Computer Networks - Project Report

Marzieh Berenjkoub

November 27, 2017

0.1 Tutorial

In this project, I develop a software for smart authentication of people behind the door. The whole program implemented in python and Java(android) using the following libraries:

· **Raspberry pi** $json, base64, pygame, pyttsx, tts_watson, cv2 and numpy, scapy$

··· **AWS** $boto, boto.s3, logging, scapy$

··· **Android** No need for special library

In order to run the code on raspberry pi, user has to select the mode which is polling or non-polling and frame rate. The whole system has the feature of adding new user. It is implemented in android program and user can register the person with an Id and a sensor data and it will be stored on server.

0.1.1 What you will Learn

In this project you will learn how to develop a multi-thread client and server program. You will learn how to work with AWS and how to use it's services such as S3 and Amazon rekognition. You will also learn Open CV face recognition library (I used Haar cascade) and how to train data set. In addition you will learn how to measure the overhead of your program dynamically and adjust the image size or frame rate based on the bandwidth to optimize your program.

0.2 System Performance

In order to measure the number of bytes that are sent from Raspberry Pi to the cloud; I used tcpdump. I filter the destination port. Figure 1 shows the result, as we see the whole packet will be split into several fragments with size of 1516kb, which include 20 byte for IP header, 32 byte for TCP header and 16 byte for ethernet header. The real size of the packet is 1448 and for one image (which has been saved in jason format)we have around 19 fragments.

Experiment 1 For measuring the overhead in code, I'm using scapy library. It is running on a separate thread and measure all the traffic that passed on this port. I disable the polling in order to measure the traffic in one direction only(from raspberry pi to AWS).

Figure 2 shows the total traffic, image size and overhead. For measuring the overhead in polling section, I disable the sending thread and activate only polling part. Figure 3 shows the measurements.

The overhead between android and AWS is almost the same as RPI to AWS since the size of the packet and json format is almost similar.

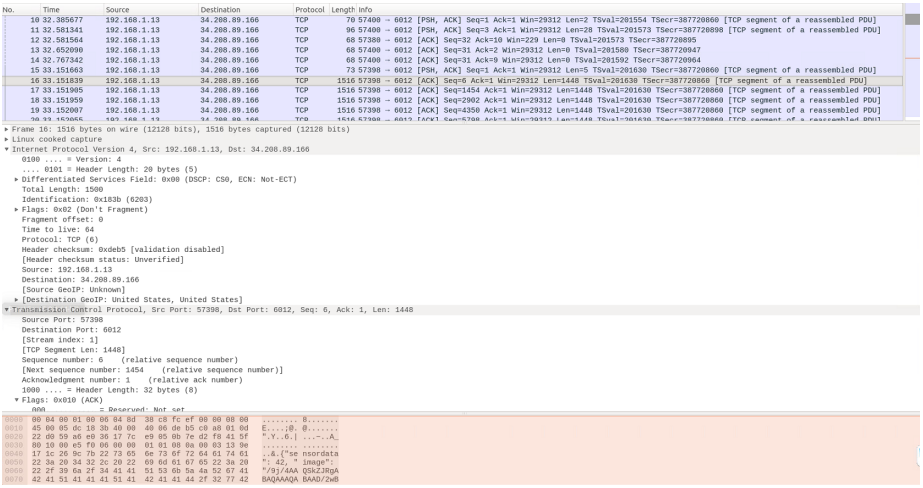

Figure 1: The output of traffic filtered on a port using tcpdump command and visualizing wireshark

Experiment2 In this experiment we play with the bandwidth and see how much we need to resize the image. The user also has to specify the frame rate, so based on that we are measuring the maximum size of the file (kb) based on the following formulea:

$$max\_imagesize = (bandwidth//((framerate) \times (1 + overhead)))/8 \quad (1)$$

The dimension of original image I'm working with has 800*600 and based on my experiment in the first part, it will generate an image with size around 30KB. Since jpeg compression has non linear correlation with dimension, I'm using binary search to find an approximate dimension close to maximum file size.

Now, I'm going to set the bandwidth to less than 30KB and find the image size considering 6% overhead. Figure 4 shows how we set the bandwidth and Table 1 shows the result of image size and dimension varying by different bandwidth and frame rate.

0.2.1 Highest Frame Rate

The highest frame rate that my program did not crash on was 15 of course without considering frame number. However, there are two problems, the first one we need to receive the frame in order in the server and we are being force to wait for some frames. In order to solve this two problems I also send frame number and I consider a threshold to time out those frames arriving with high latency. Considering these two parameters, the total frame rate decreased to 8.

