University of Bamberg

Professorship for Computer Science

Communication Services, Telecommunication Systems and Computer Networks

Project on

Video Transport from Single-Board Computers like the Rasperry Pi in the Internet of Things for Web Real-Time Commuincations

Submitted by:

Likitha Purushotham

Mahalakshmi Suravarapu

Pio Chibuzor Okongwu

Supervisor: Prof. Dr. Udo Krieger & Marcel Großmann

Bamberg, October 28, 2020

Summer Term 2020

CONTENTS

List of Figures

LIST OF TABLES

LISTINGS

LIST OF ABBREVIATIONS

API	Application Programming Interface
ARM	previously an acronym for Advanced RISC Machine and originally Acorn RISC Machine
BCM	Broadcom SoC Channel
CPU	Central Processing Unit
CSV	Comma-Separated Values
DVD	Digital Versatile Disc or Digital Video Disc
FFmpeg	Fast Forward MPEG
GB	Gigabyte
GHz	Gigahertz
GPIO	General-Purpose Input/Output
GPU	Graphics Processing Unit
HD	High-Definition
HDMI	High-Definition Multimedia Interface
IP	Internet Protocol
IoT	Internet of Things
JPEG	Joint Photographic Experts Group
LPDDR	Low-Power Double Data Rate
MB	Megabyte
MBps	Megabytes per second
MJPEG	Motion JPEG
MPEG	Moving Picture Experts Group
Mbps	Megabits per second
NAT	Network Address Translation
OS	Operating System
RAM	Random-Access Memory
RPi	Raspberry Pi
RTC	Real-Time Communication
RTP	Real-time Transport Protocol
SBC	Single-Board Computer
SD	Secure Digital
SDP	Session Description Protocol
SDRAM	Synchronous Dynamic Random-Access Memory
SoC	System-on-a-Chip
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
UI	User Interface
URL	Uniform Resource Locator
USB	Universal Serial Bus
VLC	VideoLAN Clien
WI-FI	Wireless Fidelity
WebRTC	Web Real-Time Communication
cAdvisor	Container Advisor
mm	Millimeter

I. Introduction

The rapid growth in the technology and information sector in recent years has resulted in a wide range of connectivity for various electronic and heterogeneous devices that can communicate effortlessly with each other over the Internet. Due to this technological improvement, termed as the Internet of Things (IoT), has made a gateway for various industrial sectors to inculcate IoT with varied applications that can rule over the Internet.

This technological improvement in recent years, especially in the Healthcare sector, has made video streaming in real-time improve our lives in tangible ways where one can monitor the other irrespective of the location. To achieve this goal, current multimedia streaming hardware devices are typically expensive. In order to provide improved and reliable services, we require low-cost hardware devices with easy deployment and high scalability that can be easily integrated into the already existing IoT networks. In our project, we use devices such as single-board computers (SBC) that enables video streaming using the Web Real-Time Communication (WebRTC) framework and peer-to-peer connection.

Our system built on Raspberry Pi (RPi) is completely packed into Docker Containers, which is easy to deploy using effective and readily available ARM architecture. It enables us to receive the video stream from the attached camera node and send it to the multimedia framework, which encodes, decodes, and sends it to the receiving device using a real-time communication protocol (RTP). We document the development of this system on RPi4 in this project paper.

A. Goals

Our goal is to determine the best multimedia framework supported on the RPi by comparing the utilization of working video streams on RPi3 and RPi4 in terms of CPU usage, memory usage, and network throughput.

B. Related Work

Lukas and Vanessa's [1] "WebRTC Surveillance Prototype on a Raspberry Pi" was the base for our project, where they were able to set up a video surveillance system based on peer-to-peer communication, which is deployed on a RPi3. This system support video streaming using the WebRTC framework and aiortc. They also compared different multimedia frameworks and codecs and concluded FFmpeg and GStreamer as the best using H.264 and Vp8 respectively. Based on this, we try to build the system on RPi4 and figure out the major difference in using RPi3 and RPi4 in terms of video streaming quality and resource utilization.

II. Foundations for Our System

A. Hardware Technologies

1) Raspberry Pi: The RPi[1] is a series of small SBCs which operates in an open-source ecosystem that features a Broadcom system on chip with integrated ARM architecture supported central processing unit (CPU) which is typically used in mobile phones and tablets. Due to its low-cost and availability, it provides high performance that puts the power of computing and digital making easier all over the world [2].

With various versions of RPi and evolved features, it is more widely used in many areas such as weather monitoring, learning programming skills, building hardware projects, home automation, and applications in various industries. It is a small and cheap computer that runs Linux but also provides a set of GPIO (general purpose input/output) pins that control the electronic components for physical computing and exploring the IoT [3].

2) Raspberry Pi 3 vs Raspberry Pi 4: The RPi hardware has evolved through several versions that feature variations in the type of the CPU, memory capacity, networking support, and peripheral-device support [2].

Raspberry Pi 4: RPi version 4 has a lot of advantages from the previous models. The processing speed, multimedia performance, connectivity, memory speeds, and even the secure digital (SD) card read and write speeds are much faster, and the system as a whole is robust enough. It is available with three different memory sizes - 1GB, 2GB, and 4GB, support Dual 4k with USB 3.0 - to be the right fit for any project [4].

Its main features are [5]:

- Processor: Broadcom BCM2711, Quad-core Cortex-A72 (ARM v8) 64-bit SoC @ 1.5GHz
- Memory: 1 GB, 2 GB, or 4 GB LPDDR4 SDRAM
- Connectivity: 802.11ac Wi-Fi, Gigabit Ethernet, Bluetooth 5.0
- Access: 40-pin GPIO header
- Video & Sound: 2 x micro-HDMI, 3.5 mm analogue audio-video jack
- Multimedia: H.265 4Kp60, H.264 1080p60
- SD card support: microSD card
- Input power: 5V via USB type-C up to 3A and GPIO header up to 3A

Raspberry Pi 3: Although RPi4 has extensible features, RPi3 is still on the go for its features like a full-size HDMI Port, More accessories (for now), and Gigabit Ethernet over USB 2.0.

Its main features are [5]:

- Processor: Broadcom BCM2837B0 quad-core A53 (ARMv8) 64-bit @ 1.4GHz
- Memory: 1 GB LPDDR2 SDRAM
- Connectivity: 802.11ac Wi-Fi, 300Mbps Ethernet, Bluetooth 4.0
- Access: 40-pin GPIO header
- Video & Sound: Full size – HDMI, 3.5 mm analogue audio-video jack
- Multimedia: H.264 & MPEG-4 1080p30
- SD card: microSD card
- Input power: 5V via micro USB up to2.5A and GPIO header up to 3A

[1] https://en.wikipedia.org/wiki/Raspberry_Pi

B. Software Technologies

1) WebRTC: WebRTC is an open-source application that allows web browsers and mobile applications to communicate, transmit audio, video, and other generic data seamlessly through an Application Programming Interface (API) using a RTP. With WebRTC, most of the recent web browsers have built-in (Real-Time Communication) RTC functionality, which can find the best path for peer-2-peer connection, concealment of package loss, and echo cancellation during transmission, and built-in codecs for audio and video (U. Krieger: KTR-Mobicom-M-E, marcel Großmann), [6].

There are three major APIs used in WebRTC (U. Krieger: KTR-Mobicom-M-E, marcel Großmann), [6]:

- GetUserMedia: This enables the device to gain access to the camera, microphone, or screen of the device.
- peerConnection: This encodes and decodes the media and send it over the network, and takes care of the (Network Address Translation) NAT traversal.
- DataChannel: This creates the channel, sends arbitrary data directly between the browsers with low latency.

In sending of data between peers, the client browser M first sends a GET request to the web server, the web server confirms the request with Ok. Client browser L sends a GET request to the web server, which in succession it replies with Ok. The Client browser M negotiates with client browser L for media communication using a session description protocol (SDP). Once this negotiation is established, the web server creates a peer-to-peer connection between the two clients (U. Krieger: KTR-Mobicom-M-E, marcel Großmann), [6].

In our project, WebRTC plays a very important role as the main goal is to stream video from a camera node over a WebRTC connection. The main advantage over the other (Internet Protocol) IP based video surveillance systems is that the network structure need not be configured manually nor the camera node or client require information before connecting, whereas in the case of IP based systems, they require cameras to be accessible via known public IP address or route video traffic through a server. Using WebRTC and peer-to-peer connection, traffic is sent between both peers directly if possible and routing is done dynamically. This allows for the flexible deployment of camera nodes with less or no configuration.

2) Docker: Docker is an OS-Level virtualization technology used to run a so-called container on the host. Containers are lightweight when compared to virtual machines that use lightweight kernel namespaces and runs on Build, Ship, and Run Paradigm. The main idea of containers is to collect all the tools and libraries necessary to run a specific piece of software, and then isolating that software from the rest of the system. Because containers are not full-on virtual machines, they are efficient, and Docker which is lightweight, standalone, is a leader in making containers easy to work [7].

In our project, we use Docker on RPi4 of 4GB RAM with a quad-core ARM processor of 1.5GHz to pack all the components into containers, which helps the system to be portable, modular and make deployment easy without any complicated steps.

We pull the image *arm64v8/debian:buster-backports* from the registry to build the video streaming image. Then we run the video streaming containers.

C. Multimedia Frameworks

"A multimedia framework[2] is a software framework that handles media on a computer and through a network [8]." Various multimedia frameworks are available, but we will only look at ones that are peculiar to our project.

1) CVLC: CVLC is an interfaceless VLC player [9]. It is an open-source multimedia framework, which is built-in with encoding, decoding, and transcoding libraries [10].

2) FFmpeg: FFmpeg[3] is the term for Fast Forward MPEG, which can decode, encode, transcode, mux, demux, stream, filter, and play video streams.

3) MJPG-Streamer: MJPG-Streamer[4] is a command-line application. It copies JPEG frames from multiple inputs to multiple outputs. It streams a sequence of JPEG frames [11].

