr('disabled', true);
    $('#watch').attr('disabled', true).unbind('click');
    var body = { "request": "watch", id: parseInt(selectedStream) };
    streaming.send({"message": body});
    $('#stream').append('<video class="rounded centered" id="
        waitingvideo" width=320 height=240 />');
    if(spinner == null) {
        var target = document.getElementById('stream');
        spinner = new Spinner({top:100}).spin(target);
    } else {
        spinner.spin();
    }
}
}

```

**Listing 3.5: Stream Handler 2**

```

function stopStream() {
    $('#watch').attr('disabled', true).unbind('click');
    var body = { "request": "stop" };
    streaming.send({"message": body});
    streaming.hangup();
    $('#streamset').removeAttr('disabled');
    $('#streamslist').removeAttr('disabled');
    $('#watch').html("Watch or Listen").removeAttr('disabled').click
        (startStream);
    $('#status').empty().hide();
}
}

```

### 3.4 Methodology and Experimental Setup

Due to network congestion, configuration errors and incorrect queuing. Jitter causes gaps in multimedia streaming, which further impacts the user's QoE. WebRTC measures jitter according to RFC3550 [62], difference of arrival time  $D(i,j)$  is computed as below.

$$D(i, j) = (R_j - S_j) - (R_i - S_i) \quad (3.4.1)$$

$i$  and  $j$  indicate two packets which are received consecutively.  $S_i$  and  $S_j$  are the timestamps sent from the source, where  $R_i$  and  $R_j$  represents the arrival time of packets  $i$  and  $j$ . The equation below shows the inter-arrival jitter for packet  $i$ .

$$J(i) = \frac{J(i-1) + (|D(i-1, i)| - J(i-1))}{16} \quad (3.4.2)$$

$J(i-1)$  is the inter-arrival jitter value of the previous packet. Where  $D(i-1, i)$  is the difference of arrival time between packets  $i-1$  and  $i$ . In order to ensure the output video quality of the streaming system, experiments were carried out on a specific testbed with a fully controllable environment. The main goal of these experiments is to measure the performance of WEBRTC real-time streaming, and also to compare it with other streaming technique. Studies are conducted as follow:

- Phase 1: Acquire a broad range of WEBRTC video streaming data using different tools.
- Phase 2: Setup another type of streaming system (FFserver), obtain its results and compare them with phase 1.
- Phase 3: Perform QoE measurement by comparing the recordings of the output video and the original video captured directly from the camera.

This chapter explained the details of the experimental setup, network topology, hardware and software used and also includes background information of various important parameters used in the experiments.

### 3.4.1 Performance Measures

Many parameters may impact directly or indirectly to the audio and video quality of a multimedia conversation. Hence, it is important to control and record these parameters in order to interpret quality. The important factors that define QoE are listed and explained in the following subsections.

#### Bandwidth

Bandwidth is the measurement of available bit-rate in a network [63]. In this case, bandwidth is measured in either Kilobit per second (Kb/s) or Megabits per second (Mb/s). Taking into account that bandwidth often varies depending on the Internet connection, all of the experiments were conducted locally. Hence, the Internet connection will not be the factor that affects directly to the quality of the output video. At the user end, bandwidth was recorded so as to define the quality of the received video stream.

#### Frame Rate

Frame rate, expressed in Frame Per Second (FPS), defines the number of images that are projected in animated display per second. It is important in synchronizing audio and video. In order to achieve a rather high quality video output, its frame rate has to exceed 24fps [64]. 24fps is the most accepted film frame rate as movie theaters internationally use this frame rate. Hence, in order to have a better video quality, higher output frame rate has to be ensured.

#### Packet Loss

Packet loss counts the number of packets which failed to reach their destination. It is measured in percentage of lost packets. According to [65], packet loss in WebRTC is measured through expected received packets minus the actual received packets which also includes late and duplicate packets. Additionally, the amount of expected packets is calculated by the using the highest sequence number received. Packet loss occurs mainly because of two reasons:

- (i) Congestion: Packets dropped while traveling across the network due to over-

crowding in one or more network devices.

- (ii) Corruption: Malformed packets or corrupted signal when passing through the network. Hence packet loss provides us with knowledge of the network condition and the consistency of the streaming server.

### **Delay**

End-to-end delay defines the time taken for a packet to travel from its source to its destination. Another term which is more widely used to describe delay is Round Trip Time (RTT). RTT measures the time packet takes from the source to destination and back to the source. Packets delay are classified into different category [66]: 1) Serialization delay 2) Propagation delay and 3) Switching delay. A long delay will cause late play-out at the receiver's end meaning that there will be a relatively long and unnatural pause in between a communication.

### **Jitter**

Jitter is a kind of time noise which can be positive or negative. It is defined as the variation in the delay between consecutive packets. Jitter involves in packet reordering, typically i) is the difference in arrival time and the previously sent packet. Finally,  $1/16$  is a noise reduction parameter.

In WebRTC protocol, multimedia is transmitted separately in audio and video. Meaning each of them has its own jitter buffer. As stated in WebRTC architecture [67] jitter buffer for audio is an error concealment algorithm which is used to conceal the negative effects of network jitter and packet loss. It keeps latency as low as possible while also secures the highest audio quality. Similarly, jitter buffer in video also moderates the impact of jitter and packet losses have on video quality.

### **CPU-Usage**

CPU-usage is an important factor that decides how many video channel to be streamed at the server side. When doing the comparison between the performance of WebRTC and FFserver, CPU-usage is used in justifying the consistency of each application. Besides, at the user end, devices with lower CPU-usage is used which enables us to determine how different hardware specifications can affect QoE.



### 3.4.2 Network Topology

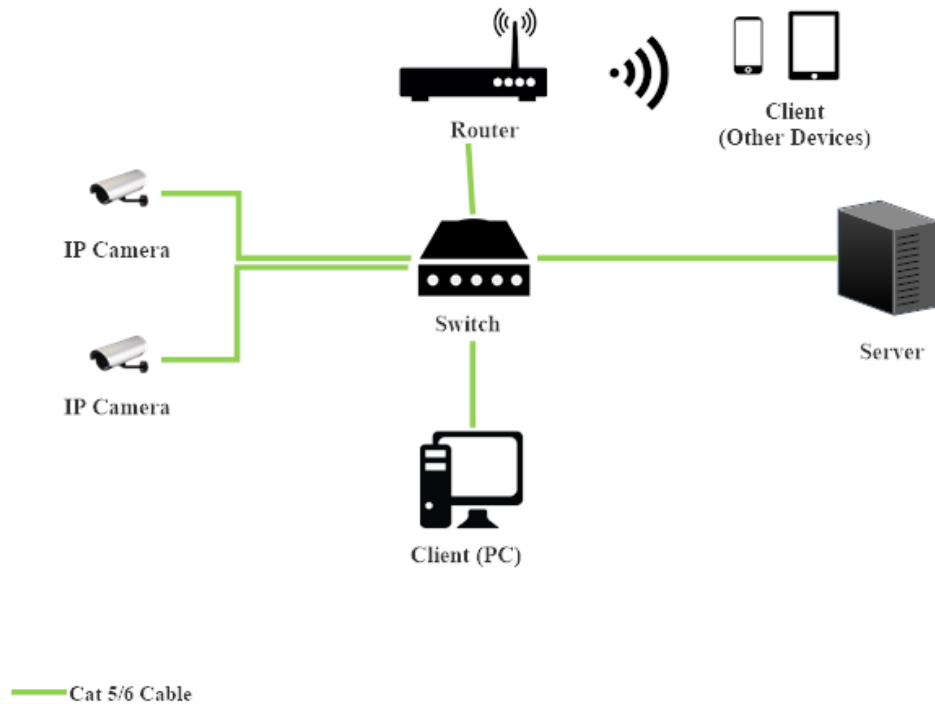


Figure 3.6: Experiment topology

Figure 3.6 illustrates the experiment topology used in setting up all the instruments. Two high definition IP cameras are used as the source. They sent real-time H264 video data and are connected and powered by a PoE switch. Server acted as the main controller among all devices. It manages camera information, configures the cameras and most importantly hosts the streaming service to the clients. Two types of clients are used to test the video streams: 1) Laptop 2) Android phone. Clients are not directly interconnected, an android phone is connected through a WIFI access point where the laptop is connected by a Local Area Network (LAN) cable to the switch. The block diagram below demonstrates the general process flow of the topology.

### 3.4.3 Hardware Specifications

**Server details** This server is configured with Ubuntu 14.04.5 LTS with specifications as follow:

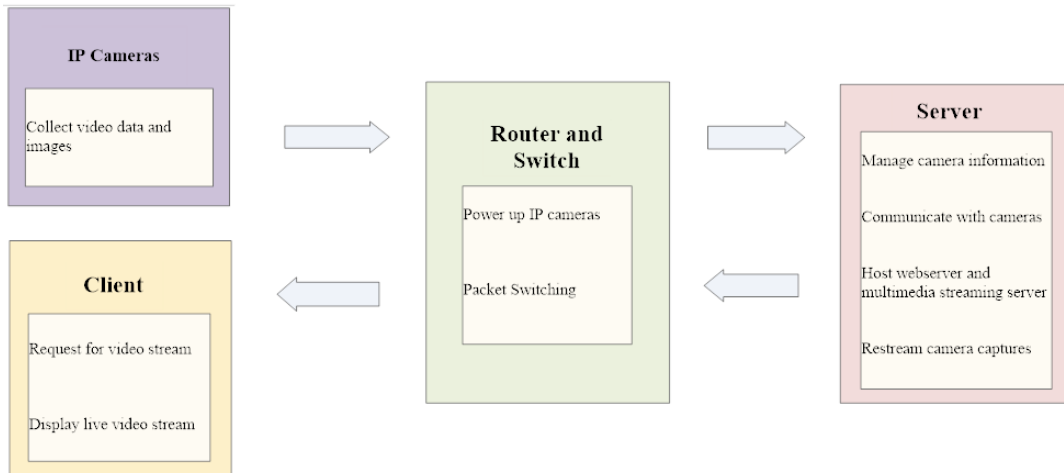
**IP camera** The first IP camera used in the entire experiment is Axis M3005-V fixed dome network camera and the second IP camera used is Hikvision DS-2CD2132-I IR fix dome camera. Their specifications are as follow:

**Table 3.1:** Server Specifications

Computer	Dell Precision T7610
Processor	12x Intel(R) Xeon(R) CPU E5-2630 v2 @ 2.60GHz
Memory	24632MB
Processor details	Intel(R) Xeon(R) CPU E5-2630v2 @ 2.60GHz 2904.48MHz x8
Network Controller	Intel Corporation 82579LM Gigabit Network,Connection, Intel Corporation 82599 10 Gigabit TN Network,Connection, Intel Corporation 82599 10 Gigabit TN Network,Connection, Intel Corporation I210 Gigabit Network,Connection

**Table 3.2:** IP Camera Specifications

Camera Name	Axis M3005-V fixed dome network camera	Hikvision DS-2CD2132-I IR fix dome camera
Video Compression	H.264 Main and Baseline Profile (MPEG-4 Part 10/AVC), Motion JPEG	H.264 / MJPEG
Resolutions	1920x1080 (HDTV 1080p) to 320x240	Maximum 2048 x 1536
Frame Rate	25/30 fps with power line frequency 50/60 Hz	50Hz: 20fps (2048 x 1536), 25fps (1920 x 1080), 25fps (1280 x 720), 60Hz: 20fps (2048 x 1536), 30fps (1920 x 1080), 30fps (1280 x 720)
Bit rate	Adjustable	32 Kbps ~ 16 Mbps
Video Streaming	Multiple, individually configurable streams,in H.264 and,Motion JPEG, Controllable frame rate and,bandwidth, VBR/CBR H.264	N/A



**Figure 3.7:** General process flow of the topology

**Clients** Two client devices are used in retrieving the test stream sent by the server. They are laptop: Qosmio F750 and android phone: HTC M9+.

**Table 3.3:** Client Specifications

Device Model	Qosmio F750	HTC One M9Plus
Operating System	Microsoft Windows 10 Home	Android version 6.0
Processor	Intel(R) Core(TM) i5-2430M CPU @ 2.40GHz, 2401 Mhz, 2 Core(s), 4 Logical Processor(s)	8x ARM Cortex-A53 @ 2.16 GHz
Memory	7.91 GB	2715MB
Network Controller	Realtek PCIe GBE Family Controller, Qualcomm Atheros AR9002WB-1NG Wireless Network Adapter Switch	Wi-Fi 802.11 a/b/g/n/ac, dual-band, Wi-Fi Direct, DLNA, hotspot

**Switch** Cisco SF300-48P switch is used throughout the entire experiments. Specifications are listed as below:

**Table 3.4:** Switch Specifications

Model Name	SF300-48P
Performance	Capacity: 13.10 Millions of Packets, per Second (mpps) (64-byte packets), Switching Capacity: 17.60, Gigabits per Second
Power Over Ethernet	48 ports (375W)
Ports	48 Fast Ethernet and, 4 Gigabit Ethernet
Flash Memory	16MB
CPU Memory	128MB

**Router** Details of the Linksys wireless-N Gigabit Router are as below:

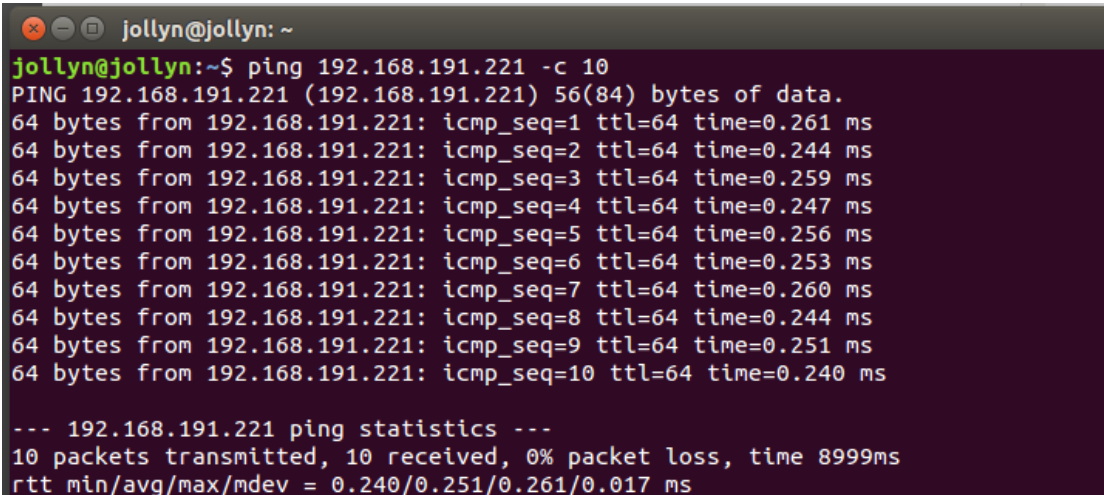
**Table 3.5:** Router Specification

Model Name	WRT310N
Standards	802.11n v2.0, 802.11g, 802.11b, 802.3, 802.3u, 802.3ab
Ports	Power, Internet, Ethernet
Interface	WAN: 1 x 10Base-T/100Base-TX/1000Base-T - RJ-45, LAN: 4 x 10Base-T/100Base-TX/1000Base-T -, RJ-45

### 3.4.4 Software Configuration

#### Optimal Network Condition

All experiments are performed under optimal network condition locally without any alterations. In order to fully test out the performance of the streaming server, factors such as delay, jitter, and packet loss were not taken into account. An ideal and stable network condition is also essential in comparing the performance between two streaming servers. Hence, several simple tests are performed to check the ping rate, routing, and bandwidth between the server and clients.



```

jollyn@jollyn: ~
jollyn@jollyn:~$ ping 192.168.191.221 -c 10
PING 192.168.191.221 (192.168.191.221) 56(84) bytes of data:
 64 bytes from 192.168.191.221: icmp_seq=1 ttl=64 time=0.261 ms
 64 bytes from 192.168.191.221: icmp_seq=2 ttl=64 time=0.244 ms
 64 bytes from 192.168.191.221: icmp_seq=3 ttl=64 time=0.259 ms
 64 bytes from 192.168.191.221: icmp_seq=4 ttl=64 time=0.247 ms
 64 bytes from 192.168.191.221: icmp_seq=5 ttl=64 time=0.256 ms
 64 bytes from 192.168.191.221: icmp_seq=6 ttl=64 time=0.253 ms
 64 bytes from 192.168.191.221: icmp_seq=7 ttl=64 time=0.260 ms
 64 bytes from 192.168.191.221: icmp_seq=8 ttl=64 time=0.244 ms
 64 bytes from 192.168.191.221: icmp_seq=9 ttl=64 time=0.251 ms
 64 bytes from 192.168.191.221: icmp_seq=10 ttl=64 time=0.240 ms

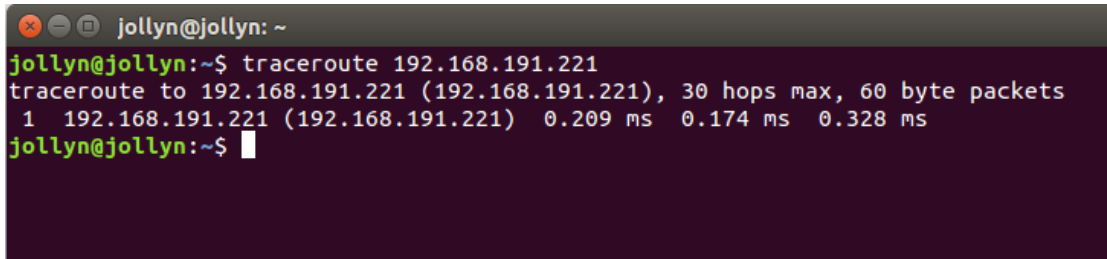
--- 192.168.191.221 ping statistics ---
10 packets transmitted, 10 received, 0% packet loss, time 8999ms
rtt min/avg/max/mdev = 0.240/0.251/0.261/0.017 ms

```

**Figure 3.8:** Ping rate from client to server

Ping basically sends Internet Control Message Protocol (ICMP) request to the desired destination and collects the response packet. It is useful in verifying Internet connection and provides us quick information about the average RTT and packet loss percentage. Referring to Figure 3.8, 10 packets were sent to test the ping rate. The average ping obtained is 0.251ms and 0% of packet loss.

