Low Latency Camera Feed Development

This work was supported in part by the NSF REU program and the donation from nVERSES CAPITAL

Researchers Introduction

Brayden Casaren

Sebastian Clarke

Rohit Karthickeyan

Ayush Iyer

Our Mission

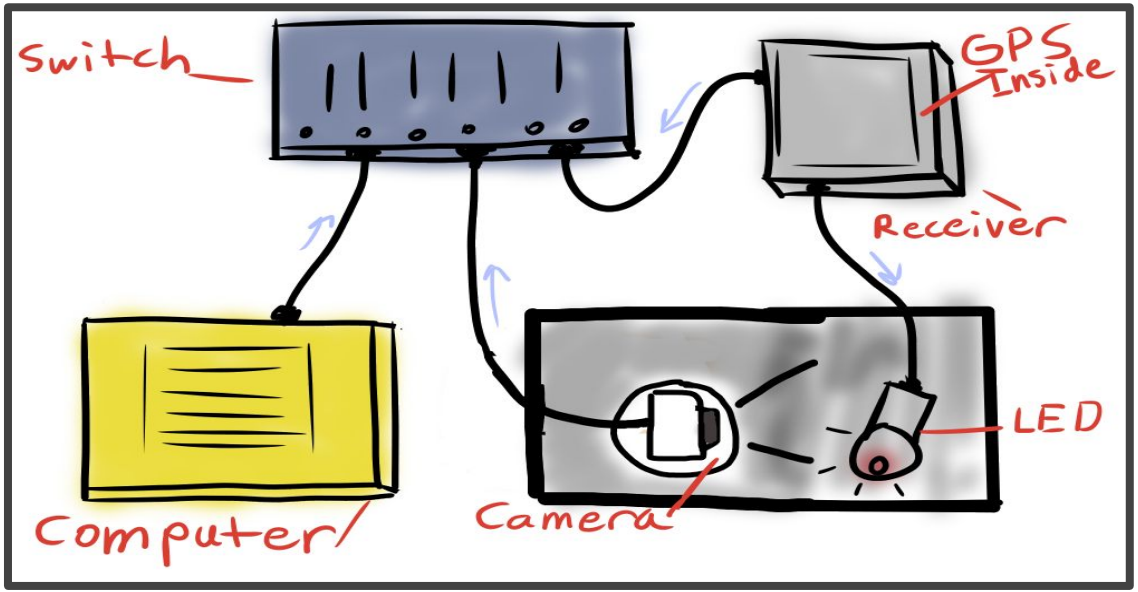
- To find the method(s) of reducing latency to a minimum in a unicast camera to computer connection over a network

The Importance of Low Latency Cameras

The modern day world requires low latency cameras for many applications including:

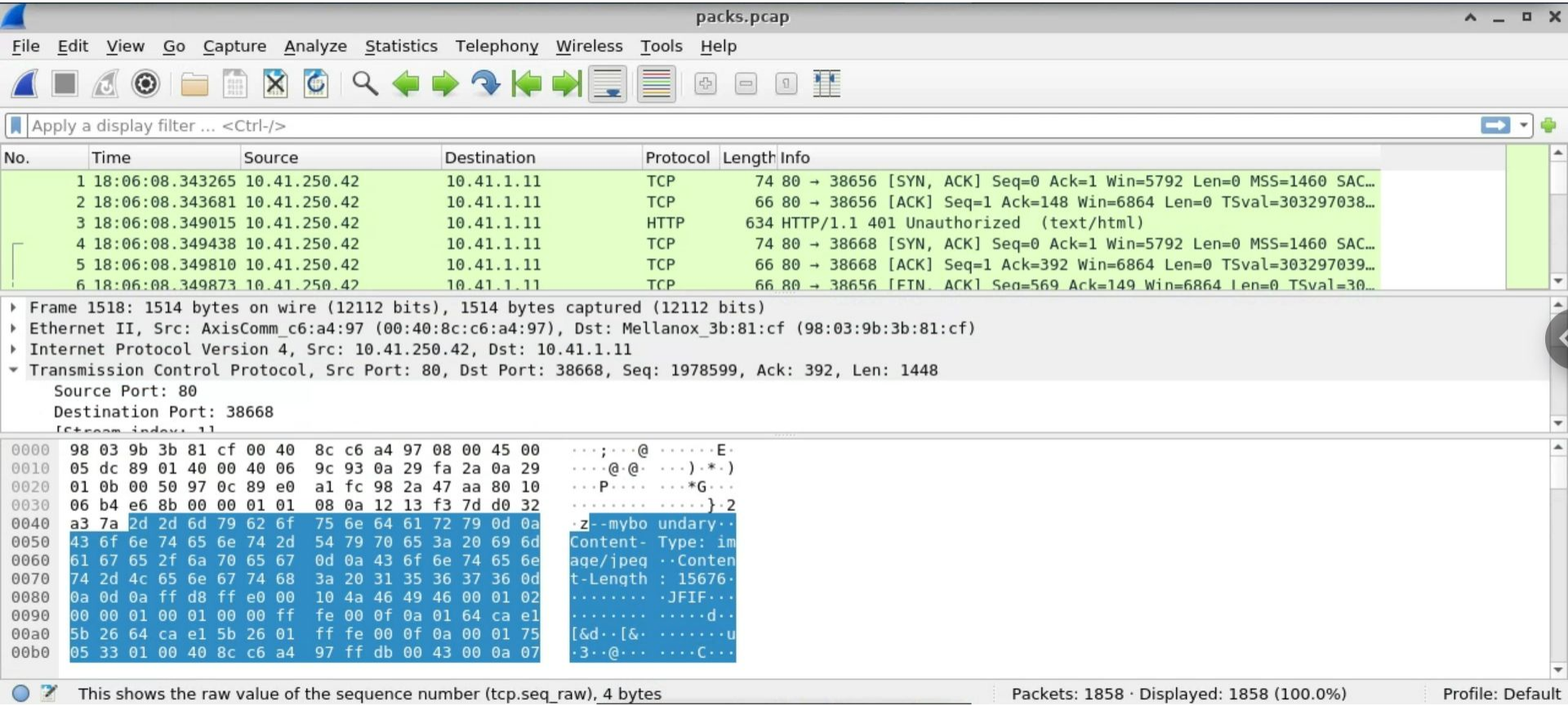
- Self driving cars
- Live Sporting Events and television
- Interviews and News
- Security and Surveillance
- Remote Work Operations(Like Zoom)

