Name : Anusha Nemilidinne

UHID : 1619484

REPORT

ON

BUILDING A PHYSICAL HOME AUTHENTICATION SYSTEM

WHAT YOU WILL LEARN ?

From this project, you will learn how to build an IOT system using socket programming, how to make the authentication system adapt to the ever changing network for example adjusting the image quality depending on the data rate at the server side to achieve the frame rate specified by user and helps you understand different parameters that determine the network overhead .

IoT Project: A Physical Authentication System For Your Home

We will build a physical authentication system in this project. We use a Raspberry Pi and a few sensors to authenticate a person and allow an admin user to view the person being authenticated and supervise the authentication process.

This project requires Raspberry Pi, a proximity sensor, and a webcam. It is implemented in python.

Set up:

1. Connect a distance sensor to Raspberry Pi. Reading data from sensor and how to connect sensor to pi

can be found in the following link:

https://github.com/johnbryanmoore/VL53L0X_rasp_python

2. Connect a webcam to Raspberry Pi. For capturing image frames, fswebcam or openCV can be used.

- For fswebcam , it has to be installed in the pi using **sudo apt-get install fswebcam.**

  Use **fswebcam <Image name>** to capture images.

- opencv 3.2 should be installed in pi which nearly takes 4 hours! Installation instructions available in the link:

  https://github.com/Tes3awy/OpenCV-3.2.0-Compiling-on-Raspberry-Pi

3. Amazon ec2 server needs to be set up for which first we need to create an account in amazon. The below link has instructions for this.

http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/get-set-up-for-amazon-ec2.html#create-a-vpc

4. Android 3.2.0 has to be installed to build the app.

After the initial set up is done, you are all set to get started with the implementation.

Implementation:

1. The first step is to capture an image using opencv or fswebcam and run face recognition software on the image collected. This is implemented in two ways:

- Using Amazon AWS Face Rekognition API. The instructions on how to use it can be found at :

  https://www.hackster.io/gr1m/raspberry-pi-facial-recognition-16e34e

  The code is present in **facematch.py** file.

- Using openCV for facerecognition. The code can be found at **face_recognition.py** python. It maintains a folder training_data that contains training images of subjects. In this project I have used training images for 3 people so it contains 3 folders one for each person and for each person a total of 12 images is taken as training data.

  It uses LBPH face recogniser and uses frontal cascade xml for training the model.

  train_model(): It trains the model .

  predictor(): Predicts the image passed to it.

  reference link: https://www.superdatascience.com/opencv-face-recognition/

- Sockets are used to facilitate communication between Ec2 server and raspberry pi and from server to android app. Since raspberry pi is behind NAT which has private IP address, the communication between server and pi happens over a connection request sent from pi to server since server cannot connect to pi.

- We maintain a simple database which is a text file which contains details of the authenticated guests. I have used Name,Age,distance,Relation,Profession as the details of a person. This is maintained at id_database.txt.

1. Authentication_m2_v3.py and client_m2_v3.py files capture images along with the distance of the person, run face recognition on the collected image, check if the user is in the database(Name and distance should match) and if the user is present send the image to the Ec2 server. I have used hand gesture as an additional feature to authenticate the user. If the user gestures a "Hi", then that is taken as password. The code for this is found at **gesture.py.**

In order for a visitor to be authenticated by the system, it checks if he is already in database, then checks if he stands within specified distance from camera and gestures a "HI"

The main file is **Authentication_m2_v3.py** which calls **client_m2_v3.py.**

This step incorporates sending of multiple images over a period of time to accomplish video streaming. The mechanism is the pi sends a connection request to server,for which server has to be listening, sends the details and image of the person to server and waits for the server for a signal which tells it to capture the next frame.

2. This also has a module which uses Amazon TTS(Text to speech service) polly which converts the authentication status to speech and it can be heard in the microphone attached to raspberrypi. This is the last step of the system.

1. If the user authenticates the new visitor to home, then the output is "Welcome home <name of the person>".

2. If user doesnot authenticate, the output is "System does not recognize you, try again"

3. In order for the raspberry pi to check the authentication status, I have used two approaches.: 1. Polling(Pi keeps checking for authentication status file in server) 2. Non polling(Instead of pi waiting for the status, as and when the server gets the status it send it directly to pi).