4) GStreamer: GStreamer[5] is an open-source multimedia framework that uses a pipeline-based workflow to combine different media processing systems. It reads files in one format, processes them, and exports them in another format [12].

In our project, we use these four multimedia frameworks. For CVLC, and FFmpeg, we use two codecs H.264 and MJPEG, to encode the video stream. And for GStreamer we use H.264. The comparison of the multimedia frameworks and codecs is discussed in Sec. IV

D. Video Codecs

"A video codec[6] is software or hardware that compresses and decompresses digital video [13]."

1) H.264: H.264[7] is one of the standard video codecs which uses the DVD standard (MPEG-2) for the quality video image. It is also known for its maximum bitrate and resolution, which is good support for 4k and up to 8k Ultra High-Definition [14].

2) MJPEG: MJPEG[8], which is also known as Motion JPEG is a stream of JPEG images transferred over https protocol. The main advantage of MJPEG is that the installation of client software is not necessary on a remote host. The video stream can be seen in any software that supports MJPEG streaming (E.g., Google Chrome, Mozilla Firefox, VLC) [11].

[2]https://en.wikipedia.org/wiki/Multimedia_framework
[3]http://ffmpeg.org/about.html
[4]https://techvedika.com/mjpeg-linux-video-streaming-and-recording-over-http/
[5]https://en.wikipedia.org/wiki/GStreamer
[6]https://en.wikipedia.org/wiki/Video_codec
[7]https://www.streamingmedia.com/Articles/ReadArticle.aspx?ArticleID=74735
[8]https://techvedika.com/mjpeg-linux-video-streaming-and-recording-over-http/

E. Tools

1) cAdvisor: cAdvisor[9] also known as Container Advisor provides resource usage and performance traits of running containers. It collects, aggregates, processes, and exports information about running containers. It monitors complete resource usage and network statistics [15].

2) Prometheus: Prometheus[10] is an open-source software used for monitoring and alerting. It records metrics in real-time using predefined queries [16].

3) Jupyter Notebook: "The Jupyter Notebook[11] is an open-source web application that allows you to create and share documents that contain live code, equations, visualizations, and narrative text [17]."

F. Versions of Hardware and Software Technologies

The versions of hardware & software technologies and tools we used in our project are listed below:

- Raspberry Pi 3: Linux Black-pearl 4.19.58-hypriotos-v8, aarch64 1.2GHz ARM
- Raspberry Pi 4: Linux Black-pearl 4.19.102-v8, aarch64 1.5GHz ARM
- Docker: v18.09.7
- cAdvisor: v0.36.0
- Prometheus: v2.21.0
- Jupyter Notebook: v6.1.3
- Frameworks: arm64v8/debian:buster-backports v0.11
- Janus (WebRTC server): v0.10.3

[9]https://github.com/google/cadvisor
[10]https://en.wikipedia.org/wiki/Prometheus_(software)
[11]https://jupyter.org

III. System Architecture and its Working

A. Proposed Architecture for Video Streaming with Raspberry Pi's

Our prototype for the project is based on the architecture consisting of RPi, camera node, client (receiving device), as shown in Fig. 1.

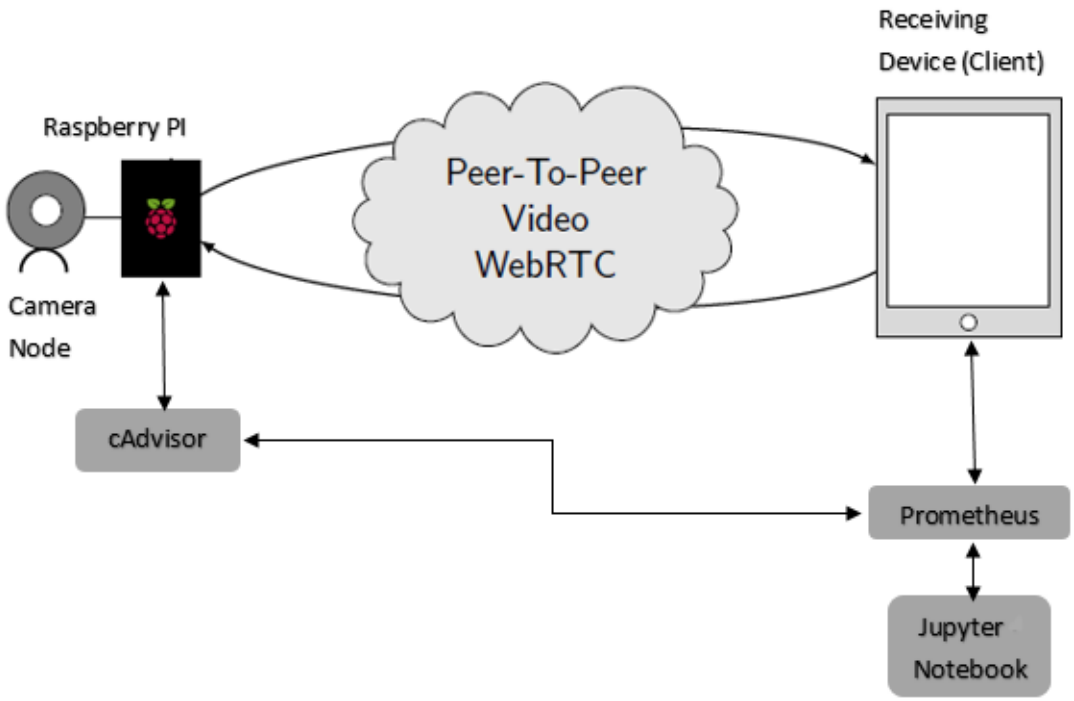

Figure 1: System Architecture

The main implementation in our project is built upon by running Docker containers to allow for easy deployment and management of its dependencies and starting all multimedia frameworks (CVLC, FFmpeg, MJPG-Streamer, GStreamer) with its codecs (H.264, MJPEG) on RPi.

We build all the necessary containers (frameworks and cAdvisor) on the RPi, where its respective Docker images are written in each of its Docker files. Navigate to each of the framework directories on RPi and start the framework by building and running Docker images, which it directs to its respective shell. We run the command to start the video streaming framework with specific encoders (H.264/MJPEG) in the shell. The workflow and commands for all the frameworks and codecs are explained in detail in Sec. A.

In order to view the video stream from RPi, a system (receiving device) that supports WebRTC peer-to-peer connection is integrated into the RPi network. Here we make use of VLC media player or web browser, based on the framework specification to view the video stream either through UDP (User Datagram Protocol) or TCP (Transmission Control Protocol) connection.

As our main goal in the project is to make a comparison between the multimedia frameworks and codecs in RPi3 and RPi4 based on the quality of video and resource utilization, we collect the statistics of the multimedia frameworks with its codecs from

both RPi3 and RPi4 through cAdvisor (Container Advisor), a tool that exposes the raw data of resource usage and performance data of the running containers.

The workflow of cAdvisor in our project is as shown below:

To start cAdvisor, follow the steps shown in Code Listing. 1

Listing 1: Build and run cAdvisor

```
$docker run \
  --volume=/:/rootfs:ro \
  --volume=/var/run:/var/run:rw \
  --volume=/sys:/sys:ro \
  --volume=/var/lib/docker/:/var/lib/docker:ro \
  --volume=/dev/disk/:/dev/disk:ro \
  --publish=8080:8080 \
  --detach=true \
  --name=cadvisor \
  unibaktr/cadvisor

To view cAdvisor:
In browser: <Ip of the Rpi>:8080
```

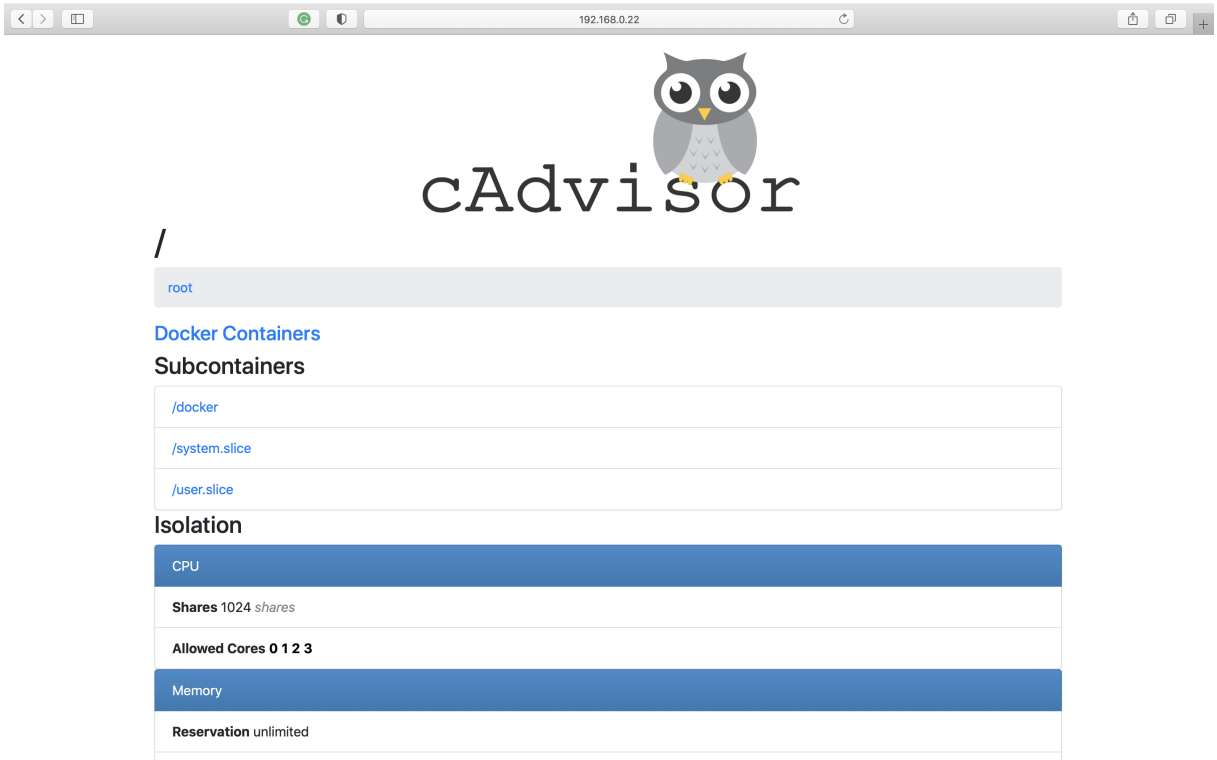

Figure 2: cAdvisor

Based on the statistics, to provide an easy, clear, and descriptive understanding of the data that has been generated, we make use of one of the graphical representations, **Boxplot**, to provide an illustration of values distributed based on mean, median, and maximum data more precisely in less space.