Besides ping, traceroute is used to measure delay. From Figure 3.9, it can be observed that the packet is directly transmitted to 192.168.191.221 using just a single hop since



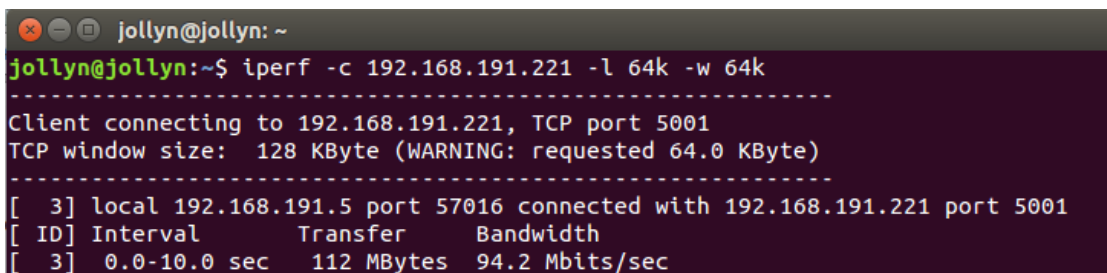
```

jollyn@jollyn: ~
jollyn@jollyn:~$ traceroute 192.168.191.221
traceroute to 192.168.191.221 (192.168.191.221), 30 hops max, 60 byte packets
 1 192.168.191.221 (192.168.191.221)  0.209 ms  0.174 ms  0.328 ms
jollyn@jollyn:~$

```

Figure 3.9: Traceroute result

the server and client is inter-connected by a switch. From this information, it is verified that the routing table of the client is set up properly. The RTT obtained is 0.209ms, 0.174ms and 0.328ms respectively.



```

jollyn@jollyn: ~
jollyn@jollyn:~$ iperf -c 192.168.191.221 -l 64k -w 64k
-----
Client connecting to 192.168.191.221, TCP port 5001
TCP window size: 128 KByte (WARNING: requested 64.0 KByte)
-----
[ 3] local 192.168.191.5 port 57016 connected with 192.168.191.221 port 5001
[ ID] Interval      Transfer    Bandwidth
[ 3] 0.0-10.0 sec   112 MBytes  94.2 Mbits/sec

```

Figure 3.10: IPerf result

IPerf is used mainly to measure the bandwidth of an IP network. It has to be installed on both client and server side. Using IPerf, it is able to select to use either UDP or TCP packet to measure the network. In this case it sent a TCP packet that is 64KB and the set the socket buffer size to 64KB. Packets are sent, and at the server side, a total of 112MB were received. The final bandwidth measured is 94.2Mbits/sec which is relatively near 100Mbits/s (Fastethernet).

### IP Camera Selections and Configurations

Throughout this entire thesis, two camera brands are used: Axis and Hikvision. Both companies are the world's market leader in network video. Axis is a company that invented the world's very first network camera back in 1996 ([68]), initiating the shift from analogue to digital. Where Hikvision is a Chinese company that focused in researching and developing continued product innovation ([69]). The main reason of choosing these two cameras branding is the support of various standards such as, ONVIF, H.264 transcoding, high resolution (FHD) video capture and RTP/RTSP. Besides, they also provides reliable after sales services.

As mentioned in the hardware details section, two IP cameras used are Axis and Hikvi-

sion. Both cameras are configured to have similar configurations through their available web GUI at the server side, Since I set an average bit rate for both camera to be 2048kb/s, Hikvision has restricted the frame rate to 22f/s, There are only several frame rates to choose and among all, the highest is 22f/s. Configurations of both cameras are simplified in the table as follow:

**Table 3.6: IP Camera Configurations**

Camera Model	AXIS M3005	HIKVISION DS-2CD2132-
Resolution	1920x1080	1920x1080
Output Codec	H264	H264
Frame rate	25f/s	22f/s
Output Bitrate	2048kb/s (Variable Bitrate)	2048kb/s (Variable Bitrate)

### 3.4.5 Data Acquisition

This section discussed the tools used in acquiring data during the streaming process. There are two ways used in collecting information regarding the WebRTC connection quality, 1) through session-related technical statistics and 2) packet captured at the client end.

#### Browser Debug Tools

Browsers that support WebRTC streaming are Google Chrome and Mozilla Firefox. Both of them offer a user interface that gathers statistics of WebRTC communications. Chrome's debug tool is named webrtc-internals where Firefox is about:webrtc.

#### Chrome: webrtc-internals

Figure 3.11 shows the snapshot of a webrtc-internals user interface. Each section numbered provides different information on the currently connected WebRTC stream [67]:

- (i) The address indicates the RTCPeerConnection that it is currently connected to. Section 2-5 contained information related to this specific RTCPeerConnection.
- (ii) This part of the user interface shows how the RTCPeerConnection is configured. In details, the STUN and TURN server used and their configurations.
- (iii) Records the timestamps and traces of the PeerConnection API calls, arguments, callbacks as well as event emitters.

(iv) This section arranges the statistics gathered from getStats() API.

(v) Shows the graphs generated from getStats() API.



Figure 3.11: A snapshot of webrtc-internals

Section 5 consists of important information in defining streaming quality. Information such as bits received per second, packet loss, packets received per second, delay, jitter, frame rate and more are well arranged to graphs, enabling users to easily analyze. Besides, webrtc-internals is able to generate a statistic file that gathers the displayed metadata for all locally connected peer connections. However, there is a limitation on this feature, it only takes maximum 1000 of samples per file with a sampling rate every second. Meaning that the dump file has to be downloaded in the middle of an active conversation if it lasts longer than 16 minutes. It is not possible to retrieve the statistic file after the communication was closed.

The dump file is presented in JSON format with no timestamps specified. Due to the

fact that the time information of the sampling rate is not included, all the data value of each generated dump file has to be examined. In this case, one dump file is retrieved in 4 minutes for a total time of 1 hour. Approximate 3640 samples are taken in a duration of 3600s. Calculation result shows that a sample was taken every 0.989 seconds. The uneven sampling rate may be caused by an unsynchronized time between the server and client. Hence, the sampling rate is assumed to be 1s throughout the experiments.

### Mozilla Firefox: about:webrtc

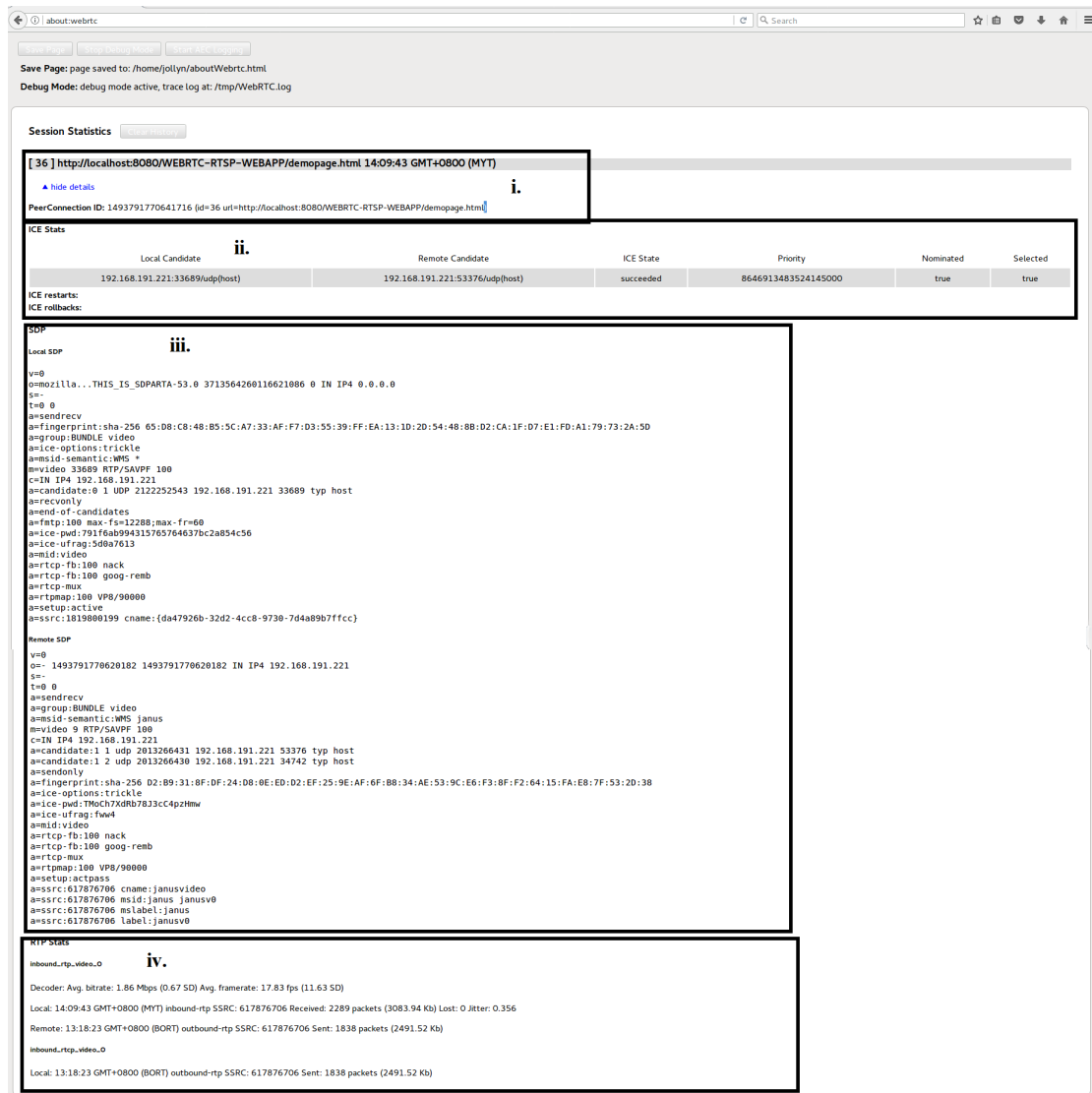


Figure 3.12: A snapshot of Firefox about:webrtc

Mozilla Firefox also includes a debugging interface for WebRTC. As you can see in Figure x, it has a totally different presentation compared to webrtc-internals. Information shown is more simplified.



- (i) Indicates the source address and the PeerConnection ID.
- (ii) ICE statistics including the information of local and remote candidate, and their status.
- (iii) Basically shows information on SDP. For example, ICE candidates, and parameters, codec parameters, video and audio lines.
- (iv) General audio and video statistics.

In Firefox `about:webrtc`, multimedia statistics such as bitrate, frame rate, packet loss, and delay are shown in average value throughout the streaming period. Extracting data file is also not supported by Firefox, making it challenging to obtain a large sample of needed data. Henceforth, Firefox is not used as the main client web browser, the entire video streaming will be done on Google Chrome instead. Table 3.7 shows the comparison of properties in `webrtc-internals` and `about:webrtc`.

**Table 3.7:** WebRTC Internal Tool: Chrome versus Firefox

	Chrome <code>webrtc-internals</code>	Firefox <code>about:webrtc</code>
Time (start, end, total duration)	✓	✓
SDP status	✓	✓
RTP/RTCP status	✓	✓
Connection log	✓	✓
Bandwidth	✓	✓
Latency	✓	✓
Throughput	✓	✓
Packet loss	✓	✓
Jitter	✓	✓
Sampling period	1 second	Only an average value given
Graphical display	✓	✗
Method to retrieve data files	✓	✗

### Wireshark

Besides using the `webrtc-internals` API mentioned in the previous session, a network packet analyzer is used, named Wireshark [70]. Wireshark is a network analyzer that is free and open-source. It is a great tool for troubleshooting network errors, debugging protocols and examining security problems. In this case, Wireshark is used mainly to analyze communication and video packets of FFserver. Results of WebRTC and FF-server captured by Wireshark are presented in later chapter of this thesis.

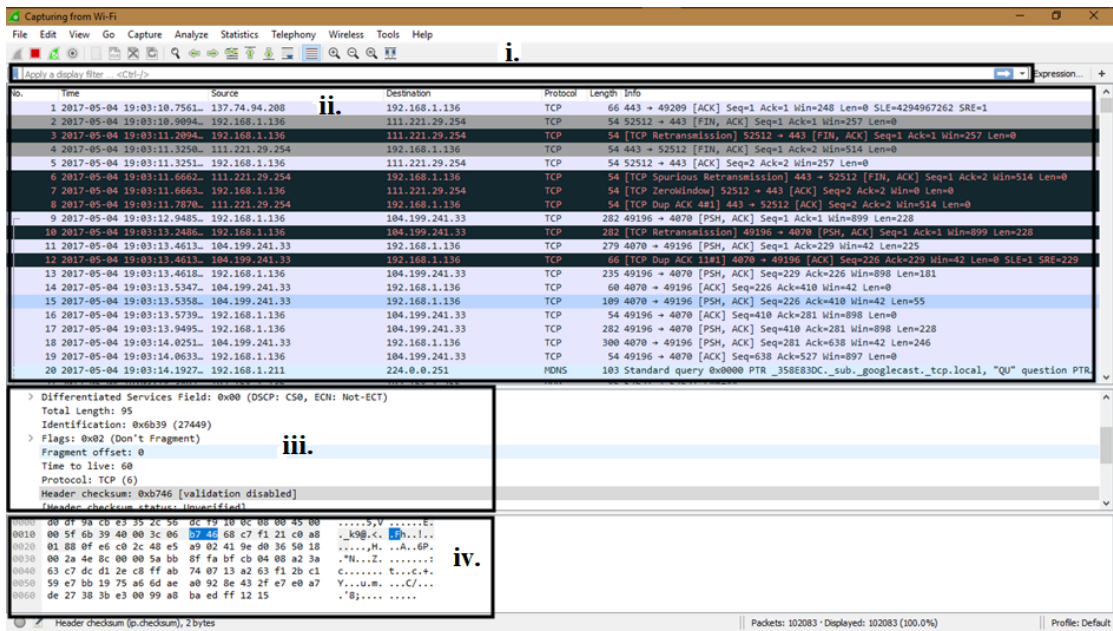


Figure 3.13: Wireshark GUI snapshot

Wireshark provides several handy features for video data interpretation [71]. To begin with, a conversation function under ‘Statistics’ menu filters all the captured into categories based on their protocol. From there, it is able to choose desired packets based on port numbers, destination or source IP address. All the packets with selected information will be shown. The next function found to be valuable is the I/O graph feature. I am able to generate graphs containing information like bit rate and packet rate according to the filtered packets. Additionally, those data can even be exported into different formats. Finally, a function called protocol hierarchy is used frequently. It displays a hierarchical tree of protocol statistics indicating the percentage of packet size received from different protocols. Figure 3.13 is a snapshot illustrating the user interface of Wireshark:

- (i) Filter toolbar
- (ii) Packet List Panel
- (iii) Packet Details Panel
- (iv) Packet Bytes Panel

### 3.4.6 Video Quality Assessment

Subjective Video Quality Assessment (VQA) is a method that applies human perspective in measuring visual quality. In fact, VQA that involves human is very challenging and

expensive. It is impossible for a human to subjectively assess a large amount of videos in a short period of time. Besides, these subjective studies have to be conducted in a controlled environment due to inconsistent viewing condition such as display device, viewing distance, ambient illumination, and more [72]. Henceforth, objective VQA algorithms are created in order to automatically measure the visual quality of videos and most importantly to avoid human involvement.

In order to perform the subjective VQA studies, the primary video source is captured from both IP camera. The transcoded videos for WebRTC streaming were also recorded for VQA comparisons. I evaluate the videos using four publicly available VQM models. They are Peak Signal to Noise Ratio (PSNR), Structural SIMilarity (SSIM) and Video Quality Metric (VQM),

### Peak Signal to Noise Ratio (PSNR)

PSNR is a kind of data fidelity metric which calculates the physical difference of two signals without considering the content. It is derived as [73]:

$$\text{MeanSquaredError}(MSE) = \frac{1}{N} \sum_{i=1}^N (X_i - Y_i)^2, \quad (3.4.3)$$

$$PSNR = 10 \cdot \log_{10} \frac{255^2}{MSE} \quad (3.4.4)$$

Where N indicates the total number of pixels in the video, 255 is the maximum intensity value of the image,  $X_i$  is the i-th pixel of the original video and  $Y_i$  similarly is the i-th pixel of the distorted video.

The difference between MSE and PSNR is that MSE measures image differences where PSNR measure image fidelity. Both models used an original image as a reference and the other to compare. Although MSE and PSNR are well-known with their simplicity and physical significance, it is proven that their accuracy in subjective rating is limited [74]. The main reason why these models do not perform is that their computation lacks Human Visual System (HVS) features. Studies showed that in the primary visual cortex of human images are not represented in pixel format. The weakness of MSE and PSNR is they only compute pixels of images but no other information perceived by a human.

### Structural SIMilarity (SSIM)

Structural SIMilarity uses the concept that natural images are very much structured. In other words, images hold pixels that exhibit strong dependencies carry crucial information about the structure of the objects visually [75]. This model is built by considering the human vision system in which it did not focus on extracting errors but specialized in obtaining structural information based on human viewing perspective. Thus SSIM is capable in providing better correlation in terms of subjective impression. The expressions below,  $l(x,y)$ ,  $c(x,y)$ , and  $s(x,y)$  are used by SSIM to compute luminance, contrast, and structure respectively.

$$l(x,y) = \frac{2\mu_x\mu_y + C_1}{\mu_x^2 + \mu_y^2 + C_1'} \quad (3.4.5)$$

$$c(x,y) = \frac{2\sigma_x\sigma_y + C_2}{\sigma_x^2 + \sigma_y^2 + C_2'} \quad (3.4.6)$$

$$s(x,y) = \frac{\sigma_{xy} + C_3}{\sigma_x\sigma_y + C_3} \quad (3.4.7)$$

Where  $\mu_x$ ,  $\mu_y$ ,  $\sigma_x$ ,  $\sigma_y$  and  $\sigma_{xy}$  are the local means and standard deviations for images  $x$  and  $y$ . Additionally,  $C_1$ ,  $C_2$  and  $C_3$  are small constants derived by

$$C_1 = (K_1 \cdot L)^2 \quad (3.4.8)$$

$$C_2 = (K_2 \cdot L)^2 \quad (3.4.9)$$

$$C_3 = \frac{C_2}{2} \quad (3.4.10)$$

$L$  represents the dynamic range of the image pixel values, for example, a 8 bits/pixel gray scale images,  $L=255$ . Where  $K_1 \ll 1$  and  $K_2 \ll 1$  are small constants. The general SSIM expression is as below:

$$SSIM = [l(x,y)]^\alpha \cdot [c(x,y)]^\beta \cdot [s(x,y)]^\gamma \quad (3.4.11)$$

where,  $\alpha > 0$ ,  $\beta > 0$  and  $\gamma > 0$  are parameters used in modifying the relative importance of luminance, contrast and structure. If  $\alpha = \beta = \gamma = 1$ , the above expression is simplified into:

$$SSIM = \frac{(2\mu_x\mu_y + C_1)(2\sigma_{xy} + C_2)}{(\mu_x^2 + \mu_y^2 + C_1)(\sigma_x^2 + \sigma_y^2 + C_2)} \quad (3.4.12)$$

The value of SSIM is  $1 \geq SSIM \geq -1$ , when the original and the distorted image are the same,  $SSIM=1$  indicating the best value achieved.