2. The file **server_m2_v3.py** should be running the whole time this process happens. It's basic functionality is to collect the images , details of that person and send those files to the android app.

3. At the android side, **SocketClient-master** provides the user with a simple GUI, which has 2 text fields for Ip address and port number of the server, image view to display images , buttons for displaying the details and authenticating the user.

Displaying images continuously is achieved as :

1. When user presses the connect button, android displays the first image.

2. Now the server sends to android to wait for the next image, and android keeps waiting for the next image. When server receives the next image from pi, it sends a signal to app saying it is sending next image and breaks out of the loop. The app now receives the image and displays it. All this happens in the background.

4. When the owner wants to add a new user to the system, I designed a utility for adding new user which is present in **add_newUser.py.** It captures images ,the count of which is user defined, adds the images to training data, trains the model of face recognizer,prompts the user for some details about that person and finally adds him to the database.

When the user authenticates the image, the status is sent to server which then sends to pi or pi detects the file depending on whether we use non-polling or polling respectively and the finally the pi converts the status to speech and plays the audio which either says welcome home or please try again!!

SYSTEM PERFORMANCE:

Factors that contribute to the system performance are:

1. Accuracy of output i.e.; how accurately the system recognizes the given image, of face recognition library provided by openCV depends on the number of images used to train the model. The more the number of images, the better the system recognizes the image. So if a slightly different version of the same image is given and if the training data doesnot have enough of such images the model will not be able to predict the image.

2. There is certain latency observed in the images displayed on the app because of the processing at pi side, network delays ,available bandwidth and image resolution.

PERFORMANCE RELATED INSIGHTS:

1. **Factors that determine the network overhead**:

In TCP transmission, along with the data payload which is the actual data, tcp protocol adds certain headers for each packet like source ip, destination port and destination ip. If to transfer a file of size of M, suppose tcp requires P packets each of MTU 1500 bytes, each packet requires 40 bytes of tcp header. So for P packets we have P*40 bytes of header. This is an over head to the network. So in order to send a file of size M, it has to send (M + P*40) bytes in total. This means if the link has a speed of M Mbps , which means a file of size M should be sent in one second but since it sends extra bytes the files take much longer time thus not utilising the bandwidth properly.

Also, in order for a socket to send data, a handshake between server and client has to be performed. Also if a packet is lost, tcp has to retransmit the packet. These are all network overheads.

How did I calculate my network overhead:

When pi sends a file to the server, I used T shark, command line tool for wireshark, to capture packets

during that period.

since we know the number of packets that are sent by pi(M/1500) , up on the inspection of the packet trace in tshark, I added the size of all packets and subtracted from it the file size to calculate the network overhead.

Total size of all P packets = S

overhead = S - M(file size)

The size of the file I sent is 66427 bytes.

M= 62327 bytes

P = Upon the inspection of packet trace, the tcp uses a total of 44 packets apart from SYN and ACK, to send the data.

so each packet size is 1448.