To draw the boxplot, we require the data to be present in a specific file format with its detailed statistics. As cAdvisor provides the raw data, we use **Prometheus**, an open-source software application used for monitoring and recording the metrics in real-time

using pre-defined queries. In our case, Prometheus scrapes the exposed raw data metrics from cAdvisor using queries and saves the metrics to the file extension we require (in our case CSV file) locally.

The workflow of Prometheus in our project is as shown below:

You can run Prometheus on Docker, as the Prometheus services are available as Docker images.

Listing 2: Dockerfile for Prometheus

```
FROM prom/prometheus
ADD prometheus.yml /etc/prometheus/
```

Listing 3: prometheus.yml file which consists of configuration

```
global:
  scrape_interval:15s
  scrape_timeout:10s
  evaluation_interval:20s
scrape_configs:
-job_name: cadvisor
  honor_timestamps:true
  scrape_interval:15s
  scrape_timeout:10s
  metrics_path: /metrics
  scheme: http
  static_configs:
  -targets:
    -<IP of the RPi>:8080
```

To build and run Prometheus on Docker, follow the steps shown in Code Listing. 4

Listing 4: Commands to build and run Prometheus on Docker

```
Navigate to the following directory:
/framework-test/ForReceiving_Device/Prometheus

$docker build -t my-prometheus .

$docker run -p 9090:9090 my-prometheus

To access the User Interface(UI) of the Prometheus:
In browser: <Ip of the Receiving Device>:9090
```

Once you build and run Prometheus, you can access the User Interface (UI) in the browser. Using predefined queries, you can narrow the metrics of a running container, as shown in Fig. 3. Here, we are scraping the CPU usage of running container CVLC, which is using codec H.264 to stream the video.

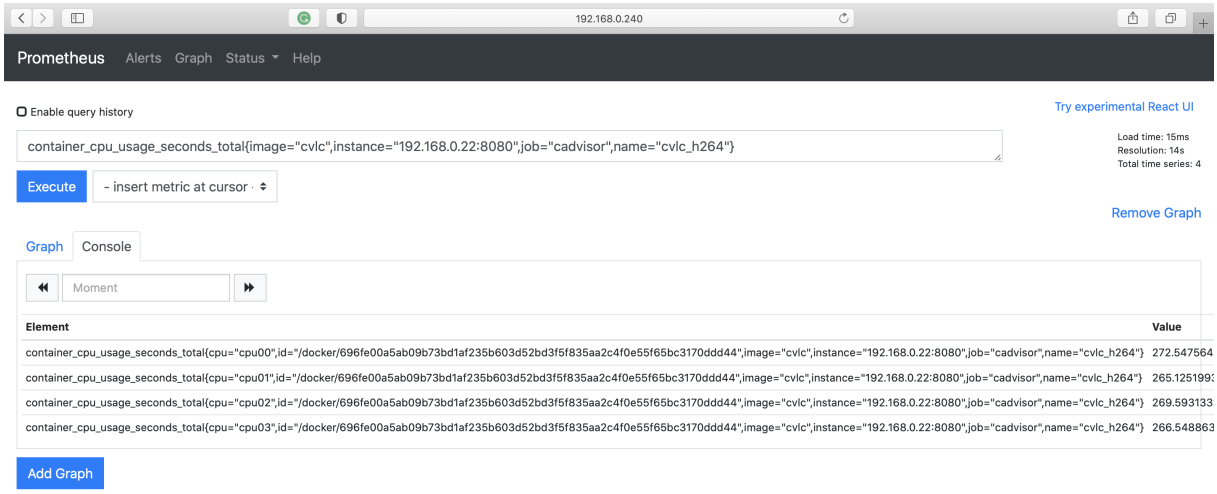

Figure 3: Prometheus UI

After narrowing down the metrics, we saved the metrics as a CSV file using Code Listing. 5.

Listing 5: Python script to save the metrics as csv file

```
import csv
import requests
import sys

if len(sys.argv) != 3:
        print('Usage: {0} http://localhost:9090 a_query'.format(sys.argv[0]))
        sys.exit(1)

response = requests.get('{0}/api/v1/query'.format(sys.argv[1]),
       params={'query':sys.argv[2]})
results = response.json()['data']['result']

# Build a list of all label names used
# gets all keys and discard _name_
labelnames = set()
for result in results:
         labelnames.update(result['metric'].keys())

# Canonicalize
labelnames.discard('__name__')
labelnames = sorted(labelnames)

# Write the header
writer = csv.writer(sys.stdout)
writer.writerow(['name', 'timestamp', 'value'] + labelnames)

# Write the samples
for result in results:
        l = [result['metric'].get('__name__', '')] + result['value']
        for label in labelnames:
                l.append(result['metric'].get(label, ''))
                 writer.writerow(l)
```

We run the Python script, as shown in Code Listing. 6. The Python file should be in the same directory where you are going to save the CSV files.

Listing 6: Run Python script

```
$python <name of python file>.py http://localhost:9090
'container_cpu_usage_seconds_total{image="cvlc",instance="<IP of the RPi>",job=
"cadvisor",name="cvlc_h264"}' | gzip > $date +"%Y_%m_%d_%I_%M_%p")_gendata.gz
```

Once the data is scraped and saved locally to the file extension (CSV files), we use these files to plot the boxplot using **Jupyter Notebook**, an open-source web application that allows to create and share documents that contain live code, equation, visualizations, and narrative text. In our case, we use Jupyter Notebook to write Python scripts and draw the boxplot to give an illustration of comparison between RPi3 and RPi4 based on multimedia frameworks statistics.

The workflow of how we configure Jupyter Notebook and write Python scripts to draw the boxplot is shown below.

You can also run Jupyter Notebook on Docker, as the Jupyter Notebook services are also available as Docker images.

Listing 7: Dockerfile for Jupyter Notebook

```
ARG BASE_CONTAINER=jupyter/datascience-notebook
FROM $BASE_CONTAINER

COPY . ./

WORKDIR $HOME
```

Listing 8: dockerBuild.sh

```
$docker build -t jupyter-notebook/hawk .
```

Listing 9: dockerRun.sh

```
$docker run \
 --env PORT=8888 \
 -it \
 -p 8888:8888 \
 jupyter-notebook/hawk
```

Listing 10: Build and run Jupyter Notebook on Docker

```
Navigate to the following directory:
/framework-test/ForReceiving_Device/jupyter-notebook

$./dockerBuild.sh

$./dockerRun.sh
```

Once you build and run Jupyter Notebook, you will get the URL to access the Jupyter Notebook server in the browser. Due to the ephemeral nature of Docker container, every saved notebook will be lost when we rebuild the container. So we dragged the necessary notebook files into the root directory of Jupyter, such that anytime we access the server, we could find the notebook which we already created.

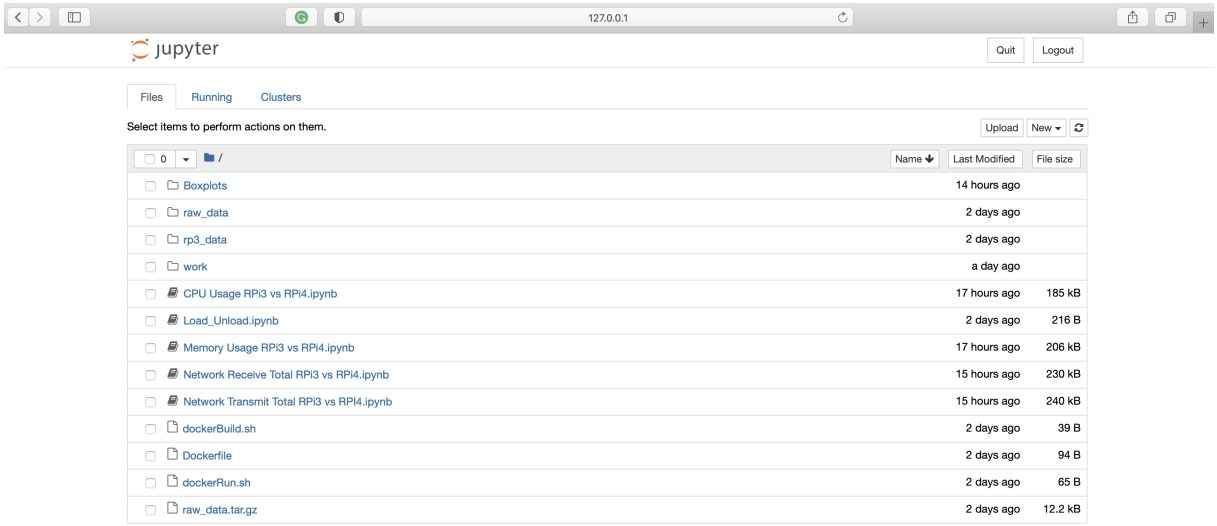

Figure 4: Jupyter Notebook Server

IV. Evaluation of Multimedia Frameworks and Codecs

A. Latency, Quality, and Resolution of Multimedia Frameworks and Codecs

After evaluating the multimedia frameworks and its associated codecs, we arrive at the statistics of each of the video stream, and from these statistics, we analyze and come to a decision of which framework shows the best results.