### Video Quality Metric (VQM)

VQM has been adopted by ANSI as the standard of objective video quality. Past studies showed that VQM presented a high correlation with subjective VQA [76]. It uses an algorithm that measures the perceptual effects of video impairments like blurring, jerky/unnatural motion, block distortion, colour distortion, and global noise. These measurements are combined forming a single metric that is able to predict the overall quality. VQM compute the input video through several stages [77]:

- (i) Calibration: This stage prepares the video for feature extraction. It calibrates the spatial and temporal shifts, the contrast and brightness offset of the processed video to the original video sequence.
- (ii) Quality Features Extraction: Quality features that characterize perceptual changes in spatial, temporal and chrominance domains are extracted from the spatial-temporal sub-regions of the video. These features are extracted using mathematical function and a visibility threshold is applied to them.
- (iii) Quality Parameters Calculation: This stage calculates a set of quality parameters that describe perceptual changes in video quality through comparing features extracted from the processed and reference video.
- (iv) VQM Calculation: The last stage is to compute an overall quality metric using a linear combination of parameters from the previous stages.

# Results and Discussion

This chapter presents the results of a general WebRTC performance including both real time streaming and streaming from standard video files. Besides, it also includes comparisons of WebRTC and FFserver followed by the WebRTC video quality assessment.

## 4.1 General Real Time WEBRTC Streaming Performance

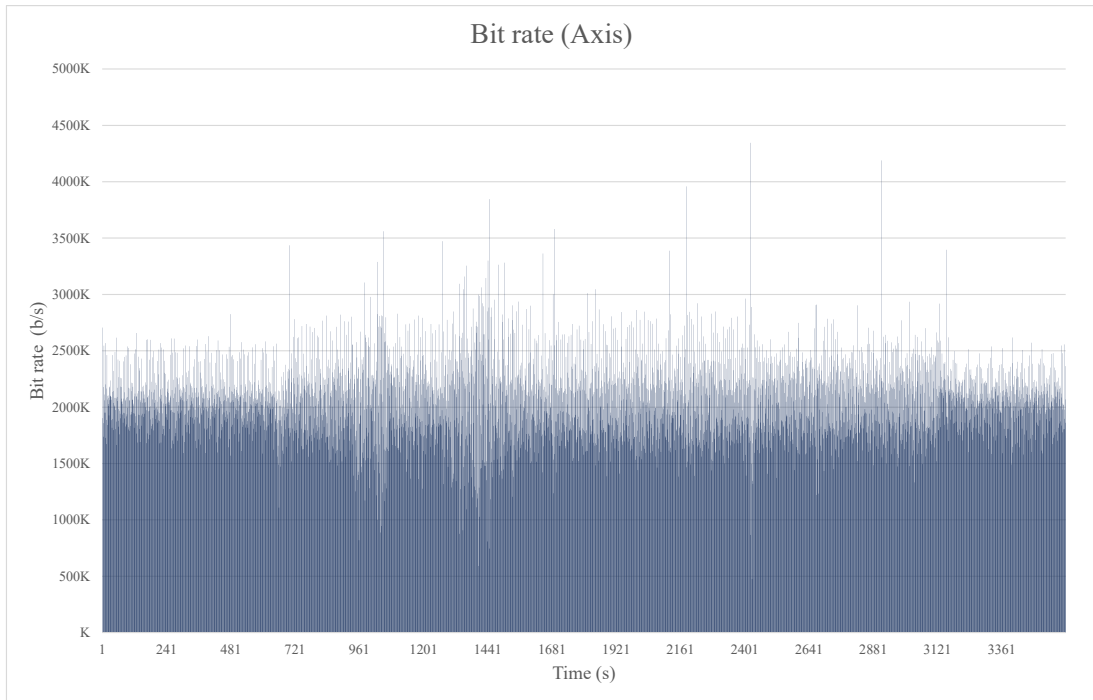
As discussed in the previous chapter, the general performance of WebRTC streaming is measured using the webrtc-internals debugging tool found in Google Chrome. Several important parameters are taken into account, they are bitrate, packet rate, packet loss, frame rate, jitter, and delay. The recording environment is carried out in a lab which involves normal human activities. Each data are recorded throughout a period of one hour using webrtc-internals and presented in graph under each section below.

### Bit Rate

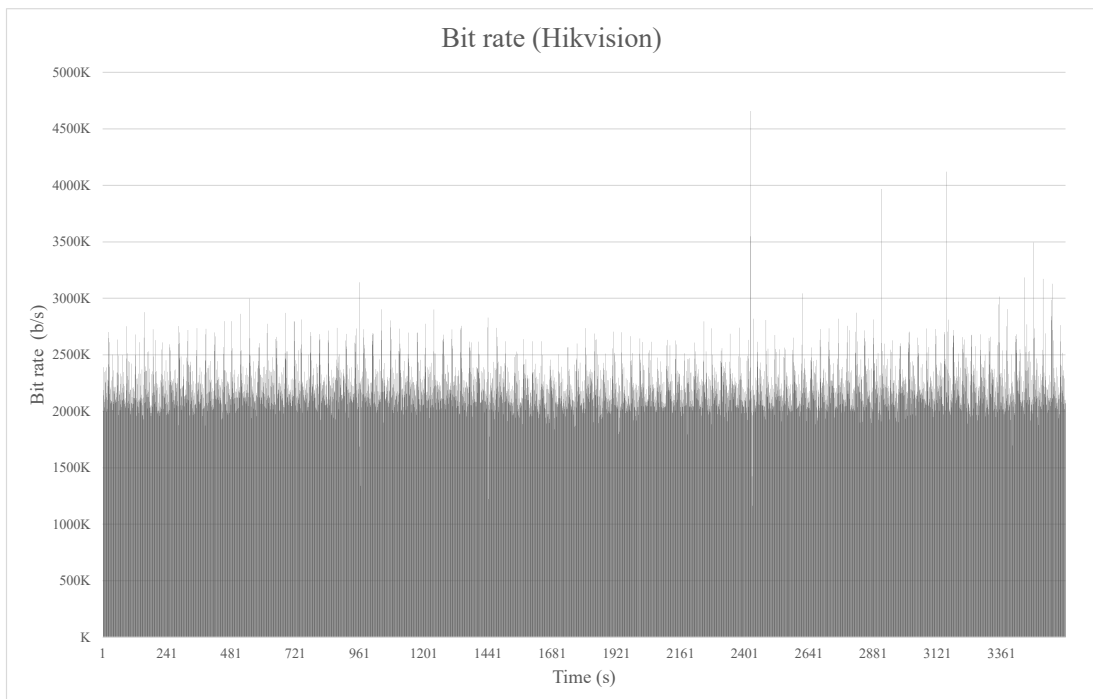
Both cameras are configured with an encoding rate of 2048kb/s (VBR), the reason behind that is because variable bit rate not only provides higher quality video but also balances quality and file size. During the live streaming process, video sources are transcoded into similarly 2048kb/s bit rate using VP8 codec. Other parameters such as frame rate and resolutions are retained.

Referring to graph 4.1 and 4.2, they illustrate the received bit rate pattern of Axis camera and Hikvision camera respectively. The highest bit rate Axis achieved is 4345Kb/s at 2422s and Hikvision 4658Kb/s at 2424s. Where the lowest bit rate for Axis is 475Kb/s at 1428s and Hikvision is 1164Kb/s at 2430s. The average bit rate of Axis is 2070804b/s

and Hikvision is 2219571b/s. Overall, Hikvision camera shows a slightly higher bit rate compared to Axis.



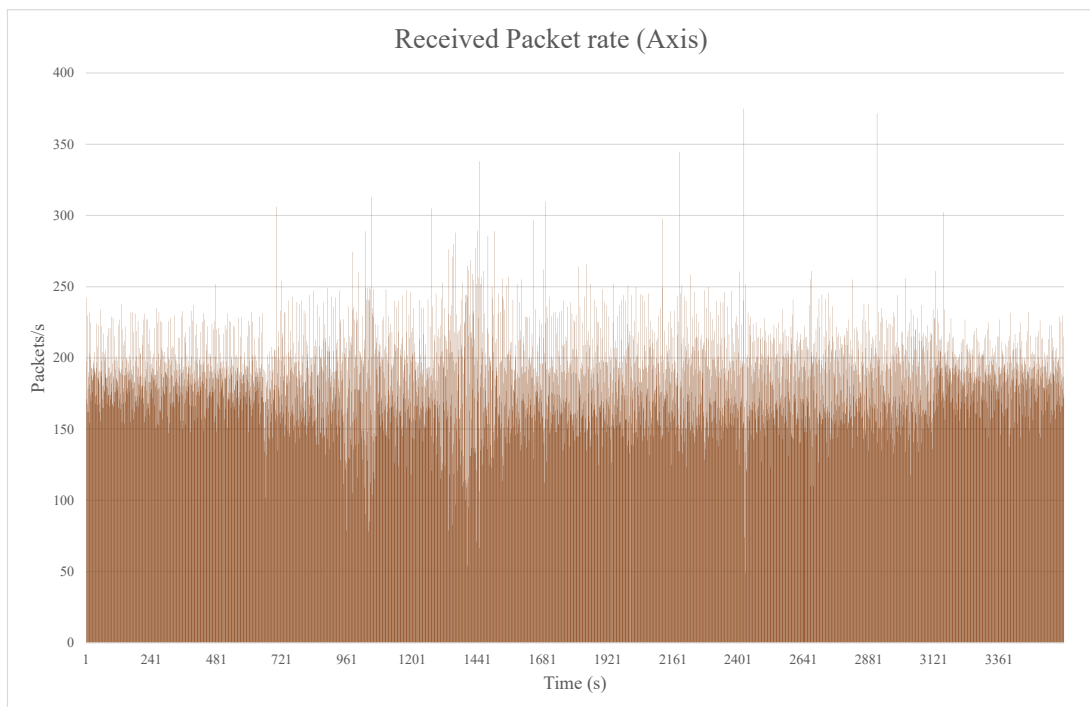
**Figure 4.1:** Graph of Bit rate (Axis Camera)



**Figure 4.2:** Graph of Bit rate (Hikvision Camera)

### Packet Rate

The obtained packet data shows that Axis has received an average of 185.71 packets per second where Hikvision has received mean packet rate of 203.43 packets per second. The peak packet rate hits 375 packets per second at 2424s (Axis) and 407.41 packets per second at 2424s (Hikvision).

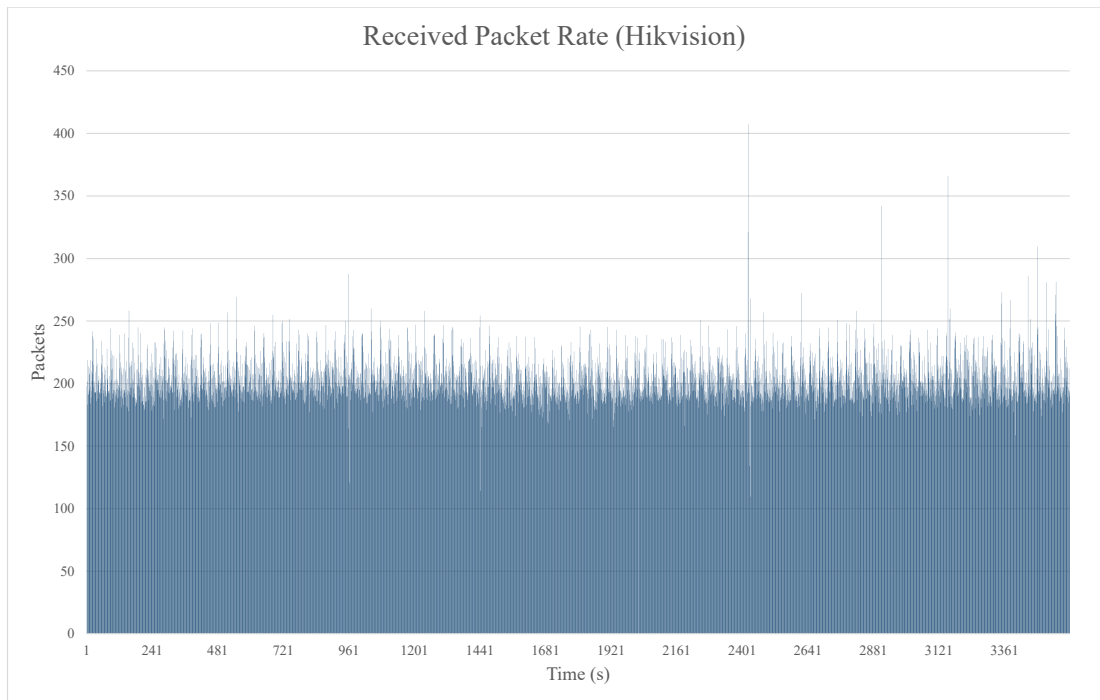


**Figure 4.3:** Graph of Packet rate (Axis Camera)

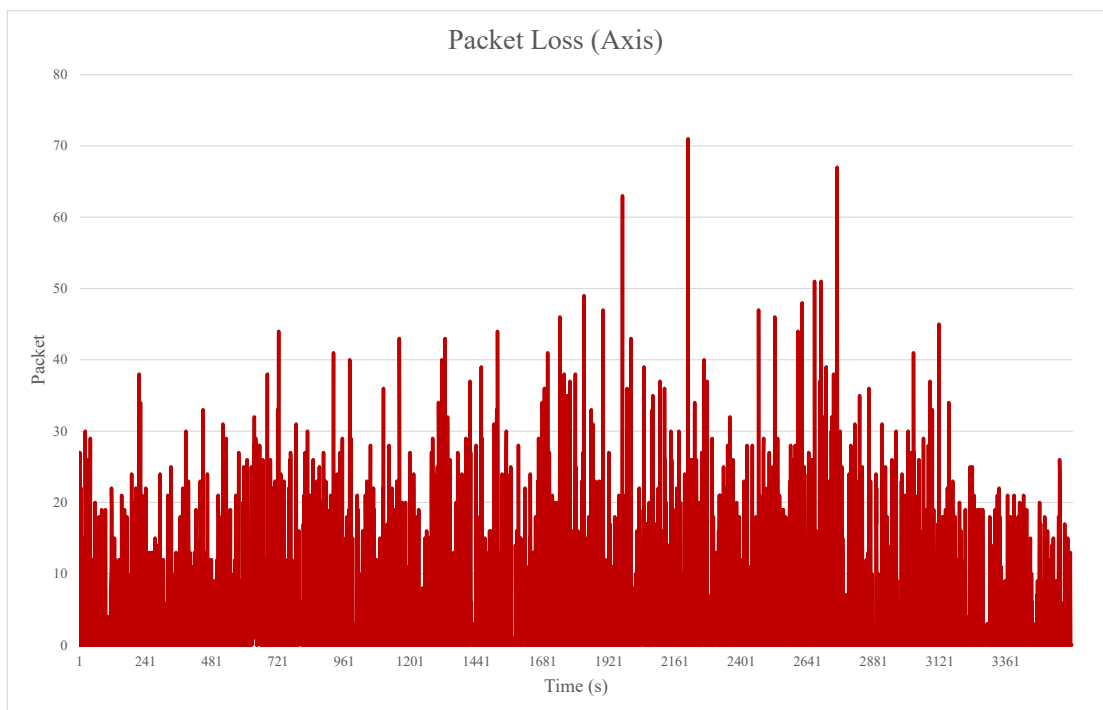
### Packet Loss

Webrtc-internal provides packet loss data in an accumulated form. Hence, the total packet loss of Axis camera is 18700 packets and Hikvision camera is 3696 packets throughout the one hour period. We are then able to calculate the packet drop percentage by using the total received packets and packet loss. The packet drop percentage of Axis is found to be 2.8% and Hikvision is 0.5%. Referring to graph 4.5 and graph 4.6, the maximum packet loss for Axis and Hikvision at an instance is 71 packets and 34 packets respectively.