Total bytes = 1448*44 = 63712 bytes

overhead = 1385

Hence tcp bandwidth overhead = 2.2 %

Screen shot of packet trace when pi sent one image to server:

```
  163 12.314882012    10.0.0.64 → 10.0.0.4     TCP 60 60213 → 22 [ACK] Seq=1 Ack=1537 Win=251 Len=0
  164 12.358564368  54.175.3.72 → 10.0.0.4     TCP 74 65534 → 40324 [SYN, ACK] Seq=0 Ack=1 Win=26847 Len=0 MSS=1460 SACK_PERM=1 TSval=603124505 TSecr=1821800
WS=128
  165 12.358637858     10.0.0.4 → 54.175.3.72  TCP 66 40324 → 65534 [ACK] Seq=1 Ack=1 Win=29312 Len=0 TSval=1821805 TSecr=603124505
  166 12.359287809     10.0.0.4 → 10.0.0.64    SSH 134 Server: Encrypted packet (len=80)
  167 12.366526743     10.0.0.4 → 54.175.3.72  TCP 1514 40324 → 65534 [ACK] Seq=1 Ack=1 Win=29312 Len=1448 TSval=1821806 TSecr=603124505
  168 12.366606431     10.0.0.4 → 54.175.3.72  TCP 1514 40324 → 65534 [ACK] Seq=1449 Ack=1 Win=29312 Len=1448 TSval=1821806 TSecr=603124505
  169 12.366729035     10.0.0.4 → 10.0.0.64    SSH 134 Server: Encrypted packet (len=80)
  170 12.366780754     10.0.0.4 → 54.175.3.72  TCP 1514 40324 → 65534 [ACK] Seq=2897 Ack=1 Win=29312 Len=1448 TSval=1821806 TSecr=603124505
  171 12.366846588     10.0.0.4 → 54.175.3.72  TCP 1514 40324 → 65534 [ACK] Seq=4345 Ack=1 Win=29312 Len=1448 TSval=1821806 TSecr=603124505
  172 12.366914922     10.0.0.4 → 54.175.3.72  TCP 1514 40324 → 65534 [ACK] Seq=5793 Ack=1 Win=29312 Len=1448 TSval=1821806 TSecr=603124505
  173 12.366975182     10.0.0.4 → 54.175.3.72  TCP 1514 40324 → 65534 [ACK] Seq=7241 Ack=1 Win=29312 Len=1448 TSval=1821806 TSecr=603124505
  174 12.367045547     10.0.0.4 → 54.175.3.72  TCP 1514 40324 → 65534 [ACK] Seq=8689 Ack=1 Win=29312 Len=1448 TSval=1821806 TSecr=603124505
  175 12.367107422     10.0.0.4 → 54.175.3.72  TCP 1514 40324 → 65534 [ACK] Seq=10137 Ack=1 Win=29312 Len=1448 TSval=1821806 TSecr=603124505
  176 12.367175600     10.0.0.4 → 54.175.3.72  TCP 1514 40324 → 65534 [ACK] Seq=11585 Ack=1 Win=29312 Len=1448 TSval=1821806 TSecr=603124505
  177 12.367229975     10.0.0.4 → 54.175.3.72  TCP 1514 40324 → 65534 [PSH, ACK] Seq=13033 Ack=1 Win=29312 Len=1448 TSval=1821806 TSecr=603124505
  178 12.372215098    10.0.0.64 → 10.0.0.4     TCP 60 53630 → 22 [ACK] Seq=1921 Ack=2385 Win=255 Len=0
  179 12.423002847  54.175.3.72 → 10.0.0.4     TCP 66 65534 → 40324 [ACK] Seq=1 Ack=2897 Win=32640 Len=0 TSval=603124521 TSecr=1821806
  180 12.423151910     10.0.0.4 → 54.175.3.72  TCP 1514 40324 → 65534 [ACK] Seq=14481 Ack=1 Win=29312 Len=1448 TSval=1821812 TSecr=603124521
  181 12.423205035     10.0.0.4 → 54.175.3.72  TCP 1514 40324 → 65534 [ACK] Seq=15929 Ack=1 Win=29312 Len=1448 TSval=1821812 TSecr=603124521
  182 12.423324150     10.0.0.4 → 54.175.3.72  TCP 1514 40324 → 65534 [ACK] Seq=17377 Ack=1 Win=29312 Len=1448 TSval=1821812 TSecr=603124521
  183 12.423399984     10.0.0.4 → 54.175.3.72  TCP 1514 40324 → 65534 [ACK] Seq=18825 Ack=1 Win=29312 Len=1448 TSval=1821812 TSecr=603124521
  184 12.424217539  54.175.3.72 → 10.0.0.4     TCP 66 65534 → 40324 [ACK] Seq=1 Ack=5793 Win=38528 Len=0 TSval=603124521 TSecr=1821806
  185 12.424294727     10.0.0.4 → 54.175.3.72  TCP 1514 40324 → 65534 [ACK] Seq=20273 Ack=1 Win=29312 Len=1448 TSval=1821812 TSecr=603124521
  186 12.424328477     10.0.0.4 → 54.175.3.72  TCP 1514 40324 → 65534 [ACK] Seq=21721 Ack=1 Win=29312 Len=1448 TSval=1821812 TSecr=603124521
  187 12.424511551     10.0.0.4 → 54.175.3.72  TCP 1514 40324 → 65534 [ACK] Seq=23169 Ack=1 Win=29312 Len=1448 TSval=1821812 TSecr=603124521
  188 12.424550092     10.0.0.4 → 54.175.3.72  TCP 1514 40324 → 65534 [ACK] Seq=24617 Ack=1 Win=29312 Len=1448 TSval=1821812 TSecr=603124521
  189 12.424611082  54.175.3.72 → 10.0.0.4     TCP 66 65534 → 40324 [ACK] Seq=1 Ack=7241 Win=41344 Len=0 TSval=603124521 TSecr=1821806
  190 12.424700874     10.0.0.4 → 54.175.3.72  TCP 1514 40324 → 65534 [ACK] Seq=26065 Ack=1 Win=29312 Len=1448 TSval=1821812 TSecr=603124521
```