As from the bitrate column in Table I, MJPG-Streamer shows to be the best multimedia framework in our project. The number of processed bits within seconds determines the video quality. The higher the bitrate, the higher is the quality of the streaming video. In our case, MJPG-Streamer shows the best results followed by FFmpeg with the MJPEG codec.

Table I: Statistics - Input

Frameworks and Codecs	Read at media	Input bitrate	Demuxed	Stream bitrate
cvlc h264	8211 KiB	272 kb/s	6221 KiB	222 kb/s
cvlc mjpeg	108015 KiB	3379 kb/s	101973 KiB	3268 kb/s
ffmpeg h264	10300 KiB	901 kb/s	9313 kb/s	830 kb/s
ffmpeg mjpeg	16544 KiB	1283 kb/s	16495 KiB	1345 kb/s
mjpgstreamer	80031 KiB	7301 kb/s	79814 KiB	7575 kb/s

The video streams are formed by a sequence of frames which are calculated per given time unit. The higher the frame rate, the smoother the video will be. Table II shows the video frame rate, each video streaming framework displays per given time, and the number of lost frames. From the statistics, we can say that there is a higher number of lost frames in CVLC, which can be due to network and bitrate fluctuation.

Table II: Statistics - Video

Frameworks and Codecs	Decoded blocks	Displayed frames	Lost frames
cvlc h264	16.487	8.243	27
cvlc mjpeg	32.765	18.002	97
ffmpeg h264	4.177	2.067	0
ffmpeg mjpeg	4.710	2.376	0
mjpgstreamer	4.314	2.162	0

Among the two video codecs (H.264, MJPEG) we used, H.264 compresses chunks of video frames at once, whereas MJPEG compresses video frames as a sequence of JPEG images. From Table III, we can say that H.264 compresses chunks of video frames, whereas MJPEG has no frame rate as it compresses video frames sequentially. Coming to the video resolution MJPG-Streamer shows High-Definition (HD) video because of its high resolution.

Table III: Codec details

Frameworks and Codecs	Video resolution	Frame rate
cvlc h264	640x354	24
cvlc mjpeg	752x416	
ffmpeg h264	752x416	25
ffmpeg mjpeg	752x416	
mjpgstreamer	1280x720	

GStreamer can interconnect with other multimedia frameworks using their available libraries such as codecs. In our project, VLC does not support GStreamer and can only view the video stream on the web browser when it is streamed through a Janus (WebRTC server). Janus, on the other hand, supports multimedia frameworks, such as GStreamer, FFmpeg, etc, and video codecs H.264 and Vp8.

1) Video Quality and Resolution of Working Video Streams: The quality was evaluated with subject to each of the frameworks (see Fig. 5, 6, 7, 8, 9), and latency was measured by making single short movements in front of the camera and finding the difference between real movements and its appearance in the stream. The pictures of the resolution and the quality of the video streams are shown below:

Figure 5: CVLC H.264

Figure 6: CVLC MJPEG

Figure 7: FFmpeg H.264

Figure 8: FFmpeg MJPEG

Fig. 9 of MJPG-Streamer shows a better video quality among all the frameworks because of its high resolution followed by is the FFmpeg with MJPEG (Fig. 8)

Figure 9: MJPG-Streamer

At first, we were able to stream video using GStreamer. Subsequently, we couldn't view the streaming video even though everything is fine from the server-side. We checked the video port from the plug-in configuration file, and we assigned a different port. Still, we couldn't view the video stream. Fig. 10 shows the continuous buffering effect.

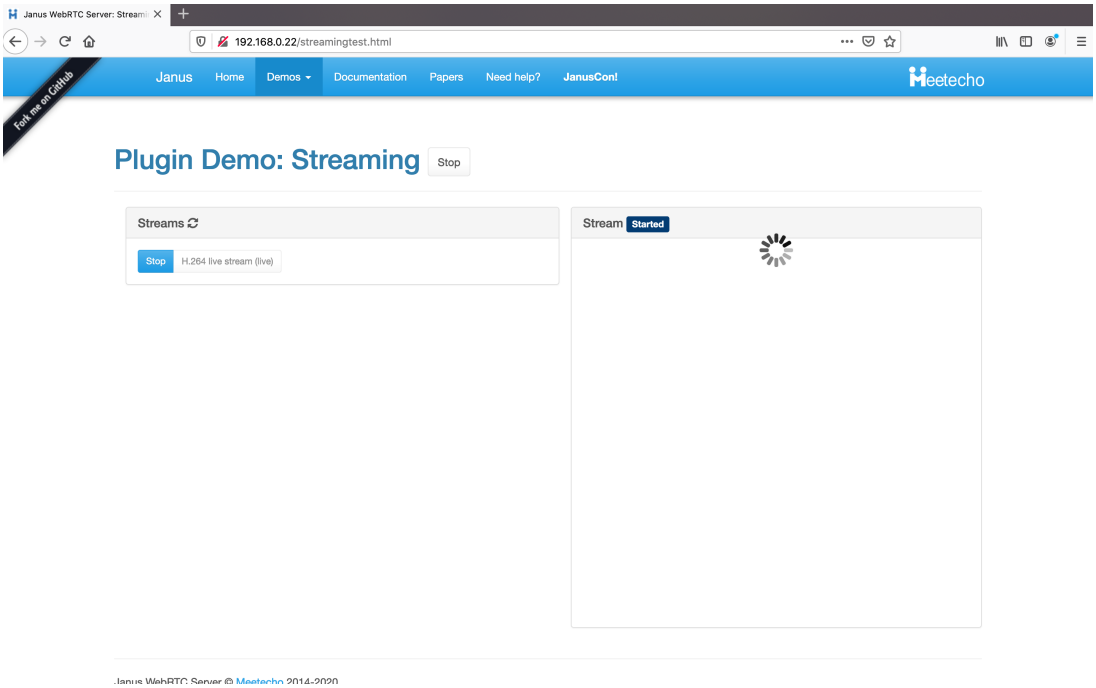

Figure 10: GStreamer H.264

B. Utilization Comparison of Working Video Streams

As our main goal in the project is to make a comparison between the multimedia frameworks on RPi3 and RPi4 based on the video quality and resource utilization, we collect the statistics of the multimedia frameworks with its codecs from both RPi3 and RPi4. Based on the statistics, to provide an easy, clear, and descriptive understanding of the data that has been generated, we make use of one of the graphical representations, Boxplot, to provide an illustration of values distributed based on mean, median, and maximum data more precisely in less space. Using Jupyter Notebook we write the Python scripts to draw the boxplots.

Here, we make a utilization comparison of RPi3 and RPi4 based on CPU usage, memory usage, and network throughput.

For all the multimedia frameworks and codecs, we took seven trials with a time interval of one minute.

CPU Utilization:

RPi is a quad-core processor. And when we scrape the metrics for CPU usage of running containers, we get values for all four cores. We calculated the mean for values of four cores to plot the boxplot.

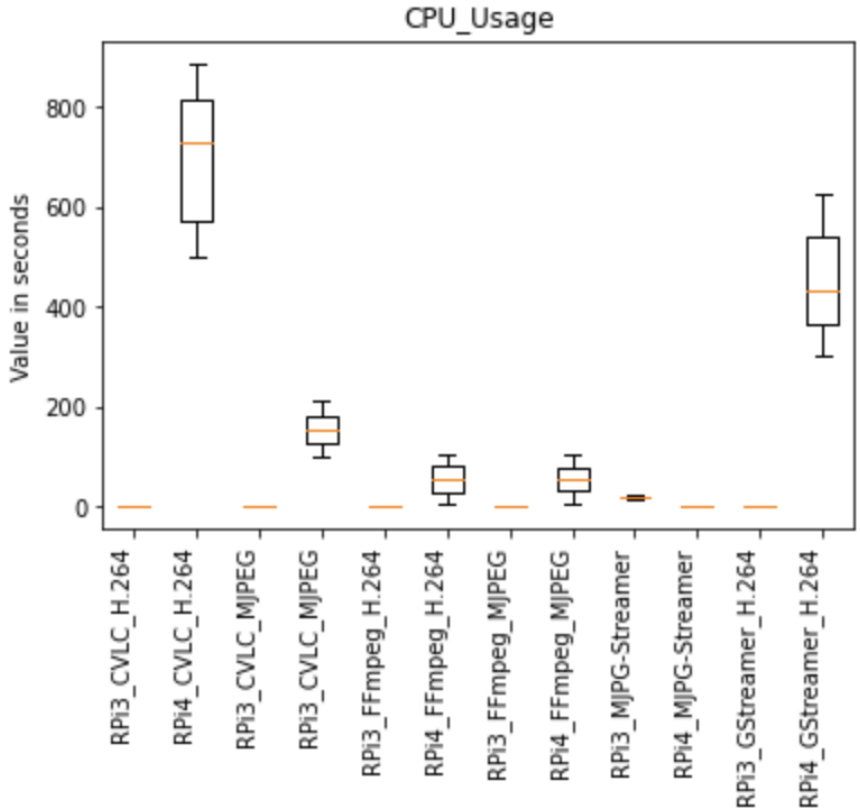

Figure 11: CPU Utilization

Fig. 11 shows the total CPU usage in seconds of all multimedia frameworks with its respective codecs. Unfortunately, in our project, we are unable to view the video stream from the RPi3 except for MJPG-Streamer, which affects the metrics showing no much

variation. Hence, we cannot conclude which multimedia framework is best in RPi3. In RPi4, the CPU utilization is high for CVLC (with H.264), which affects the latency, followed by GStreamer (with H.264) because of the running WebRTC container. MJPG-Streamer consumes a minimum amount of CPU followed by FFmpeg (with H.264 and MJPEG), which results in better performance of the system.