Project Design



Testing Setup Diagram

Packets

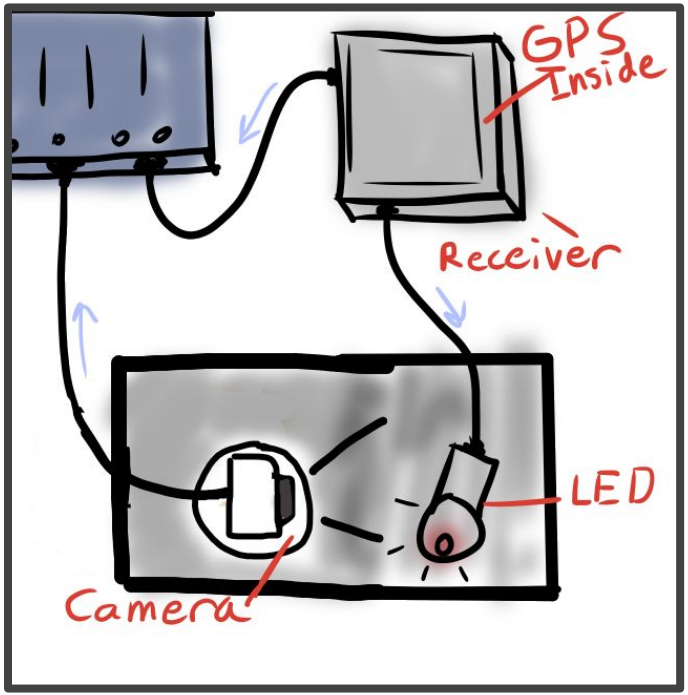

packs.pcap
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
Apply a display filter ... <Ctrl-/>
No. Time Source Destination Protocol Length Info
1 18:06:08.343265 10.41.250.42 10.41.1.11 TCP 74 80 → 38656 [SYN, ACK] Seq=0 Ack=1 Win=5792 Len=0 MSS=1460 SAC…
2 18:06:08.343681 10.41.250.42 10.41.1.11 TCP 66 80 → 38656 [ACK] Seq=1 Ack=148 Win=6864 Len=0 TSval=303297038…
3 18:06:08.349015 10.41.250.42 10.41.1.11 HTTP 634 HTTP/1.1 401 Unauthorized (text/html)
4 18:06:08.349438 10.41.250.42 10.41.1.11 TCP 74 80 → 38668 [SYN, ACK] Seq=0 Ack=1 Win=5792 Len=0 MSS=1460 SAC…
5 18:06:08.349810 10.41.250.42 10.41.1.11 TCP 66 80 → 38668 [ACK] Seq=1 Ack=392 Win=6864 Len=0 TSval=303297039…
6 18:06:08.349873 10.41.250.42 10.41.1.11 TCP 66 80 → 38656 [FIN, ACK] Seq=569 Ack=149 Win=6864 Len=0 TSval=30…
Frame 1518: 1514 bytes on wire (12112 bits), 1514 bytes captured (12112 bits)
Ethernet II, Src: AxisComm_c6:a4:97 (00:40:8c:c6:a4:97), Dst: Mellanox_3b:81:cf (98:03:9b:3b:81:cf)
Internet Protocol Version 4, Src: 10.41.250.42, Dst: 10.41.1.11
Transmission Control Protocol, Src Port: 80, Dst Port: 38668, Seq: 1978599, Ack: 392, Len: 1448
Source Port: 80
Destination Port: 38668
0000 98 03 9b 3b 81 cf 00 40 8c c6 a4 97 08 00 45 00
0010 05 dc 89 01 40 00 40 06 9c 93 0a 29 fa 2a 0a 29
0020 01 0b 00 50 97 0c 89 e0 a1 fc 98 2a 47 aa 80 10
0030 06 b4 e6 8b 00 00 01 01 08 0a 12 13 f3 7d d0 32
0040 a3 7a 2d 2d 6d 79 62 6f 75 6e 64 61 72 79 0d 0a
0050 43 6f 6e 74 65 6e 74 2d 54 79 70 65 3a 20 69 6d
0060 61 67 65 2f 6a 70 65 67 0d 0a 43 6f 6e 74 65 6e
0070 74 2d 4c 65 6e 67 74 68 3a 20 31 35 36 37 36 0d
0080 0a 0d 0a ff d8 ff e0 00 10 4a 46 49 46 00 01 02
0090 00 00 01 00 01 00 00 ff fe 00 0f 0a 01 64 ca e1
00a0 5b 26 64 ca e1 5b 26 01 ff fe 00 0f 0a 00 01 75
00b0 05 33 01 00 40 8c c6 a4 97 ff db 00 43 00 0a 07
This shows the raw value of the sequence number (tcp.seq_raw), 4 bytes
Packets: 1858 · Displayed: 1858 (100.0%)
Profile: Default

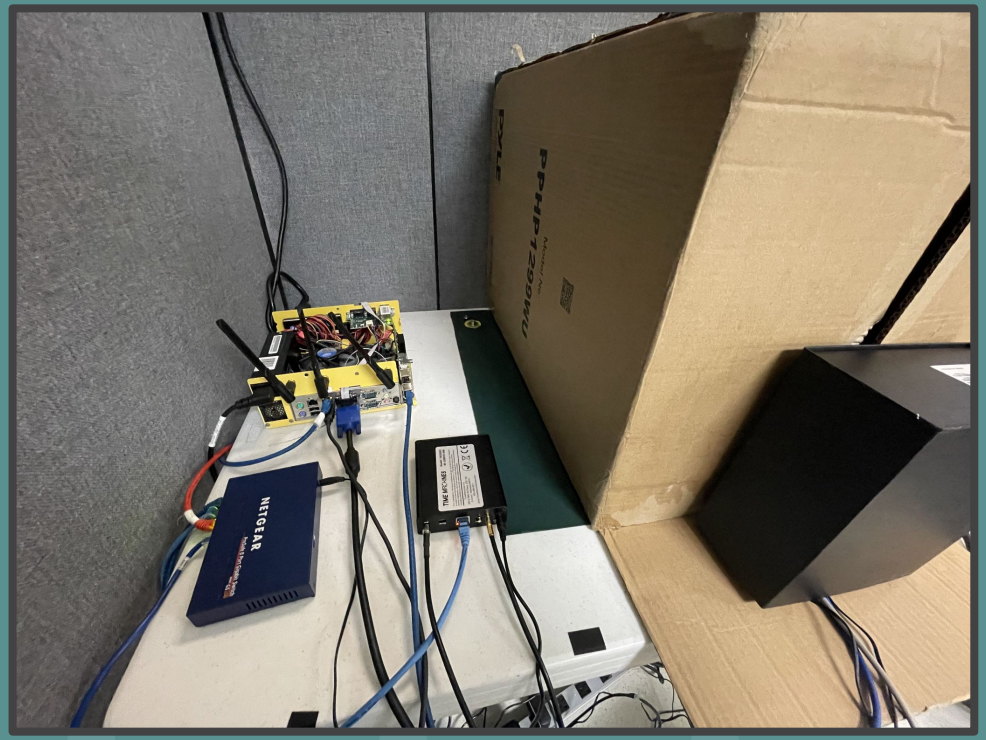
GPS, PPS, and PTP

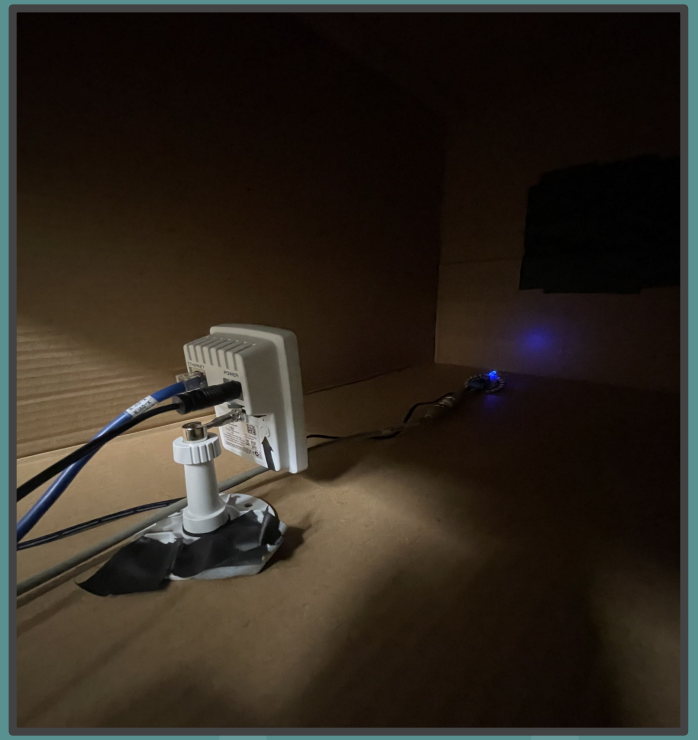
GPS: *Global Positioning System*

PPS: *Pulse per second*

PTP: *Precision Time Protocol*




NETGEAR



Creating Artificial Video

Created videos of varying noises (0%-50%) which were streamed from our secondary node to our primary over VLC