Graph:

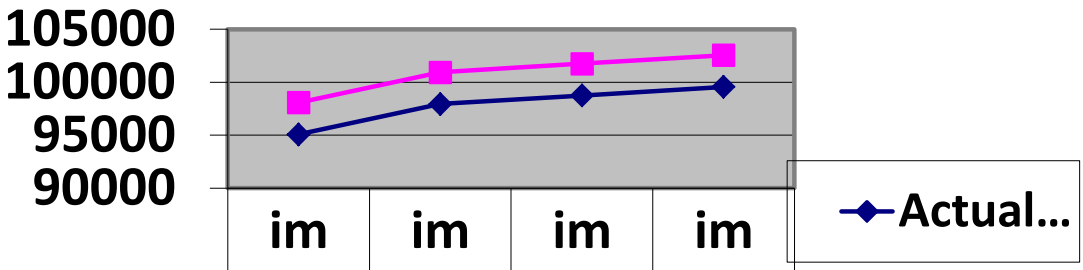

	image 1	image 2	image 3	image 4
Actual file size	95080	97939	98746	99557
Real bytes sent	98080	100939	101746	102557

2. **Processing and network tradeoff when using online API and opencv** :

Processing : It is observed that when I used Amazon AWS face rekognition API, it gave accurate output that is it recognized the image correctly.

When this is done using openCV face recognition library, it sometimes have not predicted the image accurately. However, the accuracy can be improved by increasing the number of training images.

Note : For both implementations, I have given same number of training images per person.

Network : Since the processing of an image is done on a remote server hosted on cloud when we use online API, it might take comparably more time to produce the output because of network related delays.

When we use open cv for image processing , since the entire processing is done on the local system, the results might be faster.

Hence there is a processing and network tradeoff between the two implementations.

2. Frame rate:

1. The goal of pi sending frames continuously to android is achieved as:

Once the pi sends an image to server, it waits for a signal from server to send another image. Once the android receives an image, it has to wait for server to send next image. This waiting happens in backgrund and almost negligible delay is observed. Hence the user perceives negligible delay when he/she sees the images.

The fastest frame rate that I am able to achieve is 5 frames per second. When the frame rate is greater than 5, the user starts experiencing some delay.

3.Positive and negative aspects of polling vs non-polling:

Polling : In this implementation , pi continuously checks the server for authenticatio status. This has a setback which is if during the polling interval pi doesnot find any file, it has to keep waiting for the next polling interval in order to check for the file even though the file is available immediately after when pi has first polled. This introduces delays and affects system performance.

Resource utilization of network by pi when it uses this implementation, is higher and it may take huge chunk of bandwidth for this purpose which slows down the network at pi side. This happens because in order for the pi to ping ,the request might use several packets which is an overhead for the network.

Non polling : In this implementation, pi just connects to the server. So whenever the server has the authentication status file available it just sends it immediately to the pi and then pi can resume further processing. This approach is way faster than the polling method.

Measurements:

The polling that I implemented polls the server for every 10 seconds for the status file. It sends a "1" which means it is asking the server if it has the file. The server replies back by sending "0"/"1" if it does not have or has the file respectively. so for polling once, it requires 2 packets in order to check whether the file is there.

Now for the whole process of sending images and displaying them to android, and user authenticating the visitor, took approximately 5 minutes.

This implies 300 secs. Now pi has polled 30 times which requires 60 packets in total just to check and then it has to get the file.

Now, in non polling , pi just connects and waits. The server whenever it has the file just sends it which eliminated a total of 60 extra packets.

New feature added to the system:

Along with unique face and sensor proximity distance as a password for a visitor, I have added another new feature which is gesture detection. I used opencv's gesture detection. When a guest is home, he has to wave at the camera, which is then detected by this script and is used as a password for that person. Each person in the database is assigned a specific hand gesture. If this gesture matches to the one that is detected along with the other two parameters, the user is authenticated.

Gesture - "HI"