Memory Utilization:

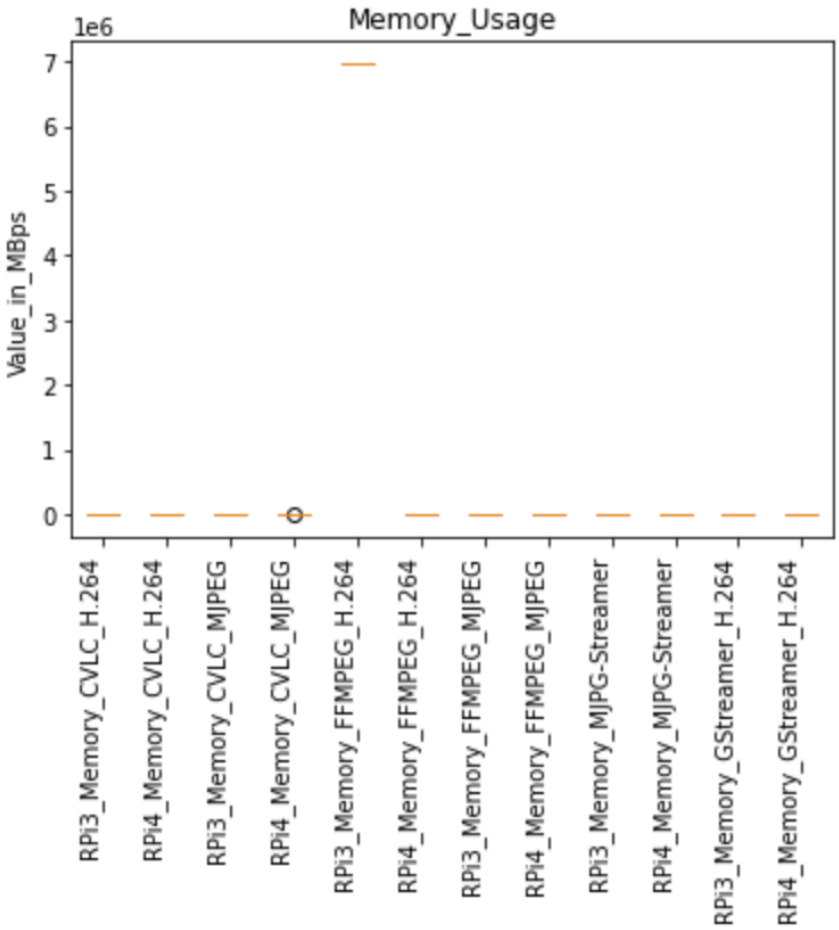

Figure 12: Memory Utilization

Fig. 12 shows the total memory usage in Megabytes per second (MBps) of all multimedia frameworks with its respective codecs. Unfortunately, in our project, we are unable to view the video stream from the RPi3 except for MJPG-Streamer, which affects the metrics showing no much variation. Even though there is no video streamed, there is a sudden spike in memory usage for FFmpeg (with H.264), which is uncertain. In RPi4, although all multimedia frameworks have unlimited memory allocation by cAdvisor, they consume a very minimal amount of memory, which is almost fixed across all the frameworks. Hence, we cannot conclude which multimedia framework is best in RPi3 and RPi4 as almost all frameworks consume a minimal amount of memory.

Network Transmit Total:

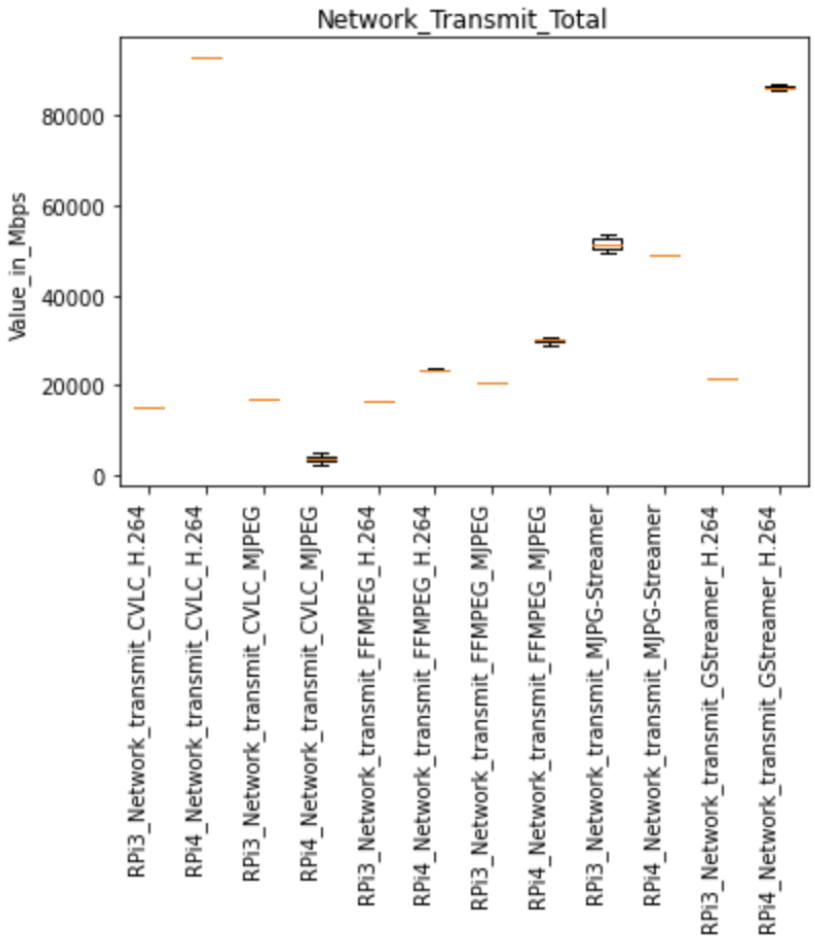

Figure 13: Network Transmit Total

Fig. 13 shows the total network transmission in Megabits per second (Mbps) for all multimedia frameworks with its respective codecs. The higher the bitrate and Network transmission (upstream), the better the video quality, which allows us to send all the allotted video frames over the network. Although we are unable to view the video stream from the RPi3 except for MJPG-Streamer, it shows the variation in values. And from the boxplot, MJPG-Streamer has a higher upstream followed by FFmpeg (with MJPEG), which shows better video quality. In RPi4, GStreamer (with H.264) has a higher upstream because of the running WebRTC container followed by MJPG-Streamer, which shows better video quality.

Network Receive Total:

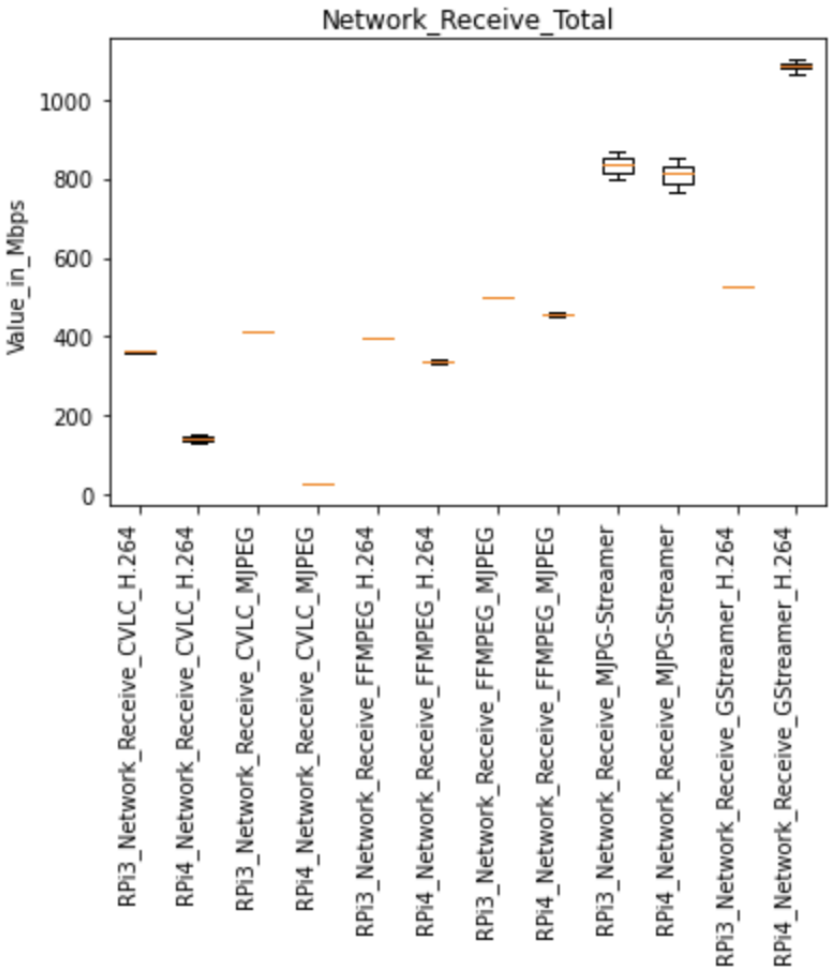

Figure 14: Network Receive Total

Fig. 14 shows the total network received in Megabits per second (Mbps) for all multimedia frameworks with its respective codecs. The receiving device receives all the packets sent from the streaming server (RPi). Although we are unable to view the video stream from the RPi3 except for MJPG-Streamer, it shows the variation in values. And from the boxplot, MJPG-Streamer has higher packets received followed by GStreamer (with H.264), which shows better video quality. In RPi4, GStreamer (with H.264) has higher packets received because of running WebRTC container followed by MJPG-Streamer, which shows better video quality. RPi4 provides better quality of video streaming as its ethernet interface is 100GB, whereas for RPi3 it is 100MB.

The individual boxplots for all the multimedia frameworks and codecs are shown in the Sec. A.

V. Conclusion and Future Work

With the rapid development in the field of Internet and technology, IoT is changing our lives by making things smarter and easier than ever before. This technological improvement has made a positive approach in many fields that are based on a peer-to-peer structure by running on cheap and easily available hardware devices. We built a multimedia system that uses RTP on the RPi4. Our system receives the video stream from the attached camera node, which is encoded and decoded by using two video codecs (H.264 and MJPEG) in four frameworks (CVLC, FFmpeg, MJPG-Streamer, GStreamer) and sent over the network using an RTP. To view the video streamed from RPi, we use the web browser or media-player (VLC) on the receiving device. As our system is virtualized using Docker containers, it makes deployment easy on RPi.