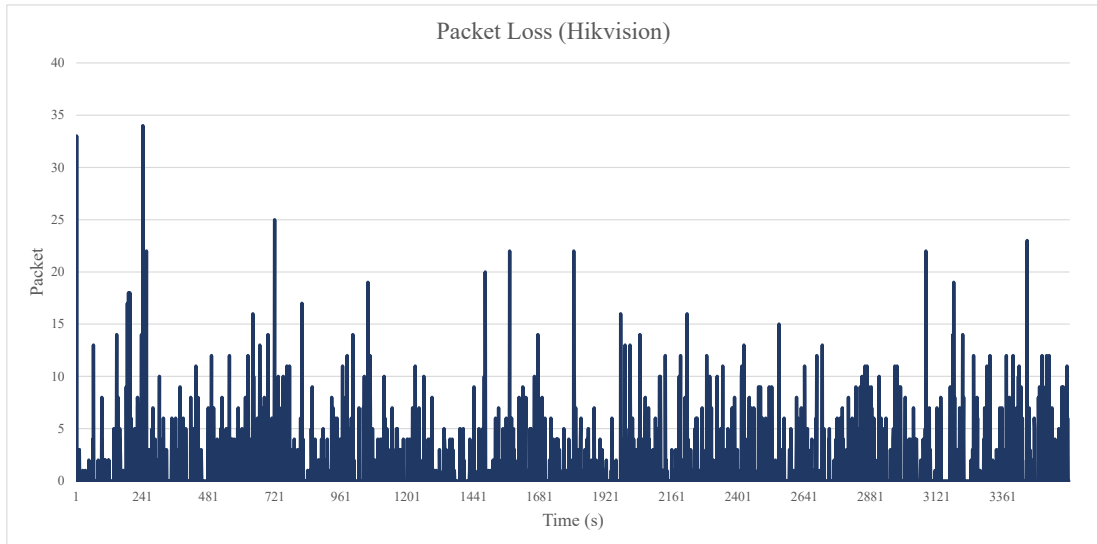
**Figure 4.4:** Graph of Packet rate (Hikvision Camera)



**Figure 4.5:** Graph of Packet loss (Axis Camera)

### Frame rate

The original frame rate is configured to be 25 frames per second (Axis) and 22 frames per second (Hikvision). We have recorded the received frame rate and the output frame rate at the player side. Frame rate details are included in the table below:



**Figure 4.6:** Graph of Packet loss (Hikvision Camera)

**Table 4.1:** WebRTC Frame rate

	Received frame rate (Axis)	Output frame rate (Axis)	Received frame rate (Hikvision)	Output frame rate (Hikvision)
Mean	19.33	19.99	29.46	29.18
Minimum	11	3	19	1
Maximum	26	30	35	46

### Jitter and Delay

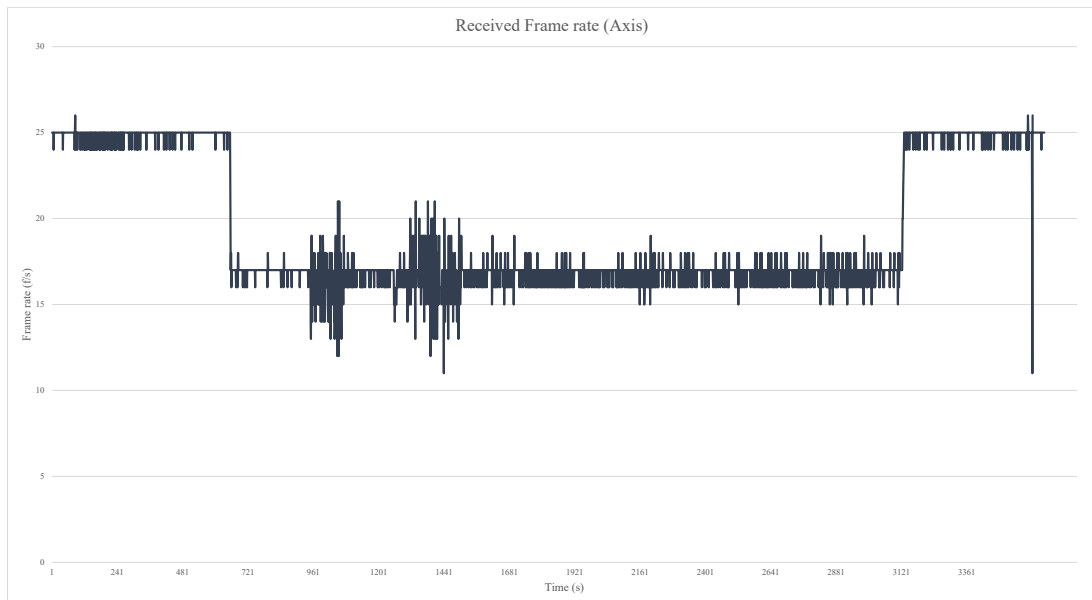
Axis and Hikvision camera have an average jitter value of 34.55ms and 39.173ms respectively. The maximum jitter captured is 231ms for Axis camera and 255ms for Hikvision. It appears that Hikvision holds higher jitter for both mean and maximum values. Similarly, for delay, Hikvision camera is recorded having higher delay, the average time is 72.29ms and maximum 289ms. Where the average delay for Axis is 67.48ms and maximum 266ms.

### Discussion

By cross interpreting all these collected data, there is some form of pattern investigated. Both Axis and Hikvision camera have the maximum bit rate at similar seconds (2422s - 2424s). This may be caused by the placement of cameras, both cameras are placed within a close distance. When an object movement is captured, the bit rate will rise accordingly. Similarly, the received packet rate is also maximum for both cameras between 2422s and 2424s. The quality of the output video for both cameras is consistent.

There is no significant bitrate drop throughout the whole period of time. Furthermore, the average bit rate of Axis (2070Kb/s) and Hikvision (2219Kb/s) are similar compared with their original camera video source (2048Kb/s).

The percentage of packet loss is recorded to be 2.77% (Axis) and 0.5% (Hikvision). The packet lost are negligible since there is no obvious distortion at the output client side. The Hikvision output frame rate at the player side has a slight decreased rate compared to the received one where Axis camera shows a higher output frame rate. There are also fluctuations observed in both Axis and Hikvision output frame rate graphs. These fluctuations are probably due to the decoding mechanism of the browser. Factors like CPU processing power will affect the decoding speed.



**Figure 4.7:** Received Frame rate (Axis Camera)

From graph 4.9 and 4.10, we observed a similar waveform of the jitter and delay collected. The jitter and delay value are peak at the same instance for both cameras. According to [78], it is found that a jitter value exceeding 500ms led to terrible user experience, regardless of the delay value. Meaning that jitter value is more crucial in defining the user experience. Considering the result we have obtained, both cameras have an average jitter value under 50ms. Although the maximum jitter of Hikvision camera hits 255ms, it decreases and stabilizes within 60s.

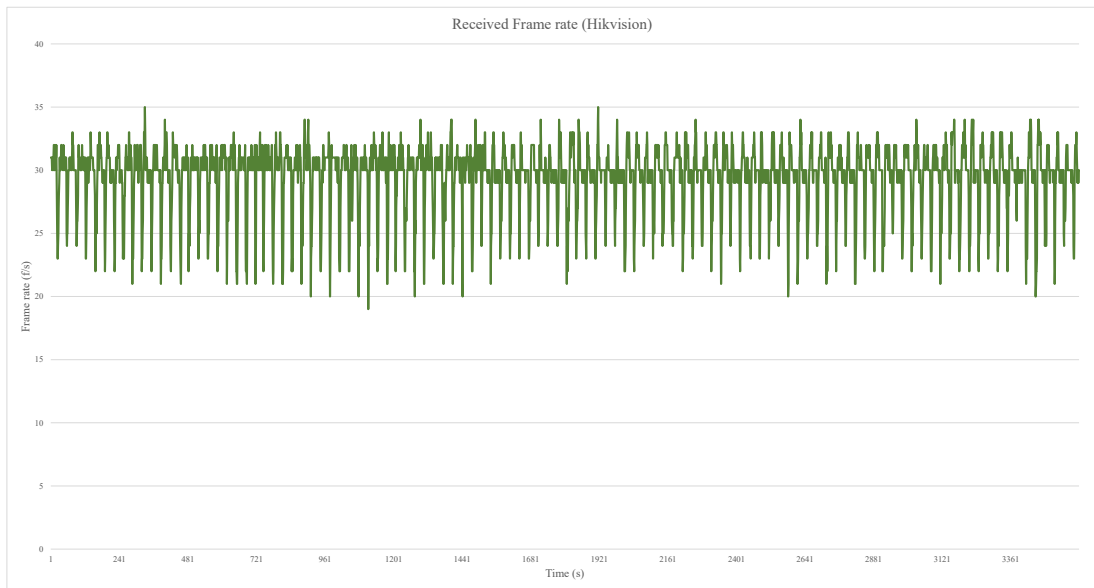


Figure 4.8: Received Frame rate (Hikvision Camera)

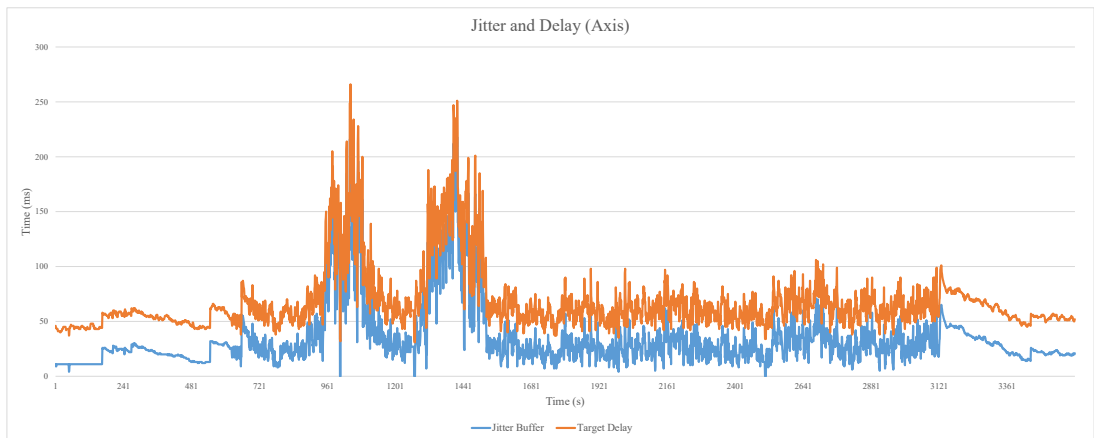


Figure 4.9: Jitter and Delay (Axis Camera)

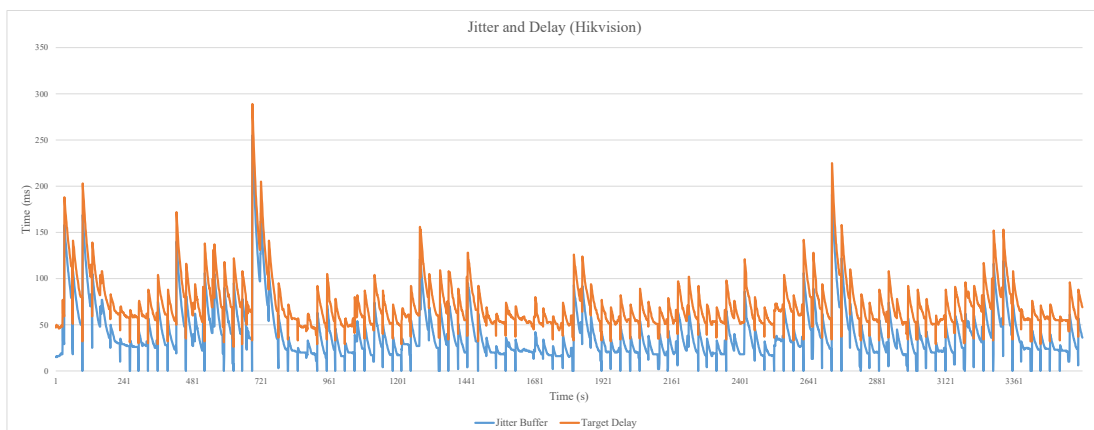
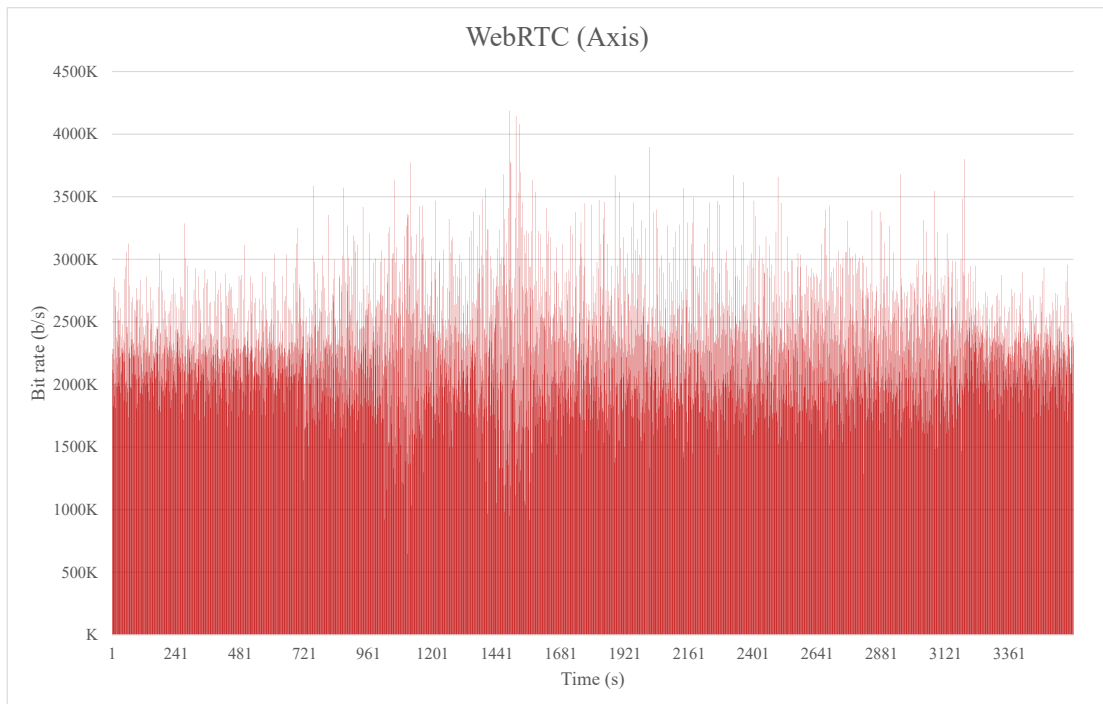


Figure 4.10: Jitter and Delay (Hikvision Camera)

## 4.2 WEBRTC versus FFserver

In this thesis, FFserver is used to compare the performance with WebRTC. We focus on equating their bandwidth, CPU and memory usages. These three factors are the ones that directly affect user experience and the collected data could directly represent streaming quality. Bandwidth is measured using Wireshark where CPU and memory usages are recorded by a program named top. These collected data are presented in graphs as follow:

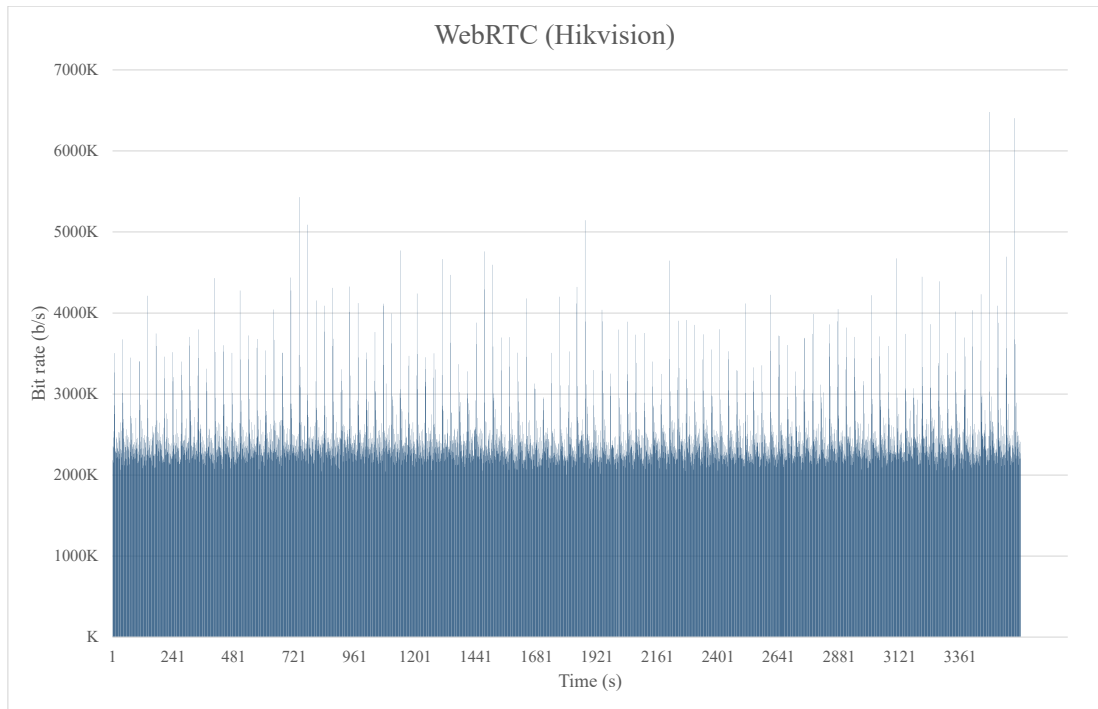


**Figure 4.11:** WebRTC Bitrate measured by Wireshark (Axis)

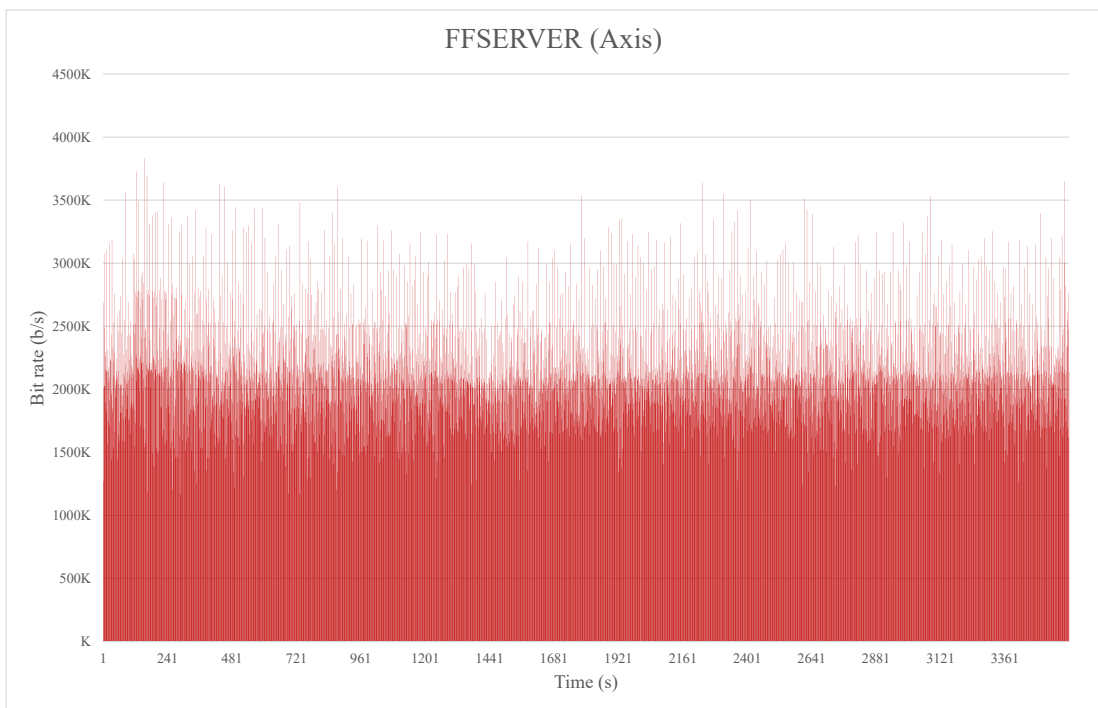
**Table 4.2:** WebRTC versus FFserver: Bit rate

	Axis (WebRTC)	Hikvision (WebRTC)	Axis (FFserver)	Hikvision (FFserver)
Mean	2294Kb/s	2455Kb/s	2142Kb/s	2167Kb/s
Minimum	646Kb/s	2003Kb/s	1165Kb/s	36Kb/s
Maximum	4187Kb/s	6480Kb/s	3830Kb/s	4236Kb/s

The mean, minimum and maximum bandwidth of both WebRTC and FFserver are arranged into a table as shown (Table 4.2). Generally, WebRTC has a higher bit rate for both Axis and Hikvision camera. This information shows that although it is not very obvious, WebRTC streaming does provide higher quality streaming compared to FFserver. However, this factor is not the conclusive reason for choosing WebRTC as our streaming server. The most crucial aspects are the CPU and memory usage, which will

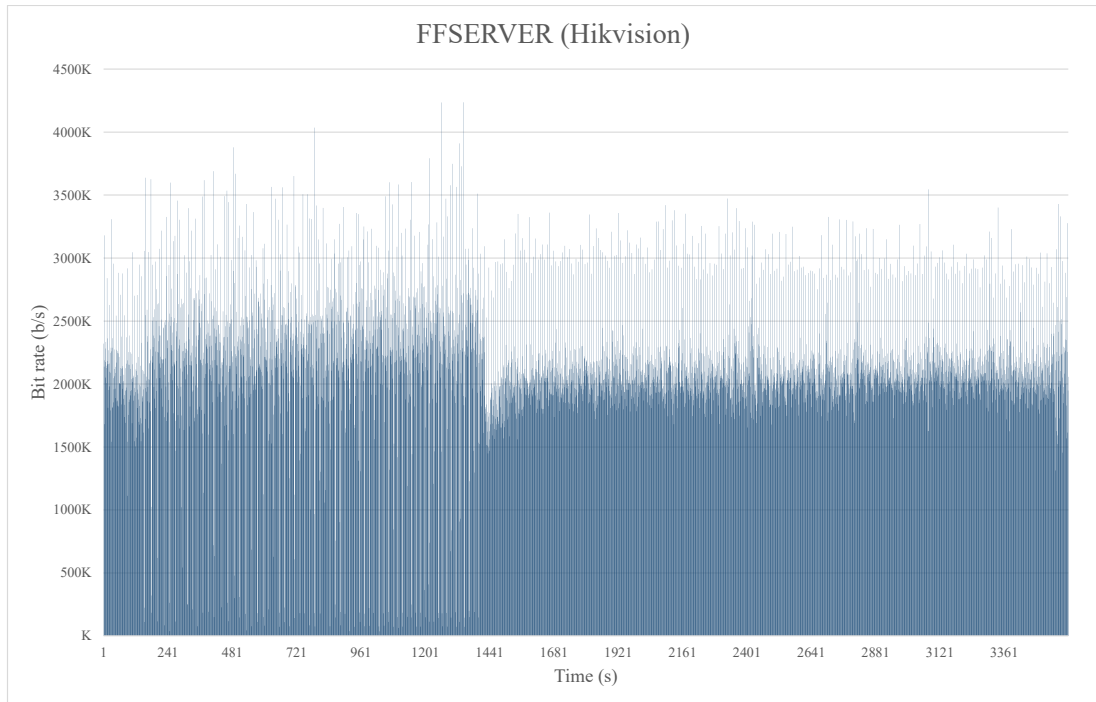


**Figure 4.12:** WebRTC Bitrate measured by Wireshark (Hikvision)



**Figure 4.13:** FFserver Bitrate measured by Wireshark (Axis)

be discussed in the next section.