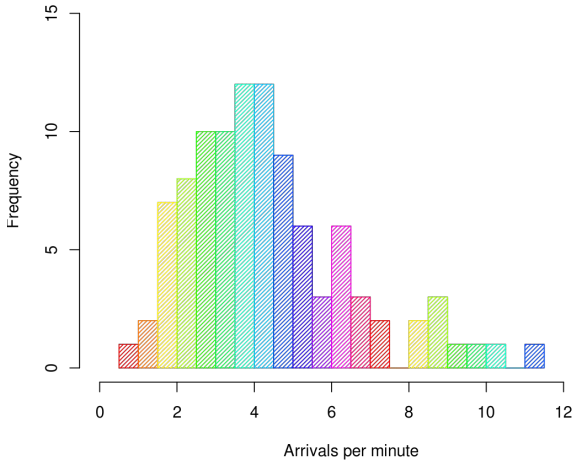
- This setup will allow us to simulate how noise affects our actual camera.

No Noise

30% Noise

Visualizing our data

- Using Matplotlib to create the histograms of our data (byte occurrences)
- Useful for visualizing data particularly for comparison and pattern recognition (more than million pieces of data)
- Want to gain information of how noise affects packets and eventually use for identify when changes occur in our video

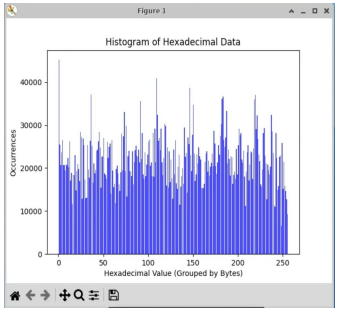


matplotlib

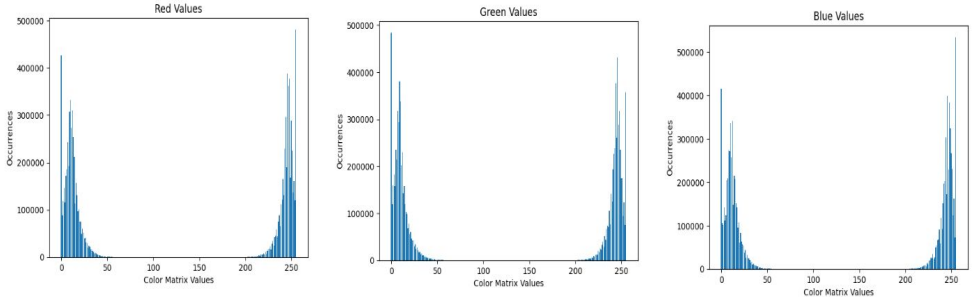
Histograms

We changed the ways our histogram worked in a couple a ways but ended up decided on this:

- Measures RGB pixel values instead of bytes
- Creates three separate graphs for each color (RGB) of how often certains values occurred