By comparing all the frameworks with its respective codecs on RPi3 and RPi4 in terms of resource utilization (CPU, memory, network throughput) and video quality, we arrive at a conclusion that MJPG-Streamer delivers the best performance overall and it only supports the MJPEG codec as WebRTC requires H.264 in RPi4. Followed by is the GStreamer providing the best results in RPi4. And about RPi3, unfortunately in our project, we are unable to view the video stream from the RPi3 except for MJPG-Streamer, which affects the result. This leads to an uncertain situation to determine which multimedia framework is the best on RPi3 and difficult to conclude which RPi is the best. In our project, we realized Janus (WebRTC server) supports very few video codecs and multimedia frameworks, this is why we haven't used MJPEG codec with GStreamer.

Also, we realized, ARM quad-core processor (RPi) does not fully support multimedia frameworks because of no built-in libraries and Graphics Processing Unit (GPU) chip.

While our prototype demonstrates the possibilities with current technology using cheap SBCs, future work may provide a more stable foundation in using various other multimedia frameworks for better video streaming quality and added new features.

APPENDIX

Steps to start the Multimedia Frameworks and Codecs

CVLC: To start CVLC, follow the steps shown in Code Listing. 11

Listing 11: Build and run CVLC

```
Navigate to the following directory:
/framework-test/ForRaspberryPi/cvlc

$docker build -t cvlc .

$docker run \
  -it \
  --privileged \
  --net=host -v /dev/video0:/dev/video0 \
  -e Ip=<IP of the Receiving Device> cvlc

h264:
$cvlc \
  -vvv v4l2:///dev/video0 \
  --sout "#transcode{vcodec=h264,
  width=640,
  fps=24,
  tune=zerolatency,
  preset=ultrafast}:udp{dst=<IP of the Receiving Device>:2000}" \
  --no-sout-audio

mjpeg:
$cvlc \
  -vvv v4l2:///dev/video0 \
  --sout "#transcode{vcodec=mjpg,
  fps=24}:udp{dst=<IP of the Receiving Device>:2000}" \
  --no-sout-audio

To view the video stream in VLC:
File->Open Network->Enter udp://@:2000
```

FFmpeg: To start FFmpeg, follow the steps shown in Code Listing. 12

Listing 12: Build and run Ffmpeg

```
Navigate to the following directory:
/framework-test/ForRaspberryPi/ffmpeg

$docker build -t ffmpeg .

$docker run \
  -it \
  --privileged \
  --net=host -v /dev/video0:/dev/video0 \
  -e Ip=<IP of the Receiving Device> ffmpeg

h264:
$ffmpeg \
  -fflags nobuffer \
  -f v4l2 \
  -framerate 30 \
  -i /dev/video0 \
```

```
  -vcodec libx264 \
  -preset ultrafast \
  -tune zerolatency \
  -pix_fmt yuv420p \
  -f mpegts udp://<IP of the Receiving Device>:2000?pkt_size=1316

mjpeg:
$ffmpeg \
  -fflags nobuffer \
  -f v4l2 \
  -framerate 30 \
  -i /dev/video0 \
  -vcodec mjpeg \
  -f mjpeg udp://<IP of the Receiving Device>:2000?pkt_size=1316

To view the video stream in VLC:
File->Open Network->Enter udp://@:2000
```

MJPG-Streamer: To start MJPG-Streamer, follow the steps shown in Code Listing. 13

Listing 13: Build and run MJPG-Streamer

```
Navigate to the following directory:
/framework-test/ForRaspberryPi/mjpgstreamer

docker build -t mjpgstreamer .

$docker run \
  -it \
  --privileged \
  --net=host \
  -v /dev/video0:/dev/video0 \
  -e Ip=<IP of the Receiving Device> mjpgstreamer

To view the video stream:
In browser: <IP of the Rpi>:8085
In vlc player: File->Open Network->Enter http://<IP of the Rpi>:8085/?action=stream
```

GStreamer: To start GStreamer, follow the steps shown in Code Listing. 14

Listing 14: Build and run GStreamer

```
To start Janus Server, navigate to the following directory:
/framework-test/ForReceivingDevice/janus_docker_for_testing

$docker run \
  --name=janus \
  --net=host \
  janus

To start GStreamer, navigate to the following directory:
/framework-test/ForRaspberryPi/gstreamer

$docker build -t gstreamer .

$docker run \
  -it \
  --privileged \
  --net=host \
  -v /dev/video0:/dev/video0 \
```

```
  -e Ip=<IP of the Receiving Device> gstreamer

h264:
./gstreamer_h264.sh

To view the video stream:
In browser: <Ip of the Rpi>
Demos->Streaming->Streams list ->H.264 live stream(live)->Watch or Listen

To stop the video stream:
Stop
```