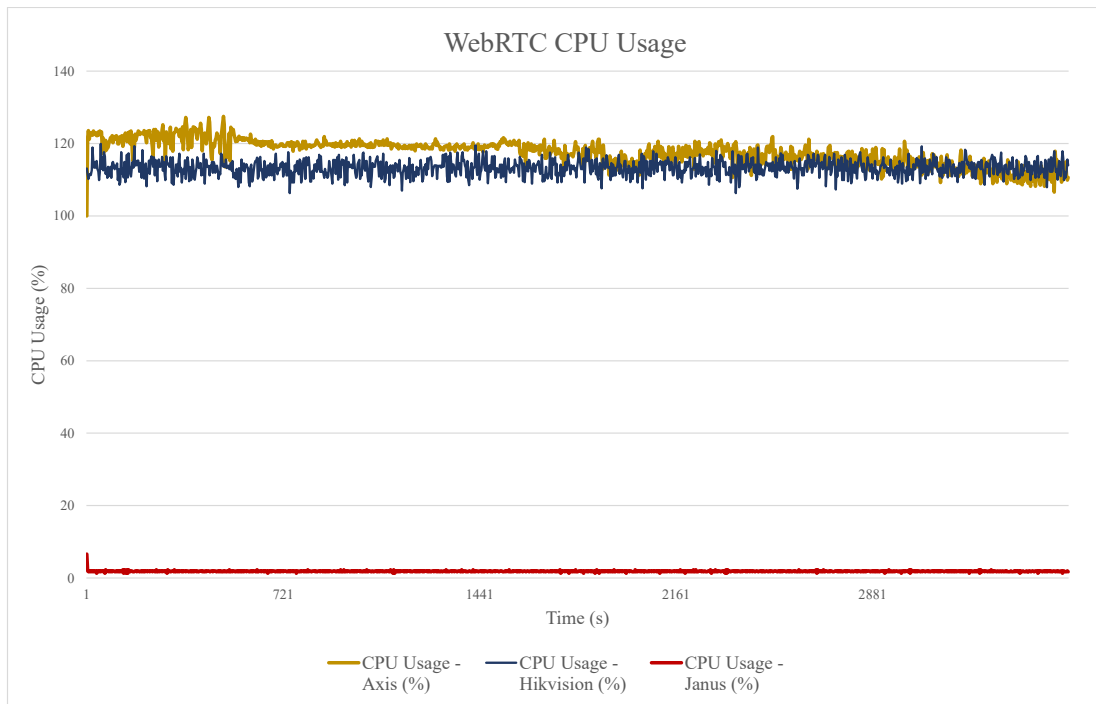


**Figure 4.14:** FFserver Bitrate measured by Wireshark (Hikvision)

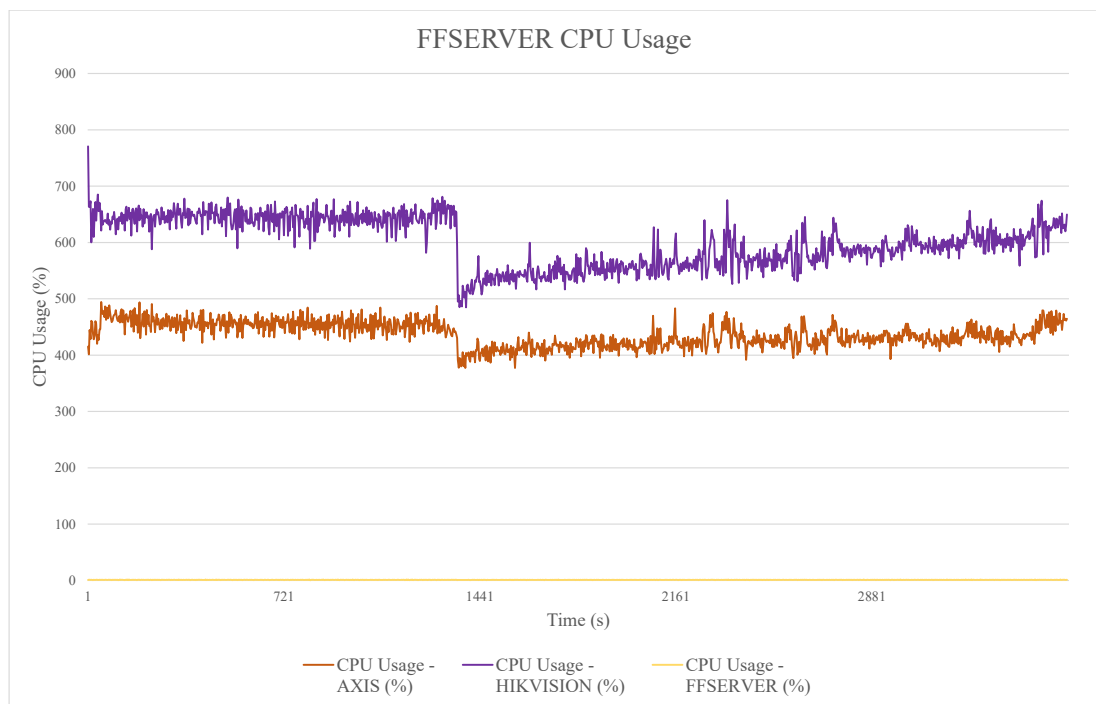
#### 4.2.1 CPU and Memory usage

The CPU and memory usage are important factors in comparing the performance of the two listed media servers. According to the specifications of our server, it consists of 12 Intel Xeon CPU cores each at rate of 2.60GHz and the total memory is approximately 2.5GB. The CPU usage was measured based on the percentage used on each core. Similarly, the memory usage was also measured by percentage. Since we are using two IP cameras, there are three tasks in order to complete the whole WebRTC streaming process. Hence, the CPU and memory usage of all three processes have to be traced, the same measurement applied to FFserver as well. The running processes of WebRTC are two transcoding tasks for each camera and one for Janus server. The results of the measured CPU usage of WebRTC and FFserver are recorded in the tables below. The first table shows the average CPU usage of each task and the following table records the total CPU usage of both streaming servers.

The complete one hour CPU usage record are shown in graph 4.15 and 4.16. One value was recorded in every 3 seconds. Hence, the graph records in total 1200 samples. From our obtained result, two of the FFserver transcoding processes (Axis and Hikvision) consumed noticeable high CPU usage as compared to WebRTC. The transcoding process of FFserver Axis camera has an average CPU usage of 437.23% where WebRTC



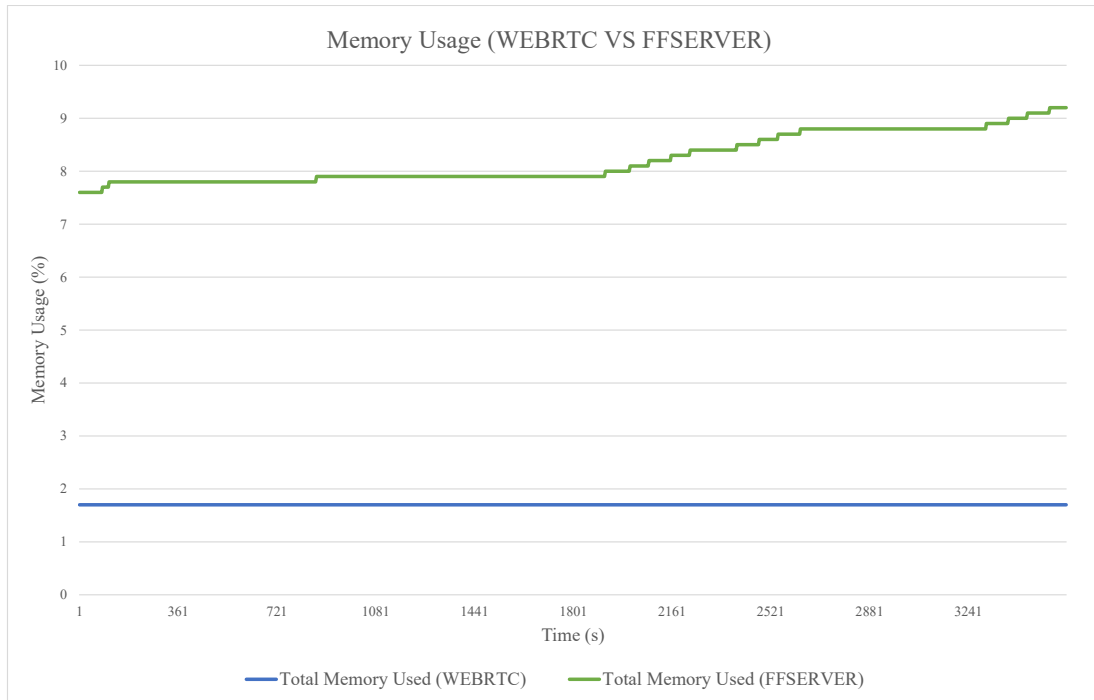
**Figure 4.15:** WebRTC CPU Usage



**Figure 4.16:** FFserver CPU Usage

is 117.59%. The difference between the two of them is up to 319.64% which the CPU consumption of FFserver is 3 times WebRTC. Whereas, Hikvision shows result that is even more drastic. FFserver Hikvision stream consumed 601.18% and WebRTC used 113.25%.





**Figure 4.17:** Total Memory Usage

**Table 4.3:** CPU Usage

Average CPU Usage (%)			Total CPU Usage (%)		
Task	WebRTC	FFserver		WebRTC	FFserver
Axis	117.59	437.23	Mean	232.70	1039.50
Hikvision	113.25	601.18	Minimum	219.70	870.70
Janus	1.86	N/A	Maximum	244.60	1186.80
FFserver	N/A	1.11			

Similarly, the total CPU usage for both server has also competed. For 2 cameras, FF-server used up nearly 11 cores. Where WebRTC used approximately 5 of them. As for the memory usage, FFserver averagely consumed 8.22% and 1.70% for WebRTC. Graph 2.6 shows the memory usage of FFserver increases gradually throughout the whole period of experiment. On the other hand, WebRTC shows a rather stable memory usage without any obvious increase. From the above observations, it is proven that the FFserver processes are bulkier as compared to WebRTC. From 4.16, there is a noticeable drop at approximately 1400s. This may be caused by a sudden image quality drop through RTSP communication. CPU processing usage increased gradually after the drop.

## 4.3 Result from Mobile Client

### 4.3.1 Android

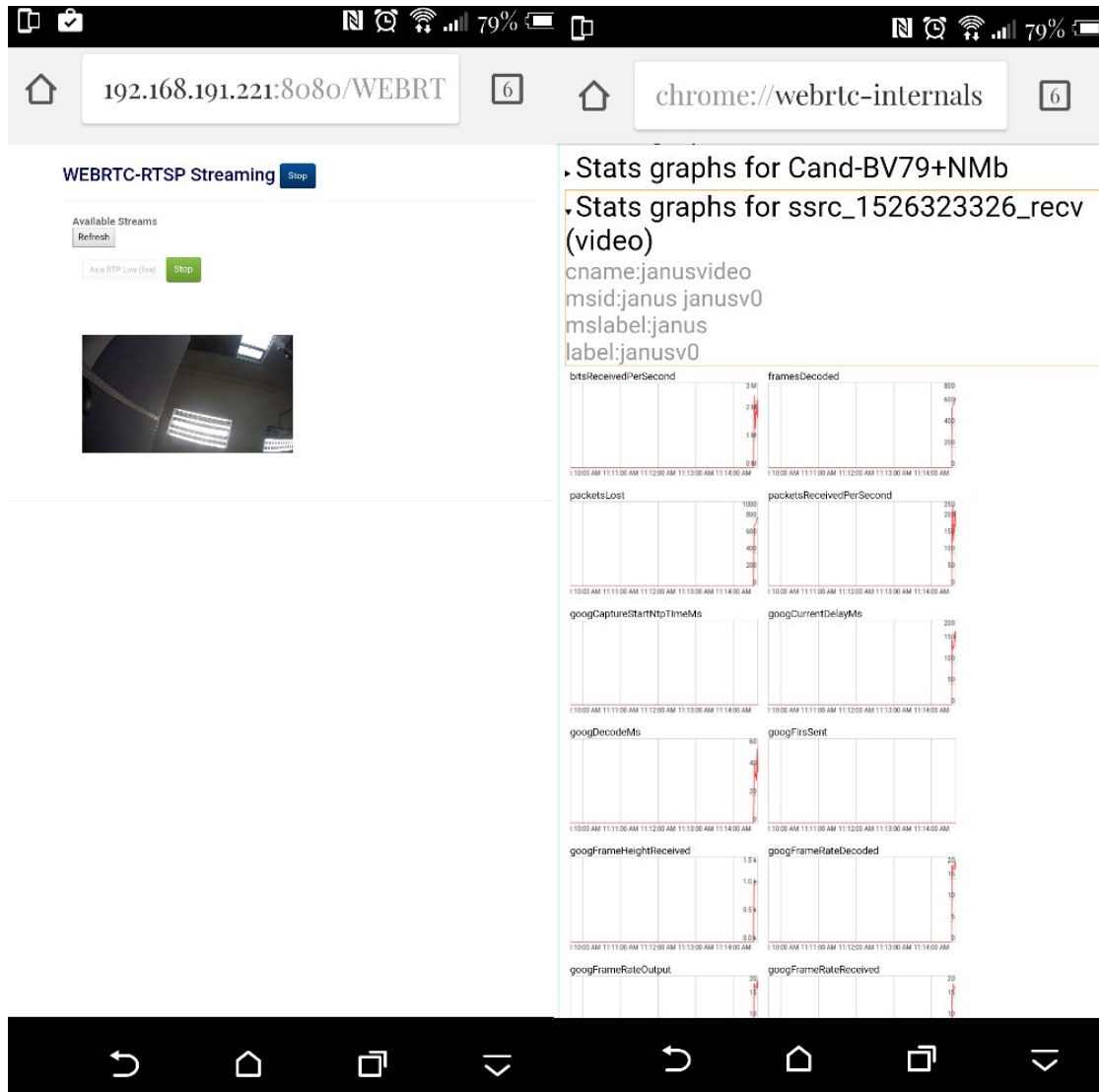


Figure 4.18: Snapshot from Android device

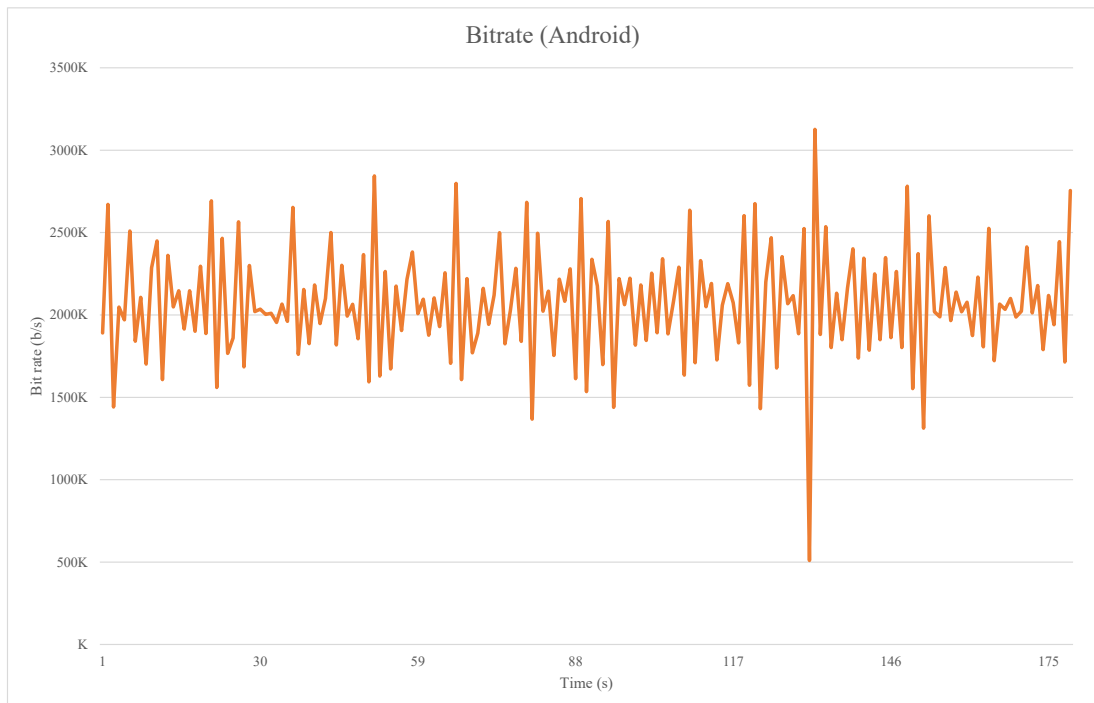
Figure 4.18 shows the screenshot of WebRTC streaming on Android platform. The streaming lasted a period of three minutes and a dump file is created and interpreted. Throughout the whole experiment, only Axis camera is considered.