Old 10% Graph



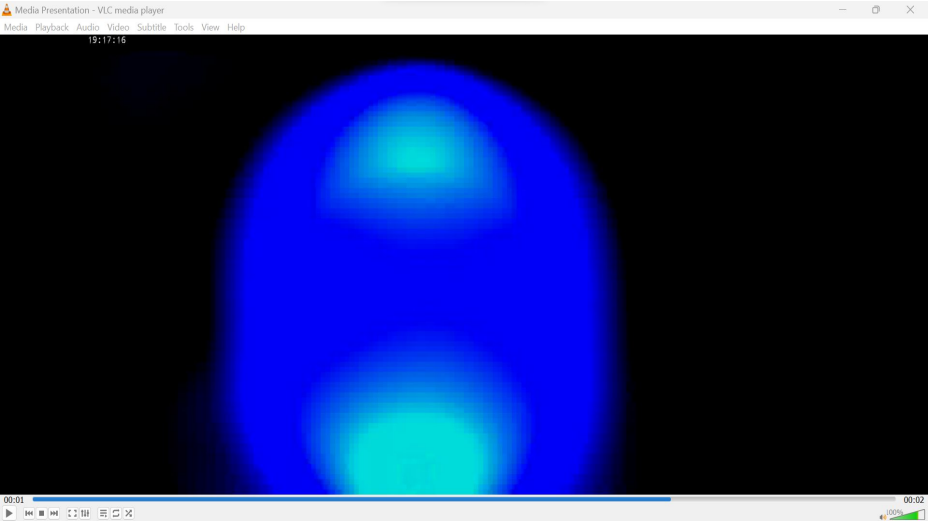
New 10% Graphs

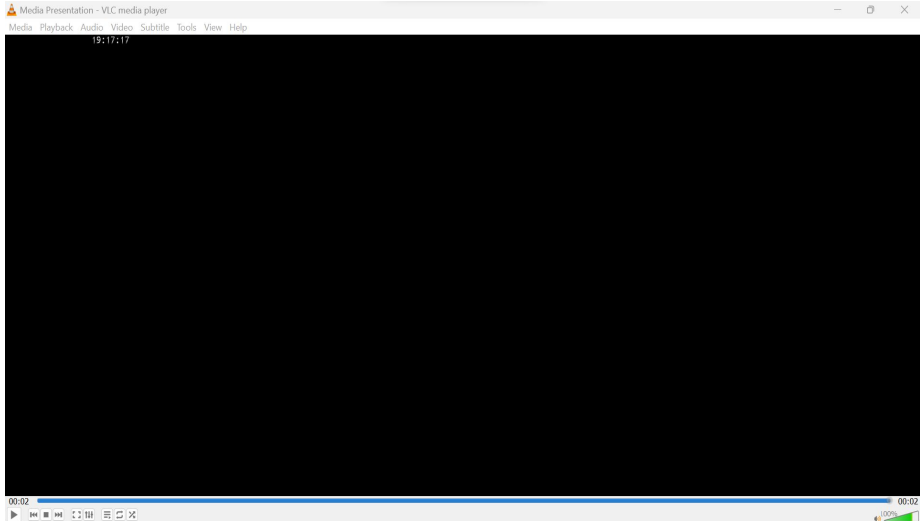
More changes we made

Through camera settings made the noise observed very little

Changed LED setup to take up more of the camera’s view

Used FFMPEG to get video into one MJPEG or and its constituent JPEG files





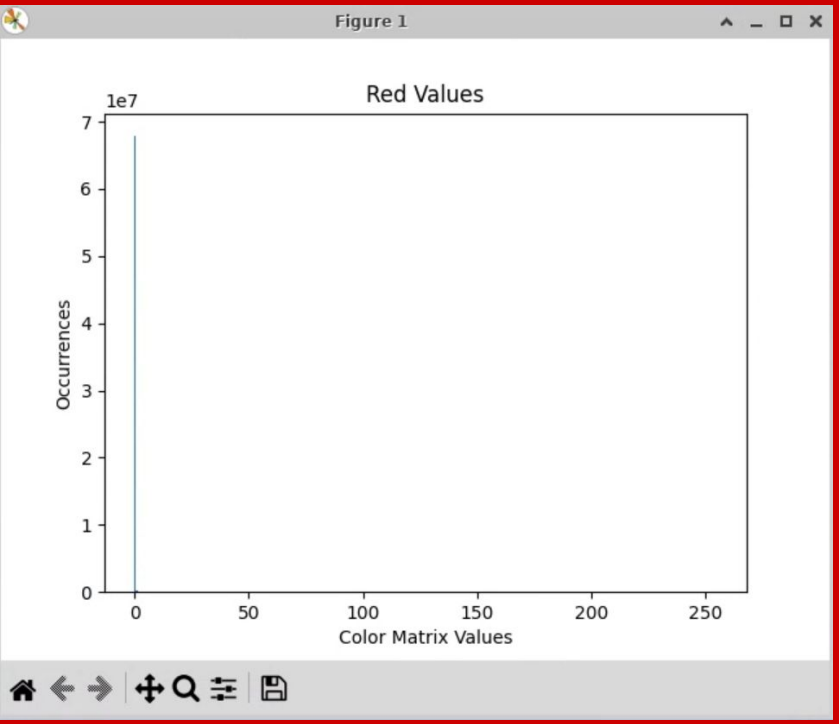


Figure 1
Red Values
1e7
7
6
5
4
3
2
1
0
Occurrences
0
50
100
150
200
250
Color Matrix Values

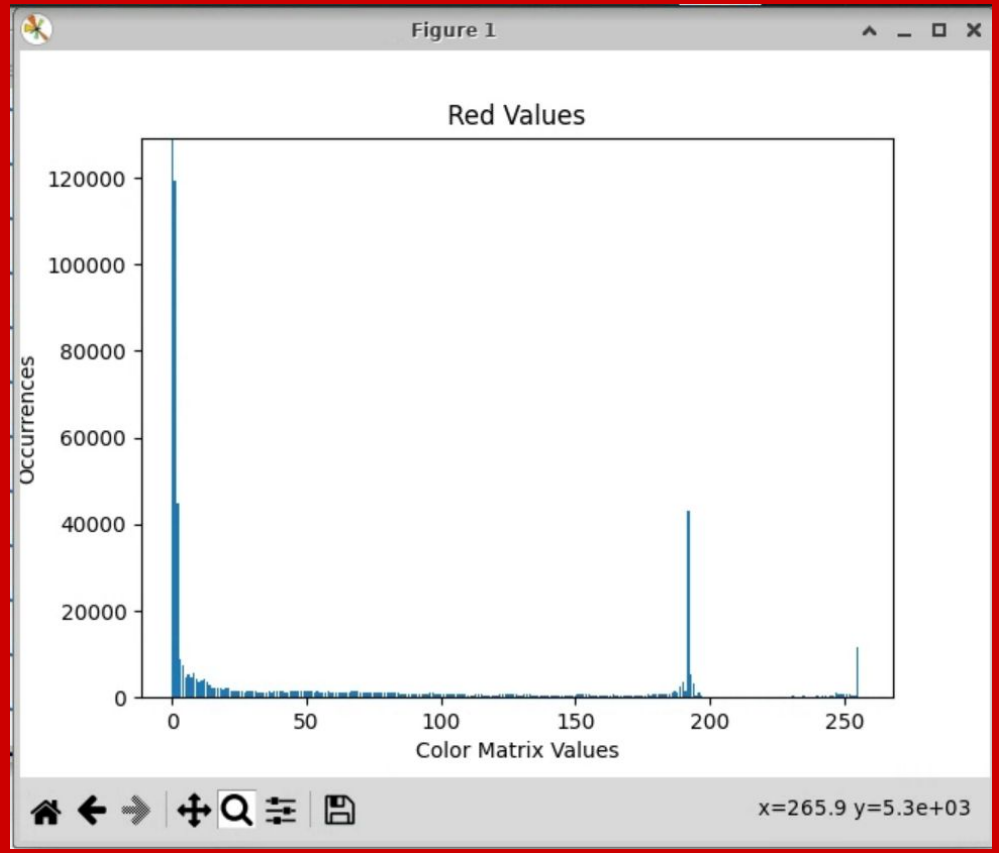


Figure 1
Red Values
120000
100000
80000
60000
40000
20000
0
Occurrences
0
50
100
150
200
250
Color Matrix Values
x=265.9 y=5.3e+03

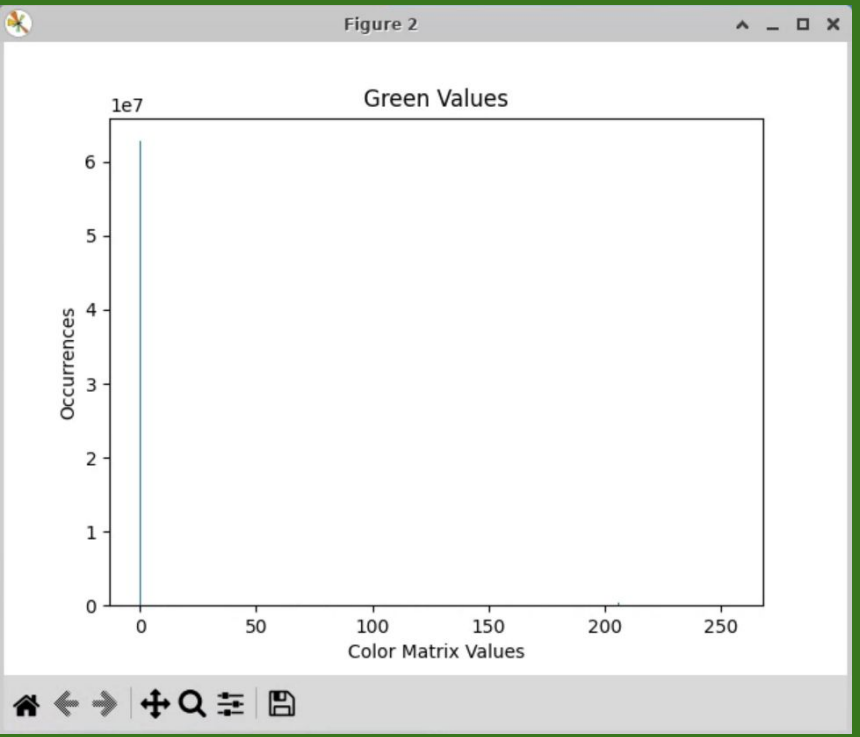


Figure 2
Green Values
1e7
6
5
4
3
2
1
0
Occurrences
0
50
100
150
200
250
Color Matrix Values