Boxplots showing the Utilization comparison of RPi3 and RPi4

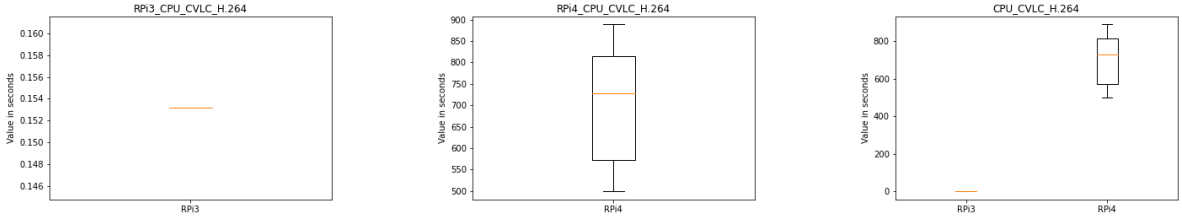

Figure 15: CPU CVLC H.264

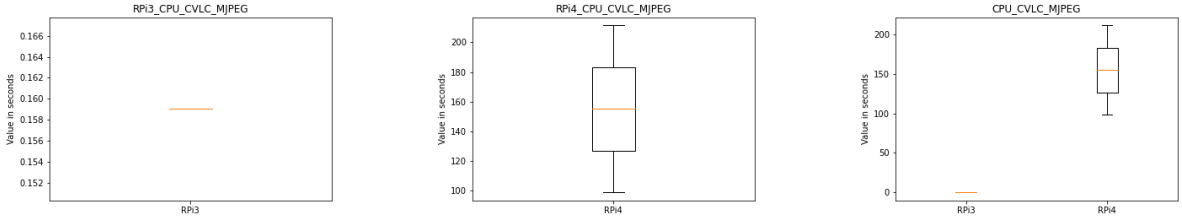

Figure 16: CPU CVLC MJPEG

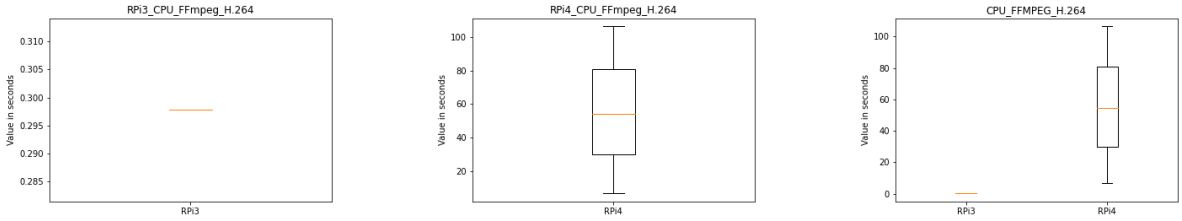

Figure 17: CPU FFmpeg H.264

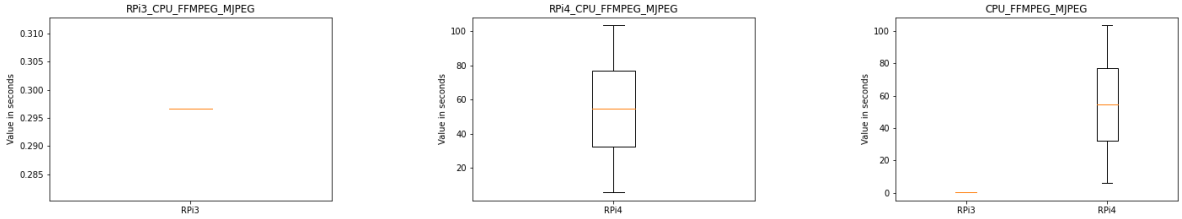

Figure 18: CPU FFmpeg MJPEG

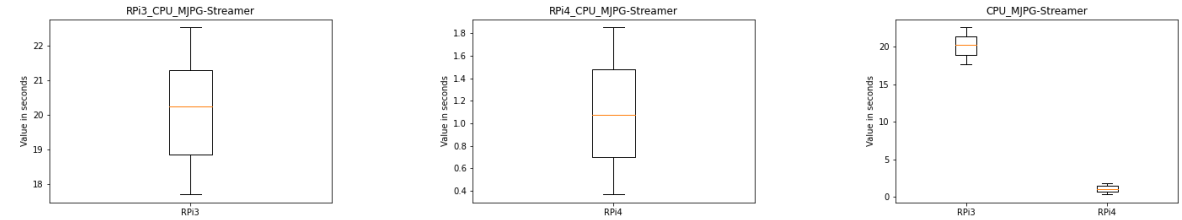

Figure 19: CPU MJPG-Streamer

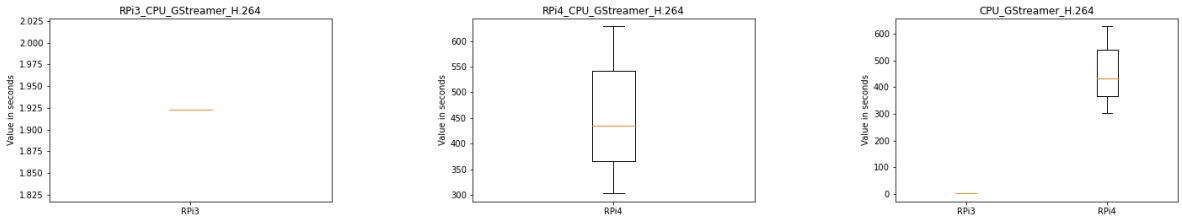

Figure 20: CPU GStreamer H.264

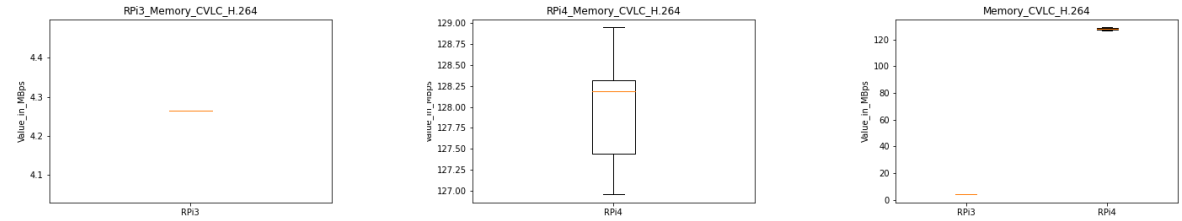

Figure 21: Memory CVLC H.264

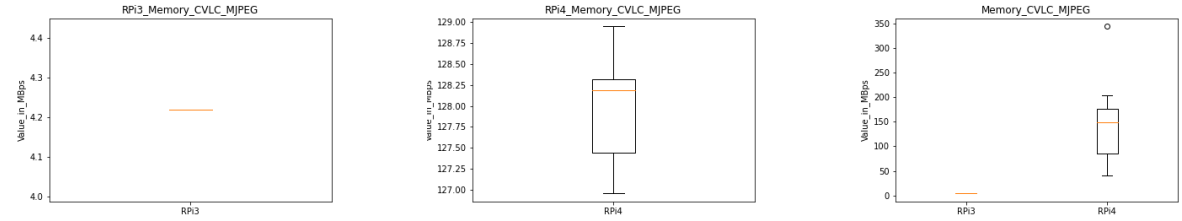

Figure 22: Memory CVLC MJPEG

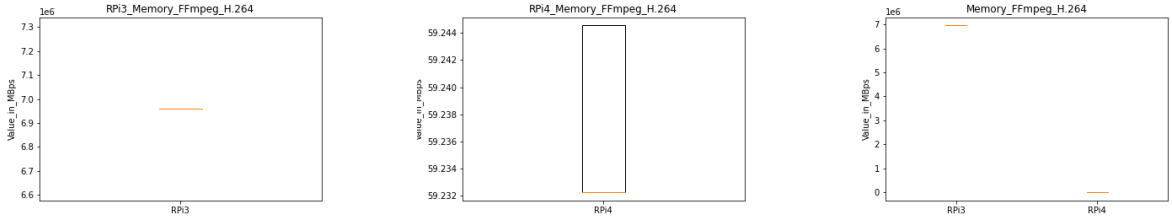

Figure 23: Memory FFmpeg H.264

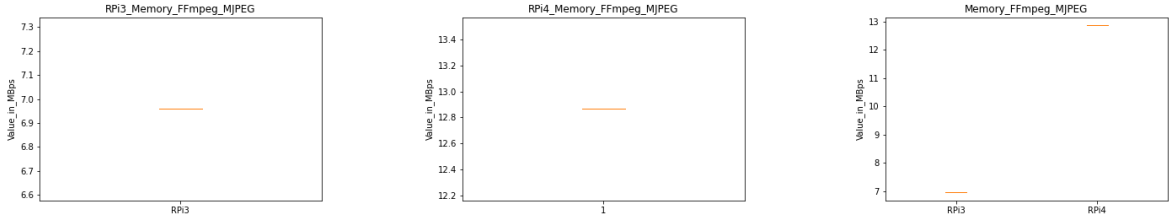

Figure 24: Memory FFmpeg MJPEG

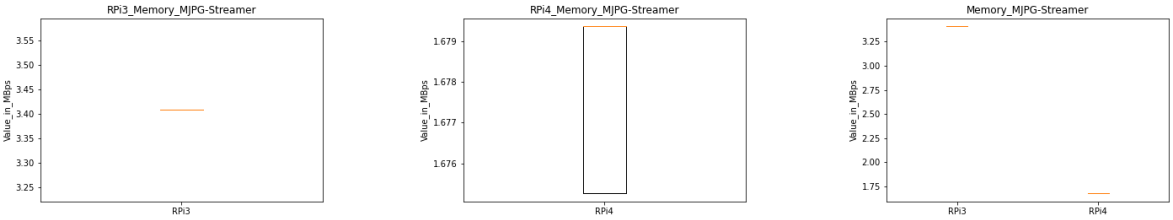

Figure 25: Memory MJPG-Streamer

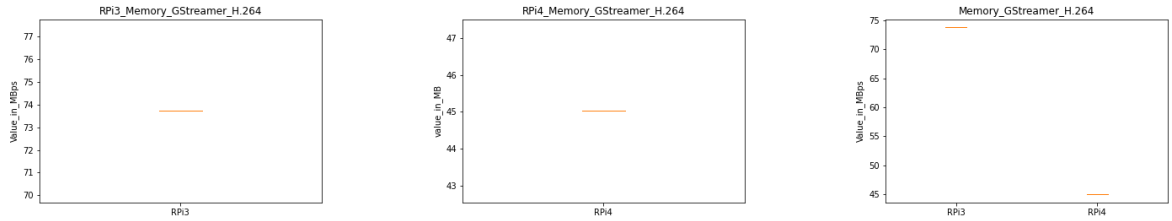

Figure 26: Memory GStreamer H.264

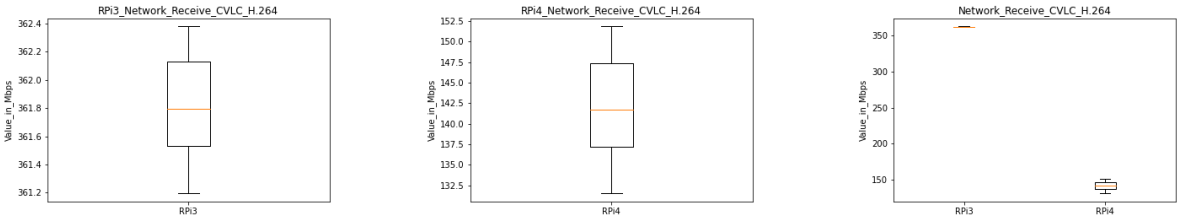

Figure 27: Network Receive CVLC H.264

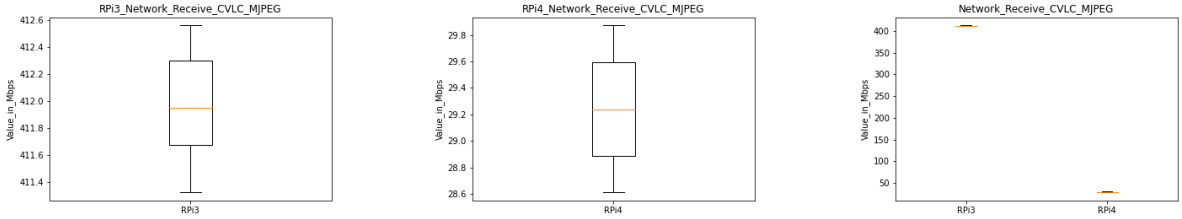

Figure 28: Network Receive CVLC MJPEG

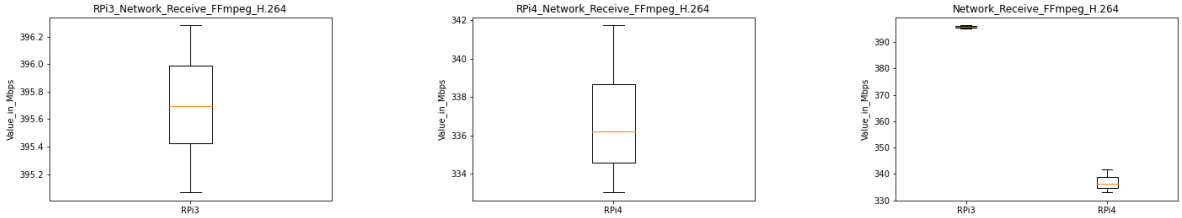

Figure 29: Network Receive FFmpeg H.264

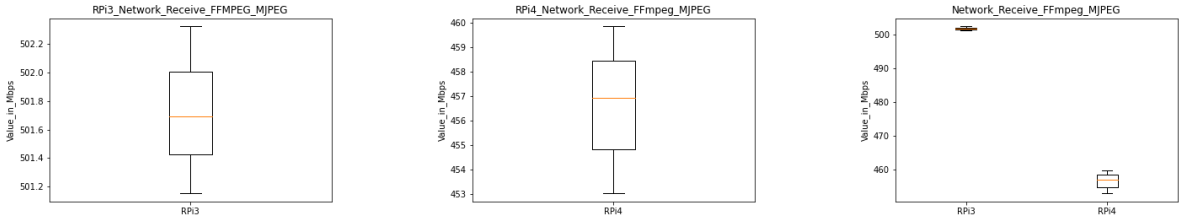

Figure 30: Network Receive FFmpeg MJPEG

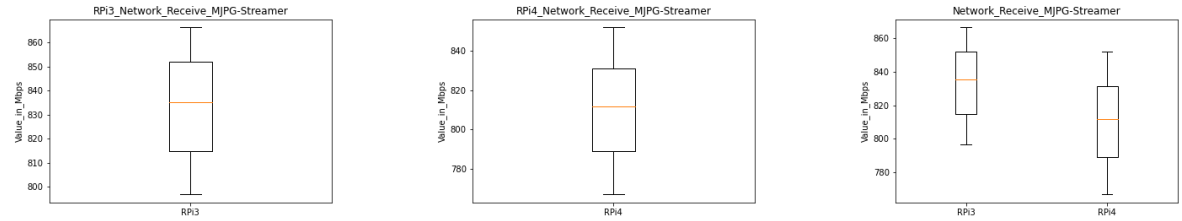

Figure 31: Network Receive MJPG-Streamer

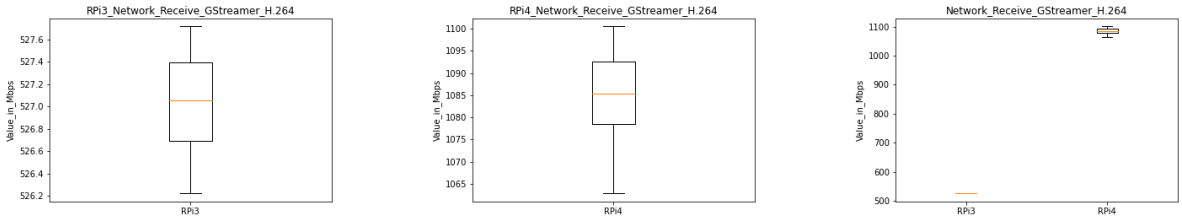

Figure 32: Network Receive GStreamer H.264

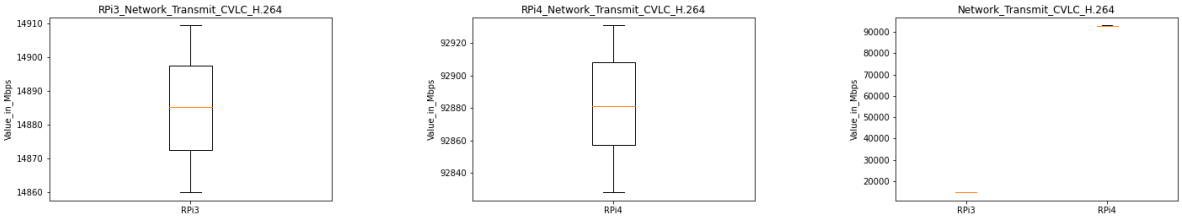

Figure 33: Network Transmit CVLC H.264

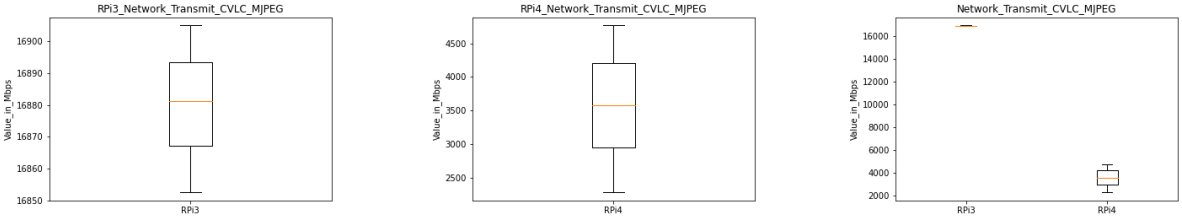

Figure 34: Network Transmit CVLC MJPEG

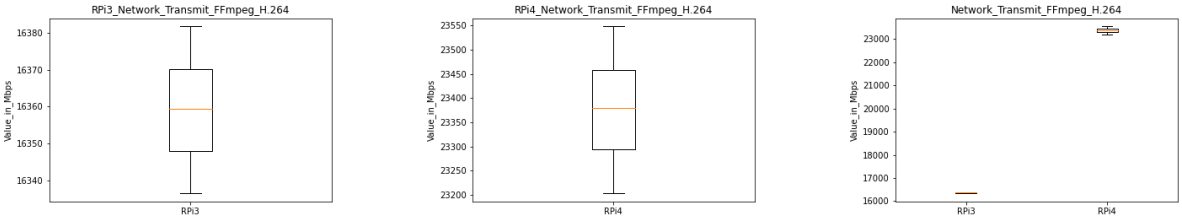

Figure 35: Network Transmit FFmpeg H.264

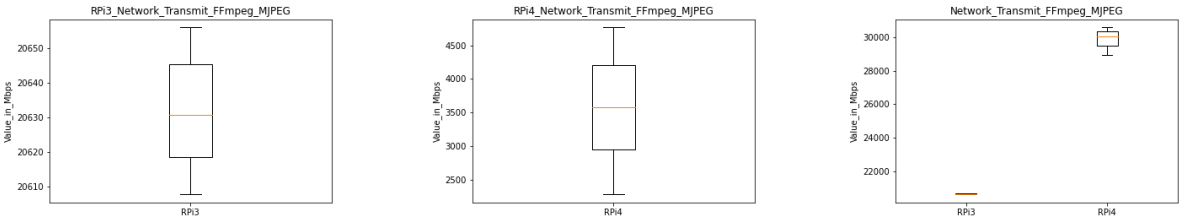

Figure 36: Network Transmit FFmpeg MJPEG

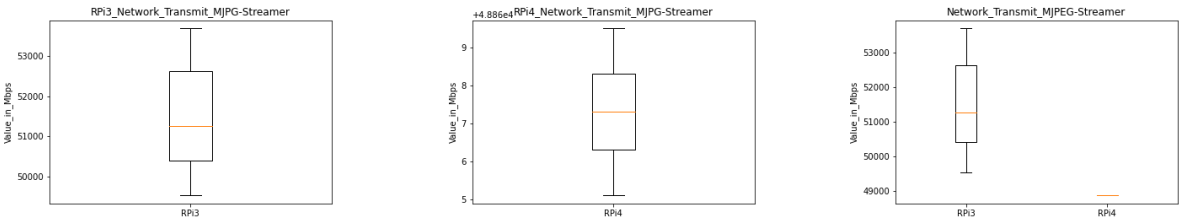

Figure 37: Network Transmit MJPG-Streamer

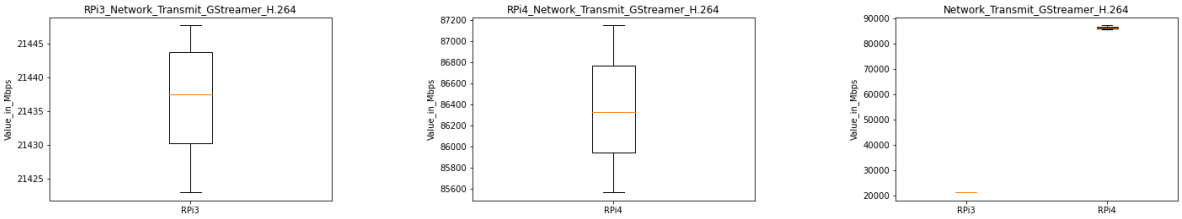

Figure 38: Network Transmit GStreamer H.264

REFERENCES

[1] L. Klinger and V. C. Krolikowsky, “Webrtc surveillance prototype on a raspberry pi,” May 5, 2020.

[2] Wikipedia contributors, “Raspberry pi — Wikipedia, the free encyclopedia,” https://en.wikipedia.org/w/index.php?title=Raspberry_Pi&oldid=982325331, 2020, [Online; accessed 12-October-2020].