0.2.2 Face Detection

In general, running API on server will generate better performance. Since even if we run face recognition of RPI, we still need to have a backup on server and the backup

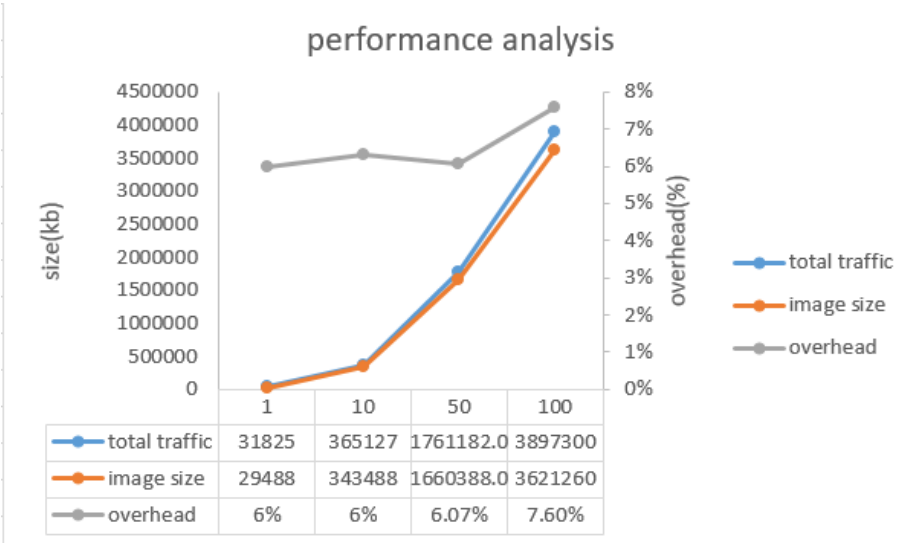

Figure 2: Performance analysis and overhead measurement with different frame number.

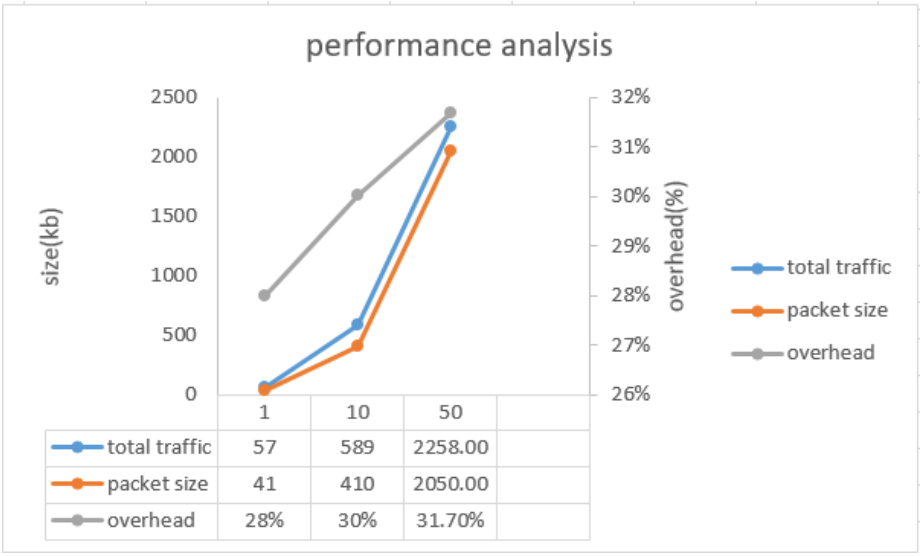

Figure 3: Performance analysis and overhead measurement with different frame number for polling part.

```
pi@raspberrypi:~ $ sudo tc qdisc add dev eth0 root handle 1:0 tbf rate 20kbit bu
ffer 1600 limit 3000
```

Figure 4: Adjusting the bandwidth manually.

Table 1: Image size and dimension computed for different frame rates and bandwidth

frame_rate/bandwidth(kb)	20		15		10	
1	18161	(640*560)	13135	(500*375)	9325	(400*300)
5	3626	(200*150)	2691	(160*120)	1900	(114*85)

has to keep updated. That cause an extra traffic for the whole system.

0.2.3 Polling vs Non-Polling

The parameter which affect polling costs in interval between sending the query. It the interval is very short like a second, we need to keep the connection, but if the interval is longer, then we can create and close the connection each time. This is one benefit of polling since we are not forced to keep the connection always. However, in Non-polling since the RPI has to created the connection, it needs to keep the connection active and the server will send it's update occasionally. This is not optimal in comparison with Polling if we do not need to keep the connection alive. In contrast, for a short interval, the cost of creating and deleting the socket is high, it is the same as Non-polling and we need to keep the connection active. In this case, Non-polling is better since it generates less traffic.