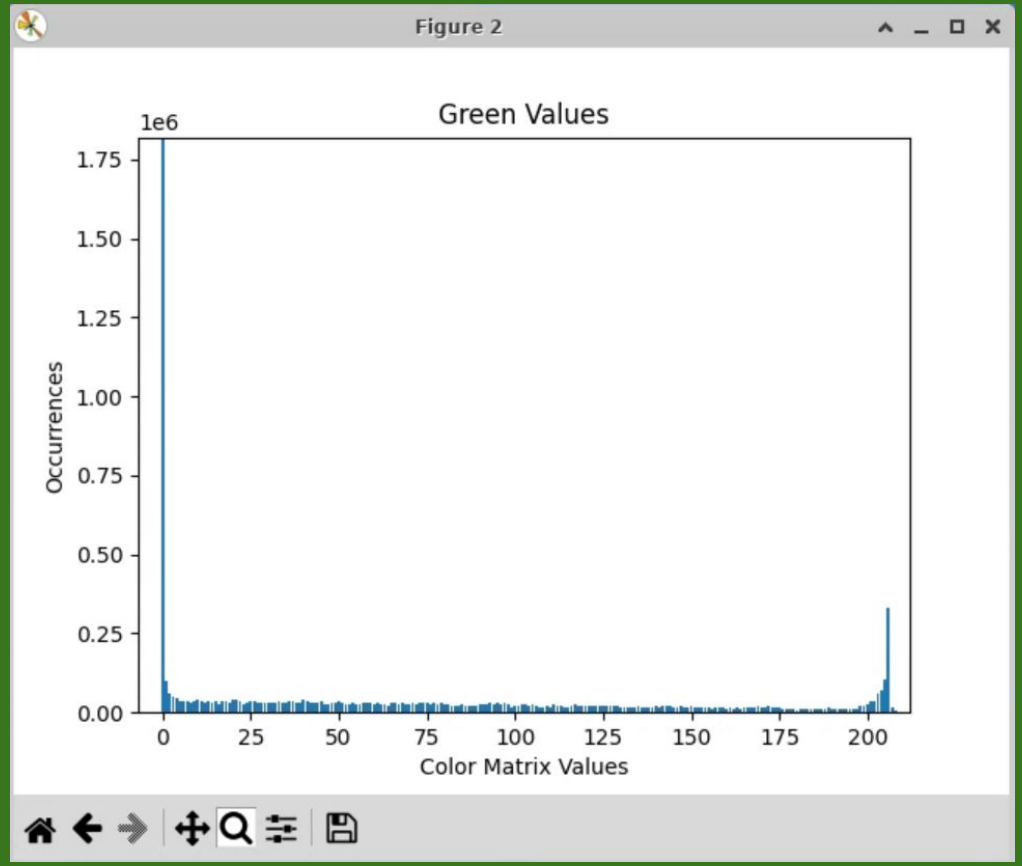


Figure 2
Green Values
1e6
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
Occurrences
0
25
50
75
100
125
150
175
200
Color Matrix Values

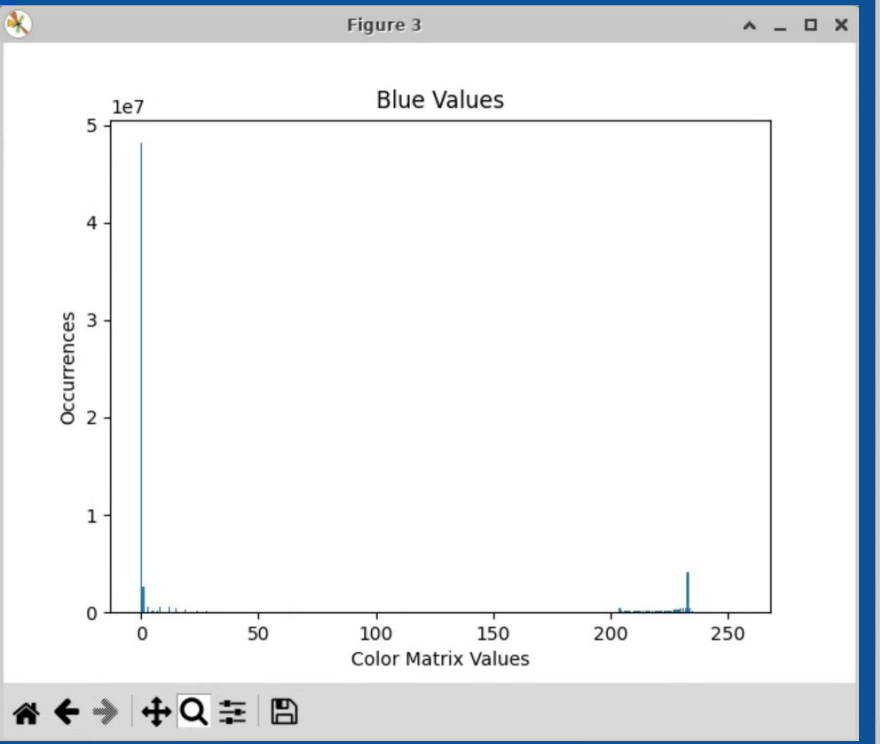


Figure 3
Blue Values
1e7
Occurrences
Color Matrix Values
0
1
2
3
4
5
0
50
100
150
200
250