The recorded data are arranged in Table 4.4. By comparing the results collected from laptop and Android, it is observed that bit rate, received and output packet rate are identical. However, it is found that the jitter, delay, and packet loss showed a noticeable difference. The average jitter value obtained from the laptop client is 34.55ms where Android client measured 53.32ms. Similarly, laptop client obtained an average 67.48ms

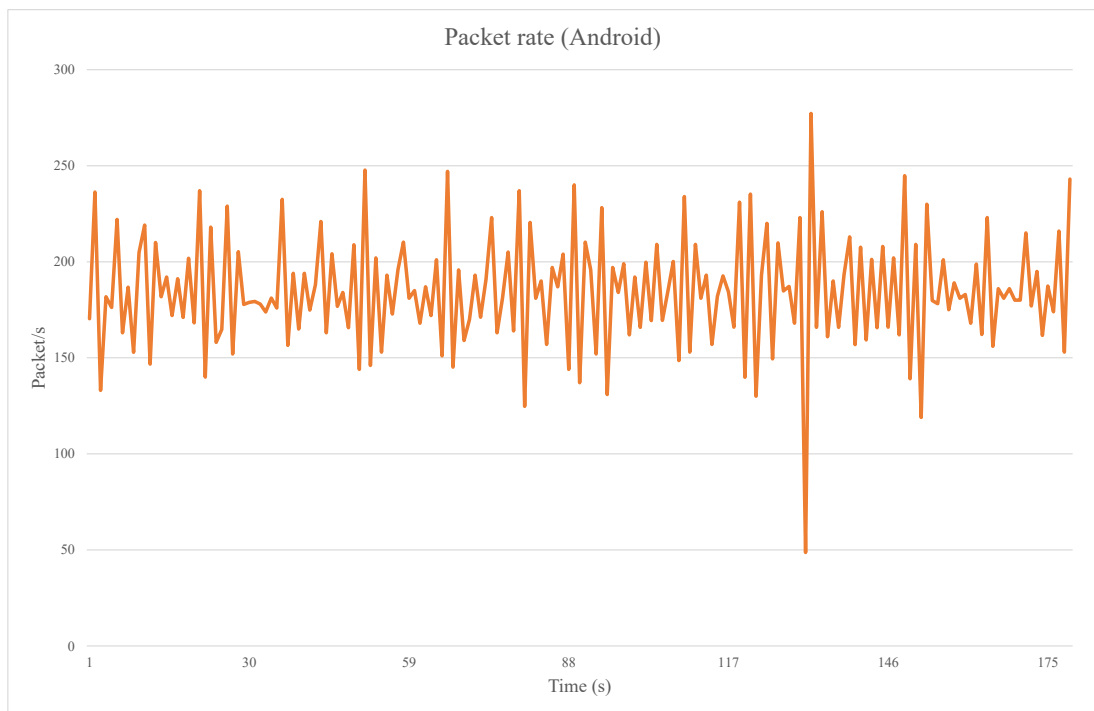
**Table 4.4:** Stream Quality Measurement (Android)

	Mean	Minimum	Maximum
Bitrate (Kb/s)	2073	510	3126
Received Packet rate (packets/s)	184.68	48.66	277.22
Received Frame rate (f/s)	16.78	12	21
Output Frame rate (f/s)	17.72	13	35
Jitter (ms)	53.32	0	153
Delay (ms)	129.96	68	225
	Mean	Sum	Percentage
Packet Loss	19 packets/s	3400 packets	10.35%

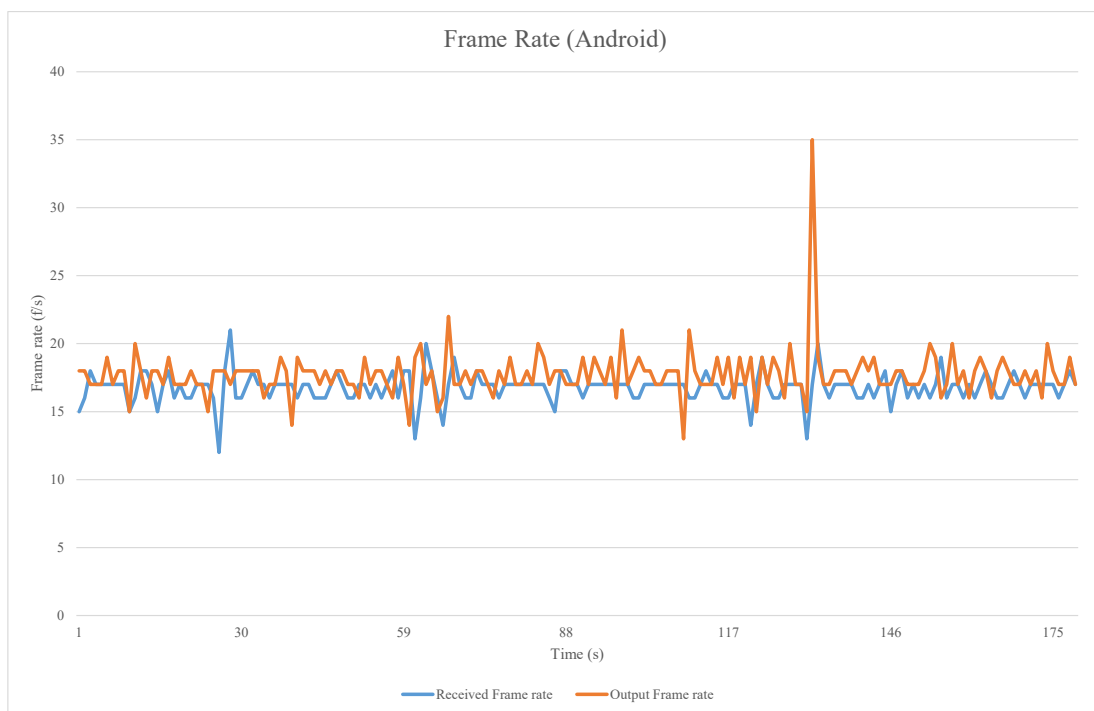
in delay where Android is 129.96ms. The packet loss for laptop client is 2.8% and Android is 10.38%. From the obtained jitter, delay and packet loss value, it is observable that by using Android client in receiving WebRTC stream, the stability is relatively low as compared to the laptop Chrome browser. One of the reasons that causes the inconsistency in receiving the video packets may be due to instability WiFi connection. Since the Android is connected to the LAN network using WiFi, it is reasonable that the connection is not as consistent and the wired ones.



**Figure 4.19:** Bit Rate(Android)



**Figure 4.20: Packet Rate(Android)**



**Figure 4.21: Frame Rate(Android)**

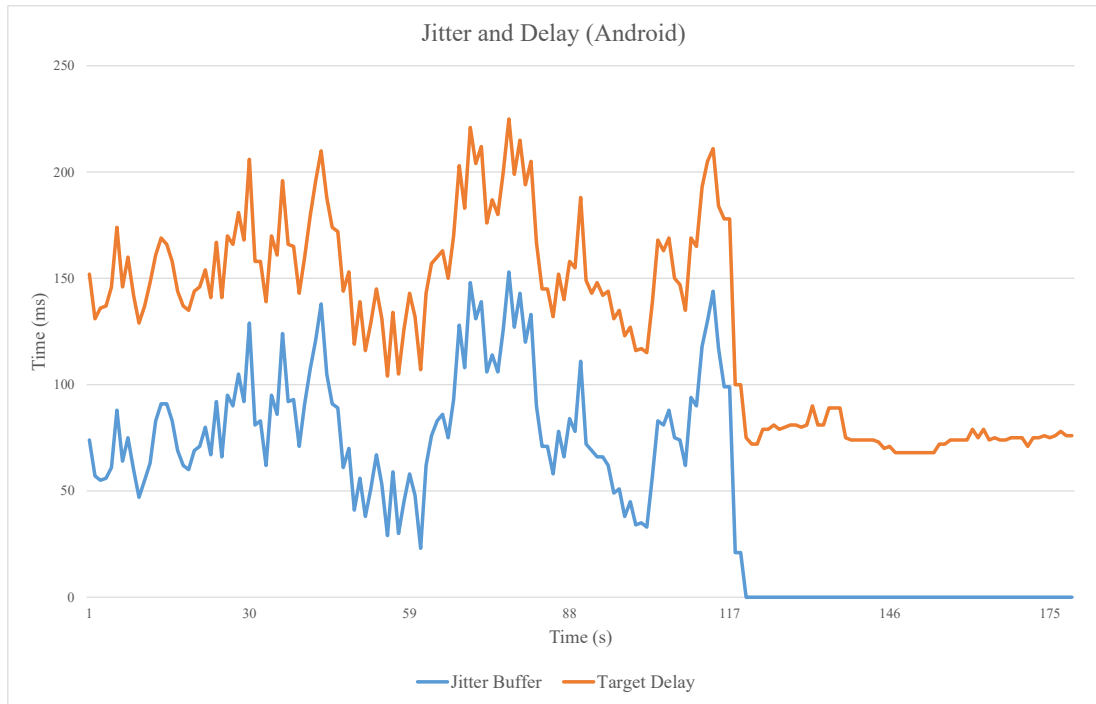


Figure 4.22: Jitter and Delay (Android)

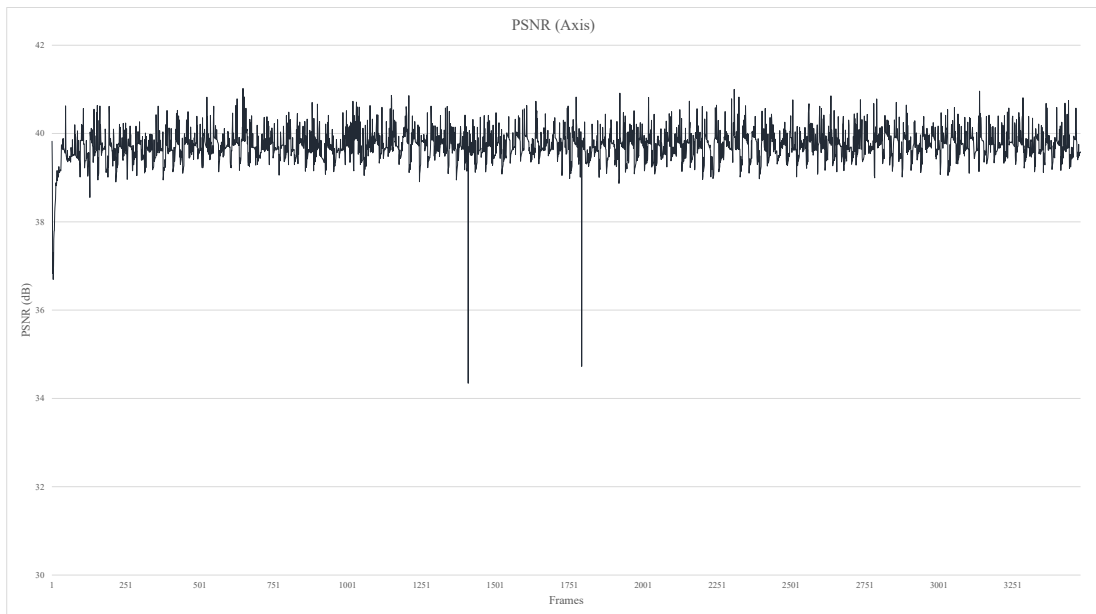
## 4.4 Video Quality Assessment

Video Quality Assessment is used to predict image quality of a video automatically. VQA is performed only on WebRTC output and since FFserver is using another totally different codec (H.264) and streaming format (FLV), VQA is not performed on FFserver. In this experiment, an original video with least distortion is recorded directly from the IP camera. It is used as a reference to compare with the transcoded video for streaming. The videos are recorded for a period of 5 minutes and they have approximately more than 3400 frames. The VQM methods that we used are PSNR, SSIM and VQM.

### 4.4.1 PSNR

Figure 4.23 displays the PSNR value obtained from 3480 frames in graph form. The average value is 39.70dB, where the least quality frame is on the 1409 frame with PSNR 34.34dB. A screenshot of the original and processed frame is shown in Figure 4.24.

Table 4.5 is a mapping of the most traditional subjective metric, Mean Opinion Score (MOS) [9]. The quality level of MOS model is rated based on a scale of 1 to 5 as observed in Table 4.5. Where 5 is the best possible score. Hence, based on the mapping table the average PSNR value we obtained (39.70dB) in our experiment is considered as having



**Figure 4.23:** PSNR (Axis)



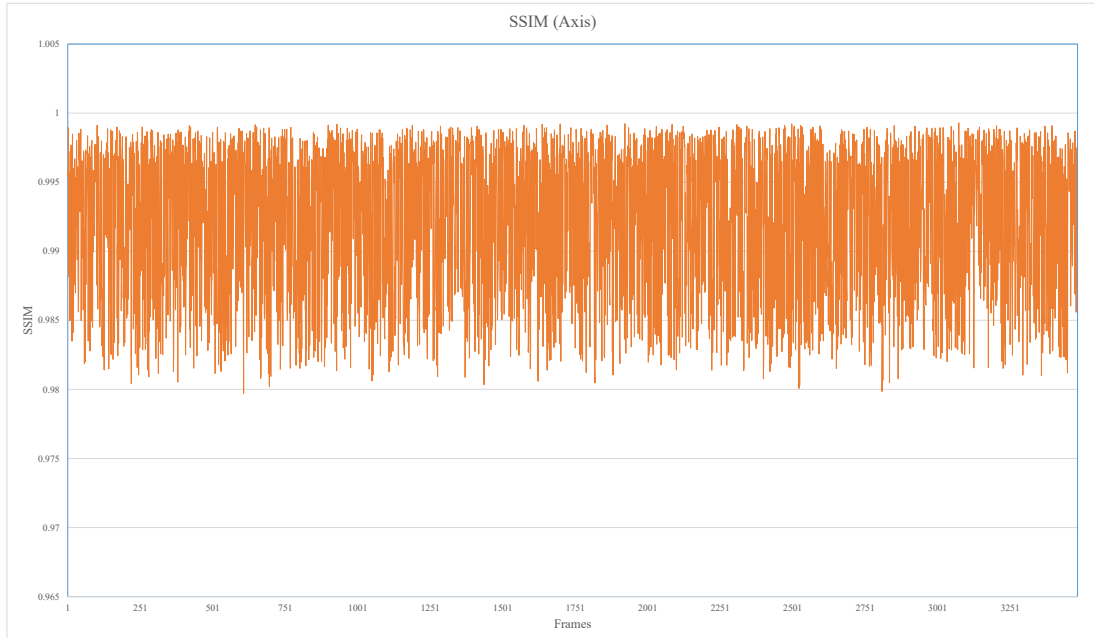
**Figure 4.24:** Left: Original Frame versus Right: Streamed Frame

**Table 4.5:** Mapping PSNR to MOS [9]

PSNR (dB)	MOS
>37	5 (Excellent)
31-37	4 (Good)
25-31	3 (Fair)
20-25	2 (Poor)
<20	1 (Bad)

excellent image quality. In a nutshell, the overall PSNR result proves that the streaming QoE for Axis camera is above standard.

#### 4.4.2 SSIM



**Figure 4.25: SSIM (Axis)**

SSIM is another method in measuring the similarity of two images. It is designed basically to improve older methods like PSNR and MSE. Since both PSNR and MSE are proven to be inconsistent with human eye perception, SSIM has a higher accuracy. The SSIM result is arranged in Figure 4.25. A mean SSIM value of 0.992 is obtained from our measurement. The highest rating for SSIM = 1. Hence our average result  $0.992 \approx 1$ , showing that the distortion is hardly recognizable by human eyes. The lowest SSIM value obtained is 0.979 on frame 607, which the lowest value has only a slight difference compared to the average one.

#### 4.4.3 VQM

VQM measures the perceptual effects such as blurring, noise, block distortion, colour distortion, and jerky motion. Figure 4.26 shows the VQM graph of Axis camera. The average value calculated is 0.253 and the lowest point is 0.1 on frame 1189. VQM value is read to be the best at 0, means no visible difference to the original. Hence, the smaller the VQM value the better.

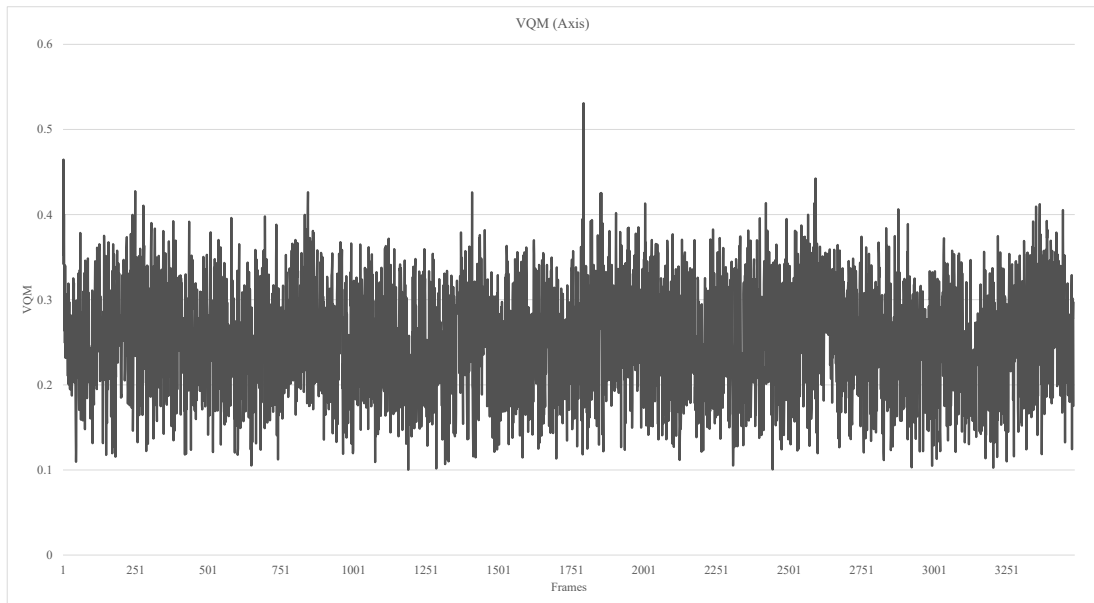


Figure 4.26: VQM (Axis)

After obtaining the VQM results, we compared it to a study conducted by the Telecommunication Networks Group of Technische Universitat Berlin [8]. They carried out an experiment in comparing the video quality of H.264 and VP8. They are using video with 200kb/s bit rate which is similar to our case. As you can see in Figure 4.27, the highest VQM value that they obtained is approximately 0.35, where in our case, it is 0.253.