[3] I. Red Hat. (2020) What is a raspberry pi? [Online]. Available: https://opensource.com/resources/raspberry-pi

[4] J. HILDENBRAND. (2020) Raspberry pi 4 vs. raspberry pi 3: Should you upgrade? [Online]. Available: https://www.androidcentral.com/raspberry-pi-4-vs-raspberry-pi-3

[5] K. WALSETH. (2019) Raspberry pi 4 vs raspberry pi 3 “a new slice of pi”. [Online]. Available: https://www.digikey.com/en/maker/blogs/2019/raspberry-pi-4-vs-raspberry-pi-3-a-new-slice-of-pi

[6] [Online]. Available: https://webrtc.org/

[7] J. Hans. (2017) Why putting the iot into docker containers will unlock it. [Online]. Available: https://www.rtinsights.com/docker-containers-for-the-iot/

[8] Wikipedia contributors, “Multimedia framework — Wikipedia, the free encyclopedia,” https://en.wikipedia.org/w/index.php?title=Multimedia_framework&oldid=971605991, 2020, [Online; accessed 27-October-2020].

[9] S. Leroux. (May 26, 2019) 5 tricks to get more out of vlc player in linux. [Online]. Available: https://itsfoss.com/vlc-pro-tricks-linux

[10] Wikipedia contributors, “Vlc media player — Wikipedia, the free encyclopedia,” https://en.wikipedia.org/w/index.php?title=VLC_media_player&oldid=979629507, 2020, [Online; accessed 12-October-2020].

[11] T. Vedika. (2020) Mjpeg linux video streaming and recording over http. [Online]. Available: https://techvedika.com/mjpeg-linux-video-streaming-and-recording-over-http/

[12] Wikipedia contributors, “Gstreamer — Wikipedia, the free encyclopedia,” https://en.wikipedia.org/w/index.php?title=GStreamer&oldid=982886621, 2020, [Online; accessed 27-October-2020].

[13] ——, “Video codec — Wikipedia, the free encyclopedia,” https://en.wikipedia.org/w/index.php?title=Video_codec&oldid=977541076, 2020, [Online; accessed 27-October-2020].

[14] J. Ozer. (April 4, 2011) What is h.264? [Online]. Available: https://www.streamingmedia.com/Articles/ReadArticle.aspx?ArticleID=74735

[15] T. cAdvisor Authors, “cadvisor,” https://github.com/google/cadvisor.git, 2014.

[16] Wikipedia contributors, “Prometheus (software) — Wikipedia, the free encyclopedia,” https://en.wikipedia.org/w/index.php?title=Prometheus_(software)&oldid=984182276, 2020, [Online; accessed 28-October-2020].

[17] P. Jupyter. (Oct 21, 2020) Jupyter. [Online]. Available: https://jupyter.org