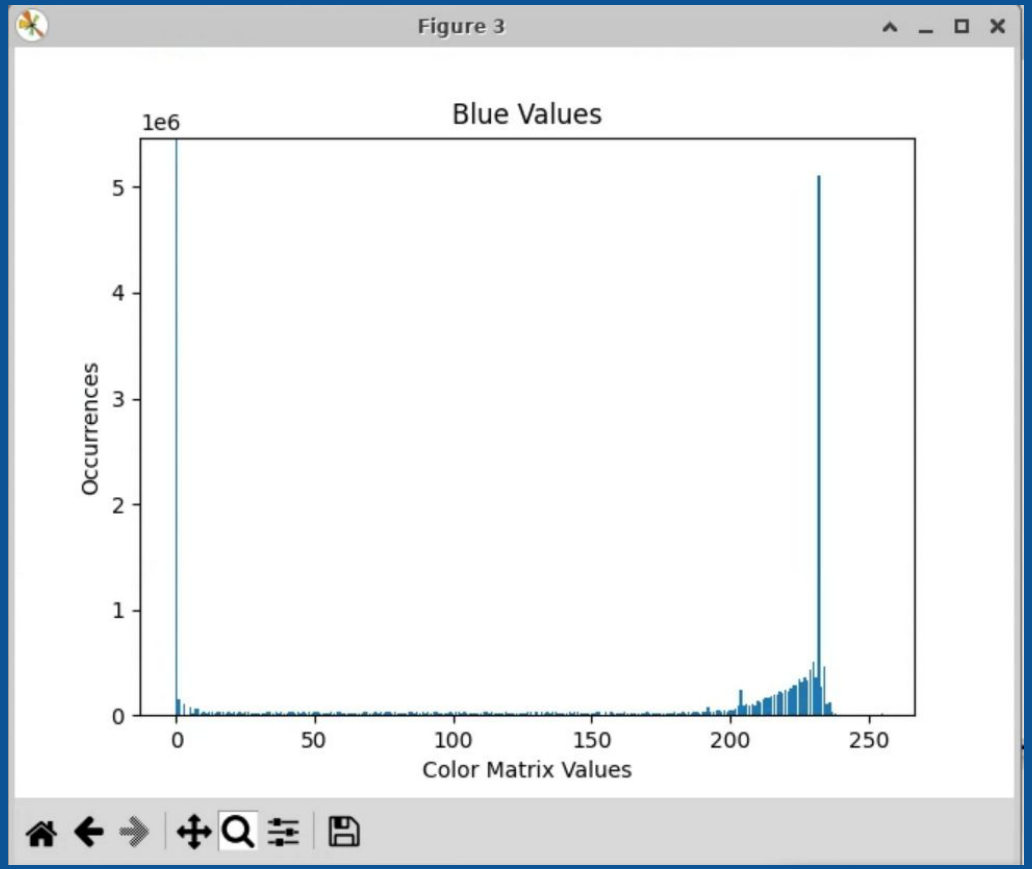
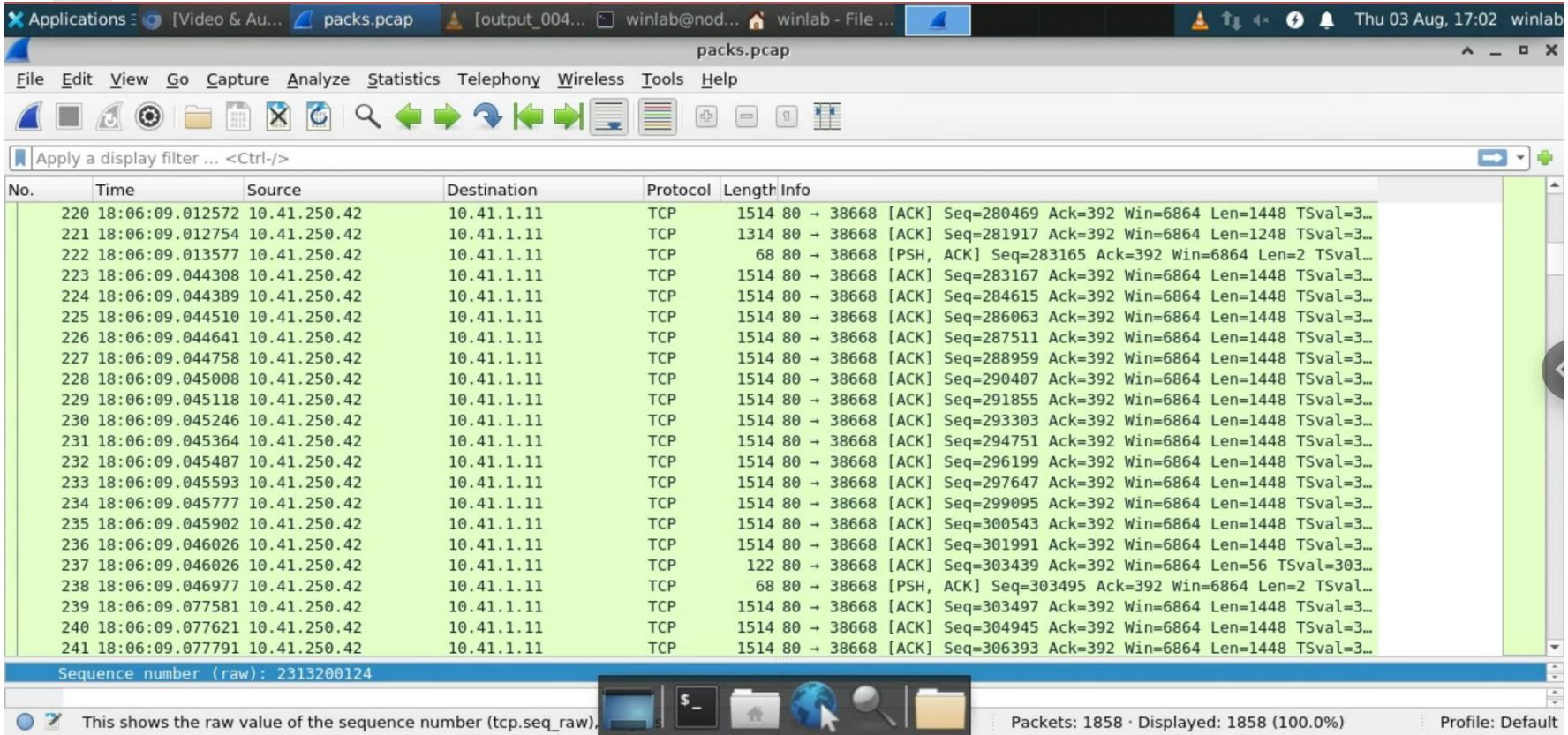
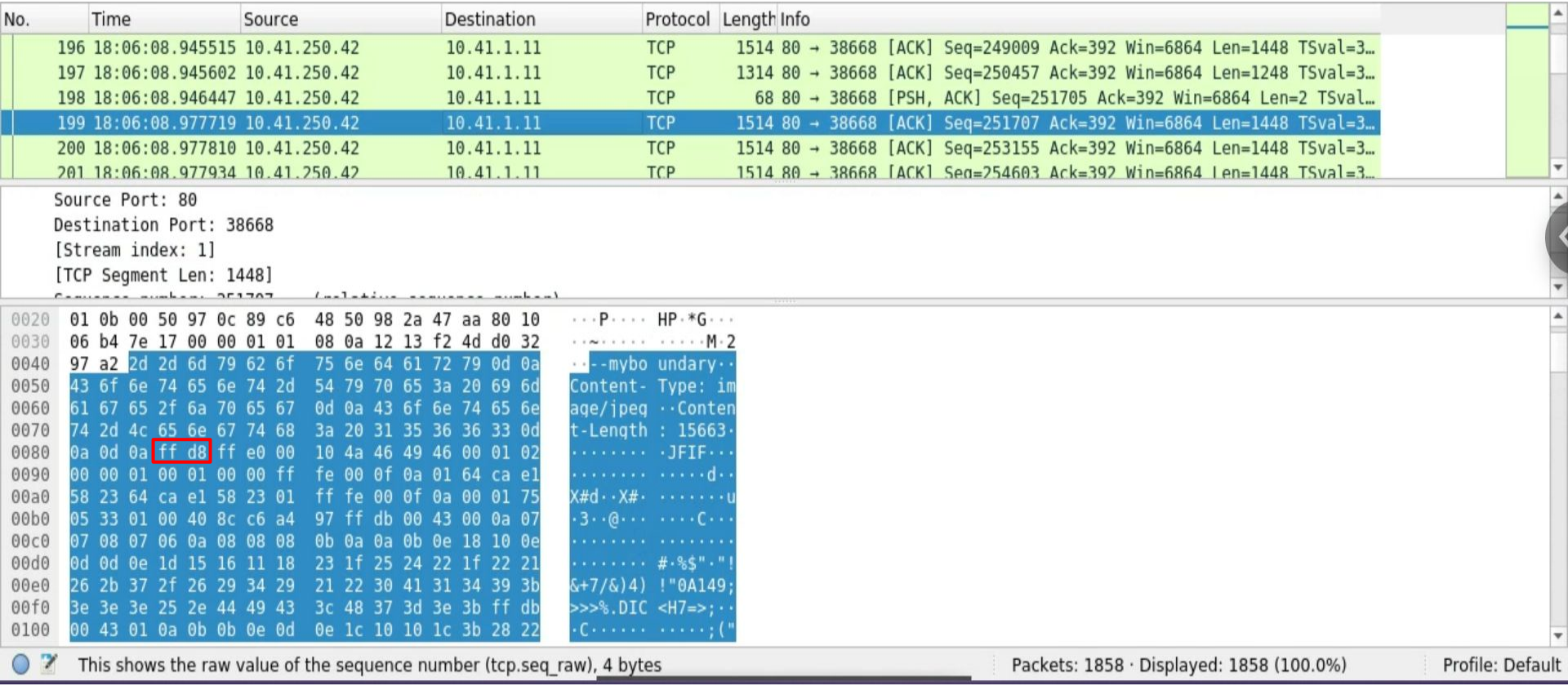
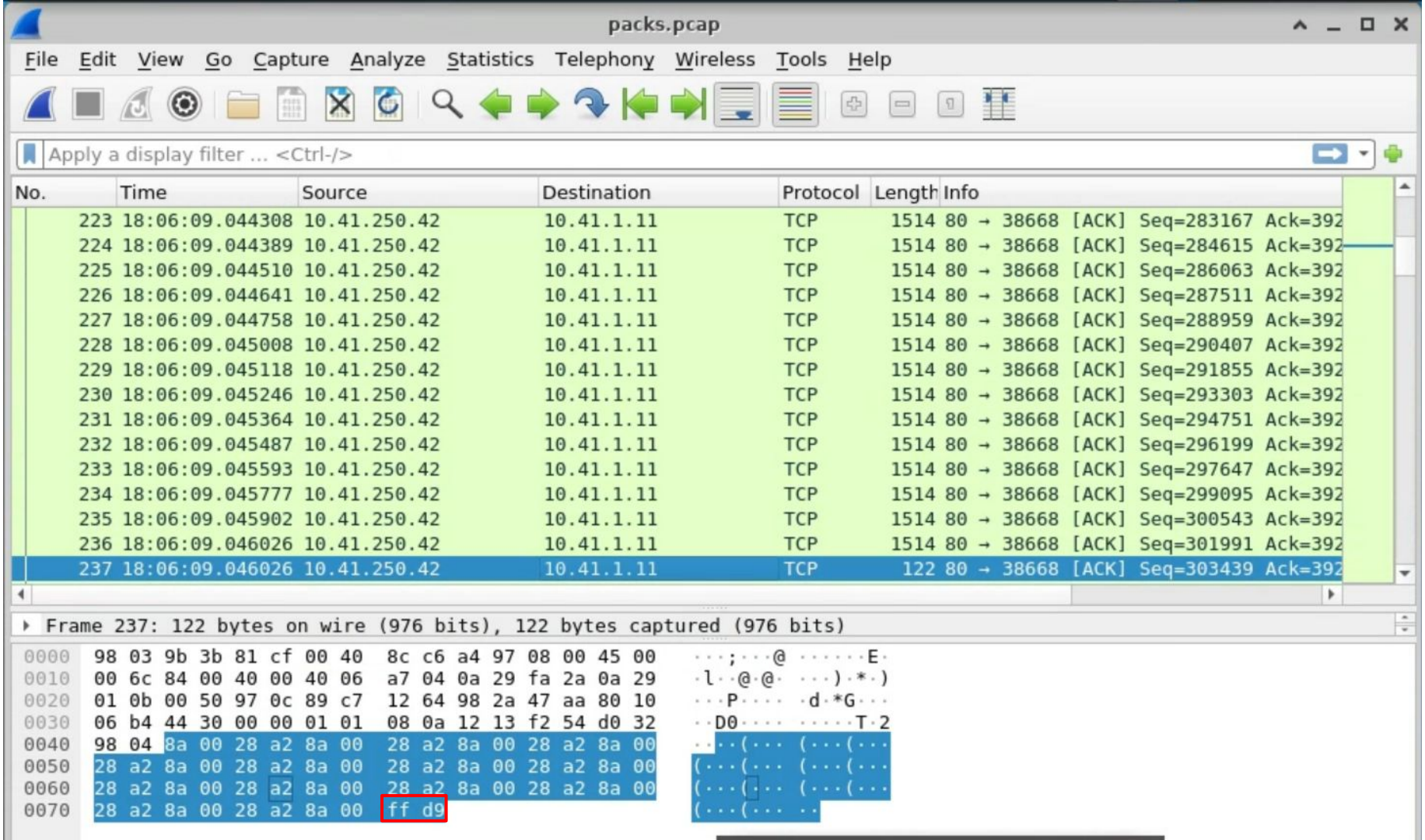