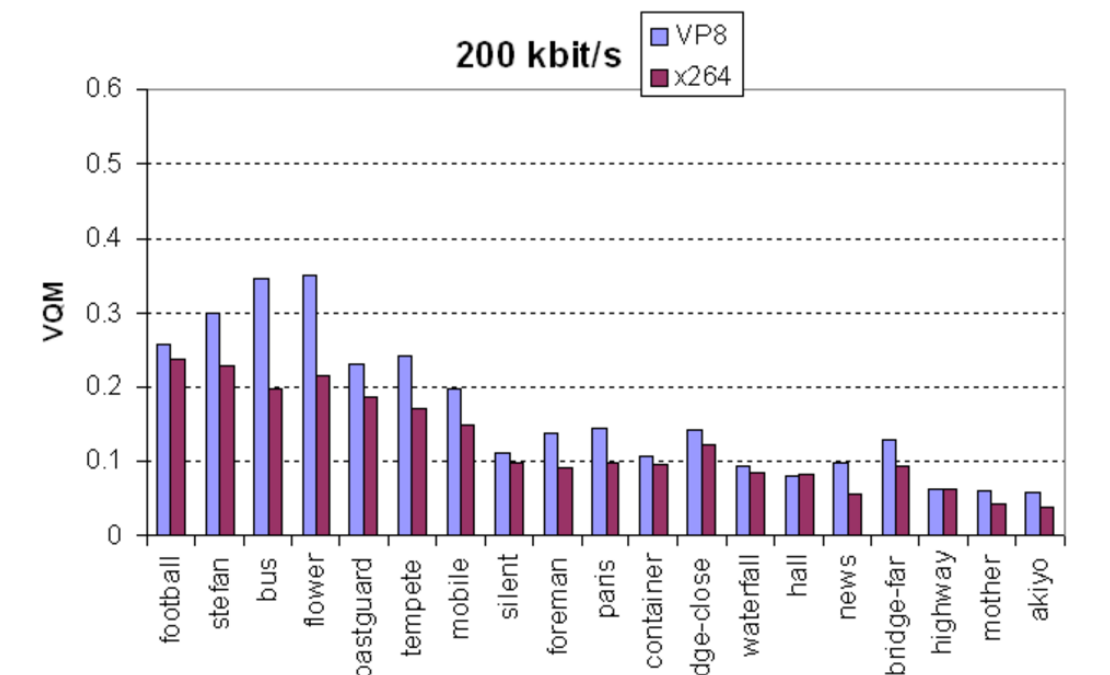


Figure 4.27: VQM of H.264 versus VP8 [8]



## 4.5 Streaming from Standard Video File

This section includes the results of WebRTC streaming from standard files. The video file used in this experiment is Big Buck Bunny which is available in the Big Buck Bunny download site [79]. Videos are prepared in three specifications as stated in Table 4.6. The 480p has the lowest quality with resolution 852x480 and approximately 500kb/s bitrate. Where 720p is the medium, 1080x720 and 1Mb/s, and 1080p the highest, 1920x1080 2Mb/s. The overall video has a duration of 09:56 minutes. Similar to the previous section, the results will be focusing on the streamed output's bitrate, packet rate, frame rate, packet loss, , and delay.

**Table 4.6:** Big Buck Bunny Video files Specifications

	Video Format	Resolution	Bitrate	Frame Rate	File Size
480p	H.264, YUV420p	852x480	499kb/s	24fps	37.3MB
720p	H.264, YUV420p	1080x720	996kb/s	24fps	74.3MB
1080p	H.264, YUV420p	1920x1080	1993kb/s	24fps	148.6MB

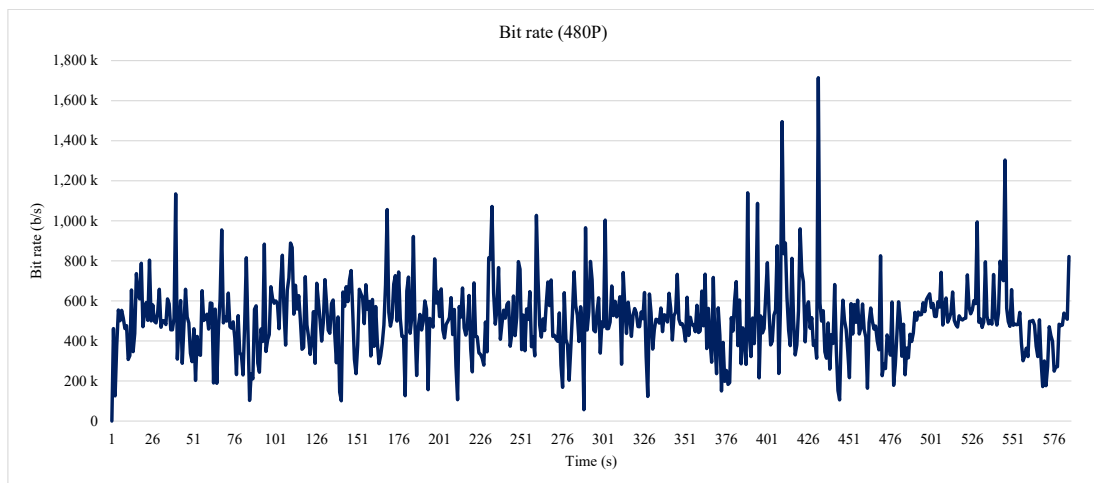
### Results from PC Client

The bit rate, jitter and target delay of each video type are presented in graphs as shown in Figure 4.28 to Figure 4.33. Where the details of packet and frame rates are arranged in table as in Table 4.7. The average bit rate of 480P video is 505kb/s, 720P is 1008kb/s and 2038kb/s. These results are comparatively near to the original bit rate as stated in Table 4.6. From our observation, the packet loss for the entire duration of 09:56 minutes, the 480P and 720P video have 0 packet loss. However, the 1080P showed 288 packets lost. This is due to the larger size and packet rate transmission and also limited bandwidth. The maximum packet amount received per second for the 1080P streaming reached up to 586 packets per second which are approximately 1.62 times the maximum value of the 720P stream.

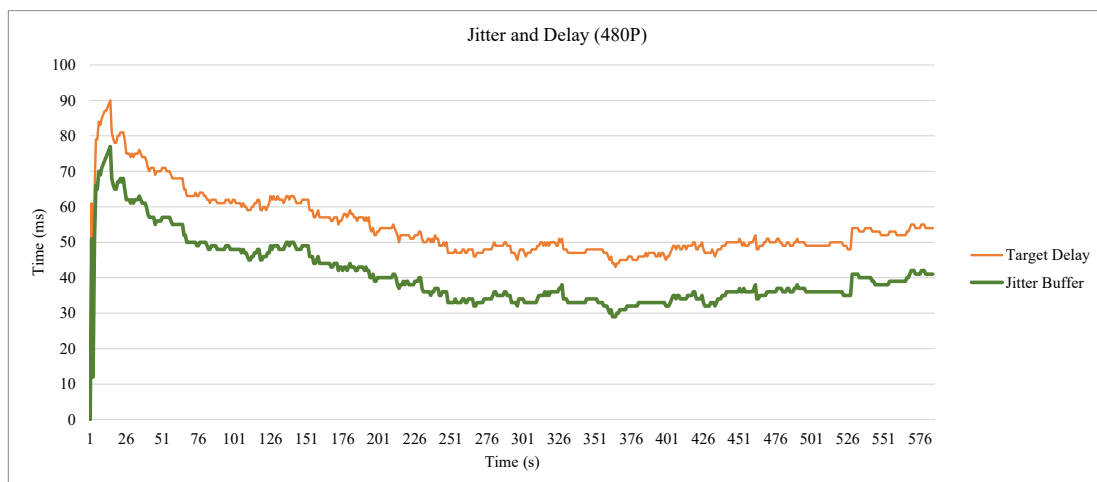
The frame rate received for all three video types are very close to the original frame rate which is 24 frames per seconds. As for jitter and target delay, we are concerned with their maximum value. The maximum jitter value for 480P is 77ms, 720P is 41ms and 1080P is 158ms. Those values obtained are nowhere near 500ms showing a relatively stable and satisfying user experience. Referring to Figure 4.29, 4.31 and 4.33, we can observed the graph pattern of jitter and delay is similar but with different values. The value for delays is slightly higher compared to jitter.

**Table 4.7:** Streaming Result from PC Client

	480P	720P	1080P
Bitrate(b/s)			
Mean	505k	1,008k	2,038k
Min	56k	66k	212k
Max	1,715k	4,059k	6,750k
Packet Rate(p/s)			
Mean	55	97	185
Min	24	20	25
Max	157	360	586
Frame Rate(f/s)			
Mean	23.90	23.89	23.80
Min	3	13	10
Max	38	27	30



**Figure 4.28:** Bitrate (Big Buck Bunny 480P)



**Figure 4.29:** Jitter and Target Delay (Big Buck Bunny 480P)

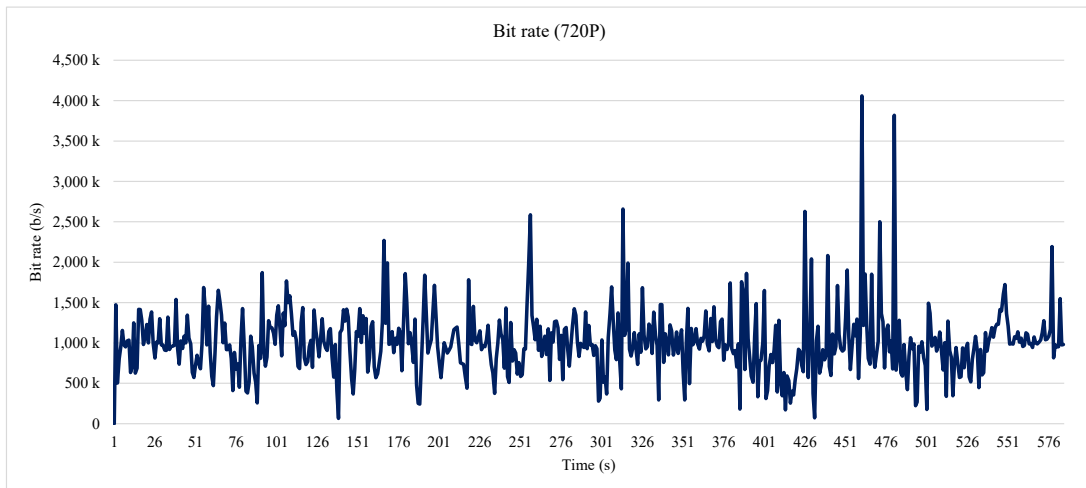


Figure 4.30: Bitrate (Big Buck Bunny 720P)

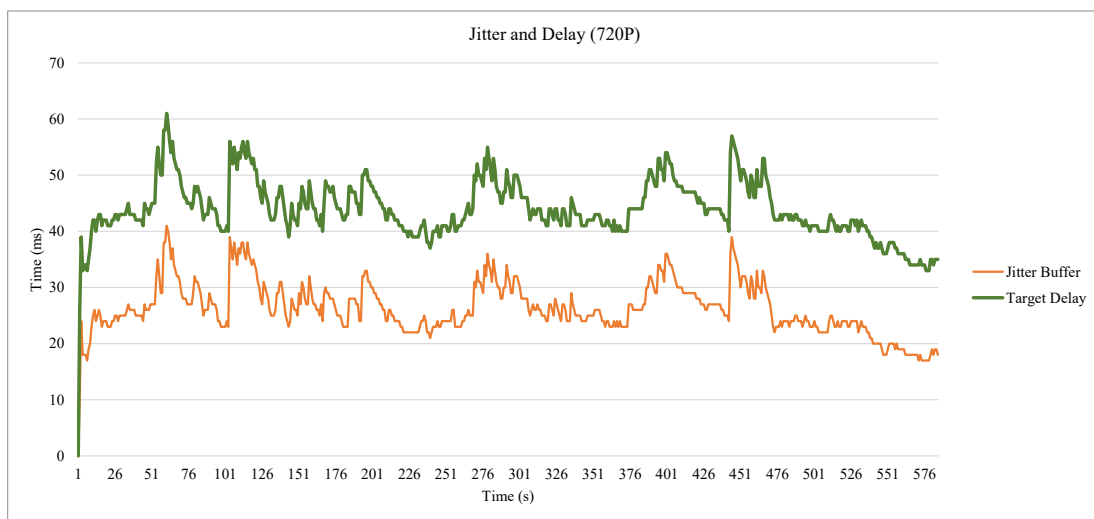


Figure 4.31: Jitter and Target Delay (Big Buck Bunny 720P)

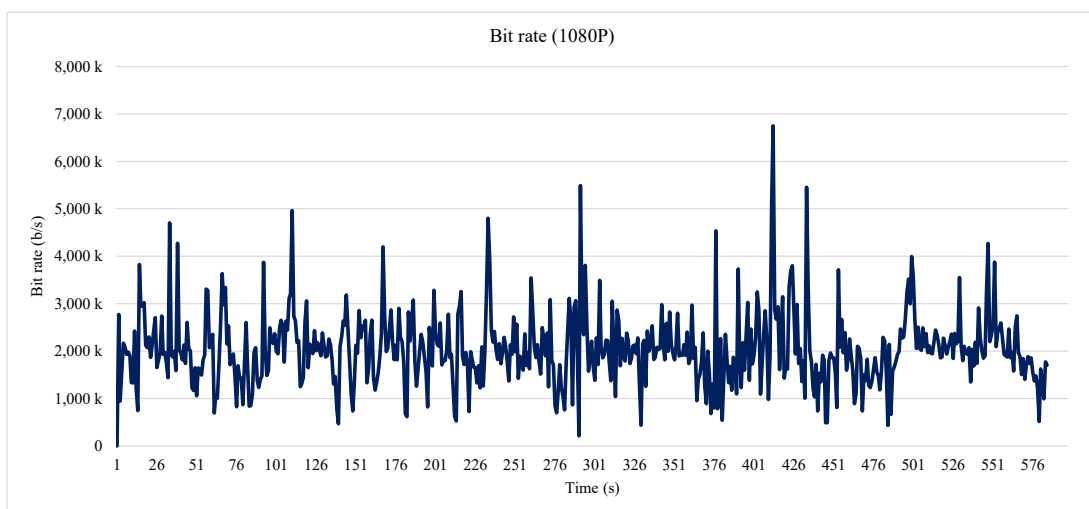
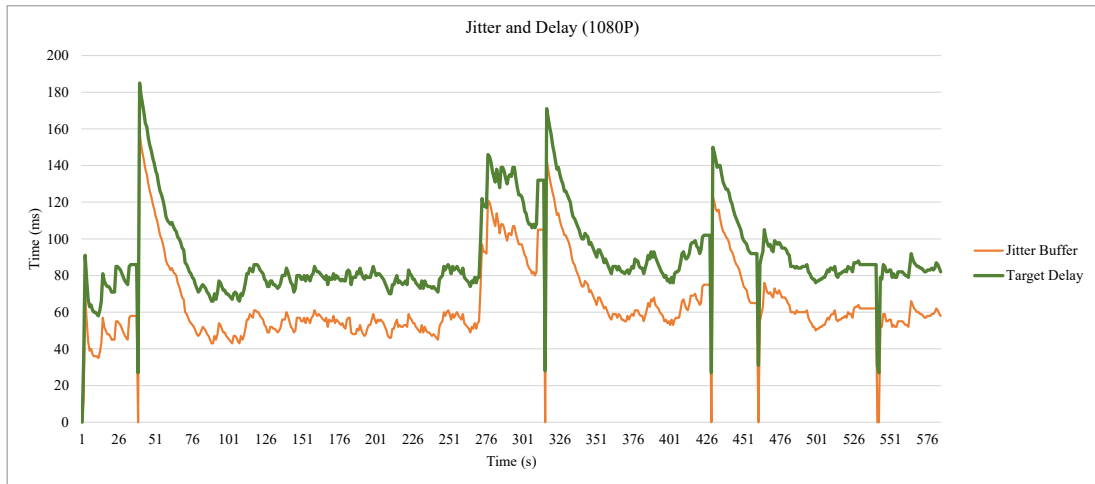


Figure 4.32: Bitrate (Big Buck Bunny 1080P)



**Figure 4.33:** Jitter and Target Delay (Big Buck Bunny 1080P)

### Results from Android Client

This section discussed the results obtained from streaming three of the Big Buck Bunny video to mobile (Android) client. The video files used are the same as the previous section as stated in 4.6. However, the streaming duration for this part is set to 3 minutes which is similar to the experiment done for the Android real time streaming section.

**Table 4.8:** Streaming Result from Android Client

	480P	720P	1080P
Bitrate(b/s)			
Mean	505k	1,016k	1,987k
Min	0	255k	376k
Max	1,092k	2,081k	4,728k
Packet Rate(p/s)			
Mean	55	98	180
Min	0	29	40
Max	105	192	413
Frame Rate(f/s)			
Mean	18	22.20	22.32
Min	0	10	11
Max	27	26	33

Overall, streaming from mobile client shows a slight decrease in several measurements, which is expected due to lower bandwidth (WIFI connection). The most obvious decreased is the received frame rate. The frame rate measured for 480P is 18fps, 720P 22fps, and 1080P 22fps. Where the frame rate of the original video is 24fps. Another aspect that is observed to be worst than PC client is the packet loss. Although the packet loss measured is not severe enough that causes interruption during streaming,

it is still worth discuss and interpret. The packet loss measured throughout the period of 3 minutes is 26 packets for 480P, 84 packets for 720P and 110 packets for 1080P. By comparing these values with the packet loss in PC client, mobile client has a much higher inconsistency and lower quality.