Figure 3
Blue Values
1e6
Occurrences
Color Matrix Values
0
1
2
3
4
5
0
50
100
150
200
250

Calculating Latency




No. Time Source Destination Protocol Length Info
196 18:06:08.945515 10.41.250.42 10.41.1.11 TCP 1514 80 → 38668 [ACK] Seq=249009 Ack=392 Win=6864 Len=1448 TSval=3…
197 18:06:08.945602 10.41.250.42 10.41.1.11 TCP 1314 80 → 38668 [ACK] Seq=250457 Ack=392 Win=6864 Len=1248 TSval=3…
198 18:06:08.946447 10.41.250.42 10.41.1.11 TCP 68 80 → 38668 [PSH, ACK] Seq=251705 Ack=392 Win=6864 Len=2 TSval…
199 18:06:08.977719 10.41.250.42 10.41.1.11 TCP 1514 80 → 38668 [ACK] Seq=251707 Ack=392 Win=6864 Len=1448 TSval=3…
200 18:06:08.977810 10.41.250.42 10.41.1.11 TCP 1514 80 → 38668 [ACK] Seq=253155 Ack=392 Win=6864 Len=1448 TSval=3…
Source Port: 80
Destination Port: 38668
[Stream index: 1]
[TCP Segment Len: 1448]
0020 01 0b 00 50 97 0c 89 c6 48 50 98 2a 47 aa 80 10
0030 06 b4 7e 17 00 00 01 01 08 0a 12 13 f2 4d d0 32
0040 97 a2 2d 2d 6d 79 62 6f 75 6e 64 61 72 79 0d 0a
0050 43 6f 6e 74 65 6e 74 2d 54 79 70 65 3a 20 69 6d
0060 61 67 65 2f 6a 70 65 67 0d 0a 43 6f 6e 74 65 6e
0070 74 2d 4c 65 6e 67 74 68 3a 20 31 35 36 36 33 0d
0080 0a 0d 0a ff d8 ff e0 00 10 4a 46 49 46 00 01 02
0090 00 00 01 00 01 00 00 ff fe 00 0f 0a 01 64 ca e1
00a0 58 23 64 ca e1 58 23 01 ff fe 00 0f 0a 00 01 75
00b0 05 33 01 00 40 8c c6 a4 97 ff db 00 43 00 0a 07
00c0 07 08 07 06 0a 08 08 08 0b 0a 0a 0b 0e 18 10 0e
00d0 0d 0d 0e 1d 15 16 11 18 23 1f 25 24 22 1f 22 21
00e0 26 2b 37 2f 26 29 34 29 21 22 30 41 31 34 39 3b
00f0 3e 3e 3e 25 2e 44 49 43 3c 48 37 3d 3e 3b ff db
0100 00 43 01 0a 0b 0b 0e 0d 0e 1c 10 10 1c 3b 28 22
This shows the raw value of the sequence number (tcp.seq_raw), 4 bytes
Packets: 1858 · Displayed: 1858 (100.0%)
Profile: Default