Based on our obtained results, we compared the delay of mobile and PC client. The maximum jitter for Android 480P is 421% higher, Android 720P is 371% higher and 1080P is 142% higher. Since the delay did not exceed 500ms, the QoE of Android client streaming is still satisfying. One noticeable aspect is that the 480P video streaming has lower performance than others. The minimum bit rate and packet rate dropped to 0 in the course of 480P streaming. Moreover, the jitter and delay are also much higher compared to the others. This may due to an unstable connection at that point in time.

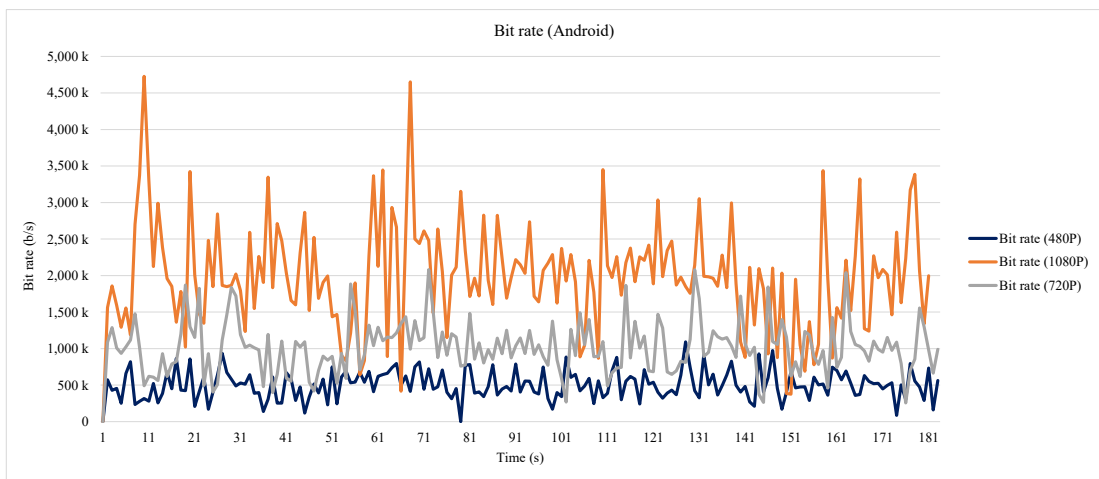


Figure 4.34: Bitrate (Big Buck Bunny)

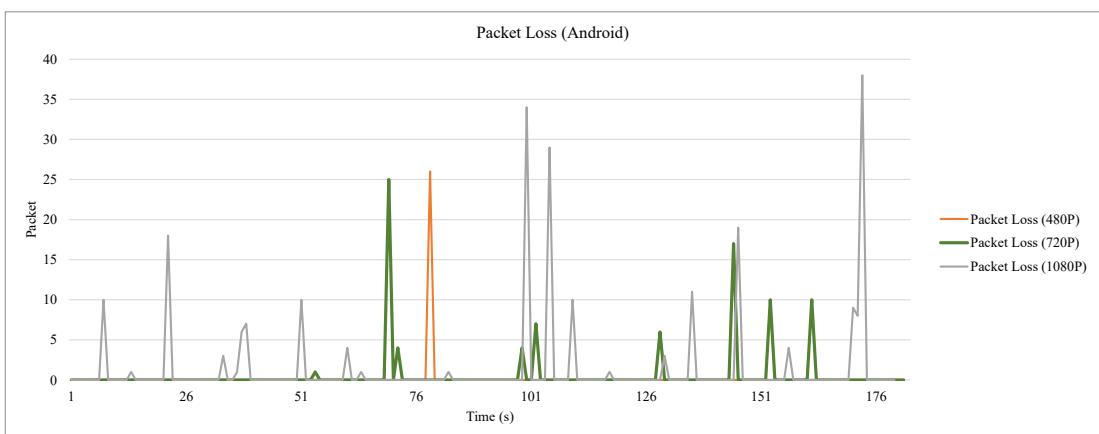


Figure 4.35: Packet Loss (Big Buck Bunny)



Figure 4.36: Frame Rate (Big Buck Bunny)

# Conclusions and Future Works

## 5.1 Conclusions

This thesis discussed and developed a real-time streaming approach for physical video surveillance system. The streaming technology embedded in our network video recorder is WebRTC. In this thesis, the focuses are mainly on developing the communication gateway in between RTSP (IP camera) and WebRTC (streaming server). Hence, Janus WebRTC gateway is used. Janus together with the WebRTC API is modified and integrated into my network video recorder system, streaming camera feeds in a real-time manner.

The first aspect of this research emphasizes on the framework, architecture, and implementation of a complete network video recorder system. The system includes basic functionalities such as recording, displaying, and managing cameras. Regarding live streaming, several technologies like FFserver and WebRTC are compared and tested. As a result, WebRTC is chosen due to the fact that it is an open-source API and most importantly, it is platform and device independent. Any browser supporting WebRTC is able to create live video to another WebRTC device or WebRTC enabled media server.

In order to measure the performance of the developed NVR system, testing environment is created and experiments are conducted. The topology of the experiment set up includes a WebRTC media server and a client as the peer. The real-time streaming performance of the two-party was measured and recorded. I focused on parameters like bitrate, packet loss, frame rate, jitter and delay since these factor impact directly to the quality of the stream. The input and output video data appeared to be relatively similar specifically in terms of bit rate and frame rate.

Additionally, experiments are also carried out to compare the performance of another streaming server, in my case, FFserver and WebRTC. Both streaming servers used the same input feed and also configured to produce similar output bit rate and frame rate. The result showed that FFserver's processes consumed a lot more CPU usage and memory. This is an important factor for us in considering the use of the more light weighted WebRTC.

On the other hand, video streams from WebRTC server to other devices other than computer is also tested. The device used is Android phone. The android phone is connected to the local WiFi access point in order to access my media server. The overall performance of WebRTC streaming on Android platform is comparatively lower than the PC client. Higher jitter, delay, and packet loss are observed. Packet loss was measure approximately 10.35% which is almost five times the packet loss found on PC client. This indicates that WiFi connection do has obvious impacts at the receiving end, causing noticeable packet drop and delay. Further QoE experiments are carried out based on PSNR, SSIM and VQM methods. The final part of the result and discussion chapter includes the experiment which the streaming source is changed from IP camera to standard H.264 video file. Three identical video file with different specifications are used. Results are collected from both the PC and Android client. They are presented in graphs and tables.

### **5.2 Hardware Acceleration Support**

The experiments conducted in this thesis only involves 2 IP cameras due to hardware restriction. Therefore, the suggested future improvement based on my thesis involves utilizing extra hardware support.

The proposed future work for this project is to replace the use of CPU with a Graphics Processing Unit (GPU) as the main transcoding processor. Currently, there is limited support of VP8 hardware acceleration support. The transcoding of VP8 is not yet fully implemented on a GPU. However, with the use of GPU, it can further reduce the number of transcoding processes and minimize CPU load, adding more numbers of camera support.



# References

- [1] "Analog vs. ip technologies." <http://www.discount-security-cameras.net/analog-vs-ip-technology.aspx>. Accessed: 2016-12-26.
- [2] S. Loreto and S. P. Romano, *Real-Time Communication with WebRTC: Peer-to-Peer in the Browser*. " O'Reilly Media, Inc.", 2014.
- [3] D. Ristic, *Learning WebRTC*. Packt Publishing Ltd, 2015.
- [4] C. O. Ohaneme, A. Azubogu, E. Ifeagwu, and L. Ohaneme, "Design and implementation of an ip-based security surveillance system," *IJCSI International Journal of Computer Science Issues*, vol. 9, no. 5, p. 393, 2012.
- [5] M. A. Hossain, "Framework for a cloud-based multimedia surveillance system," *International Journal of Distributed Sensor Networks*, vol. 2014, 2014.
- [6] T.-L. Kao, Y.-C. Lee, Y.-W. Lin, K.-T. Wu, and L.-C. Hwang, "Design of video surveillance in wireless router," in *Next-Generation Electronics (ISNE), 2013 IEEE International Symposium on*, pp. 287–290, IEEE, 2013.
- [7] R. Rashmi and B. Latha, "Video surveillance system and facility to access pc from remote areas using smart phone," in *Information Communication and Embedded Systems (ICICES), 2013 International Conference on*, pp. 491–495, IEEE, 2013.
- [8] "Quality comparison of googles vp8 and x264 using standard video sequences.." [http://www2.tkn.tu-berlin.de/research/evalvid/EvalVid/vp8\\_versus\\_x264.html](http://www2.tkn.tu-berlin.de/research/evalvid/EvalVid/vp8_versus_x264.html). Accessed : 2017 – 06 – 29.
- [9] D. Rodrigues, E. Cerqueira, and E. Monteiro, "Quality of service and quality of experience in video streaming," in *Proc. of the International Workshop on Traffic Man-*

## REFERENCES

- agement and Traffic Engineering for the Future Internet (FITraMEn2008)*, EuroNF NoE, Porto, Portugal, pp. 11–12, 2008.
- [10] N. G. La Vigne, S. S. Lowry, J. A. Markman, and A. M. Dwyer, “Evaluating the use of public surveillance cameras for crime control and prevention,” *Washington, DC: US Department of Justice, Office of Community Oriented Policing Services*, 2011.
- [11] F. Nilsson *et al.*, *Intelligent network video: Understanding modern video surveillance systems*. CRC Press, 2008.
- [12] “Live streaming vs video on-demand (vod).” <https://blog.viddler.com/cracking-the-code-on-video-live-streaming-vs-video-on-demand/>. Accessed: 2018-07-02.
- [13] “Live streaming vs video on-demand (vod).” <https://blog.viddler.com/cracking-the-code-on-video-live-streaming-vs-video-on-demand/>. Accessed: 2018-07-02.
- [14] “Adobe blogs.” <http://blogs.adobe.com/>. Accessed: 2018-07-12.
- [15] “Ffserver.” <https://trac.ffmpeg.org/wiki/ffserver>. Accessed: 2018-07-20.
- [16] “WebRTC home.” <https://webrtc.org/>. Accessed: 2016-09-15.
- [17] R. Schollmeier, “A definition of peer-to-peer networking for the classification of peer-to-peer architectures and applications,” in *Peer-to-Peer Computing, 2001. Proceedings. First International Conference on*, pp. 101–102, Aug 2001.
- [18] “Google inc..” [www.google.com](http://www.google.com). Accessed: 2016-09-16.
- [19] “On2 technologies.” <http://www.on2.com/>. Accessed: 2016-09-20.
- [20] “Vp9 video codec summary.” <http://www.webmproject.org/vp9/>. Accessed: 2016-10-06.
- [21] H. Schwarz, D. Marpe, and T. Wiegand, “Overview of the scalable video coding extension of the h. 264/avc standard,” *IEEE Transactions on circuits and systems for video technology*, vol. 17, no. 9, pp. 1103–1120, 2007.
- [22] “The webm project.” <https://www.webmproject.org/>. Accessed: 2016-10-10.
- [23] “Global ip solutions.” <http://www.gipscorp.com/>. Accessed: 2016-10-12.
- [24] “World wide web consortium (w3c).” <https://www.w3.org/>. Accessed: 2016-10-12.

## REFERENCES

- [25] "Internet engineering task force (ietf)." <https://www.ietf.org/>. Accessed: 2016-10-18.
- [26] D. Flanagan, *JavaScript: the definitive guide*. " O'Reilly Media, Inc.", 2006.
- [27] I. Fette, "The websocket protocol," 2011.
- [28] A. B. Johnston and D. C. Burnett, *WebRTC: APIs and RTCWEB Protocols of the HTML5 Real-Time Web*. Digital Codex LLC, second ed., sep 2012.
- [29] J. Rosenberg, H. Schulzrinne, G. Camarillo, A. Johnston, J. Peterson, R. Sparks, M. Handley, and E. Schooler, "Sip: session initiation protocol," tech. rep., 2002.
- [30] I. S. Graham, *The HTML sourcebook*. John Wiley & Sons, Inc., 1995.
- [31] K. Egevang and P. Francis, "The ip network address translator (nat)," tech. rep., 1994.
- [32] R. Manson, *Getting Started with WebRTC*. Packt Publishing Ltd, 2013.
- [33] M. Baugher, D. McGrew, M. Naslund, E. Carrara, and K. Norrman, "The secure real-time transport protocol (srtp)," tech. rep., 2004.
- [34] D. Wing, "Symmetric rtp/rtp control protocol (rtcp)," 2007.
- [35] D. McGrew and E. Rescorla, "Datagram transport layer security (dtls) extension to establish keys for secure real-time transport protocol (srtp)," 2010.
- [36] J. Rosenberg, "Interactive connectivity establishment (ice): A methodology for network address translator (nat) traversal for multimedia session establishment protocols," *Work in progress*, 2005.
- [37] J. Rosenberg, R. Mahy, P. Matthews, and D. Wing, "Session traversal utilities for nat (stun)," tech. rep., 2008.
- [38] J. Rosenberg, "Traversal using relays around nat (turn): Relay extensions to session traversal utilities for nat (stun)," in *work in progress*, Citeseer, 2008.
- [39] R. Jesup, S. Loreto, and M. Tuexen, "Webrtc data channels," *IETF, Standards Track, draft-ietf-rtcweb-data-channel-13.txt*, 2014.
- [40] J. J. Garrett *et al.*, "Ajax: A new approach to web applications," 2005.

## REFERENCES

- [41] K. R. Fall and W. R. Stevens, *TCP/IP illustrated, volume 1: The protocols*. Addison-Wesley, 2011.
- [42] U. D. P. UDP and D. Sockets, "User datagram protocols," *Transport*, 2014.
- [43] R. R. Stewart and Q. Xie, "Stream control transmission protocol (sctp)," 2001.
- [44] P. Srisuresh, B. Ford, and D. Kegel, "Rfc 5128 state of peer-to-peer (p2p) communication across network address translators (nats)," *IETF, March*, 2008.
- [45] R. Mahy, P. Matthews, and J. Rosenberg, "Traversal using relays around nat (turn): Relay extensions to session traversal utilities for nat (stun)," tech. rep., 2010.
- [46] F. Audet and C. Jennings, "Network address translation (nat) behavioral requirements for unicast udp," tech. rep., 2007.
- [47] S. Guha, K. Biswas, B. Ford, S. Sivakumar, and P. Srisuresh, "Nat behavioral requirements for tcp," tech. rep., 2008.
- [48] J. Rosenberg, "Tcp candidates with interactive connectivity establishment (ice)," 2012.
- [49] J. Rosenberg and H. Schulzrinne, "An offer/answer model with sdp," 2002.
- [50] S. Strowes, "Ice, turn and stun," oct 2008.
- [51] J. Rosenberg, "Interactive connectivity establishment: Nat traversal for the session initiation protocol," *IETF journal*, vol. 2, pp. 14–19, 2006.
- [52] M. Handley, C. Perkins, and V. Jacobson, "Sdp: session description protocol," 2006.
- [53] C. Jennings and J. Uberti, "Javascript session establishment protocol," 2012.
- [54] P. Saint-Andre, "Extensible messaging and presence protocol (xmpp): Core," 2011.
- [55] R. Fielding and J. Reschke, "Hypertext transfer protocol (http/1.1): Semantics and content," 2014.
- [56] D. Crockford, "Json," 2012.
- [57] J. Bangali and A. Shaligram, "Design and implementation of security systems for smart home based on gsm technology," *International Journal of Smart Home*, vol. 7, no. 6, pp. 201–208, 2013.

## REFERENCES

- [58] "Onvif." <http://www.onvif.org/>. Accessed: 2017-03-30.
- [59] "Ffmpeg." <http://FFmpeg.org>. Accessed: 2017-05-02.
- [60] "Janus webrtc gateway." <https://janus.conf.meetecho.com/>. Accessed: 2017-05-03.
- [61] A. Amirante, T. Castaldi, L. Miniero, and S. P. Romano, "Janus: a general purpose webrtc gateway," in *Proceedings of the Conference on Principles, Systems and Applications of IP Telecommunications*, p. 7, ACM, 2014.
- [62] H. S. RFC3550, S. CASNER, R. FREDERICK, and V. JACOBSON, "Rtp: A transport protocol for real time applications," *S. l.]: RFC Editor United States*, 2003.
- [63] I. Cisco Systems and C. N. A. Program, *Cisco Networking Academy Program: CCNA 1 and 2 Companion Guide*. No. v. 1-2 in Cisco Networking Academy Program series, Cisco Press, 2003.
- [64] "Choosing a frame rate." <https://documentation.apple.com/en/finalcutpro/user-manual/index.html#chapter=D%26section=4%26tasks=true>. Accessed: 2017-05-04.
- [65] "Identifiers for webrtc's statistics api." <http://w3c.github.io/webrtc-stats/>. Accessed: 2017-05-10.
- [66] S. Vegesna, *IP quality of service*. Cisco press, 2001.
- [67] "webrtc-internals," 2017.
- [68] "About axis." <https://www.axis.com/about-axis>. Accessed: 2018-06-12.
- [69] "What you need to know about hls: Pros and cons." <https://blog.red5pro.com/what-you-need-to-know-about-hls-pros-and-cons/>. Accessed: 2018-06-12.
- [70] "Wireshark," 2017.
- [71] U. Lamping and E. Warnicke, "Wireshark user's guide," *Interface*, vol. 4, no. 6, 2004.
- [72] K. Seshadrinathan, R. Soundararajan, A. C. Bovik, and L. K. Cormack, "A subjective study to evaluate video quality assessment algorithms," in *IS&T/SPIE Elec-*

## REFERENCES

- tronic Imaging*, pp. 75270H–75270H, International Society for Optics and Photonics, 2010.
- [73] M. C. Farias, *Video quality metrics*. INTECH Open Access Publisher, 2010.
- [74] A. M. Eskicioglu and P. S. Fisher, “Image quality measures and their performance,” *IEEE Transactions on communications*, vol. 43, no. 12, pp. 2959–2965, 1995.
- [75] Z. Wang, A. C. Bovik, H. R. Sheikh, and E. P. Simoncelli, “Image quality assessment: from error visibility to structural similarity,” *IEEE transactions on image processing*, vol. 13, no. 4, pp. 600–612, 2004.
- [76] V. Q. E. Group *et al.*, “Final report from the video quality experts group on the validation of objective models of video quality assessment, phase ii (fr\_tv2),” *ftp://ftp.its.bldrdoc.gov/dist/ituvidq/Boulder\_VQEG\_jan\_04/VQEG\_PhaseII\_FRTV\_Final\_Report\_SG9060E.doc*, 2003.
- [77] Y. Wang, “Survey of objective video quality measurements,” 2006.
- [78] E. Fosser and L. O. D. Nedberg, “Quality of experience of webrtc based video communication,” Master’s thesis, NTNU, 2016.
- [79] “Big buck bunny download.” <https://peach.blender.org/download/>. Accessed: 2017-11-30.