packs.pcap
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
Apply a display filter ... <Ctrl-/>
No. Time Source Destination Protocol Length Info
223 18:06:09.044308 10.41.250.42 10.41.1.11 TCP 1514 80 → 38668 [ACK] Seq=283167 Ack=392
224 18:06:09.044389 10.41.250.42 10.41.1.11 TCP 1514 80 → 38668 [ACK] Seq=284615 Ack=392
225 18:06:09.044510 10.41.250.42 10.41.1.11 TCP 1514 80 → 38668 [ACK] Seq=286063 Ack=392
226 18:06:09.044641 10.41.250.42 10.41.1.11 TCP 1514 80 → 38668 [ACK] Seq=287511 Ack=392
227 18:06:09.044758 10.41.250.42 10.41.1.11 TCP 1514 80 → 38668 [ACK] Seq=288959 Ack=392
228 18:06:09.045008 10.41.250.42 10.41.1.11 TCP 1514 80 → 38668 [ACK] Seq=290407 Ack=392
229 18:06:09.045118 10.41.250.42 10.41.1.11 TCP 1514 80 → 38668 [ACK] Seq=291855 Ack=392
230 18:06:09.045246 10.41.250.42 10.41.1.11 TCP 1514 80 → 38668 [ACK] Seq=293303 Ack=392
231 18:06:09.045364 10.41.250.42 10.41.1.11 TCP 1514 80 → 38668 [ACK] Seq=294751 Ack=392
232 18:06:09.045487 10.41.250.42 10.41.1.11 TCP 1514 80 → 38668 [ACK] Seq=296199 Ack=392
233 18:06:09.045593 10.41.250.42 10.41.1.11 TCP 1514 80 → 38668 [ACK] Seq=297647 Ack=392
234 18:06:09.045777 10.41.250.42 10.41.1.11 TCP 1514 80 → 38668 [ACK] Seq=299095 Ack=392
235 18:06:09.045902 10.41.250.42 10.41.1.11 TCP 1514 80 → 38668 [ACK] Seq=300543 Ack=392
236 18:06:09.046026 10.41.250.42 10.41.1.11 TCP 1514 80 → 38668 [ACK] Seq=301991 Ack=392
237 18:06:09.046026 10.41.250.42 10.41.1.11 TCP 122 80 → 38668 [ACK] Seq=303439 Ack=392
Frame 237: 122 bytes on wire (976 bits), 122 bytes captured (976 bits)
0000 98 03 9b 3b 81 cf 00 40 8c c6 a4 97 08 00 45 00
0010 00 6c 84 00 40 00 40 06 a7 04 0a 29 fa 2a 0a 29
0020 01 0b 00 50 97 0c 89 c7 12 64 98 2a 47 aa 80 10
0030 06 b4 44 30 00 00 01 01 08 0a 12 13 f2 54 d0 32
0040 98 04 8a 00 28 a2 8a 00 28 a2 8a 00 28 a2 8a 00
0050 28 a2 8a 00 28 a2 8a 00 28 a2 8a 00 28 a2 8a 00
0060 28 a2 8a 00 28 a2 8a 00 28 a2 8a 00 28 a2 8a 00
0070 28 a2 8a 00 28 a2 8a 00 ff d9

```
hexfile= open("output_0018.txt", "r")
packetsfile=open("P223.txt","r")
hex=""
img=""
packets=[]
file=packetsfile.read()
file=file.split()
for i in file:
    hex=hex+i.upper()
hex=hex[hex.index("FFD8"):]
image=hexfile.read()
image=image.split()
for i in image:
    img=img+i.upper()
if(img[:len(hex)]==hex):
    print("Match")
else:
    print("No Match")
```

```
winlab@node1-11:~/FFMPEGplusTCPDUMP/testjpg$ python3 compare.py
Match
```

Reducing Latency:

Viable methods:

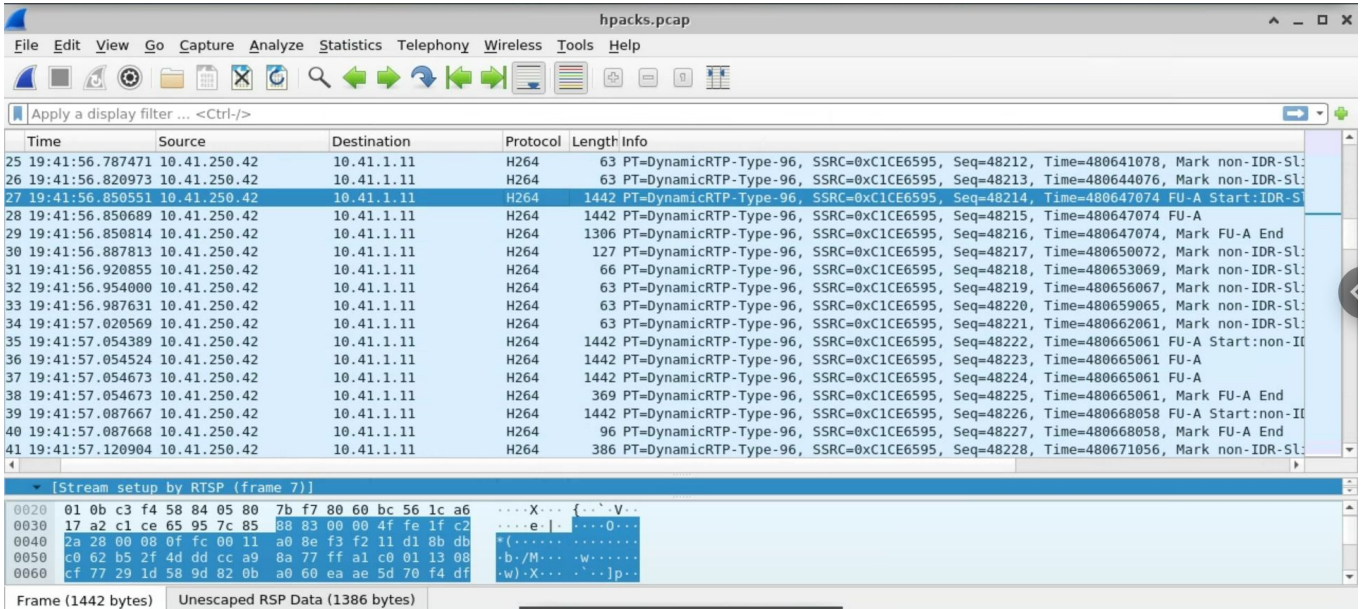
Changing Codec, Camera can run MJPEG and H264

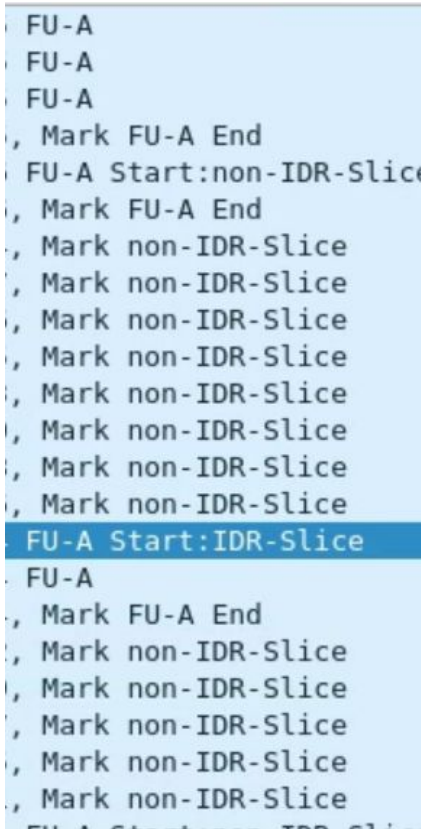
Changing Framerate, Camera can operates between 0-30 fps

Changing Compression amount, Camera defaults to 30 on scale of 0-100

Switching Codec (H264)

- 3 Types of frames: I, P, and B
- Frame referencing and motion vectors with P and B frames to reduce size
- Calculated latency to be around 34 ms for just I frames, 60 ms for GOV of 3





THANK YOU

Any Questions?